```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
## Read dataset
df = pd.read_csv('diabetes.csv')
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
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	

```
import numpy as np

## converting glucose and Insulin features into one-hot-encoding
df['Glucose'] = np.where(df.Glucose == 0, df.Glucose.median(), df['Glucose'])
df['Insulin'] = np.where(df.Insulin == 0, df.Insulin.median(), df['Insulin'])
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
0	6	148.0	72	35	30.5	33.6	
1	1	85.0	66	29	30.5	26.6	
2	8	183.0	64	0	30.5	23.3	
3	1	89.0	66	23	94.0	28.1	
4	0	137.0	40	35	168.0	43.1	

```
## Independent And Dependent Features

X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

```
X.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148.0	72	35	30.5	33.6	
1	1	85.0	66	29	30.5	26.6	
2	8	183.0	64	0	30.5	23.3	
3	1	89.0	66	23	94.0	28.1	

```
## Labels

y.head()
```

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

```
## train test split

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=10).fit(X_train, y_train)
prediction = rf.predict(X_test)
```

```
y.value_counts()
```

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

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
print("<-------------------Confusion metrics results is ------------->\n : {}".format(confusi
print("<------------------Classification report is---------------> \n: {}".format(classificat
print("<------------------ Accuracy score----------------> : {}".format(accuracy_score(y_test
```

```
<-------------------Confusion metrics results is ------------->
 : [[84 15]
 [23 32]]
<------------------Classification report is--------------->
:               precision    recall  f1-score   support

           0       0.79      0.85      0.82        99
           1       0.68      0.58      0.63        55

    accuracy                           0.75       154
```

```
   macro avg       0.73      0.72      0.72       154
weighted avg       0.75      0.75      0.75       154

<------------------ Accuracy score----------------> : 0.7532467532467533
```

The main parameters used by a Random Forest

criterion = the function used to evaluate the quality of a split

- max_depth = maximum number of levels allowed in each tree
- max_features = maximum number of features considered when splitting a node.
- max_features = maximum number of features considered when splitting a node.
- min_samples_leaf = minimum number of samples which can be stored in a tree leaf.
- min_samples_split = minimum number of samples necessary in a node to cause node splitting.
- n_estimarors = number of trees in the ensemble.

```
## Manual HyperParameter Tunning
model = RandomForestClassifier(n_estimators=500, criterion='gini',
                               max_features='sqrt', min_samples_leaf=10, random_state=100).fi
prediction = model.predict(X_test)
print("<-------------------Confusion metrics results is ------------->\n : {}".format(confusi
print("<------------------Classification report is---------------> \n: {}".format(classificat
print("<------------------ Accuracy score----------------> : {}".format(accuracy_score(y_test
```

```
<-------------------Confusion metrics results is ------------->
 : [[83 16]
 [21 34]]
<------------------Classification report is--------------->
:               precision    recall  f1-score   support

           0       0.80      0.84      0.82        99
           1       0.68      0.62      0.65        55

    accuracy                           0.76       154
   macro avg       0.74      0.73      0.73       154
weighted avg       0.76      0.76      0.76       154

<------------------ Accuracy score----------------> : 0.7597402597402597
```

Randomized SearchCV

```
from sklearn.model_selection import RandomizedSearchCV
```

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start=200, stop=2000, num=10)]

# Number of featuers to consider at every split
max_features = ['auto', 'sqrt', 'log2']

# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 1000, 10)]

# Minimum number of samples required to split a node
min_samples_split = [1, 2, 3, 4, 5, 7, 9]

# Minimum number of samples required at each leaf node
min_samples_leaf =  [1, 2, 4, 6, 8]

# Create thre random grid
random_grid = {
    'n_estimators': n_estimators,
    'max_features': max_features,
    'max_depth': max_depth,
    'min_samples_split': min_samples_split,
    'min_samples_leaf': min_samples_leaf,
    'criterion': ['entropy', 'gini']
}

print(random_grid)
```

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000], 'max_features
```

```
rf = RandomForestClassifier()
rf_randomcv = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
                                n_iter=100, cv=3, verbose=2, random_state=100, n_jobs=-1)


## fit the randomized model
rf_randomcv.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'criterion': ['entropy', 'gini'],
                                        'max_depth': [10, 120, 230, 340, 450,
                                                      560, 670, 780, 890,
                                                      1000],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'min_samples_leaf': [1, 2, 4, 6, 8],
                                        'min_samples_split': [1, 2, 3, 4, 5, 7,
                                                              9],
                                        'n_estimators': [200, 400, 600, 800,
```

```
                                                                          1000, 1200, 1400, 1600,
                                                                          1800, 2000]},
                         random_state=100, verbose=2)
```

```
rf_randomcv.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 10,
 'max_features': 'auto',
 'min_samples_leaf': 2,
 'min_samples_split': 4,
 'n_estimators': 600}
```

```
randomcv_best_params = rf_randomcv.best_estimator_
```

```
y_pred = randomcv_best_params.predict(X_test)
print("<-------------------Confusion metrics results is ------------->\n : {}".format(confusi
print("<------------------Classification report is---------------> \n: {}".format(classificat
print("<------------------ Accuracy score----------------> : {}".format(accuracy_score(y_test
```

```
<-------------------Confusion metrics results is ------------->
 : [[79 20]
 [18 37]]
<------------------Classification report is--------------->
:               precision    recall  f1-score   support

           0       0.81      0.80      0.81        99
           1       0.65      0.67      0.66        55

    accuracy                           0.75       154
   macro avg       0.73      0.74      0.73       154
weighted avg       0.76      0.75      0.75       154

<------------------ Accuracy score----------------> : 0.7532467532467533
```

GridSearchCV

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'criterion': [rf_randomcv.best_params_['criterion']],
    'max_depth': [rf_randomcv.best_params_['max_depth']],
    'max_features': [rf_randomcv.best_params_['max_features']],
    'min_samples_leaf': [rf_randomcv.best_params_['min_samples_leaf'],
                         rf_randomcv.best_params_['min_samples_leaf'] + 2,
                         rf_randomcv.best_params_['min_samples_leaf'] + 4],
    'min_samples_split': [rf_randomcv.best_params_['min_samples_split'] - 2,
```

```
                          rf_randomcv.best_params_['min_samples_split'] - 1,
                          rf_randomcv.best_params_['min_samples_split'],
                          rf_randomcv.best_params_['min_samples_split'] + 1,
                          rf_randomcv.best_params_['min_samples_leaf'] + 2],
    'n_estimators': [rf_randomcv.best_params_['n_estimators'] - 200,
                     rf_randomcv.best_params_['n_estimators'] - 100,
                     rf_randomcv.best_params_['n_estimators'],
                     rf_randomcv.best_params_['n_estimators'] + 100,
                     rf_randomcv.best_params_['n_estimators'] + 200,
                     rf_randomcv.best_params_['n_estimators'] - 600,]
}

print(param_grid)
```

```
{'criterion': ['gini'], 'max_depth': [10], 'max_features': ['auto'], 'min_samples_leaf'
```

```
1 * 1 * 1 * 3 * 5 * 6
```

```
90
```

```
rf =  RandomForestClassifier()
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=10, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 90 candidates, totalling 900 fits
GridSearchCV(cv=10, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini'], 'max_depth': [10],
                         'max_features': ['auto'],
                         'min_samples_leaf': [2, 4, 6],
                         'min_samples_split': [2, 3, 4, 5, 4],
                         'n_estimators': [400, 500, 600, 700, 800, 0]},
             verbose=2)
```

```
grid_search.best_estimator_
```

```
RandomForestClassifier(max_depth=10, min_samples_leaf=2, min_samples_split=4,
                       n_estimators=400)
```

```
best_grid = grid_search.best_estimator_
```

```
y_pred = best_grid.predict(X_test)
print("<-------------------Confusion metrics results is ------------->\n : {}".format(confusi
print("<------------------Classification report is---------------> \n: {}".format(classificat
print("<------------------ Accuracy score----------------> : {}".format(accuracy_score(y_test
```

```
<-------------------Confusion metrics results is ------------->
 : [[79 20]
 [20 35]]
```

```
<------------------Classification report is--------------->
:               precision    recall  f1-score   support

           0       0.80      0.80      0.80        99
           1       0.64      0.64      0.64        55

    accuracy                           0.74       154
   macro avg       0.72      0.72      0.72       154
weighted avg       0.74      0.74      0.74       154

<------------------ Accuracy score----------------> : 0.7402597402597403
```

Automated Hyperparameter Tuning

Automated Hyperparameter Tuning can be done by using techniques such as

- Bayesian Optimization
- Gradient Descent
- Evolutionary Algorithms

Bayesian Optimization

It uses the probability to find the minimum of a function. The final aim to find the input value of a funciton which can gives us the lowest output value. It usually performs better than random grid and manual search providing better performance in the testing phase and reduced optimization time. In Hyperopt, Bayesian Optimization can be implemented giving 3 main parameters to the function fmin.

- Objective Function = defines the loss function to minimize
- Domain Space = define the range of input value of test (in Bayesian Optimization this space creates a probability distribution for each of the used Hyperparameters)
- Optimization Algorithm = defines the search algorithm to use to select the best input values to use in each new iteration.

```
from hyperopt import hp, fmin, tpe, STATUS_OK, Trials
## hp is used to define whether we are defining interger values, floating values, or choice f
space = {
    'criterion': hp.choice('criterion', ['entropy', 'gini']),
    'max_depth': hp.quniform('max_depth', 10, 1200, 10),
    'max_featuers': hp.choice('max_features', ['auto', 'sqrt', 'log2', None]),
    'min_samples_leaf': hp.uniform('min_samples_leaf', 0, 0.5),
    'min_samples_split': hp.uniform('min_samples_split', 0, 1),
    'n_estimators': hp.choice('n_estimators', [10, 50, 300, 750, 1200, 1300, 1500])
```

```
}
space
```

```
{'criterion': <hyperopt.pyll.base.Apply at 0x7f8fc87b8690>,
 'max_depth': <hyperopt.pyll.base.Apply at 0x7f8fc8758850>,
 'max_featuers': <hyperopt.pyll.base.Apply at 0x7f8fc8758a10>,
 'min_samples_leaf': <hyperopt.pyll.base.Apply at 0x7f8fc8758cd0>,
 'min_samples_split': <hyperopt.pyll.base.Apply at 0x7f8fc8758e50>,
 'n_estimators': <hyperopt.pyll.base.Apply at 0x7f8fc8758fd0>}
```

```
def objective(space):
    model = RandomForestClassifier(criterion=space['criterion'], max_depth=space['max_depth']
                                   max_features=space['max_featuers'], min_samples_leaf=space[
                                   n_estimators=space['n_estimators'])

    accuracy = cross_val_score(model, X_train, y_train, cv=5).mean()

    # We aim to maximize accuracy, therefore we return it as a negative value
    return {'loss': -accuracy, 'status': STATUS_OK}
```

```
from sklearn.model_selection import cross_val_score
trials = Trials() # it is responsible for minimizing the function
best = fmin(fn=objective,
            space=space,
            algo=tpe.suggest,
            max_evals=80,
            trials=trials)
best
```

```
100%|██████████| 80/80 [10:18<00:00,  7.73s/it, best loss: -0.7833666533386646]
{'criterion': 1,
 'max_depth': 480.0,
 'max_features': 3,
 'min_samples_leaf': 0.0017187628384621452,
 'min_samples_split': 0.9534800729041334,
 'n_estimators': 2}
```

```
crit = {0: 'entropy', 1: 'gini'}
feat = {0:'auto', 1: 'sqrt', 2: 'log2', 3: None}
est = {0:10, 1: 50, 2:300, 4: 1200, 5:1300, 6: 1500}

print(crit[best['criterion']])
print(feat[best['max_features']])
print(est[best['n_estimators']])
```

```
gini
None
300
```

```
trainedforest = RandomForestClassifier(criterion=crit[best['criterion']], max_depth=best['max
                                       max_features=feat[best['max_features']], min_samples_le
                                       min_samples_split=best['min_samples_split'], n_estimato
```

```
predictionforest = trainedforest.predict(X_test)
print("<------------------Confusion metrics results is ------------->\n : {}".format(confusi
print("<-----------------Classification report is---------------> \n: {}".format(classificat
print("<------------------ Accuracy score----------------> : {}".format(accuracy_score(y_test
```

```
<------------------Confusion metrics results is ------------->
 : [[99  0]
 [55  0]]
<-----------------Classification report is--------------->
:               precision    recall  f1-score   support

           0       0.64      1.00      0.78        99
           1       0.00      0.00      0.00        55

    accuracy                           0.64       154
   macro avg       0.32      0.50      0.39       154
weighted avg       0.41      0.64      0.50       154

<------------------ Accuracy score----------------> : 0.6428571428571429
```

✓ 0s completed at 13:35