In [1]:
```
# Pandas is a software library written for the Python programming language for da
import pandas as pd
# NumPy is a library for the Python programming language, adding support for larg
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like p
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 one
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
```

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 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(['
```

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

# If you were unable to complete the previous lab correctly you can uncomment and

# data = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomai

data.head()
```

Out[3]:

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

In [4]:
```
X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clo

# If you were unable to complete the previous lab correctly you can uncomment and

# X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.c

X.head(100)
```

Out[4]:

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

90 rows × 83 columns

In [5]:
```
Y = data.Class.to_numpy()
```

In [6]:
```
Y
```

Out[6]:
```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 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], dtype=int64)
```

In [7]:
```
# students get this
transform = preprocessing.StandardScaler()
transform.fit(X)
```

Out[7]:
```
StandardScaler()
```

In [8]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_s
Y_test.shape
```

Out[8]:
```
(18,)
```

In [9]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

In [10]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2
lr=LogisticRegression(max_iter=1000)
logreg_cv = GridSearchCV(lr, parameters, cv=10)
logreg_cv.fit(X_train, Y_train)
```

Out[10]:
```
GridSearchCV(cv=10, estimator=LogisticRegression(max_iter=1000),
             param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                         'solver': ['lbfgs']})
```

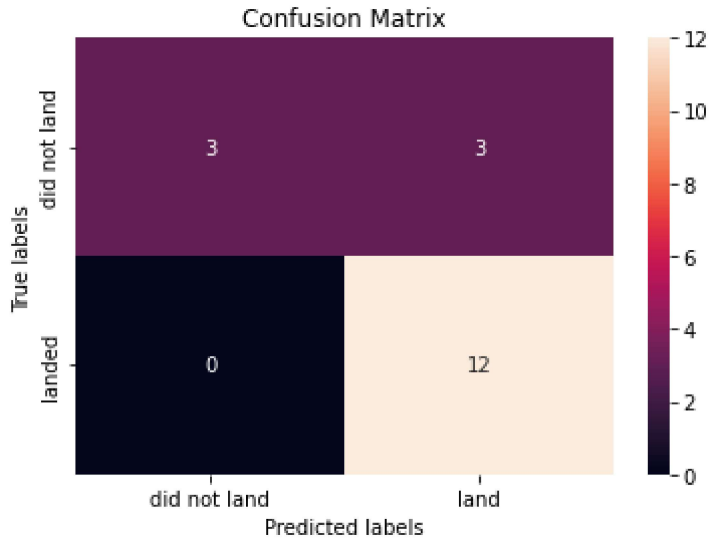
In [11]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1, 'penalty': 'l2', 'solver':
'lbfgs'}
accuracy : 0.8196428571428571
```

In [12]:
```
logreg_cv_score = logreg_cv.score(X_test, Y_test)
logreg_cv_score
```

Out[12]:
```
0.8333333333333334
```

In [13]:
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



In [14]:
```
parameters = {'kernel': ['sigmoid'],#('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

In [15]:
```
svm_cv = GridSearchCV(svm, parameters, cv=10)
svm_cv.fit(X_train, Y_train)
```

Out[15]:
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.00000000
e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.0000
0000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'kernel': ['sigmoid']})
```

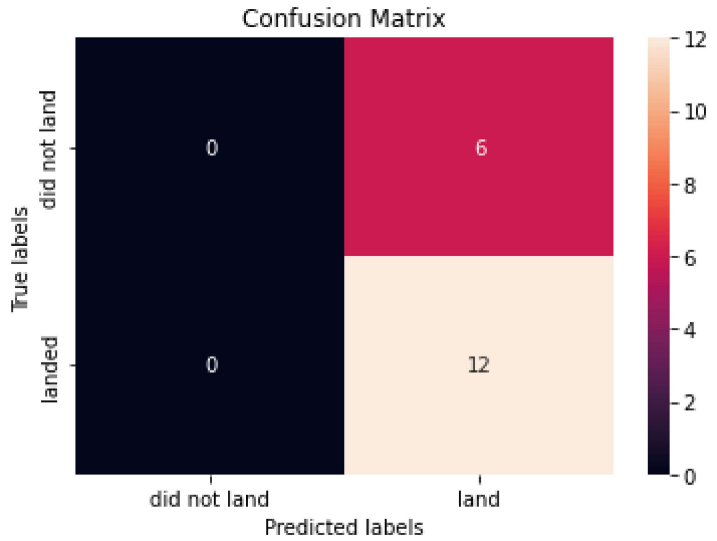
In [16]:
```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.001, 'gamma': 0.001, 'kerne
l': 'sigmoid'}
accuracy : 0.6678571428571429
```

In [17]:
```
svm_score = svm_cv.score(X_test, Y_test)
svm_score
```

Out[17]:
```
0.6666666666666666
```

In [18]:
```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



In [19]:
```
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 [20]:
```
tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_test, Y_test)
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666:
UserWarning: The least populated class in y has only 6 members, which is less t
han n_splits=10.
  warnings.warn(("The least populated class in y has only %d"
```

Out[20]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'splitter': ['best', 'random']})
```

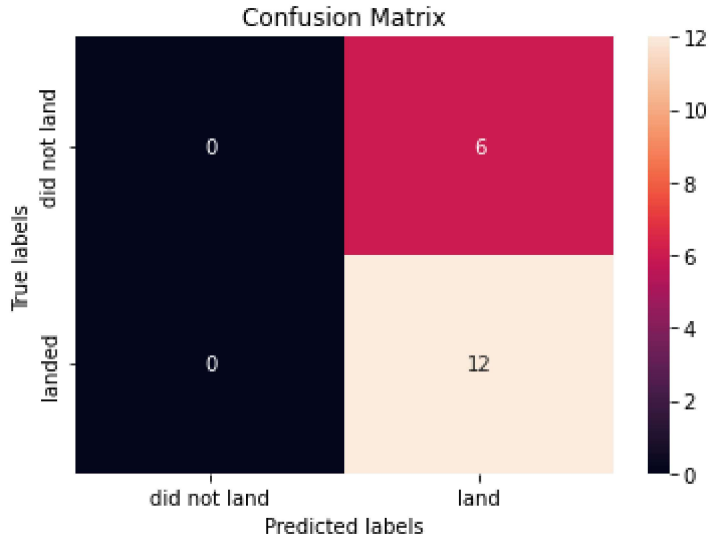
In [21]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 2,
'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitte
r': 'best'}
accuracy : 0.95
```

In [22]:
```
tree_score = tree_cv.score(X_test, Y_test)
tree_score
```

Out[22]:
```
0.9444444444444444
```

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



In [24]:
```
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 [25]:
```
KNN_cv = GridSearchCV(KNN, parameters, cv=10)
KNN_cv.fit(X_train, Y_train)
```

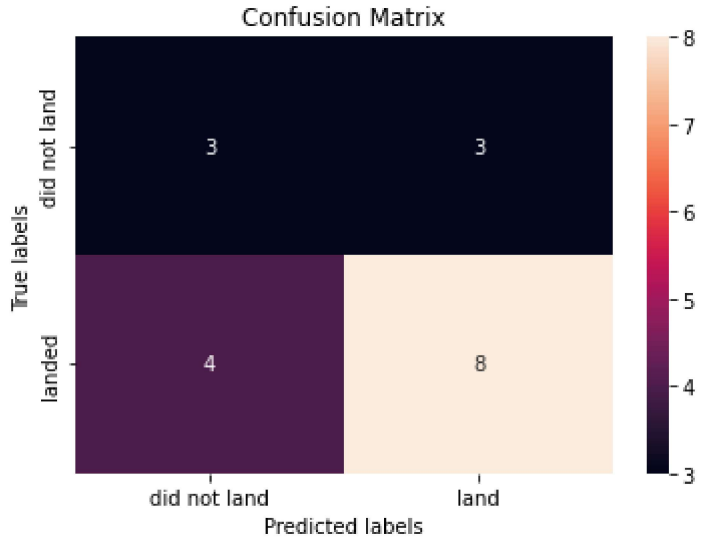
Out[25]:
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brut
e'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]})
```

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

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors':
3, 'p': 1}
accuracy : 0.6642857142857143
```

In [27]:
```
KNN_score = KNN_cv.score(X_test, Y_test)
```

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



In [29]:
```
scores = {'Logistic Regression': logreg_cv_score, 'SVM': svm_score, 'Decision Tre
print("The best model is: ", max(scores, key=scores.get), "with the accuracy of '
```

```
The best model is:  Decision Tree with the accuracy of  94.44444444444444 %
```