```
import numpy as np
import pandas as pd
import seaborn as sns
```

```
titanic=sns.load_dataset('titanic')
df=titanic
df.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	a
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	F
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	F
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	F
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	

Next steps: Generate code with df View recommended plots

```
# finding the null values
df.isnull().sum()
```

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

```
# discarding the most null columns in the above output
newdf=df.drop("deck",axis="columns")
```

```
# displaying the dataset without the deck column
newdf.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	Southampton	no	True

Next steps: Generate code with newdf View recommended plots

```
#now checking again the column having null values
newdf.isnull().sum()
```

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
embark_town      2
alive            0
alone            0
dtype: int64
```

```
# handling the null value in age column
from sklearn.impute import SimpleImputer
imp=SimpleImputer(strategy="mean")
newdf["age"]=imp.fit_transform(newdf[["age"]])
print(f"the number of null values in the age column is: {newdf.age.isnull().sum()}")
```

```
the number of null values in the age column is: 0
```

```
# printing the age column after removing the null values
newdf.age
```

```
0      22.000000
1      38.000000
```

```
2      26.000000
3      35.000000
4      35.000000
         ...
886    27.000000
887    19.000000
888    29.699118
889    26.000000
890    32.000000
Name: age, Length: 891, dtype: float64
```

```
# checkiing the null values
newdf.isnull().sum()
```

```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       2
class          0
who            0
adult_male     0
embark_town    2
alive          0
alone          0
dtype: int64
```

```
# checking the most frequent occuring in the embarked column
newdf.groupby('embarked').size()
```

```
embarked
C    168
Q     77
S    644
dtype: int64
```

```
#checking the most common occuring in the embark_town column
newdf.groupby("embark_town").size()
```

```
embark_town
Cherbourg      168
Queenstown      77
Southampton    644
dtype: int64
```

```
# replacing the null values with the most frequent occuring
newdf["embarked"].fillna("S",inplace=True)
newdf["embark_town"].fillna("Southampton",inplace=True)
```

```
# checking the null values
newdf.isnull().sum()
# we can see that we have removed and take care of all our null values
# now preprocessing is done
```

```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
embark_town    0
alive          0
alone          0
dtype: int64
```

```
# now applying the classifiers Decesion tree
# first we will convert our data from catagorical to integer
dataset=pd.get_dummies(newdf,drop_first=True)
```

```
dataset.columns
```

```
Index(['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare', 'adult_male',
       'alone', 'sex_male', 'embarked_Q', 'embarked_S', 'class_Second',
       'class_Third', 'who_man', 'who_woman', 'embark_town_Queenstown',
       'embark_town_Southampton', 'alive_yes'],
      dtype='object')
```

```
dataset.head()
```

	survived	pclass	age	sibsp	parch	fare	adult_male	alone	sex_male	embarked_Q	embarked_S	class_Second	clas
0	0	3	22.0	1	0	7.2500	True	False	True	False	True	False	
1	1	1	38.0	1	0	71.2833	False	False	False	False	False	False	
2	1	3	26.0	0	0	7.9250	False	True	False	False	True	False	
3	1	1	35.0	1	0	53.1000	False	False	False	False	True	False	
4	0	3	35.0	0	0	8.0500	True	True	True	False	True	False	

Next steps: Generate code with dataset | View recommended plots

```
dataset.columns[7]#dependent variable to be predict
```

```
'alone'
```

```
X=dataset.iloc[:,[0,1,2,3,4,5,6,8,9,10,11,13,14]]
y=dataset.iloc[:,7].values  #will store values of column 7 in the form of list in the varuiable y
```

```
len(X.columns)
```

```
13
```

```
X.head()
```

	survived	pclass	age	sibsp	parch	fare	adult_male	sex_male	embarked_Q	embarked_S	class_Second	who_man	wh
0	0	3	22.0	1	0	7.2500	True	True	False	True	False	True	
1	1	1	38.0	1	0	71.2833	False	False	False	False	False	False	
2	1	3	26.0	0	0	7.9250	False	False	False	True	False	False	
3	1	1	35.0	1	0	53.1000	False	False	False	True	False	False	
4	0	3	35.0	0	0	8.0500	True	True	False	True	False	True	

Next steps: Generate code with X | View recommended plots

```
X.columns #columns of the dataset stored in the variable x
```

```
Index(['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare', 'adult_male',
       'sex_male', 'embarked_Q', 'embarked_S', 'class_Second', 'who_man',
```

```
             'who_woman'],
            dtype='object')
```

```
#alone column y whether the person is alone or not

y
```

```
        True, False, False,  True,  True, False, False,  True, False,
        True,  True,  True, False,  True, False,  True,  True, False,
        True,  True,  True, False,  True, False, False, False,  True,
        True,  True,  True,  True,  True,  True, False, False,  True,
```

```
       False,  True,  True,  True,  True, False,  True, False,  True,
       False,  True,  True,  True, False, False,  True,  True,  True,
       False,  True,  True,  True,  True,  True,  True, False,  True,
       False, False, False,  True,  True,  True,  True,  True, False,
        True,  True,  True, False,  True,  True,  True, False,  True,
       False, False,  True,  True, False, False,  True,  True, False,
        True,  True, False, False,  True,  True,  True, False,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
       False,  True, False, False, False,  True, False, False, False,
       False, False,  True, False,  True, False, False,  True, False,
        True,  True, False,  True,  True, False,  True, False,  True,
        True, False,  True,  True,  True,  True, False, False,  True,
        True,  True,  True, False,  True,  True, False,  True,  True])
```

```
# using the holdout method to use 25% as the test set and rest as the training set

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=42) #random_state to ensure the randomness that the this function
x1_train,x1_test,y1_train,y1_test=train_test_split(X,y,test_size=0.333,random_state=42)
# always gives the same output
```

```
# standardizing the dataset
from sklearn.preprocessing import StandardScaler
sc= StandardScaler() #splitting the datset into training and testin data
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=42)
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)

x1_train,x1_test,y1_test,y1_train=train_test_split(X,y,test_size=0.333,random_state=42)
x1_train=sc.fit_transform(x1_train)
x1_test=sc.transform(x1_test)
```

```
x_test[:10]
```

```
array([[ 1.2807483 ,  0.80934914,  0.0171447 ,  0.37665554,  0.78899607,
        -0.32839086,  0.80584286,  0.72224656, -0.30835364, -1.67843464,
        -0.51725447,  0.80584286, -0.64905824],
       [-0.78079354, -0.40558395,  0.11721087, -0.46765956, -0.46887833,
        -0.42042549,  0.80584286,  0.72224656, -0.30835364,  0.59579323,
         1.93328442,  0.80584286, -0.64905824],
       [-0.78079354,  0.80934914, -0.72892862, -0.46765956, -0.46887833,
        -0.4703621 ,  0.80584286,  0.72224656, -0.30835364,  0.59579323,
        -0.51725447,  0.80584286, -0.64905824],
       [ 1.2807483 , -0.40558395, -1.80583342, -0.46765956,  0.78899607,
         0.01591384, -1.24093672, -1.38456873, -0.30835364,  0.59579323,
         1.93328442, -1.24093672, -0.64905824],
       [ 1.2807483 ,  0.80934914, -1.19045925,  0.37665554, -0.46887833,
        -0.40604181, -1.24093672, -1.38456873, -0.30835364, -1.67843464,
```

```
        -0.51725447, -1.24093672, -0.64905824],
       [ 1.2807483 , -1.62051704, -0.26739799, -0.46765956, -0.46887833,
         0.90507644, -1.24093672, -1.38456873, -0.30835364,  0.59579323,
        -0.51725447, -1.24093672,  1.5406938 ],
       [ 1.2807483 ,  0.80934914,  0.0171447 , -0.46765956, -0.46887833,
        -0.47375585, -1.24093672, -1.38456873,  3.24302966, -1.67843464,
        -0.51725447, -1.24093672,  1.5406938 ],
       [-0.78079354,  0.80934914, -1.0366157 ,  1.22097065, -0.46887833,
        -0.27497904,  0.80584286,  0.72224656, -0.30835364,  0.59579323,
        -0.51725447,  0.80584286, -0.64905824],
       [ 1.2807483 ,  0.80934914, -1.0366157 , -0.46765956, -0.46887833,
        -0.47375585, -1.24093672, -1.38456873,  3.24302966, -1.67843464,
        -0.51725447, -1.24093672,  1.5406938 ],
       [ 1.2807483 , -1.62051704, -0.80585039, -0.46765956,  2.04687047,
        -0.11434217, -1.24093672, -1.38456873, -0.30835364,  0.59579323,
        -0.51725447, -1.24093672,  1.5406938 ]])
```

```
x1_test[:10]
```

```
array([[ 1.29447892,  0.80525855,  0.01395342,  0.34215257,  0.77351281,
        -0.3234312 ,  0.80907113,  0.72864795, -0.30974338, -1.66836432,
        -0.51626013,  0.80907113, -0.65150628],
       [-0.77251161, -0.41184979,  0.11466552, -0.47080194, -0.46494166,
        -0.41555873,  0.80907113,  0.72864795, -0.30974338,  0.59938947,
         1.937008  ,  0.80907113, -0.65150628],
       [-0.77251161,  0.80525855, -0.73693573, -0.47080194, -0.46494166,
        -0.46554575,  0.80907113,  0.72864795, -0.30974338,  0.59938947,
        -0.51626013,  0.80907113, -0.65150628],
       [ 1.29447892, -0.41184979, -1.82079188, -0.47080194,  0.77351281,
         0.02122106, -1.23598526, -1.37240488, -0.30974338,  0.59938947,
         1.937008  , -1.23598526, -0.65150628],
       [ 1.29447892,  0.80525855, -1.20144551,  0.34215257, -0.46494166,
        -0.40116053, -1.23598526, -1.37240488, -0.30974338, -1.66836432,
        -0.51626013, -1.23598526, -0.65150628],
       [ 1.29447892, -1.62895814, -0.27242596, -0.47080194, -0.46494166,
         0.91128121, -1.23598526, -1.37240488, -0.30974338,  0.59938947,
        -0.51626013, -1.23598526,  1.53490462],
       [ 1.29447892,  0.80525855,  0.01395342, -0.47080194, -0.46494166,
        -0.46894293, -1.23598526, -1.37240488,  3.22847904, -1.66836432,
        -0.51626013, -1.23598526,  1.53490462],
       [-0.77251161,  0.80525855, -1.04660892,  1.15510709, -0.46494166,
        -0.26996547,  0.80907113,  0.72864795, -0.30974338,  0.59938947,
        -0.51626013,  0.80907113, -0.65150628],
       [ 1.29447892,  0.80525855, -1.04660892, -0.47080194, -0.46494166,
        -0.46894293, -1.23598526, -1.37240488,  3.22847904, -1.66836432,
        -0.51626013, -1.23598526,  1.53490462],
       [ 1.29447892, -1.62895814, -0.81435403, -0.47080194,  2.01196728,
        -0.10916644, -1.23598526, -1.37240488, -0.30974338,  0.59938947,
        -0.51626013, -1.23598526,  1.53490462]])
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
dtree=DecisionTreeClassifier()
dtree.fit(x_train,y_train)
dtree.fit(x1_train,y1_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-51-768a49962c4d> in <cell line: 3>()
      1 dtree=DecisionTreeClassifier()
      2 dtree.fit(x_train,y_train)
----> 3 dtree.fit(x1_train,y1_train)

1 frames
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py in fit(self, X, y, sample_weight, check_input)
    300
    301         if len(y) != n_samples:
--> 302             raise ValueError(
    303                 "Number of labels=%d does not match number of samples=%d"
    304                 % (len(y), n_samples)

ValueError: Number of labels=297 does not match number of samples=594
```

Next steps: Explain error

```
# applying on test set
y_pred_dtree=dtree.predict(x_test)
```

```
# validation metrics accuracy=correct predictions/all predictions

from sklearn.metrics import accuracy_score,confusion_matrix
cm=confusion_matrix(y_test,y_pred_dtree)
print(cm)
print("Decision tree model accuracy(in %)",accuracy_score(y_test,y_pred_dtree)*100)
```

```
[[ 92   0]
 [  0 131]]
Decision tree model accuracy(in %) 100.0
```

```
from sklearn import tree
print(tree.export_text(dtree))
```

```
|--- feature_3 <= -0.05
|   |--- feature_4 <= 0.16
|   |   |--- class: True
|   |--- feature_4 >  0.16
|   |   |--- class: False
|--- feature_3 >  -0.05
```

```
|   |--- class: False
```

NAIVE BAYES CLASSIFIER

```
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
```

```
x=dataset.iloc[:,[0,1,2,3,4,5,6,8,9,10,11,12,13,14]]
y=dataset.iloc[:,8].values
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=0)
```

```
gnb=GaussianNB()
y_pred=gnb.fit(x_train,y_train)
```

```
y_pred_gnb=gnb.predict(x_test)
```

** CREATING A CONFUSION MATRIX**

```
cm=confusion_matrix(y_test,y_pred_gnb)
print(cm)
print("BAYESIAN MODEL ACURRACY(IN %):",accuracy_score(y_test,y_pred_dtree)*100)
```

```
[[ 83   0]
 [  0 140]]
BAYESIAN MODEL ACURRACY(IN %): 54.7085201793722
```

KNN CLASSIFIERS

```
knn=KNeighborsClassifier(n_neighbors=7)
knn.fit(x_train,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

```
y_pred_knn=knn.predict(x_test)
```

```
cm=confusion_matrix(y_test,y_pred_knn)
print(cm)
print("K-Nearest Neighbors model accuracy:",accuracy_score(y_test,y_pred_knn)*100)
```

```
[[ 48  35]
 [ 16 124]]
K-Nearest Neighbors model accuracy: 77.13004484304933
```

USING K FOLD CROSS-VALIDATION

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

```
cv = KFold(n_splits=10, random_state=1, shuffle=True)
```

```
# Naive Bayes Classifier

gnb = GaussianNB()
y_pred = gnb.fit(X, y)

y_pred_gnb = gnb.predict(X)

from sklearn.metrics import accuracy_score, confusion_matrix
cm = confusion_matrix(y,y_pred_gnb)
print(cm)
print("Gaussian Naive Bayes model accuracy(in %):", accuracy_score(y, y_pred_gnb)*100)
```

```
[[314   0]
 [  0 577]]
Gaussian Naive Bayes model accuracy(in %): 100.0
```

```
scores = cross_val_score(gnb, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance

print('Accuracy: %.3f' % (np.mean(scores)))
```

```
Accuracy: 1.000
```

```
# K-Nearest Neighbor Classifier

knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X, y)

y_pred_knn = knn.predict(X)

cm = confusion_matrix(y,y_pred_knn)
print(cm)
print("K-Nearest Neighbors model accuracy(in %):", accuracy_score(y, y_pred_knn)*100)
```

```
[[253  61]
 [ 47 530]]
K-Nearest Neighbors model accuracy(in %): 87.87878787878788
```

```
scores = cross_val_score(knn, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f' % (np.mean(scores)))
```

```
Accuracy: 0.796
```

RANDOM SAMPLING

```
def split_data(test_size, random_state):
    from sklearn.model_selection import train_test_split
    x_train, x_test, y_train, y_test = train_test_split(X,y,test_size=test_size, random_state=random_state, shuffle=True)
    # Standardizing the Dataset
    from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()# Splitting the dataset into Training and Testing Data
    X_train = sc.fit_transform(x_train)
    X_test = sc.transform(x_test)

    gnb = GaussianNB()
    y_pred = gnb.fit(X_train, y_train)

    y_pred_gnb = gnb.predict(X_test)
    print("\nGaussian Naive Bayes model accuracy(in %):", accuracy_score(y_test, y_pred_gnb)*100)
    accuracies_gnb.append(accuracy_score(y_test, y_pred_gnb))

    # K-Nearest Neighbor Classifier

    knn = KNeighborsClassifier(n_neighbors=7)
    knn.fit(X_train, y_train)

    y_pred_knn = knn.predict(X_test)


    print("K-Nearest Neighbors model accuracy(in %):", accuracy_score(y_test, y_pred_knn)*100)
    accuracies_knn.append(accuracy_score(y_test, y_pred_knn))

    # Decision Tree Classifier

    dtree = DecisionTreeClassifier()
    dtree.fit(X_train, y_train)


    y_pred_dtree = dtree.predict(X_test)
    print("Decision Tree model accuracy(in %):", accuracy_score(y_test, y_pred_dtree)*100)
    accuracies_dtree.append(accuracy_score(y_test, y_pred_dtree))
```

```
accuracies_gnb = []
accuracies_dtree = []
accuracies_knn = []
```

```python
from sklearn.metrics import accuracy_score
num = 42
for i in range(10):
    print(f"\nIter {i}:")
    split_data(test_size = 0.25, random_state = num)
    num += 10
    print("-"*100)
```

```
Decision Tree model accuracy(in %): 100.0
----------------------------------------------------------------------------------------------------

Iter 2:

Gaussian Naive Bayes model accuracy(in %): 100.0
K-Nearest Neighbors model accuracy(in %): 98.20627802690582
Decision Tree model accuracy(in %): 100.0
----------------------------------------------------------------------------------------------------

Iter 3:

Gaussian Naive Bayes model accuracy(in %): 100.0
K-Nearest Neighbors model accuracy(in %): 99.55156950672645
Decision Tree model accuracy(in %): 100.0
----------------------------------------------------------------------------------------------------

Iter 4:

Gaussian Naive Bayes model accuracy(in %): 100.0
K-Nearest Neighbors model accuracy(in %): 99.55156950672645
Decision Tree model accuracy(in %): 100.0
----------------------------------------------------------------------------------------------------

Iter 5:

Gaussian Naive Bayes model accuracy(in %): 100.0
K-Nearest Neighbors model accuracy(in %): 99.10313901345292
Decision Tree model accuracy(in %): 100.0
----------------------------------------------------------------------------------------------------

Iter 6:
```

```
Iter 8:

Gaussian Naive Bayes model accuracy(in %): 100.0
K-Nearest Neighbors model accuracy(in %): 99.10313901345292
```