DATA ANALYSIS OF DIABETES **DATABASE** ***

```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")#call the class for adaboost
from sklearn.ensemble import AdaBoostClassifier
```

```
df=pd.read_csv("diabetes.csv")
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

```
#check null values
df.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
#check inormation
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
```

```
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
#display all rows
pd.set_option('display.max_rows',None)
```

```
#display all columns
pd.set_option('display.max_columns',None)
```

```
#seperate the independent (input X) and dependent (output Y) variable
X=df.drop("Outcome",axis=1)
Y=df["Outcome"]
```

```
#call class for training and testing
from sklearn.model_selection import train_test_split
```

```
#split data into training and testing
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3, random_state=1)
```

```
#create a function
def create_model(model):
    model.fit(X_train,Y_train)#to train
    y_pred=model.predict(X_test)
    print(classification_report(Y_test,y_pred))
    return model
```

```
#as we have used classification report we hhave to call the class for it
from sklearn.metrics import classification_report
```

```
#call class for logistic regression
from sklearn.linear_model import LogisticRegression
```

```
#create object for LogisticRegression class
lr=LogisticRegression()
```

```
#call the function "create_model"(for training and testing)
create_model(lr)
```

```
              precision    recall  f1-score   support

           0       0.79      0.90      0.84       146
           1       0.78      0.58      0.66        85
```

```
      accuracy                           0.78       231
     macro avg       0.78      0.74      0.75       231
  weighted avg       0.78      0.78      0.78       231

  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                     intercept_scaling=1, l1_ratio=None, max_iter=100,
                     multi_class='auto', n_jobs=None, penalty='l2',
                     random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                     warm_start=False)
```

```
LogisticRegression()
```

```
  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                     intercept_scaling=1, l1_ratio=None, max_iter=100,
                     multi_class='auto', n_jobs=None, penalty='l2',
                     random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                     warm_start=False)
```

```
#call the class for DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
#create object of DecisionTreeClassifier class
dt=DecisionTreeClassifier()
```

```
#call function "create_model"(for training and testing)
dt=create_model(dt)
```

```
                precision    recall  f1-score   support

             0       0.73      0.82      0.77       146
             1       0.60      0.47      0.53        85

      accuracy                           0.69       231
     macro avg       0.66      0.64      0.65       231
  weighted avg       0.68      0.69      0.68       231
```

```
#to check the information gain (if information gain is high then the column will be :
dt.feature_importances_
```

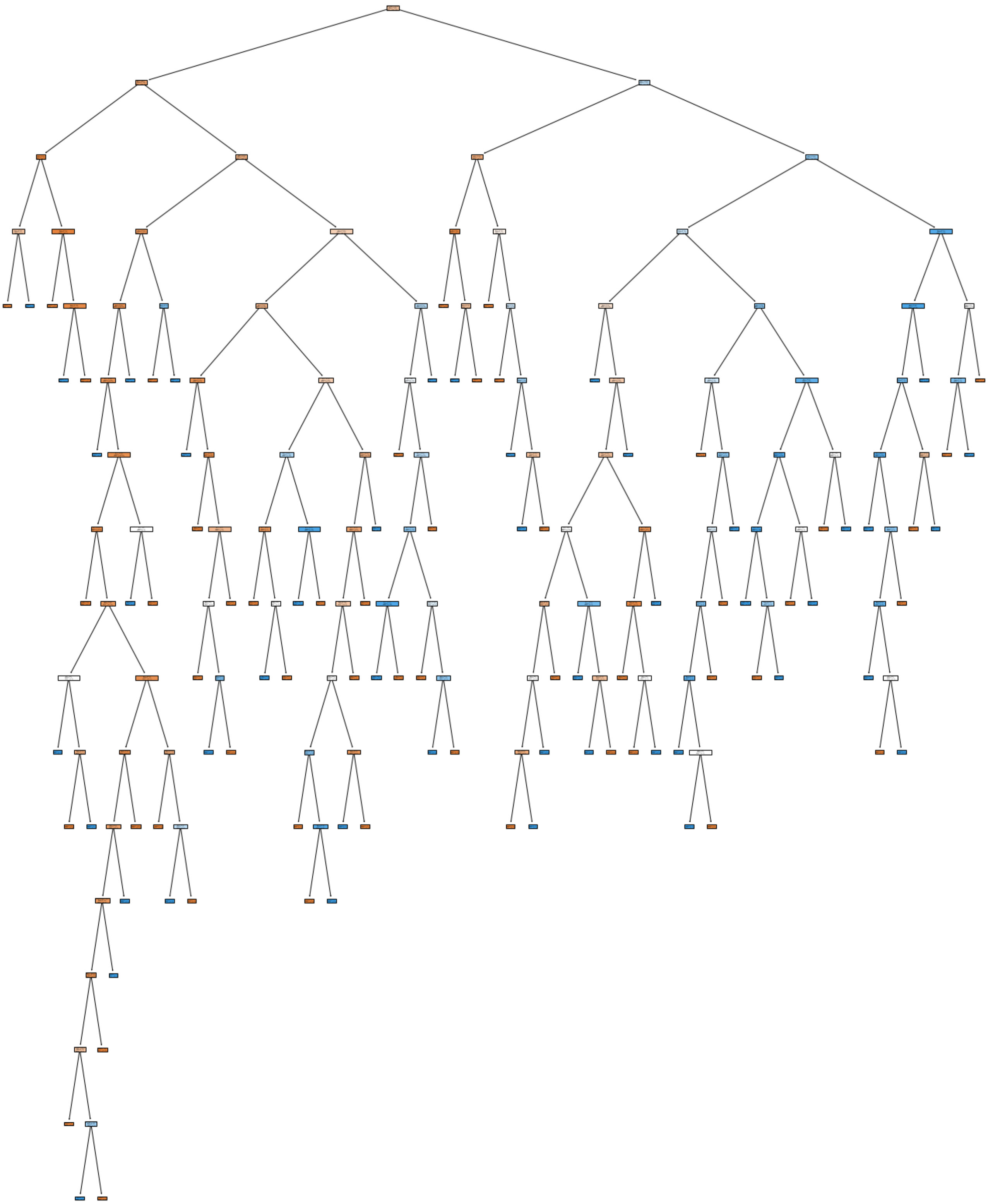
```
  array([0.0707469 , 0.26611293, 0.16709624, 0.0400096 , 0.05283241,
         0.18573917, 0.10574786, 0.11171488])
```

```
#call the class for tree
from sklearn import tree
```

```
#keep all the input variables in features
features=X.columns
```

```
#draw the tree
fig=plt.figure(figsize=(23,30))
```

```
fig=plt.figure(figsize=(25,30))
_=tree.plot_tree(dt,feature_names=features,filled=True)
```



```
#(1) max_depth
#we create a different object of DecisionTreeClasiifier and pass parameter for max_de
dt1=DecisionTreeClassifier(max_depth=5) #not mpore than 8


#call function "create_model"(for training and testing)
dt1=create_model(dt1)

                    precision    recall  f1-score   support
```

```
              0       0.80      0.82      0.81       146
              1       0.68      0.65      0.66        85

       accuracy                           0.76       231
      macro avg       0.74      0.73      0.74       231
   weighted avg       0.76      0.76      0.76       231
```

```
#it is giving good recall when the max_depth is 5 i.e 0.67
```

```
#(2) min_sample_leaf
#we create a different object of DecisionTreeClasiifier and pass parameter for min_sa
dt2=DecisionTreeClassifier(min_samples_leaf=45) #min 50
```

```
#call function "create_model"(for training and testing)
dt2=create_model(dt2)
```

```
                 precision    recall  f1-score   support

              0       0.83      0.86      0.84       146
              1       0.74      0.71      0.72        85

       accuracy                           0.80       231
      macro avg       0.79      0.78      0.78       231
   weighted avg       0.80      0.80      0.80       231
```

```
#(3) entropy
#we create a different object of DecisionTreeClasiifier and pass parameter for min_sa
dt3=DecisionTreeClassifier(min_samples_leaf=40,criterion="entropy")
```

```
#call function "create_model"(for training and testing)
dt3=create_model(dt3)
```

```
                 precision    recall  f1-score   support

              0       0.82      0.85      0.83       146
              1       0.72      0.67      0.70        85

       accuracy                           0.78       231
      macro avg       0.77      0.76      0.76       231
   weighted avg       0.78      0.78      0.78       231
```

it is not giving nice recall that is the model is getting overfit so the max-depth is the best technique for training as it is giving good recall but recall is not still that good so we will apply next algorithm

Double-click (or enter) to edit

SUPPORT VECTOR MACHINE (SVM)

```
#call class form svm
from sklearn.svm import LinearSVC


#create object for LinearSVC
svc=LinearSVC(random_state=1,C=0.0012) #c is paramete for adding error


#call function "create_model"(for training and testing)
create_model(svc)
```

```
              precision    recall  f1-score   support

           0       0.68      0.88      0.77       146
           1       0.59      0.31      0.40        85

    accuracy                           0.67       231
   macro avg       0.64      0.59      0.59       231
weighted avg       0.65      0.67      0.63       231

LinearSVC(C=0.0012, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=1, tol=0.0001,
          verbose=0)
```

```
LinearSVC(C=0.0012, random_state=1)
```

```
LinearSVC(C=0.0012, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=1, tol=0.0001,
          verbose=0)
```

ENSEMBLING TECHNIQUE

1. ADABOOST

```
#call the class for adaboost
from sklearn.ensemble import AdaBoostClassifier



#create object of Adaboost
ada=AdaBoostClassifier(n_estimators=100)


#call function "create_model"
create_model(ada)
```

```
              precision    recall  f1-score   support

           0       0.82      0.88      0.85       146
           1       0.77      0.66      0.71        85
```

```
        accuracy                           0.80       231
       macro avg       0.79      0.77      0.78       231
    weighted avg       0.80      0.80      0.80       231

    AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                       n_estimators=100, random_state=None)
```

```
AdaBoostClassifier(n_estimators=100)
```

```
    AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                       n_estimators=100, random_state=None)
```

2. GRADIENT BOOST

```
#call the class for GradientBoost
from sklearn.ensemble import GradientBoostingClassifier
```

```
#create object of GradientBoostingClassifier
gbc=GradientBoostingClassifier(n_estimators=100)
```

```
#call function"create_model"
create_model(gbc)
```

```
                  precision    recall  f1-score   support

               0       0.81      0.88      0.85       146
               1       0.76      0.65      0.70        85

        accuracy                           0.80       231
       macro avg       0.79      0.77      0.77       231
    weighted avg       0.79      0.80      0.79       231

    GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                               learning_rate=0.1, loss='deviance', max_depth=3,
                               max_features=None, max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=100,
                               n_iter_no_change=None, presort='deprecated',
                               random_state=None, subsample=1.0, tol=0.0001,
                               validation_fraction=0.1, verbose=0,
                               warm_start=False)
```

```
GradientBoostingClassifier()
```

```
    GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                               learning_rate=0.1, loss='deviance', max_depth=3,
                               max_features=None, max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=100,
                               n_iter_no_change=None, presort='deprecated',
                               random_state=None, subsample=1.0, tol=0.0001,
```

```
                                         validation_fraction=0.1, verbose=0,
                                         warm_start=False)
```

3. NAIVE

```
#create model list
model_list=[("Logistic",lr),("DecisonTree",dt),("DecisionTreeEntropy",dt3)]
```

```
#call class for hard voting
from sklearn.ensemble import VotingClassifier
```

```
# (A) HARD VOTING
```

```
f VotingClassifier
ier(estimators=model_list) #by default it takes hard voting so if we are not specify
```

```
#call function "create_model"
create_model(vc)
```

```
                   precision    recall  f1-score   support

                0       0.80      0.88      0.84       146
                1       0.75      0.61      0.68        85

         accuracy                           0.78       231
        macro avg       0.77      0.75      0.76       231
     weighted avg       0.78      0.78      0.78       231

     VotingClassifier(estimators=[('Logistic',
                                   LogisticRegression(C=1.0, class_weight=None,
                                                      dual=False, fit_intercept=True,
                                                      intercept_scaling=1,
                                                      l1_ratio=None, max_iter=100,
                                                      multi_class='auto',
                                                      n_jobs=None, penalty='l2',
                                                      random_state=None,
                                                      solver='lbfgs', tol=0.0001,
                                                      verbose=0, warm_start=False)),
                                  ('DecisonTree',
                                   DecisionTreeClassifier(ccp_alpha=0.0,
                                                          class_weight=None,
                                                          cr...
                                   DecisionTreeClassifier(ccp_alpha=0.0,
                                                          class_weight=None,
                                                          criterion='entropy',
                                                          max_depth=None,
                                                          max_features=None,
                                                          max_leaf_nodes=None,
                                                          min_impurity_decrease=0.0,
                                                          min_impurity_split=None,
                                                          min_samples_leaf=40,
                                                          min_samples_split=2,
                                                          min_weight_fraction_leaf=0.0,
```

```
                                             presort='deprecated',
                                             random_state=None,
                                             splitter='best'))],
                     flatten_transform=True, n_jobs=None, voting='hard',
                     weights=None)
```

```
#4. BOOTSTRAPING
```

```
#create model list
model_list=[("Logistic",lr),("DecisonTree",dt),("DecisionTreeEntropy",dt3)]
```

```
#call class for BaggingClassifier
from sklearn.ensemble import BaggingClassifier
```

```
object of BaggingClassifier
ngClassifier(LogisticRegression(),n_estimators=10,max_samples=10,random_state=1) #by
```

(B) PASTING

```
#create object of BaggingClassifier
bc1=BaggingClassifier(LogisticRegression(),n_estimators=10,max_samples=10,random_sta
```

```
#call function "create_model"
create_model(bc1)
```

```
              precision    recall  f1-score   support

           0       0.78      0.88      0.83       146
           1       0.74      0.56      0.64        85

    accuracy                           0.77       231
   macro avg       0.76      0.72      0.73       231
weighted avg       0.76      0.77      0.76       231

BaggingClassifier(base_estimator=LogisticRegression(C=1.0, class_weight=None,
                                                    dual=False,
                                                    fit_intercept=True,
                                                    intercept_scaling=1,
                                                    l1_ratio=None, max_iter=100,
                                                    multi_class='auto',
                                                    n_jobs=None, penalty='l2',
                                                    random_state=None,
                                                    solver='lbfgs', tol=0.0001,
                                                    verbose=0,
                                                    warm_start=False),
                  bootstrap=False, bootstrap_features=False, max_features=1.0,
                  max_samples=10, n_estimators=10, n_jobs=None, oob_score=False,
                  random_state=1, verbose=0, warm_start=False)
```

(C) RANDOM FOREST

```
#call class for RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier


#create object of RandomForestClassifier
rf=RandomForestClassifier(n_estimators=10,max_features=7,random_state=1)


#call function "create_model"
create_model(rf)
```

```
              precision    recall  f1-score   support

           0       0.76      0.90      0.82       146
           1       0.75      0.52      0.61        85

    accuracy                           0.76       231
   macro avg       0.75      0.71      0.72       231
weighted avg       0.76      0.76      0.75       231

RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features=7,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=1, verbose=0,
                       warm_start=False)
```

FEATURE SELECTION TECHNIQUE

A . ANOVA TEST

```
from sklearn.feature_selection import f_regression #f_regression is for anova test
from sklearn.feature_selection import SelectKBest
#selectKBest is important object


#create object for SlectKBest class
anova=SelectKBest(score_func=f_regression,k=6) #k is no of columns/feature which ret
#anov auser define object


#train data and return 10 best column and store in X_train_imp
X_train_imp=anova.fit_transform(X_train,Y_train)


#test the data
X_test_imp=anova.transform(X_test)


#check which features are selected and which are rejected
anova.get_support() #return ans in boolean type 0 means false 1 means true
```

```
array([ True,  True, False, False,  True,  True,  True,  True])
```

```
#again create object of logistic regression after anova test
lr1=LogisticRegression()
```

```
#train model
lr1.fit(X_train_imp,Y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
#check the score
lr1.score(X_test_imp,Y_test)
```

```
0.7748917748917749
```

```
#CHI SQUARE TEST
```

```
# before applying chi square test to check which if column has non negative
for col in X:
    print("column name:",col)
    print(df[col].unique())
```

```
 40.6 47.9 50.  25.2 40.9 37.2 44.2 29.9 31.9 28.4 43.5 32.7 67.1 45.
 34.9 27.7 35.9 22.6 33.1 30.4 52.3 24.3 22.9 34.8 30.9 40.1 23.9 37.5
 35.5 42.8 42.6 41.8 35.8 37.8 28.8 23.6 35.7 36.7 45.2 44.  46.2 35.
 43.6 44.1 18.4 29.2 25.9 32.1 36.3 40.  25.1 27.5 45.6 27.8 24.9 25.3
 37.9 27.  26.  38.7 20.8 36.1 30.7 32.3 52.9 21.  39.7 25.5 26.2 19.3
 38.1 23.5 45.5 23.1 39.9 36.8 21.8 41.  42.2 34.4 27.2 36.5 29.8 39.2
 38.4 36.2 48.3 20.  22.3 45.7 23.7 22.1 42.1 42.4 18.2 26.4 45.3 37.
 24.5 32.2 59.4 21.2 26.7 30.2 46.1 41.3 38.8 35.2 42.3 40.7 46.5 33.5
 37.3 30.3 26.3 21.7 36.4 28.5 26.9 38.6 31.3 19.5 20.1 40.8 23.4 28.3
 38.9 57.3 35.6 49.6 44.6 24.1 44.5 41.2 49.3 46.3]
column name: DiabetesPedigreeFunction
[0.627 0.351 0.672 0.167 2.288 0.201 0.248 0.134 0.158 0.232 0.191 0.537
 1.441 0.398 0.587 0.484 0.551 0.254 0.183 0.529 0.704 0.388 0.451 0.263
 0.205 0.257 0.487 0.245 0.337 0.546 0.851 0.267 0.188 0.512 0.966 0.42
 0.665 0.503 1.39  0.271 0.696 0.235 0.721 0.294 1.893 0.564 0.586 0.344
 0.305 0.491 0.526 0.342 0.467 0.718 0.962 1.781 0.173 0.304 0.27  0.699
 0.258 0.203 0.855 0.845 0.334 0.189 0.867 0.411 0.583 0.231 0.396 0.14
 0.391 0.37  0.307 0.102 0.767 0.237 0.227 0.698 0.178 0.324 0.153 0.165
 0.443 0.261 0.277 0.761 0.255 0.13  0.323 0.356 0.325 1.222 0.179 0.262
 0.283 0.93  0.801 0.207 0.287 0.336 0.247 0.199 0.543 0.192 0.588 0.539
 0.22  0.654 0.223 0.759 0.26  0.404 0.186 0.278 0.496 0.452 0.403 0.741
 0.361 1.114 0.457 0.647 0.088 0.597 0.532 0.703 0.159 0.268 0.286 0.318
 0.272 0.572 0.096 1.4   0.218 0.085 0.399 0.432 1.189 0.687 0.137 0.637
 0.833 0.229 0.817 0.204 0.368 0.743 0.722 0.256 0.709 0.471 0.495 0.18
 0.542 0.773 0.678 0.719 0.382 0.319 0.19  0.956 0.084 0.725 0.299 0.244
 0.745 0.615 1.321 0.64  0.142 0.374 0.383 0.578 0.136 0.395 0.187 0.905
 0.15  0.874 0.236 0.787 0.407 0.605 0.151 0.289 0.355 0.29  0.375 0.164
 0.431 0.742 0.514 0.464 1.224 1.072 0.805 0.209 0.666 0.101 0.198 0.652
 2.329 0.089 0.645 0.238 0.394 0.293 0.479 0.686 0.831 0.582 0.446 0.402
 1.318 0.329 1.213 0.427 0.282 0.143 0.38  0.284 0.249 0.926 0.557 0.092
 0.655 1.353 0.612 0.2   0.226 0.997 0.933 1.101 0.078 0.24  1.136 0.128
 0.422 0.251 0.677 0.296 0.454 0.744 0.881 0.28  0.259 0.619 0.808 0.34
```

```
 0.434 0.757 0.613 0.692 0.52  0.412 0.84  0.839 0.156 0.215 0.326 1.391
 0.875 0.313 0.433 0.626 1.127 0.315 0.345 0.129 0.527 0.197 0.731 0.148
 0.123 0.127 0.122 1.476 0.166 0.932 0.343 0.893 0.331 0.472 0.673 0.389
 0.485 0.349 0.279 0.346 0.252 0.243 0.58  0.559 0.302 0.569 0.378 0.385
 0.499 0.306 0.234 2.137 1.731 0.545 0.225 0.816 0.528 0.509 1.021 0.821
 0.947 1.268 0.221 0.66  0.239 0.949 0.444 0.463 0.803 1.6   0.944 0.196
 0.241 0.161 0.135 0.376 1.191 0.702 0.674 1.076 0.534 1.095 0.554 0.624
 0.219 0.507 0.561 0.421 0.516 0.264 0.328 0.233 0.108 1.138 0.147 0.727
 0.435 0.497 0.23  0.955 2.42  0.658 0.33  0.51  0.285 0.415 0.381 0.832
 0.498 0.212 0.364 1.001 0.46  0.733 0.416 0.705 1.022 0.269 0.6   0.571
 0.607 0.17  0.21  0.126 0.711 0.466 0.162 0.419 0.63  0.365 0.536 1.159
 0.629 0.292 0.145 1.144 0.174 0.547 0.163 0.738 0.314 0.968 0.409 0.297
 0.525 0.154 0.771 0.107 0.493 0.717 0.917 0.501 1.251 0.735 0.804 0.661
 0.549 0.825 0.423 1.034 0.16  0.341 0.68  0.591 0.3   0.121 0.502 0.401
 0.601 0.748 0.338 0.43  0.892 0.813 0.693 0.575 0.371 0.206 0.417 1.154
 0.925 0.175 1.699 0.682 0.194 0.4   0.1   1.258 0.482 0.138 0.593 0.878
 0.157 1.282 0.141 0.246 1.698 1.461 0.347 0.362 0.393 0.144 0.732 0.115
 0.465 0.649 0.871 0.149 0.695 0.303 0.61  0.73  0.447 0.455 0.133 0.155
 1.162 1.292 0.182 1.394 0.217 0.631 0.88  0.614 0.332 0.366 0.181 0.828
 0.335 0.856 0.886 0.439 0.253 0.598 0.904 0.483 0.565 0.118 0.177 0.176
 0.295 0.441 0.352 0.826 0.97  0.595 0.317 0.265 0.646 0.426 0.56  0.515
 0.453 0.785 0.734 1.174 0.488 0.358 1.096 0.408 1.182 0.222 1.057 0.766
 0.171]
column name: Age
[50 31 32 21 33 30 26 29 53 54 34 57 59 51 27 41 43 22 38 60 28 45 35 46
 56 37 48 40 25 24 58 42 44 39 36 23 61 69 62 55 65 47 52 66 49 63 67 72
 81 64 70 68]
```

```
#call class for chi square test
from sklearn.feature_selection import chi2


#create object
chi=SelectKBest(score_func=chi2,k=6)


#train data and return 10 best column and store in X_train_imp
X_train_imp=chi.fit_transform(X_train,Y_train)


#check which features are selected and which are rejected
anova.get_support() #return ans in boolean type 0 means false 1 means true
```

```
array([ True,  True, False, False,  True,  True,  True,  True])
```

```
#again create object of logistic regression after anova test
lr1=LogisticRegression()


#train model
lr1.fit(X_train_imp,Y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
#check the score
lr1.score(X_test_imp,Y_test)
```

```
0.6320346320346321
```

```
#2. WRAPPER METHOD
```

```
#declare empty list for the columns
columns=[]
```

```
for col in X:
    columns.append(col)
    X_new=df[columns] #input variable
    X_train,X_test,Y_train,Y_test = train_test_split(X_new,Y,test_size=0.3,random_sta
    #create a object of LogisticRegression
    lin=LogisticRegression()
    #we train the model
    lin.fit(X_train,Y_train)
    #find the score
    score1=lin.score(X_test,Y_test)
    print("Column : ",col, " Score : ",score1)
```

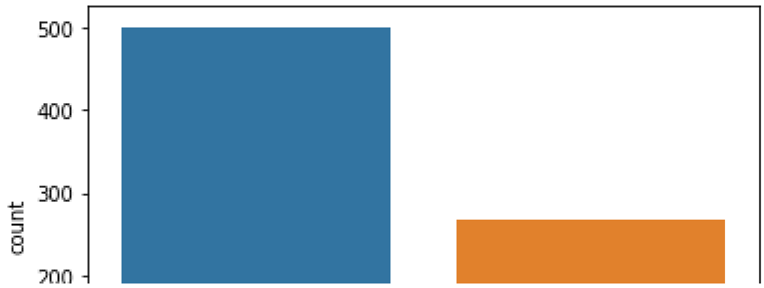
```
Column :  Pregnancies  Score :  0.6536796536796536
Column :  Glucose  Score :  0.7532467532467533
Column :  BloodPressure  Score :  0.7619047619047619
Column :  SkinThickness  Score :  0.7705627705627706
Column :  Insulin  Score :  0.7748917748917749
Column :  BMI  Score :  0.7835497835497836
Column :  DiabetesPedigreeFunction  Score :  0.7878787878787878
Column :  Age  Score :  0.7835497835497836
```

SAMPLING TECHNIQUE

```
#to count how many yes and how many no
df["Outcome"].value_counts()
```

```
0    500
1    268
Name: Outcome, dtype: int64
```

```
#display using countplot
sns.countplot(data=df,x="Outcome")
plt.show()
```



There are two types of samplying techniques suppose - yes = 10 (minority class) no = 1000 (majority class) 1) Over Sampling :when we add duplicate rows (increasing minority class) it is called over sampling 2) Under Sampling :when we delete the rows (reducing majority class) it is called as under sampling

```
#call class for over sampling and under sampling
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
```

(1) Over Sampling

```
#count yes and no begore applying sampling
pd.Series(Y_train).value_counts()
```

```
0    354
1    183
Name: Outcome, dtype: int64
```

```
#create object of RandomOverSampler
ros= RandomOverSampler()
```

```
#train the data using sampling
X_sample2,y_sample2 = ros.fit_sample(X_train,Y_train)
```

```
#count yes and no after applying sampling
pd.Series(y_sample2).value_counts()
```

```
1    354
0    354
dtype: int64
```

```
#now apply the Decision tree (of classification algorithm) using purning technique
dt4= DecisionTreeClassifier(max_depth=4)
```

```
#now train the model
dt4.fit(X_sample2,y_sample2)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
```

```
                              min_samples_leaf=1, min_samples_split=2,
                              min_weight_fraction_leaf=0.0, presort='deprecated',
                              random_state=None, splitter='best')
```

```
#now test the model
y_pred=dt4.predict(X_test)
```

```
#classification report
print(classification_report(Y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.80      0.81       146
           1       0.67      0.69      0.68        85

    accuracy                           0.76       231
   macro avg       0.74      0.75      0.75       231
weighted avg       0.76      0.76      0.76       231
```

(2) Under **sampling**

```
#create object of RandomUnderSampler
rus=RandomUnderSampler()
```

```
#train the data using sampling
X_sample1,y_sample1 = rus.fit_sample(X_train,Y_train)
```

```
#count yes and no after appyling sampling
pd.Series(y_sample1).value_counts()
```

```
1    183
0    183
dtype: int64
```

```
#now apply the Decision tree (of classification algorithm) using purning technique
dt5= DecisionTreeClassifier(max_depth=3)
```

```
#now train the model
dt5.fit(X_sample1,y_sample1)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
#now test the model
y_pred=dt5.predict(X_test)
```

```
y_pred=dt5.predict(X_test)

#classification report
print(classification_report(Y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.60      0.72       146
           1       0.57      0.91      0.70        85

    accuracy                           0.71       231
   macro avg       0.74      0.75      0.71       231
weighted avg       0.79      0.71      0.71       231
```

✓ 0s completed at 4:24 PM