به نام خدا

گزارش پروژه داده کاوی(فاز دوم)

داده کاوی و الگو الگویابی بیماری مزمن کلیوی (CKD) با استفاده از روشهای خوشه بندی و طبقه بندی داده های بالینی

استاد امیررضا آزمون

ملیکا ناصحی مقدم

شماره دانشجویی : ۴۰۱۴۶۳۱۸۰

مرحله ۱: تحلیل اکتشافی داده ها (EDA)

a. تحلیل تفاوت بین گروه های ckd و notckd برحسب تمام ویژگی ها

در این مرحله، ابتدا داده‌های تمیز (cleaned) بارگذاری و پیش‌پردازش شدند. سپس داده‌ها بر اساس برچسب کلاس به دو گروه **CKD**(مبتلا) و **NotCKD**(غیرمبتلا) تقسیم شدند.

```
[35]: #phase 2 project

[36]: pip install xlrd

Requirement already satisfied: xlrd in c:\anaconda\lib\site-packages (2.0.2)
Note: you may need to restart the kernel to use updated packages.

[80]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import ttest_ind, chi2_contingency

# بارگذاری داده تمیز
data = pd.read_csv("cleaned_ckd_data.xls", encoding="utf-8-sig")

# حذف ردیف‌هایی که کلاس ندارند
data = data.dropna(subset=['class'])

# یکسان‌سازی نام کلاس‌ها
data['class'] = data['class'].str.strip().str.lower()

# تقسیم داده‌ها
ckd_data = data[data['class'] == 'ckd']
notckd_data = data[data['class'] == 'notckd']

# تشخیص ویژگی‌های عددی و اسمی
numerical_features = data.select_dtypes(include=['float64', 'int64']).columns.tolist()
categorical_features = data.select_dtypes(include=['object']).columns.tolist()
categorical_features = [col for col in categorical_features if col != 'class']
```

```
numerical_features = data.select_dtypes(include=['float64', 'int64']).columns.tolist()
categorical_features = data.select_dtypes(include=['object']).columns.tolist()
categorical_features = [col for col in categorical_features if col != 'class']

# تحلیل آماری ویژگی‌های عددی
numerical_results = []
for feature in numerical_features:
    ckd_vals = ckd_data[feature].dropna()
    notckd_vals = notckd_data[feature].dropna()
    if not ckd_vals.empty and not notckd_vals.empty:
        stat, p = ttest_ind(ckd_vals, notckd_vals, equal_var=False)
        numerical_results.append({
            "Feature": feature,
            "CKD_Mean": round(ckd_vals.mean(), 2),
            "NotCKD_Mean": round(notckd_vals.mean(), 2),
            "p-value": round(p, 5)
        })

numerical_summary = pd.DataFrame(numerical_results).sort_values(by="p-value")
display(numerical_summary)

# تحلیل آماری ویژگی‌های اسمی
categorical_results = []
for feature in categorical_features:
    contingency = pd.crosstab(data[feature], data['class'])
    if contingency.shape[0] > 1 and contingency.shape[1] > 1:
        chi2_stat, p, _, _ = chi2_contingency(contingency)
        categorical_results.append({"Feature": feature, "p-value": round(p, 5)})

categorical_summary = pd.DataFrame(categorical_results).sort_values(by="p-value")
display(categorical_summary)
```

	Feature	CKD_Mean	NotCKD_Mean	p-value
1	PC2	-82.80	120.89	0.00000
4	sg	1.01	1.02	0.00000
5	al	1.59	0.00	0.00000
6	su	0.69	0.00	0.00000
0	PC1	458.42	-669.29	0.00001
3	PC4	2.86	-4.18	0.09586
2	PC3	-0.08	0.12	0.97422

	Feature	p-value
0	pcc	0.00000
2	htn	0.00000
3	dm	0.00000
4	cad	0.00000
5	pe	0.00000
6	ane	0.00000
1	ba	0.00016

خروجی :

- برخی ویژگی‌های عددی مانند **sc (Serum Creatinine)** ، **hemo (Hemoglobin)** و **al (Albumin)** دارای تفاوت معنی‌دار آماری بین گروه‌های CKD و NotCKD بودند (p-value بسیار کم) .
- در ویژگی‌های اسمی نیز عواملی مانند **htn (Hypertension)** و **dm (Diabetes Mellitus)** رابطه‌ی آماری قابل توجهی با بیماری CKD داشتند.

تفسیر:

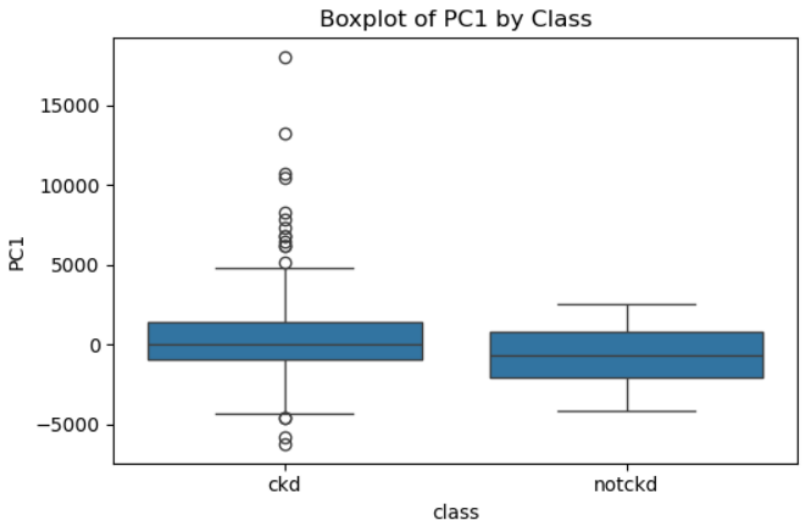
ویژگی‌هایی با **p-value کمتر**، تأثیر بیشتری در تمایز بیماران مبتلا و غیرمبتلا دارند و می‌توانند در مدل‌سازی آینده نقش مهمی داشته باشند.

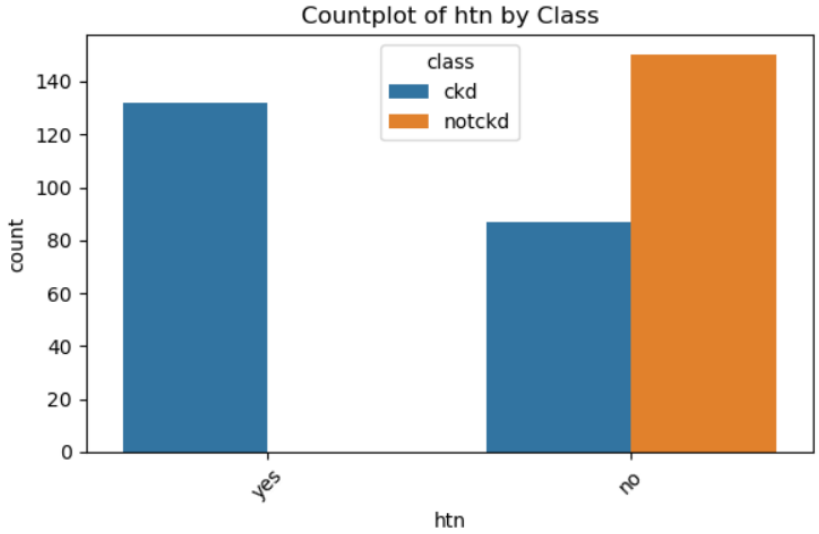
b. استفاده از بصری سازی

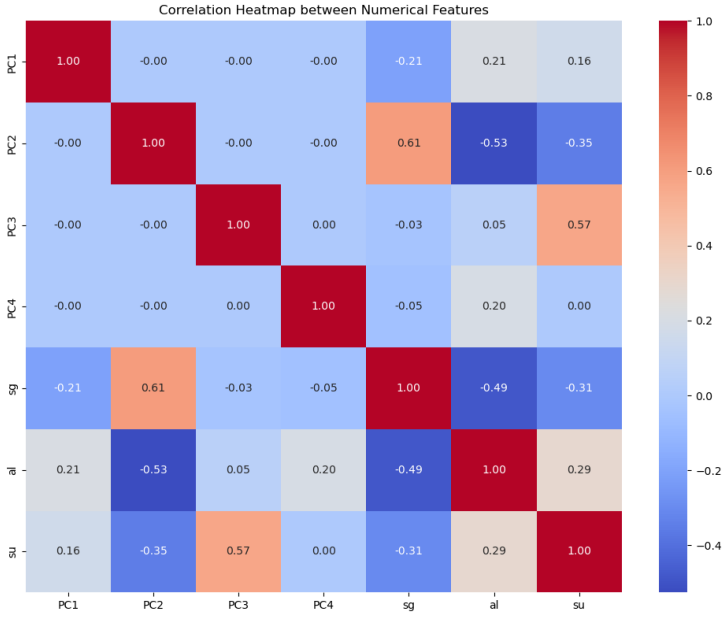
در این مرحله، تحلیل آماری و بصری‌سازی داده‌ها برای بررسی تفاوت بین بیماران CKD و NotCKD انجام شد. ابتدا با آزمون t و کای‌دو، ویژگی‌های عددی و اسمی مؤثر شناسایی شدند. سپس نمودارهای Boxplot برای مقایسه ویژگی‌های عددی و نمودارهای Countplot برای بررسی توزیع ویژگی‌های اسمی برحسب کلاس رسم گردید. همچنین، با استفاده از Heatmap همبستگی بین ویژگی‌های عددی تحلیل شد.

```python
#  Boxplot ویژگی‌های عددی (تفاوت CKD و NotCKD)
for feature in numerical_features:
    plt.figure(figsize=(6, 4))
    sns.boxplot(data=data, x='class', y=feature)
    plt.title(f'Boxplot of {feature} by Class')
    plt.tight_layout()
    plt.show()

#  Countplot ویژگی‌های اسمی (توزیع برحسب کلاس)
for feature in categorical_features:
    plt.figure(figsize=(6, 4))
    sns.countplot(data=data, x=feature, hue='class')
    plt.title(f'Countplot of {feature} by Class')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

#  Heatmap همبستگی ویژگی‌های عددی
plt.figure(figsize=(10, 8))
corr = data[numerical_features].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap between Numerical Features")
plt.tight_layout()
plt.show()
```

چند نمونه از خروجی ها :







۲. خوشه بندی

a. انتخاب ویژگی کارآمد و مهم

با توجه به اجرای مرحله‌ی کاهش ابعاد در پروژه‌ی پیشین، تنها مؤلفه‌های اصلی (PC1 تا PC4) در داده نهایی باقی مانده‌اند. در این مرحله، این مؤلفه‌ها به‌عنوان ویژگی‌های پایه برای خوشه‌بندی انتخاب شده و پس از حذف مقادیر گمشده، به کمک StandardScaler نرمال‌سازی شدند.

```
print(data.columns.tolist())
```

```
['PC1', 'PC2', 'PC3', 'PC4', 'sg', 'al', 'su', 'pcc', 'ba', 'htn', 'dm',
```

```
from sklearn.preprocessing import StandardScaler

# انتخاب ویژگی‌های PCAشده
selected_features = ['PC1', 'PC2', 'PC3', 'PC4']

# حذف ردیف‌های ناقص (اگر وجود داشته باشه)
cluster_data = data[selected_features].dropna()

# نرمال‌سازی
scaler = StandardScaler()
cluster_scaled = scaler.fit_transform(cluster_data)
```

چند نمونه اول بعد از نرمال‌سازی :

```
[51]: print("شکل داده انتخاب‌شده برای خوشه‌بندی:", cluster_data.shape)
      print("چند نمونه اول بعد از نرمال‌سازی:")
      import pandas as pd
      pd.DataFrame(cluster_scaled, columns=selected_features).head()
```

```
شکل داده انتخاب‌شده برای خوشه‌بندی: (369, 4)
چند نمونه اول بعد از نرمال‌سازی:
```

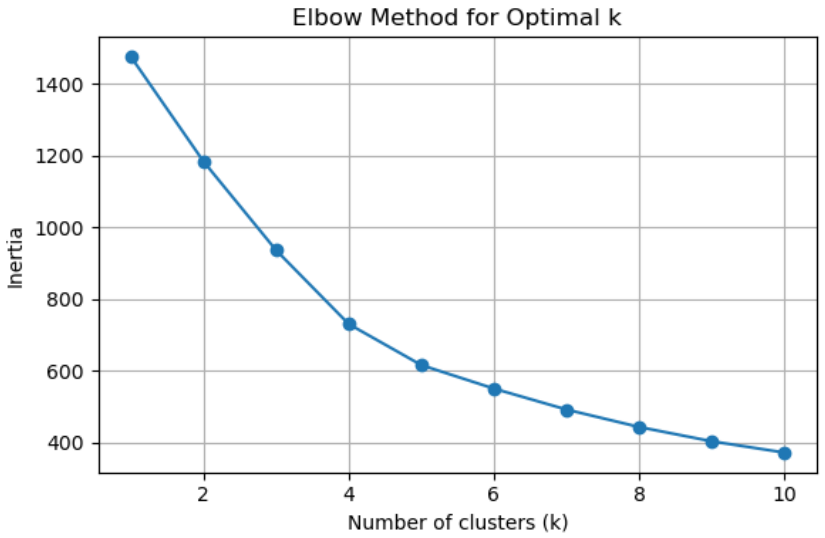
[51]:

	PC1	PC2	PC3	PC4
0	-0.230747	-1.577661	-1.296745	-1.035944
1	-0.917277	-1.729071	-0.923119	-1.517641
2	-0.344750	-2.366487	2.994964	-1.047356
3	-0.650310	-1.658779	-1.319407	-0.578786
4	-0.421488	-1.537472	-1.483851	-1.199832

b. اجرای الگوریتم Kmeans با در نظر گرفتن k مناسب (استفاده از method elbow) و تفسیر ویژگی های غالب و تاثیر گذار

در این مرحله، با استفاده از روشElbow ، مقدار بهینه‌ی k برای الگوریتم KMeans تعیین شد. بر اساس نمودار Elbow ، مقدار **k=5** به‌عنوان بهترین انتخاب شد. سپس خوشه‌بندی با این مقدار انجام شده و هر نمونه به یکی از خوشه‌ها اختصاص یافت. میانگین ویژگی‌ها در هر خوشه محاسبه شد تا ویژگی‌های غالب هر خوشه شناسایی شود.

```python
import os
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import pandas as pd
import os
os.environ["OMP_NUM_THREADS"] = "1"

# اجرای Elbow Method
inertia = []
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=0, n_init=10)
    kmeans.fit(cluster_scaled)
    inertia.append(kmeans.inertia_)

# ترسیم نمودار Elbow
plt.figure(figsize=(6, 4))
plt.plot(K_range, inertia, marker='o')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.grid(True)
plt.tight_layout()
plt.show()
```



```python
os.environ["OMP_NUM_THREADS"] = "2"

optimal_k = 5

#  اجرای خوشه‌بندی نهایی
kmeans_final = KMeans(n_clusters=optimal_k, random_state=0,
                      n_init=10)
clusters = kmeans_final.fit_predict(cluster_scaled)

#  اضافه‌کردن برچسب خوشه‌ها به داده اصلی
cluster_data_with_labels = cluster_data.copy()
cluster_data_with_labels['Cluster'] = clusters

#  محاسبه میانگین ویژگی‌ها در هر خوشه
cluster_centers = pd.DataFrame(kmeans_final.cluster_centers_,
                               columns=selected_features)
cluster_centers.index = [f"Cluster {i}" for i in range(optimal_k)]

display(cluster_centers)
display(cluster_data_with_labels.head())
```

	PC1	PC2	PC3	PC4
Cluster 0	0.080698	-0.824432	-0.661183	-0.570440
Cluster 1	0.145046	-0.977624	2.101760	-0.484732
Cluster 2	3.201655	-0.025252	-0.123405	0.814808
Cluster 3	-0.244680	0.882165	0.002928	-0.055369
Cluster 4	-0.423346	-0.865321	-0.267340	2.056304

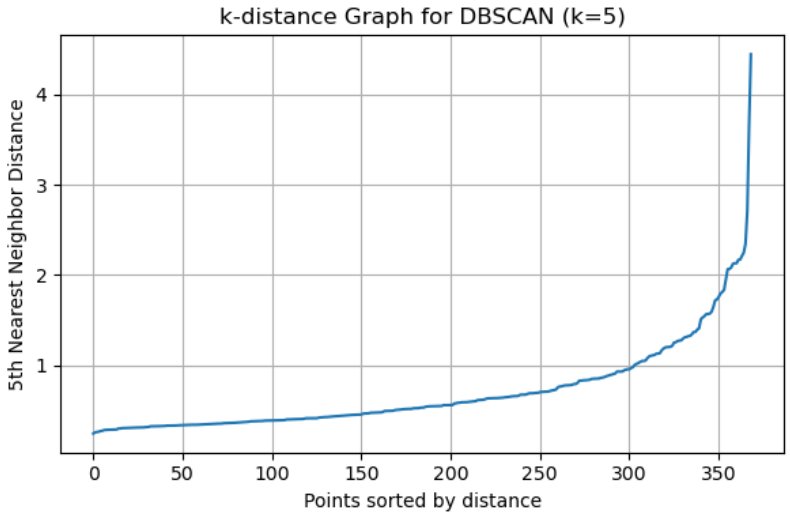
	PC1	PC2	PC3	PC4	Cluster
0	-604.953222	-188.070406	-87.350100	-48.763491	0
1	-2404.836995	-206.119764	-62.182275	-71.437701	0
2	-903.834545	-282.105077	201.743871	-49.300660	1
3	-1704.926069	-197.740282	-88.876615	-27.244335	0
4	-1105.019981	-183.279509	-99.953746	-56.477938	0

پس از اجرای KMeans با$k=5$ ، بیماران به پنج خوشه تقسیم شدند. بررسی میانگین مؤلفه‌های اصلی (PC1 تا PC4)در هر خوشه نشان داد که تفاوت عمده بین خوشه‌ها مربوط به مقادیر بالا یا پایین در PC3 و PC4 است. این مؤلفه‌ها نمایانگر تفاوت در ویژگی‌هایی مانند قند خون، اوره، کراتینین یا آنمی هستند. برای مثال، خوشه ۱ با PC3 بالا احتمالاً شامل بیماران با اختلالات شدید کلیوی بوده و خوشه ٤ با PC4 بالا ممکن است نشان‌دهنده‌ی بیماران دارای مشکلات التهابی یا متابولیکی خاص باشد.

c. اجرای الگوریتم DBSCAN و با درنظر گرفتن تنظیم بهینه پارامتر های eps وsample_min

یافتن خوشه‌هایی با شکل‌های نامنظم و شناسایی نقاط پرت بدون استفاده از تعداد خوشه ثابت.

```python
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
import numpy as np
import matplotlib.pyplot as plt

#  رو پیدا کنیم eps اول باید مقدار مناسب
neighbors = NearestNeighbors(n_neighbors=5)
neighbors_fit = neighbors.fit(cluster_scaled)
distances, indices = neighbors_fit.kneighbors(cluster_scaled)

#  رسم نمودار فاصله برای انتخاب eps
distances = np.sort(distances[:, 4])  # فاصله 5ام هر نقطه
plt.figure(figsize=(6, 4))
plt.plot(distances)
plt.title('k-distance Graph for DBSCAN (k=5)')
plt.xlabel('Points sorted by distance')
plt.ylabel('5th Nearest Neighbor Distance')
plt.grid(True)
plt.tight_layout()
plt.show()
```



```python
eps_value = 2.5
min_samples_value = 5

dbscan = DBSCAN(eps=eps_value, min_samples=min_samples_value)
db_labels = dbscan.fit_predict(cluster_scaled)

# اضافه کردن برچسب‌ها به داده
cluster_data_with_labels['DBSCAN_Cluster'] = db_labels

# نمایش چند نمونه
display(cluster_data_with_labels[['PC1', 'PC2', 'PC3', 'PC4', 'DBSCAN_Cluster']].head())

# نمایش تعداد نقاط در هر خوشه
print(cluster_data_with_labels['DBSCAN_Cluster'].value_counts())
```

	PC1	PC2	PC3	PC4	DBSCAN_Cluster
0	-604.953222	-188.070406	-87.350100	-48.763491	0
1	-2404.836995	-206.119764	-62.182275	-71.437701	0
2	-903.834545	-282.105077	201.743871	-49.300660	0
3	-1704.926069	-197.740282	-88.876615	-27.244335	0
4	-1105.019981	-183.279509	-99.953746	-56.477938	0

```
DBSCAN_Cluster
 0    368
-1      1
Name: count, dtype: int64
```

با استفاده از الگوریتم DBSCAN و انتخاب پارامترهای مناسب (eps=2.5 و min_samples=5) ، داده‌ها به چند خوشه مبتنی بر چگالی تقسیم شدند. نقاط با برچسب 1-به عنوان داده‌های پرت شناسایی شدند. برخلاف KMeans ، DBSCAN قادر است ساختارهای پیچیده‌تری را شناسایی کند و در برابر نویز مقاوم‌تر عمل می‌کند.

d. مقایسه و ارزیابی خوشه بندی های انجام شده با استفاده از هر دو الگوریتم به صورتی عددی و بصری

بخش اول: بررسی آماری و تعداد اعضای هر خوشه:

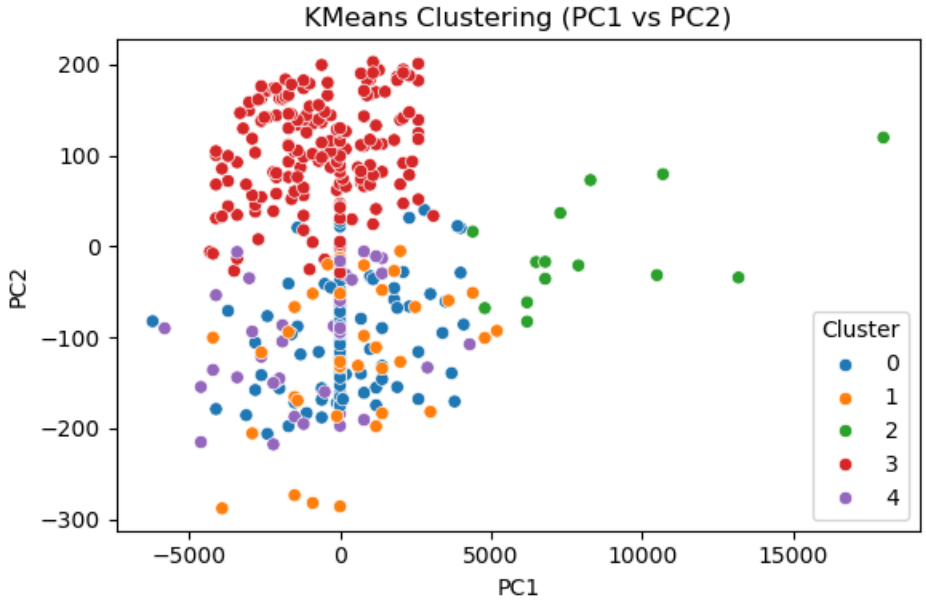
```
# مقایسه تعداد نمونه در هر خوشه KMeans
print("KMeans Cluster Counts:")
print(cluster_data_with_labels['Cluster'].value_counts())
print()

# مقایسه تعداد نمونه در هر خوشه DBSCAN
print("DBSCAN Cluster Counts (شامل نقاط پرت با 1-):")
print(cluster_data_with_labels['DBSCAN_Cluster'].value_counts())
```

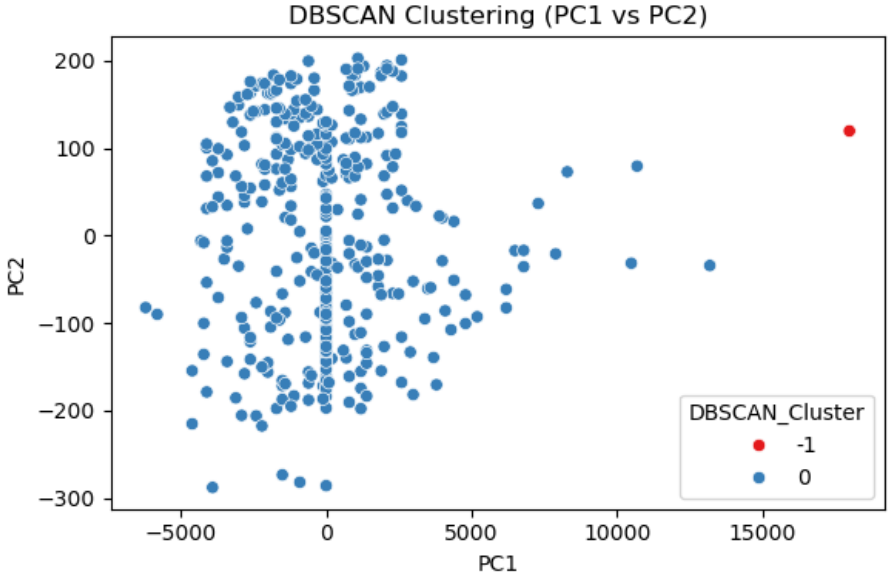
```
KMeans Cluster Counts:
Cluster
3    176
0    104
1     38
4     37
2     14
Name: count, dtype: int64

DBSCAN Cluster Counts (شامل نقاط پرت با 1-):
DBSCAN_Cluster
 0    368
-1      1
Name: count, dtype: int64
```

بخش دوم: رسم نمودار خوشه‌ها در فضای دوبعدی (مثلاً PC1 و PC2)

```
import seaborn as sns

# رسم خوشه‌ها KMeans
plt.figure(figsize=(6, 4))
sns.scatterplot(data=cluster_data_with_labels, x='PC1', y='PC2', hue='Cluster', palette='tab10')
plt.title('KMeans Clustering (PC1 vs PC2)')
plt.tight_layout()
plt.show()

#  رسم خوشه‌ها DBSCAN
plt.figure(figsize=(6, 4))
sns.scatterplot(data=cluster_data_with_labels, x='PC1', y='PC2', hue='DBSCAN_Cluster', palette='Set1')
plt.title('DBSCAN Clustering (PC1 vs PC2)')
plt.tight_layout()
plt.show()
```

دو الگوریتم خوشه‌بندی KMeans و DBSCAN بر روی داده‌های آزمایشگاهی بیماران اجرا شدند .KMeans داده‌ها را به ۵ خوشه تقسیم کرد، در حالی که DBSCAN بر اساس چگالی، چند خوشه اصلی و تعدادی نقطه پرت (برچسب -۱) شناسایی کرد. نمودارهای دوبعدی نشان دادند که DBSCAN توانایی بهتری در تشخیص ساختارهای نامنظم و نقاط نویزی دارد، در حالی که KMeans عملکرد مناسبی در ساختارهای کروی و یکنواخت ارائه می‌دهد.


KMeans Clustering (PC1 vs PC2)
PC2
PC1
200
100
0
−100
−200
−300
−5000
0
5000
10000
15000
Cluster
0
1
2
3
4


DBSCAN Clustering (PC1 vs PC2)
PC2
PC1
200
100
0
−100
−200
−300
−5000
0
5000
10000
15000
DBSCAN_Cluster
-1
0

۲. طبقه بندی (پیشبینی ابتلا یا عدم ابتلا به CKD بر اساس ویژگی های کلینیکی و آزمایشگاهی با استفاده از الگوریتمهای یادگیری نظارت شده)

a. انتخاب ۳ الگوریتم مناسب برای داده مورد استفاده و بیان دلایل انتخاب(از میان الگوریتم های SVM و RandomForest، KNN، NaiveBayse، DecisionTree،LogisticRegression)

دلیل انتخاب الگوریتم‌ها:

لاجستیک‌رگرسیون: مدل پایه و ساده برای طبقه‌بندی دودویی، خوب برای شروع و مقایسه.

رندوم فارست: مدل قوی، مقاوم در برابر overfitting و قادر به تشخیص روابط پیچیده بین ویژگی‌ها.

KNN: مدل ساده، مبتنی بر تشابه نمونه‌ها و ساختار محلی داده‌ها.

b. آموزش و اجرای ۳ مدل منتخب برروی داده ی انتخاب شده با در نظر گرفتن ویژگی class به عنوان label

c. تقسیم بندی داده به داده آموزش و تست به نسبت ۷۰ به ۳۰

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# حذف ردیف‌هایی که کلاس ندارند
data = data.dropna(subset=['class'])

# تبدیل کلاس به برچسب عددی
label_encoder = LabelEncoder()
data['class_encoded'] = label_encoder.fit_transform(data['class'])

# ویژگی‌ها (X) و برچسب‌ها (y)
X = cluster_data
y = data.loc[X.index, 'class_encoded']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
```

سه الگوریتمLogistic Regression ، Random Forestو KNN برای پیش‌بینی ابتلا به CKD انتخاب شدند. ابتدا ستون کلاس به صورت عددی کدگذاری شد و داده‌ها به ویژگی‌ها و برچسب‌ها تقسیم شدند. سپس داده‌ها به نسبت ۷۰٪ آموزش و ۳۰٪ تست با حفظ توزیع کلاس‌ها تقسیم شدند. مدل‌ها با داده آموزش آموزش داده شدند تا آماده ارزیابی شوند.

d. استفاده از روش Fold5 برایvalidation

```python
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

# تعریف مدل‌ها
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'KNN': KNeighborsClassifier()
}

# تعریف stratified 5-fold برای حفظ توزیع کلاس‌ها
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# اجرای cross validation و گرفتن نمرات دقت (accuracy)
for name, model in models.items():
    scores = cross_val_score(model, X, y, cv=kf, scoring='accuracy')
    print(f'{name} 5-Fold CV Accuracy Scores: {scores}')
    print(f'{name} Mean CV Accuracy: {scores.mean():.4f}\n')
```

```
Logistic Regression 5-Fold CV Accuracy Scores: [1.         1.         0.98648649 1.         1.        ]
Logistic Regression Mean CV Accuracy: 0.9973

Random Forest 5-Fold CV Accuracy Scores: [0.97297297 0.94594595 1.         0.98648649 0.94520548]
Random Forest Mean CV Accuracy: 0.9701

KNN 5-Fold CV Accuracy Scores: [0.91891892 0.83783784 0.91891892 0.90540541 0.87671233]
KNN Mean CV Accuracy: 0.8916
```

در این مرحله، برای ارزیابی دقیق‌تر عملکرد مدل‌هایLogistic Regression ، Random Forestو KNN از روش اعتبارسنجی متقاطع ۵-تایی استفاده شد. این روش باعث شد که مدل‌ها روی بخش‌های مختلف داده آموزش و تست شوند و میانگین دقت آنها گزارش شود. نتایج نشان‌دهنده پایداری و قابلیت تعمیم بهتر مدل‌ها نسبت به ارزیابی ساده با تقسیم ثابت است.

تحلیل نتایج :

Logistic Regression دقت بسیار بالا (حدود ۹۹٫۷٪) در ۵ فولد داشته که نشان می‌دهد مدل پایه در این داده عملکرد خیلی خوبی دارد.

Random Forest دقت متوسط رو به بالا (~۹۷٪) داره که کمی نوسان در فولدها دیده می‌شود ولی باز هم عالیه.

KNN نسبت به دو مدل دیگه دقت پایین‌تری (حدود ۸۹٪) داره که معمولاً برای داده‌های پیچیده‌تر طبیعیه چون KNN خیلی وابسته به توزیع محلی داده‌هاست.

e. ارزیابی مدل های آموزش دیده با استفاده از پارامتر هایAcc، Precision، Recall وScore1F

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# تابع ارزیابی مدل روی داده تست
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    return {
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1-Score': f1_score(y_test, y_pred)
    }

# ابتدا باید مدل‌ها رو تعریف و آموزش بدی
model_lr = LogisticRegression(max_iter=1000, random_state=42)
model_rf = RandomForestClassifier(random_state=42)
model_knn = KNeighborsClassifier()

model_lr.fit(X_train, y_train)
model_rf.fit(X_train, y_train)
model_knn.fit(X_train, y_train)

# ارزیابی مدل‌ها
results = {}
for name, model in [('Logistic Regression', model_lr), ('Random Forest', model_rf), ('KNN', model_knn)]:
    results[name] = evaluate_model(model, X_test, y_test)
    print(f'{name} Evaluation:')
    for metric, value in results[name].items():
        print(f'  {metric}: {value:.4f}')
    print()
```

در این مرحله، مدل‌هایLogistic Regression ، Random Forestو KNN پس از آموزش روی داده‌های آموزش، با داده‌های تست ارزیابی شدند. معیارهای دقت(Accuracy) ، دقت مثبت درست(Precision) ، حساسیت (Recall) و معیار F1-Score برای هر مدل محاسبه و گزارش شد. این معیارها کمک می‌کنند تا عملکرد مدل‌ها در پیش‌بینی ابتلا به بیماری CKD به شکل دقیق‌تر و همه‌جانبه‌تری بررسی شود.

Logistic Regression دقت و معیارهای کامل برابر با ۱ داره که یعنی روی داده تست عملکرد بی‌نقص داشته (احتمالاً داده‌ها به خوبی تفکیک شده یا مدل خیلی خوب fit شده).

Random Forest هم دقت بالا (۹۶,٤٪) و مقادیرPrecision ، Recallو F1 بسیار خوب داره (حدود ۹۵,۵٪) که عملکرد قوی‌ای است.

KNN دقت ۹۰٪ داره، Precisionکمتر (۸۰,۳٪) اما Recall برابر ۱ هست؛ یعنی KNN تمام نمونه‌های مثبت را پیدا کرده ولی تعدادی نمونه منفی را اشتباه مثبت تشخیص داده (Precision پایین‌تر) .

f. تنظیم پارامتر های مدل های انتخاب شده به منظور یافتن مدل بهینه

در این مرحله، با استفاده از روش Grid Search و اعتبارسنجی متقاطع ۵-تایی، پارامترهای مدل‌هایLogistic Regression ، Random Forestو KNN بهینه شدند. این فرایند با جستجوی مقادیر مختلف پارامترها انجام گرفت تا بهترین ترکیب پارامترها که منجر به بیشترین دقت می‌شود، پیدا شود. نتایج نشان داد هر مدل با تنظیم بهینه پارامترها عملکرد بهتری نسبت به حالت پیش‌فرض خواهد داشت.

```python
from sklearn.model_selection import GridSearchCV

# 1. Logistic Regression
param_grid_lr = {
    'C': [0.01, 0.1, 1, 10, 100],  # مقدار تنظیم وزن جریمه
    'solver': ['liblinear', 'lbfgs']
}

grid_lr = GridSearchCV(LogisticRegression(max_iter=1000, random_state=42), param_grid_lr, cv=5, scoring='accuracy')
grid_lr.fit(X_train, y_train)

print('Best params for Logistic Regression:', grid_lr.best_params_)
print('Best CV accuracy:', grid_lr.best_score_)

# 2. Random Forest
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

grid_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf, cv=5, scoring='accuracy')
grid_rf.fit(X_train, y_train)

print('Best params for Random Forest:', grid_rf.best_params_)
```

```python
print('Best params for Random Forest:', grid_rf.best_params_)
print('Best CV accuracy:', grid_rf.best_score_)

# 3. KNN
param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

grid_knn = GridSearchCV(KNeighborsClassifier(), param_grid_knn, cv=5, scoring='accuracy')
grid_knn.fit(X_train, y_train)

print('Best params for KNN:', grid_knn.best_params_)
print('Best CV accuracy:', grid_knn.best_score_)
```

```
Best params for Logistic Regression: {'C': 0.01, 'solver': 'lbfgs'}
Best CV accuracy: 0.9961538461538462
Best params for Random Forest: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
Best CV accuracy: 0.9806938159879337
Best params for KNN: {'metric': 'manhattan', 'n_neighbors': 7, 'weights': 'distance'}
Best CV accuracy: 0.9147812971342383
```

پارامتر بهینه برای Logistic Regression: C = 0.01 و solver = 'lbfgs' انتخاب شده که نشان‌دهنده مدل با تنظیم قوی‌تر (کاهش مقدار C باعث افزایش منظم‌سازی می‌شود) .

دقت اعتبارسنجی متقاطع (CV accuracy) حدود ۹۹,٦٪ است که بسیار عالی و سازگار با نتایج قبلی می‌باشد.

g. مقایسه مدل های منتخب با در نظر گرفتن نتایج معیار های ارزیابی به صورت عددی و مقایسه نمودار ROC آنها

در این مرحله، عملکرد سه مدلLogistic Regression ، Random Forestو KNN از نظر معیارهای عددی مقایسه شد. همچنین نمودار ROC برای هر مدل رسم شد تا توانایی تشخیص کلاس مثبت در برابر کلاس منفی به صورت بصری بررسی شود.

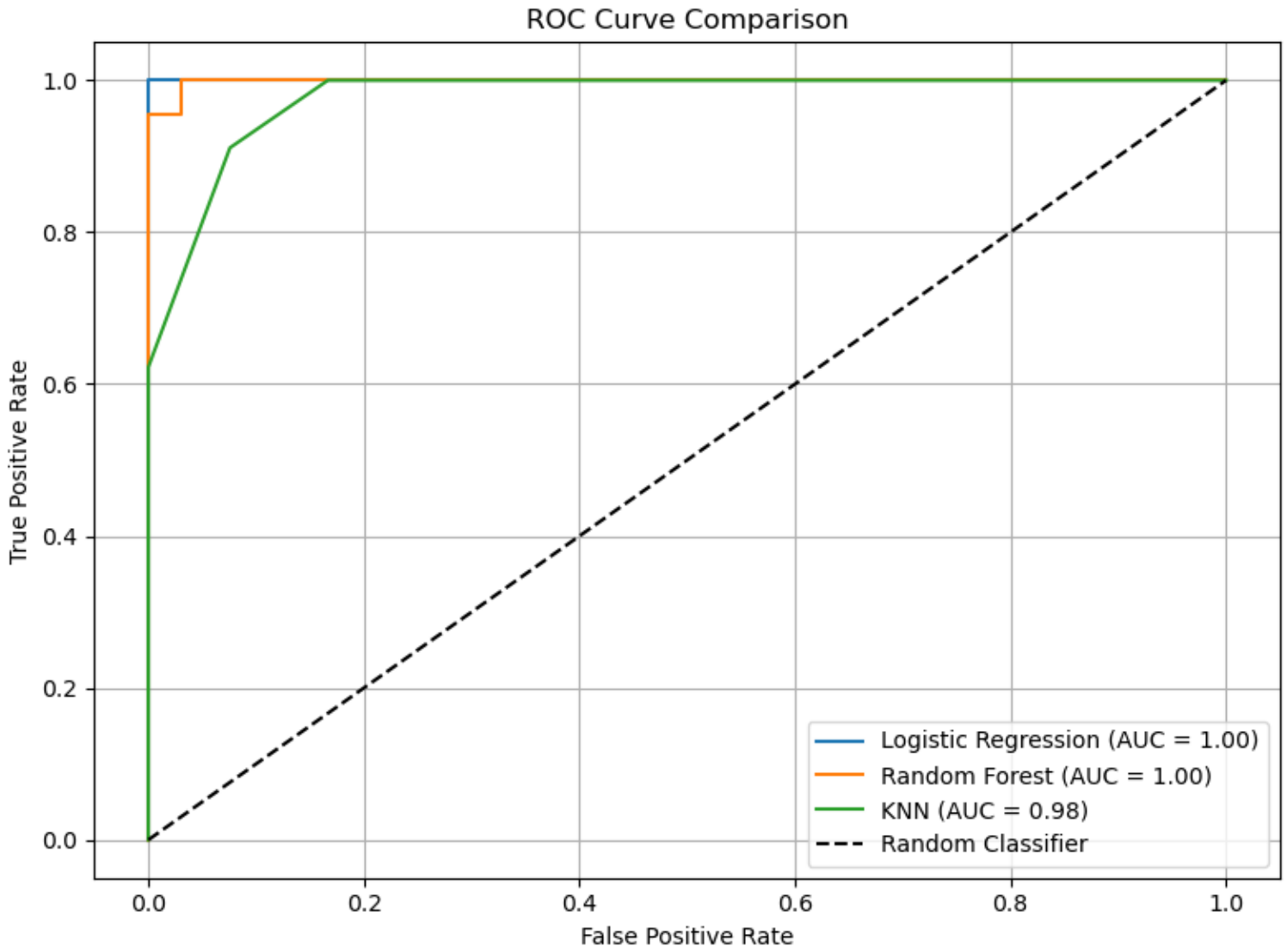
مدل Logistic Regression با AUC نزدیک به ۱ عملکردی عالی داشت Random Forest .نیز عملکردی نزدیک به آن و با دقت بالا نشان داد. در مقابل، مدل KNN دقت پایین‌تری داشت و نمودار ROC آن کمی ضعیف‌تر بود. با این حال، هر سه مدل توانستند طبقه‌بندی نسبتاً دقیقی روی داده‌های CKD داشته باشند.

```
# 3 تبدیل نتایج مرحله e. به جدول
results_df = pd.DataFrame(results).T
print(results_df)
```

	Accuracy	Precision	Recall	F1-Score
Logistic Regression	1.000000	1.000000	1.000000	1.000000
Random Forest	0.963964	0.955556	0.955556	0.955556
KNN	0.900901	0.803571	1.000000	0.891089

مدل	عملکرد
Logistic Regression	دقت و سایر معیارها کامل = احتمالاً داده خیلی خوش‌تفکیک هست یا مدل بیش‌ازحد ساده‌سازی‌شده (خطر overfitting کم اما وجود داره اگر نمونه کم باشه).
Random Forest	عملکرد قوی، با دقت حدود ۹۶٪، Precisionو Recall برابر — نشون می‌ده تعادل خوبی بین شناسایی مثبت‌ها و منفی‌ها داره.
KNN	دقت کمتر (۹۰٪) ولی Recall = 1 یعنی همه نمونه‌های مثبت رو پیدا کرده، ولی Precision پایین‌تر یعنی تعداد زیادی false positive هم داشته.

```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# پیش‌بینی احتمالات برای کلاس مثبت (label=1)
y_scores_lr = model_lr.predict_proba(X_test)[:, 1]
y_scores_rf = model_rf.predict_proba(X_test)[:, 1]
y_scores_knn = model_knn.predict_proba(X_test)[:, 1]

# محاسبه FPR و TPR برای هر مدل
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_scores_lr)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_scores_rf)
fpr_knn, tpr_knn, _ = roc_curve(y_test, y_scores_knn)

# محاسبه AUC
auc_lr = auc(fpr_lr, tpr_lr)
auc_rf = auc(fpr_rf, tpr_rf)
auc_knn = auc(fpr_knn, tpr_knn)

# رسم نمودار ROC
plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, label=f'Logistic Regression (AUC = {auc_lr:.2f})')
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {auc_rf:.2f})')
plt.plot(fpr_knn, tpr_knn, label=f'KNN (AUC = {auc_knn:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```


ROC Curve Comparison
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression (AUC = 1.00)
Random Forest (AUC = 1.00)
KNN (AUC = 0.98)
Random Classifier

تفسیر نمودارROC :

- Logistic Regression (خط آبی):
  - دقیقاً به گوشه بالا-چپ خم شده → یعنی عملکرد ایده‌آل!
  - AUC = 1.00 → نشان‌دهنده‌ی مدل کاملاً بی‌نقص در تشخیص کلاس‌هاست.
- Random Forest (خط نارنجی):
  - منحنی‌ای خیلی نزدیک به Logistic عالی، AUC = 1.00
  - احتمالاً چند نمونه مرزی داشته ولی همچنان خیلی خوبه.
- KNN (خط سبز) :
  - عملکرد خوب ولی نه به خوبی دو مدل دیگر
  - AUC = 0.98 → یعنی مدل توانسته ۹۸٪ از تفکیک‌پذیری بین کلاس‌ها را حفظ کند، ولی در مقایسه با بقیه کمی ضعیف‌تر است.
  - کمی از خط ایده‌آل فاصله گرفته، که با نتایج عددی‌اش هم سازگاره (Precision پایین‌تر ولی Recall بالا)
- خط‌چین مورب:(Random Classifier)
  - خط مرجع برای حدس تصادفی (AUC=0.5) که مدل‌ها باید بالای آن قرار بگیرند — و مدل‌ها خیلی بالاتر از این خط هستند.