به نام خدا

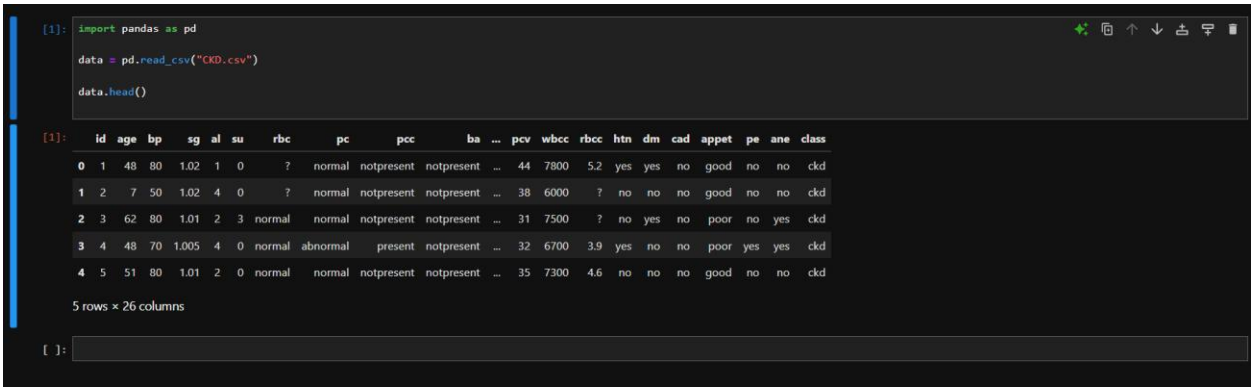
گزارش پروژه داده کاوی

پاکسازی و پیش پردازش داده های بیماری مزمن کلیوی(CKD)

مرحله ۱: بررسی اولیه داده‌ها

a. بارگذاری مجموعه داده و بررسی ساختار آن

اولین کار اینه که کتابخانه‌های لازم رو ایمپورت کنیم و داده رو بارگذاری کنیم:



```
[1]: import pandas as pd

     data = pd.read_csv("CKD.csv")

     data.head()
```

[1]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	1	48	80	1.02	1	0	?	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	2	7	50	1.02	4	0	?	normal	notpresent	notpresent	...	38	6000	?	no	no	no	good	no	no	ckd
2	3	62	80	1.01	2	3	normal	normal	notpresent	notpresent	...	31	7500	?	no	yes	no	poor	no	yes	ckd
3	4	48	70	1.005	4	0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	5	51	80	1.01	2	0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd

5 rows × 26 columns

[ ]:

در این مرحله، فایل csv داده مربوط به بیماری مزمن کلیوی (CKD) را با استفاده از کتابخانه pandas در محیط JupyterLab بارگذاری کردیم. داده شامل ٤٠٠ نمونه و ٢٥ ویژگی می‌باشد که ترکیبی از ویژگی‌های عددی و طبقه‌ای هستند. همچنین با استفاده از تابع head() پنج سطر اول داده را برای بررسی اولیه نمایش دادیم.

b. نمایش اطلاعات آماری مانند تعداد نمونه ها، نوع داده های هر ویژگی، میزان داده های گمشده

می‌خوایم بفهمیم که چند ستون داریم، چه نوع داده‌هایی دارن، آیا داده‌های گمشده هست یا نه.

```
[7]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      400 non-null    int64
 1   age     391 non-null    object
 2   bp      388 non-null    object
 3   sg      353 non-null    object
 4   al      354 non-null    object
 5   su      351 non-null    object
 6   rbc     248 non-null    object
 7   pc      335 non-null    object
 8   pcc     396 non-null    object
 9   ba      396 non-null    object
 10  bgr     356 non-null    object
 11  bu      381 non-null    object
 12  sc      383 non-null    object
 13  sod     313 non-null    object
 14  pot     312 non-null    object
 15  hemo    348 non-null    object
 16  pcv     329 non-null    object
 17  wbcc    294 non-null    object
 18  rbcc    269 non-null    object
 19  htn     398 non-null    object
 20  dm      398 non-null    object
 21  cad     398 non-null    object
 22  appet   399 non-null    object
 23  pe      399 non-null    object
 24  ane     399 non-null    object
 25  class   400 non-null    object
dtypes: int64(1), object(25)
memory usage: 81.4+ KB
```

در این مرحله، با استفاده از تابع info()اطلاعاتی از قبیل تعداد ردیف‌ها، نام ستون‌ها، نوع داده هر ستون و تعداد مقادیر غیر null بررسی شد. این بررسی مشخص می‌کند که در کدام ستون‌ها مقادیر گمشده وجود دارد و چه نوع داده‌هایی در مجموعه وجود دارد. اجرای این کد معمولاً نشون میده که بعضی ستون‌ها مقادیر objectدارن در حالی که انتظار داشتیم عددی باشن. بعداً اصلاحش می‌کنیم.

می‌خوایم بفهمیم در هر ستون چند تا مقدار گمشده (missing) داریم. توی دیتاست ما، مقدارهای گمشده با "?"نمایش داده شدن که باید شناسایی و تبدیلشون کنیم.

```
memory usage: 81.4+ KB
```

```
[9]: import numpy as np

     data.replace("?", np.nan, inplace=True)

     data.isnull().sum()
```

```
[9]: id          0
     age         9
     bp         12
     sg         47
     al         46
     su         49
     rbc       152
     pc         65
     pcc         4
     ba          4
     bgr        44
     bu         19
     sc         17
     sod        87
     pot        88
     hemo       52
     pcv        71
     wbcc      106
     rbcc      131
     htn         2
     dm          2
     cad         2
     appet       1
     pe          1
     ane         1
     class       0
     dtype: int64
```

در این مرحله ابتدا با استفاده از دستور ()replace، تمام مقادیر "?"که نشان‌دهنده داده‌های گمشده بودند، به مقدار استاندارد NaNدر پایتون تبدیل شدند. سپس با استفاده از ()isnull().sumتعداد مقادیر گمشده در هر ستون محاسبه شد. این اطلاعات به ما کمک می‌کند در مراحل بعد تصمیم بگیریم که این داده‌های گمشده را چگونه مدیریت کنیم (حذف، جایگزینی و...).

به‌دست آوردن اطلاعات آماری از داده‌ها مثل میانگین، میانه، انحراف معیار و... برای ستون‌های عددی. این کمک می‌کنه تا درک بهتری از توزیع داده‌ها پیدا کنیم و همچنین هر گونه مقدار غیرطبیعی رو شناسایی کنیم.

```
143]: data.describe()
```

143]:

	id	age	bp	bgr	bu	sc	sod	pot	hemo	pcv	wbcc	rbcc
count	400.000000	391.000000	388.000000	356.000000	381.000000	383.000000	313.000000	312.000000	348.000000	329.000000	294.000000	269.000000
mean	200.500000	51.483376	76.469072	148.036517	57.425722	3.072454	137.528754	4.627244	12.526437	38.884498	8406.122449	4.707435
std	115.614301	17.169714	13.683637	79.281714	50.503006	5.741126	10.408752	3.193904	2.912587	8.990105	2944.474190	1.025323
min	1.000000	2.000000	50.000000	22.000000	1.500000	0.400000	4.500000	2.500000	3.100000	9.000000	2200.000000	2.100000
25%	100.750000	42.000000	70.000000	99.000000	27.000000	0.900000	135.000000	3.800000	10.300000	32.000000	6500.000000	3.900000
50%	200.500000	55.000000	80.000000	121.000000	42.000000	1.300000	138.000000	4.400000	12.650000	40.000000	8000.000000	4.800000
75%	300.250000	64.500000	80.000000	163.000000	66.000000	2.800000	142.000000	4.900000	15.000000	45.000000	9800.000000	5.400000
max	400.000000	90.000000	180.000000	490.000000	391.000000	76.000000	163.000000	47.000000	17.800000	54.000000	26400.000000	8.000000

در این مرحله، با استفاده از متد describe()یک تحلیل آماری کلی از تمامی ستون‌های عددی داده‌ها انجام شد. این تحلیل شامل محاسبه میانگین، انحراف معیار، مینیمم، ماکسیمم و مقادیر چارک‌ها (quartiles) برای هر ستون عددی است. این اطلاعات به شناسایی داده‌های غیرمعمول یا ناهنجار کمک می‌کند.

c. بررسی بصری داده ها با استفاده از نمودار های مطلوب

میخواهیم با استفاده از نمودارها و تجزیه و تحلیل بصری داده‌ها، الگوهای جالب یا مشکلات احتمالی (مثل داده‌های پرت) رو شناسایی کنیم.

```
[228]: numeric_cols = ['age', 'bp', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbcc', 'rbcc']

       for col in numeric_cols:
           data[col] = pd.to_numeric(data[col], errors='coerce')
```

در این مرحله، ستون‌هایی که باید عددی باشند (مثل age, bp, bgr...) با استفاده از pd.to_numeric() به نوع عددی تبدیل شدند.

با errors='coerce'، مقادیر غیرقابل‌تبدیل مثل ? به NaN تبدیل شدند تا در مراحل بعدی راحت‌تر مدیریت شوند.

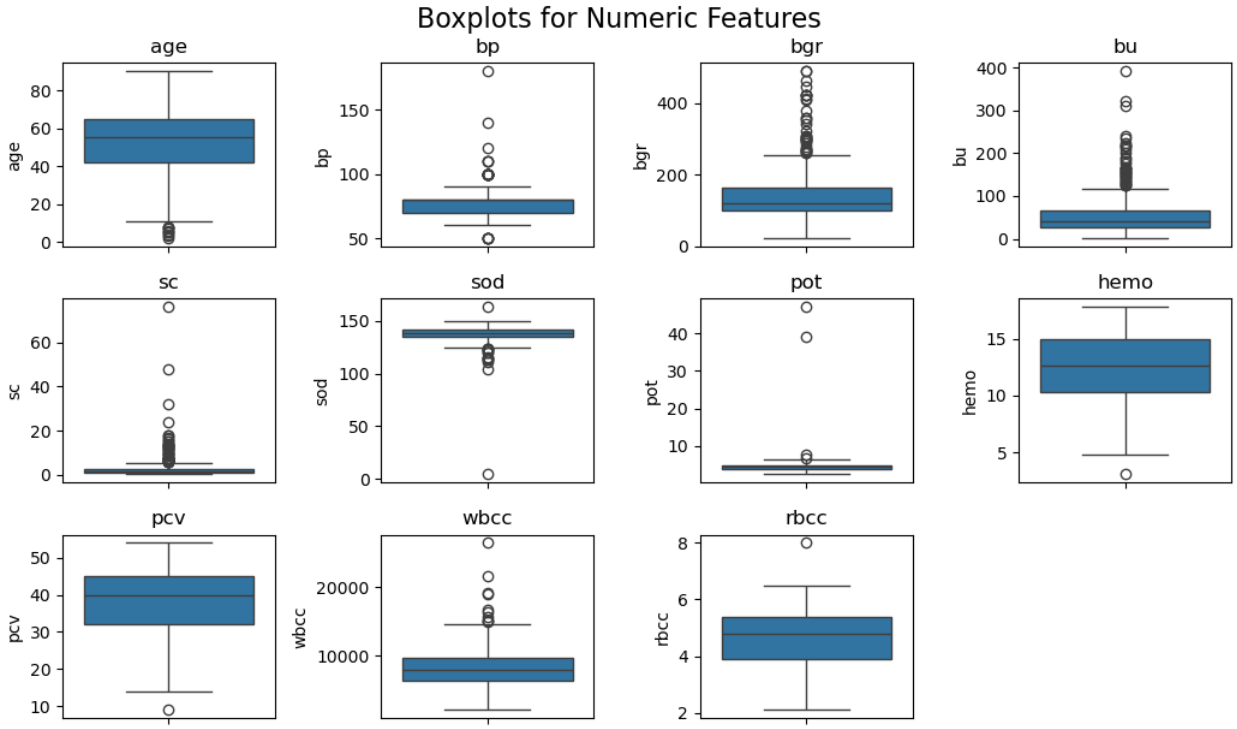
رسم boxplot:

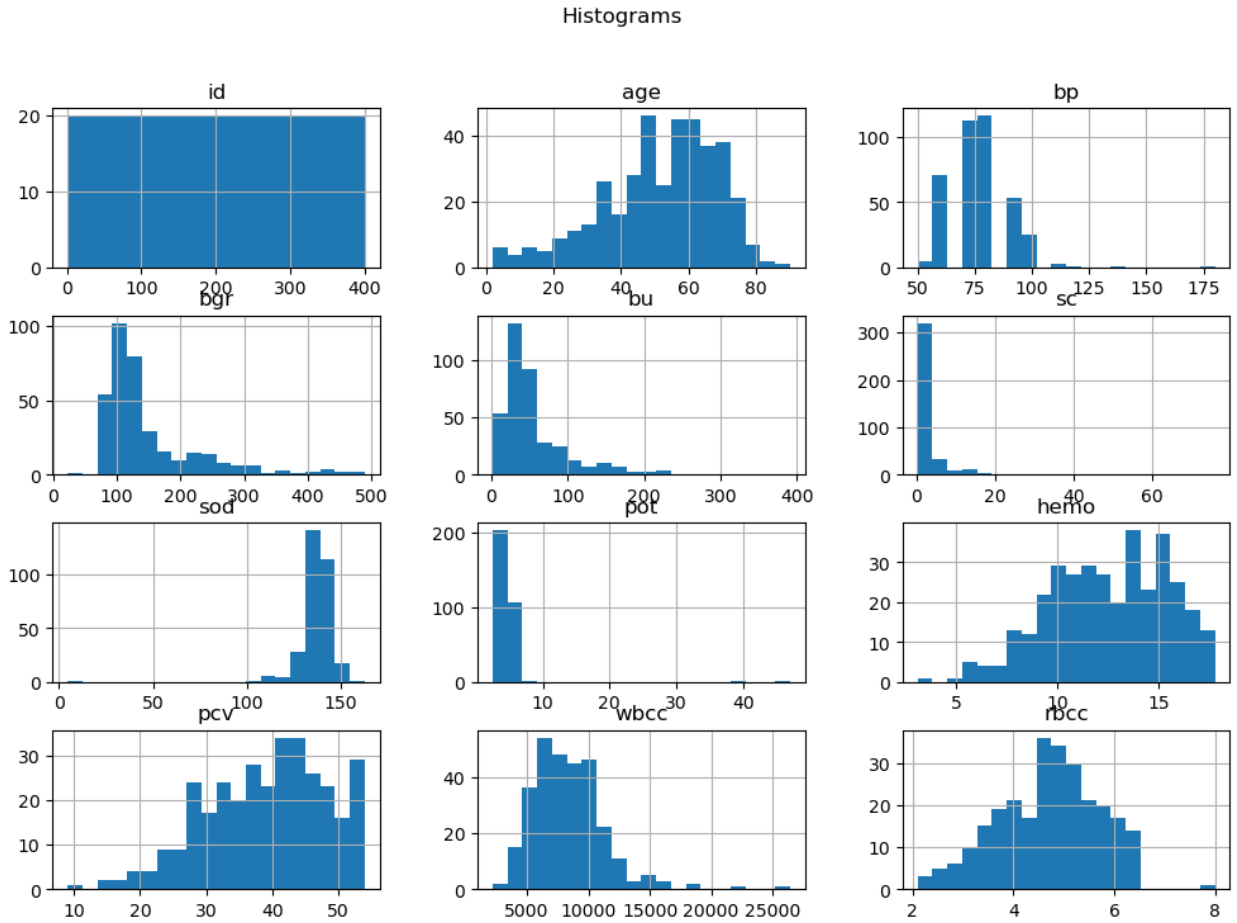
```
4]: import matplotlib.pyplot as plt
    import seaborn as sns

    plt.figure(figsize=(10.5, 6))

    for i, col in enumerate(numeric_cols):
        plt.subplot(3, 4, i + 1)
        sns.boxplot(y=data[col])
        plt.title(col)

    plt.tight_layout()
    plt.suptitle("Boxplots for Numeric Features", fontsize=16, y=1.02)
    plt.show()
```

رسم هیستوگرام:

```
[240]: data.hist(figsize=(12,8), bins=20)
       plt.suptitle("Histograms")
       plt.show()
```

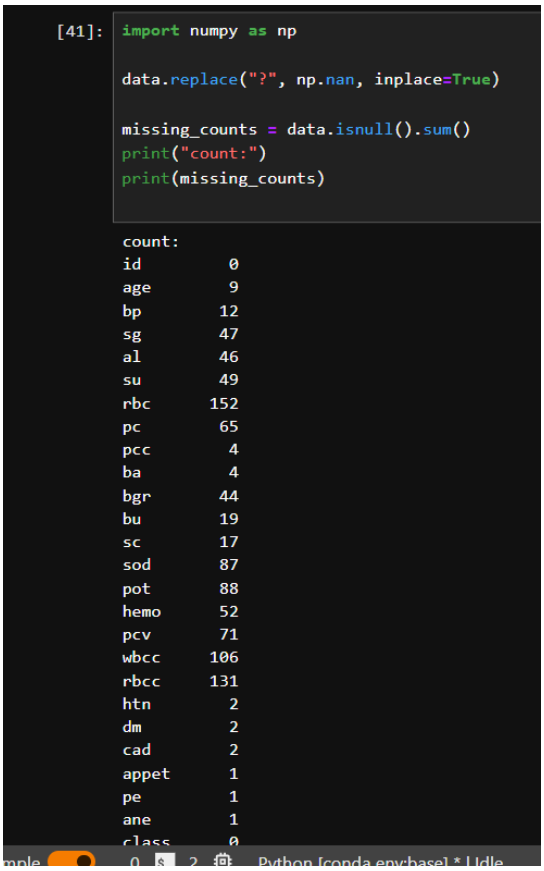
در این مرحله با استفاده از نمودارهای جعبه‌ای (Boxplot) و هیستوگرام، توزیع داده‌های عددی بررسی شد.
نمودارهای جعبه‌ای کمک می‌کنند تا داده‌های پرت (Outliers) شناسایی شوند و هیستوگرام‌ها نشان‌دهنده توزیع کلی داده‌ها هستند. این تحلیل بصری به شناسایی مشکلات احتمالی در داده‌ها کمک می‌کند که باید در مراحل بعدی آن‌ها را اصلاح کنیم.


Boxplots for Numeric Features
age
bp
bgr
bu
sc
sod
pot
hemo
pcv
wbcc
rbcc


Histograms
id
age
bp
bgr
bu
sc
sod
pot
hemo
pcv
wbcc
rbcc

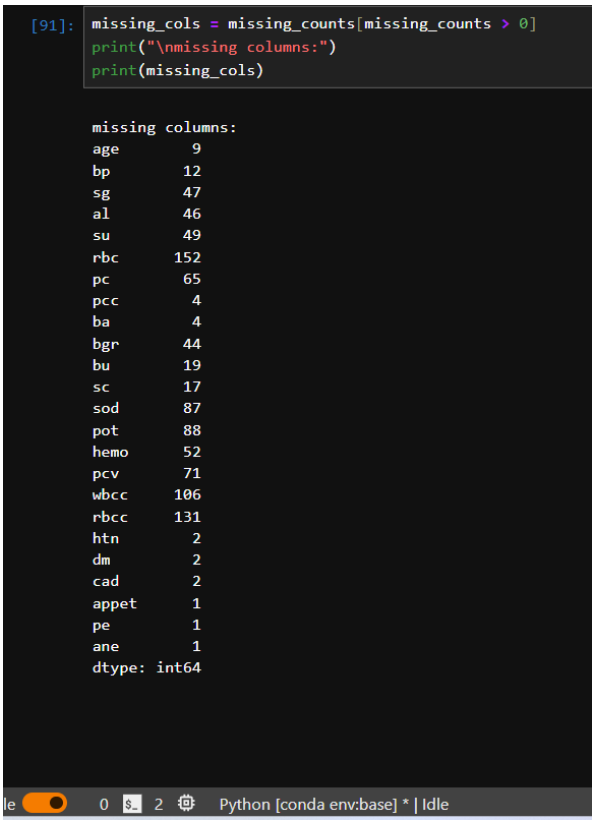
مرحله ۲: شناسایی مقادیر گمشده:

a. شناسایی مقادیر گمشده در هر ویژگی



```
[41]: import numpy as np

data.replace("?", np.nan, inplace=True)

missing_counts = data.isnull().sum()
print("count:")
print(missing_counts)
```

```
count:
id          0
age         9
bp         12
sg         47
al         46
su         49
rbc       152
pc         65
pcc         4
ba          4
bgr        44
bu         19
sc         17
sod        87
pot        88
hemo       52
pcv        71
wbcc      106
rbcc      131
htn         2
dm          2
cad         2
appet       1
pe          1
ane         1
class       0
```



```
[91]: missing_cols = missing_counts[missing_counts > 0]
print("\nmissing columns:")
print(missing_cols)
```

```
missing columns:
age         9
bp         12
sg         47
al         46
su         49
rbc       152
pc         65
pcc         4
ba          4
bgr        44
bu         19
sc         17
sod        87
pot        88
hemo       52
pcv        71
wbcc      106
rbcc      131
htn         2
dm          2
cad         2
appet       1
pe          1
ane         1
dtype: int64
```

در این مرحله ابتدا تمام مقادیر "?" که نشان‌دهنده داده‌های گمشده بودند با مقدار NaN جایگزین شدند. سپس با استفاده از توابع پایتون، تعداد مقادیر گمشده در هر ستون محاسبه شد. نتایج این مرحله مشخص کرد که برخی از ستون‌ها دارای داده‌های ناقص هستند که باید در مرحله بعد مدیریت شوند.

b. حذف نمونه‌هایی که مقدار گمشده زیادی دارند:

حالا باید تصمیم بگیریم که چه تعداد از مقادیر گمشده برای هر نمونه (ردیف) قابل قبول است. مثلا اگر بیشتر از ٪ ۳۰ داده‌ها گمشده باشه، اون نمونه حذف می‌شود.

۱. ابتدا درصد مقادیر گمشده در هر ردیف رو محاسبه می‌کنیم:

```
6]: missing_percentage = data.isnull().mean(axis=1) * 100
    print(missing_percentage)

    0      11.538462
    1      19.230769
    2      11.538462
    3       0.000000
    4       7.692308
             ...
    395     0.000000
    396     0.000000
    397     0.000000
    398     0.000000
    399     0.000000
    Length: 400, dtype: float64
```

۲. حالا نمونه‌هایی که درصد مقادیر گمشده آنها بیشتر از ٪ ۳۰ است رو حذف می‌کنیم:

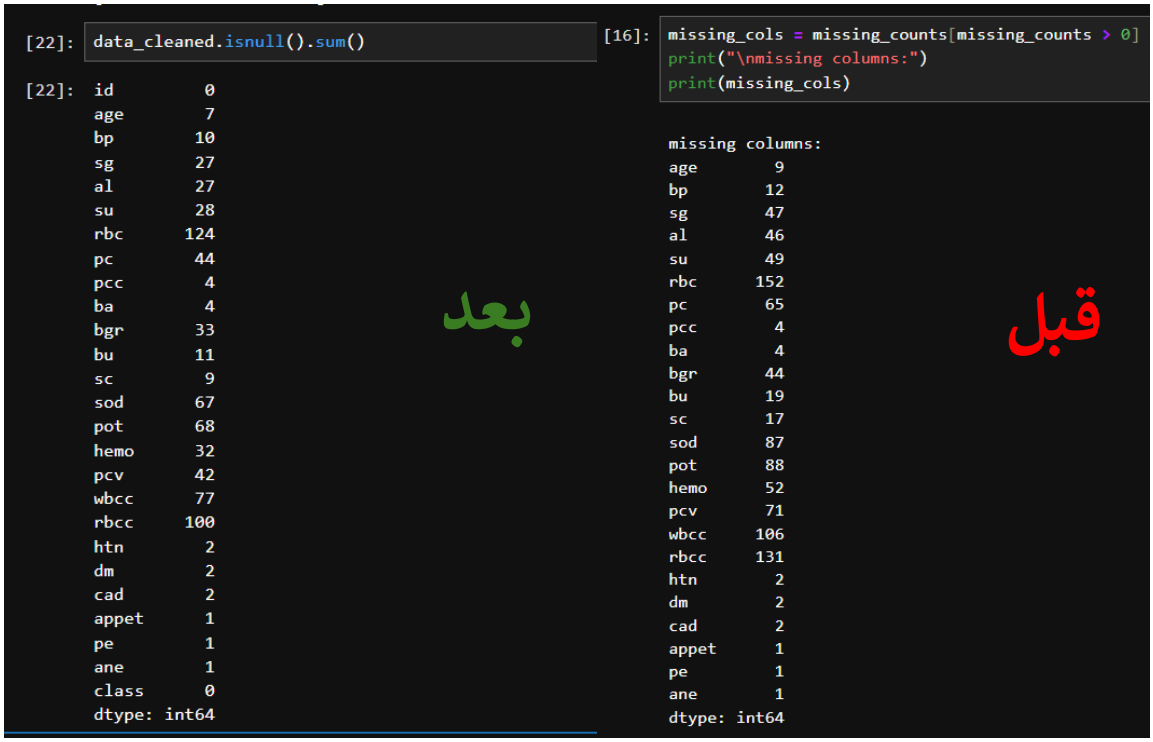
```
]: threshold = 30
   data_cleaned = data[missing_percentage < threshold]

]: print(f"Original data shape: {data.shape}")
   print(f"After removing rows with >30% missing: {data_cleaned.shape}")

   Original data shape: (400, 26)
   After removing rows with >30% missing: (369, 26)
```

و بعد بررسی می کنیم که بعد از پاک سازی چند تا از نمونه ها پاک شدن. اینجا ۳۱ تا نمونه پاک شدند.

بعد از تغییرات : میبینیم که بعضی چیزا پاک شدن و در آخر 369 تا row باقی مونده

```
]: print(data_cleaned)

      id   age    bp     sg al su     rbc        pc         pcc          ba  \
0      1  48.0  80.0   1.02  1  0     NaN    normal  notpresent  notpresent
1      2   7.0  50.0   1.02  4  0     NaN    normal  notpresent  notpresent
2      3  62.0  80.0   1.01  2  3  normal    normal  notpresent  notpresent
3      4  48.0  70.0  1.005  4  0  normal  abnormal     present  notpresent
4      5  51.0  80.0   1.01  2  0  normal    normal  notpresent  notpresent
..   ...   ...   ...    ... .. ..     ...       ...         ...         ...
395  396  55.0  80.0   1.02  0  0  normal    normal  notpresent  notpresent
396  397  42.0  70.0  1.025  0  0  normal    normal  notpresent  notpresent
397  398  12.0  80.0   1.02  0  0  normal    normal  notpresent  notpresent
398  399  17.0  60.0  1.025  0  0  normal    normal  notpresent  notpresent
399  400  58.0  80.0  1.025  0  0  normal    normal  notpresent  notpresent

      ...   pcv    wbcc  rbcc  ht   dm  cad  appet   pe  ane   class
0     ...  44.0  7800.0   5.2  ye  yes   no   good   no   no     ckd
1     ...  38.0  6000.0   NaN  no   no   no   good   no   no     ckd
2     ...  31.0  7500.0   NaN  no  yes   no   poor   no  yes     ckd
3     ...  32.0  6700.0   3.9  yes  no   no   poor  yes  yes     ckd
4     ...  35.0  7300.0   4.   no   no   no   good   no   no     ckd
..    ...   ...     ...   ...  ...  ...  ...    ...  ...  ...     ...
395   ...  47.0  6700.    .9   no   no   no   good   no   no  notckd
396   ...  54.0  7800.0   6.2      no   no   good   no   no  notckd
397   ...  49.0  6600.0   .4   no   no   no   good   no   no  notckd
398   ...  51.0  7200.0   5.9  no   no   no   good   no   no  notckd
399   ...  53.0  6800.0   .    no   no   no   good   no   no  notckd

[369 rows x 26 columns]
```

برای اینکه بررسی کنیم که آیا این تغییرات به درستی انجام شده یا نه ، می‌تونیم دوباره تعداد مقادیر گمشده رو برای داده‌های باقی‌مانده چک کنیم و میتونیم ببینیم که نسبت به قبل چقدر تغییر کرده:



c, d . مدیریت و پاک‌سازی مقادیر گمشده

ما دو رویکرد اصلی داریم:

۱. **برای ویژگی‌های عددی :(numerical)** جایگزینی با میانگین یا میانه.
۲. **برای ویژگی‌های اسمی :(nominal)** جایگزینی با mode

```
[15]: for col in numeric_cols:
          data_cleaned.loc[:, col] = data_cleaned[col].astype(float)
          data_cleaned.loc[:, col] = data_cleaned[col].fillna(data_cleaned[col].mean())

[68]: nominal_cols = data.columns.difference(numeric_cols).tolist()
      for col in nominal_cols:
          data_cleaned.loc[:, col] = data_cleaned[col].fillna(data_cleaned[col].mode()[0])
```

در این مرحله داده‌های گمشده در ستون‌های عددی با مقدار میانگین هر ستون جایگزین شدند. همچنین داده‌های گمشده در ستون‌های اسمی (nominal) با مقدار پرتکرار(mode) آن ستون جایگزین شدند. این کار باعث شد تمام مقادیر گمشده حذف شوند و مجموعه داده تمیز و آماده پردازش‌های بعدی شود.

بررسی اینکه هنوزم مقدار گمشده داریم یا نه:

```
[70]: print("Total missing values left:", data_cleaned.isnull().sum().sum())

Total missing values left: 0

[72]: missing_data_after = data_cleaned.isnull().sum()
      print(missing_data_after)

id       0
age      0
bp       0
sg       0
al       0
su       0
rbc      0
pc       0
pcc      0
ba       0
bgr      0
bu       0
sc       0
sod      0
pot      0
hemo     0
pcv      0
wbcc     0
rbcc     0
htn      0
dm       0
cad      0
appet    0
pe       0
ane      0
class    0
dtype: int64
```

همه 0 شدند

با زدن دستور head و مقایسه اون با چیزی که تو مرحله اول داشتیم میتونیم ببینیم اون داده هایی که "؟" بودن پر شدن :

```
[76]: data_cleaned.head()
```

[76]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	1	48.0	80.0	1.02	1	0	normal	normal	notpresent	notpresent	...	44.0	7800.0	5.200000	yes	yes	no	good	no	no	ckd
1	2	7.0	50.0	1.02	4	0	normal	normal	notpresent	notpresent	...	38.0	6000.0	4.707435	no	no	no	good	no	no	ckd
2	3	62.0	80.0	1.01	2	3	normal	normal	notpresent	notpresent	...	31.0	7500.0	4.707435	no	yes	no	poor	no	yes	ckd
3	4	48.0	70.0	1.005	4	0	normal	abnormal	present	notpresent	...	32.0	6700.0	3.900000	yes	no	no	poor	yes	yes	ckd
4	5	51.0	80.0	1.01	2	0	normal	normal	notpresent	notpresent	...	35.0	7300.0	4.600000	no	no	no	good	no	no	ckd

5 rows × 26 columns

مرحله۳. شناسایی و حذف داده‌های پرت

در این مرحله، ما داده‌هایی که به‌طور غیرعادی از بقیه داده‌ها دور هستند (داده‌های پرت) را شناسایی و حذف می‌کنیم.

a. استفاده از روش IQR (Interquartile Range) یا Z-score برای تشخیص داده‌های پرت

```
[82]: outlier_data = data_cleaned.copy()

def find_outliers_iqr(df, col):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
    return outliers

for col in numeric_cols:
    outliers = find_outliers_iqr(outlier_data, col)
    print(f"{col}: {len(outliers)} outliers")
```

```
age: 8 outliers
bp: 35 outliers
bgr: 46 outliers
bu: 46 outliers
sc: 44 outliers
sod: 18 outliers
pot: 12 outliers
hemo: 1 outliers
pcv: 2 outliers
wbcc: 15 outliers
rbcc: 30 outliers
```

در این بخش، برای هر ویژگی عددی، از روش **Interquartile Range (IQR)** جهت شناسایی داده‌های پرت استفاده شد. روند کار به این شکل بوده:

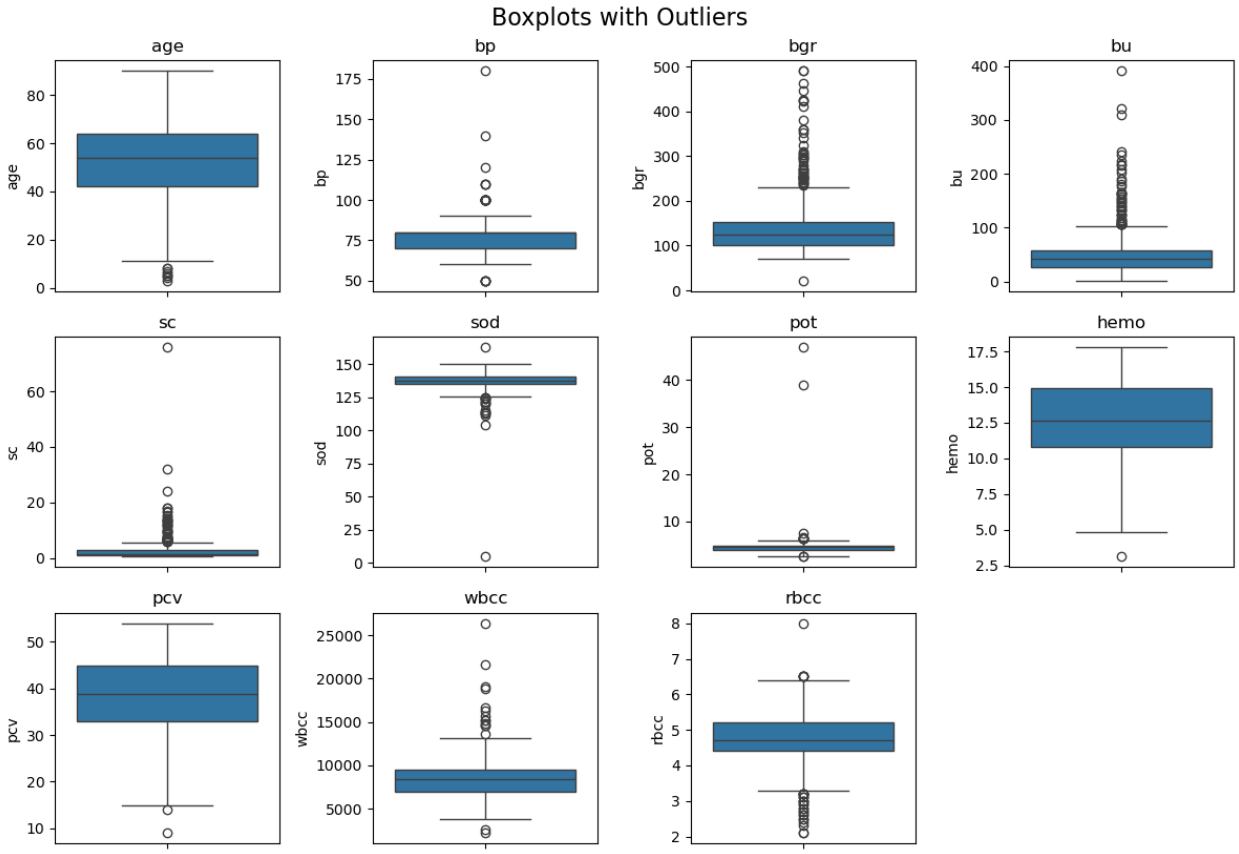
- محاسبه‌ی چارک اول (Q1) و چارک سوم (Q3) برای هر ستون.
- محاسبه‌ی IQR به صورت : IQR = Q3 - Q1
- تعیین محدوده‌ی قابل قبول داده‌ها:
  - **حد پایین** : Q1 - 1.5 × IQR
  - **حد بالا** : Q3 + 1.5 × IQR
- داده‌هایی خارج از این محدوده‌ها به عنوان **Outlier** شناسایی شدند.

تعداد داده‌های پرت برای هر ستون چاپ شد تا تصمیم‌گیری درباره‌ی حذف یا جایگزینی آن‌ها انجام شود.

b. نمایش داده های پرت با Boxplot

در این مرحله با رسم دوباره نمودارهای جعبه‌ای (Boxplot) برای همه ستون‌های عددی، داده‌های پرت (Outliers)رو به‌صورت بصری بررسی کردیم. با استفاده از seaborn و matplotlib، برای هر ویژگی عددی یه نمودار کشیدیم تا بهتر ببینیم کدوم ستون‌ها داده‌های پرت زیادی دارن. این دید کلی کمک می‌کنه تصمیم بگیریم که تو مراحل بعد با این داده‌ها چی کار کنیم.

```
[22]: import matplotlib.pyplot as plt
      import seaborn as sns

      plt.figure(figsize=(12, 8))

      for i, col in enumerate(numeric_cols):
          plt.subplot(3, 4, i + 1)
          sns.boxplot(y=outlier_data[col])
          plt.title(col)

      plt.tight_layout()
      plt.suptitle("Boxplots with Outliers", fontsize=16, y=1.02)
      plt.show()
```



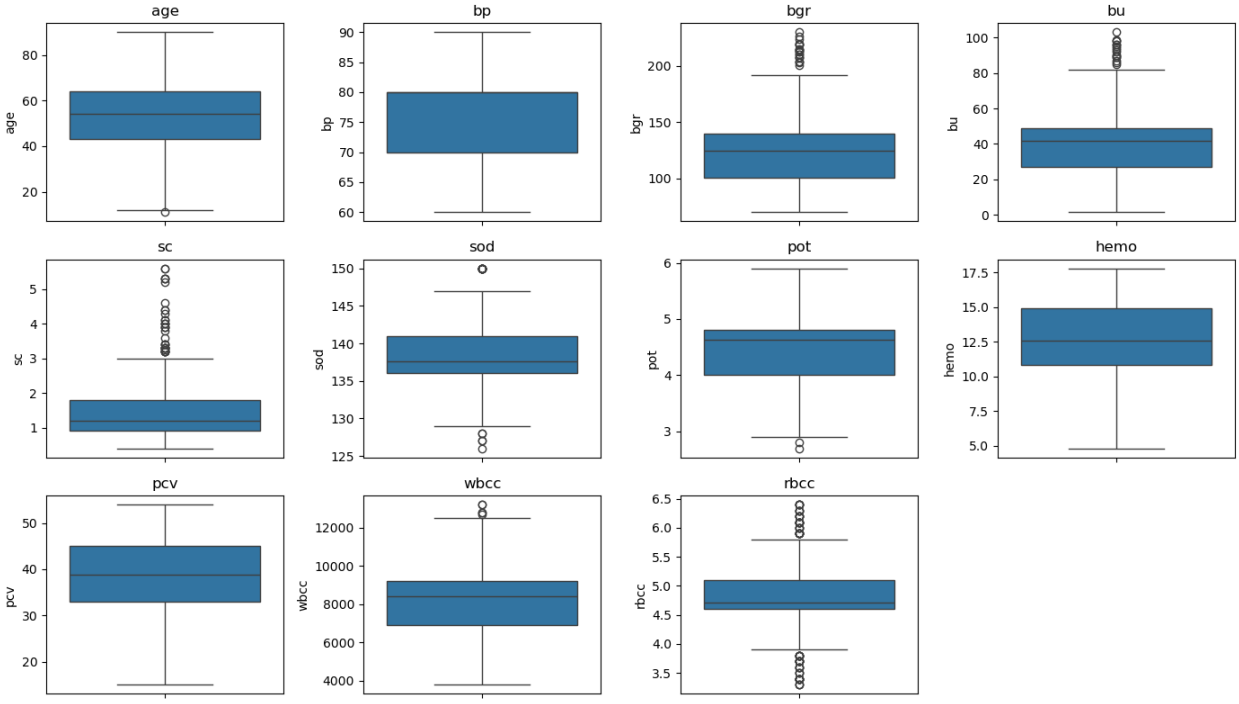
c. جایگزینی داده‌های پرت در صورت لزوم

در این مرحله، داده‌های پرت (Outliers) رو با مقدار میانه (median) همون ستون جایگزین کردیم. برای هر ویژگی عددی، ابتدا حدود بالا و پایین بر اساس IQR محاسبه شد، سپس مقادیری که خارج از این بازه بودن به‌عنوان داده پرت در نظر گرفته شدن و با مقدار میانه جایگزین شدن. این روش باعث حفظ ساختار کلی داده میشه بدون اینکه نمونه‌ها حذف بشن.

```
0]: data_no_outliers = data_cleaned.copy()

    for col in numeric_cols:
        Q1 = data_cleaned[col].quantile(0.25)
        Q3 = data_cleaned[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        median_value = data_cleaned[col].median()
        outlier_mask = (data_cleaned[col] < lower_bound) | (data_cleaned[col] > upper_bound)
        data_no_outliers.loc[outlier_mask, col] = median_value
```

در این مرحله برای اطمینان از حذف یا اصلاح داده‌های پرت، دوباره نمودارهای Boxplot برای هر ویژگی عددی رسم کردیم. همون‌طور که در نمودارها دیده میشه، مقادیر پرت دیگه وجود ندارن یا خیلی کمتر شدن، که نشون میده جایگزینی با مقدار میانه به خوبی انجام شده و داده‌ها حالا تمیزتر و مناسب‌تر برای تحلیل‌های بعدی هستن.

```
plt.figure(figsize=(14, 8))

for i, col in enumerate(numeric_cols):
    plt.subplot(3, 4, i + 1)
    sns.boxplot(y=data_no_outliers[col])
    plt.title(col)

plt.tight_layout()
plt.suptitle("Boxplots After Replacing Outliers", fontsize=16, y=1.02)
plt.show()
```


Boxplots After Replacing Outliers
age
bp
bgr
bu
sc
sod
pot
hemo
pcv
wbcc
rbcc

در این بخش، بعد از جایگزینی مقادیر پرت با میانه، دوباره بررسی کردیم که آیا هنوز هم داده پرت در مجموعه وجود دارد یا نه. با محاسبه مجدد IQR و حدود بالا و پایین، تعداد مقادیر پرت باقی‌مانده در هر ویژگی عددی محاسبه شد. نتایج نشون می‌ده که بیشتر مقادیر پرت قبلی برطرف شدن و تعداد کمی (اگر اصلاً باشه) باقی مونده، که یعنی داده‌هامون حالا آماده‌ی مرحله‌ی بعدی هستن.

[92]:
```
print("Remaining outliers after replacement:")
for col in numeric_cols:
    Q1 = data_no_outliers[col].quantile(0.25)
    Q3 = data_no_outliers[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = ((data_no_outliers[col] < lower_bound) | (data_no_outliers[col] > upper_bound)).sum()
    print(f"{col}: {outliers} outliers")
```

```
Remaining outliers after replacement:
age: 1 outliers
bp: 0 outliers
bgr: 22 outliers
bu: 17 outliers
sc: 34 outliers
sod: 22 outliers
pot: 2 outliers
hemo: 0 outliers
pcv: 0 outliers
wbcc: 5 outliers
rbcc: 68 outliers
```

چراOutlier ها صفر نشدند؟

بعد از اینکه مقدارهای پرت (Outlier) رو با میانه‌ی هر ستون جایگزین کردیم، شاید انتظار داشتیم که دیگه هیچ مقدار پرت باقی نمونه. ولی وقتی دوباره چک کردیم، دیدیم هنوز چند تا Outlier داریم.

دلیلش اینه که بعد از جایگزینی، دوباره محدوده‌ی IQR یعنی بازه‌ای که برای تشخیص Outlier استفاده می‌کنیم (حساب می‌شه. حالا ممکنه اون عددی که به‌جاش گذاشتیم (مثلاً میانه) خودش توی بازه‌ی جدید هم هنوز بیرون حساب بشه! یعنی یه جورایی جایگزینی باعث تغییر توی توزیع داده شده و بعضی مقدارها دوباره از نظر تعریف جدید، پرت به‌نظر میان.

پس اینکه چند تا Outlier بعد از جایگزینی باقی بمونه، چیز عجیبی نیست.

مرحله ٤ . تبدیل و استانداردسازی داده ها

a. استانداردسازی متغیرهای عددی با Min-Max Scaling یا Standard Scaling

```
[94]: from sklearn.preprocessing import StandardScaler

numeric_cols = ['age', 'bp', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbcc', 'rbcc']

numeric_data = data_cleaned[numeric_cols]

scaler = StandardScaler()
scaled_data = scaler.fit_transform(numeric_data)

data_scaled = pd.DataFrame(scaled_data, columns=numeric_cols)

print(data_scaled.describe())
```

```
                age            bp           bgr            bu            sc  \
count  3.690000e+02  3.690000e+02  3.690000e+02  3.690000e+02  3.690000e+02
mean  -2.310708e-16 -1.540472e-16  2.503267e-16 -5.776770e-17  1.925590e-17
std    1.001358e+00  1.001358e+00  1.001358e+00  1.001358e+00  1.001358e+00
min   -2.901721e+00 -1.932264e+00 -1.693865e+00 -1.105317e+00 -4.717517e-01
25%   -5.649178e-01 -4.718793e-01 -6.249712e-01 -5.905277e-01 -3.773961e-01
50%    1.540986e-01  2.583132e-01 -3.002439e-01 -2.877103e-01 -3.207828e-01
75%    7.532789e-01  2.583132e-01  7.860463e-02  3.529486e-02 -1.884491e-02
max    2.311148e+00  7.560239e+00  4.638317e+00  6.757840e+00  1.379481e+01

                sod           pot          hemo           pcv          wbcc  \
count  3.690000e+02  3.690000e+02  3.690000e+02  3.690000e+02  3.690000e+02
mean   9.050273e-16  5.776770e-17 -1.540472e-16  1.925590e-16  3.851180e-17
std    1.001358e+00  1.001358e+00  1.001358e+00  1.001358e+00  1.001358e+00
min   -1.397801e+01 -7.299426e-01 -3.436649e+00 -3.525813e+00 -2.367441e+00
25%   -2.751968e-01 -2.520094e-01 -6.531543e-01 -6.933245e-01 -5.746495e-01
50%    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
75%    3.548178e-01  5.523338e-02  8.289664e-01  7.229199e-01  4.171075e-01
max    2.664871e+00  1.446151e+01  1.877296e+00  1.785103e+00  6.863528e+00

               rbcc
count  3.690000e+02
mean   3.080944e-16
```

استانداردسازی ویژگی‌های عددی(Standard Scaling)

برای اینکه مدل‌های یادگیری ماشین بتونن بهتر یاد بگیرن، خوبه که همه‌ی ویژگی‌های عددی در یک مقیاس مشابه قرار بگیرن. چون بعضی الگوریتم‌ها (مثل KNN یاSVM) به اختلاف مقیاس حساسن.

در این مرحله از **StandardScaler** استفاده کردیم که داده‌ها رو طوری تغییر می‌ده که میانگین هر ستون صفر و انحراف معیارش یک بشه. اینطوری همه‌ی ویژگی‌ها تقریباً در یک بازه نرمال قرار می‌گیرن و تاثیر هیچکدوم به‌خاطر مقیاس بزرگ‌تر یا کوچکتر بیشتر یا کمتر نمی‌شه.

در پایان هم با describe()خلاصه‌ای آماری از داده‌های استاندارد شده گرفتیم تا مطمئن بشیم همه‌چی درست انجام شده.

مرحله ۵. تبدیل ویژگی های دسته ای به عددی

a. استفاده از Label Encoding برای تمام ویژگی هایnominal

```
]: from sklearn.preprocessing import LabelEncoder

   encoded_data = data_cleaned.copy()

   le = LabelEncoder()

   for col in nominal_cols:
       encoded_data[col] = le.fit_transform(encoded_data[col])

   print(encoded_data[nominal_cols].head())
```

	al	ane	appet	ba	cad	class	dm	htn	id	pc	pcc	pe	rbc	sg	su
0	1	0	0	0	0	0	1	1	0	1	0	0	1	3	0
1	4	0	0	0	0	0	0	0	1	1	0	0	1	3	0
2	2	1	1	0	0	0	1	0	2	1	0	0	1	1	3
3	4	1	1	0	0	0	0	1	3	0	1	1	1	0	0
4	2	0	0	0	0	0	0	0	4	1	0	0	1	1	0

ویژگی‌های اسمی) مثل ‘rbc’, ‘dm’, ‘htn’ و (...نمی‌تونن به‌صورت مستقیم وارد مدل‌های عددی بشن، چون این مدل‌ها فقط با اعداد سر و کار دارن.

برای حل این موضوع، از **LabelEncoder** استفاده کردیم. این ابزار مقدارهای متنی رو به عدد تبدیل می‌کنه. مثلاً "yes" میشه ۱ و "no" میشه ۰. این تبدیل کمک می‌کنه که داده‌ها قابل استفاده برای مدل‌سازی بشن، بدون اینکه معنی‌شون رو از دست بدن.

در نهایت با head()چند سطر اول از ستون‌های تبدیل‌شده رو بررسی کردیم تا مطمئن بشیم همه‌چیز درست انجام شده.

b. استفاده از One-Hot Encoding برای ویژگی های چندحالتی مانند rbc, pc, appet

```
[106]: multiclass_cols = ['rbc', 'pc', 'appet']

       data_encoded = pd.get_dummies(data_cleaned, columns=multiclass_cols, drop_first=True)

       print("ستون‌های جدید بعد از One-Hot Encoding:")
       print([col for col in data_encoded.columns if any(prefix in col for prefix in multiclass_cols)])

       print("\nشکل نهایی داده‌ها:")
       print(data_encoded.shape)
       data_encoded.head()
```

```
ستون‌های جدید بعد از One-Hot Encoding:
['pcc', 'pcv', 'rbcc', 'rbc_normal', 'pc_normal', 'appet_poor']

شکل نهایی داده‌ها:
(369, 26)
```

[106]:

	id	age	bp	sg	al	su	pcc	ba	bgr	bu	...	rbcc	htn	dm	cad	pe	ane	class	rbc_normal	pc_normal	appet_poor
0	1	48.0	80.0	1.02	1	0	notpresent	notpresent	121.000000	36.0	...	5.200000	yes	yes	no	no	no	ckd	True	True	False
1	2	7.0	50.0	1.02	4	0	notpresent	notpresent	147.190476	18.0	...	4.707435	no	no	no	no	no	ckd	True	True	False
2	3	62.0	80.0	1.01	2	3	notpresent	notpresent	423.000000	53.0	...	4.707435	no	yes	no	no	yes	ckd	True	True	True
3	4	48.0	70.0	1.005	4	0	present	notpresent	117.000000	56.0	...	3.900000	yes	no	no	yes	yes	ckd	True	False	True
4	5	51.0	80.0	1.01	2	0	notpresent	notpresent	106.000000	26.0	...	4.600000	no	no	no	no	no	ckd	True	True	False

5 rows × 26 columns

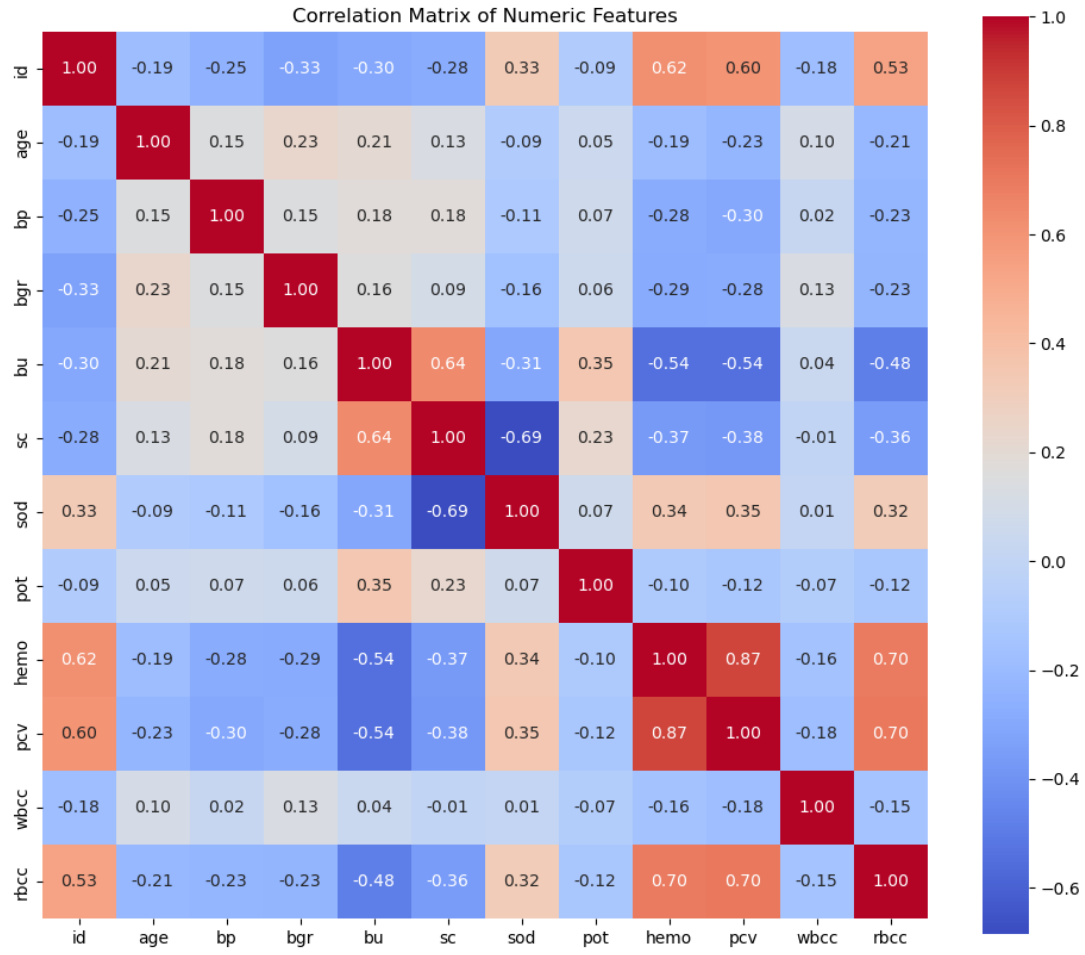
بعضی از ویژگی‌ها مثل rbc، pc و appetبیش از دو حالت دارند) مثلاً"normal" ، "abnormal"، ."poor")
برای اینکه این ستون‌ها رو قابل استفاده در مدل‌سازی کنیم، باید به ویژگی‌های عددی تبدیل بشن.
در این مرحله از روش **One-Hot Encoding**استفاده کردیم، که برای هر مقدار ممکن یک ستون جدید درست می‌کنه. مثلاً ستون rbcبه rbc_normalو rbc_abnormalتبدیل میشه. چون drop_first=Trueگذاشتیم، یکی از حالت‌ها حذف میشه تا از هم‌خطی جلوگیری کنیم.
در نهایت، شکل نهایی داده‌ها و نام ستون‌های جدید رو چاپ کردیم تا بررسی کنیم همه چیز درست انجام شده.

مرحله ٦. بررسی همبستگی و کاهش ابعاد

a. بررسی همبستگی ویژگی ها با هم و حذف متغیرهای وابسته

در این مرحله، با استفاده از heatmap همبستگی بین ویژگی‌های عددی بررسی شد. این نمودار نشون می‌ده که کدوم ویژگی‌ها رابطه قوی (مثبت یا منفی) با هم دارن.
همبستگی بالا (نزدیک به ۱ یا -۱) بین دو ستون ممکنه باعث افزونگی (Redundancy) در مدل‌سازی بشه. در چنین مواردی ممکنه یکی از اون ستون‌ها رو حذف کنیم یا از روش‌هایی مثل **PCA** برای کاهش ابعاد استفاده کنیم.

```
14]: import seaborn as sns
     import matplotlib.pyplot as plt

     numeric_features = data_encoded.select_dtypes(include=[np.number])

     corr_matrix = numeric_features.corr()

     plt.figure(figsize=(12, 10))
     sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)
     plt.title("Correlation Matrix of Numeric Features")
     plt.show()
```



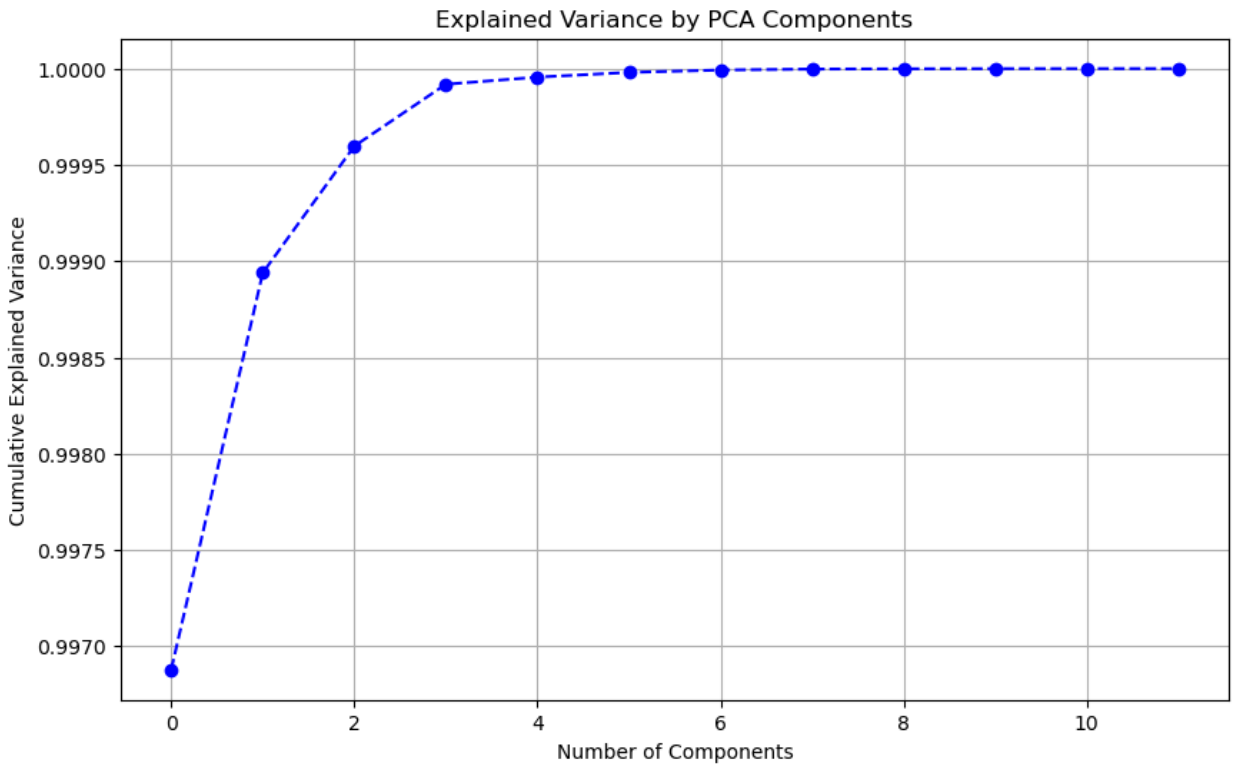
b. استفاده از PCA در صورت نیاز به کاهش ابعاد

در این مرحله برای کاهش ابعاد داده‌ها از **تحلیل مؤلفه‌های اصلی (PCA)** استفاده کردیم. هدف اینه که با نگه‌داشتن تعداد کمتری از مؤلفه‌ها(Components) ، بیشترِ اطلاعات موجود در داده‌ها حفظ بشه.

با استفاده از نمودار **Cumulative Explained Variance** می‌تونیم ببینیم که چند مؤلفه اول چه درصدی از واریانس داده‌ها رو پوشش میدن. مثلاً اگه ۴ مؤلفه اول بتونن بالای ٪ ۹۰ واریانس رو توضیح بدن، میشه فقط اونا رو نگه داشت و باقی رو حذف کرد.

این کار باعث ساده‌تر شدن مدل، کاهش پیچیدگی و در مواردی بهبود عملکرد مدل می‌شه.

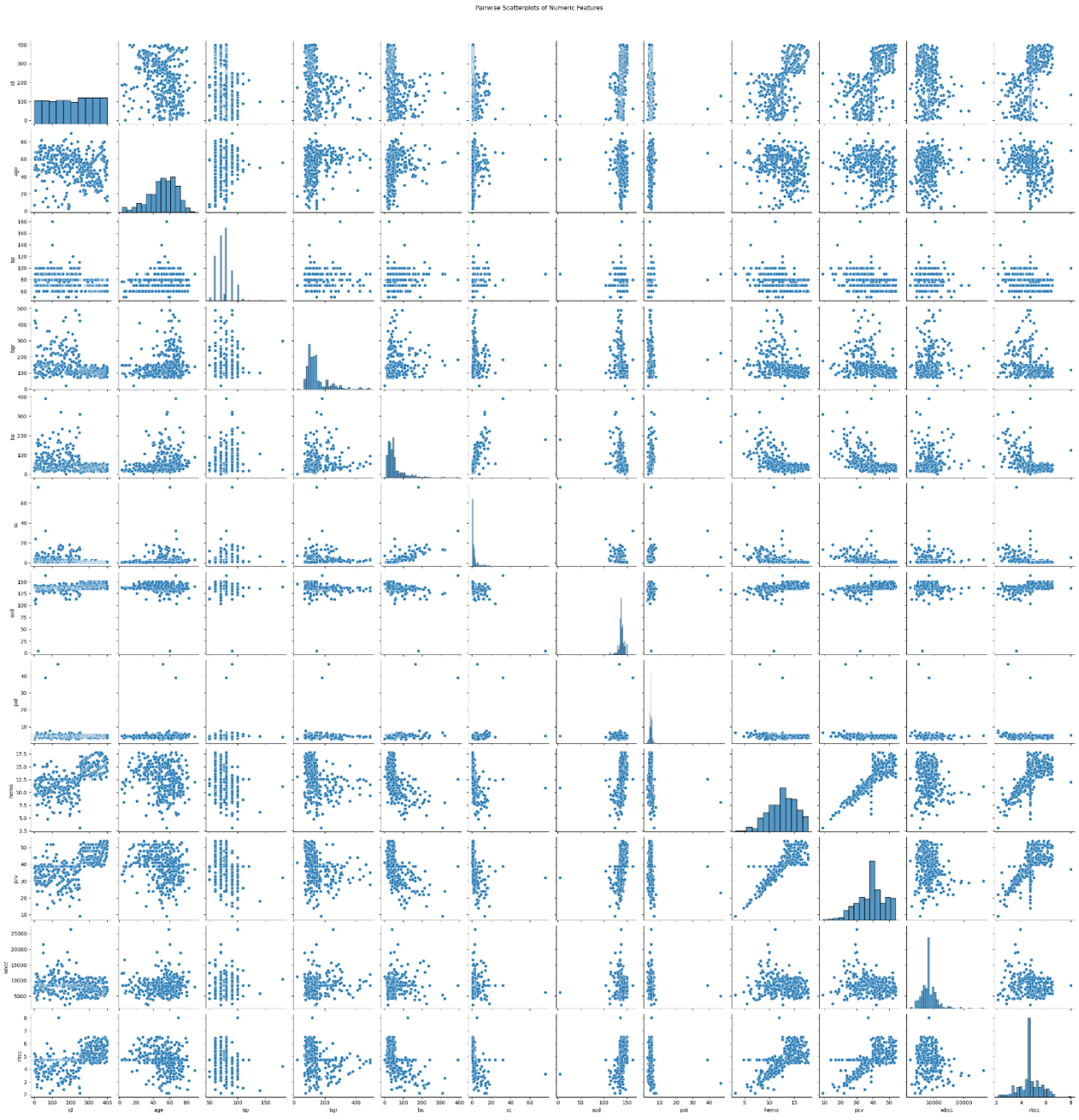
```
16]: from sklearn.decomposition import PCA

pca = PCA()
pca.fit(numeric_features)

explained_variance = pca.explained_variance_ratio_

plt.figure(figsize=(10, 6))
plt.plot(np.cumsum(explained_variance), marker='o', linestyle='--', color='b')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance by PCA Components')
plt.grid(True)
plt.show()
```



برای اینکه ببینیم بین ویژگی‌های عددی چه ارتباط‌هایی وجود داره، از Pairplot استفاده کردیم. این نمودار ترکیبی از scatterplot بین جفت ویژگی‌ها و histogram برای توزیع هر ویژگیه. این کار کمک می‌کنه الگوها، همبستگی‌ها و داده‌های پرت رو راحت‌تر شناسایی کنیم.

```
import seaborn as sns
sns.pairplot(numeric_features)
plt.suptitle("Pairwise Scatterplots of Numeric Features", y=1.02)
plt.show()
```

Pairwise Scatterplots of Nu



مرحله کاهش ابعاد با استفاده ازPCA

۱. **بررسی تعداد ردیف‌ها :**تعداد ردیف‌ها در دو DataFrame (data_encoded و numeric_features) ابتدا بررسی شد تا مطمئن شویم که تعداد ردیف‌ها برابر است.
۲. **بازنشانی ایندکس‌ها :**ایندکس‌ها برای هر دو DataFrame با استفاده از reset_index() بازنشانی شدند تا ترتیب ردیف‌ها به درستی حفظ شود.
۳. **اجرای PCA:** اگر تعداد ردیف‌ها برابر بود، PCA با ۴ مؤلفه اصلی اجرا شد.
٤. **ترکیب داده‌ها :**پس از اجرایPCA ، داده‌های اصلی با داده‌های کدگذاری‌شده ترکیب شدند.
٥. **خطا در صورت عدم تطابق :**در صورتی که تعداد ردیف‌ها برابر نبود، پیامی مبنی بر خطا نمایش داده شد.

```
]: print("Number of rows in data_encoded before reset:", data_encoded.shape[0])
   print("Number of rows in numeric_features before reset:", numeric_features.shape[0])

   data_encoded.reset_index(drop=True, inplace=True)
   numeric_features.reset_index(drop=True, inplace=True)

   print("Number of rows in data_encoded after reset:", data_encoded.shape[0])
   print("Number of rows in numeric_features after reset:", numeric_features.shape[0])

   if data_encoded.shape[0] == numeric_features.shape[0]:
       pca_final = PCA(n_components=4)
       principal_components = pca_final.fit_transform(numeric_features)

       pca_df = pd.DataFrame(principal_components, columns=['PC1', 'PC2', 'PC3', 'PC4'])

       final_df = pd.concat([pca_df, data_encoded.drop(columns=numeric_features.columns)], axis=1)

       print("Final shape after PCA:", final_df.shape)
       final_df.head()
   else:
       print("Error: Number of rows in data_encoded and numeric_features do not match!")

   Number of rows in data_encoded before reset: 369
   Number of rows in numeric_features before reset: 369
   Number of rows in data_encoded after reset: 369
   Number of rows in numeric_features after reset: 369
   Final shape after PCA: (369, 18)
```

ابعاد نهایی :

```
final_df.shape

(369, 18)
```

مرحله ۷. ذخیره داده های پردازش شده

در پایان فرآیند پاک‌سازی و پیش‌پردازش داده‌ها، مجموعه داده نهایی که شامل ویژگی‌های کاهش‌یافته با PCA و داده‌های بدون مقدار گمشده و پرت است، با استفاده از دستور to_csv() در قالب فایل CSV ذخیره شد:

```
151]: final_df.to_csv("cleaned_ckd_data.csv", index=False, encoding='utf-8-sig')
```

این فایل می‌تواند به‌عنوان ورودی در مدل‌سازی و تحلیل‌های فاز بعدی پروژه مورد استفاده قرار گیرد.