```
PHYTON PAKIETY
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 22 21:47:52 2023

@author: Michał Olszewski
"""
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

lista = [-1, 2, 4]

npArray = np.array(lista)
type(npArray)

npArrayFloat = np.array(lista, dtype=float)
npArrayFloat

"""
        PAKET NUMPY
"""
#lista podstawowa
lista = [1, 2, 3]

#tablica jednowymiarowa
npArray = np.array(lista)

#tablica z jawnym typem
npArray = np.array(lista, dtype=float)

#tworzenie #1
np.arange(0, 10)

#tworzenie #2
np.linspace(1, 2, 11)

#tworzenie #3 - losowe
np.random.randint(0, 10)
np.random.randint(0, 10, 3)
np.random.random()

np.random.random_sample(10)


#właściwości
npArray = np.arange(20)

#rozmiar i kształt
npArray.size
npArray.shape

#zmiana kształtu
npArray2 = npArray.reshape((4, 5))
npArray2
npArray2.shape
```

```
#dostęp do elementów
npArray2[1, 1]
npArray2[1][1]

npArray2[1]

#wycinanie - jak w listach

#tylko pierwsza kolumna, wiersze 0 i 1
npArray2[0:2, 0]
npArray2[:, 0]


#tablice wielowymiarowe

#2D - losowe
npArray2D = np.random.random((10, 2))
npArray2D.shape

#3D - losowe
npArray3D = np.random.random((10, 2, 2))
npArray3D.shape

#tworzenie #4 - tylko zera, tylko jedynki, po diagonali
np.zeros((2, 2))
np.ones((2, 2))
np.diag([1, 2, 3, 4])

#przetwarzanie tablicy - funkcje zwektoryzowane (działają na każdym elemencie
tablicy)
#np.abs, np.power, np.cumsum - sumy czesciowe kolejnych elementow, np.sum, np.prod

npArray = np.arange(10)

np.abs(npArray)
np.power(npArray, 2)
np.cumsum(npArray)
np.sum(npArray)
np.prod(npArray)

#wyszukiwanie elementów
npArray = np.random.randint(0, 8, 10)

#unikalny, maksymalny, indeks maksymalnego
np.unique(npArray)
np.max(npArray)
np.argmax(npArray)

#iloczyn skalarny
np.dot([1, 2, 3], [2, 3, 4])

#specjalne wartości - nieliczba
np.NaN
np.nan

#rozszerzanie tablicy
np.repeat([1, 2, 3], 3)
np.tile([1, 2, 3], 3)
```

```
"""
        PAKET MATPLOTLIB
"""

#prosty wykres
ax_x = np.linspace(-5, 5, 100)
ax_y = np.power(ax_x, 2)

plt.xlabel('Oś X')
plt.ylabel('Oś Y')
plt.title('Tytuł')

plt.plot(ax_x, ax_y)

plt.show()


#wykres rozrzutu
os_x = np.random.sample(100)
os_y = np.random.sample(100)

plt.scatter(os_x, os_y)

plt.show()


#wiele funkcji na jednym
#oś X
ax_x = np.linspace(-5, 5, 100)

#oś Y dla f1 i f2
ax_y1 = np.power(ax_x, 2)
ax_y2 = np.power(ax_x, 3)

plt.plot(ax_x, ax_y1, label="second power")
plt.plot(ax_x, ax_y2, label="third power")

plt.legend(loc='lower right')

plt.show()



#dwa osobne wykresy w jednym

#wykresy 2x1
fix, ax = plt.subplots(2, 1)

ax[0].plot(ax_x, ax_y)
ax[1].scatter(os_x, os_y)

plt.show()

#wykresy 1x2
fix, ax = plt.subplots(1, 2)
```

```
ax[0].plot(ax_x, ax_y)
ax[1].scatter(os_x, os_y)
plt.show()

#wykresy 2x2
os_x = np.random.sample(100)
os_y = np.random.sample(100)

ax_x = np.linspace(-5, 5, 100)
ax_y = np.power(ax_x, 2)

fig, ax = plt.subplots(2, 2)

ax[0, 0].plot(ax_x, ax_y)
ax[1, 0].scatter(os_x, os_y)
ax[0, 1].scatter(os_x, os_y)
ax[1, 1].plot(ax_x, ax_y)

plt.show()


#zadanie
X = np.random.random(100)

#50 wierszy, 2 kolumny
X = X.reshape((50, 2))

Y = np.linspace(0, 1, 21)
Y

plt.plot(X, X, color='red')

#kolumna 0 i kolumna 1
plt.scatter(X[:, 0], X[:, 1], color='black')



"""
        PAKET PANDAS
"""

lista = [1, 2, 3, 5]

#Series - poj. kolumna, szeregi czasowe
data = pd.Series(list1)
data

#DataFrame jedna kolumna
dic = {'X': [1, 2, 3, 5]}
data = pd.DataFrame(dic)
data

#DataFrame dwie kolumny
dic = {'X': [1, 2, 3, 5], 'Y': ['A', 'B', 'C', 'D']}
data = pd.DataFrame(dic)
data
```

```
#podstawy

#typ każdej kolumny z osobna
data.dtypes

#wiersze tylko kolumny X
data['X']

#nazywanie wierszy (domyślnie ponumerowane liczbami)
data = pd.DataFrame(dic, index = ['r1', 'r2', 'r3', 'r4'])
data


#które wpisy uwzględnić?
dic = {'X': [1, 2, 3, 5, 1], 'Y': ['A', 'B', 'C', 'E', 'A'], 'Z': [0, 0, 0, 0, 0]}

data = pd.DataFrame(dic, columns=['X', 'Z'])
data

data = pd.DataFrame(dic, columns=['X', 'Y'])
data

#dodawanie wierszy
#append został usunięty w pandas 2.0, trza użyć concat
#przedtem konwertując słownik (wiersz) na DataFrame
#pd.DataFrame akceptuje tylko listy wiec pojedynczy wiersz obejmujemy klamrami []
row = {'X':2, 'Y':'B'}
df1 = data
df2 = pd.DataFrame([row])
dfLista = [df1, df2]

#concat zwraca nowy DataFrame, NIE modyfikuje istniejącego
data = pd.concat(dfLista, ignore_index=True)

#dodawanie kolumny
temp = data

#insert - modyfikuje istniejący
temp.insert(2, "Z", [4, 5, 6, 7, 8, 9])
temp

#usuwanie wierszy kolumn - drop
#inplace=True - modyfikuje istniejący, w przeciwnym razie zwraca nowy DataFrame bez
tego elementu
temp.drop(["Z"], axis='columns', inplace=True)
temp

#usuwanie wiersza o indeksie 4
temp.drop([4], inplace=True)
temp


dic = {'X': [1, 2, 3, 5], 'Y': ['A', 'B', 'C', 'D']}

#nazwane indeksy wierszy
data = pd.DataFrame(dic, index = ['r1', 'r2', 'r3', 'r4'])
data
```

```
#dostęp do wierszy, tylko kolumna X
X = data["X"]
X

type(X)

#dostęp do nazwanego wiersza, wszystkie kolumny
r = data.loc['r1']
r

#dostęp do wiersza według indeksu
data.iloc[0]

#iloc - dostęp po indeksie wiersza i kolumny
data

data.iloc[0, 0]
data.iloc[0, 1]

#wszystkie wiersze, kolumna o indeksie 1
data.iloc[:, 1]

#dostęp do wierszy według kolumny - dostęp przez "kropke" (zamiast string jako
parametr)
data.X

#zakres wierszy, kolumna o indeksie 1
data.iloc[1:3, 1]

#odwołania warunkowe
data

data_g3 = data['X'] > 2.5
print(data[data_g3])

data_gB = data['Y'] == 'B'
print(data[data_gB])

data_gBC = (data['Y'] == 'B') | (data['Y'] == 'C')
print(data[data_gBC])

data
data_gBC = data['Y'].isin(['B', 'C'])
print(data[data_gBC])

#format CSV

#zapis
data.to_csv('data.csv')

#wczytanie
new_data = pd.read_csv('data.csv')

#Porównaj zmienną new_data ze zmienną data
#new_data ma nienazwaną kolumne, tam gdzie były nazwy indeksów
#nowe indeksy są 0, 1, 2, 3 itd.

new_data
```

```
data

#wczytywanie - określanie która kolumna zawiera nazwy indeksów
new_data = pd.read_csv('data.csv', index_col = 0)
new_data

#wczytywanie - określenie że plik nie ma nagłówka
new_data = pd.read_csv('data.csv', header = None, index_col = 0)
new_data

#wczytywanie - określenie że nazwy kolumn są w wierszu 1 (nagłówek)
new_data = pd.read_csv('data.csv', header=1, index_col=0)
new_data

#pierwsze n wierszy
data.head(n = 2)

#ostatnie n wierszy
data.tail(n = 2)

#kształt
data.shape

#nazwy kolumn
data.columns
print(data)

#zmiana nazw kolumn
data.columns = ['C1', 'C2']
print(data)

#Radzenie sobie z brakujacymi danymi

#Sprawdzanie brakujacych wartosci
#znowu, append nie zadziała bo zostal usuniety, trzeba uzyc concat
#concat jest w module pandas, nie jako metoda w DataFrame

row = {'C1': 16}

df1 = data
df2 = pd.DataFrame([row])
dfLista = [df1, df2]
df12 = pd.concat(dfLista, ignore_index=True)

df1
df2
df12

df12.isnull()

#Liczba brakujących wartosci w kolumnach
df12
df12.isnull().sum()


#Usuwanie kolumn zawierajacych brakujące dane
data_temp = data.dropna(axis=1)
print(data_temp)
```

```
#Usuwanie wierszy zawierajacych brakujące dane
data_temp = data.dropna(axis=0)
print(data_temp)

#Uzupełnianie brakujących danych konkretnymi wartosciami
data_temp = data
data_temp = data_temp.fillna('AA')
data_temp.isnull().sum()
print(data_temp)


#wiersze w ktorych wartosci sie powtarzaja na wszystkich pozycjach usuwamy
unique_data = data.drop_duplicates()
unique_data

#Wyswietlenie statystyk opisowych
data.describe()
data

#Wyswietlenie macierzy korelacji
###data.corr() #działa tylko na danych numerycznych

"""
    Wizualizacja danych - Pandas + Matplotlib
"""
#Histogram
gauss = np.random.normal(0, 1, 200)
dfg = pd.DataFrame(gauss, columns = ['X'])
dfg
dfg.plot.hist(bins=4)
plt.show()



dfg.plot.hist(bins=8, density=True) #Wyświelta częstość, a nie liczbę wystąpień
plt.show()




dfg.plot.hist(bins=12)
plt.show()

#jądrowe estymatory gęstoci (funkcja przybliżacją rzeczywistą funkcję gęstosci
danych)
#Histogram
dfg.plot.hist(bins=8, density=True)
plt.show()



#Estymator jądrowy gęstosci
dfg.plot.density()
plt.show()
```

```
#wykresy pudełkowe
"""
Wykresy pudełkowe (boxploty)
są alternatywną graficzną prezentacją danych.
Pudełko przedstawia przedział od 1 do 3 kwartyla,
linia wewnątrz pudełka określa medianę.
Okręgi suerują wartości odstające.
"Wąsy" to odwartości odpowiedniego kwaryle dodany lub
odjęty rozstęp kwartylowy (wysokość pudełka).
"""


#Wykres pudełkowy dla jednej zmiennej
dfg.plot.box()



dic = {'X': [1, 2, 3, 5, 2]}
temp = pd.DataFrame(dic)
temp.insert(1, "Z", [4, 5, 6, 7, 8])
temp




temp

#Wykres pudełkowy, wiele zmiennych
temp.plot.box()


#wykres kolumnowy, przygotowanie danych do wyswietlenia
dic = {'X': [1, 2, 3, 4, 5], 'Y': ['A', 'B', 'C', 'D', 'E']}
data = pd.DataFrame(dic, columns=['X', 'Y'])
data.plot.bar(x='Y', y='X')

#konwersja z DataFrame na ndarray od numpy

#szybsza
nd1 = data.values
nd1

#standardowa
nd1 = np.array(data)
nd1
```

SI KERAS

```
from sklearn.datasets import load_iris
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

data = load_iris()
x_train, y_train = data.data, data.target

model = Sequential()
model.add(Dense(32, activation='relu', input_dim = 4))
model.add(Dense(8, activation='relu'))
model.add(Dense(3, activation='softmax'))

model.summary()

model.compile(loss="sparse_categorical_crossentropy", optimizer='SGD')
```

LSTM

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow. keras.datasets import mnist
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error

#1. Wczytaj dane z pliku 'moneypl_euro.csv'
data = pd.read_csv('moneypl_euro.csv', sep=',')

#2. Do zmiennej X przypisz wartości z kolumny 'Kurs średni
colX = data['Kurs średni']

#3. Skonwertuj wartości na przedział od 0 do 1

#Skaler, który będzie normalizować nasze dane, a gdy będzie potrzeba denormalizował
przetworzone (znormalizowane) dane
scaler = MinMaxScaler(feature_range=(0, 1))

#Znormalizowane dane
X_scaled = scaler.fit_transform(colX.values.reshape(-1, 1))

#4. Ze zbioru X utwórz ciągi danych.
#a.
"""
 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] ->
 [
     [1, 2, 3, 4, 5],
     [2, 3, 4, 5, 6],
     [3, 4, 5, 6, 7],
     [4, 5, 6, 7, 8],
     [5, 6, 7, 8, 9],
     [6, 7, 8, 9, 10]
 ]
"""
length = 10
#b. Utwórz zbiór X_train tak, aby każdy wiersz był ciągiem kolejnych length
elementów
X_train = []
for i in range(len(X_scaled) - length):
 X_train.append(X_scaled[i:i + length])

X_train = np.array(X_train)

#Kształt danych wejściowych i prze-kształt-conych
X_scaled.shape
X_train.shape

#c. Utwórz zbiór Y_train tak, aby każdy wiersz był kolejną wartością odpowiadającą
wierszowi z X_train
```

```
"""
     [1, 2, 3, 4, 5] -> 6
     [2, 3, 4, 5, 6] -> 7
     [3, 4, 5, 6, 7] -> 8
     [4, 5, 6, 7, 8] -> 9
     [5, 6, 7, 8, 9] -> 10
     [6, 7, 8, 9, 10] -> 11
"""
Y_train = []
for i in range(len(X_scaled) - length):
 Y_train.append(X_scaled[i + length])

Y_train = np.array(Y_train)

#Kształ danych prze-kształt-conych
Y_train.shape

X_train[0]
X_train[1]
Y_train[0]

#5. Podziel ciągi danych na zbiór uczący i testowy. Niech zbiór testowy składa się z
30
# ostatnich ciągów, a zbiór uczących z wszystkich poza nimi.

#Zbiór testowy
X_test = X_train[-30:]
Y_test = Y_train[-30:]

X_test.shape
Y_test.shape

#Zbiór uczący
X_train = X_train[:-30]
Y_train = Y_train[:-30]

X_train.shape
Y_train.shape

#6. Utwórz sieć złożoną z:
model = Sequential()
#a. jednej warstwy LSTM (np. z 50 neuronami)
model.add(tf.keras.layers.LSTM(500, input_shape=(length, 1)))
#b. warstwy Dense z jednym wektorem i liniową funkcją aktywacji
model.add(Dense(1, activation='linear'))

model.compile(optimizer='adam', loss='mean_squared_error', metrics=['acc'])

#7. Przygotuj dane, tak by pasowały do warstwy LSTM.
#Zmiana kształtu danych wejściowych
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

X_train.shape
X_test.shape

#8. Naucz model na danych uczących.
model.fit(X_train, Y_train, epochs=100, batch_size=32, verbose=1)
```

```
#9. Do osobnych zmiennych zapisz predykcję modelu dla danych uczących i testowych.
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)

#10. Przeskaluj zmienne przewidziane przez model oraz zmienne Y_train i Y_test do
#oryginalnych zakresów.
train_predict_org = scaler.inverse_transform(train_predict)
test_predict_org = scaler.inverse_transform(test_predict)

Y_train_org = scaler.inverse_transform(Y_train)
Y_test_org = scaler.inverse_transform(Y_test)

import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

#11. Wykonaj wykres dla oryginalnych danych i prognozowanych wartości na zbiorze
#uczącym
plt.figure(figsize=(10,6))
plt.plot(Y_train_org, label='Real')
plt.plot(train_predict_org, label='Predicted')
plt.title('Prediction on Training Set')
plt.legend()
plt.show()

#12. Wykonaj wykres dla oryginalnych danych i prognozowanych wartości na zbiorze
#testowym
plt.figure(figsize=(10,6))
plt.plot(Y_test_org, label='Real')
plt.plot(test_predict_org, label='Predicted')
plt.title('Prediction on Test Set')
plt.legend()
plt.show()

#13. Oblicz średni błąd absolutny na obu zbiorach.
mae_train = mean_absolute_error(Y_train_org, train_predict_org)
mae_test = mean_absolute_error(Y_test_org, test_predict_org)
```

PHYTON PAKIETY

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 23 16:09:44 2023

@author: Michał Olszewski
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#otwórz
#według zadania plik ma nagłówek ale go tam nie ma????
dane = pd.read_csv("iris.data", header=None, names=[0, 1, 2, 3, 'Name'])
dane

#lub
#dane.columns = [0, 1, 2, 3, 'Name']
len(dane)
dane.shape


#zmień na ndarray, wydziel atrybuty do X, klasy do Y

#klasy są w ostatniej kolumnie
X = dane.iloc[:, :4].values
X

Y = dane.iloc[:, 4].values
Y
#LUB
Y = dane['Name'].values
Y
#LUB
Y = dane.Name.values
Y



#Ksztalt X
X.shape
Y.shape

#Wykres pudełkowy

#z ndarray przy pomocy matplotlib
plt.boxplot(X)

#z DataFrame bezposrednio
dane.iloc[:, :4].plot.box()


#suma kwadratów różnic - ndarray i ndarray
def sumaKwadratowRoznic(arr1, arr2):
   kwadratyRoznic = (arr1 - arr2)**2
```

```
    suma = np.sum(kwadratyRoznic)

    return round(suma, 2)


#wychodzi 0.29, według pytania zła odpowiedź, testowane róWnież na poprzedniej
wersji funkcji (prawidłowej) i wychodzi też 0.29
sumaKwadratowRoznic(X[0], X[1])


temp = np.array([4.8, 3.1, 4.44, 0.9])
#numpy broadcast - "rozciąganie" jednej z tablic jesli wymiary sie nie zgadzaja
#tylko dla pewnych przypadkow wymiarow
((X - temp)**2)

def najblizszySasiadIndeks(temp, dane):
    return (((X - temp)**2)
            .sum(axis=1)
            .round(2)
            .argmin())

najblizszySasiadIndeks(temp, X)


def kNajblizszychSasiadowIndeksy(temp, dane, ilu):
    return (((X - temp)**2)
            .sum(axis=1)
            .round(2)
            .argsort()
            [:ilu])

#zadanie
kSasiadow = kNajblizszychSasiadowIndeksy(temp, X, 10)
os_X = np.arange(1, 11)
os_Y = kSasiadow

plt.plot(os_X, os_Y)


uniqueKlasy, uniqueLiczenie = np.unique(Y[kk], return_counts=True)

def klasy(temp, dane, ilu, ile_klas=3):
    sasiedziIdx = kNajblizszychSasiadowIndeksy(temp, dane, ilu)

    #tablica nazw klas, tablica liczby wystapien klas
    klasy, klasyLiczba = np.unique(Y[sasiedziIdx], return_counts=True)

    #indeks gdzie liczba klas jest najwieksza
    klasyMaxIdx = klasyLiczba.argmax()

    #nazwa klasy, wedlug indeksu wczesniej obliczonego (tj. max)
    return klasy[klasyMaxIdx]

klasy(temp, X, 10)
```

```
PHYTON PODSTAWY
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 21 11:33:07 2023

@author: Michał Olszewski
"""
import os
import itertools
os.getcwd()

#otwórz
dane = open("iris.data", 'r')


#podziel na linie
dane = dane.read().split("\n")

len(dane)


#wywal dwie ostatnie puste linie
dane = dane[0:-2]
len(dane)



#podziel string w każdym wierszu na osobne dane
nowa = [e.split(",") for e in dane]

#konwertuj 4 pierwsze na float
nowa = [[float(e[0]), float(e[1]), float(e[2]), float(e[3]), e[4]] for e in nowa]


#zadanie
nowa[42][0]
nowa[83][1]
nowa[55][2]
nowa[102][3]
nowa[30][4]


def sumaKwadratowRoznic(l1, l2):
    #iteracja po elemencie z l1 i l2 naraz, przedtem wywal stringa
    suma = sum([(e1 - e2)**2 for e1, e2 in zip(l1[0:4], l2[0:4])])

    #zaokraglij
    return round(suma, 2)


#zadanie
sumaKwadratowRoznic(nowa[1], nowa[20])
sumaKwadratowRoznic(nowa[4], nowa[50])
sumaKwadratowRoznic(nowa[6], nowa[108])
sumaKwadratowRoznic(nowa[55], nowa[122])

#
#najblizszy sasiad do temp
```

```
#

temp = [4.8, 3.1, 4.44, 0.9, '']


#suma kwadratow roznic temp z kazdym, zapisz dodatkowo indeks wiersza gdzie min się
znajduje
sumyKwadratowRoznicTemp = [[sumaKwadratowRoznic(rekord, temp), i] for i, rekord in
enumerate(nowa)]
sumyKwadratowRoznicTemp

#wartosc minimum + wiersz w którym minimum się znajduje
#lambda k: ... - licz min według 1 kolumny (wartosci min)
minnn, minIdx = min(sumyKwadratowRoznicTemp, key = lambda k: k[0])
minnn



#n najblizszych sasiadow
def kilkaSasiadow(temp, dane, ilu):
    #jak wyzej
    sumyKwadratowRoznicTemp = [[sumaKwadratowRoznic(rekord, temp), i]  for i, rekord
in enumerate(dane)]

    #posortuj rosnaca (od min do max) według wartosci min (lambda e: ...)
    posortowana = sorted(sumyKwadratowRoznicTemp, key = lambda e: e[0])

    #zwroc sąsiadów najbliższych według zmiennej ilu
    return posortowana[:ilu]


#zadanie
kilkaSasiadow(temp, nowa, 3)

def wyznaczKlase(temp, dane, ilu, ileKlas=3):
    #najblizsi sasiedzi
    sasiedzi = kilkaSasiadow(temp, dane, ilu)

    #w ks są wiersze pary [wartosc-min, idx], na podstawie indeksu idx pobierz nazwe
klasy ((ostatni element) z danych
    ks = [nowa[idx][-1] for minimalna, idx in sasiedzi]


    #zlicz wystapienia kazdej z klas

    #slownik nazwa klasy -> liczba wystapien
    kDict = {}

    for nazwa in ks:

        #pierwszy raz napotykamy nazwe to inicjalizujemy w slowniku licznik zerem
        if not(nazwa in kDict.keys()):
            kDict[nazwa] = 0

        #bierzemy nazwa i dla niej zwiekszamy licznik
        kDict[nazwa] += 1

    #wyznaczenie zwyciezcy, wyznacz max według liczby wystąpień
```

```
    maxName = max(kDict.items(), key=lambda k: k[1])

    return maxName[0]

#setosa - 0
#versicolor - 1
#virginica - 2

#zadanie
temp = [4.8, 3.1, 4.44, 0.9]

wyznaczKlase(temp, nowa, 1)
wyznaczKlase(temp, nowa, 2)
wyznaczKlase(temp, nowa, 50)

#zadanie
temp = [2.8, 3.1, 4.44, 1.9]


wyznaczKlase(temp, nowa, 1)
wyznaczKlase(temp, nowa, 2)
wyznaczKlase(temp, nowa, 3)
```

SIECI KONWOLUCYJNE

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow. keras.datasets import mnist
import tensorflow as tf

from tensorflow. keras.layers import Dense, Flatten
from tensorflow. keras.layers import Conv2D, MaxPool2D

import matplotlib.pyplot as plt
import numpy as np



#Ładowanie zbioru danych
(x_train, y_train), (x_test, y_test) = mnist.load_data()

#Kształkt
x_train.shape

#Zmiana kształtu danych na 60k elementów po 28x28, jeden kanał
x_train = x_train.reshape((60000, 28, 28, 1))
x_test = x_test.reshape((10000, 28, 28, 1))

#Normalizacja danych do [0, 1]
x_train = np.array(x_train, dtype=float)
x_train /= 255
x_test = np.array(x_test, dtype=float)
x_test /= 255

#Budowa sieci
model = Sequential()
model.add(Conv2D(32, kernel_size=(4, 4), activation='relu', input_shape=(28, 28,
1)))
model.add(MaxPool2D(pool_size=(3, 3)))

model.add(Conv2D(8, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(10, activation='softmax'))

#Kompilacja
model.compile(loss="sparse_categorical_crossentropy", optimizer='SGD', metrics =
['acc'])

model.summary()

#Trenowanie
model.fit(x_train, y_train, epochs = 10, validation_data = (x_test, y_test))

#Statystyki
loss = model.history.history['loss']
vloss = model.history.history['val_loss']
acc_list = model.history.history['acc']
vacc_list = model.history.history['val_acc']
```

```
#Dokładność na zbiorze treningowym
y_hat = model.predict(x_train)
classes = y_hat.argmax(axis=1)
acc =sum(y_train == classes) / len(x_train)

proc=acc*100
print("\nacc:")
print("{:.0f}%".format(proc))

#Dokładność na zbiorze testowym
y_hat = model.predict(x_test)
classes = y_hat.argmax(axis=1)
acc =sum(y_test == classes) / len(x_test)

proc=acc*100
print("\nacc:")
print("{:.0f}%".format(proc))

import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"


os_x = np.arange(10)+1
plt.plot(os_x, loss)
plt.plot(os_x, acc_list)

fig, ax = plt.subplots(2,1)
ax[0].plot(os_x, loss, label="train")
ax[0].plot(os_x, vloss, label="test")
ax[0].legend()
ax[1].plot(os_x, acc_list, label="train")
ax[1].plot(os_x, vacc_list, label="test")
ax[1].legend(loc='lower right')

plt.show()
```

SIECI NEURONOWE

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  4 14:42:56 2023

@author: michalo
"""

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.datasets import mnist
import tensorflow as tf

import matplotlib.pyplot as plt
import numpy as np



(x_train, y_train), (x_test, y_test) = mnist.load_data()


x_train.shape

x_train = x_train.reshape((60000, -1))
x_test = x_test.reshape((10000, -1))

x_train = np.array(x_train, dtype=float)
x_train /= 255
x_test = np.array(x_test, dtype=float)
x_test /= 255


# Wyświetlanie kształtu danych
print("Kształt x_train po zmianie wymiarowości:", x_train.shape)
print("Kształt x_test po zmianie wymiarowości:", x_test.shape)


model = Sequential()
model.add(Dense(32, activation='relu', input_dim = 784))
model.add(Dense(8, activation='relu'))
model.add(Dense(10, activation='softmax'))

#Summary
#Do każdego wyjścia należy dodać jeszcze wagi bias
#784*32 + 32 = 25120
#32*8 + 8 = 264
#8*10 + 10 = 90
model.summary()

model.compile(loss="sparse_categorical_crossentropy", optimizer='SGD', metrics =
['acc'])


model.fit(x_train, y_train, epochs = 4, validation_data = (x_test, y_test))
```

```
loss = model.history.history['loss']
vloss = model.history.history['val_loss']
acc_list = model.history.history['acc']
vacc_list = model.history.history['val_acc']



y_hat = model.predict(x_train)
classes = y_hat.argmax(axis=1)
acc =sum(y_train == classes) / len(x_train)

proc=acc*100
print("\nacc:")
print("{:.0f}%".format(proc))

import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"


os_x = np.arange(4)+1
plt.plot(os_x, loss)
plt.plot(os_x, acc_list)

fig, ax = plt.subplots(2,1)
ax[0].plot(os_x, loss, label="train")
ax[0].plot(os_x, vloss, label="test")
ax[0].legend()
ax[1].plot(os_x, acc_list, label="train")
ax[1].plot(os_x, vacc_list, label="test")
ax[1].legend(loc='lower right')

plt.show()
```