دوره دیتا ساینس کاربردی

Space X

Project

dataroadmap

مدرس: مونا حاتمی

جلسه دهم

Space X Project

Import Libraries

In [1]:

```
# Pandas is a software library written for the Python programming language for data manipulation and a
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large, multi-dimensional
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting framework. We
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interfac
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
```

Function in Python

```
def add(a):
    """ for add"""
    b=a+3
    c=a+b
    print(c)
```

```
add(2)
```

```
7
```

```
def my_func():
    x = 10
    print("Value inside function:",x)
```

```
my_func()
```

```
Value inside function: 10
```

Function for Plot

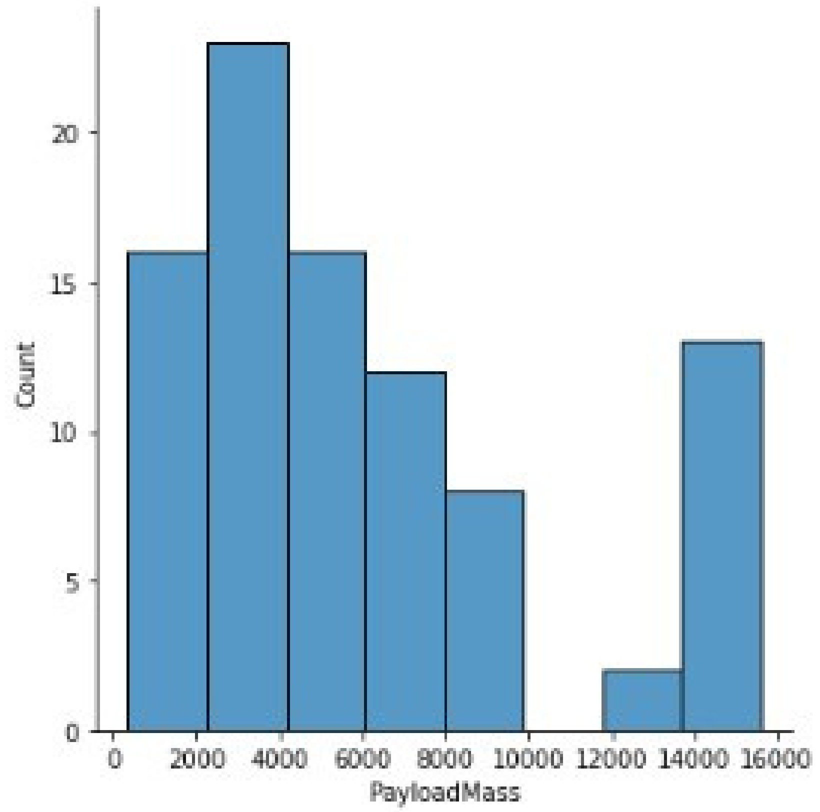
```
def plot_confusion_matrix(y,y_predict):
    "this function plots the confusion matrix"
    from sklearn.metrics import confusion_matrix

    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'])
```

Read Data

```
data = pd.read_csv('dataset_falcon9.csv')

data.head(100)
```

4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	
85	86	2020-09-03	Falcon 9	15400.000000	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3032383ecb6bb2
86	87	2020-10-06	Falcon 9	15400.000000	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3032383ecb6bb2
87	88	2020-10-18	Falcon 9	15400.000000	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3032383ecb6bb2
88	89	2020-10-24	Falcon 9	15400.000000	VLEO	CCAFS SLC 40	True ASDS	3	True	True	True	5e9e3033383ecbb9e5
89	90	2020-11-05	Falcon 9	3681.000000	MEO	CCAFS SLC 40	True ASDS	1	True	False	True	5e9e3032383ecb6bb2

90 rows × 18 columns



Preprocessing

```
Preprocessed = pd.read_csv('preprocessed_dataset.csv')
Preprocessed.head(100)
```

]:

	Unnamed: 0	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Class	Orbit_ES-L1	...	Serial_B1048	Serial_B1049	Serial_
0	0	6104.959412	1	0	0	0	1.0	0	0	0	...	0	0	
1	1	525.000000	1	0	0	0	1.0	0	0	0	...	0	0	
2	2	677.000000	1	0	0	0	1.0	0	0	0	...	0	0	
3	3	500.000000	1	0	0	0	1.0	0	0	0	...	0	0	
4	4	3170.000000	1	0	0	0	1.0	0	0	0	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
85	85	15400.000000	2	1	1	1	5.0	2	1	0	...	0	0	
86	86	15400.000000	3	1	1	1	5.0	2	1	0	...	0	0	
87	87	15400.000000	6	1	1	1	5.0	5	1	0	...	0	0	
88	88	15400.000000	3	1	1	1	5.0	2	1	0	...	0	0	
89	89	3681.000000	1	1	0	1	5.0	0	1	0	...	0	0	

90 rows × 89 columns

Preprocessing- Standardize

```
x['PayloadMass'].mean()
```

```
]: 6104.959411764707
```

```
x['PayloadMass'].std()
```

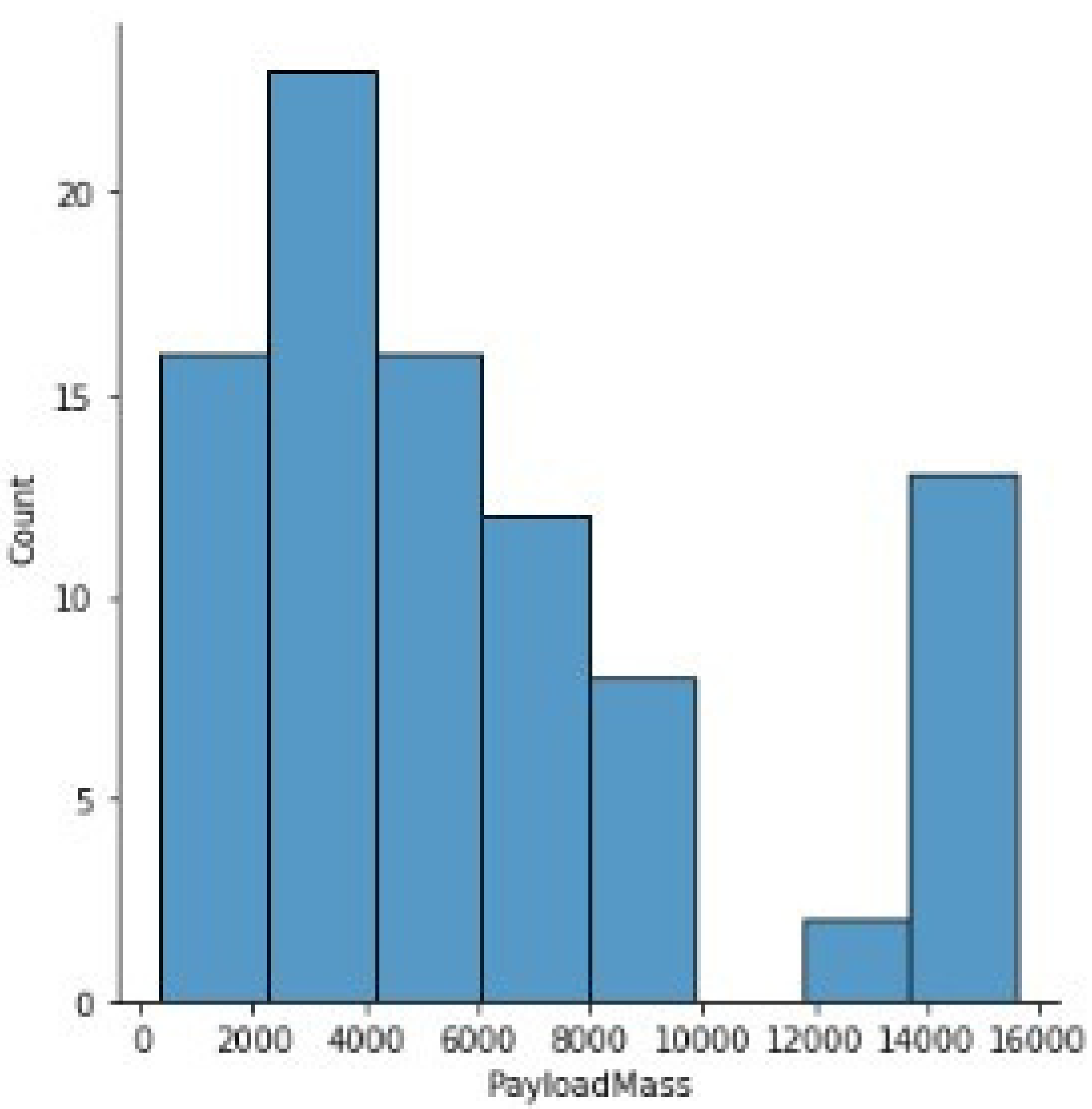
```
]: 4694.671719712728
```

```
x['Flights'].mean()
```

```
]: 1.788888888888889
```

```
x['Flights'].std()
```

```
]: 1.2131715741866367
```



Standardize Formula

Standardised Value

Original Value

$$x' = \frac{x - \mu}{\sigma}$$

Sample Mean

Sample Standard Deviation

Standardize in Scikit learn

```
# Preprocessing allows us to standarsize our data
from sklearn import preprocessing
```

```
transform = preprocessing.StandardScaler()
x_scaled = transform.fit_transform(X)
x_scaled
```

```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01, ...,
        -2.15665546e-01, -1.85695338e-01, -1.05999788e-01],
       [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01, ...,
        -2.15665546e-01, -1.85695338e-01, -1.05999788e-01],
       [-1.63592675e+00, -1.16267307e+00, -6.53912840e-01, ...,
        -2.15665546e-01, -1.85695338e-01, -1.05999788e-01],
       ...,
       [ 1.63592675e+00,  1.99100483e+00,  3.49060516e+00, ...,
        -2.15665546e-01, -1.85695338e-01, -1.05999788e-01],
```

Array to Dataframe

[24]:

```
col=X.columns
X = pd.DataFrame(x_scaled, columns=col)
X
```

Out[24]:

	Unnamed: 0	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Ort
0	-1.712912	-1.948145e-16	-0.653913	-1.870829	-0.835532	-1.933091	-1.575895	-0.973440	-0.106	
1	-1.674419	-1.195232e+00	-0.653913	-1.870829	-0.835532	-1.933091	-1.575895	-0.973440	-0.106	
2	-1.635927	-1.162673e+00	-0.653913	-1.870829	-0.835532	-1.933091	-1.575895	-0.973440	-0.106	
3	-1.597434	-1.200587e+00	-0.653913	-1.870829	-0.835532	-1.933091	-1.575895	-0.973440	-0.106	
4	-1.558942	-6.286706e-01	-0.653913	-1.870829	-0.835532	-1.933091	-1.575895	-0.973440	-0.106	
...	...	...	...	...	...	...	...	...	...	
85	1.558942	1.991005e+00	0.174991	0.534522	1.196843	0.517306	0.945537	0.202528	-0.106	
86	1.597434	1.991005e+00	1.003894	0.534522	1.196843	0.517306	0.945537	0.202528	-0.106	
87	1.635927	1.991005e+00	3.490605	0.534522	1.196843	0.517306	0.945537	1.966480	-0.106	

```
X['PayloadMass'].mean()
```

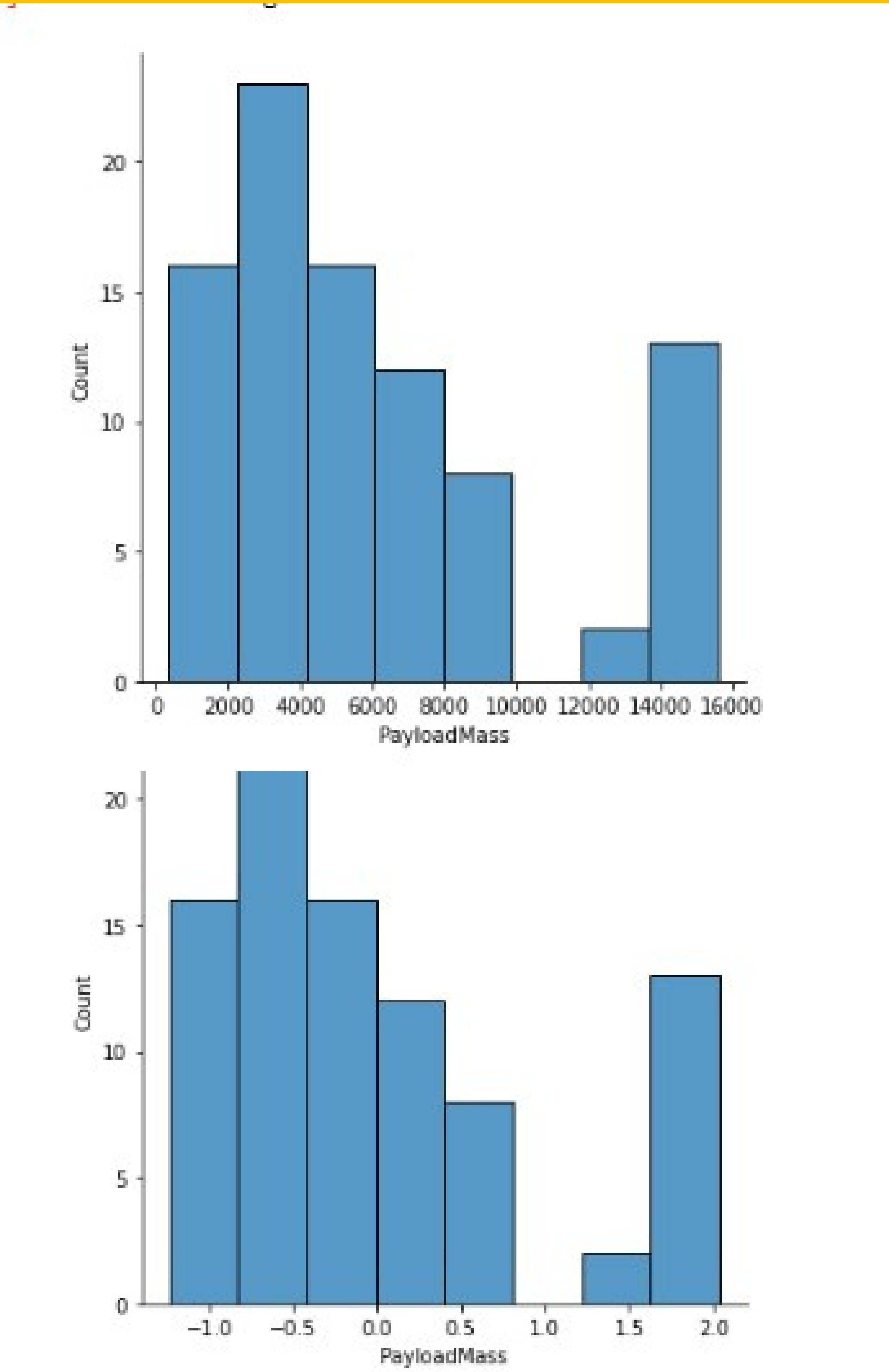
5]: -5.304398895431304e-17

```
X['PayloadMass'].std()
```

5]: 1.0056022847309865

```
sns.displot(data=X, x="PayloadMass")
```

7]: <seaborn.axisgrid.FacetGrid at 0x1dc10e4ccd0>



Train - Test Split

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, random_state=101)
```

Logistic Regression

```
lr=LogisticRegression()
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}
logreg_cv = GridSearchCV(lr, parameters,cv=4)
logreg_cv.fit(X_train, Y_train)
```

```
]: GridSearchCV(cv=4, estimator=LogisticRegression(),
                param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                            'solver': ['lbfgs']})
```

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

```
tuned hpyerparameters :(best parameters)  {'C': 1, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.875
```

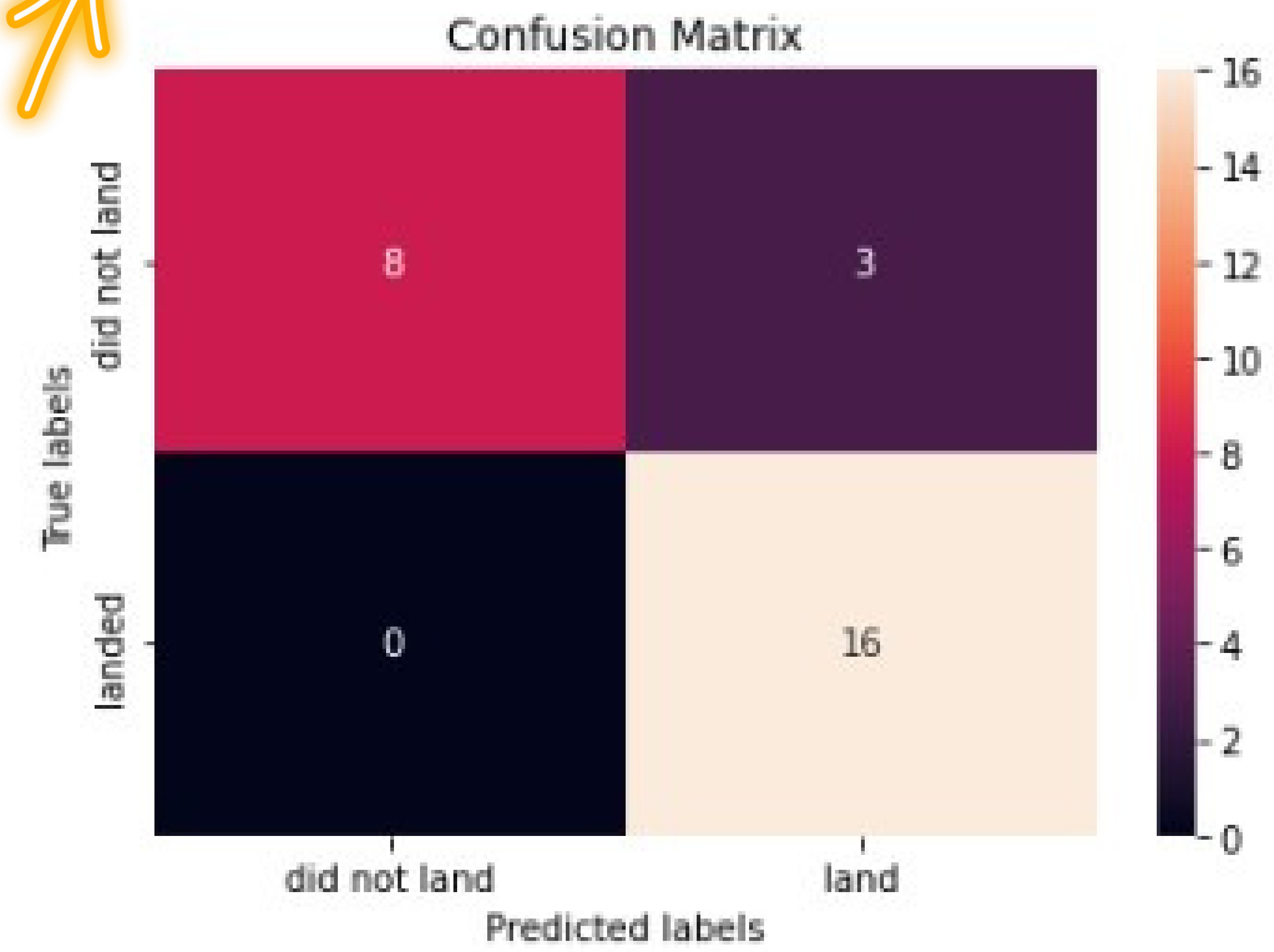
Create List for collecting results

```
accu=[]
methods=[]
accu.append(logreg_cv.score(X_test,Y_test))
methods.append('logistic regression')
logreg_cv.score(X_test,Y_test)
```

```
0.8888888888888888
```

Confusion Matrix

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




```
def plot_confusion_matrix(y,y_predict):
    "this function plots the confusion mat
    from sklearn.metrics import confusion_

    cm = confusion_matrix(y, y_predict)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax);
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['did not land
```

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Support Vector Machine

In [27]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': (0.5, 1, 1.5)}
svm = SVC()
```

In [28]:
```
svm_cv = GridSearchCV(svm, parameters, cv = 10)
svm_cv.fit(X_train, Y_train)
```

Out[28]:
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': (0.5, 1, 1.5),
                         'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid')})
```

In [29]:
```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

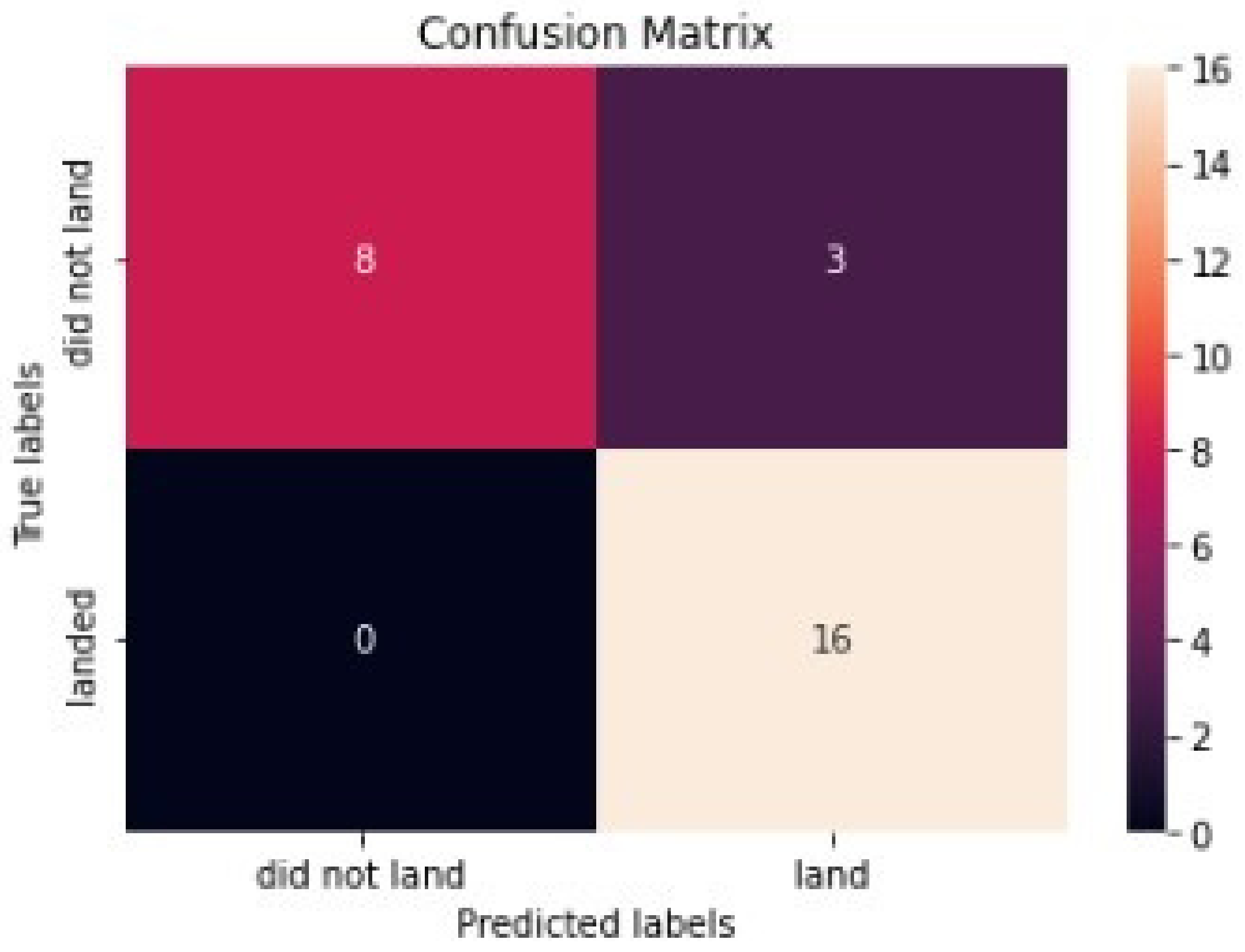
```
tuned hpyerparameters :(best parameters)  {'C': 1, 'kernel': 'sigmoid'}
accuracy : 0.9380952380952381
```

Confusion Matrix

In [30]:
```
accu.append(svm_cv.score(X_test,Y_test))
methods.append('support vector machine')
svm_cv.score(X_test,Y_test)
```
Out[30]: 0.8888888888888888

Confusion Matrix

In [31]:
```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Decission Trees

In [32]:
```
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()
```

In [33]:
```
tree_cv = GridSearchCV(tree, parameters, cv = 10)
tree_cv.fit(X_train, Y_train)
```

Out[33]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         '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],
                         'splitter': ['best', 'random']})
```

In [34]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 16, 'max_f
af': 2, 'min_samples_split': 2, 'splitter': 'best'}
accuracy : 0.9547619047619047
```

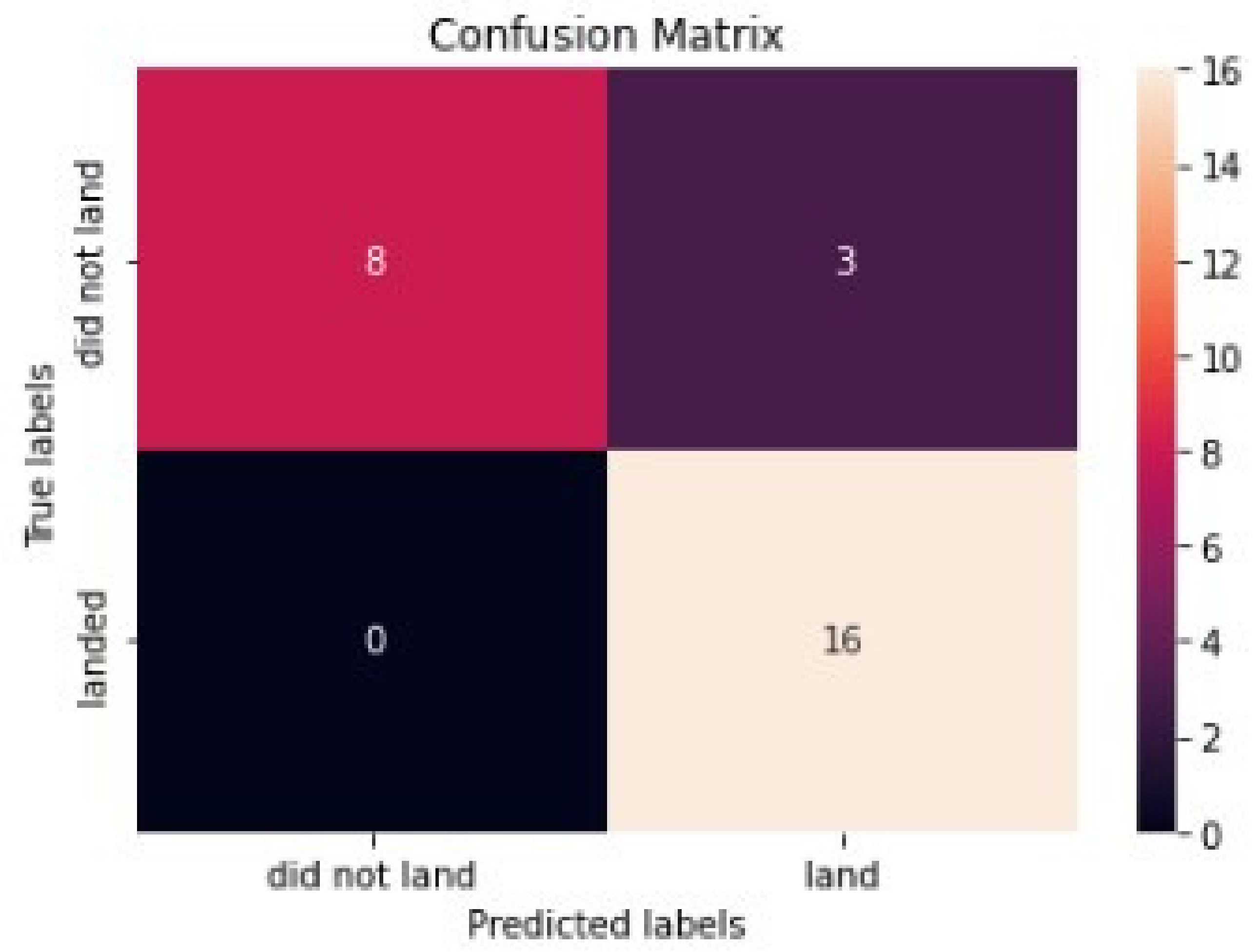
Confusion Matrix

In [35]:
```
accu.append(tree_cv.score(X_test,Y_test))
methods.append('decision tree classifier')
tree_cv.score(X_test,Y_test)
```

Out[35]: 0.6666666666666666

Confusion Matrix

In [36]:
```
yhat = svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



K Nearest Neighbor

In [37]:
```
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 [38]:
```
knn_cv = GridSearchCV(KNN, parameters, cv = 10)
knn_cv.fit(X_train, Y_train)
```

Out[38]:
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]})
```

In [39]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

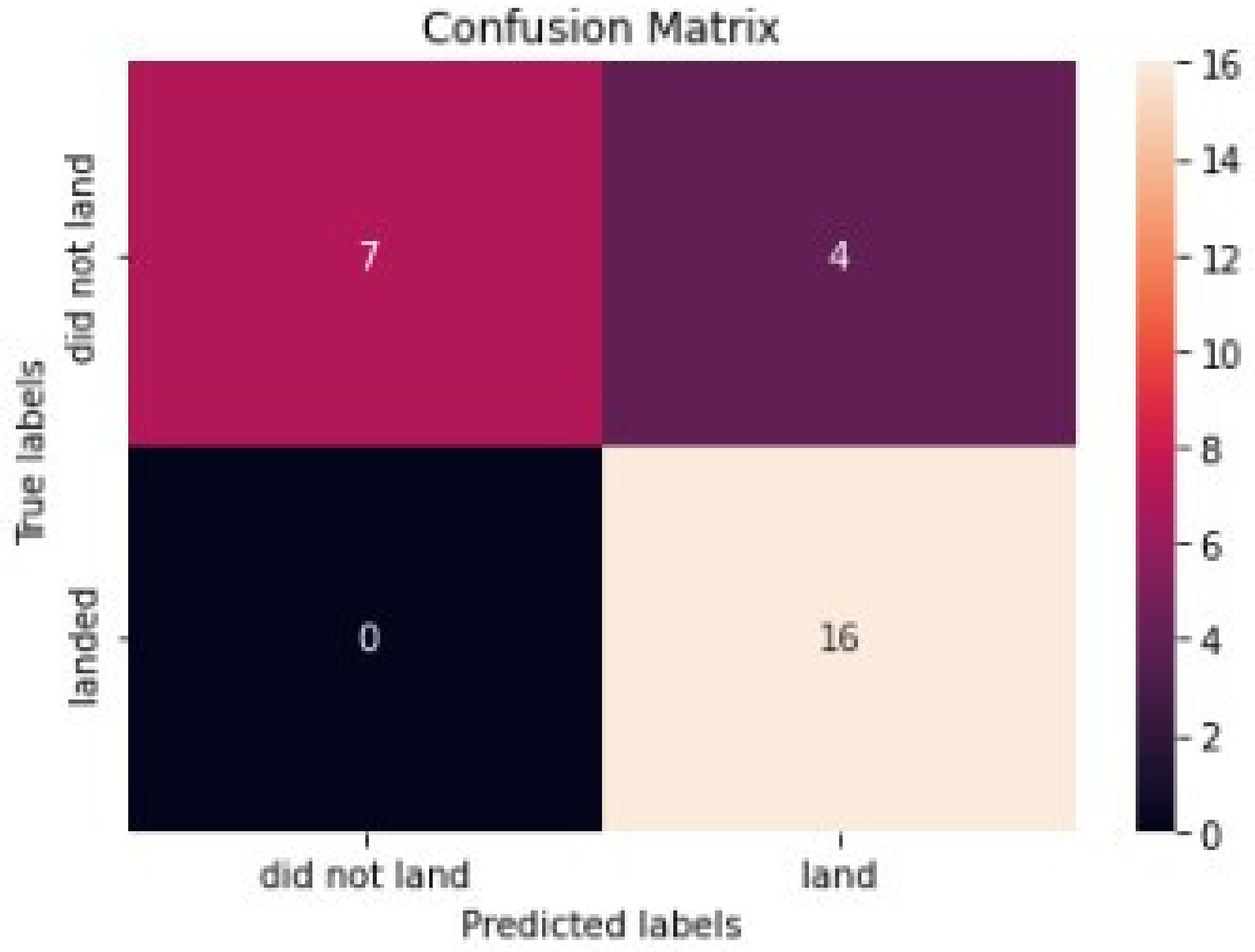
```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 4, 'p': 1}
accuracy : 0.8928571428571429
```

Confusion Matrix

In [40]:
```
accu.append(knn_cv.score(X_test,Y_test))
methods.append('k nearest neighbors')
knn_cv.score(X_test,Y_test)
```
Out[40]: 0.8518518518518519

Confusion Matrix

In [42]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```

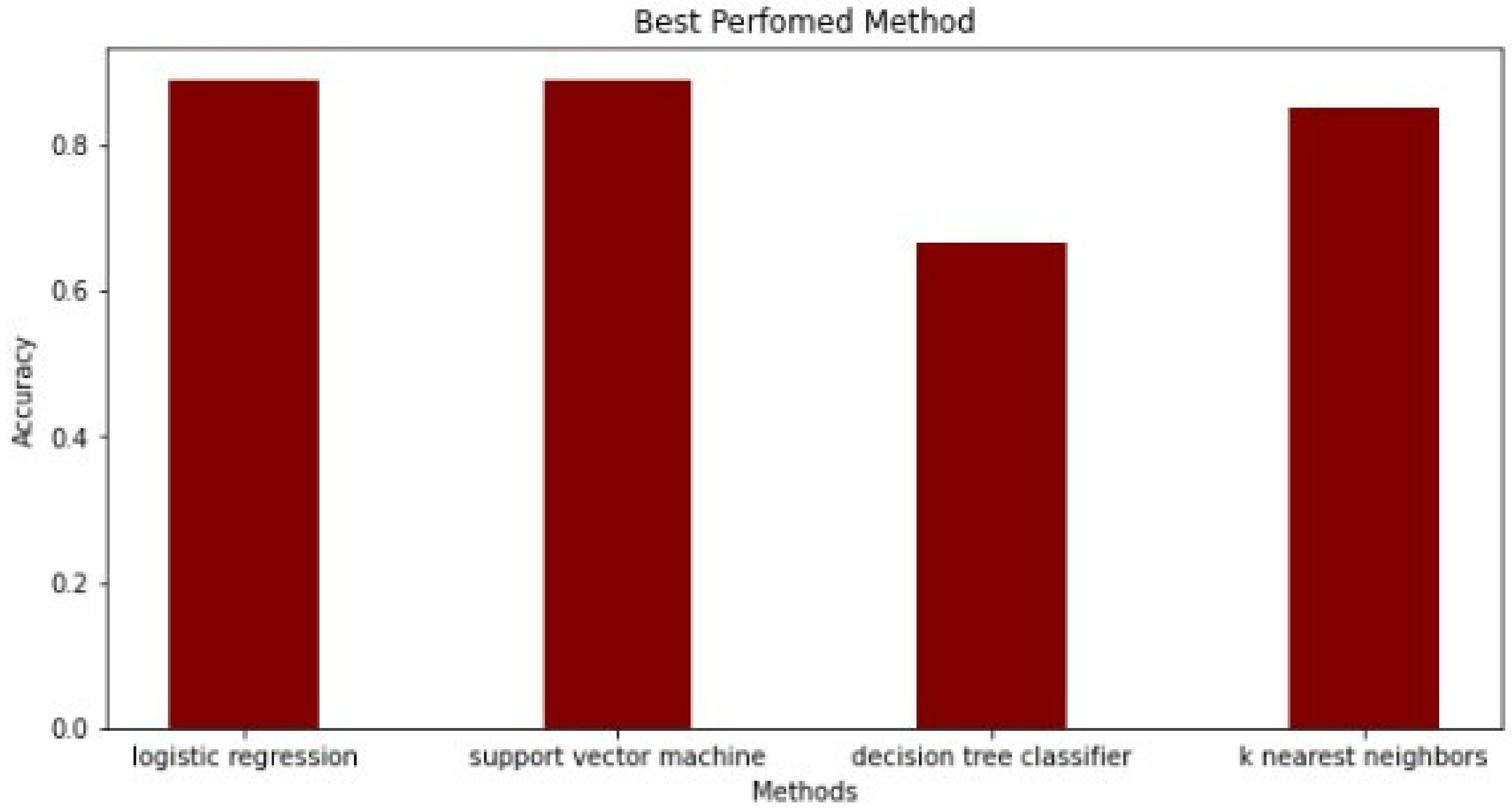


The Best Performed Model

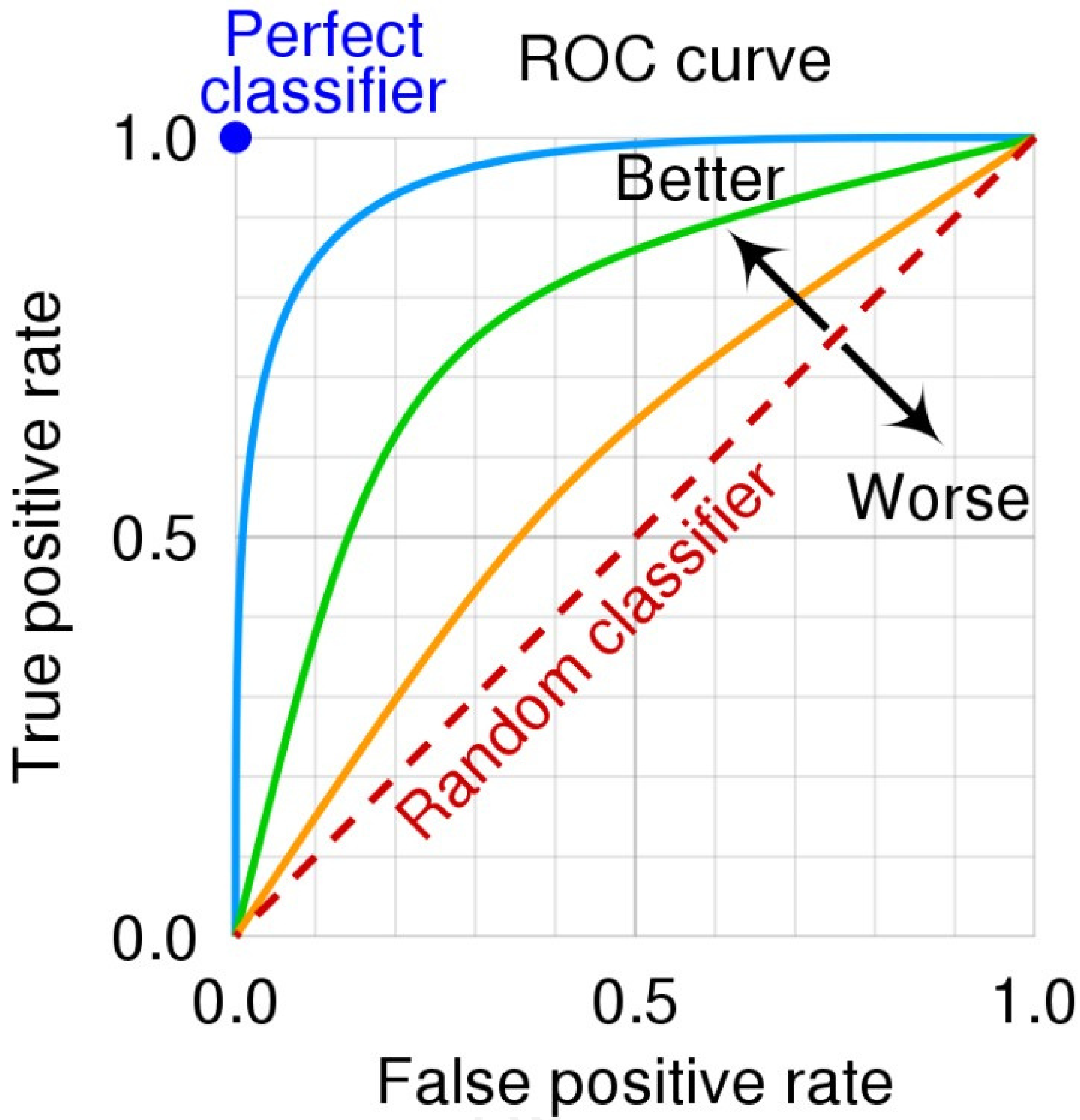
```python
print(methods)
print(accu)
```

```
['logistic regression', 'support vector machine', 'decision tree classifier', 'k nearest neighbors']
[0.8888888888888888, 0.8888888888888888, 0.6666666666666666, 0.8518518518518519]
```

```python
fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(methods, accu, color ='maroon',
        width = 0.4)

plt.xlabel("Methods")
plt.ylabel("Accuracy")
plt.title("Best Perfomed Method")
plt.show()
```



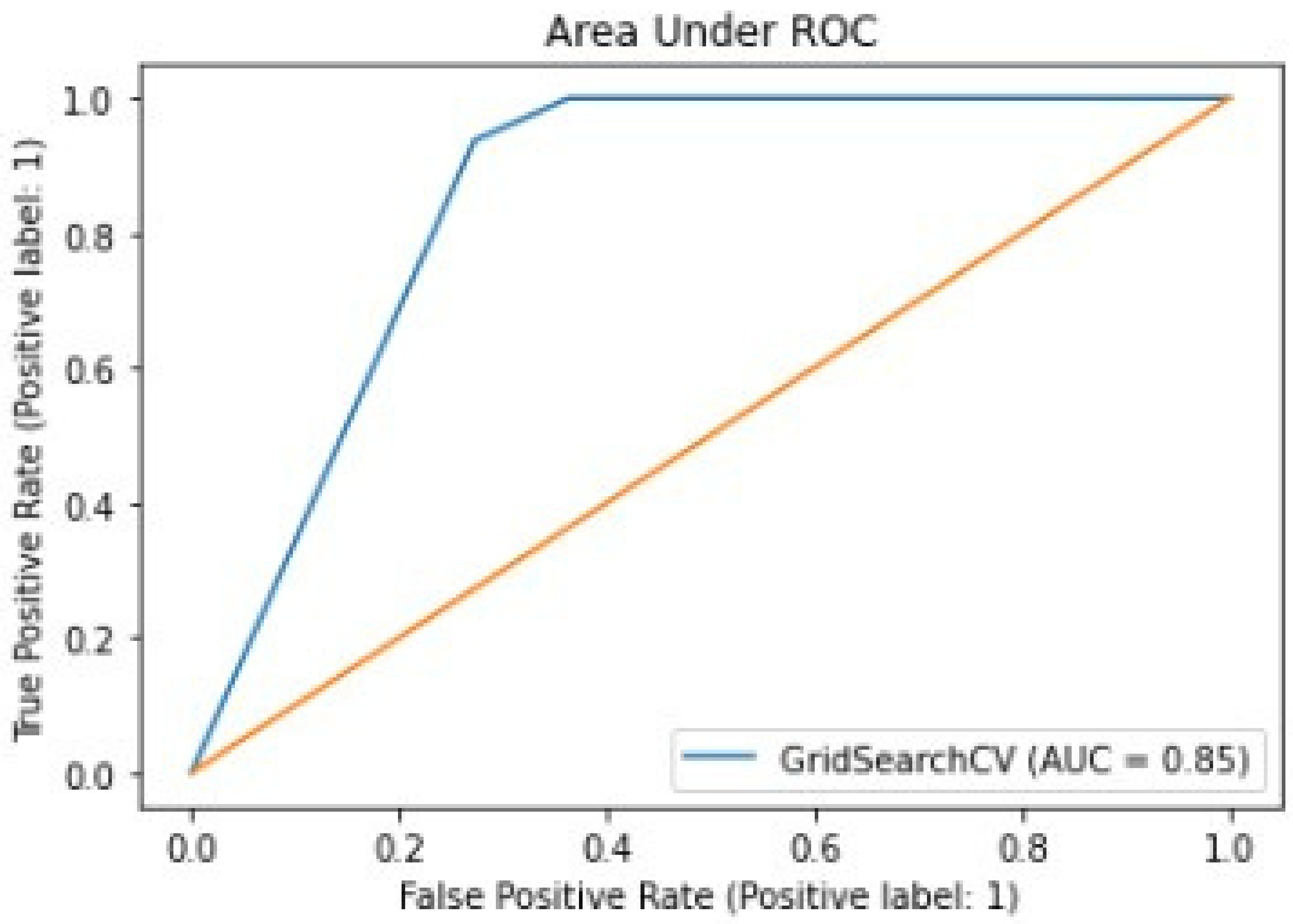
Receiver Operating Characteristic Curve (ROC Curve)



ROC Curve in sklearn

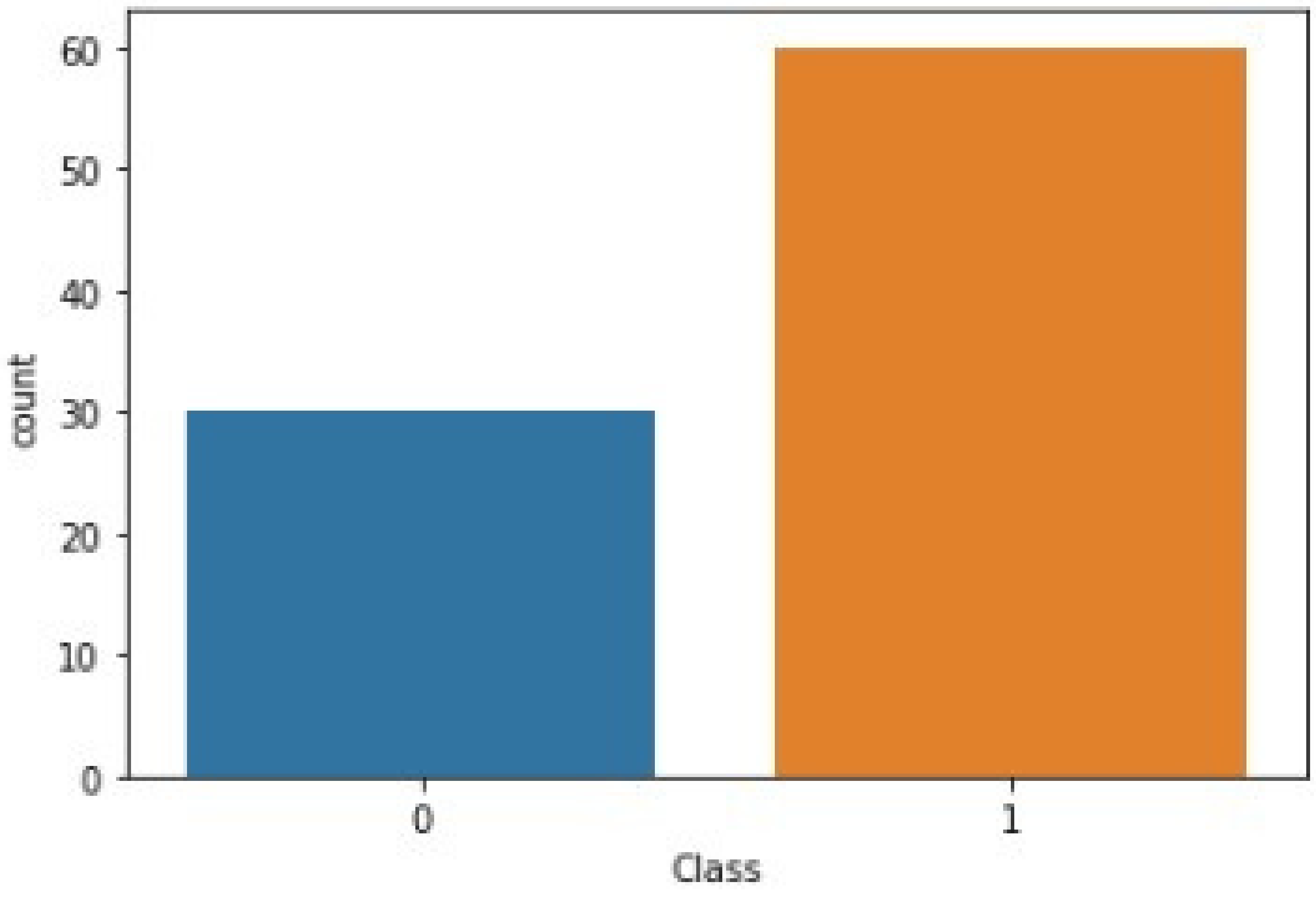
```
In [68]: from sklearn.metrics import RocCurveDisplay
```

```
In [70]: RocCurveDisplay.from_estimator(knn_cv,X_test,Y_test)
         plt.plot([0,1],[0,1])
         plt.title('Area Under ROC')

Out[70]: Text(0.5, 1.0, 'Area Under ROC')
```



Imbalance Data

```
In [71]:  sns.countplot(x ='Class', data = data)

Out[71]: <AxesSubplot:xlabel='Class', ylabel='count'>
```



Assignment:

تمرین:

کدهای ارائه شده در درس را بررسی کرده و پروژه خود را در در گیت هاب آپلود کنید.