KOD BLOKLARI

```
if x==y:
    print("___", x, "=" y)
else:
```

```
for k in range(5,10):
    print("___", k)
```

```
while a<=10:
    print(a)
```

```
def myfunction (var1, var2, cookie):
    print ("___", cookie)
    return var1* var2 +cookie
```

```
array = [0,1,2,3]        → this is array
np_array = np.array([0,1,2])   → this is numpy array
```

- Dataset okuma -

```
my_url = "_____"
my_dataset = pd.read_csv(my_url)
```

Çıktının columnlarına isim vermek için

```
name_c=["x1", "x2" ---]
my_dataset = pd.read_csv( my_url , names = name_c.)
print ( my_dataset)
```

iloc ile input - output ayırdık (satır, sütun)

```
my_input = my_dataset.iloc[0:150, 0:4]
my_output = my_dataset.iloc[0:150, 4:5]
```

→ Önceden tanımlanmış my_output var fakat bu my_output sütun vektörü şeklinde bunu kullanırken hata alıyorum o bunu arraye çevirip öyle kullanmam lazım

```
my_output_array = np.ravel(my_output)
```

ravel ile array'e çevrilmiş

```
my_iris.fit(train_input, train_output_array)
print("Expected", test_output_array)
print("Calculated", my_iris.predict(test_input))
```

Datayı Random ayırmak için.

```
X_train, X_test, y_train, y_test = train_test_split(
        my_dataset.iloc[0:150, 0:4],
        my_dataset.iloc[0:150, 4:6],
        test_size = 0.25)
```

Uygulamanın içindeki veri setlerini kullanarak data çekerken

```
myirisdata = load_iris()
X = myirisdata['data']
Y = myirisdata['target']
```

- Görselleştirmek için Confus. Matrixi

```
cm = confusion_matrix(Y_test, Y_test_predicted)
disp = ConfusionMatrixDisplay(confusion_matrix = cm)
disp.plot()
plt.show()
```

How to add library -

```
import pandas as py
import numpy as np

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import matplotlib.pyplot as myplotter


from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_fscore_support

from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report

from sklearn.svm import SVC

from google.colab import drive

from sklearn.model_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import VotingClassifier

from sklearn.ensemble import GradientBoostingClassifier
```

KNN ALGORITHM

```
from sklearn.neighbors import KNeighborsClassifier

input = [[0],[1],[2],[3]]
output = [0,0,1,1]

my_blackbox = KNeighborsClassifier(n_neighbors=1,
                                   weights="uniform")
my_blackbox.fit(input,output)

print(my_blackbox.predict([[1.6]]))  # 1.6'nın nerede old. tahmin edecek.

print(my_blackbox.predict_proba([[1.6]])) # 1.6'nın sınıflara ait olma
                                          ihtimalini verir.
```

→ n_neighbors = n → n tane komşuya bakacağını söyler

→ weights = "uniform" → herkesin eşit oy hakkı vardır.

= "distance" → aralarındaki uzaklık hesaba katılır.

DECISION TREE ALGORITHM

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import matplotlib.pyplot as myplotter

myirisdata = load_iris()
X = myirisdata['data']
Y = myirisdata['target']

mymachine = DecisionTreeClassifier(criterion = "gini",
                                   max_depth = 20)
mymachine.fit(X,Y)

myplotter.figure(figsize = (17,17))
plot_tree = (mymachine, filled = True)
myplotter.show()
```

→ criterion = "gini" →

= "entropy" →

→ max_depth = 20 → her zaman böyle kullanabilirsin

CONFUSION MATRIX

with KNN

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_fscore_support

myirisdata = load_iris()
X = myirisdata['data']
Y = myirisdata['target']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25)
mymachine = KNeighborsClassifier(n_neighbors=5, weights='uniform')
mymachine.fit(X_train, Y_train)

Y_test_predicted= mymachine.predict(X_test)

print(confusion_matrix(Y_test, Y_test_predicted))
print(precision_recall_fscore_support(Y_test, Y_test_predicted))
```

with DT

```
from sklearn.tree import DecisionTreeClassifier

mymachine = DecisionTreeClassifier(criterion="gini", max_depth=20)
mymachine.fit(X_train, Y_train)

Y_test_predicted = mymachine.predict(X_test)

print(confusion_matrix(Y_test, Y_test_predicted))
print(precision_recall_fscore_support(Y_test, Y_test_predicted))
```

Confusion Matrixi görselleştirmek için ↙

```
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(Y_test, Y_test_predicted)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```

NORMALIZATION

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.datasets import load_iris

mydata = load_iris()
X = mydata['data']
Y = mydata['target']

mystandard = StandardScaler()
mystandard.fit(X)
X1 = mystandard.transform(X)

minmax = MinMaxScaler()
minmax.fit(X)
X2 = minmax.transform(X)

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)
X1_ , X1_ , Y1_ , Y1_ = " (X1,Y, " )
X2_ , X2_ , Y2_ , Y2_ = " (X2,Y, " )

from sklearn.neighbors import KNeighborsClassifier
myknn = KNeighborsClassifier(n_neighbors=5, weights='uniform')
myknn1 = "
myknn2 = "

myknn.fit(X_train, Y_train)
myknn1.fit(X1_train, Y_train)
myknn2.fit(X2_train, Y_train)

Y_test_calculated = myknn.predicted(X_test)
Y1_test_ " = " " (X1_test)
Y2_test_ " = " " (X2_test)

print('Orijinal Data')
print(confusion_matrix(Y_test, Y_test_calculated)
print(classification_report(Y_test, Y_test_calculated)
```

Normalize için:
Sadece inputlar normalize edilir, outputlar edilmez

→ Bu üçlüyü standard Scaler → Y1
Minmax Scaler → Y2 için de yaz.

SUPPORT VECTOR CLASSIFIER

```
msvc = SVC((C=10.0, kernel='linear')
msvc = SVC((C=10.0, kernel='poly', degree=5)

mysvc.fit(X1_train, Y1_train)
Y1_test_calculate = mysvc.predict(X1_test)

print(confusion_matrix(Y1_test, Y1_test_calculate))
print(classification_report(Y1_test, Y1_test_calculate))
```

→ C = n → C=margin , n genelde 10 alınıyor

→ kernel = 'linear' →

= 'poly' → degree = n eklenir.

= 'rbf' →

= 'sigmoid' →

```
from sklearn.ensemble import RandomForestClasifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
import numpy as np
from sklearn.model_selection import train_test_split
from google.colab import drive                    } drive'dan dosya secebilmek
drive.mount('/content/gdrive')                    } için.

import pandas as pd
mydata = pd.read_csv(" ——— .csv")
mydata.head()

X = mydata.drop(columns = ['Outcome'])
y = mydata['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

knn

```
knn = KNeighborsClasifier
params_knn = {'n_neighbors': np.arrange (1,25)}
knn_gs = GridSearchCV (knn, params_knn, cv=5)
knn_gs.fit (X_train, y_train)
knn_best = knn_gs.best_estimator_
print(knn_gs.best_params_)
```

randomforest

```
rf = RandomForestClassifier()
params_rf = {'n_estimators': [50,100,200]}
rf_gs = GridSearchCV (rf, params_rf, cv=5)
rf_gs.fit (X_train, y_train)
rf_best = rf_gs.best_estimator_
print(rf_gs.best_params_)
```

Logistic regression

```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(max_iter= 1000)
log_reg.fit (X_train, y_train)
```

boosting

```
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier()
params_gbc = {'n_estimators': [50,100,200]}
gbc_gs = GridSearchCV (gbc, params_gbc, cv=5)
gbc_gs.fit( X_train, y_train)
gbc_best = gbc_gs.best_estimator_
print (gbc_gs.best_params_)
```

↓

#en iyi knn score 'a baktım, knn'de en iyi knn'i buldum.

```
print('knn', knn_best.score(X_test,y_test))
print('rf', rf_best.score(X_test, y_test))
print('log_reg', log_reg.score(X_test, y_test))
print('gbc', gbc_best.score(X_test,y_test))

from sklearn.ensemble import VotingClassifier

estimators=[('knn', knn_best), ('rf', rf_best),
                ('log_reg', log_reg), ('gbc', gbc_best)]
ensemble = VotingClassifier(estimators, voting='hard')
ensemble.fit(X_train, y_train)
print(ensemble.score(X_test, y_test))
```