Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №6 з дисципліни
«Аналіз даних в інформаційних системах»

„Класифікація та кластеризація”

Виконав(ла) *ІП-11 Тарасьонок Дмитро Євгенович*
(шифр, прізвище, ім'я, по батькові)

Перевірила *Ліхоузова Т. А.*
(прізвище, ім'я, по батькові)

Київ 2023

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – ознайомитись з методами класифікації та кластеризації; моделями, що використовують дерева прийняття рішень; інструментами факторного аналізу методом головних компонент та методом найбільшої подібності.

2 ЗАВДАННЯ

2.1 Основне завдання

Для даних по титаніку titanic.csv побудувати модель, в якій можна визначити, чи виживе пасажир, заповнивши решту параметрів.

Використати декілька методів. Порівняти результати.

2.2 Додаткове завдання

Використовуючи файл Data2.csv

1. визначити, який регіон домінує в кластерах по ВВП на душу населення та щільності населення
2. вивести частотні гістограми всіх показників файла Data2.csv, використовуючи цикл
3. створити функцію, яка на вхід отримує два набори даних, перевіряє чи є лінійна залежність та виводить True чи False (будемо розуміти під «є лінійна залежність», якщо коефіцієнт кореляції по модулю більше 0,8)

3 ВИКОНАННЯ ОСНОВНОГО ЗАВДАННЯ

Для початку завантажимо дані, перевіримо, чи є серед них пропущені.

```
In 2
titanic = pd.read_csv('titanic.csv', index_col=0)
titanic
```

Out 2 — 891 rows × 11 columns pd.Dataframe

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...
887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

```
In 3
titanic.isna().any()
```

Out 3

```
Survived    False
Pclass      False
Name        False
Sex         False
Age          True
SibSp       False
Parch       False
Ticket      False
Fare        False
Cabin        True
Embarked     True
dtype: bool
```

Рис. 3.1 – Завантаження даних, перевірка на пропущені значення

Бачимо, що є пропущені значення в стовпцях «Cabin», «Embarked» та «Age». Відкинемо рядкові стовпці, які ні на що не впливають.

```
In 4
titanic.drop(['Name', 'Ticket', 'Cabin'], axis=1, inplace=True)
titanic
```

Executed in 95ms, 7 Apr at 01:27:07

Out 4 — 1-36 — 891 rows × 8 columns pd.Dataframe

PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
1	0	3	male	22.0	1	0	7.2500	S
2	1	1	female	38.0	1	0	71.2833	C
3	1	3	female	26.0	0	0	7.9250	S
4	1	1	female	35.0	1	0	53.1000	S
5	0	3	male	35.0	0	0	8.0500	S
6	0	3	male	NaN	0	0	8.4583	Q
7	0	1	male	54.0	0	0	51.8625	S
8	0	3	male	2.0	3	1	21.0750	S
9	1	3	female	27.0	0	2	11.1333	S
10	1	2	female	14.0	1	0	30.0708	C
11	1	3	female	4.0	1	1	16.7000	S
12	1	1	female	58.0	0	0	26.5500	S
13	0	3	male	20.0	0	0	8.0500	S

Рис. 3.2 – Відкидання зайвих стовпців

Тепер перевіримо, скільки саме значень пропущено в стовпці «Embarked».

```
In 5  1  titanic.Embarked.isna().sum()
         Executed in 147ms, 7 Apr at 01:27:07

Out 5    2
```

Рис. 3.3 – Кількість пропущених значень у стовпці «Embarked»

Бачимо, що таких усього 2, тому пропонується заповнити їх тим, що найчастіше повторюється. Виведемо кількість повторів кожного зі значень.

```
In 6  1  titanic.Embarked.value_counts()
         Executed in 133ms, 7 Apr at 01:27:07
```

Out 6 — 3 rows — Length: 3, dtype: int64 pd.Series

	Embarked
S	644
C	168
Q	77

Рис. 3.4 – Кількість кожного значення в стовпці «Embarked»

Найчастіше повторюється значення «S», а отже заповнимо пропуски таким значенням.

```
In 7  1  titanic.Embarked.fillna(titanic.Embarked.value_counts().idxmax(), inplace=True)
      2  titanic
         Executed in 117ms, 7 Apr at 01:27:07
```

Out 7 — 1-36 — 891 rows × 8 columns pd.Dataframe

PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
1	0	3	male	22.0	1	0	7.2500	S
2	1	1	female	38.0	1	0	71.2833	C
3	1	3	female	26.0	0	0	7.9250	S
4	1	1	female	35.0	1	0	53.1000	S
5	0	3	male	35.0	0	0	8.0500	S
6	0	3	male	NaN	0	0	8.4583	Q
7	0	1	male	54.0	0	0	51.8625	S
8	0	3	male	2.0	3	1	21.0750	S
9	1	3	female	27.0	0	2	11.1333	S
10	1	2	female	14.0	1	0	30.0708	C
11	1	3	female	4.0	1	1	16.7000	S
12	1	1	female	58.0	0	0	26.5500	S
13	0	3	male	20.0	0	0	8.0500	S

Рис. 3.5 – Заповнення пропущених значень стовпця «Embarked»

Далі займемося заповненням пропусків віку. Можемо припустити, що для чоловіків та жінок середній вік є різним, а отже обчислимо його.

```
In 8  1  mean_ages = titanic.groupby('Sex').Age.mean()
      2  mean_ages
```
Executed in 86ms, 7 Apr at 01:27:07

Out 8 — 2 rows — Length: 2, dtype: float64 pd.Series

Sex	Age
female	27.915709
male	30.726645

Рис. 3.6 – Розрахунок середнього віку в залежності від статі

Припущення справдилося, а тому заповнюватимемо пропуски в залежності від віку. Наостанок перевіримо, чи залишилися десь пропущені значення.

```
In 9  1  def fill_age(row: pd.Series):
      2      if row.Age != row.Age:
      3          row.Age = mean_ages[row.Sex]
      4      return row
```
Executed in 71ms, 7 Apr at 01:27:07

Рис. 3.7 – Функція заповнення пропусків у стовпці віку

```
In 10  1  titanic = titanic.apply(fill_age, axis=1)
       2  titanic.isna().any()
Executed in 240ms, 7 Apr at 01:27:07
```

Out 10 — 8 rows — Length: 8, dtype: bool pd.Series

	<unnamed>
Survived	False
Pclass	False
Sex	False
Age	False
SibSp	False
Parch	False
Fare	False
Embarked	False

Рис. 3.8 – Заповнення пропущених значень у стовпці віку, перевірка на наявність пропущених значень у наборі даних

Далі, оскільки в нас є категоріальні значення («Sex», «Embarked»), використаємо метод pandas.get_dummies(), який перетворить ці значення в матрицю з бінарними змінними.

```
In 11  1  titanic = pd.get_dummies(titanic, columns=['Sex', 'Embarked'])
       2  titanic
Executed in 222ms, 7 Apr at 01:27:07
```

Out 11 — 1-36 — 891 rows × 11 columns pd.Dataframe

PassengerId	Survived	Pclass	Age	Fare	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
1	0	3	22.000000	7.2500	0	1	0	0	1
2	1	1	38.000000	71.2833	1	0	1	0	0
3	1	3	26.000000	7.9250	1	0	0	0	1
4	1	1	35.000000	53.1000	1	0	0	0	1
5	0	3	35.000000	8.0500	0	1	0	0	1
6	0	3	30.726645	8.4583	0	1	0	1	0
7	0	1	54.000000	51.8625	0	1	0	0	1
8	0	3	2.000000	21.0750	0	1	0	0	1
9	1	3	27.000000	11.1333	1	0	0	0	1
10	1	2	14.000000	30.0708	1	0	1	0	0
11	1	3	4.000000	16.7000	1	0	0	0	1
12	1	1	58.000000	26.5500	1	0	0	0	1
13	0	3	20.000000	8.0500	0	1	0	0	1

Рис. 3.9 – Перетворення категоріальних значень на матрицю з бінарними змінними

Після цього відобразимо теплову карту кореляції.

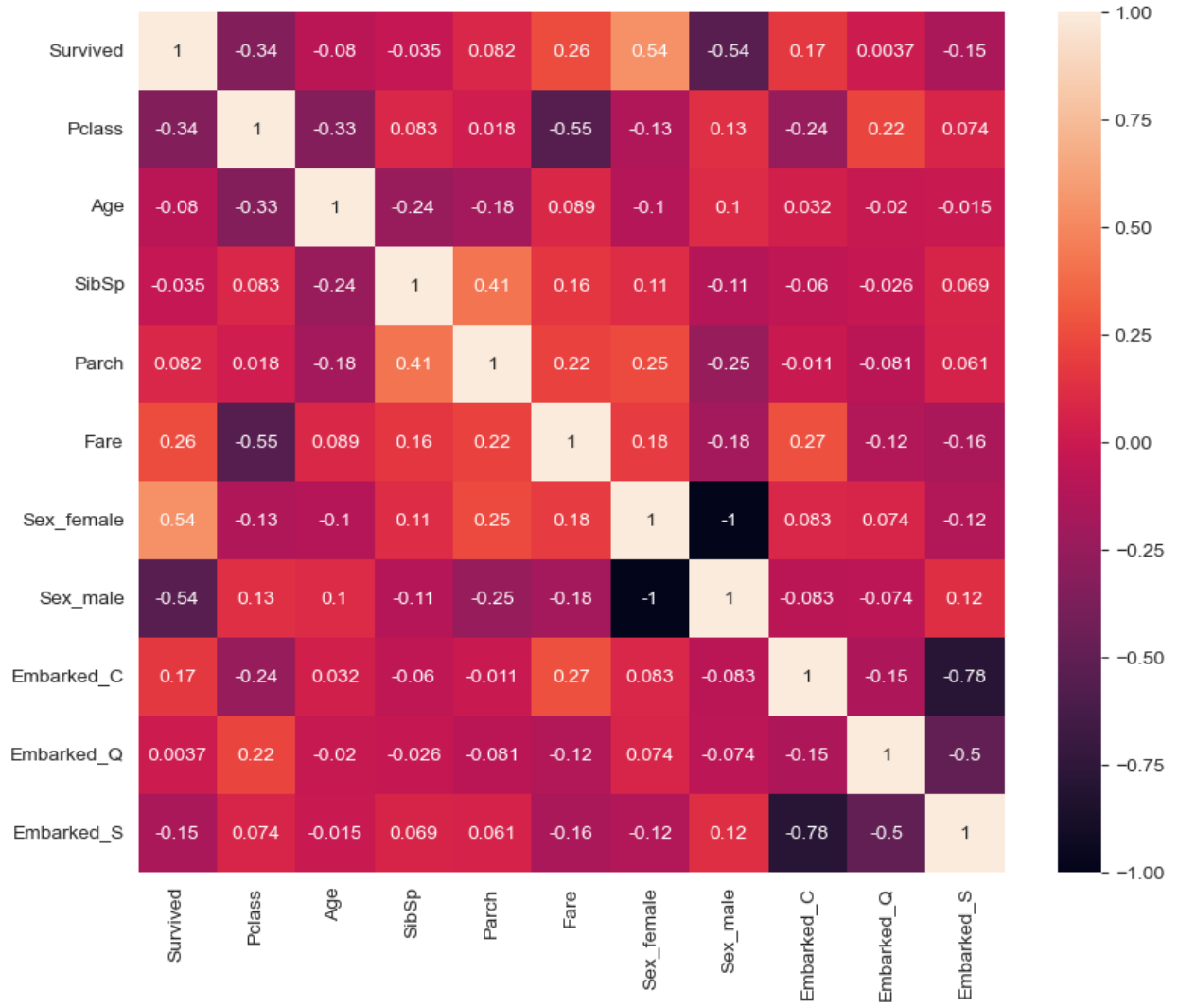

Рис. 3.10 – Теплова карта кореляції

Бачимо, що стовпці «SibSp», «Parch» мало впливають на те, чи виживе пасажир (коефіцієнти кореляції -0.035 та 0.082 відповідно), а отже їх можна відкинути.

```
In 13  1 titanic.drop(['SibSp', 'Parch'], axis=1, inplace=True)
       2 titanic
Executed in 18ms, 7 Apr at 01:27:08
```

Out 13 — 1-36 — 891 rows × 9 columns pd.Dataframe

PassengerId	Survived	Pclass	Age	Fare	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
1	0	3	22.000000	7.2500	0	1	0	0	1
2	1	1	38.000000	71.2833	1	0	1	0	0
3	1	3	26.000000	7.9250	1	0	0	0	1
4	1	1	35.000000	53.1000	1	0	0	0	1
5	0	3	35.000000	8.0500	0	1	0	0	1
6	0	3	30.726645	8.4583	0	1	0	1	0
7	0	1	54.000000	51.8625	0	1	0	0	1
8	0	3	2.000000	21.0750	0	1	0	0	1
9	1	3	27.000000	11.1333	1	0	0	0	1
10	1	2	14.000000	30.0708	1	0	1	0	0
11	1	3	4.000000	16.7000	1	0	0	0	1
12	1	1	58.000000	26.5500	1	0	0	0	1
13	0	3	20.000000	8.0500	0	1	0	0	1

Рис. 3.11 – Видалення незначущих стовпців

Далі розділимо дані на навчальну та тестову вибірки в пропорції 75% на 25%.

```
In 14  1  X_train, X_test, y_train, y_test = train_test_split(titanic.drop('Survived', axis=1), titanic.Survived)
          Executed in 15ms, 7 Apr at 01:27:08
```

Рис. 3.12 – Розділення набору даних на навчальну та тестову вибірки

Після того, як усі описані дії проведено, можемо приступити до процесу класифікації. Використовуватимуться наступні методи: метод K-найближчих сусідів, дерево рішень та випадкові ліси. Для кожного з цих методів за допомогою крос-валідації підберемо найкращі гіперпараметри. Почнемо з методу K-найближчих сусідів.

```
In 15  1  knn_cv = GridSearchCV(KNeighborsClassifier(), param_grid={'n_neighbors': range(1, 101)}, cv=5, n_jobs=-1)
       2  knn_cv.fit(X_train, y_train)
       3  knn_cv.best_score_
          Executed in 6s, 7 Apr at 01:27:14

Out 15    0.7246661429693637
```

Рис. 3.13 – Крос-валідація для знаходження гіперпараметру K методу K-найближчих сусідів

Бачимо, що оцінка на навчальній вибірці склала 0.724, накреслимо ці оцінки на графіку залежності оцінки від кількості сусідів.

```
In 16
_, axes = plt.subplots()
axes.set_xlabel('K')
axes.set_ylabel('mean_test_score')
axes.set_title('Dependence of K-Neighbors Classifier accuracy on hyperparameter K')
sns.lineplot(x=range(1, 101), y=knn_cv.cv_results_['mean_test_score'], ax=axes);
Executed in 295ms, 7 Apr at 01:27:14
```

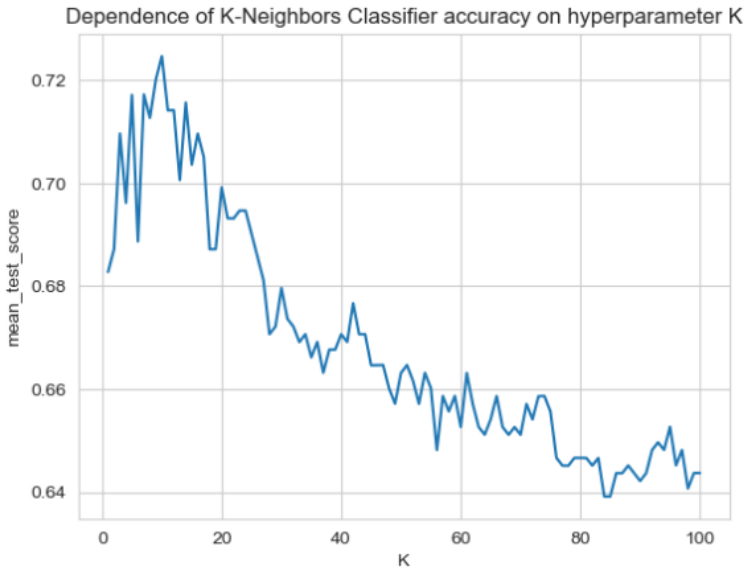

Рис. 3.14 – Графік залежності точності методу від гіперпараметру K

Далі визначимо середньоквадратичну похибку на тестовій вибірці.

```
In 17
knn_score = mean_squared_error(knn_cv.predict(X_test), y_test)
knn_score
Executed in 50ms, 7 Apr at 01:27:14
```

```
Out 17
0.26905829596412556
```

Рис. 3.15 – Середньоквадратична похибка на тестовій вибірці для методу K-найближчих сусідів

Вона склала 0.269, переходимо до методу дерев рішень.

```
In 18
dtc_cv = GridSearchCV(DecisionTreeClassifier(), param_grid={'max_depth': range(1, 101)})
dtc_cv.fit(X_train, y_train)
dtc_cv.best_score_
Executed in 2s, 7 Apr at 01:27:17
```

```
Out 18
0.8143081584558411
```

Рис. 3.16 – Крос-валідація для підбору значення максимальної глибини дерева рішень

Тут оцінка склала вже 0.814, накреслимо її на графіку залежності оцінки від максимальної глибини.

```
In 19
_, axes = plt.subplots()
axes.set_xlabel('max_depth')
axes.set_ylabel('mean_test_score')
axes.set_title('Dependence of Decision Tree Classifier accuracy on hyperparameter max_depth')
sns.lineplot(x=range(1, 101), y=dtc_cv.cv_results_['mean_test_score'], ax=axes);
Executed in 264ms, 7 Apr at 01:27:17
```

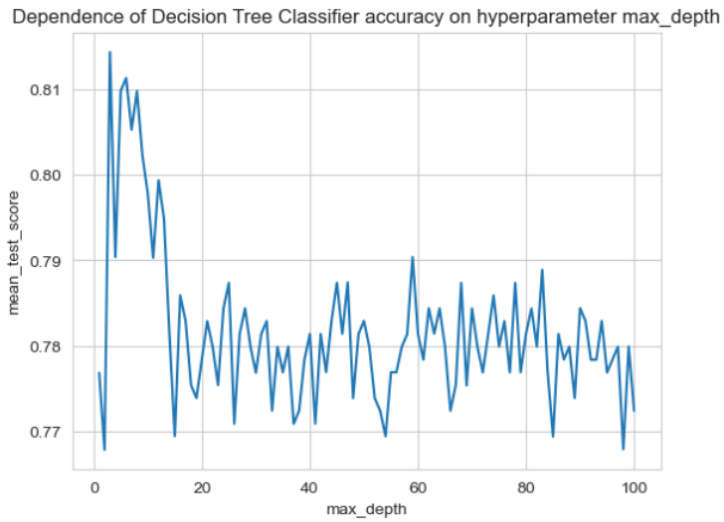

Рис. 3.17 – Графік залежності точності дерева прийняття рішень від максимальної глибини

Тепер обрахуємо середньоквадратичну похибку для цього методу.

```
In 20
dtc_score = mean_squared_error(dtc_cv.predict(X_test), y_test)
dtc_score
Executed in 49ms, 7 Apr at 01:27:17
```

```
Out 20
0.18834080717488788
```

Рис. 3.18 – Середньоквадратична похибка на тестовій вибірці для дерева рішень

Вона склала 0.188, що є доволі хорошим результатом. Перейдемо до випадкових лісів.

```
In 21
rfc_cv = GridSearchCV(RandomForestClassifier(), param_grid={'n_estimators': range(1, 101)})
rfc_cv.fit(X_train, y_train)
rfc_cv.best_score_
Executed in 43s, 7 Apr at 01:28:00
```

```
Out 21
0.8128717315677253
```

Рис. 3.19 – Крос-валідація для підбору гіперпараметру кількості дерев у методі випадкових лісів

Тут отримали оцінку на навчальній вибірці 0.812. Відобразимо залежність оцінки від кількості дерев на графіку.

```
In 22
_, axes = plt.subplots()
axes.set_xlabel('n_estimators')
axes.set_ylabel('mean_test_score')
axes.set_title('Dependence of the accuracy of the random forest method on the number of trees')
sns.lineplot(x=range(1, 101), y=rfc_cv.cv_results_['mean_test_score'], ax=axes);
Executed in 307ms, 7 Apr at 01:28:00
```

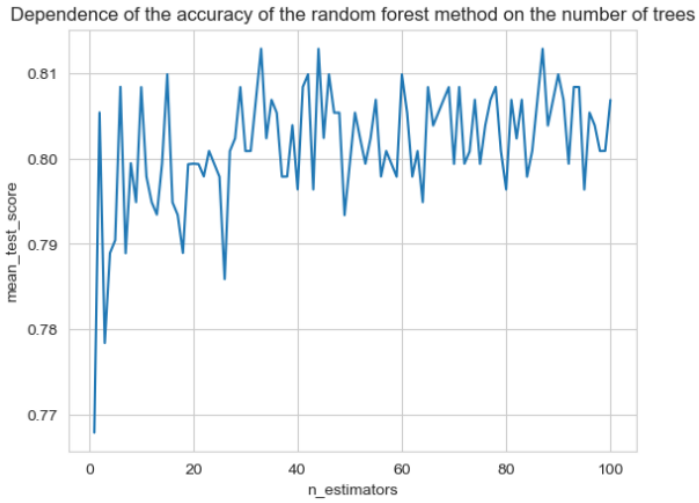

Рис. 3.20 – Залежність точності методу випадкових лісів від кількості дерев

Тепер обрахуємо середньоквадратичну похибку для випадкових лісів.

```
In 23
rfc_score = mean_squared_error(rfc_cv.predict(X_test), y_test)
rfc_score
Executed in 126ms, 7 Apr at 01:28:01

Out 23    0.17040358744394618
```

Рис. 3.21 – Середньоквадратична похибка на тестовій вибірці для методу випадкових лісів

Вона склала 0.170. Відобразимо стовпчасту діаграму середньоквадратичної похибки для кожного методу.

```
In 24  1  sns.barplot(
       2      x=['K-Nearest Neighbors', 'Decision Tree', 'Random Forest'],
       3      y=[knn_score, dtc_score, rfc_score]
       4  );
```

Executed in 201ms, 7 Apr at 01:28:01

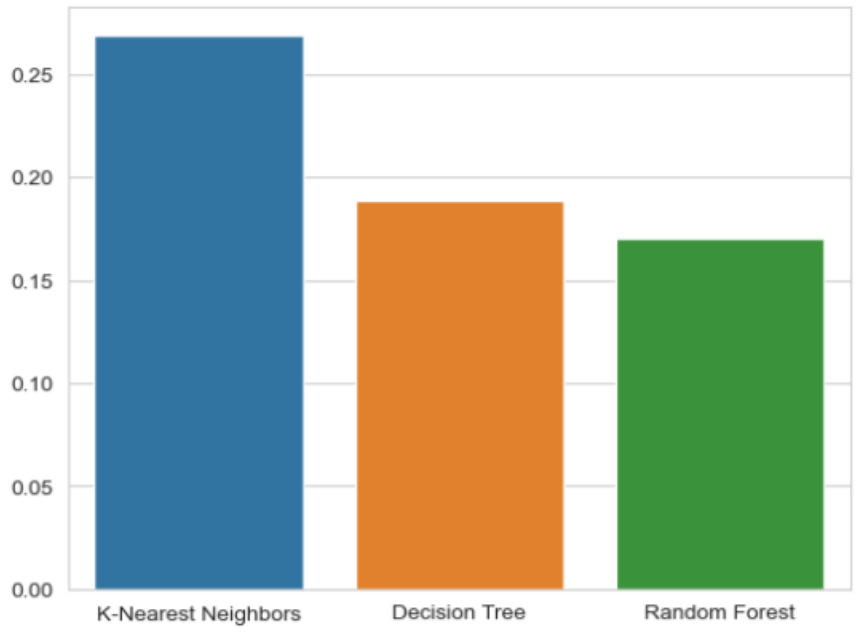

Рис. 3.22 – Стовпчаста діаграма середньоквадратичної помилки кожного з методів

Перевіривши різні методи, я дійшов до висновку, що найкращим на заданих даних є метод випадкових лісів із середньоквадратичною похибкою 0.179, що є доволі хорошим результатом.

4 ВИКОНАННЯ ДОДАТКОВОГО ЗАВДАННЯ

4.1 Визначити, який регіон домінує в кластерах по ВВП на душу населення та щільності населення

Для початку проведемо все ті ж дії з цим набором, що й у попередніх лабораторних роботах: завантажимо дані, виправимо помилки, а також додамо стовпець щільності населення.

In 2
```
df = pd.read_csv('Data2.csv', on_bad_lines='skip', encoding='cp1252', delimiter=';')
df.columns = ['country_name', 'region', 'gdp_per_capita', 'population', 'co2_emission', 'area']
df.set_index('country_name', inplace=True)
df.gdp_per_capita = df.gdp_per_capita.astype(str).str.replace(',', '.').astype(float)
df.co2_emission = df.co2_emission.astype(str).str.replace(',', '.').astype(float)
df.area = df.area.astype(str).str.replace(',', '.').astype(float)
df.gdp_per_capita = df.gdp_per_capita.abs()
df.area = df.area.abs()
df.gdp_per_capita.fillna(df.gdp_per_capita.mean(), inplace=True)
df.population.fillna(df.population.mean(), inplace=True)
df.co2_emission.fillna(df.co2_emission.mean(), inplace=True)
df.population = df.population.astype(int)
df['density'] = df.population / df.area
df
```
Executed in 39ms, 7 Apr at 01:27:28

Out 2 — 1-36 — 217 rows × 6 columns pd.Dataframe

country_name	region	gdp_per_capita	population	co2_emission	area	density
Afghanistan	South Asia	561.778746	34656032	9809.225000	652860.0	53.083405
Albania	Europe & Central Asia	4124.982390	2876101	5716.853000	28750.0	100.038296
Algeria	Middle East & North Africa	3916.881571	40606052	145400.217000	2381740.0	17.048902
American Samoa	East Asia & Pacific	11834.745230	55599	165114.116337	200.0	277.995000
Andorra	Europe & Central Asia	36988.622030	77281	462.042000	470.0	164.427660
Angola	Sub-Saharan Africa	3308.700233	28813463	34763.160000	1246700.0	23.111786
Antigua and Barbuda	Latin America & Caribbean	14462.176280	100963	531.715000	440.0	229.461364
Argentina	Latin America & Caribbean	12440.320980	43847430	204024.546000	2780400.0	15.770188
Armenia	Europe & Central Asia	3614.688357	2924816	5529.836000	29740.0	98.346200
Aruba	Latin America & Caribbean	13445.593416	104822	872.746000	180.0	582.344444
Australia	East Asia & Pacific	49755.315480	24127159	361261.839000	7741220.0	3.116713
Austria	Europe & Central Asia	44757.634900	8747358	58712.337000	83879.0	104.285435
Azerbaijan	Europe & Central Asia	3878.709257	9762274	37487.741000	86600.0	112.728337

Рис. 4.1 – Завантаження даних та виправлення помилок

Далі виокремимо ознаки, за якими й будемо проводити кластеризацію: ВВП на душу населення.

```
In 3  1  gdp_features = df['gdp_per_capita'].to_numpy().reshape(-1, 1)
      2  gdp_features
Executed in 40ms, 7 Apr at 01:27:29
```

Out 3

1-36 | 217 rows × 1 columns | np.ndarray

	0
0	561.778746
1	4124.982390
2	3916.881571
3	11834.745230
4	36988.622030
5	3308.700233
6	14462.176280
7	12440.320980
8	3614.688357
9	13445.593416
10	49755.315480
11	44757.634900
12	3878.709257
13	28785.477670

Рис. 4.2 – Виділення ознак для кластеризації за ВВП на душу населення

Після цього необхідно знайти точку локтя для моделі кластеризації методу K-середніх. Визначимо максимальну кількість кластерів як 13 та обрахуємо для кожної кількості кластерів інерцію.

In 4

```
max_clusters = 13
n_clusters_list = list(range(1, max_clusters + 1))
sse = []

for n_clusters in n_clusters_list:
    kmeans = KMeans(n_clusters=n_clusters, n_init=10)
    kmeans.fit(gdp_features)
    sse.append(kmeans.inertia_)

gdp_kmeans_results = pd.DataFrame.from_dict({
    'n_clusters': n_clusters_list,
    'sse': sse
}).reset_index(drop=True).set_index('n_clusters')
gdp_kmeans_results
```

Executed in 2s, 7 Apr at 01:27:31

Out 4

13 rows | 13 rows × 1 columns pd.Dataframe

n_clusters	sse
1	6.150152e+10
2	1.580377e+10
3	9.052527e+09
4	4.492411e+09
5	2.984519e+09
6	1.933117e+09
7	1.430037e+09
8	9.406344e+08
9	7.658671e+08
10	6.027662e+08
11	4.400069e+08
12	3.331458e+08
13	2.880841e+08

Рис. 4.3 – Обрахунок інерції для різної кількості кластерів

```
In 5  1  gdp_kmeans_results.plot();
Executed in 280ms, 7 Apr at 01:27:31
```

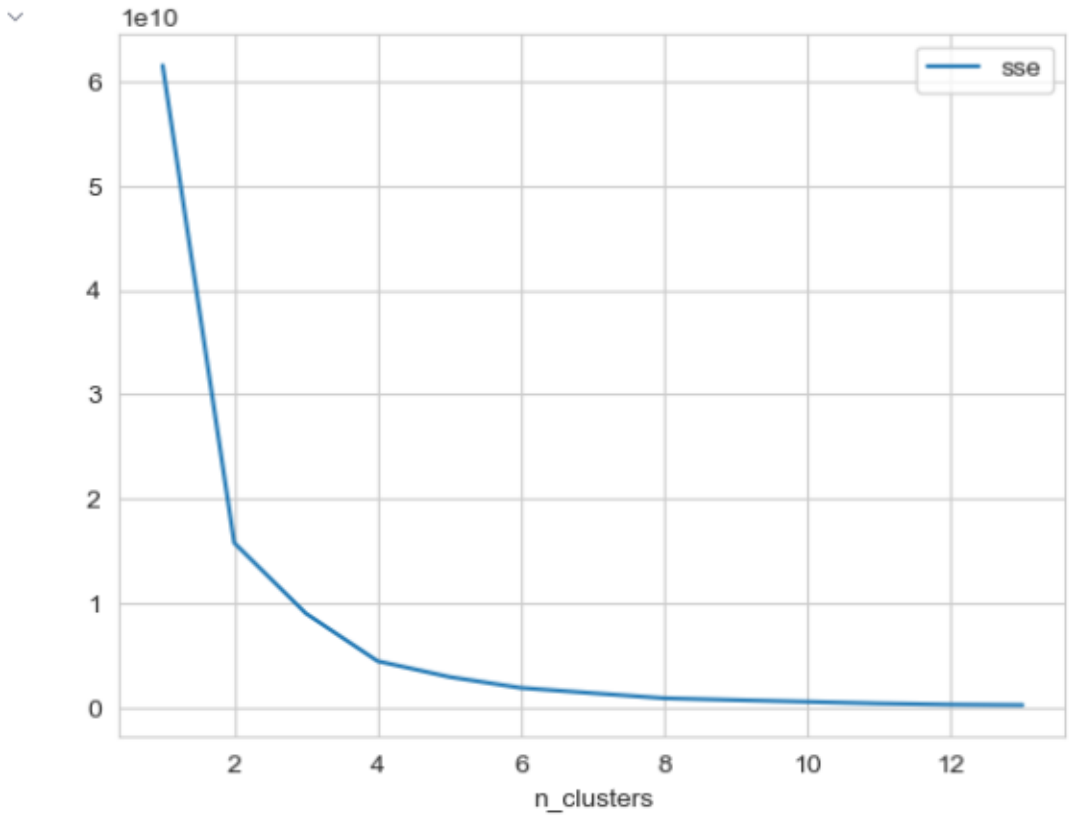

Рис. 4.4 – Графік залежності інерції від кількості кластерів

Обрахувавши інерцію, можемо знайти точку локтя.

```
In 6  1  gdp_kl = KneeLocator(n_clusters_list, sse, curve='convex', direction='decreasing')
      2  gdp_kl.elbow
Executed in 13ms, 7 Apr at 01:27:32

Out 6    3
```

Рис. 4.5 – Знаходження точки локтя

Точка локтя дорівнює трьом, а отже використовуватимемо для цієї моделі 3 кластери. Навчимо цю модель та виведемо номери кластерів для кожного значення.

```
In 7  1  gdp_kmeans = KMeans(n_clusters=gdp_kl.elbow, n_init=10)
      2  gdp_kmeans.fit(gdp_features)
```
Executed in 200ms, 7 Apr at 01:27:32

Out 7

```
KMeans(n_clusters=3, n_init=10)
```

```
In 8  1  gdp_kmeans.labels_
```
Executed in 13ms, 7 Apr at 01:27:32

Out 8 — 1-36 — 217 rows × 1 columns np.ndarray

	0
0	2
1	2
2	2
3	0
4	1
5	2
6	0
7	0
8	2
9	0
10	1
11	1
12	2
13	0

Рис. 4.6 – Навчання методу K-середніх та виведення міток кластеру для кожного значення

Підрахуємо кількість ознак у кожному з кластерів.

```
In 9  1  _, gdp_countries_count = np.unique(gdp_kmeans.labels_, return_counts=True)
      2  gdp_countries_count
```
Executed in 27ms, 7 Apr at 01:27:32

Out 9 — 3 rows — 3 rows × 1 columns np.ndarray

	0
0	66
1	28
2	123

Рис. 4.7 – Кількість ознак у кожному кластері

Можемо вивести перелік країн із кожного кластеру.

```
In 10  1  first_gdp_cluster = df[gdp_kmeans.labels_ == 0]
       2  first_gdp_cluster
```
Executed in 13ms, 7 Apr at 01:27:32

Out 10 — 1-36 — 66 rows × 6 columns pd.Dataframe

country_name	region	gdp_per_capita	population	co2_emission	area	density
American Samoa	East Asia & Pacific	11834.745230	55599	165114.116337	200.0	277.995000
Antigua and Barbuda	Latin America & Caribbean	14462.176280	100963	531.715000	440.0	229.461364
Argentina	Latin America & Caribbean	12440.320980	43847430	204024.546000	2780400.0	15.770188
Aruba	Latin America & Caribbean	13445.593416	104822	872.746000	180.0	582.344444
Bahamas, The	Latin America & Caribbean	28785.477670	391232	2416.553000	13880.0	28.186744
Bahrain	Middle East & North Africa	22579.093420	1425171	31338.182000	771.0	1848.470817
Barbados	Latin America & Caribbean	15891.626550	284996	1272.449000	430.0	662.781395
Bermuda	North America	13445.593416	65331	575.719000	50.0	1306.620000
British Virgin Islands	Latin America & Caribbean	13445.593416	30661	179.683000	150.0	204.406667
Brunei Darussalam	East Asia & Pacific	26939.417510	423196	9108.828000	5770.0	73.344194
Cayman Islands	Latin America & Caribbean	13445.593416	60765	542.716000	264.0	230.170455
Channel Islands	Europe & Central Asia	13445.593416	164541	165114.116337	190.0	866.005263
Chile	Latin America & Caribbean	13792.926050	17909754	82562.505000	756096.0	23.687143

Рис. 4.8 – Перший кластер

```
In 11  1  second_gdp_cluster = df[gdp_kmeans.labels_ == 1]
       2  second_gdp_cluster
```
Executed in 71ms, 7 Apr at 01:27:32

Out 11 — 28 rows — 28 rows × 6 columns pd.Dataframe

country_name	region	gdp_per_capita	population	co2_emission	area	density
Andorra	Europe & Central Asia	36988.62203	77281	4.620420e+02	470.0	164.427660
Australia	East Asia & Pacific	49755.31548	24127159	3.612618e+05	7741220.0	3.116713
Austria	Europe & Central Asia	44757.63490	8747358	5.871234e+04	83879.0	104.285435
Belgium	Europe & Central Asia	41271.48215	11348159	9.335082e+04	30530.0	371.705175
Canada	North America	42183.29510	36286425	5.371935e+05	9984670.0	3.634214
Denmark	Europe & Central Asia	53578.75657	5731118	3.349804e+04	42922.0	133.524020
Finland	Europe & Central Asia	43401.22834	5495096	4.730063e+04	338420.0	16.237504
France	Europe & Central Asia	36857.11923	66896109	3.032756e+05	549087.0	121.831529
Germany	Europe & Central Asia	42161.31966	82667685	7.198834e+05	357380.0	231.315924
Guam	East Asia & Pacific	35562.56753	162896	1.651141e+05	540.0	301.659259
Hong Kong SAR, China	East Asia & Pacific	43740.99288	7346700	4.622254e+04	1105.0	6648.597285
Iceland	Europe & Central Asia	59764.70538	334252	1.983847e+03	103000.0	3.245165
Ireland	Europe & Central Asia	64175.43824	4773095	3.406643e+04	70280.0	67.915410

Рис. 4.9 – Другий кластер

```
In 12  1  third_gdp_cluster = df[gdp_kmeans.labels_ == 2]
       2  third_gdp_cluster
```
Executed in 138ms, 7 Apr at 01:27:32

Out 12 — 1-36 — 123 rows × 6 columns pd.Dataframe

country_name	region	gdp_per_capita	population	co2_emission	area	density
Afghanistan	South Asia	561.778746	34656032	9809.225	652860.0	53.083405
Albania	Europe & Central Asia	4124.982390	2876101	5716.853	28750.0	100.038296
Algeria	Middle East & North Africa	3916.881571	40606052	145400.217	2381740.0	17.048902
Angola	Sub-Saharan Africa	3308.700233	28813463	34763.160	1246700.0	23.111786
Armenia	Europe & Central Asia	3614.688357	2924816	5529.836	29740.0	98.346200
Azerbaijan	Europe & Central Asia	3878.709257	9762274	37487.741	86600.0	112.728337
Bangladesh	South Asia	1358.779029	162951560	73189.653	147630.0	1103.783513
Belarus	Europe & Central Asia	4989.427763	9507120	63497.772	207600.0	45.795376
Belize	Latin America & Caribbean	4744.736397	366954	495.045	22970.0	15.975359
Benin	Sub-Saharan Africa	789.440411	10872298	6318.241	114760.0	94.739439
Bhutan	South Asia	2773.547135	797765	1001.091	38394.0	20.778377
Bolivia	Latin America & Caribbean	3104.956089	10887882	20410.522	1098580.0	9.910869
Bosnia and Herzegovina	Europe & Central Asia	4808.405425	3516816	22233.021	51210.0	68.674400

Рис. 4.10 – Третій кластер

Після цього можемо вивести максимальний ВВП на душу населення в кожному регіоні в кожному кластері.

```
In 13  1  first_gdp_cluster.groupby('region').gdp_per_capita.max().sort_values(ascending=False)
Executed in 108ms, 7 Apr at 01:27:32
```

Out 13 — 7 rows — Length: 7, dtype: float64 pd.Series

region	gdp_per_capita
Latin America & Caribbean	30790.104790
Europe & Central Asia	30661.221810
East Asia & Pacific	27538.806130
Middle East & North Africa	27359.230330
Sub-Saharan Africa	15075.719440
North America	13445.593416
South Asia	9875.278428

Рис. 4.11 – Максимальний ВВП на душу населення за регіонами в першому кластері

```
In 14  1  second_gdp_cluster.groupby('region').gdp_per_capita.max().sort_values(ascending=False)
Executed in 113ms, 7 Apr at 01:27:32
```

Out 14 — 4 rows — Length: 4, dtype: float64 pd.Series

region	gdp_per_capita
Europe & Central Asia	100738.68420
East Asia & Pacific	74017.18471
Middle East & North Africa	59324.33877
North America	57638.15909

Рис. 4.12 – Максимальний ВВП на душу населення за регіонами в другому кластері

```
In 15  1  third_gdp_cluster.groupby('region').gdp_per_capita.max().sort_values(ascending=False)
Executed in 242ms, 7 Apr at 01:27:32
```

Out 15 — 6 rows — Length: 6, dtype: float64 pd.Series

region	gdp_per_capita
Sub-Saharan Africa	9630.944028
Europe & Central Asia	9522.771041
East Asia & Pacific	9508.237750
Latin America & Caribbean	9364.821525
Middle East & North Africa	8257.294391
South Asia	3909.989066

Рис. 4.13 – Максимальний ВВП на душу населення за регіонами в третьому кластері

Бачимо, що в першому кластері домінує Латинська Америка та Кариби, у другому – Південна Азія, а в третьому – Європа та Центральна Азія. Далі проведемо ці ж дії для щільності населення.

```
In 16  1  density_features = df['density'].to_numpy().reshape(-1, 1)
       2  density_features
```

Executed in 236ms, 7 Apr at 01:27:32

Out 16

1-36 217 rows × 1 columns np.ndarray

	0
0	53.083405
1	100.038296
2	17.048902
3	277.995000
4	164.427660
5	23.111786
6	229.461364
7	15.770188
8	98.346200
9	582.344444
10	3.116713
11	104.285435
12	112.728337
13	28.186744

Рис. 4.14 – Виділення ознак для кластеризації за щільністю населення

In 17

```
max_clusters = 13
n_clusters_list = list(range(1, max_clusters + 1))
sse = []

for n_clusters in n_clusters_list:
    kmeans = KMeans(n_clusters=n_clusters, n_init=10)
    kmeans.fit(density_features)
    sse.append(kmeans.inertia_)

density_kmeans_results = pd.DataFrame.from_dict({
    'n_clusters': n_clusters_list,
    'sse': sse
}).reset_index(drop=True).set_index('n_clusters')
density_kmeans_results
```

Executed in 2s, 7 Apr at 01:27:35

Out 17

13 rows 13 rows × 1 columns pd.Dataframe

n_clusters	sse
1	8.750913e+08
2	1.238429e+08
3	2.474531e+07
4	1.014364e+07
5	5.840313e+06
6	2.994677e+06
7	2.103726e+06
8	1.442330e+06
9	9.872417e+05
10	7.072497e+05
11	4.518318e+05
12	3.021762e+05
13	2.414094e+05

Рис. 4.15 – Обрахунок інерції для різної кількості кластерів

```
In 18  1  density_kmeans_results.plot();
```
Executed in 234ms, 7 Apr at 01:27:35

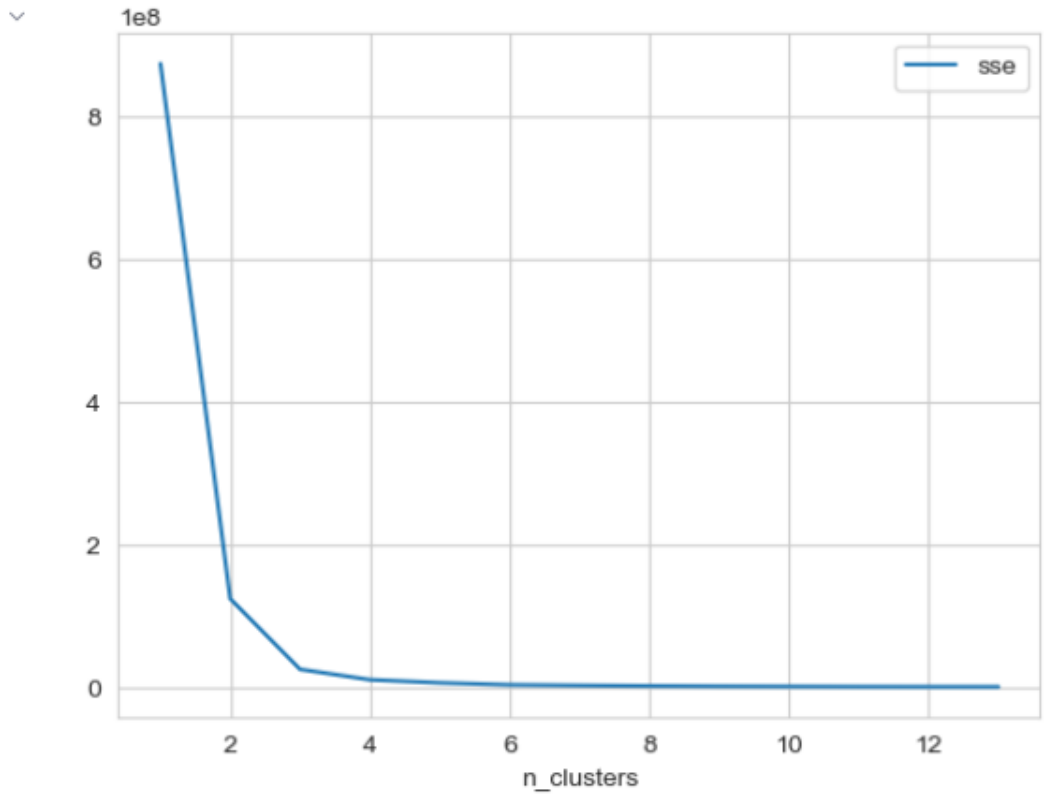

Рис. 4.16 – Графік залежності інерції від кількості кластерів

```
In 19  1  density_kl = KneeLocator(n_clusters_list, sse, curve='convex', direction='decreasing')
       2  density_kl.elbow
```
Executed in 23ms, 7 Apr at 01:27:35

```
Out 19    3
```

Рис. 4.17 – Точка локтя

Бачимо, що точка локтя знову дорівнює трьом, а отже знову використовуватимемо 3 кластери.

```
In 20  1  density_kmeans = KMeans(n_clusters=density_kl.elbow, n_init=10)
       2  density_kmeans.fit(density_features)
```
Executed in 197ms, 7 Apr at 01:27:35

```
Out 20    KMeans
          KMeans(n_clusters=3, n_init=10)
```

Рис. 4.18 – Навчання методу K-середніх

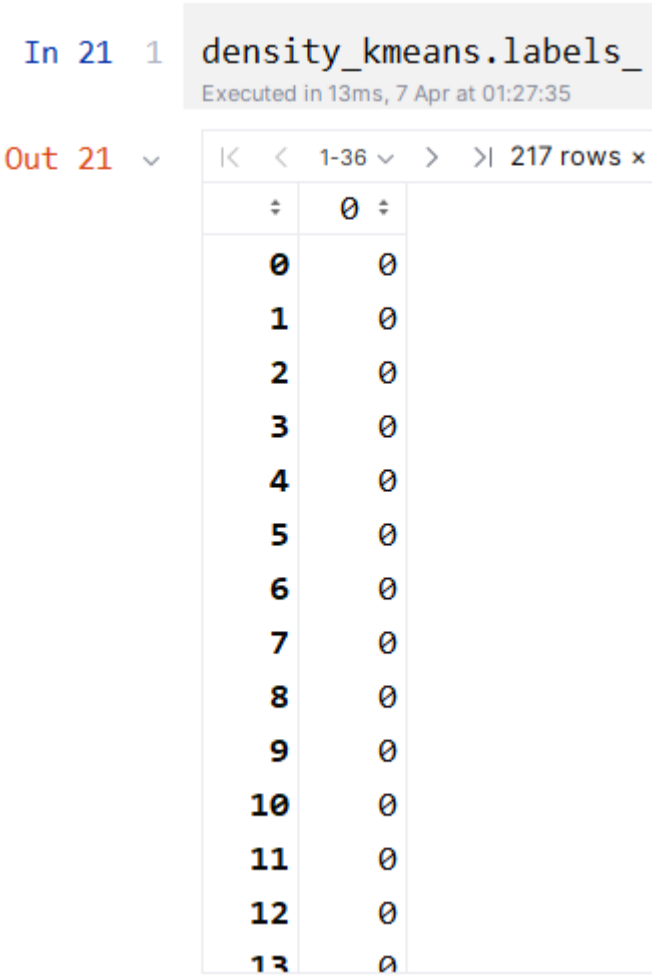

Рис. 4.19 – Мітки кластерів для кожної ознаки

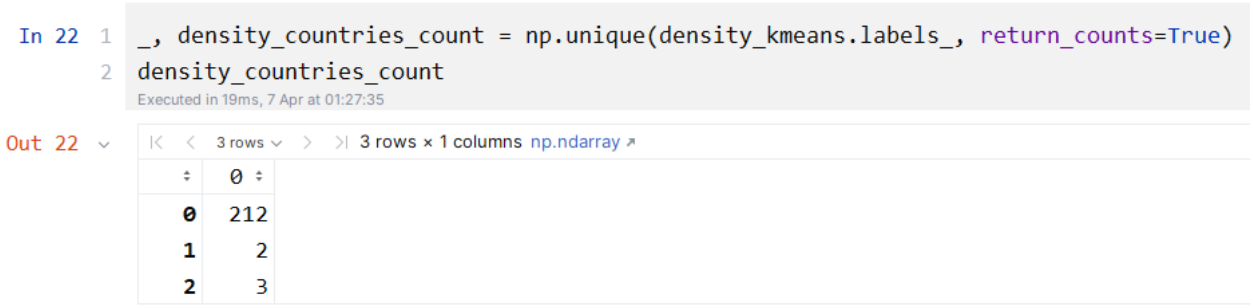

Рис. 4.20 – Кількість ознак у кожному кластері

Бачимо, що в першому кластері аж 212 країн, тоді як у другому – дві, а в третьому – 3. Оскільки кластери малі, можемо вивести перелік країн кожного з кластерів.

```
In 23
first_density_cluster = df[density_kmeans.labels_ == 0]
first_density_cluster
```
Executed in 33ms, 7 Apr at 01:27:35

Out 23 — 1-36 — 212 rows × 6 columns pd.Dataframe

country_name	region	gdp_per_capita	population	co2_emission	area	density
Afghanistan	South Asia	561.778746	34656032	9809.225000	652860.0	53.083405
Albania	Europe & Central Asia	4124.982390	2876101	5716.853000	28750.0	100.038296
Algeria	Middle East & North Africa	3916.881571	40606052	145400.217000	2381740.0	17.048902
American Samoa	East Asia & Pacific	11834.745230	55599	165114.116337	200.0	277.995000
Andorra	Europe & Central Asia	36988.622030	77281	462.042000	470.0	164.427660
Angola	Sub-Saharan Africa	3308.700233	28813463	34763.160000	1246700.0	23.111786
Antigua and Barbuda	Latin America & Caribbean	14462.176280	100963	531.715000	440.0	229.461364
Argentina	Latin America & Caribbean	12440.320980	43847430	204024.546000	2780400.0	15.770188
Armenia	Europe & Central Asia	3614.688357	2924816	5529.836000	29740.0	98.346200
Aruba	Latin America & Caribbean	13445.593416	104822	872.746000	180.0	582.344444
Australia	East Asia & Pacific	49755.315480	24127159	361261.839000	7741220.0	3.116713
Austria	Europe & Central Asia	44757.634900	8747358	58712.337000	83879.0	104.285435
Azerbaijan	Europe & Central Asia	3878.709257	9762274	37487.741000	86600.0	112.728337

Рис. 4.21 – Перший кластер

```
In 24
second_density_cluster = df[density_kmeans.labels_ == 1]
second_density_cluster
```
Executed in 89ms, 7 Apr at 01:27:35

Out 24 — 2 rows — 2 rows × 6 columns pd.Dataframe

country_name	region	gdp_per_capita	population	co2_emission	area	density
Macao SAR, China	East Asia & Pacific	74017.184710	612167	1283.450000	30.3	20203.531353
Monaco	Europe & Central Asia	13445.593416	38499	165114.116337	2.0	19249.500000

Рис. 4.22 – Другий кластер

```
In 25
third_density_cluster = df[density_kmeans.labels_ == 2]
third_density_cluster
```
Executed in 82ms, 7 Apr at 01:27:35

Out 25 — 3 rows — 3 rows × 6 columns pd.Dataframe

country_name	region	gdp_per_capita	population	co2_emission	area	density
Gibraltar	Europe & Central Asia	13445.593416	34408	528.048	10.0	3440.800000
Hong Kong SAR, China	East Asia & Pacific	43740.992880	7346700	46222.535	1105.0	6648.597285
Singapore	East Asia & Pacific	52962.491570	5607283	56372.791	719.0	7798.724618

Рис. 4.23 – Третій кластер

Можемо зробити висновок, що, скоріше за все, кластеризація відпрацювала коректно, оскільки в другий кластер потрапили країни з дуже великою щільністю населення, а в третій – із більшою, ніж середня. Виведемо максимальну щільність населення в кожному регіоні в кожному кластері.

```
In 26  1  first_density_cluster.groupby('region').density.max().sort_values(ascending=False)
```
Executed in 65ms, 7 Apr at 01:27:35

Out 26 — 7 rows — Length: 7, dtype: float64 pd.Series

region	density
Middle East & North Africa	1848.470817
South Asia	1391.640000
North America	1306.620000
Latin America & Caribbean	1176.617647
Europe & Central Asia	866.005263
East Asia & Pacific	652.450000
Sub-Saharan Africa	619.349510

Рис. 4.24 – Максимальна щільність населення за регіонами в першому кластері

```
In 27  1  second_density_cluster.groupby('region').density.max().sort_values(ascending=False)
```
Executed in 36ms, 7 Apr at 01:27:35

Out 27 — 2 rows — Length: 2, dtype: float64 pd.Series

region	density
East Asia & Pacific	20203.531353
Europe & Central Asia	19249.500000

Рис. 4.25 – Максимальна щільність населення за регіонами в другому кластері

```
In 28  1  third_density_cluster.groupby('region').density.max().sort_values(ascending=False)
```
Executed in 147ms, 7 Apr at 01:27:35

Out 28 — 2 rows — Length: 2, dtype: float64 pd.Series

region	density
East Asia & Pacific	7798.724618
Europe & Central Asia	3440.800000

Рис. 4.26 – Максимальна щільність населення за регіонами в третьому кластері

Як бачимо, в першому кластері домінує регіон Близький Схід та Північна Африка, у другому та третьому – Східна Азія та Тихий океан.

Отримані кластери, особливо при кластеризації по щільності населення, було б добре перевірити на точність. Для цього можемо використати оцінку силуету.

```
In 29  1  silhouette_score(gdp_features, gdp_kmeans.labels_)
Executed in 226ms, 7 Apr at 01:27:36

Out 29    0.6694916845572668
```

Рис. 4.27 – Оцінка силуету моделі кластеризації за ВВП на душу населення

```
In 30  1  silhouette_score(density_features, density_kmeans.labels_)
Executed in 332ms, 7 Apr at 01:27:36

Out 30    0.9532906385438781
```

Рис. 4.28 – Оцінка силуету моделі кластеризації за щільністю

У випадку кластеризації за ВВП на душу населення вона дорівнює 0.669, що можна вважати непоганим результатом, а у випадку кластеризації за щільністю населення – аж 0.953, що є дуже високим результатом.

4.2 Вивести частотні гістограми всіх показників файла Data2.csv, використовуючи цикл

Для початку відкинемо всі нечислові дані (регіон).

```
In 31  1  numeric = df.select_dtypes(exclude=[object])
       2  numeric
Executed in 318ms, 7 Apr at 01:27:36
```

Out 31 — 1-36 — 217 rows × 5 columns pd.Dataframe

country_name	gdp_per_capita	population	co2_emission	area	density
Afghanistan	561.778746	34656032	9809.225000	652860.0	53.083405
Albania	4124.982390	2876101	5716.853000	28750.0	100.038296
Algeria	3916.881571	40606052	145400.217000	2381740.0	17.048902
American Samoa	11834.745230	55599	165114.116337	200.0	277.995000
Andorra	36988.622030	77281	462.042000	470.0	164.427660
Angola	3308.700233	28813463	34763.160000	1246700.0	23.111786
Antigua and Barbuda	14462.176280	100963	531.715000	440.0	229.461364
Argentina	12440.320980	43847430	204024.546000	2780400.0	15.770188
Armenia	3614.688357	2924816	5529.836000	29740.0	98.346200
Aruba	13445.593416	104822	872.746000	180.0	582.344444
Australia	49755.315480	24127159	361261.839000	7741220.0	3.116713
Austria	44757.634900	8747358	58712.337000	83879.0	104.285435
Azerbaijan	3878.709257	9762274	37487.741000	86600.0	112.728337

Рис. 4.29 – Цифрові дані набору даних

Потім просто пройдемося циклом по кожному стовпцю й відобразимо для них частотні гістограми за допомогою звичайного методу .hist() класу DataFrame.

```
In 32
fig, axes = plt.subplots(math.ceil(numeric.shape[1] / 2), numeric.shape[1] // 2, figsize=(15, 10))

for ax, column in zip(axes.ravel(), numeric.columns):
    ax.set_title(column)
    numeric[column].hist(ax=ax)
Executed in 1s, 7 Apr at 01:27:37
```

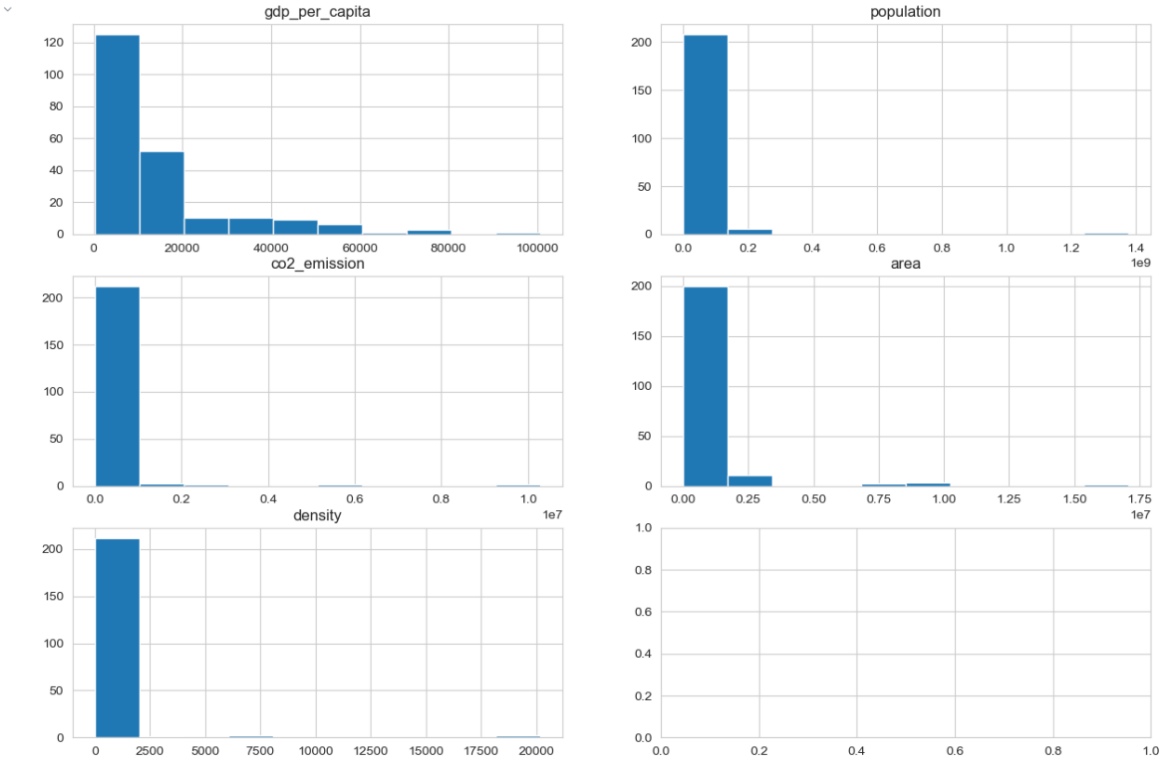

Рис. 4.30 – Частотні гістограми

4.3 Створити функцію, яка на вхід отримує два набори даних, перевіряє чи є лінійна залежність та виводить True чи False (будемо розуміти під «є лінійна залежність», якщо коефіцієнт кореляції по модулю більше 0,8)

Таку перевірку зробити дуже просто: треба всього викликати метод .corrwith() класу DataFrame, задіяти для отриманих значень модуль та перевірити, чи є такі, що перевищують 0.8.

```
In 33
def has_linear_dependence(first_df: pd.DataFrame, second_df: pd.DataFrame) -> bool:
    corr = first_df.corrwith(second_df).abs()
    return (0.8 <= corr).any()
Executed in 28ms, 7 Apr at 01:27:37
```

Рис. 4.31 – Функція перевірки на наявність лінійної залежності між двома наборами даних

Створимо тестові набори даних.

```
In 34  1  df1 = pd.DataFrame.from_dict({
       2      'a': [1, 2, 3],
       3      'b': [4, 8, 12]
       4  })
       5  df2 = pd.DataFrame.from_dict({
       6      'a': [2, 3, 5],
       7      'b': [0, -10, 3]
       8  })
Executed in 52ms, 7 Apr at 01:27:37
```

Рис. 4.32 – Створення тестових наборів даних

Бачимо, що в них явно є залежність між стовпцями «a» та вона явно відсутня між стовпцями «b». Перевіримо, що поверне створена функція.

```
In 35  1  has_linear_dependence(df1, df2)
Executed in 38ms, 7 Apr at 01:27:37

Out 35    True
```

Рис. 4.33 – Тестування функції

Вона повернула True, як і очікувалося, а значить функція працює коректно.

5 ВИСНОВОК

У ході даної лабораторної роботи було досліджено набір даних пасажирів Титаніку, на основі якого побудовано декілька класифікаційних моделей, що були засновані на наступних методах: К-найближчих сусідів, дерева рішень, випадкових лісів. Для кожного методу було обраховано середньоквадратичну похибку, за значеннями якої було встановлено, що найкращим методом для такого набору даних є метод випадкових лісів, який показав точність у 81.28% на навчальній вибірці та середньоквадратичну помилку 0.17. Було помічено, що в методі дерева прийняття рішень дуже велику роль грає максимальна глибина.

Після цього було побудовано дві моделі кластеризації набору даних країн: за ВВП на душу населення та щільністю населення. Для кожної моделі було обраховано точку локтя: в обох випадках вона дорівнювала трьом. Потім було виведено, які країни входять до кожного з кластерів, а також які регіони домінують за ознакою кластеризації в кожному з кластерів. Також для обох моделей було обраховано оцінку силуету: для першої моделі вона склала 0.669, а для другої – 0.953.

Потім було зображено гістограми розподілу для кожної з ознак, а в кінці створено функцію, яка визначає, чи є лінійна залежність між двома наборами даних: її було перевірено на власноруч створеному тестовому наборі даних.