Imports

```
import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from random import seed, randrange
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from scipy.stats import mode
from collections import Counter
from statistics import mean
```

Divide Into Test And Train Data

```
def divideTestAndTrain(dataset):
  return train_test_split(dataset, test_size=0.2)
```

با استفاده از تابع بالا، دیتاست را تبدیل به دو دیتاست ترین و تست می‌کنیم که به صورت ۸۰-۲۰ تقسیم شده اند.

Drop Label

```
def droplabel(dataset, lbl):
  labelFree = dataset.drop(lbl, axis=1)
  label = dataset[lbl]
  return labelFree, label
```

در این تابع لیبل را از دیتاست جدا می‌کنیم. درواقع ستونی که میخواهیم پیشبینی کنیم را لیبل و بقیه‌ی فیچرها را لیبل‌فری می‌نامیم و جدا می‌کنیم.

Making Decision Tree

```
def makeTree():
  classifier = DecisionTreeClassifier()
  return classifier
```

با استفاده از تابعی که در کتابخانه‌ها وجود دارد یک درخت تصمیم می‌سازیم و آن را برمی‌گردانیم.

Fit Data

```
def fitting(classifier, labelFree_train, label_train):
  classifier = classifier.fit(labelFree_train, label_train)
  return classifier
```

به تابع فیت، فیچرهای دیتاست ترین و لیبل دیتاست ترین را می‌دهیم و درخت را فیت می‌کنیم.

Prediction

```
def predict(classifier, labelFree_test):
  return classifier.predict(labelFree_test)
```

با استفاده از تابع پردیکت، فیچرهای دیتاست تست را می‌دهیم و لیبل را پیش‌بینی می‌کنیم.

Evaluate

```
def getAccuracy(label_test, label_pred):
  return accuracy_score(label_test, label_pred)
```

در این تابع می‌توانیم با استفاده از توابع موجود، اکیورسی (دقت) را محاسبه کنیم. این گونه که دیتاست پیش‌بینی شده و دیتاست لیبل اصلی را می‌دهیم و تابع، اکیورسی را خروجی می‌دهد.

Decision Tree

```
def DecisionTree(dataset, lbl):
  train, test = divideTestAndTrain(dataset)

  labelFree_train, label_train = droplabel(train, lbl)
  labelFree_test, label_test = droplabel(test, lbl)
  tree = makeTree()
  tree = fitting(tree, labelFree_train, label_train)
  pred = predict(tree, labelFree_test)
  return getAccuracy(label_test, pred)
```

در درخت تصمیم، ابتدا داده‌های ترین و تست را جدا می‌کنیم. برای هرکدام از داده‌های ترین و تست، لیبل و فیچرها را جدا می‌کنیم. سپس درخت را می‌سازیم و فیت می‌کنیم و پیشبینی می‌کنیم. در آخر هم دقت را برمی‌گردانیم.

Make Trees

```
def makeTrees(dataset, treeNum, sampleNum):
  trainDatas = []
  for i in range(treeNum):
    trainDatas.append(dataset.sample(n = sampleNum[i], replace=True))
  return trainDatas
```

در این تابع، به اندازه‌ی ورودی، دیتاست را تقسیم می‌کنیم. تعداد درخت و سایز هر درخت را می‌دهیم و دیتاست داده شده را با جاگذاری تقسیم می‌کنیم.

Voting

```
def vote(predictions):
  return mode(predictions, axis = 0)[0][0]
```

از جواب‌های پیشبینی تمام درخت‌ها را مود می‌گیرد و یک پیشبینی نهایی برمی‌گرداند. در واقع انگار که برای هر فیچر، رای‌گیری می‌کند و بیشترین تعداد برای یک فیچر خاص را، پیشبینی نهایی آن فیچر در نظر می‌گیرد.

Bagging

```
def Bagging(dataset, treeNum, sampleNum, lbl):
  predictions= []

  train, test = divideTestAndTrain(dataset)
  labelFree_test, label_test = droplabel(test, lbl)

  trainDatas = makeTrees(train, treeNum, sampleNum)

  for i in trainDatas:
    labelFree, label = droplabel(i, lbl)
    tree = makeTree()
    fittedTree = fitting(tree, labelFree, label)
    pred = predict(fittedTree, labelFree_test)
    predictions.append(pred)

  result = vote(predictions)
  return getAccuracy(label_test, result)
```

در این قسمت، بعد از جدا کردن دیتاست تست و ترین، دیتاهای ترین را تبدیل به ۵ دسته‌ی ۱۵۰تایی می‌کنیم. سپس برای هر کدام از دسته‌ها یک درخت تصمیم می‌سازیم و فیت می کنیم و پیشبینی می‌کنیم. در آخر هم رای گیری می‌کنیم و پیشبینی نهایی را برمی‌گردانیم.

Random Forest

```
def randomForest(dataset, treeNum, sampleNum, lbl):
  trainDatas = []
  seed(time.time())
  for i in range(treeNum):
    sampleNum.append(randrange(0,len(dataset)))

  train, test = divideTestAndTrain(dataset)
  labelFree_test, label_test = droplabel(test, lbl)

  trainDatas = makeTrees(train, treeNum, sampleNum)

  seed(time.time())
  predictions = []
  for i in trainDatas:
    featureNum = randrange(1,13)
    # print(featureNum)
```

```
    selectedFeatures = []
    datasetTemp = i
    j = 0
    while(j < featureNum):
    # for j in range(featureNum):
      randFeature = datasetTemp.columns[randrange(0,len(datasetTemp.columns))]
      if randFeature == lbl:
        # print(j)
        continue
      selectedFeatures.append(randFeature)
      datasetTemp = datasetTemp.drop(randFeature, axis=1)
      j += 1

    selectedFeatures.append(lbl)
    data = i[selectedFeatures]

    labelFree, label = droplabel(data, lbl)
    tree = makeTree()
    fittedTree = fitting(tree, labelFree, label)
    selectedFeatures.remove(lbl)
    pred = predict(fittedTree, labelFree_test[selectedFeatures])
    predictions.append(pred)

  result = vote(predictions)
  return getAccuracy(label_test, result)
```

در جنگل رندوم، دیتاست اولیه را تبدیل به ترین و تست می‌کنیم. سپس دیتاست ترین را تبدیل به دسته‌هایی می‌کنیم که تعداد دسته‌ها رندوم است. سپس هرکدام از دسته‌ها را برمی‌داریم و تعدادی رندوم از فیچرهایش را نگه می‌داریم. سپس از روی این دیتاست‌هایی که هم تعداد سطرها و هم تعداد ستون‌هایشان فرق دارد و به صورت رندوم انتخاب شده است، درخت تصمیم می‌سازیم. سپس بعد از فیت کردن، پیشبینی می‌کنیم و رای می‌گیریم و پیش‌بینی نهایی را به دست می‌آوریم. سپس دقت را برمی‌گردانیم.

Main

```
treeNum = 5
sampleNum = []
lbl = 'target'
dataset = pd.read_csv("data.csv")

# 1
print("2.1")
result = []
for i in range(100):
  acc = DecisionTree(dataset, lbl)
  result.append(acc)
result = np.array(result)
print("Decision Tree: ", result.mean())

# 2
print("2.2")
for i in range(treeNum):
  sampleNum.append(150)
result = []
for i in range(100):
  acc = Bagging(dataset, treeNum, sampleNum, lbl)
  result.append(acc)
result = np.array(result)
print("Bagging: ", result.mean())

# 3
print("2.3")
for col in dataset.columns:
  result = []
  if col == 'target':
    continue
  data = dataset.drop(col, axis=1)
  for i in range(100):
    acc = DecisionTree(data, lbl)
    result.append(acc)
  result = np.array(result)
  print(col, result.mean())

# 4
print("2.4")
seed(time.time())
selectedFeatures = []
datasetTemp = dataset
datasetTemp = datasetTemp.drop('target', axis=1)
for i in range(5):
  randFeature = datasetTemp.columns[randrange(0,len(datasetTemp.columns))]
  selectedFeatures.append(randFeature)
  datasetTemp = datasetTemp.drop(randFeature, axis=1)
selectedFeatures.append('target')
```

```
# print(dataset[selectedFeatures])
print(DecisionTree(dataset[selectedFeatures], lbl))

# 5
print("2.5")
result = []
for i in range(100):
  acc = randomForest(dataset, treeNum, sampleNum, lbl)
  result.append(acc)
result = np.array(result)
print("Random Forest: ", result.mean())
```

```
2.1
Decision Tree:  0.7496721311475407
2.2
Bagging:  0.777049180327869
2.3
age 0.7509836065573771
sex 0.7368852459016394
cp 0.7272131147540984
trestbps 0.7596721311475408
chol 0.7575409836065574
fbs 0.7575409836065573
restecg 0.7486885245901638
thalach 0.761967213114754
exang 0.7537704918032786
oldpeak 0.7468852459016392
slope 0.750327868852459
ca 0.7180327868852459
thal 0.7434426229508196
2.4
0.7213114754098361
2.5
Random Forest:  0.7678688524590165
```

1. همچنین کمک می‌کند که درخت اورفیت نشود. بیشتر اوقات روی درخت تصمیم پیاده‌سازی می‌شود. اما می‌توان روی الگوریتم‌های دیگر هم پیاده‌سازی کرد. بوت استرپینگ این گونه است که زمانی که داده‌های ترین را تقسیم می‌کنیم و می‌خواهیم تعدادی از داده‌ها را انتخاب کنیم، این کار را با جایگذاری انجام می‌دهیم. به این معنی که بعضی از داده‌ها می‌توانند دوبار در دیتا تکرار شوند. به دلیل رندوم بودن، واریانس و انحراف معیار را افزایش می‌دهد.

2. اورفیتینگ به این معنی است که پیشبینی نهایی ما خیلی به دیتاست ترین نزدیک شده باشد. یعنی پیچیدگی مسئله رو اینقدر بالا برده باشیم که داده‌ی ترین ما را خیلی خوب پیش‌بینی کند. اما اگر داده‌ی تست را به آن بدهیم، دقت آن پایین باشد. در واقع یادگیری ما بیش از اندازه فیت ترین دیتای ما شده است. درخت تصمیم به اورفیتینگ حساس است زیرا درواقع داده‌ی ترین را نمی‌بیند و یادگیری آن به صورت کامل از فیت کردن صورت می‌گیرد. بنابراین درخت تصمیم به اورفیتینگ حساس است. در بگینگ دیتاها را به دسته‌هایی تقسیم می‌کنیم. اینگونه، درخت‌ها به کل دیتا اورفیت نمی‌شوند. اما به بخشی از دیتا اورفیت ممکن از بشوند.

3. رندوم فارست تلاش می‌کند که مسئله‌ی اورفیتینگی که در بگینگ رخ می‌دهد را حل کند. تفاوت بین این دو هم این است که در بگینگ تمام فیچرها را برای پیش بینی در نظر می‌گیریم. و فیچرهای درخت‌ها یکسان هستند که این ممکن است باعث ایجاد اورفیتینگ شود. اما در رندوم فارست برای اینکه مسئله‌ی اورفیت رخ ندهد و هر دفعه سعی کنیم بهترین فیچرها را انتخاب کند و اینکه در هر درخت فیچرهایی که انتخاب می‌کند متفاوت و به صورت رندوم است.

4. میانگین دقت سه روش درخت تصمیم و بگینگ و رندوم فارست به ترتیب: Decision Tree: 0.7491803278688525 Bagging: 0.7786885245901638 Random Forest: 0.7647540983606557 از این دیتاها می‌توان فهمید که هرچه‌قدر درخت‌های بیشتر با فیچرهای رندوم داشته باشیم، دقت مدل ما بالاتر می‌رود.