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 languag
import pandas as pd
# NumPy is a library for the Python programming language, adding support
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLa
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It p
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
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
```

```
<ipython-input-2-b7d446354769>:2: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major rele
ase of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and b
etter interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
54466

  import pandas as pd
```

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_tickl
    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.clou
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
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.clou
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_GE
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	(
...	...	...	...	...	...	...	
85	86.0	15400.000000	2.0	5.0	2.0	0.0	(
86	87.0	15400.000000	3.0	5.0	2.0	0.0	(
87	88.0	15400.000000	6.0	5.0	5.0	0.0	(
88	89.0	15400.000000	3.0	5.0	2.0	0.0	(
89	90.0	3681.000000	1.0	5.0	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 [12]:
```
# Display the column names of the DataFrame
print(data.columns)
```

```
Index(['FlightNumber', 'Date', 'BoosterVersion', 'PayloadMass', 'Orbit',
       'LaunchSite', 'Outcome', 'Flights', 'GridFins', 'Reused', 'Legs',
       'LandingPad', 'Block', 'ReusedCount', 'Serial', 'Longitude', 'Latit
ude',
       'Class'],
      dtype='object')
```

In [13]:
```
# Access the 'Class' column and convert it to a NumPy array
Y = data['Class'].to_numpy()

# Print or check the output (optional)
print(Y)
```

```
[0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 1 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1
 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 1
 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

TASK 2

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

In [15]:
```
# students get this
transform = preprocessing.StandardScaler()

# Import preprocessing if not already done
from sklearn import preprocessing

# Step 1: Fit the scaler and transform the data in X
X = transform.fit_transform(X)

# Step 2: (optional) Check the transformed data
print(X)
```

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

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 [16]:
```
# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,

# Check the shapes of the resulting sets (optional)
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
(72, 83) (18, 83) (72,) (18,)
```

we can see we only have 18 test samples.

In [17]:
```
Y_test.shape
```

Out[17]:
```
(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 [18]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

In [19]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 l
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]:
```
# Define the parameters for GridSearchCV
parameters = {'C': [0.01, 0.1, 1, 10], 'penalty': ['l2'], 'solver': ['lbf

# Step 1: Create a logistic regression object
lr = LogisticRegression()
```

```
# Step 2: Create the GridSearchCV object
logreg_cv = GridSearchCV(lr, parameters, cv=10)

# Step 3: Fit the GridSearchCV object to the training data
logreg_cv.fit(X_train, Y_train)

# Step 4: Output the best parameters and the best score
print("Tuned hyperparameters (best parameters):", logreg_cv.best_params_)
print("Best accuracy score:", logreg_cv.best_score_)
```

```
Tuned hyperparameters (best parameters): {'C': 0.01, 'penalty': 'l2', 'sol
ver': 'lbfgs'}
Best accuracy score: 0.8464285714285713
```

TASK 5

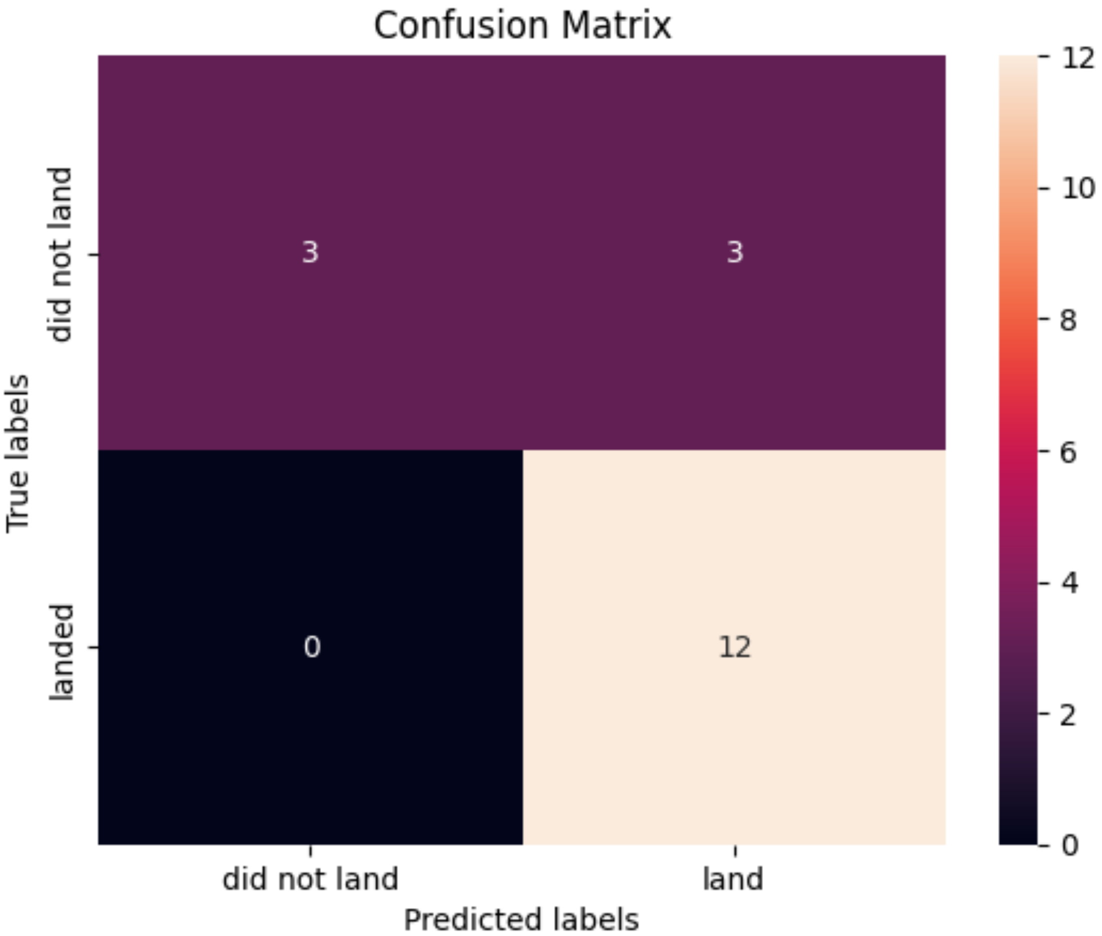
Calculate the accuracy on the test data using the method `score` :

In [22]:
```
# Calculate the accuracy on the test data
test_accuracy = logreg_cv.score(X_test, Y_test)

# Print the test accuracy
print("Test Accuracy:", test_accuracy)
```

```
Test Accuracy: 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 problem is false positives.

Overview:

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

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

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 [24]:
```
# Define the parameter grid
parameters = {
    'kernel': ('linear', 'rbf', 'poly', 'sigmoid'),
    'C': np.logspace(-3, 3, 5),
    'gamma': np.logspace(-3, 3, 5)
}

# Step 1: Create a support vector machine (SVM) object
svm = SVC()

# Step 2: Create a GridSearchCV object for SVM with 10-fold cross-validat
```

```
svm_cv = GridSearchCV(svm, parameters, cv=10)

# Step 3: Fit the GridSearchCV object to the training data
svm_cv.fit(X_train, Y_train)

# Step 4: Output the best parameters and accuracy score
print("Tuned hyperparameters (best parameters):", svm_cv.best_params_)
print("Best accuracy score:", svm_cv.best_score_)
```

```
Tuned hyperparameters (best parameters): {'C': 1.0, 'gamma': 0.03162277660
168379, 'kernel': 'sigmoid'}
Best accuracy score: 0.8482142857142856
```

TASK 7

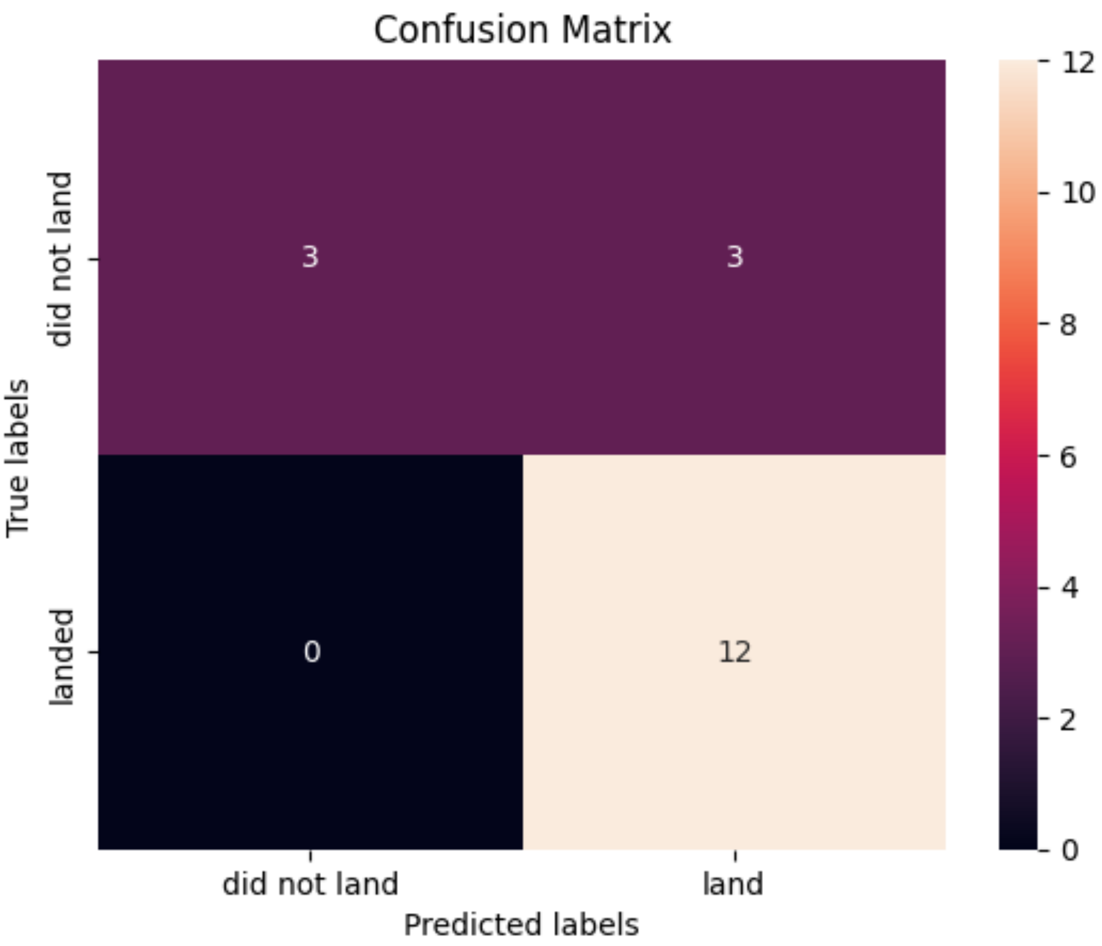
Calculate the accuracy on the test data using the method `score`:

In [25]:
```
# Calculate the accuracy on the test data for SVM
svm_test_accuracy = svm_cv.score(X_test, Y_test)

# Print the test accuracy for SVM
print("Test Accuracy for SVM:", svm_test_accuracy)
```

```
Test Accuracy for SVM: 0.8333333333333334
```

We can plot the confusion matrix

In [26]:
```
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 [28]:

```
# Define the parameter grid for Decision Tree
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]
}

# Step 1: Create a decision tree classifier object
tree = DecisionTreeClassifier()

# Step 2: Create a GridSearchCV object for Decision Tree
tree_cv = GridSearchCV(tree, parameters, cv=10)

# Step 3: Fit the GridSearchCV object to the training data
tree_cv.fit(X_train, Y_train)

# Step 4: Output the best parameters and accuracy score for Decision Tree
print("Tuned hyperparameters (best parameters):", tree_cv.best_params_)
```

```python
print("Best accuracy score for Decision Tree:", tree_cv.best_score_)
```

```
/lib/python3.12/site-packages/sklearn/model_selection/_validation.py:547:
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 e
rror_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------
------
3240 fits failed with the following error:
Traceback (most recent call last):
  File "/lib/python3.12/site-packages/sklearn/model_selection/_validation.
py", line 895, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/lib/python3.12/site-packages/sklearn/base.py", line 1467, in wrap
per
    estimator._validate_params()
  File "/lib/python3.12/site-packages/sklearn/base.py", line 666, in _vali
date_params
    validate_parameter_constraints(
  File "/lib/python3.12/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 {'sqrt', 'log2'} or None. Got
'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/lib/python3.12/site-packages/sklearn/model_selection/_search.py:1051: Use
rWarning: One or more of the test scores are non-finite: [       nan
nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.80714286 0.81785714 0.80357143 0.81964286 0.7375     0.75178571
 0.73928571 0.73392857 0.73928571 0.875      0.83214286 0.7625
 0.76428571 0.73392857 0.73214286 0.78928571 0.7625     0.74821429
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.80178571 0.81964286 0.80535714 0.83392857 0.79285714 0.83571429
 0.80357143 0.83035714 0.79107143 0.7375     0.77321429 0.80535714
 0.78928571 0.78035714 0.775      0.80357143 0.80357143 0.77678571
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.77678571 0.80535714 0.81785714 0.75357143 0.75       0.73928571
 0.77678571 0.84821429 0.71964286 0.79107143 0.72142857 0.83214286
 0.73571429 0.87321429 0.77678571 0.79642857 0.81785714 0.72142857
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.80714286 0.83392857 0.76785714 0.80714286 0.78928571 0.79107143
```

```
 0.80535714 0.72321429 0.71964286 0.73214286 0.83214286 0.77678571
 0.775      0.83214286 0.77678571 0.75357143 0.81785714 0.74642857
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.80357143 0.76071429 0.81964286 0.79107143 0.78928571 0.80357143
 0.76428571 0.81428571 0.73392857 0.73392857 0.80535714 0.81964286
 0.78928571 0.78035714 0.88928571 0.70892857 0.775      0.82321429
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.80357143 0.88571429 0.79107143 0.8875     0.77857143 0.81785714
 0.76071429 0.775      0.80535714 0.70714286 0.78928571 0.81607143
 0.77857143 0.74821429 0.77857143 0.80714286 0.75357143 0.775
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.79642857 0.74821429 0.83035714 0.82142857 0.81785714 0.80535714
 0.77678571 0.78035714 0.81785714 0.7625     0.775      0.79464286
 0.71964286 0.76964286 0.77857143 0.80714286 0.725      0.81785714
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.69285714 0.74821429 0.79285714 0.75178571 0.80535714 0.78928571
 0.80535714 0.77678571 0.77678571 0.79107143 0.77678571 0.79285714
 0.80357143 0.725      0.7625     0.73928571 0.74821429 0.77857143
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.76071429 0.81964286 0.7625     0.74821429 0.80714286 0.80357143
 0.74642857 0.73571429 0.81785714 0.75       0.83214286 0.81785714
 0.76428571 0.79285714 0.81785714 0.77857143 0.77321429 0.83214286
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.76071429 0.81964286 0.74821429 0.73928571 0.81785714 0.7625
 0.74821429 0.7625     0.84642857 0.78928571 0.74642857 0.75178571
 0.78928571 0.77678571 0.63928571 0.81785714 0.79464286 0.80357143
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.78928571 0.84642857 0.8625     0.78928571 0.84642857 0.7625
 0.76607143 0.81785714 0.74821429 0.84642857 0.79285714 0.83214286
 0.75       0.84642857 0.84642857 0.75178571 0.71964286 0.72321429
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.84642857 0.80357143 0.69285714 0.80357143 0.74642857 0.83392857
 0.78928571 0.80357143 0.83035714 0.83392857 0.7625     0.80357143
 0.82857143 0.81785714 0.7625     0.84464286 0.77857143 0.775
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.80535714 0.77678571 0.81607143 0.775      0.81964286 0.83571429
 0.74642857 0.80714286 0.70892857 0.77678571 0.78928571 0.80357143
 0.775      0.84642857 0.76428571 0.84642857 0.77678571 0.80535714
        nan        nan        nan        nan        nan        nan
```

```
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
 0.78928571 0.78928571 0.75357143 0.7625     0.74642857 0.80535714
 0.76071429 0.74642857 0.80714286 0.84821429 0.7625     0.80535714
 0.75178571 0.84642857 0.78214286 0.83214286 0.78928571 0.78928571
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
 0.83214286 0.76428571 0.72142857 0.75178571 0.83214286 0.76428571
 0.7375     0.72142857 0.79107143 0.79285714 0.76428571 0.77857143
 0.70535714 0.70535714 0.77678571 0.80357143 0.80178571 0.69642857
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
 0.82142857 0.81428571 0.72142857 0.79285714 0.775      0.80357143
 0.76607143 0.80535714 0.79107143 0.72142857 0.7625     0.73392857
 0.80357143 0.83035714 0.72142857 0.80357143 0.75357143 0.83214286
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
 0.79107143 0.80357143 0.75178571 0.73571429 0.82142857 0.83214286
 0.725      0.69642857 0.82142857 0.70535714 0.69107143 0.77678571
 0.80178571 0.73571429 0.775      0.81964286 0.84464286 0.80535714
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
 0.75178571 0.78035714 0.76428571 0.72321429 0.77857143 0.83392857
 0.74821429 0.75       0.72321429 0.83392857 0.81964286 0.81964286
 0.72678571 0.80535714 0.72142857 0.76071429 0.775      0.78035714]
  warnings.warn(
Tuned hyperparameters (best parameters): {'criterion': 'gini', 'max_dept
h': 10, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_spli
t': 5, 'splitter': 'best'}
Best accuracy score for Decision Tree: 0.8892857142857142
```

TASK 9

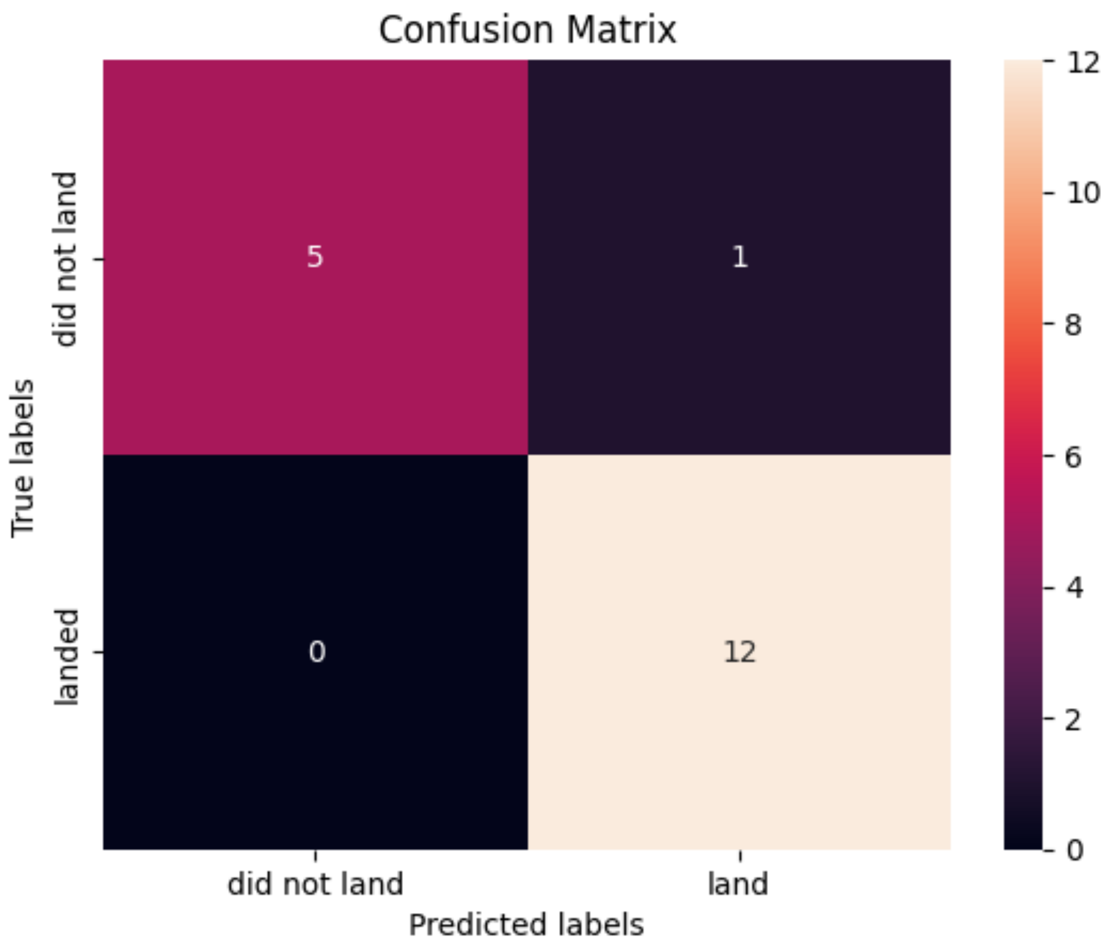
Calculate the accuracy of tree_cv on the test data using the method `score`:

In [29]:
```python
# Calculate the accuracy on the test data for Decision Tree
tree_test_accuracy = tree_cv.score(X_test, Y_test)

# Print the test accuracy for Decision Tree
print("Test Accuracy for Decision Tree:", tree_test_accuracy)
```

```
Test Accuracy for Decision Tree: 0.9444444444444444
```

We can plot the confusion matrix

In [30]:
```python
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 [31]:
```
# Define the parameter grid for KNN
parameters = {
    'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'p': [1, 2]
}

# Step 1: Create a KNN object
KNN = KNeighborsClassifier()

# Step 2: Create a GridSearchCV object for KNN
knn_cv = GridSearchCV(KNN, parameters, cv=10)

# Step 3: Fit the GridSearchCV object to the training data
knn_cv.fit(X_train, Y_train)

# Step 4: Output the best parameters and accuracy score for KNN
print("Tuned hyperparameters (best parameters):", knn_cv.best_params_)
print("Best accuracy score for KNN:", knn_cv.best_score_)
```

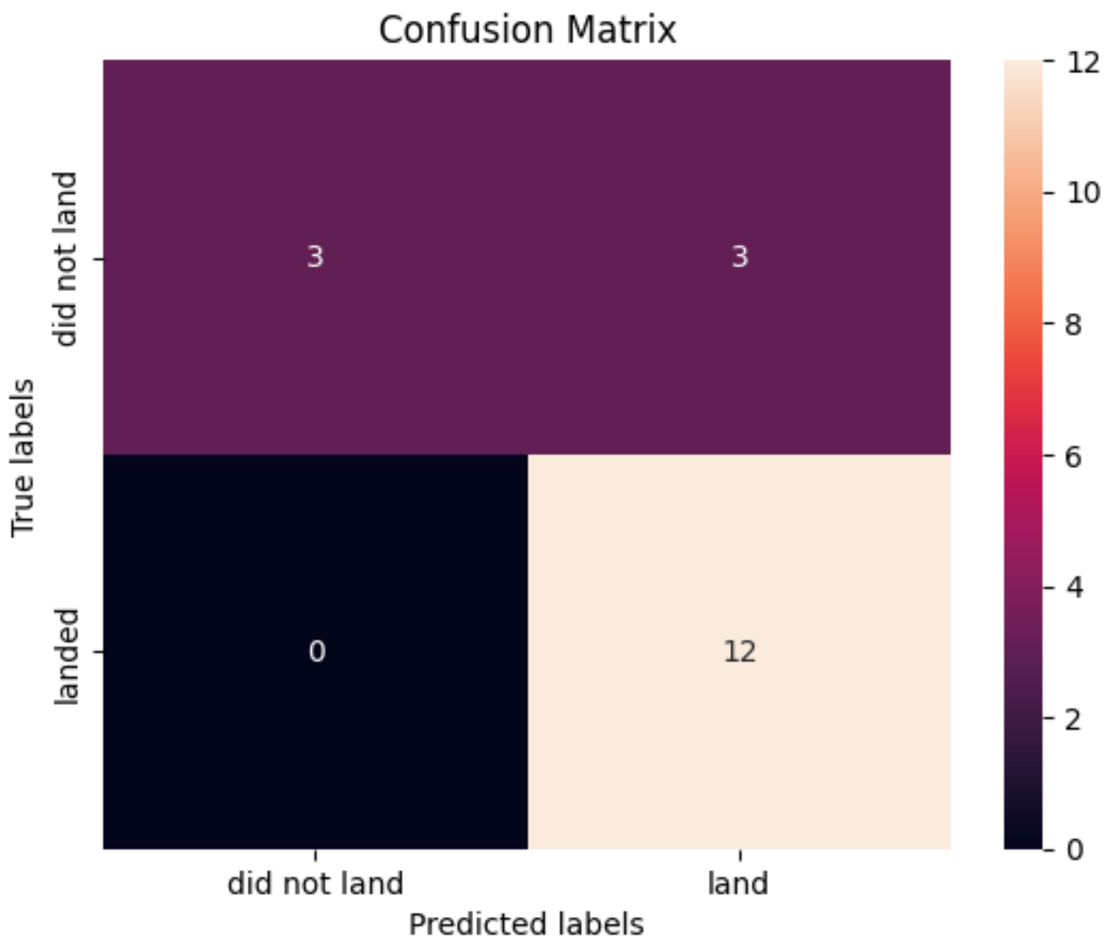
```
Tuned hyperparameters (best parameters): {'algorithm': 'auto', 'n_neighbor
s': 10, 'p': 1}
Best accuracy score for KNN: 0.8482142857142858
```

TASK 11

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

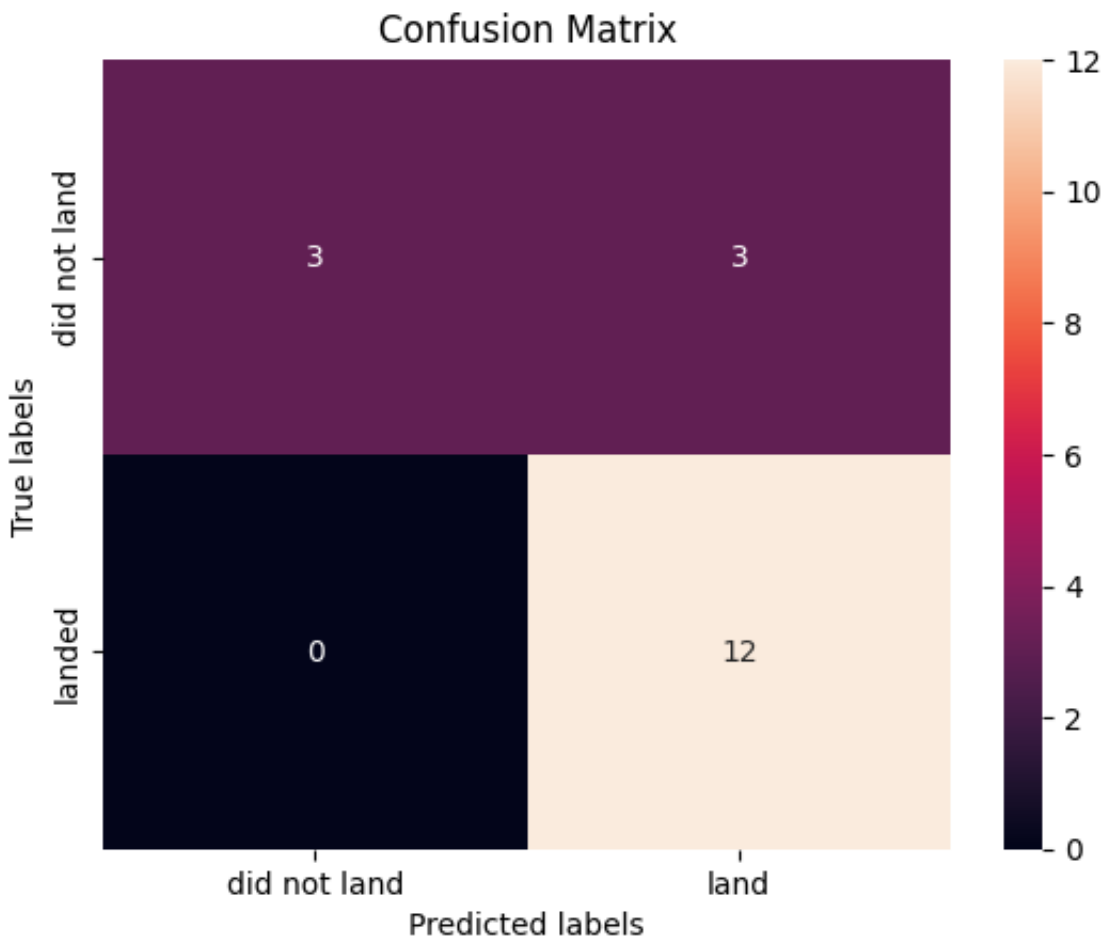
In [32]:
```
# Calculate the accuracy on the test data for KNN
knn_test_accuracy = knn_cv.score(X_test, Y_test)

# Print the test accuracy for KNN
print("Test Accuracy for KNN:", knn_test_accuracy)

# Generate predictions using the test set
yhat = knn_cv.predict(X_test)

# Plot the confusion matrix
plot_confusion_matrix(Y_test, yhat)
```

```
Test Accuracy for KNN: 0.8333333333333334
```



We can plot the confusion matrix

In [33]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



TASK 12

Find the method performs best:

In [41]:
```
# Compare the test accuracy of all models
print("Logistic Regression Test Accuracy:", test_accuracy)
print("SVM Test Accuracy:", svm_test_accuracy)
print("Decision Tree Test Accuracy:", tree_test_accuracy)
print("KNN Test Accuracy:", knn_test_accuracy)

# Determine the best method based on accuracy
best_model = max(test_accuracy, svm_test_accuracy, tree_test_accuracy, kn

print("The best performing method is:", best_model)
```

```
Logistic Regression Test Accuracy: 0.8333333333333334
SVM Test Accuracy: 0.8333333333333334
Decision Tree Test Accuracy: 0.9444444444444444
KNN Test Accuracy: 0.8333333333333334
The best performing method is: 0.9444444444444444
```

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|--!>