```python
import piplite
await piplite.install(['numpy'])
await piplite.install(['pandas'])
await piplite.install(['seaborn'])

# Pandas is a software library written for the Python programming
language for data manipulation and analysis.
import pandas as pd
# NumPy is a library for the Python programming language, adding
support for large, multi-dimensional arrays and matrices, along with a
large collection of high-level mathematical functions to operate on
these arrays
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a
MatLab like plotting framework. We will use this in our plotter
function to plot data.
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib.
It provides a high-level interface for drawing attractive and
informative statistical graphics
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

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(['did not land', 'landed'])
    plt.show()

from js import fetch
import io

URL1 = "https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/
dataset_part_2.csv"
resp1 = await fetch(URL1)
text1 = io.BytesIO((await resp1.arrayBuffer()).to_py())
data = pd.read_csv(text1)

data.head()

   FlightNumber        Date BoosterVersion  PayloadMass Orbit
LaunchSite  \
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

       Outcome  Flights  GridFins  Reused   Legs LandingPad  Block  \
0    None None        1     False   False  False        NaN    1.0
1    None None        1     False   False  False        NaN    1.0
2    None None        1     False   False  False        NaN    1.0
3  False Ocean        1     False   False  False        NaN    1.0
4    None None        1     False   False  False        NaN    1.0

   ReusedCount Serial   Longitude   Latitude  Class
0            0  B0003  -80.577366  28.561857      0
1            0  B0005  -80.577366  28.561857      0
2            0  B0007  -80.577366  28.561857      0
3            0  B1003 -120.610829  34.632093      0
4            0  B1004  -80.577366  28.561857      0

URL2 = 'https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/
dataset_part_3.csv'
resp2 = await fetch(URL2)
text2 = io.BytesIO((await resp2.arrayBuffer()).to_py())
X = pd.read_csv(text2)

X.head(100)
```

```
    FlightNumber   PayloadMass  Flights  Block  ReusedCount  Orbit_ES-
L1  \
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

    Orbit_GEO  Orbit_GTO  Orbit_HEO  Orbit_ISS  ...  Serial_B1058  \
0         0.0        0.0        0.0        0.0  ...           0.0
1         0.0        0.0        0.0        0.0  ...           0.0
2         0.0        0.0        0.0        1.0  ...           0.0
3         0.0        0.0        0.0        0.0  ...           0.0
4         0.0        1.0        0.0        0.0  ...           0.0
..        ...        ...        ...        ...  ...           ...
85        0.0        0.0        0.0        0.0  ...           0.0
86        0.0        0.0        0.0        0.0  ...           1.0
87        0.0        0.0        0.0        0.0  ...           0.0
88        0.0        0.0        0.0        0.0  ...           0.0
89        0.0        0.0        0.0        0.0  ...           0.0

    Serial_B1059  Serial_B1060  Serial_B1062  GridFins_False
GridFins_True  \
0            0.0           0.0           0.0             1.0
0.0
1            0.0           0.0           0.0             1.0
0.0
2            0.0           0.0           0.0             1.0
0.0
3            0.0           0.0           0.0             1.0
0.0
4            0.0           0.0           0.0             1.0
0.0
```

```
..          ...          ...          ...          ...
...
85          0.0          1.0          0.0          0.0
1.0
86          0.0          0.0          0.0          0.0
1.0
87          0.0          0.0          0.0          0.0
1.0
88          0.0          1.0          0.0          0.0
1.0
89          0.0          0.0          1.0          0.0
1.0

    Reused_False  Reused_True  Legs_False  Legs_True
0            1.0          0.0         1.0        0.0
1            1.0          0.0         1.0        0.0
2            1.0          0.0         1.0        0.0
3            1.0          0.0         1.0        0.0
4            1.0          0.0         1.0        0.0
..           ...          ...         ...        ...
85           0.0          1.0         0.0        1.0
86           0.0          1.0         0.0        1.0
87           0.0          1.0         0.0        1.0
88           0.0          1.0         0.0        1.0
89           1.0          0.0         0.0        1.0

[90 rows x 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']).

```
y = data['Class']
type(y)

pandas.core.series.Series

# It seems weird to convert a column to a numpy array, then reconvert it to a pandas series
# we can immediately extract the pandas series from the dataframe
y = X['Class']
X.drop(['Class'], axis=1, inplace=True)
type(y)
```

TASK 2

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

```
from sklearn import preprocessing as prep

# students get this
transform = preprocessing.StandardScaler()

X = prep.StandardScaler().fit_transform(X)
X
```

```
array([[-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

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=2)
y_test.shape # we have 18 samples
```

```
(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.

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.

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

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

# details of parameters
#
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model
.LogisticRegression.html
parameters = {"C":[0.01,0.1,1], 'penalty':['l2'], 'solver':['lbfgs']}

lr = LogisticRegression()

logreg_cv = GridSearchCV(lr, parameters, cv=10)
logreg_cv.fit(X, y)
logreg_cv.best_estimator_

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 : 0.8222222222222222
```

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_.

```
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 : 0.8222222222222222
```

TASK 5

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

```
print('score on train data: ', logreg_cv.score(X_train, y_train))  #
R² score on train data
print('score on test data : ', logreg_cv.score(X_test, y_test))  # R²
score on test data
```

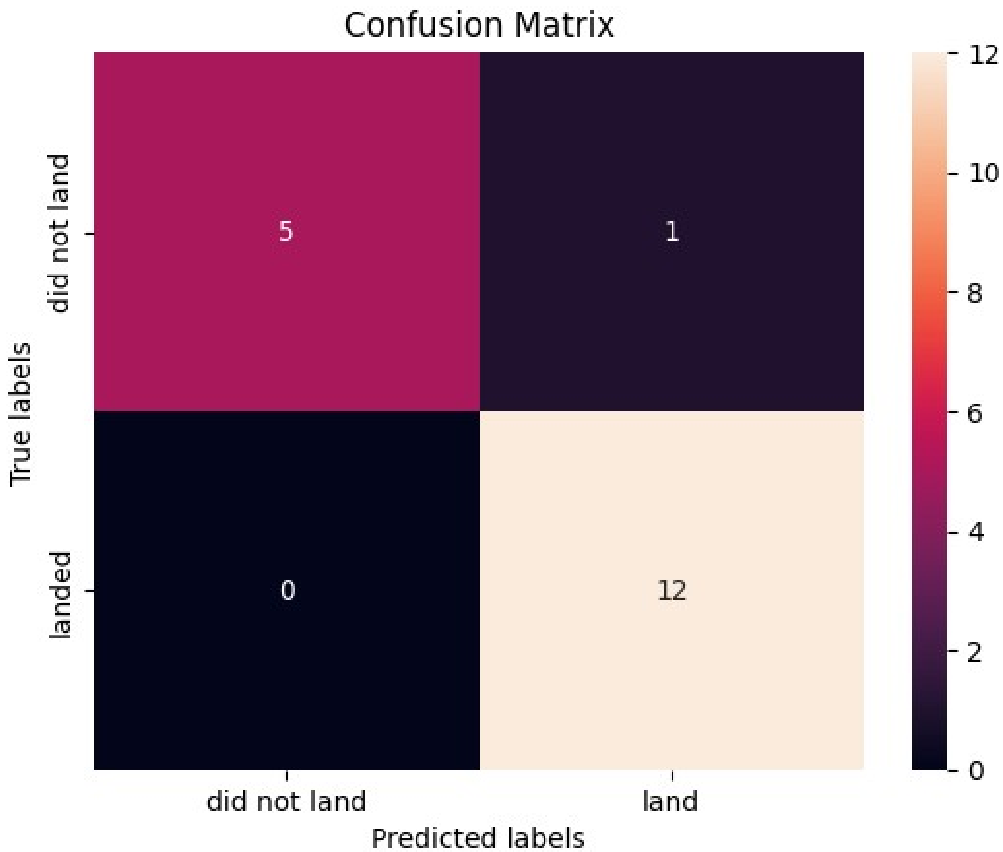
```
score on train data:  0.875
score on test data :  0.9444444444444444
```

Lets look at the confusion matrix:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=2)
y_test.shape # we have 18 samples
```

```
(18,)
```

```
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.

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

```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}  # from 10^-3 to 10^3 in 6 steps with equal quotients
svm = SVC()

svm_cv = GridSearchCV(svm, parameters, cv=10)
svm_cv.fit(X, y)
svm_cv.best_estimator_

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 : 0.8222222222222223

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 : 0.8222222222222223
```

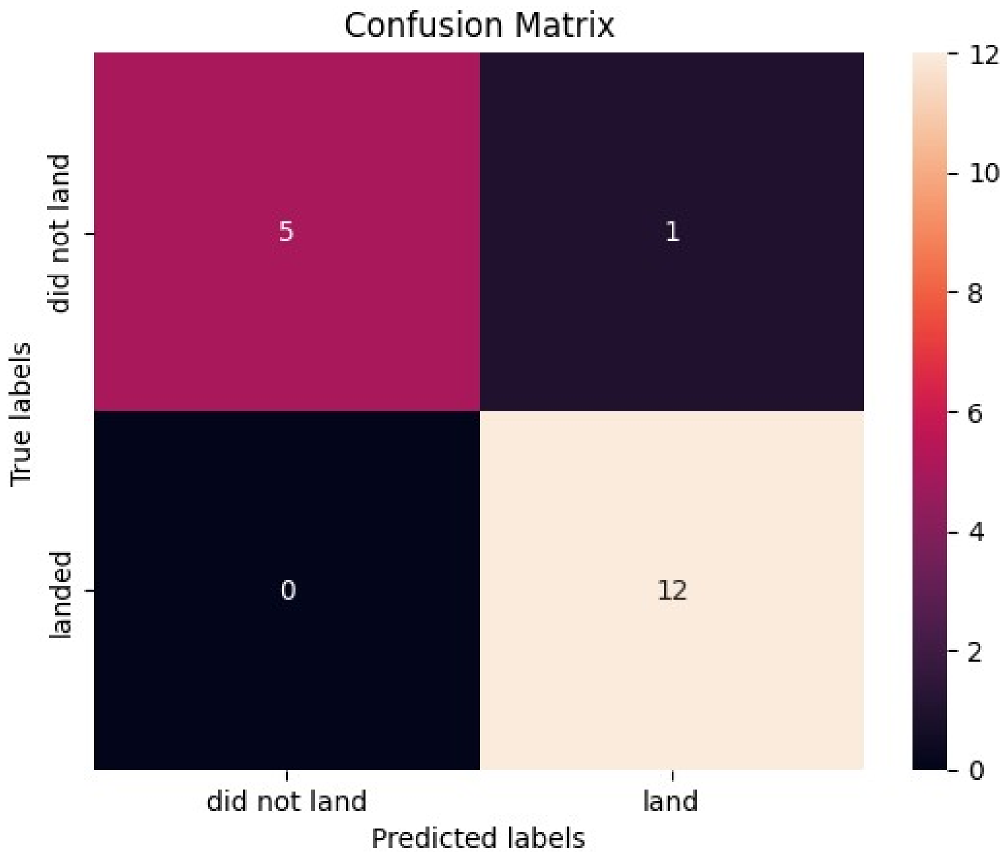
TASK 7

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

```
print('score on train data: ', svm_cv.score(X_train, y_train))  # R² score on train data
print('score on test data : ', svm_cv.score(X_test, y_test))  # R² score on test data

score on train data:  0.8611111111111112
score on test data :  0.9444444444444444
```

We can plot the confusion matrix

```
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.

```
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()

print("tuned hpyerparameters :(best parameters)
",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)

tuned hpyerparameters :(best parameters)  {'criterion': 'gini',
'max_depth': 6, 'max_features': 'sqrt', 'min_samples_leaf': 2,
'min_samples_split': 5, 'splitter': 'best'}
accuracy : 0.888888888888889
```

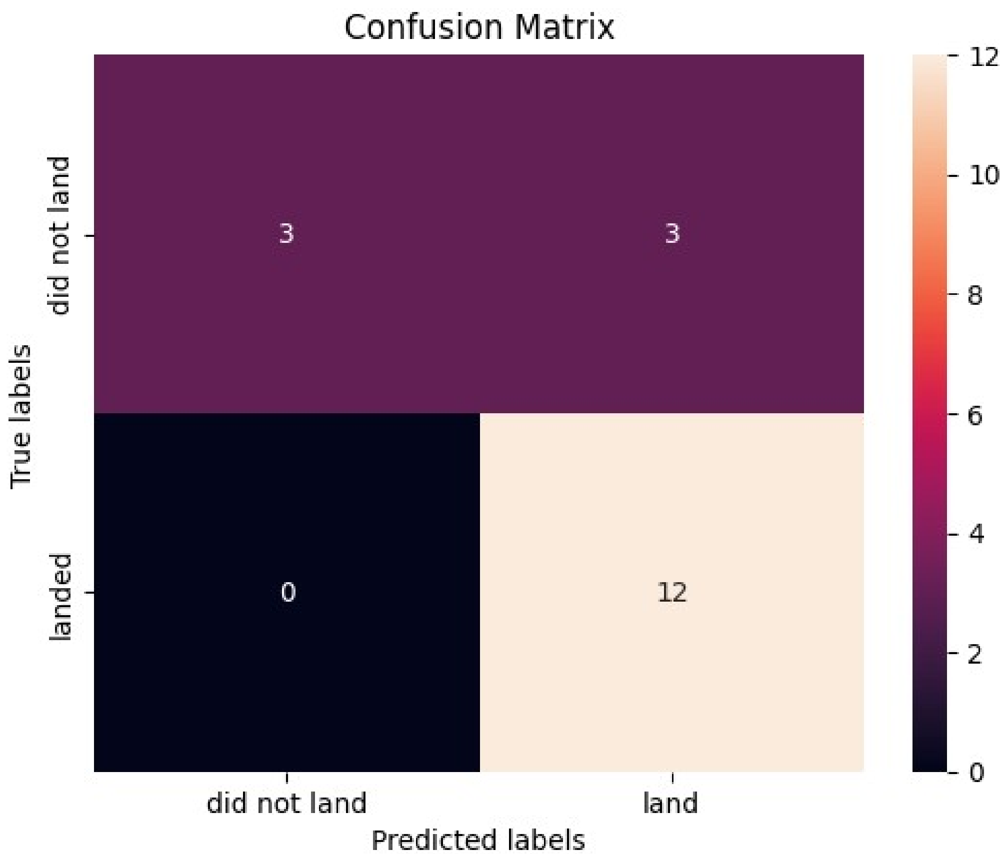
TASK 9

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

```
print('score on train data: ', tree_cv.score(X_train, y_train))  # R²
score on train data
print('score on test data : ', tree_cv.score(X_test, y_test))  # R²
score on test data

score on train data:  0.8472222222222222
score on test data :  0.8333333333333334
```

We can plot the confusion matrix

```
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.

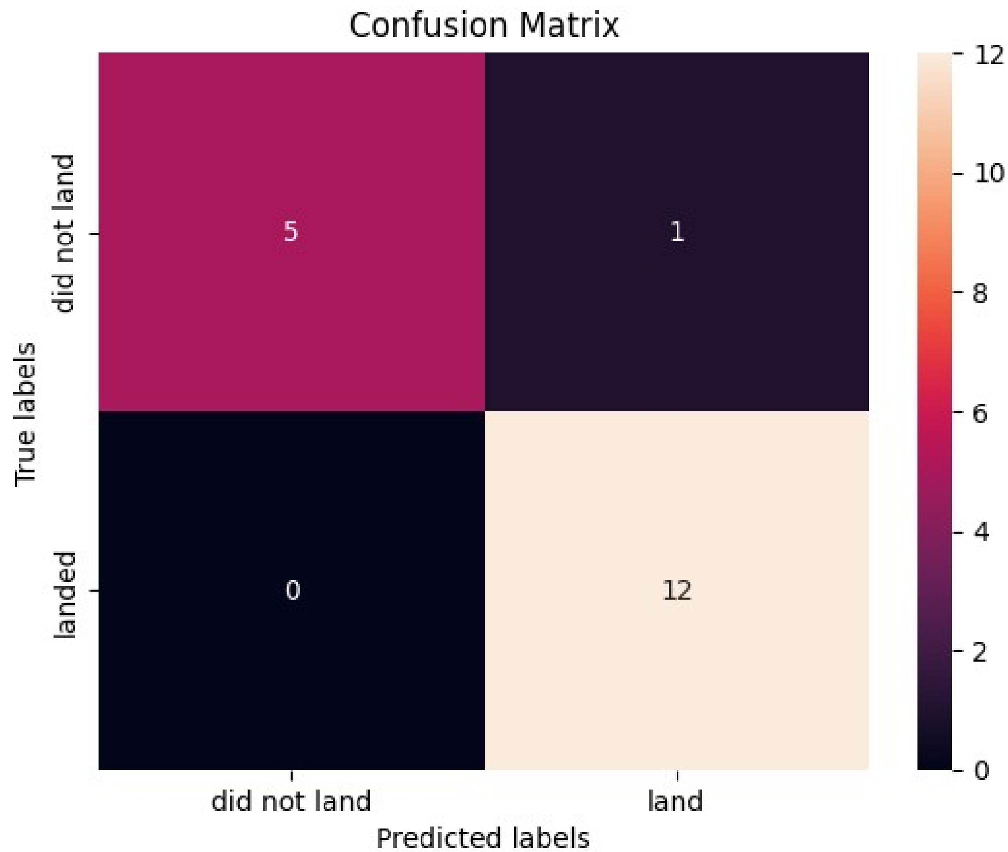
```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()

knn_cv = GridSearchCV(KNN, parameters, cv=10)
knn_cv.fit(X, y)
knn_cv.best_estimator_

print("tuned hpyerparameters :(best parameters) ",
knn_cv.best_params_)
print("accuracy :", knn_cv.best_score_)

tuned hpyerparameters :(best parameters)  {'algorithm': 'auto',
'n_neighbors': 5, 'p': 1}
accuracy : 0.8444444444444444

print('score on train data: ', knn_cv.score(X_train, y_train))  # R²
score on train data
print('score on test data : ', knn_cv.score(X_test, y_test))  # R²
score on test data

score on train data:  0.875
score on test data :  0.9444444444444444
```

TASK 11

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

We can plot the confusion matrix

```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(y_test,yhat)
```



ASK 12

Find the method performs best:

```
Classification model:   Accuracy      Score on Train data
Score on Test data
Logistic Regression       0.822            0.875
0.944
SVM                       0.822            0.861
0.944
Decision Tree             0.888            0.847
0.833
KNN                       0.844            0.875
0.944
```

We can see that the classification models such as Logistic Regression,Support vector machine(SVM_ and K_Nearest Neighbours(KNN) have the same Score on Test i.e. 0.944. Therefore Logistic Regression, Support vector machine(SVM) and K_Nearest Neighbours(KNN) are the best performing models. However KNN has the highest values of Accuracy, Score on Training data and Score on Test data. So, we can say that KNN is the best performing model among all.