



# 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 [1]: import piplite
await piplite.install(['numpy'])
await piplite.install(['pandas'])
await piplite.install(['seaborn'])
```

```
In [ ]:
```

In [ ]:

We will import the following libraries for the lab

```
In [2]: # 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
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting framework. We will use t
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawi
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
```

```
<ipython-input-2-b7d446354769>:2: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better interoperability with other librar
ies)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466
```

```
import pandas as pd
```

In [ ]:

In [ ]:

This function is to plot the confusion matrix.

```
In [3]: 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()
```

```
In [ ]:
```

```
In [ ]:
```

## Load the dataframe

Load the data

```
In [4]: from js import fetch  
        import io  
  
        URL1 = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset-1/train.csv"  
        resp1 = await fetch(URL1)  
        text1 = io.BytesIO((await resp1.arrayBuffer()).to_py())  
        data = pd.read_csv(text1)
```

```
In [ ]:
```

```
In [5]: data.head()
```

Out [5]:

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

In [ ]:

In [ ]:

In [ ]:

```
In [6]: URL2 = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/data_v2.csv'
resp2 = await fetch(URL2)
text2 = io.BytesIO((await resp2.arrayBuffer()).to_py())
X = pd.read_csv(text2)
```

In [ ]:

In [ ]:

In [7]: `X.head(100)`

Out[7]:

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

90 rows × 83 columns

In [ ]:

In [ ]:

## 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 [8]: Y = data['Class'].to_numpy()
        Y.dtype
```

```
Out[8]: dtype('int64')
```

In [ ]:

In [ ]:

## TASK 2

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

```
In [9]: # students get this
transform = preprocessing.StandardScaler()
X = transform.fit_transform(X)
X
```

```
Out[9]: 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]])
```

In [ ]:

In [ ]:

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

```
In [ ]:
```

```
In [ ]:
```

we can see we only have 18 test samples.

```
In [11]: print ('Train set:')
print('X-train= ', X_train.shape, 'Y-train= ', Y_train.shape)
print ('Test set:')
print('X-test= ', X_test.shape, 'Y-test= ', Y_test.shape)
```

```
Train set:
X-train= (72, 83) Y-train= (72,)
Test set:
X-test= (18, 83) Y-test= (18,)
```

```
In [ ]:
```

```
In [ ]:
```

## 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`.

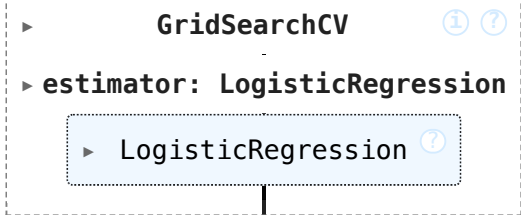
```
In [12]: parameters = {'C': [0.01, 0.1, 1],
                        'penalty': ['l2'],
                        'solver': ['lbfgs']}
```

```
In [ ]:
```



```
In [13]: parameters = {"C": [0.01, 0.1, 1], 'penalty': ['l2'], 'solver': ['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
logreg_cv = GridSearchCV(lr, parameters, cv=10)
logreg_cv.fit(X_train, Y_train)
```

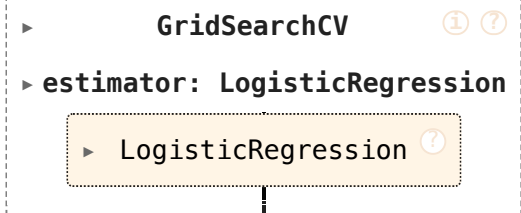
```
Out[13]:
```



```
In [ ]:
```

```
In [14]: GridSearchCV(cv=10, estimator=LogisticRegression(),
                  param_grid={"C": [0.01, 0.1, 1], 'penalty': ['l2'],
                              'solver': ['lbfgs']})
```

```
Out[14]:
```



```
In [ ]:
```

```
In [ ]:
```

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 [15]: 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
```

```
In [ ]:
```

In [ ]:

## TASK 5

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

```
In [16]: print("Logistic Regression test data accuracy :", logreg_cv.score(X_test, Y_test))
```

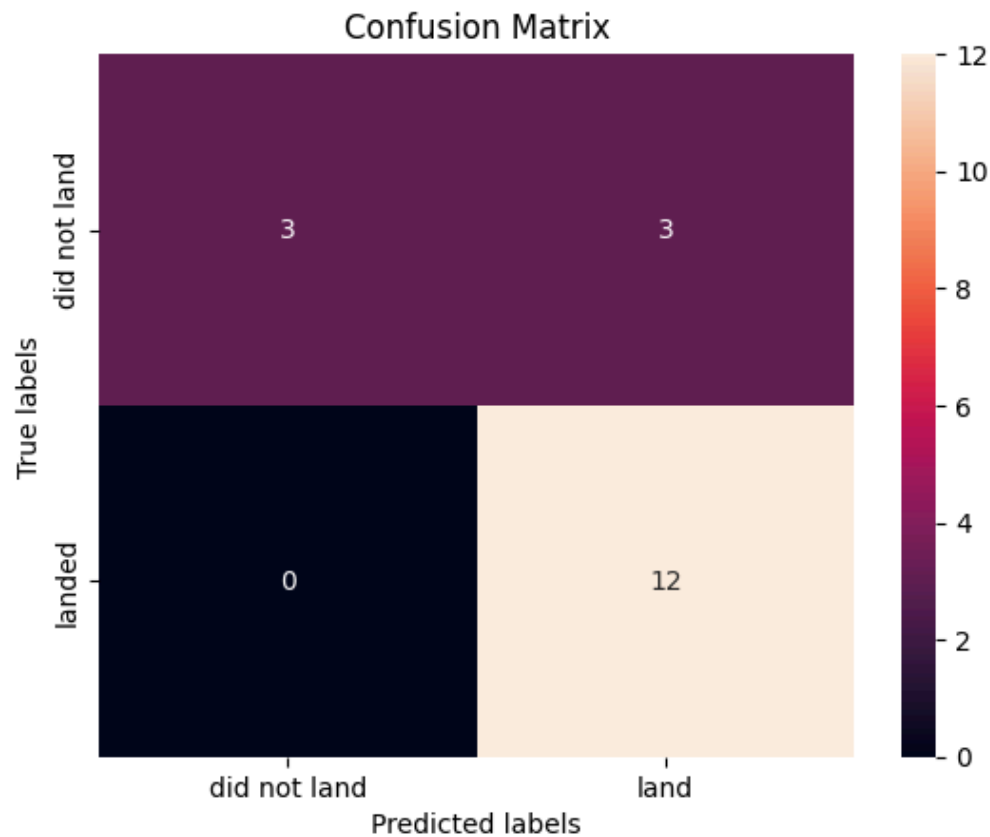
Logistic Regression test data accuracy : 0.8333333333333334

In [ ]:

In [ ]:

Lets look at the confusion matrix:

```
In [17]: yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
plt.show()
```



In [ ]:

In [ ]:

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

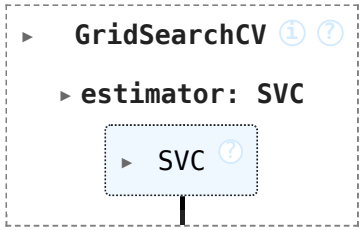
Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with `cv = 10`. Fit the object to find the best parameters from the dictionary `parameters` .

```
In [18]: parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
                        'C': np.logspace(-3, 3, 5),
                        'gamma':np.logspace(-3, 3, 5)}
# Create a support vector machine
svm = SVC()
```

In [ ]:

```
In [19]: svm_cv = GridSearchCV(svm, parameters, cv= 10)
svm_cv.fit(X_train, Y_train)
```

```
Out[19]:
```



In [ ]:

```
In [20]: 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 : 0.8482142857142856
```

In [ ]:

In [ ]:

## TASK 7

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

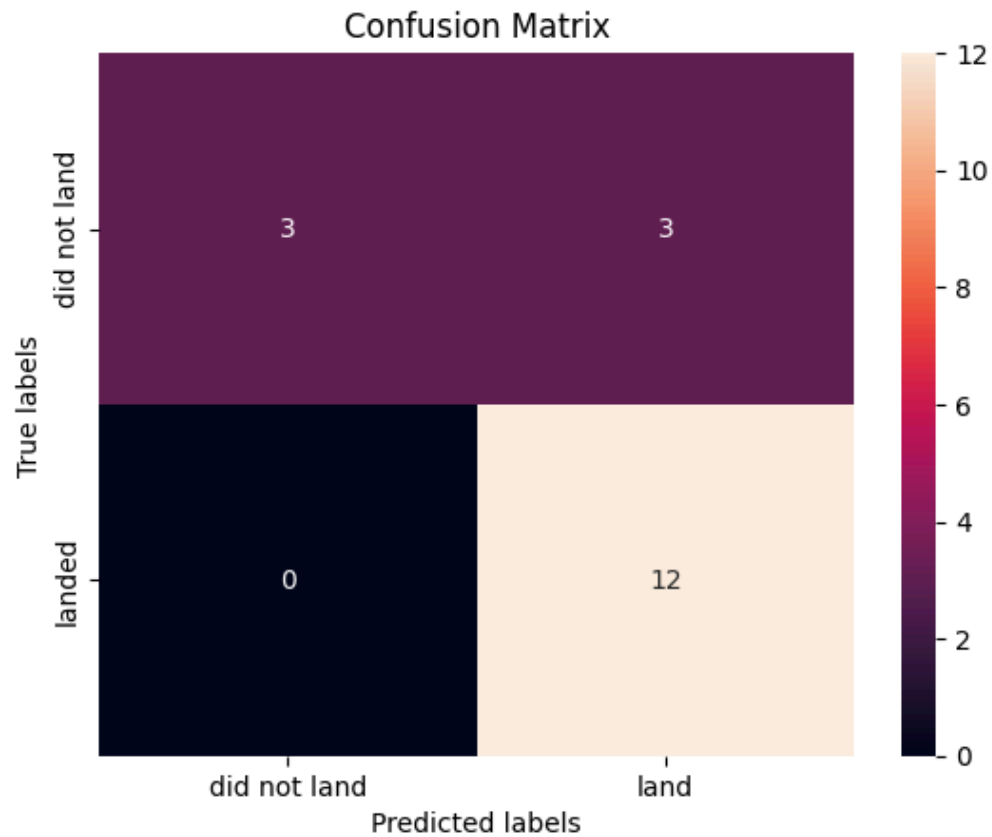
```
In [21]: print("SVM test data accuracy :",svm_cv.score(X_test, Y_test))
```

SVM test data accuracy : 0.8333333333333334

In [ ]:

We can plot the confusion matrix

```
In [22]: yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



In [ ]:

## 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 [28]: 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]}

# Create a decision tree classifier object
tree = DecisionTreeClassifier()
```

```
In [29]: 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
0.80357143 0.80535714 0.70357143 0.83392857 0.77678571 0.82321429
0.7375     0.79107143 0.7625     0.80535714 0.80357143 0.7625
0.77857143 0.77678571 0.7625     0.71964286 0.80714286 0.775
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
0.80535714 0.875     0.78928571 0.85      0.73392857 0.84642857
0.76785714 0.80178571 0.81964286 0.79107143 0.7625     0.84821429
0.66607143 0.83392857 0.84642857 0.78928571 0.72142857 0.79464286
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
0.75      0.79107143 0.81964286 0.78928571 0.80535714 0.81964286
0.74821429 0.73214286 0.77678571 0.80357143 0.80714286 0.81964286
0.73392857 0.75178571 0.78928571 0.77678571 0.80357143 0.79285714
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
```

nan	nan	nan	nan	nan	nan
0.79285714	0.83214286	0.77857143	0.82142857	0.76428571	0.7375
0.79285714	0.75	0.79107143	0.74821429	0.80714286	0.80535714
0.71964286	0.7625	0.80535714	0.77678571	0.79107143	0.84642857
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
0.81964286	0.7625	0.72142857	0.79107143	0.77857143	0.83392857
0.70535714	0.71964286	0.66071429	0.80535714	0.7625	0.79285714
0.72321429	0.78214286	0.80357143	0.76428571	0.83214286	0.81785714
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
0.81785714	0.70714286	0.78928571	0.69464286	0.80535714	0.73392857
0.77857143	0.80535714	0.7375	0.72321429	0.77678571	0.79107143
0.78035714	0.7625	0.74821429	0.80357143	0.7625	0.80357143
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
0.7625	0.775	0.73214286	0.77857143	0.70714286	0.87678571
0.775	0.74107143	0.78928571	0.69285714	0.77321429	0.83214286
0.83214286	0.84821429	0.80535714	0.725	0.69464286	0.78035714
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
0.7375	0.79107143	0.75	0.77857143	0.79285714	0.83392857
0.77678571	0.80357143	0.7625	0.73214286	0.77678571	0.79107143
0.71071429	0.80535714	0.80535714	0.83571429	0.81428571	0.78214286
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
0.75178571	0.775	0.78928571	0.79107143	0.79107143	0.80535714
0.74642857	0.7625	0.77857143	0.68928571	0.79464286	0.80535714
0.80178571	0.70892857	0.71964286	0.80357143	0.72321429	0.77857143
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
0.7625	0.69107143	0.7625	0.71964286	0.7625	0.80892857
0.78928571	0.68214286	0.77678571	0.80357143	0.74821429	0.70714286
0.67678571	0.70714286	0.75892857	0.73392857	0.71785714	0.79107143
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
nan	nan	nan	nan	nan	nan
0.80357143	0.79464286	0.79285714	0.80535714	0.77678571	0.875



```

0.83214286 0.80357143 0.86428571 0.79642857 0.84642857 0.72142857
0.81964286 0.79464286 0.81785714 0.82142857 0.775      0.7625
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
0.83392857 0.83214286 0.78035714 0.80357143 0.80357143 0.78928571
0.77678571 0.79642857 0.87321429 0.84642857 0.7375      0.81964286
0.84821429 0.78928571 0.81964286 0.81964286 0.75178571 0.85178571
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
0.775      0.79107143 0.775      0.81964286 0.83214286 0.81964286
0.77678571 0.7625      0.75178571 0.83214286 0.77678571 0.77678571
0.7375      0.775      0.79107143 0.80535714 0.80357143 0.85892857
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
0.83214286 0.80357143 0.80535714 0.80535714 0.775      0.77857143
0.775      0.74821429 0.80357143 0.74821429 0.775      0.83214286
0.79107143 0.82142857 0.75      0.76607143 0.81964286 0.80357143
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
0.73392857 0.80535714 0.77678571 0.7375      0.7625      0.77857143
0.77857143 0.76428571 0.69107143 0.74821429 0.79107143 0.80714286
0.75357143 0.76071429 0.80535714 0.74464286 0.69464286 0.79107143
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
0.85714286 0.77857143 0.76428571 0.86071429 0.81964286 0.79107143
0.83035714 0.79107143 0.79285714 0.83214286 0.7625      0.79464286
0.77678571 0.75      0.86071429 0.73392857 0.71964286 0.74821429
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
0.74821429 0.76964286 0.81607143 0.78035714 0.78928571 0.80535714
0.80535714 0.77678571 0.67857143 0.775      0.82857143 0.79107143
0.78035714 0.76071429 0.78214286 0.81785714 0.80357143 0.78928571
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
      nan      nan      nan      nan      nan      nan
0.73571429 0.77857143 0.71964286 0.74821429 0.80357143 0.71964286
0.77857143 0.76428571 0.74821429 0.72678571 0.79107143 0.83392857

```

```
0.7625    0.7875    0.79107143 0.86071429 0.81964286 0.7625    ]
warnings.warn(
```

Out[29]:

```
GridSearchCV
  estimator: DecisionTreeClassifier
    DecisionTreeClassifier
```

In [ ]:

```
In [30]: print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters) {'criterion': 'gini', 'max_depth': 14, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 10, 'splitter': 'random'}
accuracy : 0.8767857142857143
```

## TASK 9

Calculate the accuracy of tree\_cv on the test data using the method `score` :

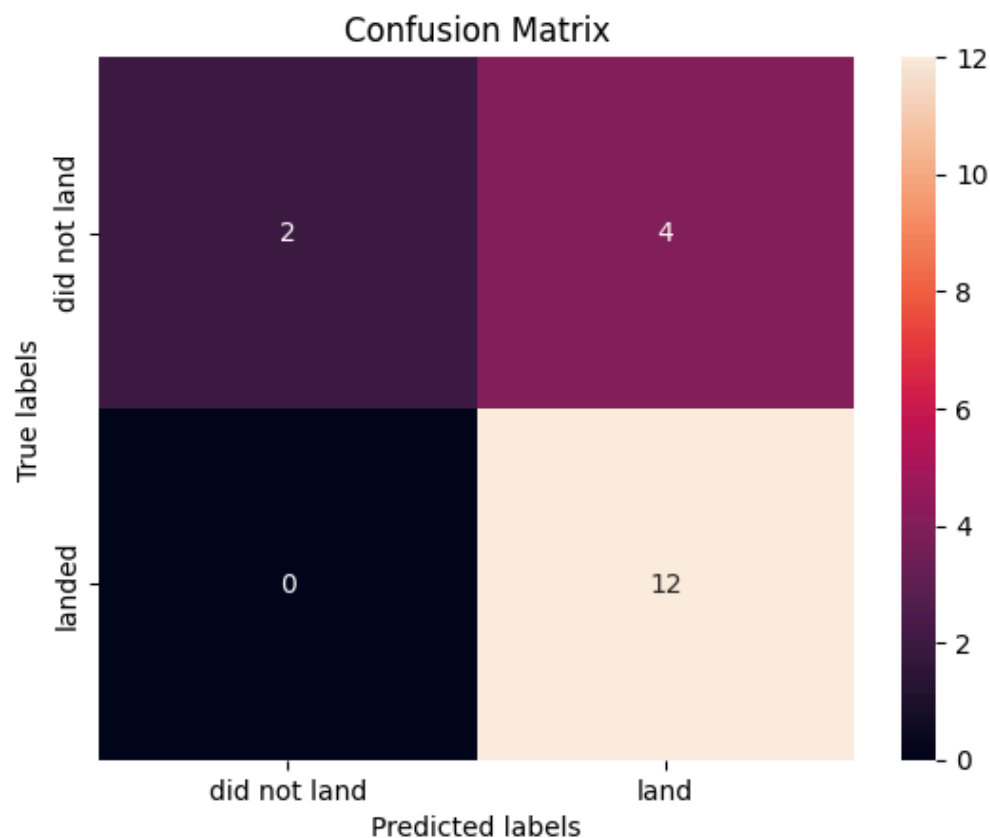
```
In [31]: print("Decision Tree accuracy on test set :",tree_cv.score(X_test, Y_test))
```

```
Decision Tree accuracy on test set : 0.7777777777777778
```

In [ ]:

We can plot the confusion matrix

```
In [32]: yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



In [ ]:

## TASK 10

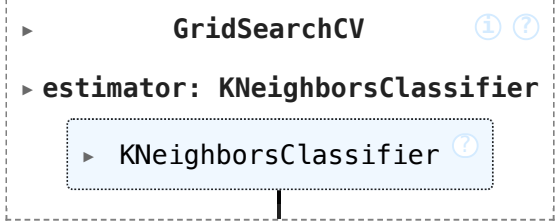
Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with `cv = 10`. Fit the object to find the best parameters from the dictionary `parameters` .

```
In [33]: 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 [34]: knn_cv = GridSearchCV(KNN, parameters, cv= 10)
knn_cv.fit(X_train, Y_train)
```

```
Out[34]:
```



```
In [ ]:
```

```
In [35]: print("tuned hyperparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

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

```
In [ ]:
```

## TASK 11

Calculate the accuracy of knn\_cv on the test data using the method `score` :

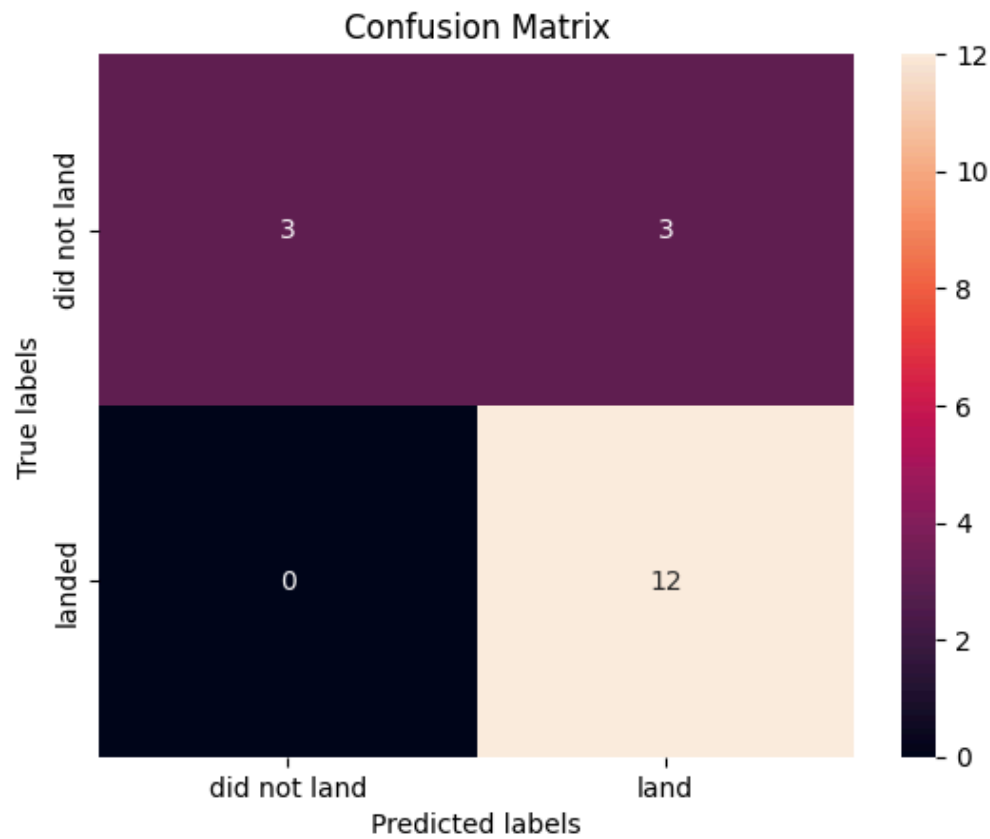
```
In [37]: knn_cv.score(X_test, Y_test)
```

```
Out[37]: 0.8333333333333334
```

```
In [ ]:
```

We can plot the confusion matrix

```
In [38]: yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



In [ ]:

## TASK 12

Find the method performs best:

```
In [41]: Report = pd.DataFrame({'Method' : ['Test Data Accuracy']})

knn_accuracy=knn_cv.score(X_test, Y_test)
Decision_tree_accuracy=tree_cv.score(X_test, Y_test)
SVM_accuracy=svm_cv.score(X_test, Y_test)
Logistic_Regression=logreg_cv.score(X_test, Y_test)

Report['Logistic_Reg'] = [Logistic_Regression]
```

```
Report['SVM'] = [SVM_accuracy]
Report['Decision Tree'] = [Decision_tree_accuracy]
Report['KNN'] = [knn_accuracy]
#Report
Report.transpose()
```

Out[41]:

Method	Test Data Accuracy
Logistic_Reg	0.833333
SVM	0.833333
Decision Tree	0.777778
KNN	0.833333

In [ ]:

In [ ]:

In [ ]:

## 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|--!>

IBM Corporation 2022. All rights reserved.