[(https://skills.network/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=1000SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2022-01-01)](https://skills.network/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=1000SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2022-01-01)

Sricharan Nallan Chakravarthula

Space X Falcon 9 First Stage Landing Prediction

Assignment: Machine Learning Prediction

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 (EDA)
- Create a column in the DataFrame for the class
- Normalize/Standardize the data
- Split the data into training and testing sets
- Find best Hyperparameter for SVM, Classification Trees and Logistic Regression using the testing data set

Supress Warnings

In [1]:
```
# Surpress warnings:
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
print("Done.")
```

```
Done.
```

Import Libraries

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
print("All libraries have been imported.")
```

```
All libraries have been imported.
```

Define Useful Functions

> **Note**
> This code bit was provided.

In [3]:
```
# 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_tickla
```

Start Here

Load the dataframe

Prepare DataFrame

> **Note**

In [4]:
```
# Load the dataset from (dataset_part_2.csv)
URL1 = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud,
data = pd.read_csv(URL1)
```

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

Out[5]:

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

In [6]:
```
data.shape
```

Out[6]:
```
(90, 18)
```

In [7]:
```python
list(data.columns)
```

Out[7]:
```
['FlightNumber',
 'Date',
 'BoosterVersion',
 'PayloadMass',
 'Orbit',
 'LaunchSite',
 'Outcome',
 'Flights',
 'GridFins',
 'Reused',
 'Legs',
 'LandingPad',
 'Block',
 'ReusedCount',
 'Serial',
 'Longitude',
 'Latitude',
 'Class']
```

In [8]:
```python
# Load the dataset from (dataset_part_3.csv)
URL2 = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud,
X = pd.read_csv(URL2)
```

In [9]:
```python
X.head(5)
```

Out[9]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_G
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	
4	5.0	3170.000000	1.0	1.0	0.0	0.0	0.0	

5 rows × 83 columns

In [10]:
```python
X.shape
```

Out[10]:
```
(90, 83)
```

In [11]:
```
list(X.columns)
```

```
Out[11]: ['FlightNumber',
 'PayloadMass',
 'Flights',
 'Block',
 'ReusedCount',
 'Orbit_ES-L1',
 'Orbit_GEO',
 'Orbit_GTO',
 'Orbit_HEO',
 'Orbit_ISS',
 'Orbit_LEO',
 'Orbit_MEO',
 'Orbit_PO',
 'Orbit_SO',
 'Orbit_SSO',
 'Orbit_VLEO',
 'LaunchSite_CCAFS SLC 40',
 'LaunchSite_KSC LC 39A',
 'LaunchSite_VAFB SLC 4E',
 'LandingPad_5e9e3032383ecb267a34e7c7',
 'LandingPad_5e9e3032383ecb554034e7c9',
 'LandingPad_5e9e3032383ecb6bb234e7ca',
 'LandingPad_5e9e3032383ecb761634e7cb',
 'LandingPad_5e9e3033383ecbb9e534e7cc',
 'Serial_B0003',
 'Serial_B0005',
 'Serial_B0007',
 'Serial_B1003',
 'Serial_B1004',
 'Serial_B1005',
 'Serial_B1006',
 'Serial_B1007',
 'Serial_B1008',
 'Serial_B1010',
 'Serial_B1011',
 'Serial_B1012',
 'Serial_B1013',
 'Serial_B1015',
 'Serial_B1016',
 'Serial_B1017',
 'Serial_B1018',
 'Serial_B1019',
 'Serial_B1020',
 'Serial_B1021',
 'Serial_B1022',
 'Serial_B1023',
 'Serial_B1025',
 'Serial_B1026',
 'Serial_B1028',
 'Serial_B1029',
 'Serial_B1030',
 'Serial_B1031',
 'Serial_B1032',
 'Serial_B1034',
 'Serial_B1035',
 'Serial_B1036',
 'Serial_B1037',
```

```
 'Serial_B1038',
 'Serial_B1039',
 'Serial_B1040',
 'Serial_B1041',
 'Serial_B1042',
 'Serial_B1043',
 'Serial_B1044',
 'Serial_B1045',
 'Serial_B1046',
 'Serial_B1047',
 'Serial_B1048',
 'Serial_B1049',
 'Serial_B1050',
 'Serial_B1051',
 'Serial_B1054',
 'Serial_B1056',
 'Serial_B1058',
 'Serial_B1059',
 'Serial_B1060',
 'Serial_B1062',
 'GridFins_False',
 'GridFins_True',
 'Reused_False',
 'Reused_True',
 'Legs_False',
 'Legs_True']
```

Task
1

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 [12]:
```
# Dependent Variable
Y = data['Class'].to_numpy()
type(Y)
```

Out[12]:
```
numpy.ndarray
```

In [13]:
```
Y.shape
```

Out[13]:
```
(90,)
```

In [14]:
```
Y
```

Out[14]:
```
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])
```

Task 2

TASK 2

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

In [15]:
```
# Normalize dataset
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:1]
```

Out[15]:
```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01,
        -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
        -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
        -5.51677284e-01,  3.44342023e+00, -1.85695338e-01,
        -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
        -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
        -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
        -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
         9.43398113e+00, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
        -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
        -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
        -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
        -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
        -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
        -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
         1.93309133e+00, -1.93309133e+00]])
```

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

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 [16]:
```
# Split the data into Training and Test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, r
print(f'X_train: {X_train.shape}  Y_train: {Y_train.shape}')
print(f'X_test: {X_test.shape}  Y_test: {Y_test.shape}')
```

```
X_train: (72, 83)  Y_train: (72,)
X_test: (18, 83)  Y_test: (18,)
```

we can see we only have 18 test samples.

[0] Logistic Regression

Task
4

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` .

GridSearchCV()... Grid Search Cross-Validation ...is part of the Scikit-learn library for Python

In [17]:
```python
# Set parameters, create model, perform GridSearchCV to identify the best p
parameters_lr = {"C":[0.01,0.1,1],
                 'penalty':['l2'],
                 'solver':['lbfgs']}  # l1 lasso l2 ridge

lr = LogisticRegression()
logreg_cv = GridSearchCV(lr, parameters_lr, cv=5)
logreg_cv.fit(X_train, Y_train)
print("Tuned Hyperparameters (Best Parameters): ", logreg_cv.best_params_)
print("Accuracy: ", logreg_cv.best_score_)
```

```
Tuned Hyperparameters (Best Parameters):  {'C': 0.1, 'penalty': 'l2', 'so
lver': 'lbfgs'}
Accuracy:  0.8333333333333334
```

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_` .

Task
5

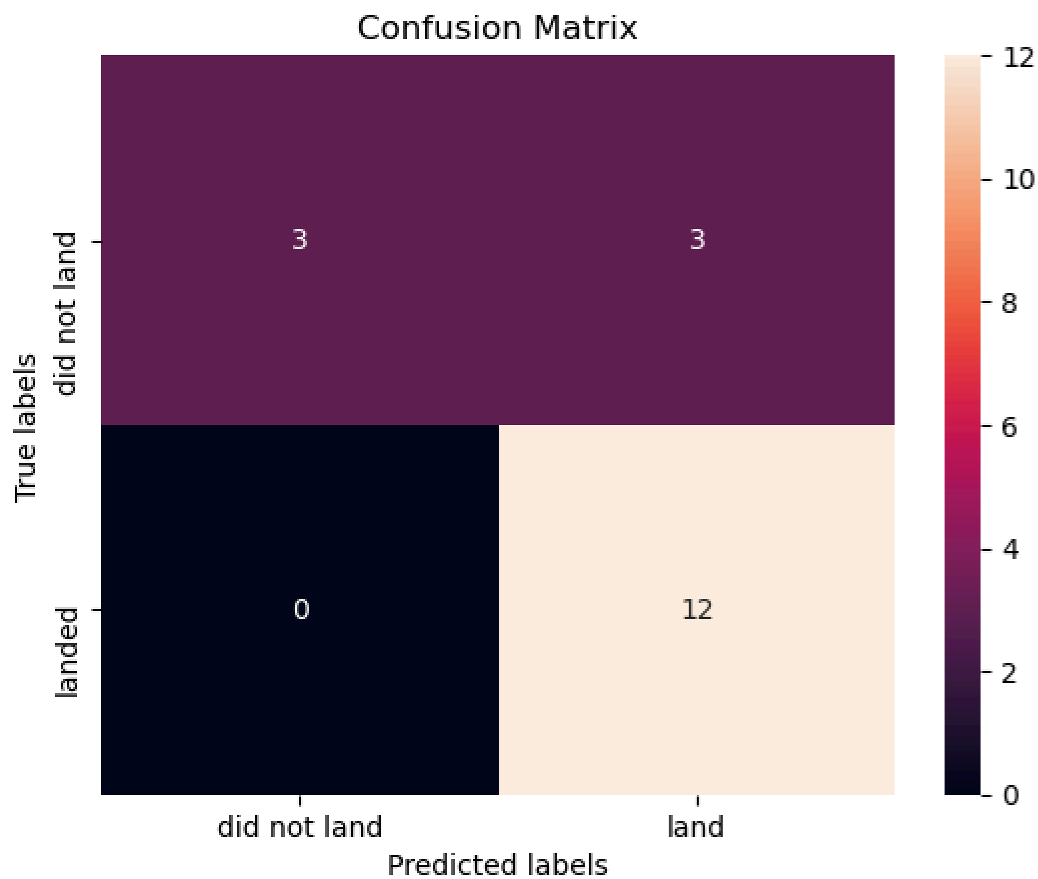
TASK 5

Calculate the accuracy on the test data using the method `score` and plot the confusion matrix.

In [18]:
```python
# Calculate Score
logistic_regression_score = logreg_cv.score(X_test, Y_test)
logistic_regression_score
```

Out[18]:
```
0.8333333333333334
```

In [19]:
```
# Plot Confusion Matrix
yhat_lr = logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test, yhat_lr)
plt.show()
```



Examining the confusion matrix, we see that logistic regression can distinguish between the different classes. We see that the major problem is false positives.

[1] SVM (Support Vector Machine)

Task 6

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 [20]:
```
# Set parameters, create model, perform GridSearchCV to identify the best 
parameters_svm = {'kernel':('linear', 'rbf', 'poly', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}

svm = SVC()
svm_cv = GridSearchCV(svm, parameters_svm, cv=5)
svm_cv.fit(X_train, Y_train)
print("Tuned Hyperparameters (Best Parameters): ", svm_cv.best_params_)
print("Accuracy: ", svm_cv.best_score_)
```

```
Tuned Hyperparameters (Best Parameters):  {'C': 0.03162277660168379, 'gam
ma': 0.001, 'kernel': 'linear'}
Accuracy:  0.8333333333333334
```

Task
7

TASK 7

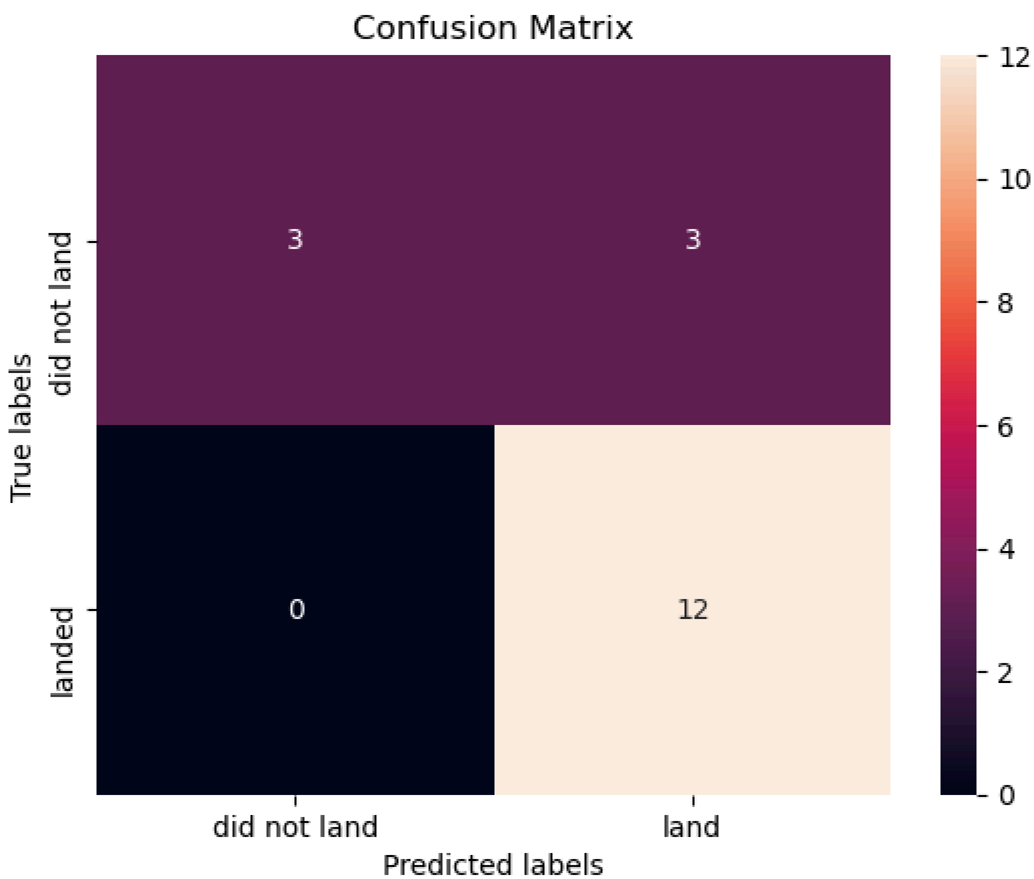
Calculate the accuracy on the test data using the method `score` and plot the confusion matrix.

In [21]:
```
# Calculate Score
svm_score = svm_cv.score(X_test, Y_test)
svm_score
```

Out[21]: 0.8333333333333334

In [22]:

```python
# Plot Confusion Matrix
yhat_svm = svm_cv.predict(X_test)
plot_confusion_matrix(Y_test, yhat_svm)
plt.show()
```



[2] Decision Tree

Task 8

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 [23]:
```
# Set parameters, create model, perform GridSearchCV to identify the best p
parameters_tree = {'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_tree, cv=5)
tree_cv.fit(X_train, Y_train)
print("Tuned Hyperparameters (Best Parameters): ", tree_cv.best_params_)
print("Accuracy: ", tree_cv.best_score_)
```

```
Tuned Hyperparameters (Best Parameters):  {'criterion': 'gini', 'max_dept
h': 4, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_spli
t': 10, 'splitter': 'best'}
Accuracy:  0.875
```

Task
9

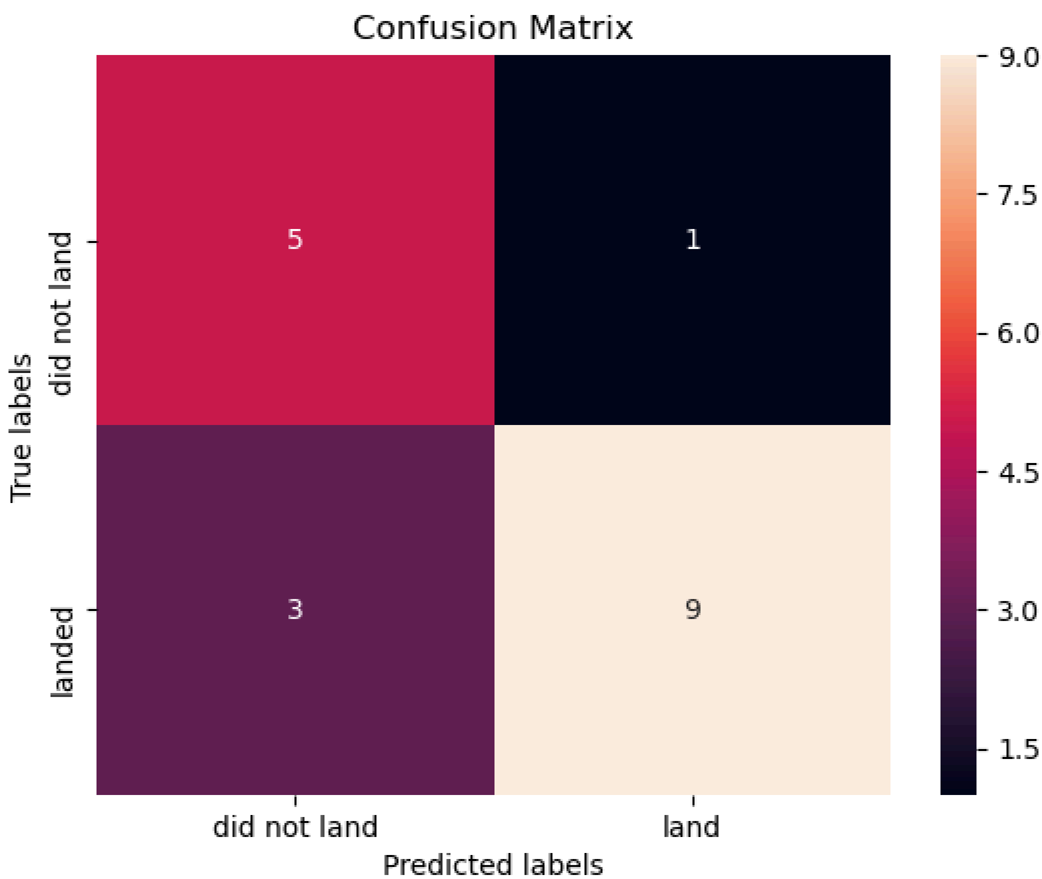
TASK 9

Calculate the accuracy on the test data using the method `score` and plot the confusion matrix.

In [24]:
```
# Calculate Score
decision_tree_score = tree_cv.score(X_test, Y_test)
decision_tree_score
```

Out[24]:
```
0.7777777777777778
```

In [25]:
```
# Plot Confusion Matrix
yhat_tree = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test, yhat_tree)
plt.show()
```



KNN (k-Nearest Neighbors)

Task 10

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 [26]:
```
# Set parameters, create model, perform GridSearchCV to identify the best p
parameters_knn = {'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_knn, cv=5)
knn_cv.fit(X_train, Y_train)
print("Tuned Hyperparameters (Best Parameters): ", knn_cv.best_params_)
print("Accuracy: ", knn_cv.best_score_)
```

```
Tuned Hyperparameters (Best Parameters):  {'algorithm': 'auto', 'n_neighb
ors': 7, 'p': 1}
Accuracy:  0.8472222222222222
```

Task
11

TASK 11

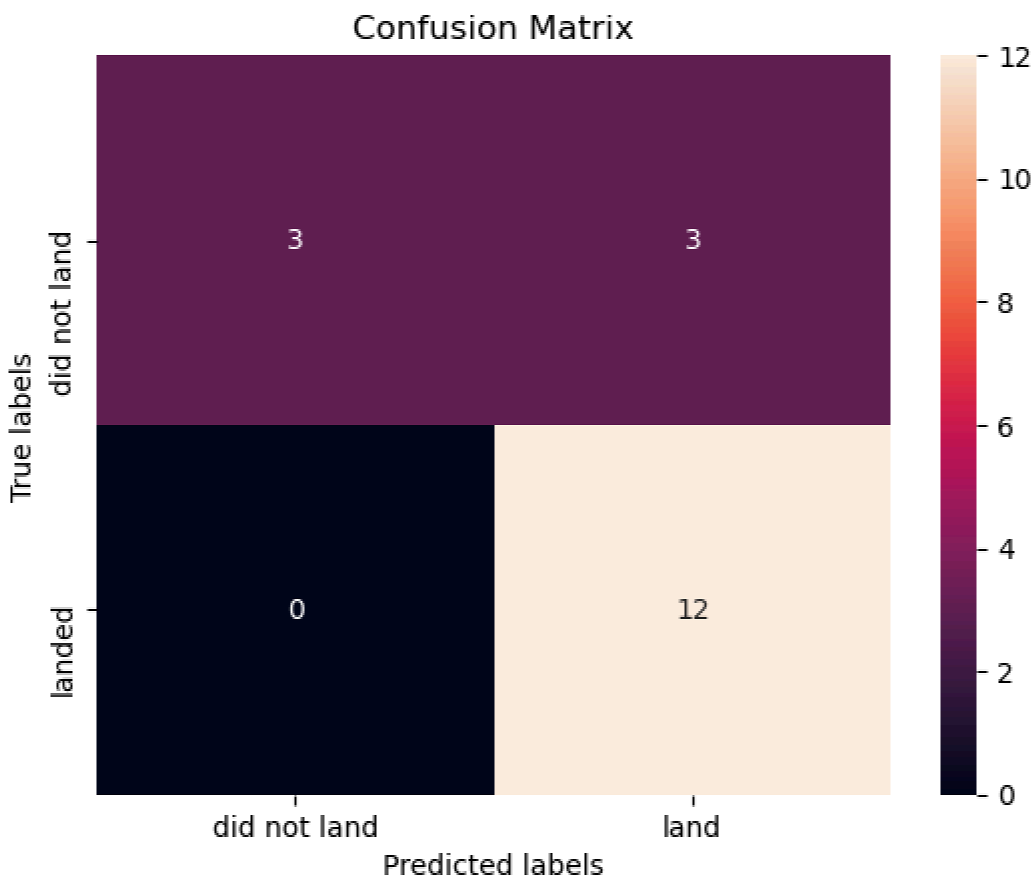
Calculate the accuracy on the test data using the method `score` and plot the confusion matrix.

In [27]:
```
# Calculate Score
knn_score = knn_cv.score(X_test, Y_test)
knn_score
```

Out[27]:
```
0.8333333333333334
```

In [28]:

```
# Plot Confusion Matrix
yhat_knn = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test, yhat_knn)
plt.show()
```



Compare Scores

> **Note**
> The objective here is to identify the method that performed the best.

Task
12

TASK 12

In [29]:

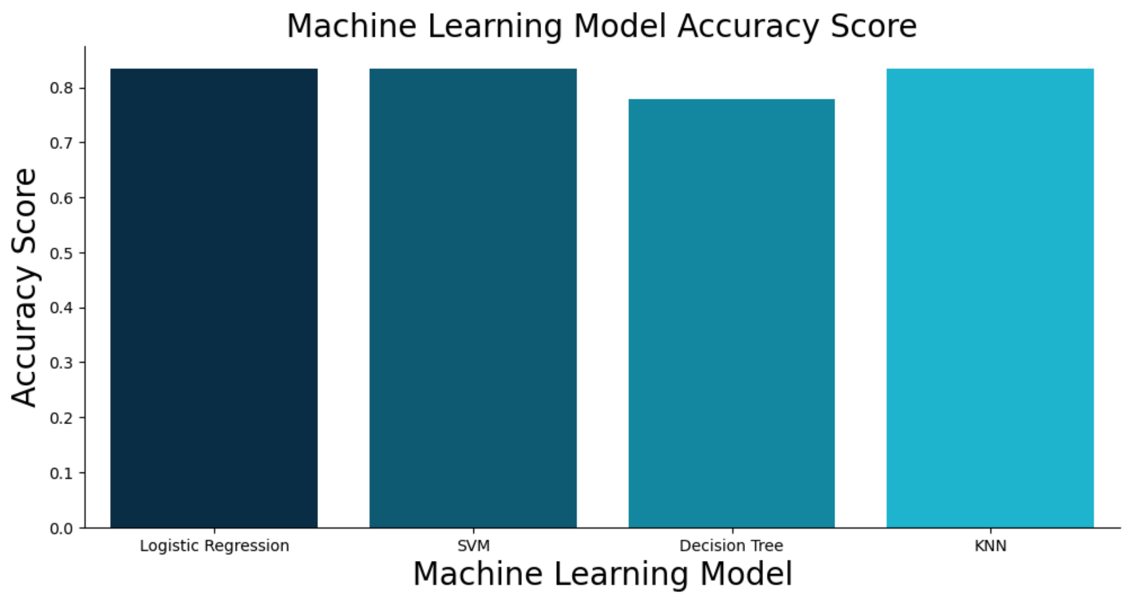
```
# Compile model scores into a DataFrame and then identify and label the ma
ranking_dict = {'Model':['Logistic Regression', 'SVM', 'Decision Tree', 'K
ranking_df = pd.DataFrame(ranking_dict)

def assign_status(score):
    if score == ranking_df['Score'].max():
        return 'Max Score'
    else:
        return '0'

ranking_df['Max Score'] = ranking_df['Score'].apply(assign_status)
ranking_df
```

Out[29]:

	Model	Score	Max Score
0	Logistic Regression	0.833333	Max Score
1	SVM	0.833333	Max Score
2	Decision Tree	0.777778	0
3	KNN	0.833333	Max Score

In [30]:

```python
# Plot the model accuracy on a bar chart
colors_dict = {'Logistic Regression':'#003355', 'SVM':'#006688', 'Decision
sns.catplot(kind="bar", data=ranking_df, x="Model", y="Score", aspect=2, p
plt.xlabel("Machine Learning Model", fontsize=20)
plt.ylabel("Accuracy Score", fontsize=20)
plt.title("Machine Learning Model Accuracy Score", fontsize=20)
plt.show()
```



Conclusion

> **Note**
> All models performed equally well except for the Decision Tree model which performed poorly relative to the other models

End Here