

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()
Y[:10]
```

```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0])
```

TASK 2

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

```
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']}
```

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

```
GridSearchCV(cv=10, error_score='raise-deprecating',
            estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
            intercept_scaling=1, max_iter=100, multi_class='warn',
            n_jobs=None, penalty='l2', random_state=None, solver='warn',
            tol=0.0001, verbose=0, warm_start=False),
            fit_params=None, iid='warn', n_jobs=None,
            param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'], 'solver': ['lbfgs']},
            pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
            scoring=None, verbose=0)
```

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 hyperparameters :(best parameters) ", logreg_cv.best_params_)
print("accuracy :", logreg_cv.best_score_)
```

```
tuned hyperparameters :(best parameters) {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8472222222222222
```

TASK 5

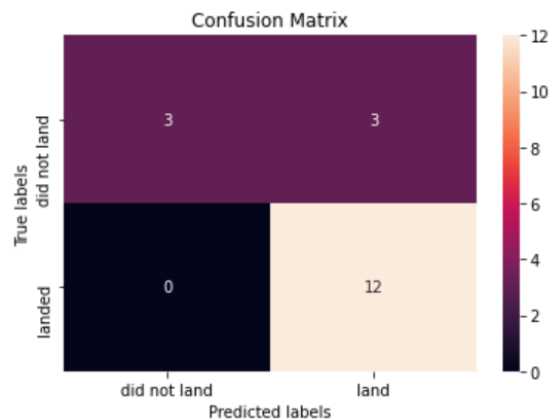
Calculate the accuracy on the test data using the method `score`:

```
logreg_cv.score(X_test, Y_test)
```

```
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 major problem is false positives.

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()
```

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

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
                           kernel='rbf', max_iter=-1, probability=False, random_state=None,
                           shrinking=True, tol=0.001, verbose=False),
             fit_params=None, iid='warn', n_jobs=None,
             param_grid={'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid'), 'C': array([1.00000e-03, 3.16228e-02, 1.00000e+00, 3.16228e+01, 1.00000e+03]), 'gamma': array([1.00000e-03, 3.16228e-02, 1.00000e+00, 3.16228e+01, 1.00000e+03])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring=None, verbose=0)
```

```
print("tuned hyperparameters :(best parameters) ", svm_cv.best_params_)
print("accuracy : ", svm_cv.best_score_)
```

```
tuned hyperparameters :(best parameters) {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8472222222222222
```

TASK 7

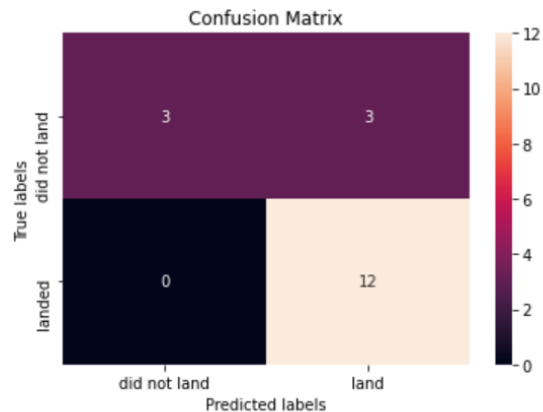
Calculate the accuracy on the test data using the method `score`:

```
svm_cv.score(X_test, Y_test)
```

```
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()
```

```
tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                                              max_features=None, max_leaf_nodes=None,
                                              min_impurity_decrease=0.0, min_impurity_split=None,
                                              min_samples_leaf=1, min_samples_split=2,
                                              min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                                              splitter='best'),
             fit_params=None, iid='warn', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'], 'splitter': ['best', 'random'], '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]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring=None, verbose=0)
```

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

```
tuned hyperparameters :(best parameters) {'criterion': 'entropy', 'max_depth': 10, 'max_features': 'sqrt', 'min_sa
mples_leaf': 2, 'min_samples_split': 2, 'splitter': 'random'}
accuracy : 0.9027777777777778
```

TASK 9

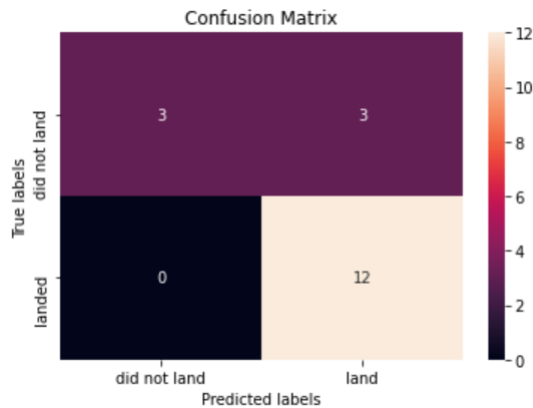
Calculate the accuracy of `tree_cv` on the test data using the method `score` :

```
tree_cv.score(X_test,Y_test)
```

```
0.8333333333333334
```

We can plot the confusion matrix

```
yhat = svm_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_train,Y_train)
```

```
GridSearchCV(cv=10, error_score='raise-deprecating',
            estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
            metric_params=None, n_jobs=None, n_neighbors=5, p=2,
            weights='uniform'),
            fit_params=None, iid='warn', n_jobs=None,
            param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'], 'p': [1, 2]},
            pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
            scoring=None, verbose=0)
```

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

```
tuned hyperparameters :(best parameters) {'algorithm': 'auto', 'n_neighbors': 9, 'p': 1}
accuracy : 0.8472222222222222
```

TASK 11

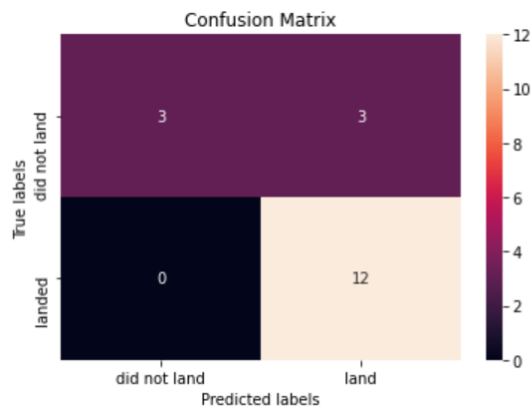
Calculate the accuracy of tree_cv on the test data using the method `score` :

```
knn_cv.score(X_test,Y_test)
```

```
0.8333333333333334
```

We can plot the confusion matrix

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



TASK 12

Find the method performs best:

```
print('Accuracy for Logistics Regression method:', logreg_cv.score(X_test, Y_test))
print('Accuracy for Support Vector Machine method:', svm_cv.score(X_test, Y_test))
print('Accuracy for Decision tree method:', tree_cv.score(X_test, Y_test))
print('Accuracy for K nearsdt neighbors method:', knn_cv.score(X_test, Y_test))
```

```
Accuracy for Logistics Regression method: 0.8333333333333334
Accuracy for Support Vector Machine method: 0.8333333333333334
Accuracy for Decision tree method: 0.8333333333333334
Accuracy for K nearsdt neighbors method: 0.8333333333333334
```