Задачі кластеризації та класифікації

Мета

Ознайомитись з різновидами моделей для задач кластеризації та класифікації, а також методами побудови та оцінки цих моделей.
Після завершення цієї лабораторної роботи ви зможете:

- Використовувати прості моделі для задач кластеризації та класифікації
- Використовувати перехресну перевірку для оцінки якості моделі
- Обирати оптимальну складність моделі для уникнення перенавчання
- Вдосконалювати моделі за допомогою підбору параметрів

Завдання, що оцінюються

1. Скачайте дані із файлу 'clean_data2.csv' (Data2.csv з виправленими помилками та заповненими пропусками). Виконайте кластеризацію по ВВП на душу населення та щільності населення.
2. Використайте метод ліктя для підбору оптимальної кількості кластерів.
3. Визначіть, який регіон домінує в кожному з кластерів.
4. Побудуйте кілька (3-5) моделей класифікації, що визначають регіон, до якого належить країна, по ознаках 'GDP per capita', 'Population', 'CO2 emission', 'Area'. Оцініть точність класифікації (використайте 20% загального набору в якості тестових даних).
5. Для однієї з моделей виконайте підбір параметра. Обгрунтуйте ваш вибір.

Завдання #1:

Виконайте кластеризацію по ВВП на душу населення та щільності населення.

Зчитую дані з файлу у датафрейм

```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import linkage, dendrogram
from sklearn.metrics import mean_squared_error, r2_score, classification_report, silhouette_score, confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from google.colab import drive
drive.mount('/content/drive')

path = "/content/drive/My Drive/data/clean_data2.csv"
df = pd.read_csv(path)
```

```
Mounted at /content/drive
```

Виділяю параметри для кластеризації

```
features = df[['GDP per capita', 'Population density']]
print(features)
```

```
      GDP per capita  Population density
0         561.778746           53.083405
```

```
1       4124.982390          100.038296
2       3916.881571           17.048902
3      11834.745230          277.995000
4      36988.622030          164.427660
..              ...                 ...
212    13445.593416          294.145714
213     2943.404534          756.074086
214      990.334774           52.245796
215     1269.573537           22.045136
216     1029.076649           41.330643

[217 rows x 2 columns]
```

Будую модель методом k середніх з кількістю кластерів 5

```
kmeans = KMeans(
init='random',
n_clusters=5,
n_init=10,
max_iter=300
)
kmeans.fit(features)
```

```
KMeans
KMeans(init='random', n_clusters=5, n_init=10)
```

Отримані центри кластерів:

```
kmeans.cluster_centers_
```

```
array([[ 3195.25239747,   124.48051682],
       [77937.389578  ,  4182.98337309],
       [32920.43433938,   286.99742244],
       [49062.061954  ,  1103.58673324],
       [14276.03079606,   608.53993555]])
```

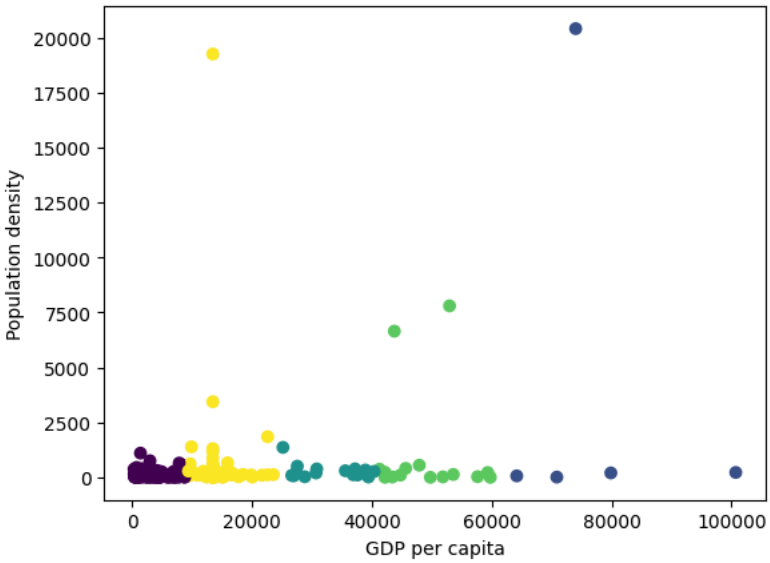
Масив із номерами кластерів для кожного рядка даних:

```
kmeans.labels_
```

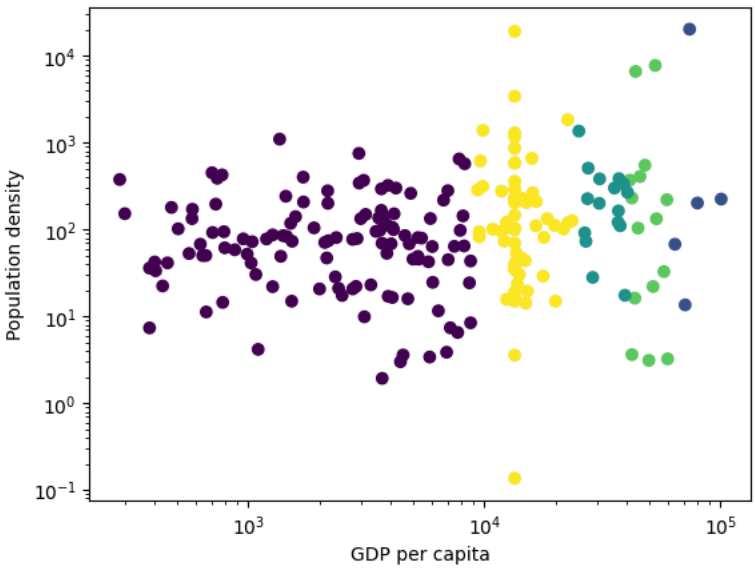
```
array([0, 0, 0, 4, 2, 0, 4, 4, 0, 4, 3, 3, 0, 2, 4, 0, 4, 0, 3, 0, 0, 4,
       0, 0, 0, 0, 0, 4, 2, 0, 0, 0, 0, 0, 0, 3, 4, 0, 0, 4, 4, 0, 0, 0,
       0, 0, 4, 0, 4, 4, 4, 4, 4, 3, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 4, 0,
       3, 2, 4, 0, 0, 0, 3, 0, 4, 4, 4, 4, 2, 0, 0, 0, 0, 0, 0, 3, 4, 3,
       0, 0, 0, 0, 1, 4, 2, 2, 0, 2, 0, 0, 0, 0, 4, 2, 0, 2, 0, 0, 4, 0,
       0, 0, 4, 4, 4, 1, 1, 0, 0, 0, 4, 4, 0, 2, 0, 0, 4, 0, 0, 0, 4, 0,
       0, 0, 0, 0, 0, 0, 0, 3, 4, 2, 0, 4, 0, 4, 1, 4, 0, 4, 4, 0, 0, 0,
       0, 4, 4, 2, 3, 4, 0, 0, 0, 3, 0, 4, 0, 0, 4, 0, 3, 4, 4, 4, 0, 0,
       0, 4, 2, 0, 4, 4, 4, 0, 0, 0, 0, 3, 1, 4, 0, 0, 0, 0, 0, 0, 4, 0,
       4, 0, 4, 0, 0, 0, 2, 2, 3, 4, 0, 0, 4, 0, 4, 0, 0, 0, 0],
      dtype=int32)
```

Візуалізую отримані кластери:

```
plt.xlabel('GDP per capita')
plt.ylabel('Population density')
plt.scatter(df[['GDP per capita']], df[['Population density']], c=kmeans.labels_)
plt.show()
```




```
plt.xlabel('GDP per capita')
plt.ylabel('Population density')
plt.scatter(df[['GDP per capita']], df[['Population density']], c=kmeans.labels_)
plt.xscale('log')
plt.yscale('log')
plt.show()
```



Завдання #2:

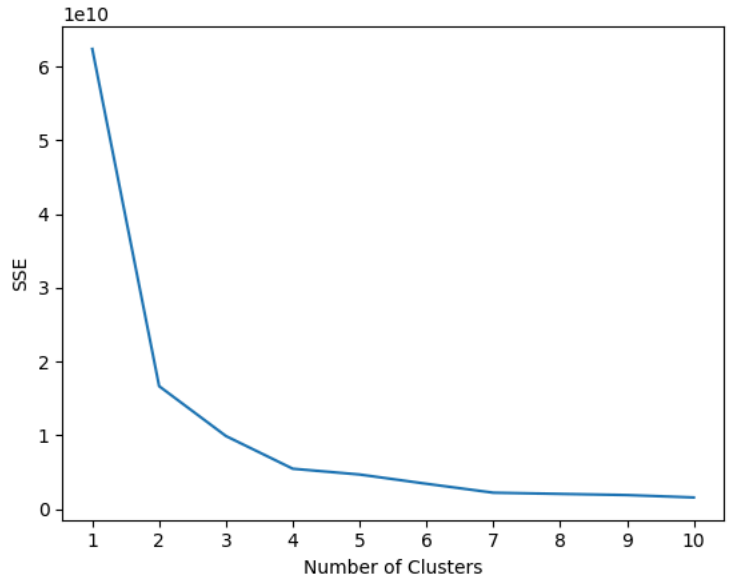
Використайте метод ліктя для підбору оптимальної кількості кластерів.

Визначаю оптимальну кількість кластерів. Скористаюсь методом "ліктя". Для цього ініціалізую алгоритм k середніх кількістю кластерів від 1 до 10 і для кожної моделі рахую суму квадратів похибок (евклідових відстаней точок кластерів від відповідних центрів):

```
kmeans_kwargs = {
'init': 'random',
'n_init': 10,
'max_iter': 300,
'random_state': 42,
}
sse = []
max_kernels = 10
for k in range(1, max_kernels + 1):
  kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
  kmeans.fit(features)
  sse.append(kmeans.inertia_)
```

Візуалізую отримані результати:

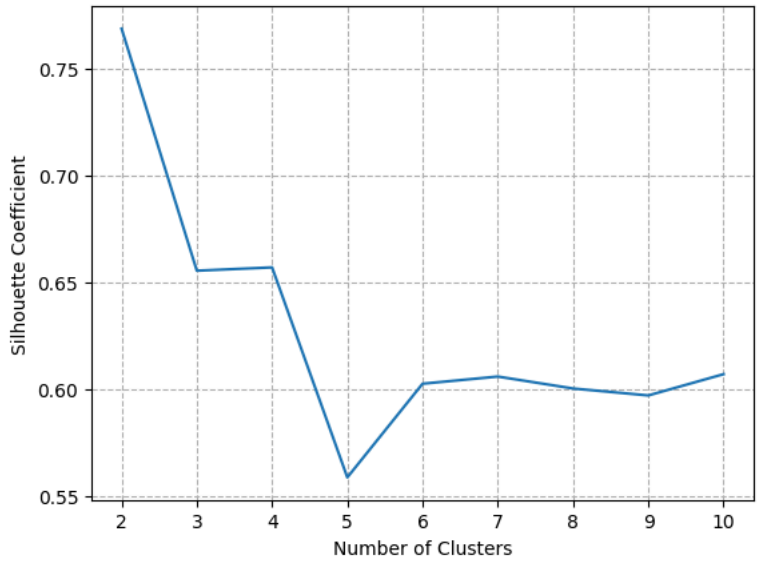
```
plt.plot(range(1, max_kernels + 1), sse)
plt.xticks(range(1, max_kernels + 1))
plt.xlabel('Number of Clusters')
plt.ylabel('SSE')
plt.show()
```



Оптимальна кількість кластерів дорівнює 3 або 4

```
silhouette_coefficients = []
for k in range(2, max_kernels + 1):
  kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
  kmeans.fit(features)
  score = silhouette_score(features, kmeans.labels_)
  silhouette_coefficients.append(score)

plt.plot(range(2, max_kernels + 1), silhouette_coefficients)
plt.xticks(range(2, max_kernels + 1))
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Coefficient')
plt.grid(linestyle='--')
plt.show()
```



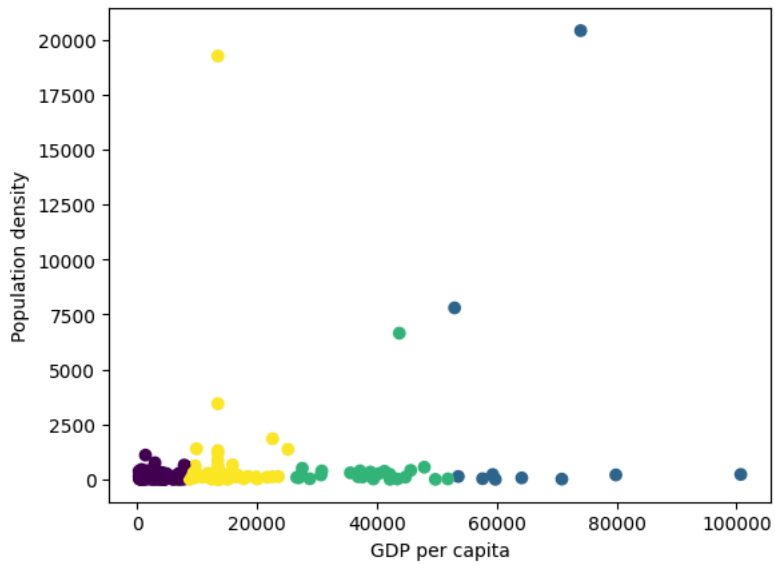
Використавши додатковий аналіз за допомогою графіку коефіцієнтів силуетів, бачимо, що значення коефіцієнта для чотирьох кластерів є більшим ніж для трьох, що і вказує на оптимальну кількість кластерів.

Візуалізую для оптимальної кількості кластерів:

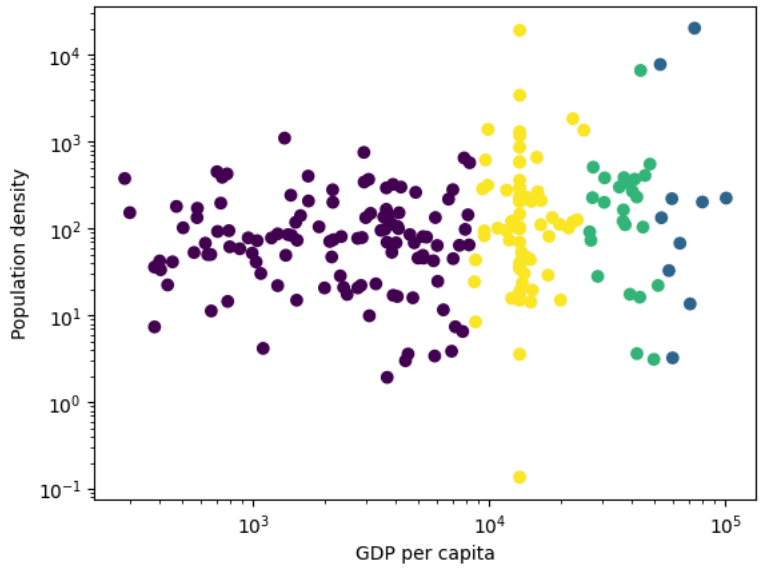
```
kmeans_main = KMeans(
init='random',
n_clusters=4,
n_init=10,
max_iter=300
)
kmeans_main.fit(features)
```

```
KMeans
KMeans(init='random', n_clusters=4, n_init=10)
```

```
plt.xlabel('GDP per capita')
plt.ylabel('Population density')
plt.scatter(df[['GDP per capita']], df[['Population density']], c=kmeans_main.labels_)
plt.show()
```



```
plt.xlabel('GDP per capita')
plt.ylabel('Population density')
plt.scatter(df[['GDP per capita']], df[['Population density']], c=kmeans_main.labels_)
plt.xscale("log")
plt.yscale("log")
plt.show()
```



Завдання #3:

Визначіть, який регіон домінує в кожному з кластерів.

Додаю мітки кластерів в датафрейм

```
df['Cluster'] = kmeans_main.labels_
```

Визначаю домінуючий регіон для кожного кластера

```
region_counts = df.groupby('Cluster')['Region'].value_counts()
region_counts
```

```
Cluster  Region
0        Sub-Saharan Africa            42
         East Asia & Pacific           21
         Europe & Central Asia         18
         Latin America & Caribbean     18
         Middle East & North Africa    10
         South Asia                     7
1        Europe & Central Asia          6
         East Asia & Pacific            2
         Middle East & North Africa     1
         North America                  1
2        Europe & Central Asia         12
         East Asia & Pacific            7
         Middle East & North Africa     3
         Latin America & Caribbean      2
         North America                  1
3        Latin America & Caribbean     22
         Europe & Central Asia         22
         East Asia & Pacific            7
         Middle East & North Africa     7
         Sub-Saharan Africa             6
         North America                  1
         South Asia                     1
Name: count, dtype: int64
```

```
dominant_regions = region_counts.groupby(level=0).idxmax() # level=0: групує дані за
dominant_regions
```

```
Cluster
0           (0, Sub-Saharan Africa)
1        (1, Europe & Central Asia)
2        (2, Europe & Central Asia)
3    (3, Latin America & Caribbean)
Name: count, dtype: object
```

Завдання #4:

Побудуйте кілька (3-5) моделей класифікації, що визначають регіон, до якого належить країна, по ознаках 'GDP per capita', 'Population', 'CO2 emission', 'Area'. Оцініть точність класифікації (використайте 20% загального набору в якості тестових даних).

Обираю потрібні ознаки:

```
all_features=pd.get_dummies(df[['GDP per capita', 'Population', 'CO2 emission', 'Area']])
all_features[['Region']] = df[['Region']]
all_features
```

	GDP per capita	Population	CO2 emission	Area	Region
0	561.778746	34656032.0	9809.225000	652860	South Asia
1	4124.982390	2876101.0	5716.853000	28750	Europe & Central Asia
2	3916.881571	40606052.0	145400.217000	2381740	Middle East & North Africa
3	11834.745230	55599.0	165114.116337	200	East Asia & Pacific
4	36988.622030	77281.0	462.042000	470	Europe & Central Asia
...	...	...	...	...	...
212	13445.593416	102951.0	165114.116337	350	Latin America & Caribbean
213	2943.404534	4551566.0	165114.116337	6020	Middle East & North Africa
214	990.334774	27584213.0	22698.730000	527970	Middle East & North Africa
215	1269.573537	16591390.0	4503.076000	752610	Sub-Saharan Africa
216	1029.076649	16150362.0	12020.426000	390760	Sub-Saharan Africa

Next steps: View recommended plots

Розділяю датасет на навчальну і тестову вибірки за допомогою функції `train_test_split()`:

```
df_train, df_test = train_test_split(
all_features,
test_size=0.2,
random_state=1
)
```

```
df_train.head()
print(df_test.count())
print()
print(df_train.count())
```

```
GDP per capita    44
Population        44
CO2 emission      44
Area              44
Region            44
dtype: int64

GDP per capita    173
Population        173
CO2 emission      173
Area              173
Region            173
dtype: int64
```

Розміщую цільові дані - Region - в окремому датафреймі:

```
x_train = df_train[['GDP per capita', 'Population', 'CO2 emission', 'Area']]
y_train = df_train[['Region']]

x_test = df_test[['GDP per capita', 'Population', 'CO2 emission', 'Area']]
y_test = df_test[['Region']]
```

Навчання та тестування моделей:

Для навчання були обрані наступні методи:

- k-nearest neighbors;
- Support vector machines;
- Decision Tree;
- Random Forest;
- Extra Trees;
- AdaBoost;
- Gradient Boosting

Метод 1 K-nearest neighbors

Будую модель

```
KNN_model = KNeighborsClassifier(n_neighbors=5)
KNN_model.fit(x_train.values, y_train.values)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: Dat
  return self._fit(X, y)
```

KNeighborsClassifier

KNeighborsClassifier()

Оцінюю точність

```
print('mean accuracy = ', KNN_model.score(x_test.values, y_test.values))
```

```
mean accuracy =  0.22727272727272727
```

Метод 2 Support vector machines

Будую модель

```
SVC_model = SVC()
SVC_model.fit(x_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConvers
  y = column_or_1d(y, warn=True)
```

SVC

SVC()

Оцінюю точність

```
print('mean accuracy = ', SVC_model.score(x_test, y_test))
```

```
mean accuracy =  0.3181818181818182
```

Метод 3 Decision Tree

Будую модель

```
decision_tree = DecisionTreeClassifier(max_depth=3, random_state=1)
decision_tree.fit(x_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, random_state=1)
```

Оцінюю точність

```
decision_tree.score(x_test, y_test)
```

```
0.5227272727272727
```

Метод 4 Random Forest

Будую модель

```
randomforest = RandomForestClassifier(max_depth=5)
randomforest.fit(x_train, y_train)
```

```
<ipython-input-31-d0afa267809d>:2: DataConversionWarning: A column-vector y was passe
  randomforest.fit(x_train, y_train)
RandomForestClassifier
RandomForestClassifier(max_depth=5)
```

Оцінюю точність

```
print('mean accuracy = ', randomforest.score(x_test, y_test))
```

```
mean accuracy =  0.5
```

Метод 5 Extra Trees

Будую модель

```
extratrees = ExtraTreesClassifier(max_depth=5)
extratrees.fit(x_train, y_train)
```

```
<ipython-input-33-0fa63d06422f>:2: DataConversionWarning: A column-vector y was passe
  extratrees.fit(x_train, y_train)
ExtraTreesClassifier
ExtraTreesClassifier(max_depth=5)
```

Оцінюю точність

```
extratrees.score(x_test, y_test)
```

```
0.5227272727272727
```

Метод 6 AdaBoost

Будую модель

```
adaboost = AdaBoostClassifier(learning_rate=0.3)
adaboost.fit(x_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConvers
  y = column_or_1d(y, warn=True)
AdaBoostClassifier
AdaBoostClassifier(learning_rate=0.3)
```

Оцінюю точність

```
adaboost.score(x_test, y_test)
```

```
0.4772727272727273
```

Метод 7 Gradient Boosting

Будую модель

```
gradboost = GradientBoostingClassifier(learning_rate=0.05)
gradboost.fit(x_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_gb.py:437: DataConversionWa
  y = column_or_1d(y, warn=True)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.05)
```

Оцінюю точність

```
gradboost.score(x_test, y_test)
```

```
0.4772727272727273
```

Завдання #5:

Для однієї з моделей виконайте підбір параметра. Обгрунтуйте ваш вибір.

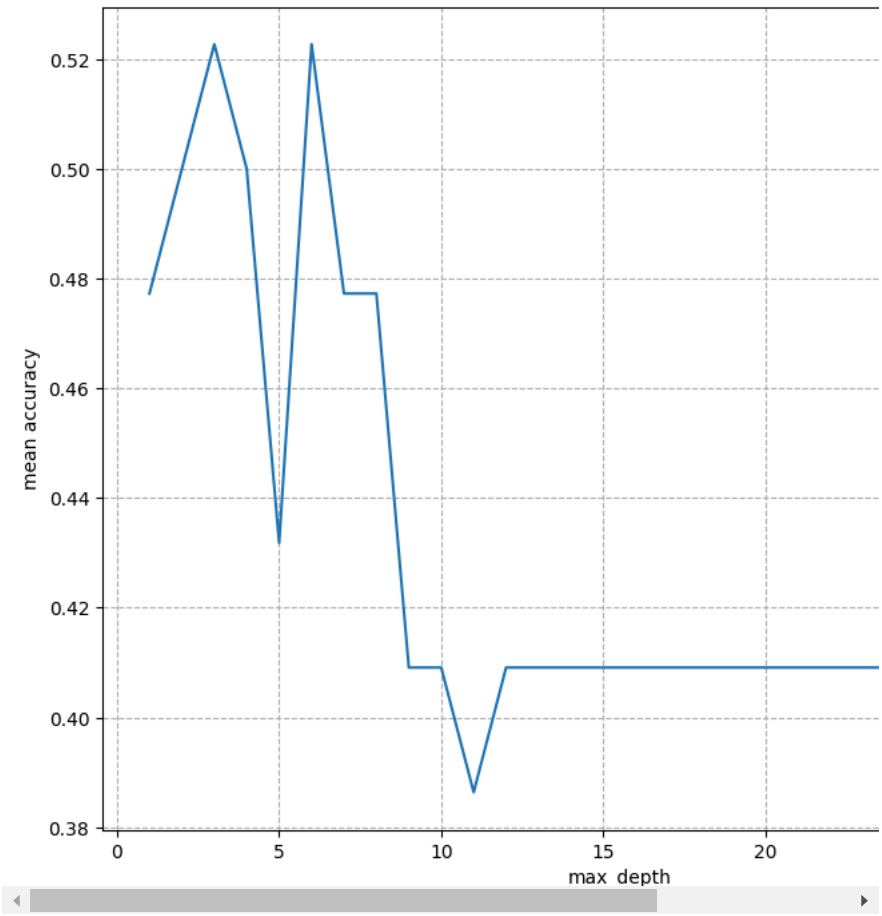
Найкращу оцінку дає модель Decision Tree, тому скористаюсь методом “ліктя” для визначення параметра max-depth для методу Decision Tree.

Для цього ініціалізую алгоритм з параметром від 1 до 30 і для кожної моделі порахую точність передбачення:

```
accur = []
max_kernels = 30
for k in range(1, max_kernels + 1):
  decision_tree = DecisionTreeClassifier(max_depth=k, random_state=1)
  decision_tree.fit(x_train, y_train)
  accur.append(decision_tree.score(x_test, y_test))
```

Отримані показники якості візуалізую на графіку:

```
plt.figure(figsize=(10, 8))
plt.plot(range(1, max_kernels + 1), accur)
plt.xticks(range(0, max_kernels + 1,5))
plt.xlabel('max_depth')
plt.ylabel('mean accuracy')
plt.grid(linestyle='--')
plt.show()
```



З графіку бачу, що найкраще значення параметра знаходиться в межах від 1 до 8 , тому що, як видно з графіку, оптимальні значення параметра лежать в діапазоні від 1 до 8. Це особливо стосується глибини 3 та 6, які дають найвищий результат. При подальшому збільшенні параметра max_depth ми спостерігаємо значне зниження оцінки, яка після досягнення глибини 12 стабілізується і становить приблизно 0.41. Тому рекомендується вибрати глибину 3 або 6, оскільки більші значення глибини не призведуть до покращення результату.

Додаткове завдання:

Використовуючи файл Data5.csv:

1. Визначити кластер країн з найкращим розвитком (кластеризувати по Ie, Iec, Is; для k-середніх використати 4 кластера). Побудувати центри кластерів.
2. Провести кластеризацію по Cql, порахувати скільки країн потрапило в різні кластери, якщо порівнювати з п1.

Згідно з методологією вимірювання сталого розвитку країн, сталий розвиток оцінюється за допомогою відповідного індексу у просторі трьох вимірів: економічного (Iec), екологічного (Ie) і соціально-інституціонального (Is). Цей індекс є вектором, норма якого визначає рівень сталого розвитку, а його просторове положення в системі координат (Iec,Ie,Is) характеризує міру «гармонійності» цього розвитку.

▶ Натисніть тут, щоб побачити підказку

```
# Напишіть ваш код нижче та натисніть Shift+Enter для виконання

df = pd.read_csv(
    '/content/drive/My Drive/data/Data5.csv', encoding='windows-1251', sep=';', decimal=','
).rename(columns={'Unnamed: 0': 'Country'})

print(df.head())
```

```
      Country  ISO         UA       Cql        Ie       Iec        Is
0     Albania  ALB    Албанія  0.973924  0.605348  0.538673  0.510113
1     Algeria  DZA      Алжир  0.782134  0.587219  0.348159  0.497986
2      Angola  AGO     Ангола  0.372344  0.274394  0.332117  0.346907
```

```
3  Argentina  ARG  Аргентина  0.883830  0.699685  0.281995  0.518820
4    Armenia  ARM   Вірменія  1.016499  0.718327  0.535648  0.486498
```

Вибір ознак

```
features = df[['Ie', 'Iec', 'Is']]
```

Кластеризація k-середніх

```
kmeans1 = KMeans(
    n_clusters=4,
    init='random',
    n_init=10,
    max_iter=300
)
kmeans1.fit(features)
```

```
KMeans
KMeans(init='random', n_clusters=4, n_init=10)
```

Додавання міток кластерів та обчислення середніх значень кластерів

```
df['claster'] = kmeans1.labels_
mean_values = df.groupby('claster').apply(lambda group: (group['Ie'] + group['Iec'] + group['Is']).mean() / 3)
```