```
# import numpy as np
# import pandas as pd

# import matplotlib.pyplot as plt

# import sklearn.neural_network

# xs=np.array([
#     0,0,
#     0,1,
#     1,0,
#     1,1
# ]).reshape(4,2)
# ys=np.array([0,1,1,0])#([0, 1, 1, 0])
# ys.reshape(4,)
#
model=sklearn.neural_network.MLPClassifier(activation="relu",max_iter=
10000,learning_rate="constant",hidden_layer_sizes=(4)) #hidden layer
akta and neuron 3 ta aro ->hidden_layer_sizes=(2, 2, 4))#3 hidden
layer and first layer and second has 2 neurons and third has 4 neurons
.
# model.fit(xs,ys)


# import matplotlib.pyplot as plt
# plt.scatter(xs[:,0], xs[:,1], c=ys, cmap=plt.cm.brg)
# plt.show()

# print("score",model.score(xs,ys))
# print("prediction",model.predict(xs))
# print ("expected",np.array([0,1,1,0]))
# xs

# import numpy as np
# import pandas as pd
# import matplotlib.pyplot as plt
# from sklearn.datasets import load_iris
# from sklearn.model_selection import train_test_split



import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from ucimlrepo import fetch_ucirepo

# fetch dataset
heart_disease = fetch_ucirepo(id=45)

# data (as pandas dataframes)
X = heart_disease.data.features
```

```
y = heart_disease.data.targets

# metadata
# print(heart_disease.metadata)

# # variable information
# print(heart_disease.variables)
X.shape
y.shape

X.shape

y.shape

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def mean_squared_error(y_pred, y_true):
    return ((y_pred - y_true) ** 2).sum() / (2 * y_pred.size)

def accuracy(y_pred, y_true):
    if isinstance(y_true, pd.DataFrame):
        y_true = y_true.values

    acc = y_pred.argmax(axis=1) == y_true.argmax(axis=1)
    return acc.mean()

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=20, random_state=4)
# y_train = np.reshape(y_train, (y_train.shape[0], 1))
# y_test = np.reshape(y_test, (y_test.shape[0], 1))

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=20, random_state=4)

learning_rate = 0.1
iterations = 10000
N = y_train.size

input_size = 13
hidden_size = 2
output_size = 1
results = pd.DataFrame(columns=['mse', 'accuracy'])

np.random.seed(10)

W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size))
```

```
W2 = np.random.normal(scale=0.5, size=(hidden_size, output_size))




results_list = []

for itr in range(iterations):

    Z1 = np.dot(X_train, W1)
    A1 = sigmoid(Z1)

    Z2 = np.dot(A1, W2)
    A2 = sigmoid(Z2)


    mse = mean_squared_error(A2, y_train)
    # acc = accuracy(A2.values, y_train.values)
    acc = accuracy(A2, y_train)
    results_list.append({"mse": mse, "accuracy": acc})


    E1 = A2 - y_train
    dW1 = E1 * A2 * (1 - A2)
    E2 = np.dot(dW1, W2.T)
    dW2 = E2 * A1 * (1 - A1)


    w2_update = np.dot(A1.T, dW1) / N
    w1_update = np.dot(X_train.T, dW2) / N

    W2 = W2 - learning_rate * w2_update
    W1 = W1 - learning_rate * w1_update

Z1 = np.dot(X_test, W1)
A1 = sigmoid(Z1)
Z2 = np.dot(A1, W2)
A2 = sigmoid(Z2)

acc = accuracy(A2, y_test)
print("Accuracy is {}".format(acc))
```

```
---------------------------------------------------------------------------
-----
ModuleNotFoundError                       Traceback (most recent call
last)
<ipython-input-1-b0a2d226f21b> in <cell line: 4>()
      2 import pandas as pd
```

```
      3 import matplotlib.pyplot as plt
----> 4 from ucimlrepo import fetch_ucirepo
      5
      6 # fetch dataset

ModuleNotFoundError: No module named 'ucimlrepo'

---------------------------------------------------------------------------
-----
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
-----
```

```
#  x = pd.DataFrame(iris.data , columns=['SL','SW','PL','PW'])
#  y = pd.DataFrame(iris.target , columns=['Target'])
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib inline

df=pd.read_csv('/content/drive/MyDrive/versity/Thesis/data/diabetes.cs
v')

df.head(10)

df.isnull().sum()

#display dataset randomly
df.sample(10)

#shape of the dataset
df.shape


df.columns


"""# Split the data frame into X and y"""

target_name='Outcome'
y=df['Outcome']
X=df.drop(target_name,axis=1)
```

```
y

X.head()

"""# Feature scaling tecniques

"""
#standard scaler
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X)
SSX=scaler.transform(X)
SSX
#scaler = StandardScaler()
#scaler.fit(X_train)
#X_train = scaler.transform(X_train)
#X_test = scaler.transform(X_test)
"""# Train Test split

"""

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(SSX,y,test_size=0.25,ra
ndom_state=42)
X_train.shape,y_train.shape

X_test.shape,y_test.shape


#Linear Regression


from sklearn.linear_model import LinearRegression
Linearregression=LinearRegression()
Linearregression.fit(X_train,y_train)

linear_predict=Linearregression.predict(X_test)
df=pd.DataFrame({'Actual':y_test,'Predicted':linear_predict}).round(0)
df

#check accuracy mean
from sklearn.metrics import mean_absolute_error
mae_for_linear=mean_absolute_error(y_test,linear_predict)
print(f"mean absolute error {mae_for_linear}")

# from sklearn.metrics import accuracy_score
# accuracy = accuracy_score(y_test , linear_predict)

# Logistic Regression
```

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(X_train,y_train)
lr_predict=lr.predict(X_test)
df=pd.DataFrame({'Actual':y_test,'Predicted':lr_predict}).round(0)
df

from sklearn.metrics import mean_absolute_error
mae_for_log=mean_absolute_error(y_test,lr_predict)
print(f"mean absolute error {mae_for_log}")

# KNN Classification

from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

range_k = range(1,15)
scores = {}
scores_list = []
for k in range_k:
  classifier=KNeighborsClassifier(n_neighbors=k)
  classifier.fit(X_train,y_train)
  knn_predict=classifier.predict(X_test)
  scores[k]=metrics.accuracy_score(y_test,knn_predict)
  scores_list.append(metrics.accuracy_score(y_test,knn_predict))

result=metrics.confusion_matrix(y_test,knn_predict)
print("Confusion Matrix:")
print(result)
result1 = metrics.classification_report(y_test , knn_predict)
print("Classification Report:",)

"""**Now, we will be plotting the relationship between the values of K
and the corresponding testing
accuracy. It will be done using matplotlib library.**
"""

plt.plot(range_k,scores_list)
plt.xlabel("Values of K")
plt.ylabel("Accuracy")

classifier = KNeighborsClassifier(n_neighbors = 2)
classifier.fit(X_train , y_train)

knn_predict=classifier.predict(X_test)
df=pd.DataFrame({'Actual':y_test,'Predicted':knn_predict}).round(0)
df

from sklearn.metrics import mean_absolute_error
knn_predict_mean=mean_absolute_error(y_test,knn_predict)
```

```
print(f"mean absolute error{knn_predict_mean}")

#naive bayes
from sklearn.naive_bayes import GaussianNB
nb=GaussianNB()
nb.fit(X_train,y_train)

nb_predict=nb.predict(X_test)
df=pd.DataFrame({'Actual':y_test,'Predicted':nb_predict}).round(0)
df

from sklearn.metrics import mean_absolute_error
nb_predict_mean=mean_absolute_error(y_test,nb_predict)
print(f"mean absolute error{nb_predict_mean}")

import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import sklearn.metrics as sm
%matplotlib inline
iris = datasets.load_iris()
print(iris.target_names)
x = pd.DataFrame(iris.data , columns=['SL','SW','PL','PW'])
y = pd.DataFrame(iris.target , columns=['Target'])
iris_k_mean_model = KMeans(n_clusters =3) # we set K = 3
iris_k_mean_model.fit(x)

predictedY=iris_k_mean_model.predict(x)
sm.accuracy_score(predictedY , y['Target'])
sm.confusion_matrix(predictedY , y['Target'])
```