Tuần 1: Xóa các phần tử bị quên không liệt kê trong bảng cuối cùng. Tính trung bình và phương sai

import pandas as pd

df = pd.read\_csv('data.csv', index\_col='region\_name')

df.info()

# Представим ситуацию, что из-за невнимательности операциониста, регионы:

# Республика Алтай, Магаданская обл. оказались не представлены в итоговой сводке.

df.drop(['Республика Алтай', 'Магаданская обл.'], inplace=True)

df.sort\_values('salary', inplace=True)

print(df.head(6).tail(1))

print(df.head(49).tail(1))

print(df.head(51).tail(1))

print(df['salary'].mean())

print(df['salary'].median())

Tuần 2: Sử dụng phương pháp chính di chuyển tới tọa độ mới và

Bài 1: in ra tọa độ các thành phần

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

import numpy as np

f = np.loadtxt('34\_25.csv', delimiter=',') #Lấy dữ liệu

f\_mean\_cols = np.mean(f, axis=0)           #Tính trung bình mẫu

for i in range(len(f)): f[i] -= f\_mean\_cols #Thay vị trí của từng giá trị trong f bằng phương sai so với trung bình mẫu -> ma trận hiệp phương sai

# Áp dụng PCA cho 2 thành phần chính

pca = PCA(n\_components=2, svd\_solver='full')

z = pca.fit\_transform(f)  #Chuyển dữ liệu f vào tọa độ mới đc tạo bởi 2tp chính đã tìm ra ở trên.

print('Доля объясненной дисперсии: ', sum(pca.explained\_variance\_ratio\_)) # Tính tổng tỉ lệ phương sai giải tích -> Có bn % dữ liệu đc bảo toàn

print('Первый элемент:')

print(z[0])

# Khởi tạo PCA với số lượng thành phần chính bằng số lượng đặc trưng(10)

pca\_full = PCA(n\_components=f.shape[1])

# Áp dụng PCA cho toàn bộ tập dữ liệu

pca\_full.fit(f)

# Tính tỉ lệ tích lũy của phương sai giải tích

cumulative\_explained\_variance = pca\_full.explained\_variance\_ratio\_.cumsum()

# Tính số lượng thành phần chính mà khi đó phương sai giải tích lớn hơn 0.85

min\_components = (cumulative\_explained\_variance > 0.85).argmax() + 1

print(min\_components, cumulative\_explained\_variance)

# Vẽ đồ thị 2D biểu diễn dữ liệu

plt.scatter(x=z[:,0], y=z[:,1])

plt.show()

Bài 2: Vẽ hình từ dữ liệu

import numpy as np

from PIL import Image

# Менять только название файла, катинка сама появится

loadings = np.loadtxt('X\_loadings\_417.csv', delimiter=';')

reduced = np.loadtxt('X\_reduced\_417.csv', delimiter=';')

x = loadings @ reduced.transpose()

with Image.new(mode='L', size=x.shape) as img:

    for i in range(x.shape[0]):

        for j in range(x.shape[1]):

            img.putpixel((i, j), int(x[i,j] \* 255))

    img.show()

Tuần 4: Cho 1 bảng.

Bài 1: Yêu cầu tạo mẫu huấn luyện và tính các giá trị

from sklearn.linear\_model import LinearRegression

import pandas as pd

import numpy as np

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

print('Mean X:', df['X'].mean())

print('Mean Y:', df['Y'].mean())

x = np.array(df['X']).reshape((len(df['X']), 1))

y = np.array(df['Y']).reshape((len(df['Y']), 1))

l = LinearRegression()

l.fit(x, y)

print('Коэффициент тета\_1:', l.coef\_)

print('Коэффициент тета\_0:', l.intercept\_)

print('R^2 статистика:', l.score(x, y))

Bài 2:

from sklearn.linear\_model import LinearRegression

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

from sklearn.decomposition import PCA

from sklearn.metrics import r2\_score

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

# Các tham số

df = pd.read\_csv('fish\_train.csv')

test\_size = 0.2

random\_state=21

def get\_splits():

    species = df['Species']

    x = df.drop(['Weight', 'Species'], axis=1)

    y = df['Weight']

    return train\_test\_split(x, y, test\_size=test\_size, random\_state=random\_state, stratify=species)

# Xây dựng mô hình cơ bản

x\_train, x\_test, y\_train, y\_test = get\_splits()

print('Среднее колонки Width:', x\_train['Width'].mean())

lr = LinearRegression()

lr.fit(x\_train, y\_train)

y\_pred = lr.predict(x\_test)

print('R2 оценка для тестового набора:', r2\_score(y\_test, y\_pred))

# Thêm tính năng tiền xử lý

# Ma trận hiệp phương sai

print('Матрица корреляции:')

correlation\_matrix = df.corr()

plt.figure(figsize=(10, 8))

sns.heatmap(correlation\_matrix, annot=True, fmt=".2f")

plt.show()

x\_train, x\_test, y\_train, y\_test = get\_splits()

# Tỉ lệ phương sai được giải thích

pca = PCA(n\_components=1, svd\_solver='full')

pca.fit(x\_train[['Length1', 'Length2', 'Length3']])

print('Доля объясненной дисперсии:', pca.explained\_variance\_ratio\_)

df['Lengths'] = pca.transform(df[['Length1', 'Length2', 'Length3']])

df.drop(['Length1', 'Length2', 'Length3'], axis=1, inplace=True)

x\_train, x\_test, y\_train, y\_test = get\_splits()

lr = LinearRegression()

lr.fit(x\_train, y\_train)

y\_pred = lr.predict(x\_test)

print('R2 оценка для тестового набора:', r2\_score(y\_test, y\_pred))

# Sửa đổi đặc điểm

df[['Width', 'Height', 'Lengths']] = df[['Width', 'Height', 'Lengths']].apply(lambda x: x\*\*3)

x\_train, x\_test, y\_train, y\_test = get\_splits()

print('Среднее Width после возведения в куб:', x\_train['Width'].mean())

unique\_species = df['Species'].unique()

for i in unique\_species:

    plt.scatter(df[df['Species'] == i]['Width'], df[df['Species'] == i]['Weight'], label=i)

plt.legend()

plt.xlabel('Width')

plt.ylabel('Weight')

plt.show()

lr = LinearRegression()

lr.fit(x\_train, y\_train)

y\_pred = lr.predict(x\_test)

print('R2 оценка для тестового набора:', r2\_score(y\_test, y\_pred))

dummies = pd.get\_dummies(df['Species'])

df[list(dummies.columns)] = dummies

x\_train, x\_test, y\_train, y\_test = get\_splits()

lr = LinearRegression()

lr.fit(x\_train, y\_train)

y\_pred = lr.predict(x\_test)

print('R2 оценка для тестового набора:', r2\_score(y\_test, y\_pred))

df.drop(list(dummies.columns), axis=1, inplace=True)

dummies = pd.get\_dummies(df['Species'], drop\_first=True)

df[list(dummies.columns)] = dummies

x\_train, x\_test, y\_train, y\_test = get\_splits()

lr = LinearRegression()

lr.fit(x\_train, y\_train)

y\_pred = lr.predict(x\_test)

print('R2 оценка для тестового набора:', r2\_score(y\_test, y\_pred))

Tuần 5: Cho 1 bảng

Bài 1: In ra các khoảng cách tới đối tượng lân cận gần nhất

from math import sqrt

from sklearn.preprocessing import MinMaxScaler

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import f1\_score

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

# Первый набор данных

# data1.csv

task1 = pd.read\_csv('data1.csv', index\_col='id')

# Координаты для заданий

x = 39

y = 22

task1['euclid'] = task1.apply(lambda row: sqrt((row['X'] - x)\*\*2 + (row['Y'] - y)\*\*2), axis=1)

task1['manhatton'] = task1.apply(lambda row: abs(row['X'] - x) + abs(row['Y'] - y), axis=1)

task1.sort\_values(by='euclid', inplace=True)

#Đối tượng gần nhất bằng euclid

print('1.1:',task1.head(1)['euclid'].values[0])

#Số nhận dạng 3 đối tượng gần nhất

print('1.2:', ','.join(map(str,task1.head(3).reset\_index()['id'])))

#NHập số lớp cho điểm đó tại k=3

print('1.3', ','.join(map(str,task1.head(3)['Class'])), '(Выберите, какого класса больше)')

task1.sort\_values(by='manhatton', inplace=True)

#Khoảng cách gần nhất từ tọa độ manhatton

print('1.4:',task1.head(1)['manhatton'].values[0])

#3 đối tượng gần nhất theo số liệu khối

print('1.5:', ','.join(map(str,task1.head(3).reset\_index()['id'])))

#Lớp cho điểm đó tại k = 3

print('1.6', ','.join(map(str,task1.head(3)['Class'])), '(Выберите, какого класса больше)')

# Второй набор данных

# data2.csv

task2 = pd.read\_csv('data2.csv')

#   Xóa education và marital-status.

# In ra số lượng đối tượng không phải là số.

to\_delete = ['education', 'marital-status']

task2.drop(to\_delete, axis=1, inplace=True)

print('2.1:')

print(task2.dtypes.value\_counts())

#Tỉ lệ các lớp đối tượng

print('2.2:', task2['label'].value\_counts().loc[0] / len(task2))

# 1. Xây dựng mô hình cơ bản

# Chỉ chọn các tính năng số từ tập dữ liệu.

non\_numeric\_columns = [col for col in task2.columns if task2.dtypes.loc[col] == 'object']

task2\_clean = task2.drop(non\_numeric\_columns, axis=1)

# При помощи train\_test\_split() разбейте набор данных на обучающую

# и тестовую выборки с параметрами test\_size=0.2, random\_state=27.

# Используйте стратификацию по колонке label.

x\_train, x\_test, y\_train, y\_test = train\_test\_split(

    task2\_clean.drop('label', axis=1),

    task2\_clean['label'],

    test\_size=0.2,

    random\_state=27,

    stratify=task2\_clean['label']

)

#Gia trị trung bình mẫu của fnlwgt

print('2.3:', x\_train['fnlwgt'].mean())

knn1 = KNeighborsClassifier()

knn1.fit(x\_train, y\_train)

y\_pred = knn1.predict(x\_test)

#f1\_score

print('2.4:', f1\_score(y\_test, y\_pred))

# Chia tỉ lệ tính năng.

scaler = MinMaxScaler()

x\_train = pd.DataFrame(scaler.fit\_transform(x\_train), columns=x\_train.columns)

x\_test = pd.DataFrame(scaler.transform(x\_test), columns=x\_test.columns)

#Khi đó trung bình mẫu của fnlwgt

print('2.5:', x\_train['fnlwgt'].mean())

knn2 = KNeighborsClassifier()

knn2.fit(x\_train, y\_train)

y\_pred = knn2.predict(x\_test)

#f1\_score

print('2.6:', f1\_score(y\_test, y\_pred))

#chọn biểu đồ phù hợp

print('2.7: Откроются новые окна для просмотра, выберите соответствующие')

fig, axs = plt.subplots(1, len(non\_numeric\_columns))

for ax, col in zip(axs, non\_numeric\_columns):

    c = task2[col].value\_counts()

    ax.bar(c.index, c)

    ax.set\_title(col)

plt.show()

#Số hàng trong bản gốc k tính các cột chỉ định xóa

rows\_with\_nan = task2.apply(lambda row: (row=='?').any(), axis=1)

print('2.8:', rows\_with\_nan.sum())

task2\_with\_dummies = task2[~rows\_with\_nan].copy()

for col in non\_numeric\_columns:

    dummies = pd.get\_dummies(task2\_with\_dummies[col], drop\_first=True)

    task2\_with\_dummies[dummies.columns] = dummies

    task2\_with\_dummies.drop(col, axis=1, inplace=True)

# Tổng số tính nawg được truy xuất

print('2.9:', len(task2\_with\_dummies.columns) - 1)

#Sử dụng tập dữ liệu thu được, huấn luyện mô hình phân loại theo cách tương tự như đã thực hiện đối với mô hình cơ sở

x\_train, x\_test, y\_train, y\_test = train\_test\_split(

    task2\_with\_dummies.drop('label', axis=1),

    task2\_with\_dummies['label'],

    test\_size=0.2,

    random\_state=27,

    stratify=task2\_with\_dummies['label']

)

scaler = MinMaxScaler()

x\_train = pd.DataFrame(scaler.fit\_transform(x\_train), columns=x\_train.columns)

x\_test = pd.DataFrame(scaler.transform(x\_test), columns=x\_test.columns)

knn3 = KNeighborsClassifier()

knn3.fit(x\_train, y\_train)

y\_pred = knn3.predict(x\_test)

#Tính f1\_score

print('2.10:', f1\_score(y\_test, y\_pred))

#Sử dụng tập dữ liệu gốc (không có cột education và marital-status), điền vào chỗ trống giá trị xuất hiện thường xuyên nhất trong cột.

#Tiếp theo - tương tự như trường hợp trước: one-hot, chia nhỏ, mở rộng quy mô, đào tạo và đánh giá. Tính f1\_score() của mô hình.

for col in task2.columns:

    task2[col].replace('?', task2[col].mode().values[0], inplace=True)

for col in non\_numeric\_columns:

    dummies = pd.get\_dummies(task2[col], drop\_first=True)

    task2[dummies.columns] = dummies

    task2.drop(col, axis=1, inplace=True)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(

    task2.drop('label', axis=1),

    task2['label'],

    test\_size=0.2,

    random\_state=27,

    stratify=task2['label']

)

scaler = MinMaxScaler()

x\_train = pd.DataFrame(scaler.fit\_transform(x\_train), columns=x\_train.columns)

x\_test = pd.DataFrame(scaler.transform(x\_test), columns=x\_test.columns)

knn4 = KNeighborsClassifier()

knn4.fit(x\_train, y\_train)

y\_pred = knn4.predict(x\_test)

#Tính f1\_score

print('2.11:', f1\_score(y\_test, y\_pred))

Bài 2: Nhập bảng vào mảng, sau đó tính toán

from math import log, exp

# Вбейте данные сюда

spam = 24

not\_spam = 26

spam\_words = 135

not\_spam\_words = 145

words = {

    'Offer': (12, 15),

    'Purchase': (10, 5),

    'Bonus': (15, 6),

    'Free': (9, 0),

    'Cash': (18, 16),

    'Investment': (7, 2),

    'Money': (2, 21),

    'Gift': (8, 4),

    'Bill': (33, 17),

    'Access': (21, 59)

}

text = 'Money Cash Remove Bonus Online Gift Access'.split(' ')

print('P("спам") =', spam / (spam + not\_spam))

f\_spam = log(spam / (spam + not\_spam))

f\_not\_spam = log(not\_spam / (spam + not\_spam))

r = len(list(filter(lambda x: x not in words, text)))

for word in text:

    a, b = words.get(word, (0, 0))

    f\_spam += log((1 + a) / (spam\_words + len(words) + r))

    f\_not\_spam += log((1 + b) / (not\_spam\_words + len(words) + r))

print('F("спам") =', f\_spam)

print('F("не спам") =', f\_not\_spam)

print('P("спам"|Письмо) =', exp(f\_spam) / (exp(f\_spam) + exp(f\_not\_spam)))

Tuần 6: Chuẩn bị và phân tích dữ liệu sơ cấp

from sklearn.linear\_model import LogisticRegression

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

from sklearn.metrics import f1\_score

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import re

# 1. Подготовка и первичный анализ данных

df = pd.read\_csv('titanic\_train.csv')

#Đặt số lượng giá trị tối thiểu của age

print('Установите количество пропущенных значений колонки age:', df['age'].isna().sum())

#Tính tỉ lệ sống sót

print('Вычислите долю выживших:', df['survived'].sum() / df['survived'].count())

#Xác định tỷ lệ giá trị bị thiếu trong mỗi tính năng và loại bỏ những tính năng có tỷ lệ giá trị bị thiếu lớn hơn một phần ba. Đồng thời loại bỏ cột ticket

#Tính giá trị trung bình mẫu của cột fam\_size

cols\_to\_delete = ['ticket']

for col in df.columns:

    if (df[col].isna().sum() / len(df) > 1/3): cols\_to\_delete.append(col)

print('Колонки, которые будут удалены:', cols\_to\_delete)

df.drop(cols\_to\_delete, axis=1, inplace=True)

df['fam\_size'] = df['sibsp'] + df['parch']

df.drop(['sibsp', 'parch'], axis=1, inplace=True)

print('Вычислите выборочное среднее колонки fam\_size:', df['fam\_size'].mean())

#Tính toán số lượng yếu tố dự đoán của tập dữ liệu kết quả.

print('Вычислите число предикторов полученного набора данных:', len(df.columns) - 1)

#Dựa trên số liệu thống kê có sẵn, hãy ước tính khả năng sống sót nếu hành khách là đại diện của loại sau: female and pclass:3.

print('вероятность выжить если пассажир -- представитель следующей категории: female and pclass:3:', df[(df['sex'] == 'female') & (df['pclass'] == 3)]['survived'].sum() / len(df[(df['sex'] == 'female') & (df['pclass'] == 3)]))

#Biểu đồ

df[df['survived'] == 1]['age'].hist(bins=15)

plt.show()

# 2. Dựng mô hình dựa theo số liệu bài cho

random\_state = 17

test\_size = 0.2

max\_iter = 1000

#Tính f1\_score

df\_only\_numeric = df.select\_dtypes(include=np.number).dropna()

x\_train, x\_test, y\_train, y\_test = train\_test\_split(df\_only\_numeric.drop('survived', axis=1), df\_only\_numeric['survived'], random\_state=random\_state, test\_size=test\_size, stratify=df\_only\_numeric['survived'])

lre = LogisticRegression(random\_state=random\_state, max\_iter=max\_iter)

lre.fit(x\_train, y\_train)

y\_pred = lre.predict(x\_test)

print('f1\_score = ', f1\_score(y\_test, y\_pred))

#Từ tập dữ liệu gốc, chỉ chọn các đặc điểm số và điền vào các khoảng trống bằng giá trị trung bình của cột.

#Tương tự hơn nữa: phân chia, đào tạo, đánh giá.

#Tính f1\_score

df\_only\_numeric = df.select\_dtypes(include=np.number)

for col in df\_only\_numeric.columns:

    df\_only\_numeric[col].fillna(df\_only\_numeric[col].mean(), inplace=True)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(df\_only\_numeric.drop('survived', axis=1), df\_only\_numeric['survived'], random\_state=random\_state, test\_size=test\_size, stratify=df\_only\_numeric['survived'])

lre = LogisticRegression(random\_state=random\_state, max\_iter=max\_iter)

lre.fit(x\_train, y\_train)

y\_pred = lre.predict(x\_test)

print('f1\_score = ', f1\_score(y\_test, y\_pred))

#Từ tập dữ liệu gốc, chọn các đặc điểm số cũng như cột name. Tạo một cột honorific riêng và đặt các giá trị chức danh (Mr., Ms., Dr., v.v.) được trích xuất từ ​​cột tên vào đó.

df\_only\_numeric = df.select\_dtypes(include=np.number)

df\_only\_numeric['honorific'] = df['name'].apply(lambda x: re.search(r' (\S+?)\. ', x).group(1))

#Tính số lượt truy cập duy nhất.

print('Число уникальных обращений:', len(df\_only\_numeric['honorific'].unique()))

#Tính tỷ lệ các hàng có giá trị Master so với số lượng tất cả nam giới

print('Вычислите долю строк со значением Master относительно числа всех мужчин:', len(df\_only\_numeric[(df\_only\_numeric['honorific'] == 'Master') & (df['sex'] == 'male')]) / len(df\_only\_numeric[df['sex'] == 'male']))

#Tạo mảng theo đề bài

replacements = {

    'Mr': ['Rev', 'Col', 'Dr', 'Major', 'Don', 'Capt'],

    'Mrs': ['Dona', 'Countess'],

    'Miss': ['Mlle', 'Ms']

}

def replace\_honorific(x):

    for k in replacements:

        if (x in replacements[k]): return k

    return x

df\_only\_numeric['honorific'] = df\_only\_numeric['honorific'].apply(replace\_honorific)

print('Вычислите средний возраст категории Master:', df\_only\_numeric.groupby('honorific')['age'].mean()['Master'])

df\_only\_numeric['age'] = df\_only\_numeric.groupby('honorific')['age'].transform(lambda x: x.fillna(x.mean()))

df\_only\_numeric.drop('honorific', axis=1, inplace=True)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(df\_only\_numeric.drop('survived', axis=1), df\_only\_numeric['survived'], random\_state=random\_state, test\_size=test\_size, stratify=df\_only\_numeric['survived'])

lre = LogisticRegression(random\_state=random\_state, max\_iter=max\_iter)

lre.fit(x\_train, y\_train)

y\_pred = lre.predict(x\_test)

print('f1\_score = ', f1\_score(y\_test, y\_pred))

# 3. Модель с использованим категориальных признаков

#Trong tập dữ liệu gốc, hãy điền vào các khoảng trống trong cột age bằng các giá trị dựa trên lần truy cập (như ở bước trước).

#Sau đó, xóa name và honorific

df['age'] = df\_only\_numeric['age']

df.drop('name', axis=1, inplace=True)

dummies = pd.get\_dummies(df.select\_dtypes(exclude=np.number), prefix=df.select\_dtypes(exclude=np.number).columns, drop\_first=True)

df[dummies.columns] = dummies

df.drop(df.select\_dtypes(exclude=np.number).columns, axis=1, inplace=True)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(df.drop('survived', axis=1), df['survived'], random\_state=random\_state, test\_size=test\_size, stratify=df['survived'])

lre = LogisticRegression(random\_state=random\_state, max\_iter=max\_iter)

lre.fit(x\_train, y\_train)

y\_pred = lre.predict(x\_test)

print('f1\_score = ', f1\_score(y\_test, y\_pred))

**RYBEK**

import pandas as pd

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

from sklearn.preprocessing import MinMaxScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import f1\_score

from sklearn.neighbors import KNeighborsClassifier

# Đọc tệp CSV

df = pd.read\_csv("report.csv")

# Lọc các dòng có giá trị MIP trong khoảng từ 10 đến 100

filtered\_df = df[(df['MIP'] >= 10) & (df['MIP'] <= 100)]

# In số hàng trong mẫu kết quả

print("Số hàng trong mẫu kết quả:", len(filtered\_df))

#Gía trị trung bình mẫu

print(filtered\_df['MIP'].mean())

#Gía trị tối thiểu:

print(filtered\_df['MIP'].min())

#2

# Sắp xếp dataframe theo cột 'SIP' theo thứ tự tăng dần

df\_sorted = filtered\_df.sort\_values(by='SIP', ascending=True)

# Chia khung dữ liệu đã sắp xếp thành các tính năng và biến mục tiêu

X = df\_sorted.drop('TARGET', axis=1)

y = df\_sorted['TARGET']

# Chia dữ liệu thành các tập huấn luyện và kiểm tra với sự phân tầng và trạng thái ngẫu nhiên cụ thể

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=33, stratify=y)

# Xác định giá trị tối đa của cột STDC từ tập huấn luyện

print(X\_train['STDC'].max())

#3

# Khởi tạo StandardScaler

scaler  = MinMaxScaler()

# Điều chỉnh bộ chia tỷ lệ cho phù hợp với dữ liệu huấn luyện và chuyển đổi nó

X\_train = pd.DataFrame(scaler.fit\_transform(X\_train), columns=X\_train.columns)

X\_test = pd.DataFrame(scaler.transform(X\_test), columns=X\_test.columns)

# Xác định giá trị trung bình của cột 'STDIP' từ tập huấn luyện được chia tỷ lệ

print("mean", X\_train['STDIP'].mean())

#Используйте полученные данные и обучите алгоритмы

#Khởi tạo mô hình hồi quy logistic với tham số mặc định

logreg\_model=LogisticRegression()

# Điều chỉnh mô hình trên dữ liệu huấn luyện đã chia tỷ lệ

logreg\_model.fit(X\_train, y\_train)

# Dự đoán trên dữ liệu thử nghiệm được chia tỷ lệ

y\_pred = logreg\_model.predict(X\_test)

# Tạo ma trận nhầm lẫn

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

# Trích xuất giá trị True Positive (TP)

tp\_value = conf\_matrix[1, 1]

print(tp\_value)

#Tính f1\_score

print(f1\_score(y\_test, y\_pred))

# Khởi tạo mô hình KNeighbors Classifier với n\_neighbors=3

knn\_model = KNeighborsClassifier(n\_neighbors=3)

# Điều chỉnh mô hình trên dữ liệu huấn luyện đã chia tỷ lệ

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

# Dự đoán trên dữ liệu thử nghiệm được chia tỷ lệ

y\_pred\_knn = knn\_model.predict(X\_test)

# Tạo ma trận nhầm lẫn cho các dự đoán của mô hình KNN

conf\_matrix\_knn = confusion\_matrix(y\_test, y\_pred\_knn)

# Trích xuất giá trị True Positive (TP) cho mô hình KNN

tp\_value\_knn = conf\_matrix\_knn[1, 1]

print(tp\_value\_knn)

#Tính f1\_score

print(f1\_score(y\_test, y\_pred\_knn))

import pandas as pd

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

from sklearn.preprocessing import MinMaxScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import f1\_score

from sklearn.neighbors import KNeighborsClassifier

df = pd.read\_csv("report.csv")

filter\_data1=df[(df['TARGET']==0) & (df['MIP']>=68.1640625)& (df['MIP']<= 172.5625)]

filter\_data2=df[(df['TARGET']==1) & (df['MIP']>=6.4140625) & (df['MIP']<= 113.6484375)]

filter\_data = filter\_data1.append(filter\_data2)

#1

print("Mean: ", filter\_data['STDIP'].mean())

#2

data\_sorted = filter\_data.sort\_values(by='SIP', ascending=True)

X = data\_sorted.drop('TARGET', axis=1)

y = data\_sorted['TARGET']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.2, random\_state=23, stratify=y)

print("среднее для столбца SIP из тренировочной выборки: ", X\_train['SIP'].mean())

#3

scaler = MinMaxScaler()

X\_train = pd.DataFrame(scaler.fit\_transform(X\_train), columns=X\_train.columns)

X\_test = pd.DataFrame(scaler.transform(X\_test), columns=X\_test.columns)

print("выборочное среднее для столбца STDIP из тренировочной выборки: ", X\_train['STDIP'].mean())

#4

logreg\_model=LogisticRegression(random\_state=23)

logreg\_model.fit(X\_train, y\_train)

y\_pred = logreg\_model.predict(X\_test)

conf\_matrix=confusion\_matrix(y\_test, y\_pred)

#True Positive

tp\_value=conf\_matrix[1,1]

print(tp\_value)

#False Nefative

fn\_value=conf\_matrix[1,0]

print(fn\_value)

#f1

print(f1\_score(y\_test,y\_pred))

# модель KNeighborsClassifier(n\_neighbors=3)

knn\_model=KNeighborsClassifier(n\_neighbors=3)

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

y\_pred\_knn=knn\_model.predict(X\_test)

conf\_matrix\_knn=confusion\_matrix(y\_test, y\_pred\_knn)

#True Positive

tp\_value=conf\_matrix\_knn[1,1]

print(tp\_value)

#False Nefative

fn\_value=conf\_matrix\_knn[0,1]

print(fn\_value)

#f1

print(f1\_score(y\_test,y\_pred\_knn))