Space X Falcon 9 First Stage Landing Prediction

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

```
import piplite
await piplite.install(['numpy'])
await piplite.install(['pandas'])
await piplite.install(['seaborn'])
We will import the following libraries for the lab

# 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
```

This function is to plot the confusion matrix.

```
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()
```

Load the dataframe

Load the data

```
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 BoosterVersionPayloadMass
     Orbit    LaunchSite    Outcome   Flights   GridFins
     Reused    Legs LandingPad    Block    ReusedCount
     Serial    Longitude Latitude  Class
0    1    2010-06-04    Falcon 9  6104.959412   LEO
     CCAFS SLC 40   None None 1    False    False
     False    NaN  1.0  0    B0003    -80.577366
     28.561857 0
1    2    2012-05-22    Falcon 9  525.000000    LEO
     CCAFS SLC 40   None None 1    False    False
     False    NaN  1.0  0    B0005    -80.577366
     28.561857 0
2    3    2013-03-01    Falcon 9  677.000000    ISS
     CCAFS SLC 40   None None 1    False    False
     False    NaN  1.0  0    B0007    -80.577366
     28.561857 0
3    4    2013-09-29    Falcon 9  500.000000    PO   VAFB
SLC 4E    False Ocean   1    False    False    False
     NaN  1.0  0    B1003    -120.610829   34.632093 0
4    5    2013-12-03    Falcon 9  3170.000000   GTO
     CCAFS SLC 40   None None 1    False    False
     False    NaN  1.0  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
       Orbit_GEO    Orbit_GTO    Orbit_HEO    Orbit_ISS    ...    Serial_B1058
       Serial_B1059 Serial_B1060 Serial_B1062 GridFins_False      GridFins_True
       Reused_False Reused_True  Legs_False   Legs_True
0      1.0    6104.959412  1.0    1.0    0.0    0.0    0.0    0.0    0.0    0.0
       ...    0.0    0.0    0.0    0.0    1.0    0.0    1.0    0.0    1.0    0.0
1      2.0    525.000000   1.0    1.0    0.0    0.0    0.0    0.0    0.0    0.0
       ...    0.0    0.0    0.0    0.0    1.0    0.0    1.0    0.0    1.0    0.0
2      3.0    677.000000   1.0    1.0    0.0    0.0    0.0    0.0    0.0    1.0
       ...    0.0    0.0    0.0    0.0    1.0    0.0    1.0    0.0    1.0    0.0
3      4.0    500.000000   1.0    1.0    0.0    0.0    0.0    0.0    0.0    0.0
       ...    0.0    0.0    0.0    0.0    1.0    0.0    1.0    0.0    1.0    0.0
4      5.0    3170.000000  1.0    1.0    0.0    0.0    0.0    1.0    0.0    0.0
       ...    0.0    0.0    0.0    0.0    1.0    0.0    1.0    0.0    1.0    0.0
...    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
       ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
85     86.0   15400.000000 2.0    5.0    2.0    0.0    0.0    0.0    0.0    0.0
       ...    0.0    0.0    1.0    0.0    0.0    1.0    0.0    1.0    0.0    1.0
86     87.0   15400.000000 3.0    5.0    2.0    0.0    0.0    0.0    0.0    0.0
       ...    1.0    0.0    0.0    0.0    0.0    1.0    0.0    1.0    0.0    1.0
87     88.0   15400.000000 6.0    5.0    5.0    0.0    0.0    0.0    0.0    0.0
       ...    0.0    0.0    0.0    0.0    0.0    1.0    0.0    1.0    0.0    1.0
88     89.0   15400.000000 3.0    5.0    2.0    0.0    0.0    0.0    0.0    0.0
       ...    0.0    0.0    1.0    0.0    0.0    1.0    0.0    1.0    0.0    1.0
89     90.0   3681.000000  1.0    5.0    0.0    0.0    0.0    0.0    0.0    0.0
       ...    0.0    0.0    0.0    1.0    0.0    1.0    1.0    0.0    0.0    1.0
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']).

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

TASK 2

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

```
# 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

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=2)
```

we can see we only have 18 test samples.

```
Y_test.shape
(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.

```
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()
```

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_.

```
from sklearn.model_selection import GridSearchCV

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

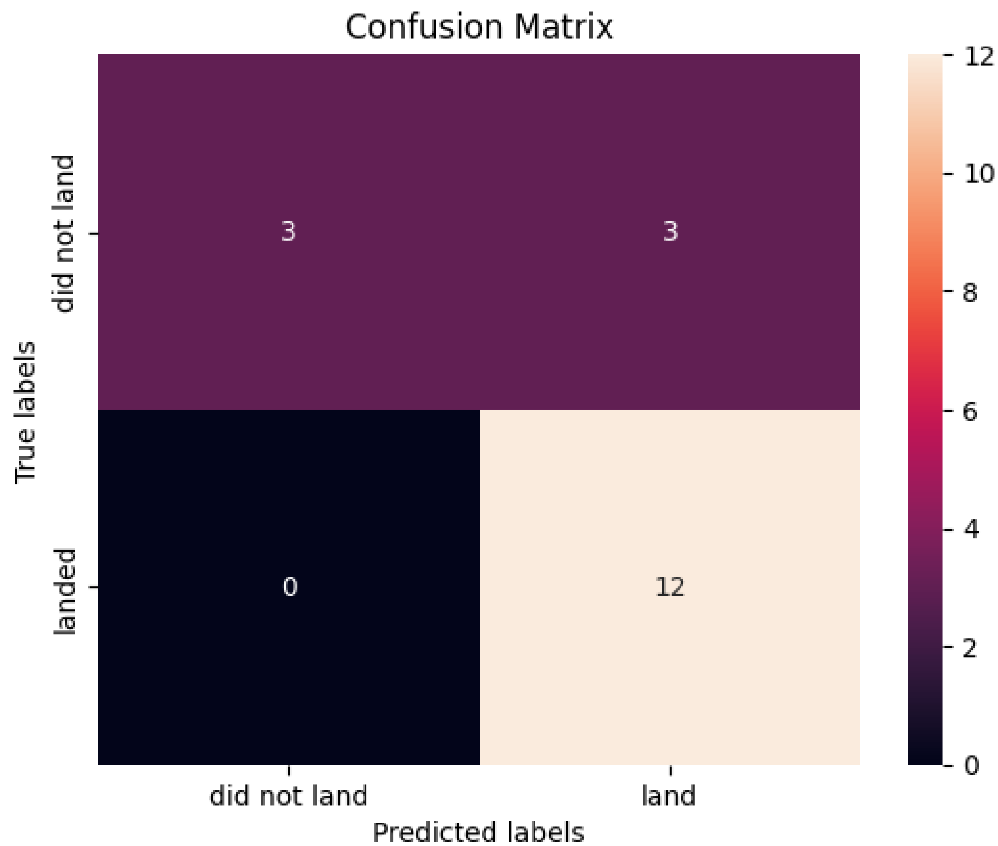
TASK 5

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

```
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:

```
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)}
svm = SVC()
from sklearn.model_selection import GridSearchCV

svm_cv = GridSearchCV(svm, parameters, cv=10)
svm_cv.fit(X_train, Y_train)

GridSearchCV?i
estimator: SVC
 SVC?

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

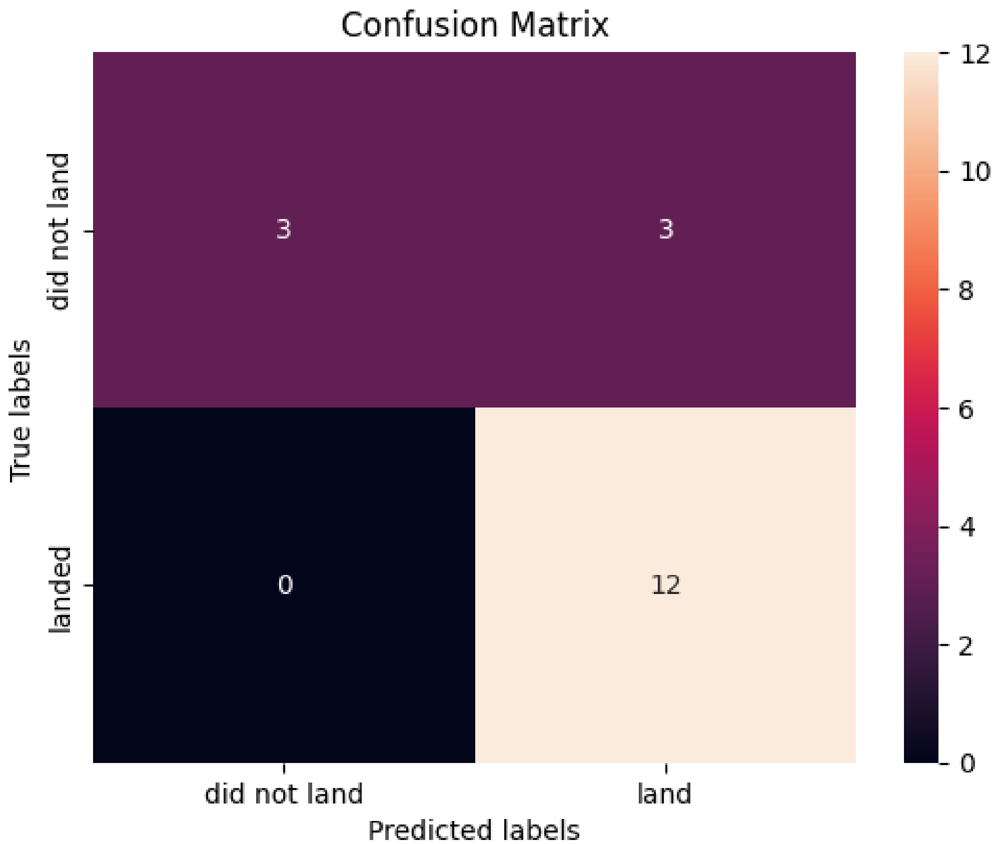
Calculate the accuracy on the test data using the method score:

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

```
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()

from sklearn.model_selection import GridSearchCV

tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, Y_train)
/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
error_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 wrapper
    estimator._validate_params()
  File "/lib/python3.12/site-packages/sklearn/base.py", line 666, in
_validate_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:
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
 0.7625     0.77678571 0.7375     0.79642857 0.74642857 0.73392857
 0.77678571 0.70892857 0.71071429 0.81964286 0.73571429 0.74821429
 0.84821429 0.84642857 0.80535714 0.75357143 0.78928571 0.69642857
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.80535714 0.83392857 0.80357143 0.84642857 0.80357143 0.80357143
 0.81964286 0.81785714 0.80357143 0.8625     0.80535714 0.82142857
 0.78214286 0.80535714 0.7625     0.81785714 0.81785714 0.80535714
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.79285714 0.80357143 0.81964286 0.79464286 0.77678571 0.83214286
 0.72142857 0.75       0.84642857 0.80357143 0.74642857 0.80535714
 0.74642857 0.76785714 0.78928571 0.81785714 0.8625     0.76071429
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
```

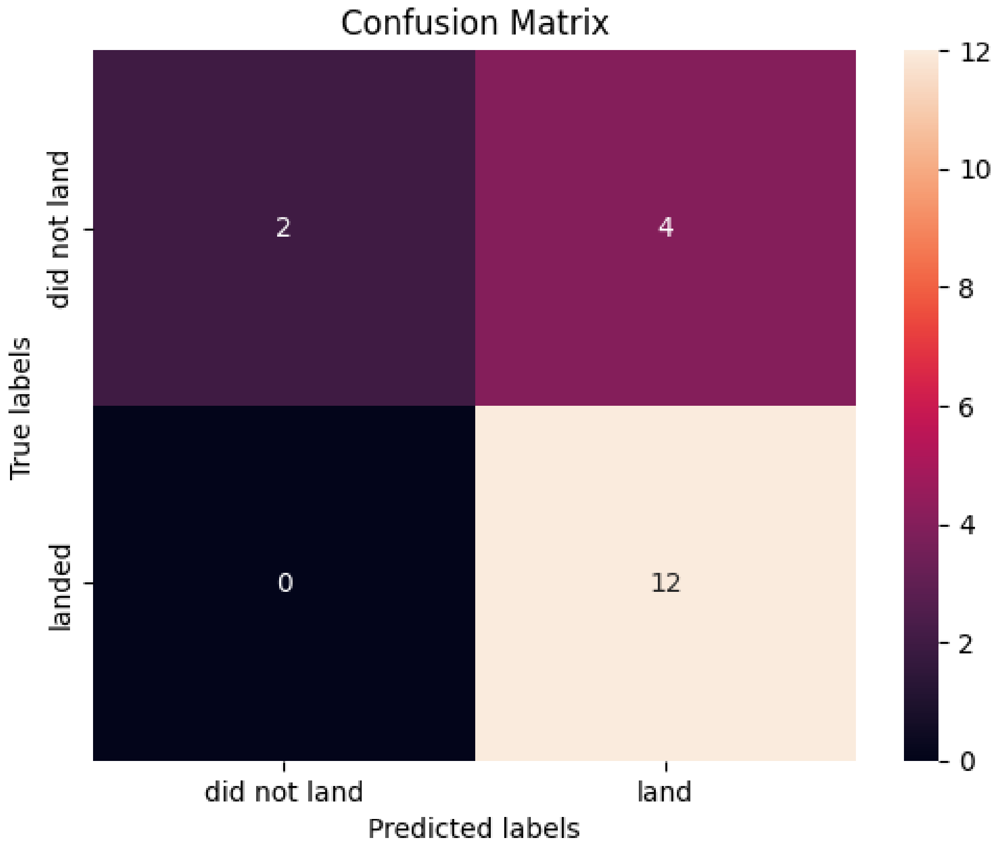
```
       nan        nan        nan        nan        nan        nan
0.775      0.79285714 0.84821429 0.81964286 0.81964286 0.78214286
0.68928571 0.80357143 0.84642857 0.76607143 0.80535714 0.74821429
0.77678571 0.80535714 0.78928571 0.80714286 0.78928571 0.77678571
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.775      0.81785714 0.73392857 0.81785714 0.73392857 0.80535714
0.72142857 0.76428571 0.81607143 0.79285714 0.75178571 0.79464286
0.81964286 0.76071429 0.75357143 0.69642857 0.73571429 0.80357143
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.74821429 0.73392857 0.7625     0.75178571 0.775      0.78928571
0.82142857 0.77678571 0.79107143 0.72857143 0.80357143 0.76785714
0.73571429 0.73392857 0.76785714 0.775      0.79464286 0.79107143
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.79285714 0.78214286 0.70535714 0.80892857 0.73571429 0.83214286
0.75       0.775      0.7375     0.74821429 0.76428571 0.79285714
0.77678571 0.76071429 0.81785714 0.775      0.80535714 0.78928571
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.72321429 0.81607143 0.79285714 0.78035714 0.78928571 0.73392857
0.775      0.75       0.83214286 0.7625     0.77857143 0.77678571
0.81428571 0.78214286 0.78035714 0.75357143 0.78928571 0.68214286
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.73928571 0.75       0.80535714 0.80357143 0.86071429 0.80535714
0.73392857 0.77678571 0.80714286 0.77678571 0.73214286 0.79285714
0.65178571 0.84821429 0.81785714 0.78035714 0.80535714 0.84821429
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.775      0.725      0.78035714 0.83392857 0.73392857 0.75178571
0.80357143 0.78928571 0.725      0.75178571 0.84821429 0.80535714
0.76071429 0.65357143 0.725      0.73928571 0.69642857 0.78214286
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.80535714 0.75357143 0.82142857 0.81785714 0.775      0.79107143
0.84464286 0.81964286 0.81964286 0.79464286 0.74642857 0.7625
0.83214286 0.76428571 0.71071429 0.81964286 0.80535714 0.80535714
       nan        nan        nan        nan        nan        nan
```

```
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.81785714 0.77678571 0.77857143 0.81964286 0.80357143 0.79285714
 0.81964286 0.69642857 0.81964286 0.83214286 0.78928571 0.73392857
 0.77678571 0.7625     0.78928571 0.79107143 0.70357143 0.78928571
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.80357143 0.79285714 0.81785714 0.74821429 0.81964286 0.79107143
 0.78928571 0.80714286 0.79107143 0.74642857 0.80357143 0.80535714
 0.775      0.77678571 0.77678571 0.84642857 0.87678571 0.81964286
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.77678571 0.81964286 0.69285714 0.76071429 0.74642857 0.76428571
 0.80357143 0.80714286 0.77678571 0.80357143 0.81785714 0.75178571
 0.78035714 0.83214286 0.75357143 0.84821429 0.78035714 0.86071429
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.75       0.76428571 0.74821429 0.77678571 0.75892857 0.80357143
 0.7375     0.78928571 0.69464286 0.77678571 0.76964286 0.78928571
 0.7625     0.7625     0.83214286 0.81607143 0.78928571 0.77678571
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.80714286 0.74107143 0.775      0.76071429 0.80535714 0.81785714
 0.72142857 0.75       0.70535714 0.73214286 0.76428571 0.775
 0.75178571 0.77678571 0.77857143 0.78214286 0.77678571 0.75
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.80535714 0.65357143 0.73392857 0.78928571 0.80357143 0.80357143
 0.78928571 0.78928571 0.7625     0.77857143 0.80535714 0.81964286
 0.77678571 0.73392857 0.77678571 0.76785714 0.78928571 0.80535714
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.73571429 0.81785714 0.79107143 0.70714286 0.77678571 0.83571429
 0.775      0.84642857 0.72142857 0.81428571 0.68928571 0.81785714
 0.73392857 0.79285714 0.80357143 0.7625     0.80535714 0.81964286]
  warnings.warn(
  GridSearchCV?i
estimator: DecisionTreeClassifier
 DecisionTreeClassifier?
```

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()
from sklearn.model_selection import GridSearchCV

knn_cv = GridSearchCV(KNN, parameters, cv=10)
knn_cv.fit(X_train, Y_train)

  GridSearchCV?i
estimator: KNeighborsClassifier
 KNeighborsClassifier?

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

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

TASK 11¶

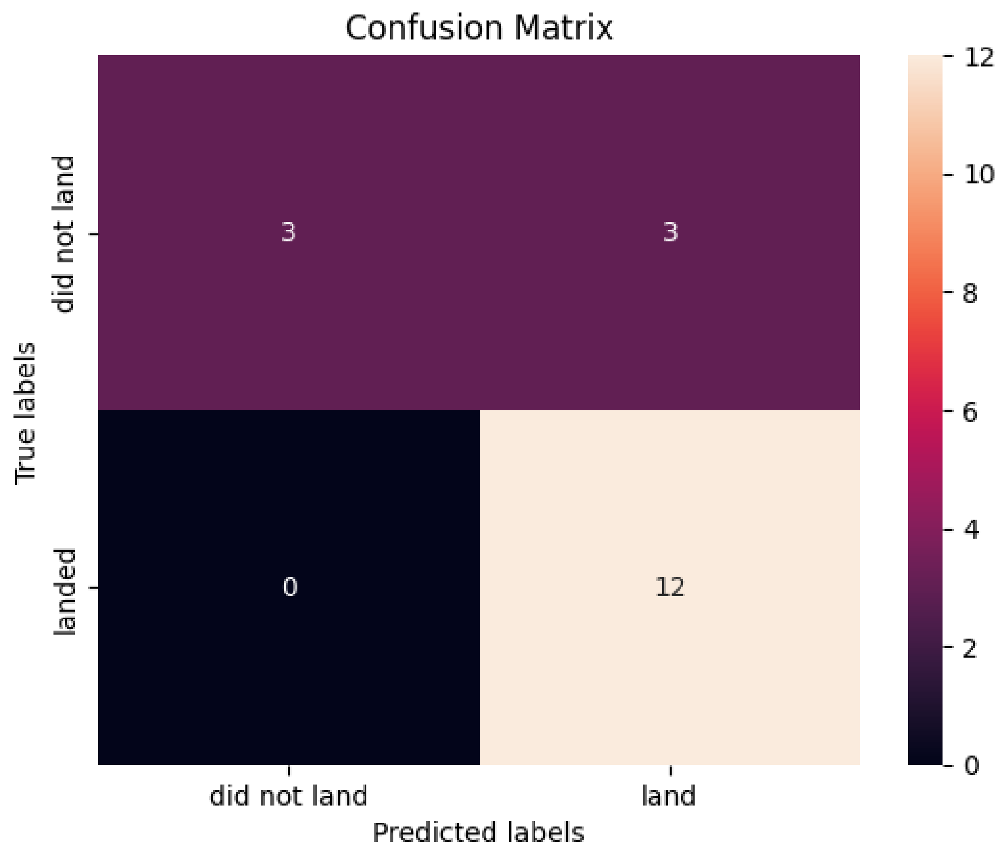
Calculate the accuracy of knn_cv on the test data using the method score:

```
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

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



TASK 12

Click to add a cell.

/DS0203EN/module_4/

Authors

Pratiksha Verma