Space X Falcon 9 First Stage Landing Prediction

Hands on Lab: Complete the Machine Learning Prediction lab

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 [ ]:
```
!pip install numpy
!pip install pandas
!pip install seaborn
!pip install scikit-learn
```

We will import the following libraries for the lab

In [1]:
```
# Pandas is a software library written for the Python programming l
import pandas as pd
# NumPy is a library for the Python programming language, adding su
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib
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 fin
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 [2]:
```
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 c
    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
    plt.show()
```

Load the dataframe

Load the data

In [3]:
```
data = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-stor
```

In [4]:
```
data.head()
```

Out[4]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	O
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	

In [5]:
```
X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage
```

In [6]:
```
X.head(100)
```

Out[6]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	C
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 [7]:
```
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 [8]:
```
# 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`.

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 [9]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size
```

we can see we only have 18 test samples.

In [10]:
```
Y_test.shape
```

Out[10]:
```
(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 [11]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

In [12]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}
lr=LogisticRegression()
```

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

Out[12]:

```
▸ GridSearchCV

▸ best_estimator_:
  LogisticRegression

  ▸ 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 [13]:

```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_pa
print("accuracy :",logreg_cv.best_score_)
```

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

TASK 5

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

In [14]:

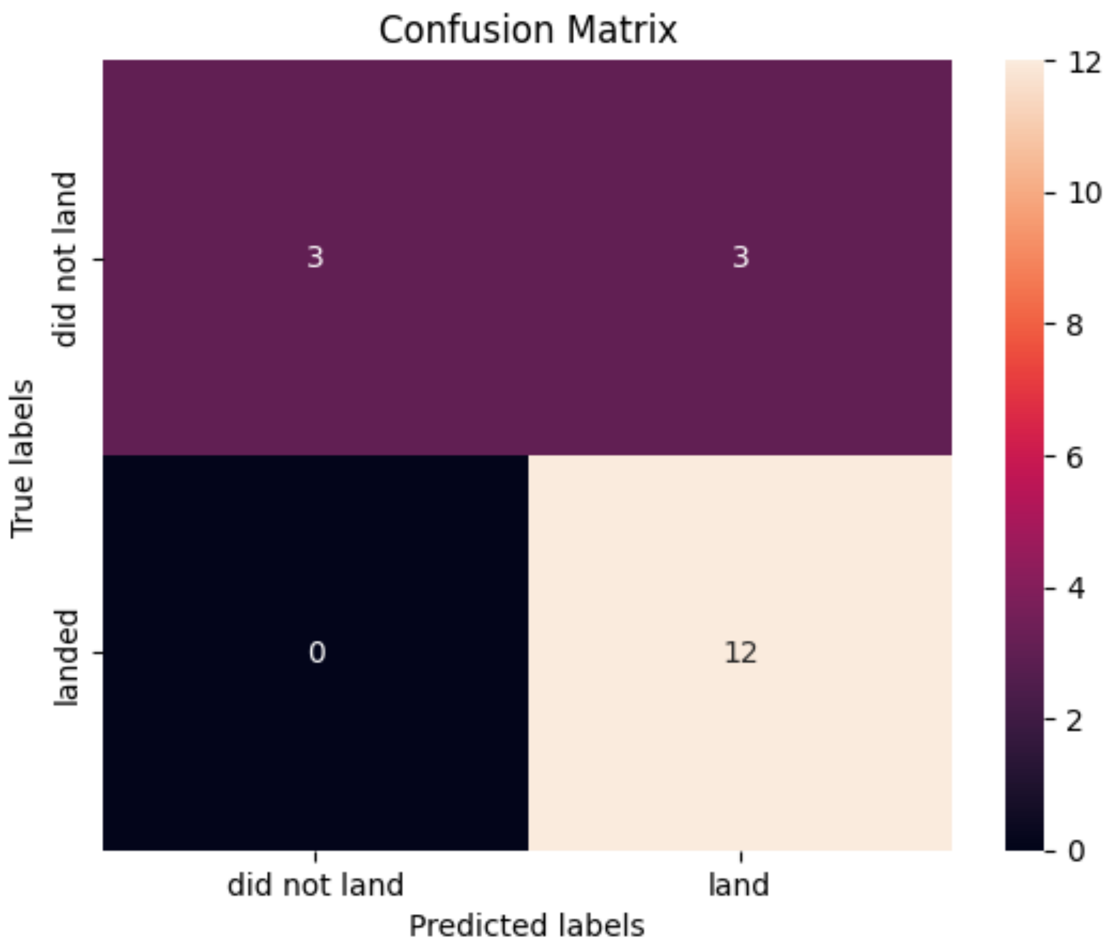
```
accuracy = logreg_cv.score(X_test, Y_test)
print("Accuracy on test data:", accuracy)
```

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

Lets look at the confusion matrix:

In [15]:

```
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 [16]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

In [17]:
```
svm_cv = GridSearchCV(svm, parameters, cv=10)
svm_cv.fit(X_train, Y_train)
```

Out[17]:

```
▸ GridSearchCV
▸ best_estimator_: SVC
  ▸ SVC
```

In [18]:

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

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.0316
2277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

TASK 7

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

In [19]:

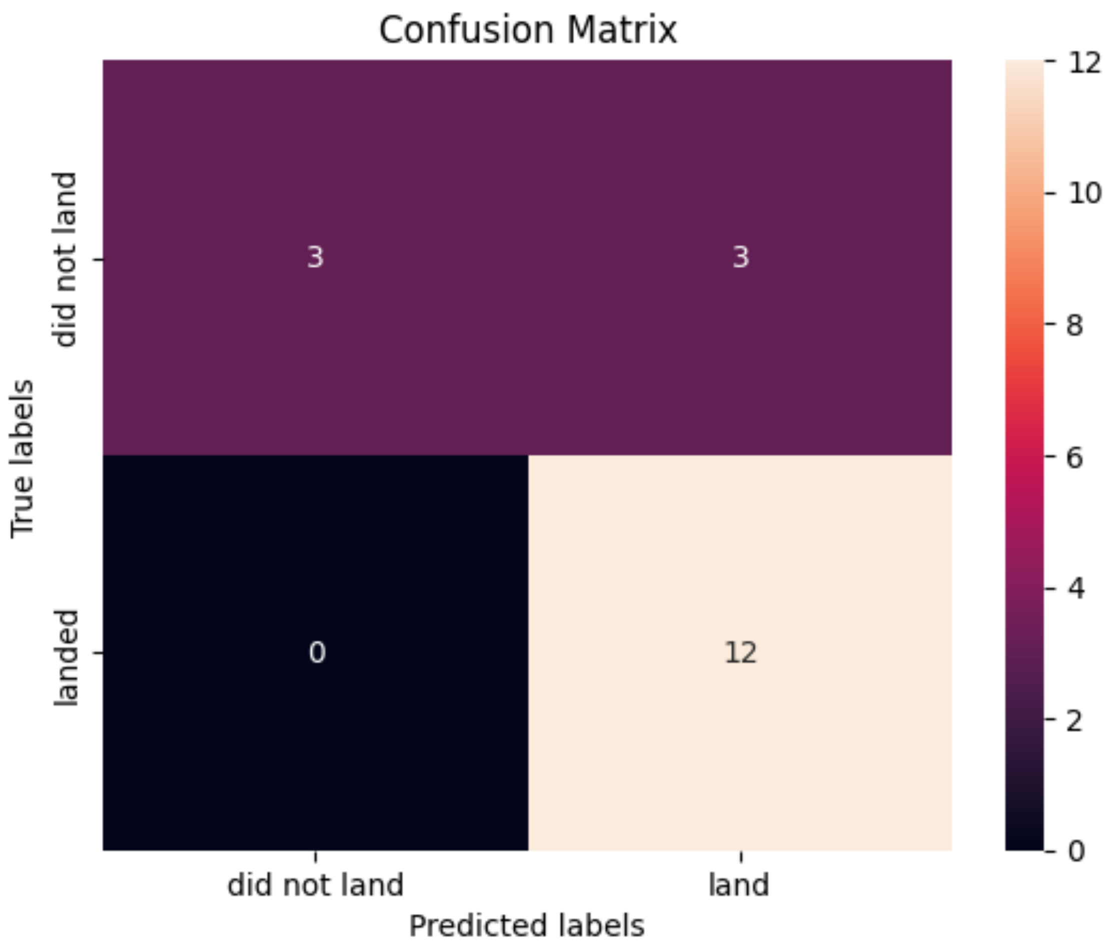
```
accuracy = svm_cv.score(X_test, Y_test)
print("Accuracy on test data:", accuracy)
```

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

We can plot the confusion matrix

In [20]:

```
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 [21]:
```
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()
```

In [22]:
```
tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, Y_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/si
te-packages/sklearn/model_selection/_validation.py:528: FitFailedWar
ning:
3240 fits failed out of a total of 6480.
The score on these train-test partitions for these parameters will b
e set to nan.
If these failures are not expected, you can try to debug them by set
ting error_score='raise'.
```

```
Below are more details about the failures:
--------------------------------------------------------------------------------
3240 fits failed with the following error:
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/model_selection/_validation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/base.py", line 1382, in wrapper
    estimator._validate_params()
  File "/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/base.py", line 436, in _validate_params
    validate_parameter_constraints(
  File "/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/utils/_param_validation.py", line 98, 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)
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/model_selection/_search.py:1108: UserWarning: 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
 0.83392857 0.77678571 0.73214286 0.69642857 0.73392857 0.79107143
 0.83214286 0.76785714 0.69642857 0.77678571 0.7875     0.84821429
 0.75178571 0.80892857 0.775      0.74821429 0.72321429 0.73928571
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.76071429 0.84285714 0.7625     0.76607143 0.80535714 0.79285714
 0.79107143 0.80535714 0.80357143 0.80714286 0.83214286 0.81785714
 0.73928571 0.83571429 0.72321429 0.84642857 0.7625     0.84464286
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.775      0.76607143 0.80714286 0.84642857 0.71071429 0.79642857
 0.81964286 0.80535714 0.78928571 0.76428571 0.81785714 0.79107143
 0.74642857 0.8375     0.7375     0.78035714 0.74821429 0.775
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.75178571 0.75357143 0.79107143 0.80357143 0.73214286 0.72142857
 0.70535714 0.74821429 0.76071429 0.81785714 0.76071429 0.7625
 0.70535714 0.80357143 0.81785714 0.79107143 0.78571429 0.70714286
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
```

```
 0.86071429 0.7625     0.74821429 0.76428571 0.82142857 0.77678571
 0.85       0.77678571 0.68214286 0.81964286 0.73928571 0.83214286
 0.83214286 0.77678571 0.81785714 0.66428571 0.77678571 0.74464286
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.79107143 0.76428571 0.80535714 0.75178571 0.74821429 0.84642857
 0.7625     0.7625     0.78928571 0.78928571 0.80714286 0.775
 0.77678571 0.76071429 0.73392857 0.775      0.71964286 0.84821429
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.80714286 0.80714286 0.83214286 0.75       0.79285714 0.80357143
 0.72142857 0.80535714 0.75       0.72142857 0.79285714 0.78928571
 0.83392857 0.76071429 0.83392857 0.80535714 0.76785714 0.81071429
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.78928571 0.73571429 0.83392857 0.75       0.73392857 0.82142857
 0.73571429 0.80714286 0.75178571 0.81785714 0.81785714 0.86071429
 0.81428571 0.83214286 0.74107143 0.77678571 0.77857143 0.77678571
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.69285714 0.76428571 0.68214286 0.73928571 0.80535714 0.83392857
 0.79107143 0.74642857 0.7625     0.77678571 0.71964286 0.80714286
 0.77678571 0.77678571 0.77678571 0.77678571 0.65357143 0.78035714
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.79464286 0.78928571 0.78214286 0.86071429 0.72142857 0.76071429
 0.73392857 0.78214286 0.79107143 0.74642857 0.77678571 0.79285714
 0.77678571 0.7375     0.79107143 0.81964286 0.77678571 0.70357143
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.84642857 0.84821429 0.79107143 0.83214286 0.88928571 0.75
 0.79107143 0.7625     0.80357143 0.75       0.80535714 0.80714286
 0.78928571 0.775      0.7625     0.78035714 0.73392857 0.8625
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.73571429 0.81964286 0.80714286 0.79285714 0.79107143 0.7625
 0.73392857 0.84642857 0.75178571 0.70535714 0.76607143 0.7625
 0.79285714 0.80535714 0.77678571 0.775      0.85892857 0.80535714
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.74642857 0.81964286 0.84464286 0.86071429 0.80535714 0.79285714
 0.74821429 0.80892857 0.81785714 0.81964286 0.80535714 0.775
 0.68035714 0.79107143 0.69285714 0.83392857 0.75       0.74821429
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.73571429 0.7375     0.81964286 0.76428571 0.74821429 0.80535714
 0.79107143 0.81785714 0.80357143 0.76071429 0.80714286 0.74642857
```

```
 0.71071429 0.83214286 0.70535714 0.79107143 0.80535714 0.80535714
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.77678571 0.7625     0.77678571 0.78928571 0.79464286 0.79107143
 0.71964286 0.77678571 0.80714286 0.80714286 0.81964286 0.78928571
 0.76071429 0.77678571 0.7625     0.80892857 0.77857143 0.83392857
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.77857143 0.74821429 0.83214286 0.79285714 0.75178571 0.84821429
 0.72321429 0.7375     0.77678571 0.69642857 0.77678571 0.83392857
 0.80535714 0.78035714 0.78928571 0.76785714 0.74642857 0.80357143
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.79107143 0.79285714 0.80535714 0.68214286 0.76428571 0.77857143
 0.75       0.79107143 0.80714286 0.73392857 0.7625     0.72142857
 0.80535714 0.79107143 0.75       0.7625     0.79107143 0.80357143
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.79107143 0.83392857 0.79464286 0.72142857 0.74642857 0.80535714
 0.81964286 0.69464286 0.74821429 0.76071429 0.76071429 0.74642857
 0.73928571 0.84642857 0.82142857 0.80357143 0.81785714 0.81964286]
  warnings.warn(
```

Out[22]:

```
▸ GridSearchCV
▸ best_estimator_: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

In [23]:

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', '
max_depth': 4, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_s
amples_split': 10, 'splitter': 'best'}
accuracy : 0.8892857142857142
```

TASK 9

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

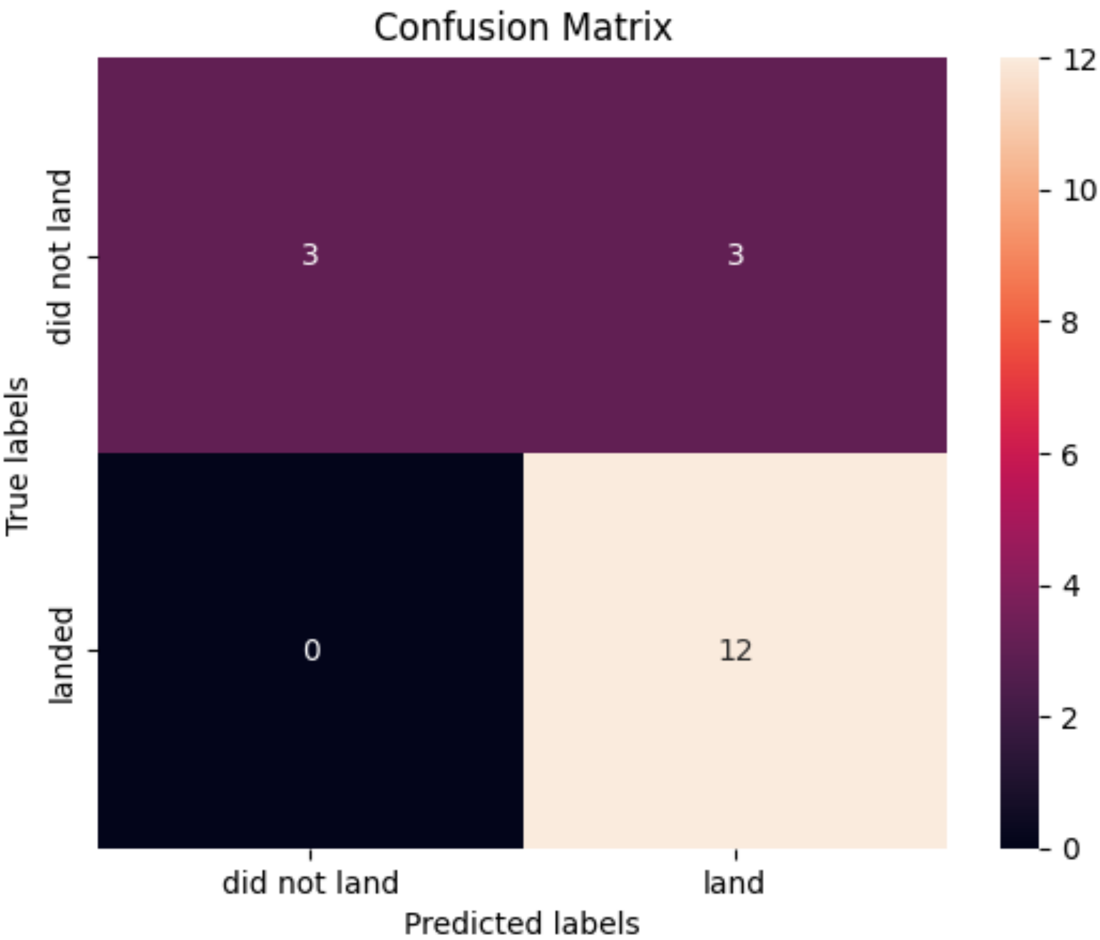
In [24]:

```python
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 [25]:
```
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 [26]:
```
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 [27]:
```
knn_cv = GridSearchCV(KNN, parameters, cv=10)
knn_cv.fit(X_train, Y_train)
```

Out[27]:

```
▸ GridSearchCV

▸ best_estimator_:
  KNeighborsClassifier

  ▸ KNeighborsClassifier
```

In [28]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_param
print("accuracy :",knn_cv.best_score_)
```

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

TASK 11

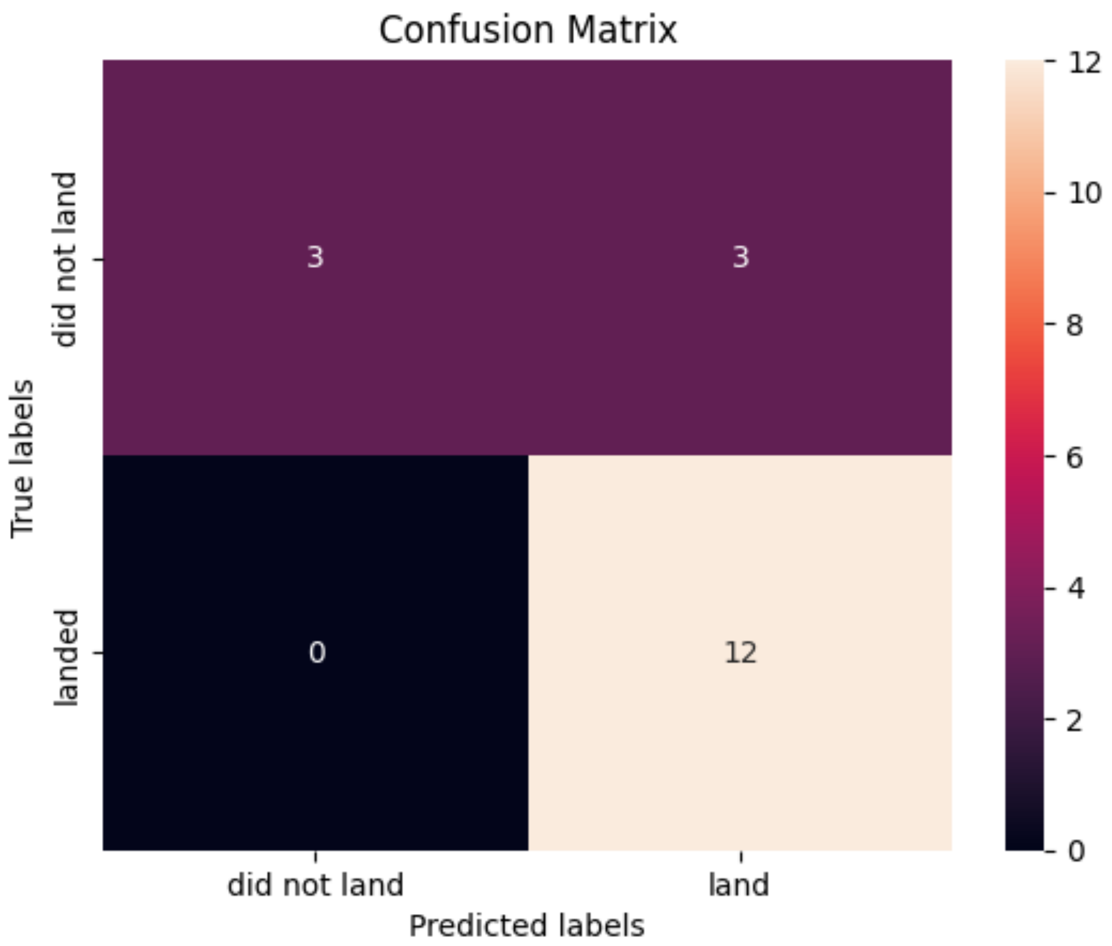
Calculate the accuracy of knn_cv on the test data using the method `score` :

In [29]:
```
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

In [30]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



TASK 12

Find the method performs best:

In [31]:
```
models = {'Logistic Regression': logreg_cv, 'SVM': svm_cv, 'Decisio
best_model = None
best_accuracy = 0

for name, model in models.items():
    accuracy = model.score(X_test, Y_test)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_model = name

print(f"The best performing model is {best_model} with an accuracy
```

```
The best performing model is Logistic Regression with an accuracy of
0.83
```

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|--!>