**LAB4\_Linear\_Regression**

**Source Code**

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn import preprocessing

from sklearn.model\_selection import train\_test\_split

# read data

df = pd.read\_csv('housing.csv')

# fill missing value

df.fillna(method='ffill', inplace=True)

# set independent, target

y = df['median\_house\_value']

X = df.drop(['median\_house\_value'], axis=1)

# encoding categorical data

ordinal = preprocessing.OrdinalEncoder()

ocean = pd.DataFrame(df['ocean\_proximity'])

ordinal.fit(ocean)

X['ocean\_proximity'] = pd.DataFrame(ordinal.transform(ocean))

# standard scaling

stdscaler = preprocessing.StandardScaler()

stdscaler.fit(X)

X = stdscaler.transform(X)

# split data 4:1

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, test\_size=0.2, random\_state=1)

# set regression model

LR = LinearRegression()

LR. fit(X\_train,y\_train)

# predict value and accuracy

predict = pd.DataFrame(LR.predict(X\_test))

accuracy = LR.score(X\_test,y\_test)

# output

print("\n\nSplit data 4:1")

print("\n Test model result")

print(predict)

print("\n Original data result")

print(y\_test)

print("\nAccuracy : "+str(accuracy))

# split data 3:2

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.6, test\_size=0.4, random\_state=1)

# set regression model

LR = LinearRegression()

LR. fit(X\_train,y\_train)

# predict value and accuracy

predict = pd.DataFrame(LR.predict(X\_test))

accuracy = LR.score(X\_test,y\_test)

# output

print("\n\nSplit data 3:2")

print("\n Test model result")

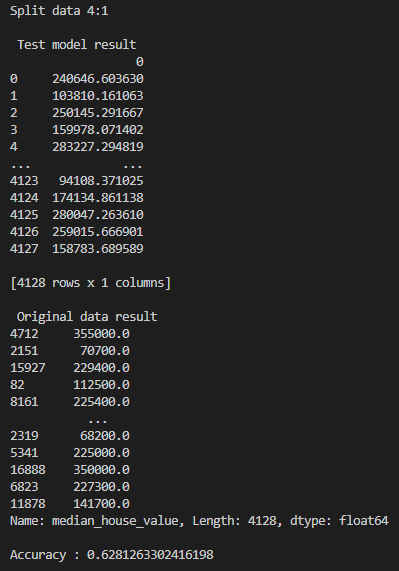
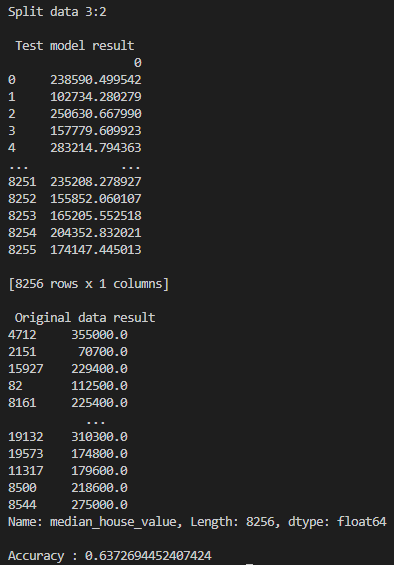
print(predict)

print("\n Original data result")

print(y\_test)

print("\nAccuracy : "+str(accuracy))

**Output Screen Capture**

**LAB4\_Decision\_Tree**

**Source Code**

import numpy as np

import pandas as pd

from sklearn import preprocessing

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from sklearn.tree import DecisionTreeClassifier

# read data

df = pd.read\_csv('winequality-red.csv',index\_col = None, sep=';')

# check dirty data

print("\nCheck dirty data")

print(df.isnull().sum())

print("\n\n")

# result : no dirty data

# set independent, target

y = df['quality']

X = df.drop(['quality'], axis=1)

# standard scaling

stdscaler = preprocessing.StandardScaler()

stdscaler.fit(X)

X=stdscaler.transform(X)

# split data 9:1

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.9, test\_size=0.1, shuffle=True, random\_state=1)

# decision tree setting

dt = DecisionTreeClassifier()

dt.fit(X\_train,y\_train)

# predcit value and accuracy

predict = pd.DataFrame(dt.predict(X\_test))

accuracy = dt.score(X\_test,y\_test)

# output

print("Split data 9:1")

print("\n Test model result")

print(predict)

print("\n Original data result")

print(y\_test)

print("\nAccuracy : "+str(accuracy))

# split data 8:2

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, test\_size=0.2, shuffle=True, random\_state=1)

# decision tree setting

dt = DecisionTreeClassifier()

dt.fit(X\_train,y\_train)

# predcit value and accuracy

predict = pd.DataFrame(dt.predict(X\_test))

accuracy = dt.score(X\_test,y\_test)

# output

print("Split data 8:2")

print("\n Test model result")

print(predict)

print("\n Original data result")

print(y\_test)

print("\nAccuracy : "+str(accuracy))

# split data 7:3

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.7, test\_size=0.3, shuffle=True, random\_state=1)

# decision tree setting

dt = DecisionTreeClassifier()

dt.fit(X\_train,y\_train)

# predcit value and accuracy

predict = pd.DataFrame(dt.predict(X\_test))

accuracy = dt.score(X\_test,y\_test)

# output

print("\n\nSplit data 7:3")

print("\n Test model result")

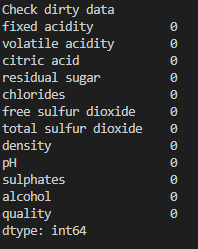
print(predict)

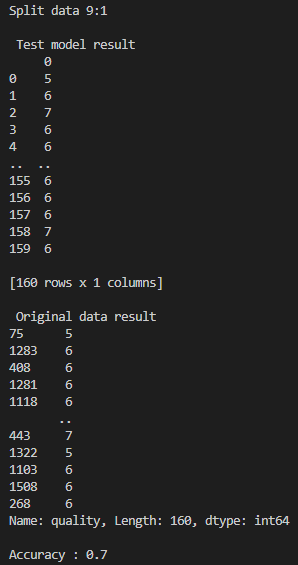
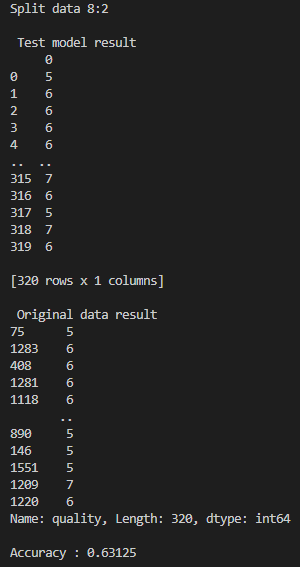
print("\n Original data result")

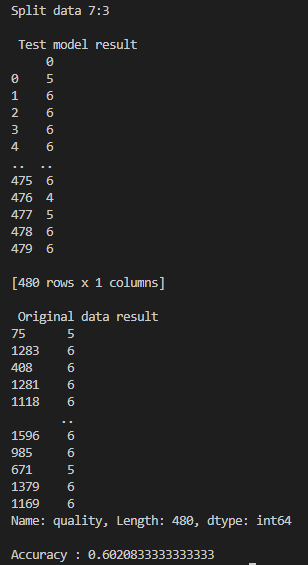
print(y\_test)

print("\nAccuracy : "+str(accuracy))

**Output Screen Capture**





**LAB4\_KNN**

**Source Code**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn import preprocessing

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import KFold

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import RandomizedSearchCV

# read data / use mnist\_test because of laptop performance

df = pd.read\_csv('mnist\_test.csv')

# check dirty data

print("\nCheck dirty data")

print(df.isnull().sum())

print("\n\n")

# result : no dirty data

# set independent, target

y = df['label']

X = df.drop(['label'], axis=1)

# standard scaling

stdscaler = preprocessing.StandardScaler()

stdscaler.fit(X)

X=stdscaler.transform(X)

# cross validation

kf5 = KFold(n\_splits=5, shuffle = True, random\_state=1)

# case : K = 3

for k in [3,5]:

    i=1

    for train, test in kf5.split(X):

        print("Split " + str(i) + "\n")

        i+=1

        X\_train, X\_test = X[train], X[test]

        y\_train, y\_test = y[train], y[test]

        knn\_cv = KNeighborsClassifier(n\_neighbors=k)

        scores = cross\_val\_score(knn\_cv,X\_train,y\_train,cv=5)

        print("Case : initial K = "+str(k))

        print(scores)

        print("scores mean : {}".format(np.mean(scores)))

        knn = KNeighborsClassifier()

        param = {'n\_neighbors':np.arange(1,25)}

        knn\_gs = GridSearchCV(knn, param\_grid=param, cv=5)

        knn\_gs.fit(X\_train,y\_train)

        knn\_rgs = RandomizedSearchCV(knn, param, cv=5, scoring='accuracy')

        knn\_rgs.fit(X\_train,y\_train)

        print("\nCase : Hyperparameter tuning by GridSearch")

        print(knn\_gs.best\_params\_)

        print(knn\_gs.best\_score\_)

        print("\nCase : Hyperparameter tuning by Randomized GridSearch")

        print(knn\_rgs.best\_params\_)

        print(knn\_rgs.best\_score\_)

        print("\n")

**Output Screen Capture**

