|  |
| --- |
| **Name: MALAVIKA A**  **Roll No: 16**  **Batch: RMCA B**  **Date: 12-09-2022** |

**DATA SCIENCE LAB**

**Experiment No.: 1**

**Aim**

Zscores

**Procedure**

1. **Zscore(data)**

import pandas as pd

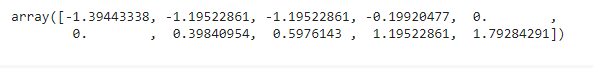
import numpy as np

import scipy.stats as stats

data = np.array([6, 7, 7, 12, 13, 13, 15, 16, 19, 22])

stats.zscore(data)

**OUTPUT**



1. **Zscore(axis)**

import pandas as pd

import numpy as np

import scipy.stats as stats

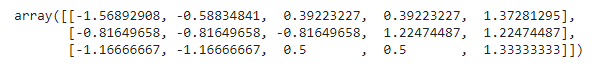
data = np.array([[5, 6, 7, 7, 8],

                 [8, 8, 8, 9, 9],

                 [2, 2, 4, 4, 5]])

stats.zscore(data, axis=1)

**OUTPUT**



1. **Apply()**

import pandas as pd

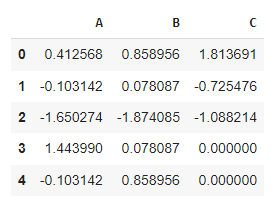
import numpy as np

import scipy.stats as stats

data = pd.DataFrame(np.random.randint(0, 10, size=(5, 3)), columns=['A', 'B', 'C'])

data.apply(stats.zscore)

**OUTPUT**



1. **Scaler.transform()**

from sklearn.preprocessing import MinMaxScaler

data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]

scaler = MinMaxScaler()

print(scaler.fit(data))

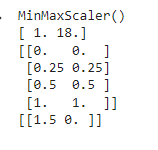
MinMaxScaler()

print(scaler.data\_max\_)

print(scaler.transform(data))

print(scaler.transform([[2, 2]]))

**OUTPUT**



1. **RandomForestClassifier()**

import numpy as np

from scipy.stats import randint

from sklearn.experimental import enable\_halving\_search\_cv  # noqa

from sklearn.model\_selection import HalvingRandomSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import make\_classification

rng = np.random.RandomState(0)

X, y = make\_classification(n\_samples=700, random\_state=rng)

clf = RandomForestClassifier(n\_estimators=10, random\_state=rng)

param\_dist = {

    "max\_depth": [3, None],

    "max\_features": randint(1, 11),

    "min\_samples\_split": randint(2, 11),

    "bootstrap": [True, False],

    "criterion": ["gini", "entropy"],

}

rsh = HalvingRandomSearchCV(

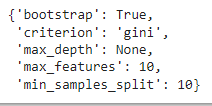
    estimator=clf, param\_distributions=param\_dist, factor=2, random\_state=rng

)

rsh.fit(X, y)

rsh.best\_params\_

**OUTPUT**



1. **SelfTrainingClassifier**

import numpy as np

from sklearn import datasets

from sklearn.semi\_supervised import SelfTrainingClassifier

from sklearn.svm import SVC

rng = np.random.RandomState(42)

iris = datasets.load\_iris()

random\_unlabeled\_points = rng.rand(iris.target.shape[0]) < 0.3

iris.target[random\_unlabeled\_points] = -1

svc = SVC(probability=True, gamma="auto")

self\_training\_model = SelfTrainingClassifier(svc)

self\_training\_model.fit(iris.data, iris.target)

**OUTPUT**



1. **suptitle**

from sklearn.ensemble import RandomForestRegressor

from sklearn.datasets import fetch\_california\_housing

from sklearn.inspection import plot\_partial\_dependence

X, y = fetch\_california\_housing(return\_X\_y=True, as\_frame=True)

features = ["MedInc", "AveOccup", "HouseAge", "AveRooms"]

est = RandomForestRegressor(n\_estimators=10)

est.fit(X, y)

display = plot\_partial\_dependence(

    est,

    X,

    features,

    kind="individual",

    subsample=50,

    n\_jobs=3,

    grid\_resolution=20,

    random\_state=0,

)

display.figure\_.suptitle(

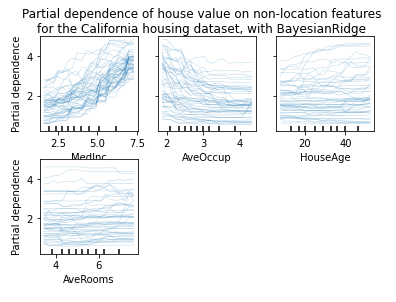
    "Partial dependence of house value on non-location features\n"

    "for the California housing dataset, with BayesianRidge"

)

display.figure\_.subplots\_adjust(hspace=0.3)

**OUTPUT**



1. **load\_iris**

from sklearn.datasets import load\_iris

from sklearn.preprocessing import MinMaxScaler

import numpy as np

X, y = load\_iris(return\_X\_y=True)

print(X.shape)

**OUTPUT**



1. **x\_scaled.min**

from sklearn.datasets import load\_iris

from sklearn.preprocessing import MinMaxScaler

import numpy as np

X\_scaled.min(axis=0)

**OUTPUT**



1. **x\_scaled.max**

from sklearn.datasets import load\_iris

from sklearn.preprocessing import MinMaxScaler

import numpy as np

X\_scaled.max(axis=0)

**OUTPUT**



1. **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=0.2, random\_state=42, stratify = y)

**OUTPUT**



1. **KNeighborsClassifier**

from sklearn.neighbors import KNeighborsClassifier

model=KNeighborsClassifier(n\_neighbors=5,metric='minkowski')

model.fit(X\_train, y\_train)

y\_pred=model.predict(X\_test)

model.score(X\_test, y\_test)

**OUTPUT**



1. **MinMaxScaler**

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

fig, axes = plt.subplots(1,2)

axes[0].scatter(X[:,0], X[:,1], c=y)

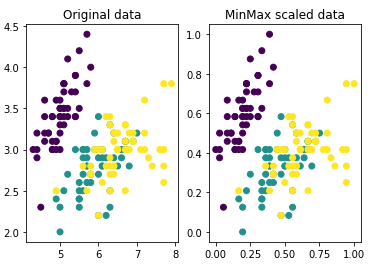
axes[0].set\_title("Original data")

axes[1].scatter(X\_scaled[:,0], X\_scaled[:,1], c=y)

axes[1].set\_title("MinMax scaled data")

plt.show()

**OUTPUT**



1. **Euclidean distance**

from math import sqrt

def euclidean\_distance(row1, row2):

  distance = 0.0

  for i in range(len(row1)-1):

    distance += (row1[i] - row2[i])\*\*2

  return sqrt(distance)

dataset = [[2.7810836,2.550537003,0],

  [1.465489372,2.362125076,0],

  [3.396561688,4.400293529,0],

  [1.38807019,1.850220317,0],

  [3.06407232,3.005305973,0],

  [7.627531214,2.759262235,1],

  [5.332441248,2.088626775,1],

  [6.922596716,1.77106367,1],

  [8.675418651,-0.242068655,1],

  [7.673756466,3.508563011,1]]

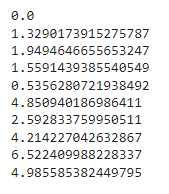
row0 = dataset[0]

for row in dataset:

  distance = euclidean\_distance(row0, row)

  print(distance)

**OUTPUT**



1. **Neighbors**

from math import sqrt

def euclidean\_distance(row1, row2):

  distance = 0.0

  for i in range(len(row1)-1):

    distance += (row1[i] - row2[i])\*\*2

  return sqrt(distance)

def get\_neighbors(train, test\_row, num\_neighbors):

  distances = list()

  for train\_row in train:

    dist = euclidean\_distance(test\_row, train\_row)

    distances.append((train\_row, dist))

  distances.sort(key=lambda tup: tup[1])

  neighbors = list()

  for i in range(num\_neighbors):

    neighbors.append(distances[i][0])

  return neighbors

dataset = [[2.7810836,2.550537003,0],

  [1.465489372,2.362125076,0],

  [3.396561688,4.400293529,0],

  [1.38807019,1.850220317,0],

  [3.06407232,3.005305973,0],

  [7.627531214,2.759262235,1],

  [5.332441248,2.088626775,1],

  [6.922596716,1.77106367,1],

  [8.675418651,-0.242068655,1],

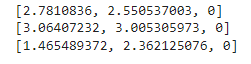
  [7.673756466,3.508563011,1]]

neighbors = get\_neighbors(dataset, dataset[0], 3)

for neighbor in neighbors:

  print(neighbor)

**OUTPUT**



1. **Expected ,got**

from math import sqrt

def euclidean\_distance(row1, row2):

  distance = 0.0

  for i in range(len(row1)-1):

    distance += (row1[i] - row2[i])\*\*2

  return sqrt(distance)

def get\_neighbors(train, test\_row, num\_neighbors):

  distances = list()

  for train\_row in train:

    dist = euclidean\_distance(test\_row, train\_row)

    distances.append((train\_row, dist))

  distances.sort(key=lambda tup: tup[1])

  neighbors = list()

  for i in range(num\_neighbors):

    neighbors.append(distances[i][0])

  return neighbors

def predict\_classification(train, test\_row, num\_neighbors):

  neighbors = get\_neighbors(train, test\_row, num\_neighbors)

  output\_values = [row[-1] for row in neighbors]

  prediction = max(set(output\_values), key=output\_values.count)

  return prediction

dataset = [[2.7810836,2.550537003,0],

  [1.465489372,2.362125076,0],

  [3.396561688,4.400293529,0],

  [1.38807019,1.850220317,0],

  [3.06407232,3.005305973,0],

  [7.627531214,2.759262235,1],

  [5.332441248,2.088626775,1],

  [6.922596716,1.77106367,1],

  [8.675418651,-0.242068655,1],

  [7.673756466,3.508563011,1]]

prediction = predict\_classification(dataset, dataset[0], 3)

print('Expected %d, Got %d.' % (dataset[0][-1], prediction))

**OUTPUT**



1. **Iris MinMaxScaler**

from sklearn.datasets import load\_iris

from sklearn.preprocessing import MinMaxScaler

import numpy as np

X, y = load\_iris(return\_X\_y=True)

print(X.shape)

scaler = MinMaxScaler()

scaler.fit(X)

X\_scaled = scaler.transform(X)

X\_scaled

**OUTPUT**

