Before building any ML model with the given data, we split our dataset into test and train set in certain percentage depends upon the availability of count of dataset. Mostly, **Test Set : 20 -30 % of data** & **Train Set : 70–80 % of data**where, Accuracy / performance of model will be checked by the test dataset. But this 70- 30 % volume of data is randomly selected out of all datapoints which leads to fluctuation in accuracy. This is controlled by assigning a definite value to Variable **random_state**.

Random state will decide the splitting of data into test and train set and using a particular finite number(It can take any positive value) will ensure same results will be reproduced again and again. But for different random_state splitting of test and train will be different and hence accuracy obtained will be different and results in fluctuation of accuracy .

We need to validate the accuracy of our ML model and here comes the role of cross validation: It is a technique for evaluating the accuracy of ML models by training a models using different subsets of data for certain number of iterations. The final output of the model will be average of all. It also mitigates the effect of overfitting.

Mostly there are 6 types of cv methods:

1. Hold Out Method
2. Leave One out Cross Validation

3. K Fold Cross Validation
4. Stratified K Fold Cross Validation
5. Time Series Cross Validation
6. Repeated Random Test-Train Splits or Monte Carlo cross-validation

Lets see one by one:

1.Hold Out Method

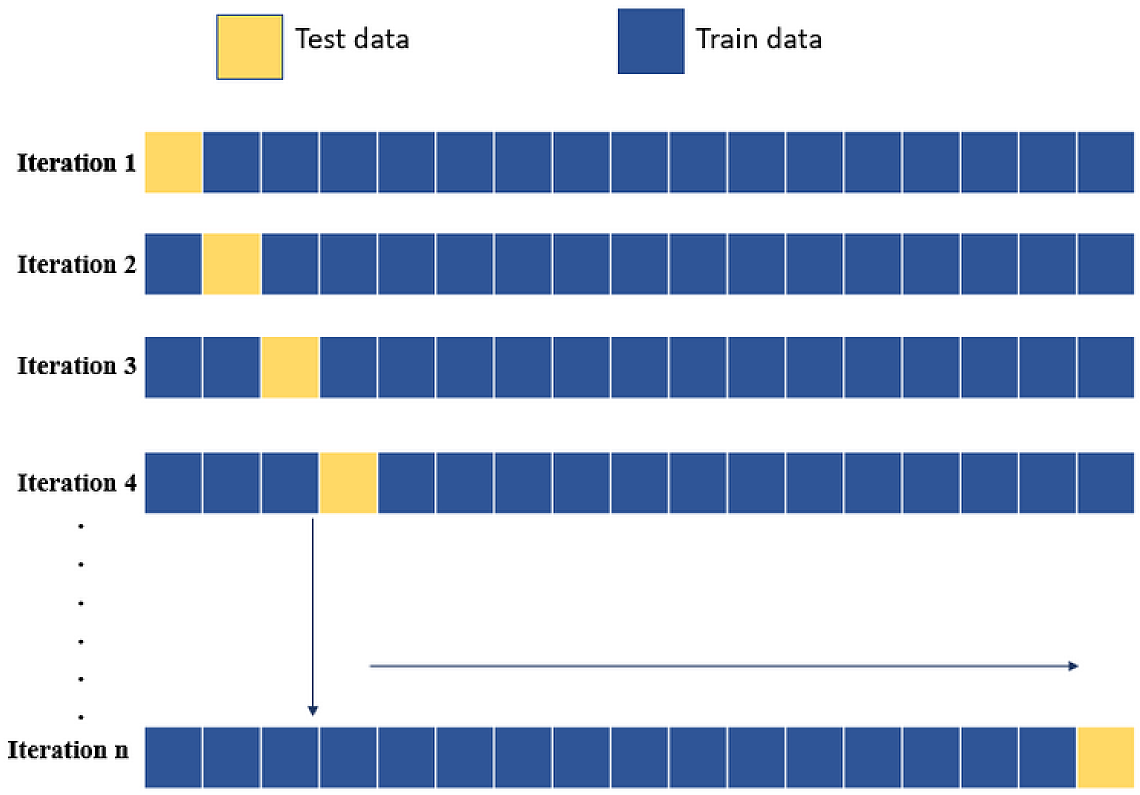
This is simply splitting the data into training & test set. Percentage of training data is more than test data. Post that using training set for training the model and remaining test set for error estimation.

Disadvantages : There will be chances of high variance because any random sample of data and pattern associated with it may get selected into test data. Since we are validating model with test data, accuracy and model generalization would be negatively affected.

2.Leave One Out Cross Validation (LOOCV):

In this, out of all data points **one data** is left as test data and rest as training data. So for **n** data points we have to perform **n** iterations to cover each data point.

Leave-P-Out Cross Validation is a special case leaving p datapoints for testing and validation and n-p for training the model.



LOOCV iterations. Credits: Author

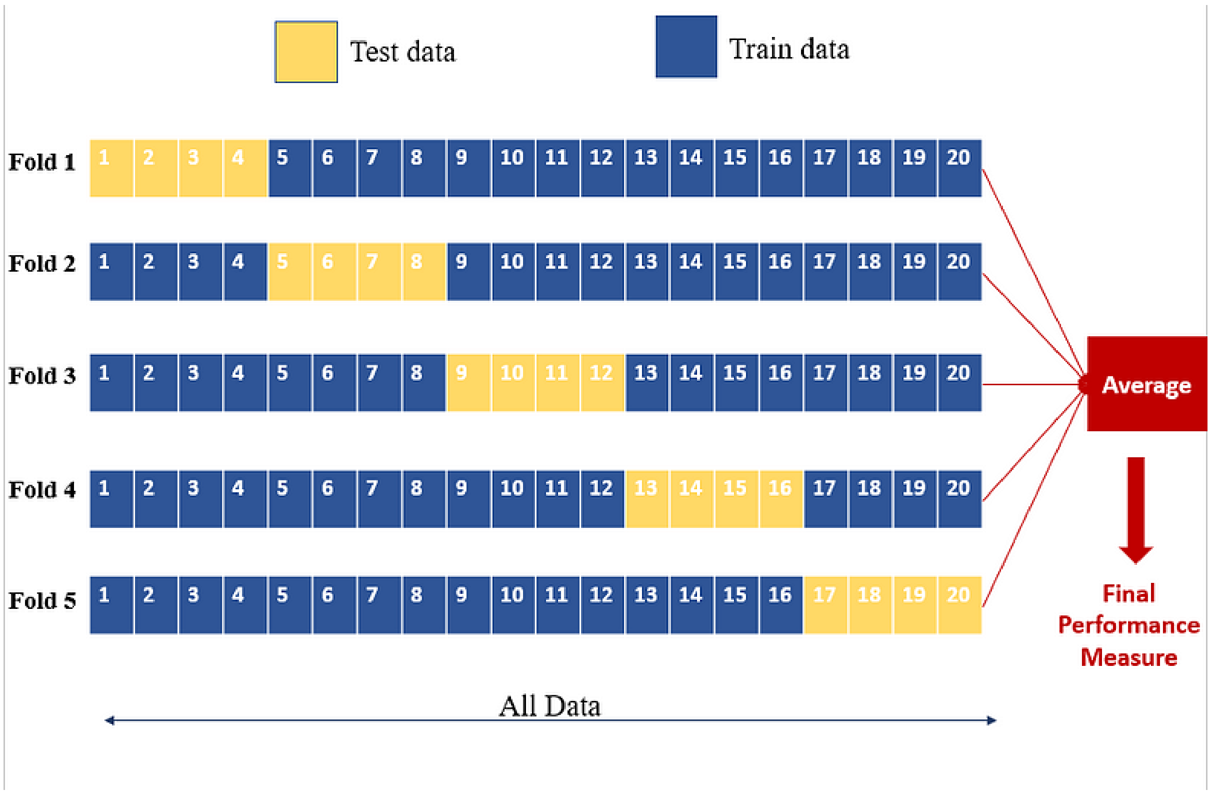
Shortcomings :

i) High Computing power is required since many iterations are required for each data point of large datasets.

ii) Since **n-1** data points are used as training data, overfitting will happen results in low bias but it won't produce a generalized model resulting in high error and low accuracy. It was used long back, now a days no one uses it.

3. K fold Cross Validation :

In this, Whole **n dataset** is divided into k parts with **n/k =p** and then this **p** will be taken as test data in each iteration and next p in next iteration and so on till **k** iterations . For ex. 20 datapoints for 5 fold cross validation, 20/5 =4, so the given dataset will be divided as shown in below image. Sets will be different for different fold. **Each data will be considered one time in test set and k-1 times in training set enhancing the effectiveness of this method**. Each fold will give different accuracy and final accuracy will be average of all these 5 accuracies. Also, we will be able to obtain **minimum** and **maximum** accuracy of this particular model.



5-fold cross validation iterations. Credits : Author

Advantages:

i) Efficient use of data as each data point is used for both training and testing purpose.

Low bias because most of the data is used for training.

Low variance because almost each datapoint is used in test set as well.

ii)Accuracy is high.

Ideally a value between 5 -10 is preferred for K. But it can take any value. Higher value of K will leads in accuracy similar to LOOCV method.

Disadvantages :

i) I**mbalanced dataset results low accuracy with this method** , Lets say for a binary classification problem, in test data we have maximum instances of output 1 so it won't give accurate result with respect to particular model. Or in price prediction, all the data selected for test set have high price so again accuracy will be affected.

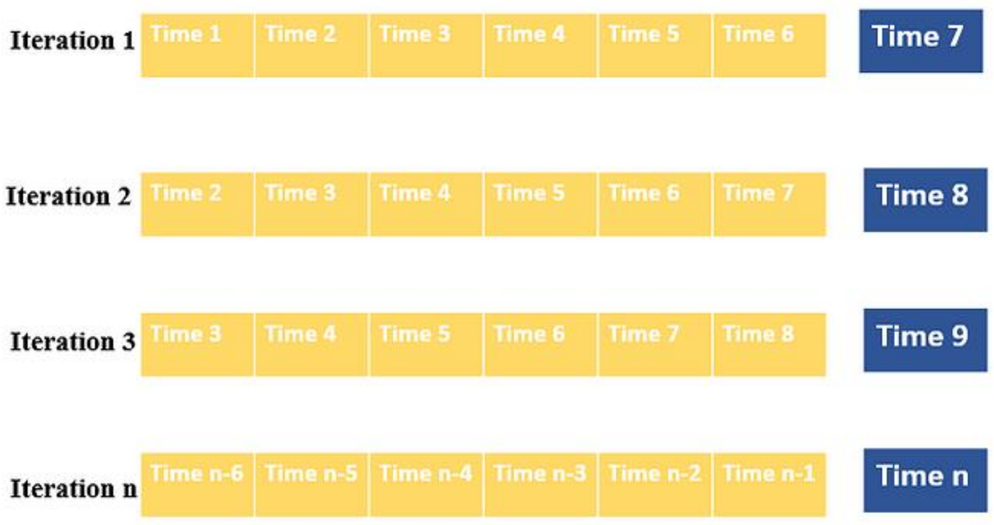
To overcome this, we use stratified Cross Validation.

4. Stratified Cross Validation :

In this, random sample populated in train and test dataset is such that the number of instances of each class in each iterations of training and test data splitting is taken in good proportion of yes and no, 0 & 1, or highs and lows so that model gives good accuracy.

5. Time Series Cross Validation :

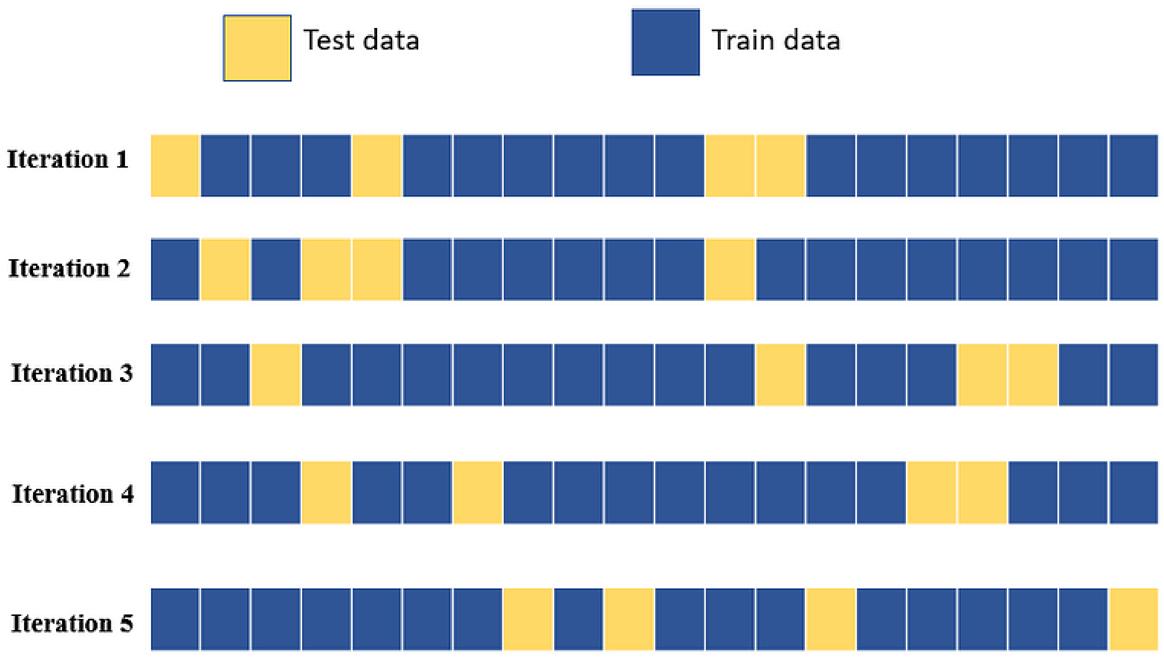
It is completely for time series data like stock price prediction, sales prediction. Input is sequentially getting added into the training data as shown below.



Time Series CV. credits : Author

6.Repeated Random Test-Train Splits or Monte Carlo cross-validation:

It involves both traditional train test split and K-fold CV. Here random splitting of dataset is done into train and test set and then further process of splitting and performance measurement is repeated for number of times specified by us. Cross Validation is performed.



Repeated Random Test Train Split. Credits: Author

Disadvantages:

i)It is not suitable for imbalance dataset.

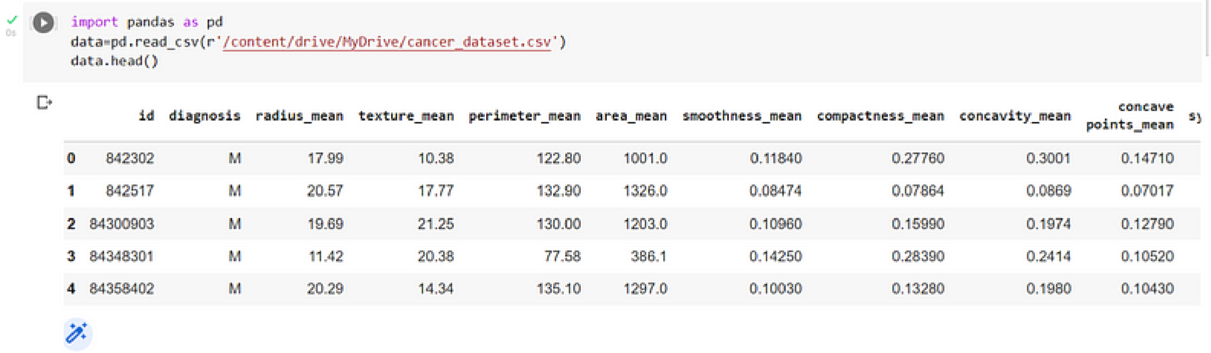
ii)Chances are that some samples didn't get selected for either of train and test data.

Practical Implications Using Sklearn and Python:

Now we are implementing all above techniques using python and sklearn for building a simple ML model. It's simply for understanding the Cross validation techniques so other hyperparameters of Regressor Classifiers are at default value.

We are considering a dataset of cancer to predict the type of cancer on the basis of various feature i.e, Benign (B) & Malignant (M).

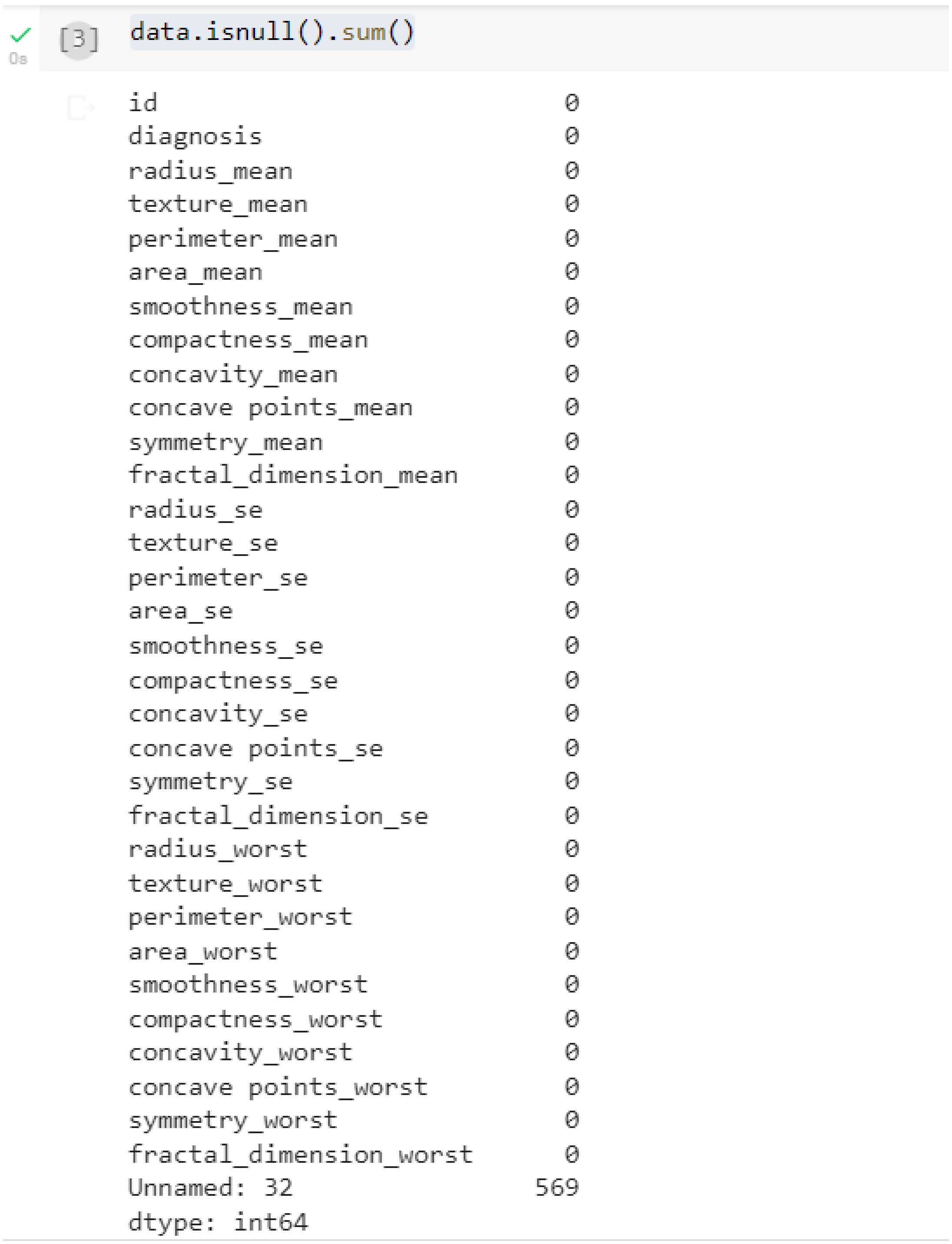
```
import pandas as pd
data=pd.read_csv(r'/content/drive/MyDrive/cancer_dataset.csv')
data.head()
```



```
import pandas as pd
data=pd.read_csv(r'/content/drive/MyDrive/cancer_dataset.csv')
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	sy
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	

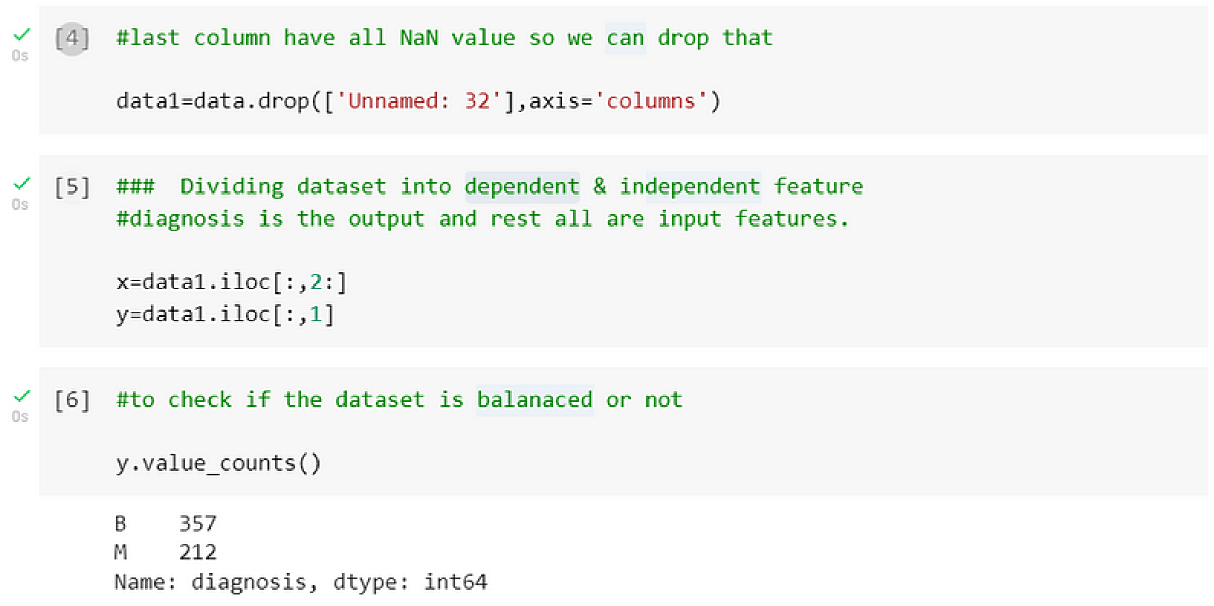
```
#Removing Null Values
data.isnull().sum()
```



```
[3] data.isnull().sum()
```

```
id                          0
diagnosis                   0
radius_mean                 0
texture_mean                0
perimeter_mean              0
area_mean                   0
smoothness_mean             0
compactness_mean            0
concavity_mean              0
concave points_mean         0
symmetry_mean               0
fractal_dimension_mean      0
radius_se                   0
texture_se                  0
perimeter_se                0
area_se                     0
smoothness_se               0
compactness_se              0
concavity_se                0
concave points_se           0
symmetry_se                 0
fractal_dimension_se        0
radius_worst                0
texture_worst               0
perimeter_worst             0
area_worst                  0
smoothness_worst            0
compactness_worst           0
concavity_worst             0
concave points_worst        0
symmetry_worst              0
fractal_dimension_worst     0
Unnamed: 32               569
dtype: int64
```

```
#last column have all NaN value so we can drop
thatdata1=data.drop(['Unnamed: 32'],axis='columns')### Dividing
dataset into dependent & independent feature#diagnosis is the
output and rest all are input features.x=data1.iloc[:,2:]
y=data1.iloc[:,1]#to check if the dataset is balanaced or
noty.value_counts()
```



```
[4] #last column have all NaN value so we can drop that

    data1=data.drop(['Unnamed: 32'],axis='columns')
```

```
[5] ###  Dividing dataset into dependent & independent feature
    #diagnosis is the output and rest all are input features.

    x=data1.iloc[:,2:]
    y=data1.iloc[:,1]
```

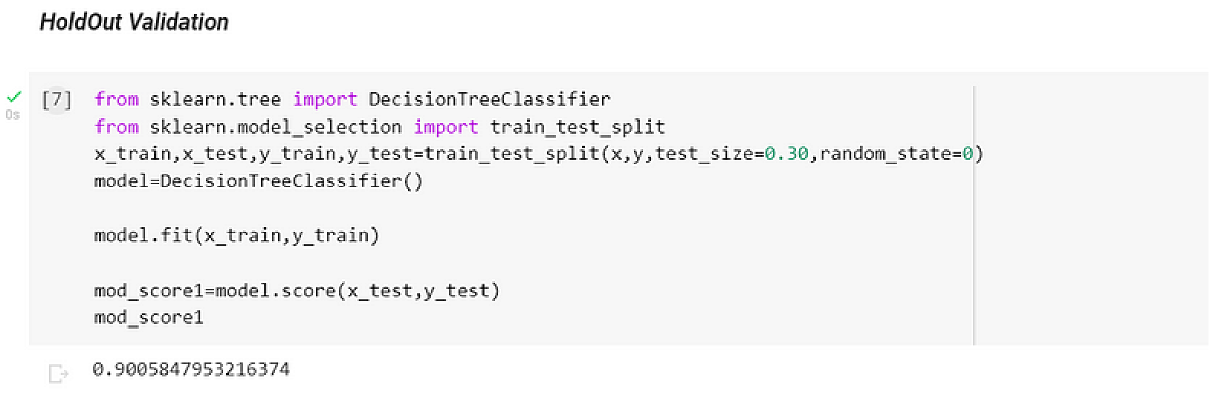
```
[6] #to check if the dataset is balanaced or not

    y.value_counts()
```

```
B    357
M    212
Name: diagnosis, dtype: int64
```

Now we are building ML models using Different CV techniques.

1.HoldOut Validation

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import
train_test_splitx_train,x_test,y_train,y_test=train_test_split(x
,y,test_size=0.30,random_state=0)
model=DecisionTreeClassifier()
model.fit(x_train,y_train)
mod_score1=model.score(x_test,y_test)
mod_score1
```



HoldOut Validation

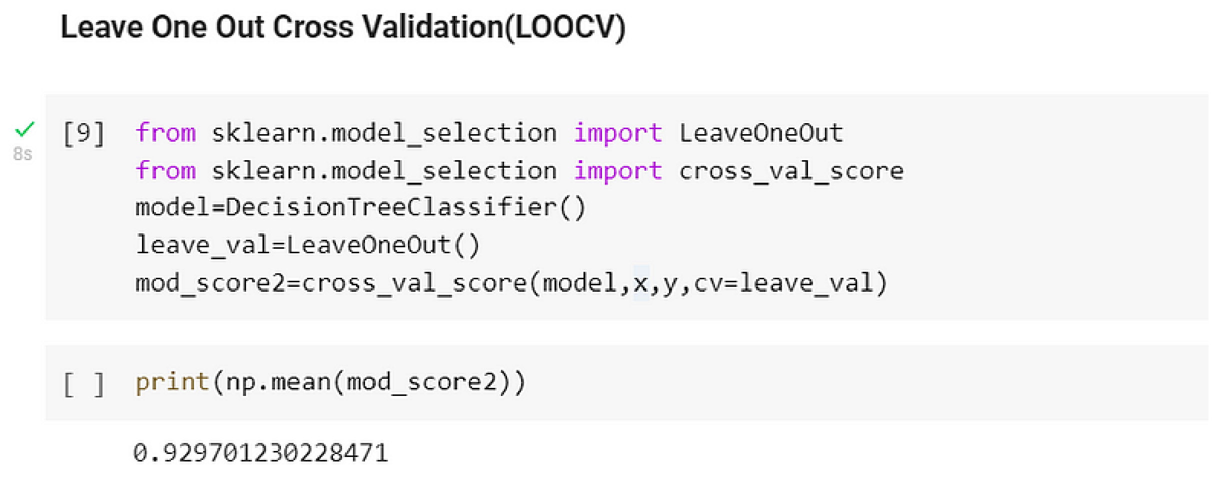
```
[7] from sklearn.tree import DecisionTreeClassifier
    from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=0)
    model=DecisionTreeClassifier()

    model.fit(x_train,y_train)

    mod_score1=model.score(x_test,y_test)
    mod_score1
```

```
0.9005847953216374
```

2. Leave One Out Cross Validation(LOOCV)

```
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
model=DecisionTreeClassifier()
```

```
leave_val=LeaveOneOut()
mod_score2=cross_val_score(model,x,y,cv=leave_val)print(np.mean(
mod_score2))
```



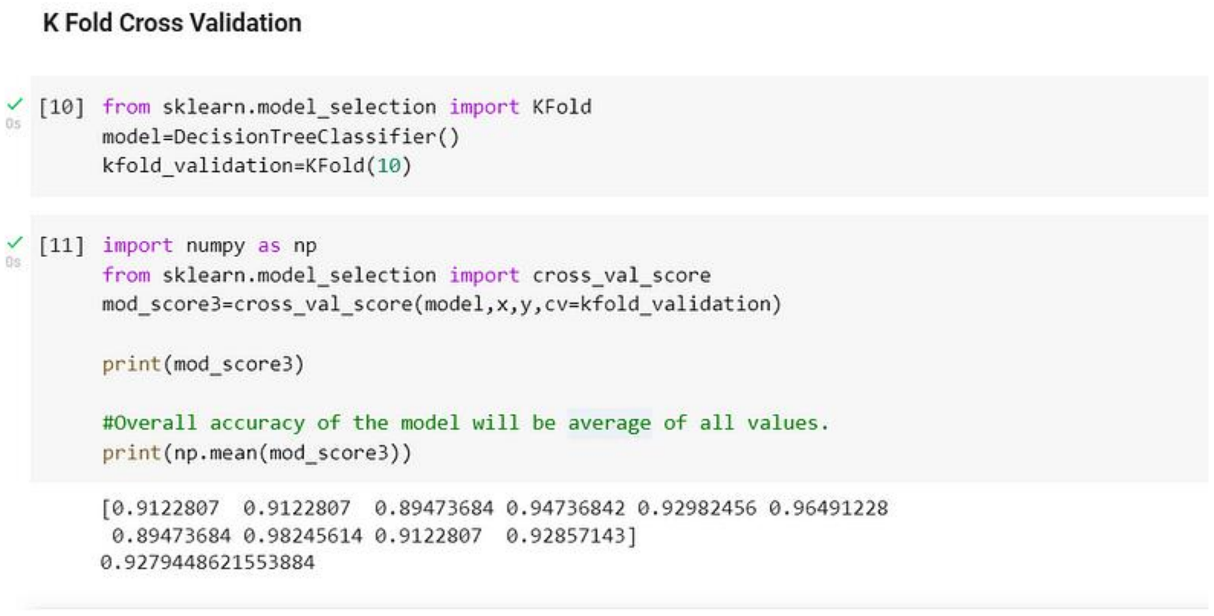
Leave One Out Cross Validation(LOOCV)

```
[9]  from sklearn.model_selection import LeaveOneOut
     from sklearn.model_selection import cross_val_score
     model=DecisionTreeClassifier()
     leave_val=LeaveOneOut()
     mod_score2=cross_val_score(model,x,y,cv=leave_val)
```

```
[ ]  print(np.mean(mod_score2))

     0.929701230228471
```

3.K Fold Cross Validation

```
from sklearn.model_selection import KFold
model=DecisionTreeClassifier()
kfold_validation=KFold(10)import numpy as np
from sklearn.model_selection import cross_val_score
mod_score3=cross_val_score(model,x,y,cv=kfold_validation)
print(mod_score3)#Overall accuracy of the model will be average
of all values.
print(np.mean(mod_score3))
```



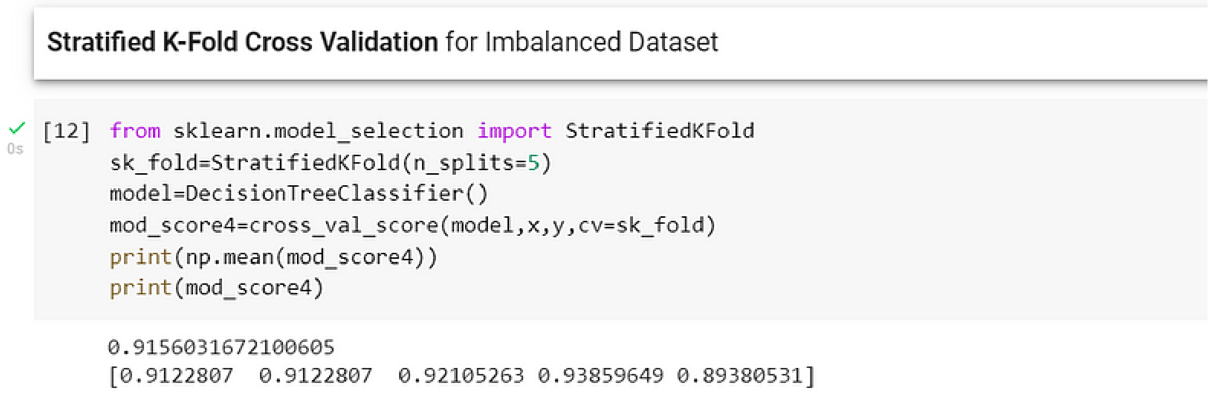
K Fold Cross Validation

```
[10] from sklearn.model_selection import KFold
     model=DecisionTreeClassifier()
     kfold_validation=KFold(10)
```

```
[11] import numpy as np
     from sklearn.model_selection import cross_val_score
     mod_score3=cross_val_score(model,x,y,cv=kfold_validation)

     print(mod_score3)

     #Overall accuracy of the model will be average of all values.
     print(np.mean(mod_score3))

     [0.9122807  0.9122807  0.89473684 0.94736842 0.92982456 0.96491228
      0.89473684 0.98245614 0.9122807  0.92857143]
     0.9279448621553884
```

4. Stratified K-Fold Cross Validation

```
from sklearn.model_selection import StratifiedKFold
sk_fold=StratifiedKFold(n_splits=5)
model=DecisionTreeClassifier()
mod_score4=cross_val_score(model,x,y,cv=sk_fold)
print(np.mean(mod_score4))
print(mod_score4)
```



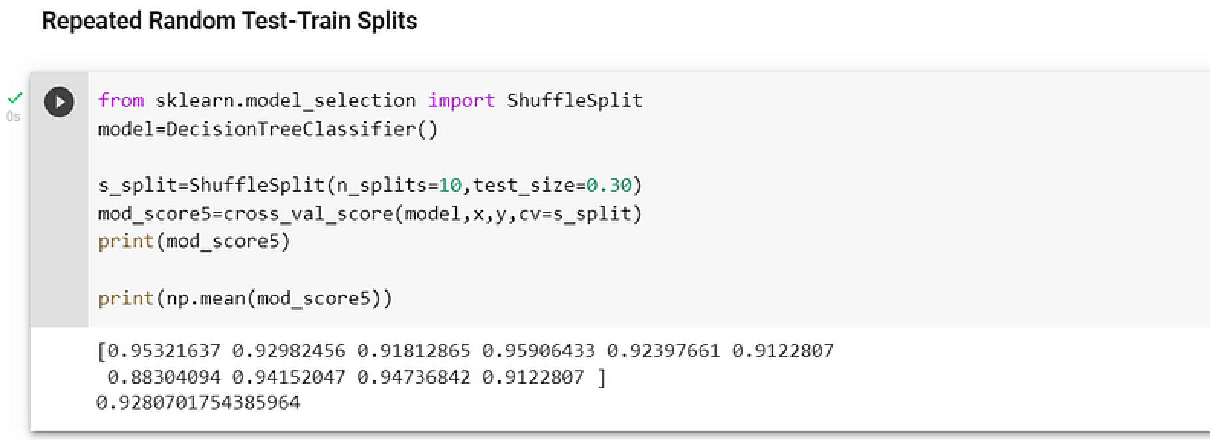
Stratified K-Fold Cross Validation for Imbalanced Dataset

```
[12] from sklearn.model_selection import StratifiedKFold
     sk_fold=StratifiedKFold(n_splits=5)
     model=DecisionTreeClassifier()
     mod_score4=cross_val_score(model,x,y,cv=sk_fold)
     print(np.mean(mod_score4))
     print(mod_score4)
```

```
0.9156031672100605
[0.9122807  0.9122807  0.92105263 0.93859649 0.89380531]
```

5.Repeated Random Test-Train Split

```
from sklearn.model_selection import ShuffleSplit
model=DecisionTreeClassifier()
s_split=ShuffleSplit(n_splits=10,test_size=0.30)
mod_score5=cross_val_score(model,x,y,cv=s_split)
print(mod_score5)
print(np.mean(mod_score5))
```



Repeated Random Test-Train Splits

```
from sklearn.model_selection import ShuffleSplit
model=DecisionTreeClassifier()

s_split=ShuffleSplit(n_splits=10,test_size=0.30)
mod_score5=cross_val_score(model,x,y,cv=s_split)
print(mod_score5)

print(np.mean(mod_score5))
```

```
[0.95321637 0.92982456 0.91812865 0.95906433 0.92397661 0.9122807
 0.88304094 0.94152047 0.94736842 0.9122807 ]
0.9280701754385964
```

With this, we have covered almost every point of cross validation.