Chronic Kidney Disease (CKD) Clustering

1 Project

- **Motivation**: In my work with binary classification tasks, I've often wondered how effective clustering methods can be when labels are unavailable. Exploring clustering performance under such conditions not only challenges traditional supervised learning approaches but also reflects real-world scenarios where labeled data is scarce.
- **Problem**: Many supervised learning models predict diseases and perform binary classification (e.g., disease/no disease). However, diseases like Chronic Kidney Disease (CKD) are influenced by various etiologies and progress through distinct stages. Beyond the common binary categorization (CKD/no CKD), can unsupervised learning identify meaningful subgroups within CKD?
- **Approaches**:
  - **Unsupervised Learning**: Apply dimensionality reduction techniques (e.g., t-SNE, PCA) for visualization, followed by K-Means clustering method to identify potential subtypes or patterns in unlabeled data.
  - **Supervised Learning**: Train and evaluate models such as Random Forest, Gradient Boosting, and Neural Networks using labeled data to compare their classification accuracy with the clustering results.

---

2 Data

- **Acquisition**: The dataset, "Chronic Kidney Disease," is sourced from the UCI Machine Learning Repository. It was donated by Rubini, Soundarapandian, and Eswaran on 7/2/2015.
- **Description**:
  - **Instances**: 400
  - **Features**: 24 (14 numerical, 10 categorical)
    1. `age` : age
    2. `bp` : blood pressure
    3. `sg` : specific gravity
    4. `al` : albumin
    5. `su` : sugar
    6. `rbc` : red blood cells
    7. `pc` : pus cell
    8. `pcc` : pus cell clumps
    9. `ba` : bacteria
    10. `bgr` : blood glucose random
    11. `bu` : blood urea
    12. `sc` : serum creatinine
    13. `sod` : sodium
    14. `pot` : potassium
    15. `hemo` : hemoglobin
    16. `pcv` : packed cell volume
    17. `wc` : white blood cell count
    18. `rc` : red blood cell count
    19. `htn` : hypertension
    20. `dm` : diabetes mellitus
    21. `cad` : coronary artery disease
    22. `appet` : appetite
    23. `pe` : pedal edema
    24. `ane` : anemia
    25. `classification` : classification

Reference

Rubini, L., Soundarapandian, P., & Eswaran, P. (2015). *Chronic Kidney Disease* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5G020

Import Libraries

In [47]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
import warnings
warnings.filterwarnings("ignore")
```

Load Data

In [48]:
```
# Download the dataset and read the data
df = pd.read_csv('C:/Users/User/Downloads/kidney_disease.csv')
df.head()
```

Out[48]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd

5 rows × 26 columns

In [49]:
```
df.shape
```

Out[49]:
```
(400, 26)
```

In [50]:
```
df.drop('id', axis=1,inplace=True)
```

In [51]:
```
df.shape
```

Out[51]:
```
(400, 25)
```

---

3 EDA

Step 1: Data Cleaning

1. **Typos**:

- **Check Unique Values**: Detect typos in each column.
- **Clean up Typos**: Replace typos in `pcv`, `wc`, `rc`, `dm`, `cad`, and `classification` column.

In [52]:
```
# Check unique values
for i in df.columns:
    print('\nUnique values in "{}":\n'.format(i),df[i].unique())
```

```
Unique values in "age":
 [48.  7. 62. 51. 60. 68. 24. 52. 53. 50. 63. 40. 47. 61. 21. 42. 75. 69.
 nan 73. 70. 65. 76. 72. 82. 46. 45. 35. 54. 11. 59. 67. 15. 55. 44. 26.
 64. 56.  5. 74. 38. 58. 71. 34. 17. 12. 43. 41. 57.  8. 39. 66. 81. 14.
 27. 83. 30.  4.  3.  6. 32. 80. 49. 90. 78. 19.  2. 33. 36. 37. 23. 25.
 20. 29. 28. 22. 79.]

Unique values in "bp":
 [ 80.  50.  70.  90.  nan 100.  60. 110. 140. 180. 120.]

Unique values in "sg":
 [1.02  1.01  1.005 1.015   nan 1.025]

Unique values in "al":
 [ 1.  4.  2.  3.  0. nan  5.]

Unique values in "su":
 [ 0.  3.  4.  1. nan  2.  5.]

Unique values in "rbc":
 [nan 'normal' 'abnormal']

Unique values in "pc":
 ['normal' 'abnormal' nan]

Unique values in "pcc":
 ['notpresent' 'present' nan]

Unique values in "ba":
 ['notpresent' 'present' nan]

Unique values in "bgr":
 [121.  nan 423. 117. 106.  74. 100. 410. 138.  70. 490. 380. 208.  98.
 157.  76.  99. 114. 263. 173.  95. 108. 156. 264. 123.  93. 107. 159.
 140. 171. 270.  92. 137. 204.  79. 207. 124. 144.  91. 162. 246. 253.
 141. 182.  86. 150. 146. 425. 112. 250. 360. 163. 129. 133. 102. 158.
 165. 132. 104. 127. 415. 169. 251. 109. 280. 210. 219. 295.  94. 172.
 101. 298. 153.  88. 226. 143. 115.  89. 297. 233. 294. 323. 125.  90.
 308. 118. 224. 128. 122. 214. 213. 268. 256.  84. 105. 288. 139.  78.
 273. 242. 424. 303. 148. 160. 192. 307. 220. 447. 309.  22. 111. 261.
 215. 234. 131. 352.  80. 239. 110. 130. 184. 252. 113. 230. 341. 255.
 103. 238. 248. 120. 241. 269. 201. 203. 463. 176.  82. 119.  97.  96.
  81. 116. 134.  85.  83.  87.  75.]

Unique values in "bu":
 [ 36.   18.   53.   56.   26.   25.   54.   31.   60.  107.   55.   72.
  86.   90.  162.   46.   87.   27.  148.  180.  163.    nan  50.   75.
  45.   28.  155.   33.   39.  153.   29.   65.  103.   70.   80.   20.
 202.   77.   89.   24.   17.   32.  114.   66.   38.  164.  142.   96.
 391.   15.  111.   73.   19.   92.   35.   16.  139.   48.   85.   98.
 186.   37.   47.   52.   82.   51.  106.   22.  217.   88.  118.   50.1
  71.   34.   40.   21.  219.   30.  125.  166.   49.  208.  176.   68.
 145.  165.  322.   23.  235.  132.   76.   42.   44.   41.  113.    1.5
 146.   58.  133.  137.   67.  115.  223.   98.6 158.   94.   74.  150.
  61.   57.   95.  191.   93.  241.   64.   79.  215.  309.   10. ]

Unique values in "sc":
 [ 1.2   0.8   1.8   3.8   1.4   1.1  24.    1.9   7.2   4.    2.7   2.1
  4.6   4.1   9.6   2.2   5.2   1.3   1.6   3.9  76.    7.7    nan  2.4
  7.3   1.5   2.5   2.    3.4   0.7   1.   10.8   6.3   5.9   0.9   3.
  3.25  9.7   6.4   3.2  32.    0.6   6.1   3.3   6.7   8.5   2.8  15.
  2.9   1.7   3.6   5.6   6.5   4.4  10.2  11.5   0.5  12.2   5.3   9.2
 13.8  16.9   6.    7.1  18.    2.3  13.   48.1  14.2  16.4   2.6   7.5
  4.3  18.1  11.8   9.3   6.8  13.5  12.8  11.9  12.   13.4  15.2  13.3
  0.4 ]

Unique values in "sod":
 [  nan 111.  142.  104.  114.  131.  138.  135.  130.  141.  139.    4.5
 136.  129.  140.  132.  133.  134.  125.  163.  137.  128.  143.  127.
 146.  126.  122.  147.  124.  115.  145.  113.  120.  150.  144. ]

Unique values in "pot":
 [ nan  2.5  3.2  4.   3.7  4.2  5.8  3.4  6.4  4.9  4.1  4.3  5.2  3.8
  4.6  3.9  4.7  5.9  4.8  4.4  6.6 39.   5.5  5.   3.5  3.6  7.6  2.9
  4.5  5.7  5.4  5.3 47.   6.3  5.1  5.6  3.   2.8  2.7  6.5  3.3]

Unique values in "hemo":
 [15.4 11.3  9.6 11.2 11.6 12.2 12.4 10.8  9.5  9.4  9.7  9.8  5.6  7.6
 12.6 12.1 12.7 10.3  7.7 10.9  nan 11.1  9.9 12.5 12.9 10.1 12.  13.
  7.9  9.3 15.  10.   8.6 13.6 10.2 10.5  6.6 11.   7.5 15.6 15.2  4.8
  9.1  8.1 11.9 13.5  8.3  7.1 16.1 10.4  9.2  6.2 13.9 14.1  6.  11.8
 11.7 11.4 14.   8.2 13.2  6.1  8.  12.3  8.4 14.3  9.   8.7 10.6 13.1
 10.7  5.5  5.8  6.8  8.8  8.5 13.8 11.5  7.3 13.7 12.8 13.4  6.3  3.1
 17.  15.9 14.5 15.5 16.2 14.4 14.2 16.3 14.8 16.5 15.7 13.3 14.6 16.4
 16.9 16.  14.7 16.6 14.9 16.7 16.8 15.8 15.1 17.1 17.2 15.3 17.3 17.4
 17.7 17.8 17.5 17.6]

Unique values in "pcv":
 ['44' '38' '31' '32' '35' '39' '36' '33' '29' '28' nan '16' '24' '37' '30'
 '34' '40' '45' '27' '48' '\t?' '52' '14' '22' '18' '42' '17' '46' '23'
 '19' '25' '41' '26' '15' '21' '43' '20' '\t43' '47' '9' '49' '50' '53'
 '51' '54']

Unique values in "wc":
 ['7800' '6000' '7500' '6700' '7300' nan '6900' '9600' '12100' '4500'
 '12200' '11000' '3800' '11400' '5300' '9200' '6200' '8300' '8400' '10300'
 '9800' '9100' '7900' '6400' '8600' '18900' '21600' '4300' '8500' '11300'
 '7200' '7700' '14600' '6300' '\t6200' '7100' '11800' '9400' '5500' '5800'
 '13200' '12500' '5600' '7000' '11900' '10400' '10700' '12700' '6800'
 '6500' '13600' '10200' '9000' '14900' '8200' '15200' '5000' '16300'
 '12400' '\t8400' '10500' '4200' '4700' '10900' '8100' '9500' '2200'
 '12800' '11200' '19100' '\t?' '12300' '16700' '2600' '26400' '8800'
 '7400' '4900' '8000' '12000' '15700' '4100' '5700' '11500' '5400' '10800'
 '9900' '5200' '5900' '9300' '9700' '5100' '6600']

Unique values in "rc":
 ['5.2' nan '3.9' '4.6' '4.4' '5' '4.0' '3.7' '3.8' '3.4' '2.6' '2.8' '4.3'
 '3.2' '3.6' '4' '4.1' '4.9' '2.5' '4.2' '4.5' '3.1' '4.7' '3.5' '6.0'
 '5.0' '2.1' '5.6' '2.3' '2.9' '2.7' '8.0' '3.3' '3.0' '3' '2.4' '4.8'
 '\t?' '5.4' '6.1' '6.2' '6.3' '5.1' '5.8' '5.5' '5.3' '6.4' '5.7' '5.9'
```

```
 '6.5']

Unique values in "htn":
 ['yes' 'no' nan]

Unique values in "dm":
 ['yes' 'no' ' yes' '\tno' '\tyes' nan]

Unique values in "cad":
 ['no' 'yes' '\tno' nan]

Unique values in "appet":
 ['good' 'poor' nan]

Unique values in "pe":
 ['no' 'yes' nan]

Unique values in "ane":
 ['no' 'yes' nan]

Unique values in "classification":
 ['ckd' 'ckd\t' 'notckd']
```

In [53]:
```python
# Clean up typos
df['pcv'] = df['pcv'].replace(['\t?', '\t43'], [np.nan, '43'])
df['wc'] = df['wc'].replace(['\t6200', '\t8400', '\t?'], ['6200', '8400', np.nan])
df['rc'] = df['rc'].replace('\t?', np.nan)
df['dm'] = df['dm'].replace({'\tno': 'no', '\tyes': 'yes', ' yes': 'yes'})
df['cad'] = df['cad'].replace('\tno', 'no')
df['classification'] = df['classification'].replace('ckd\t', 'ckd')

# Replace 'ckd' and 'notckd' with 'yes' and 'no' in 'classification' column
df['classification'] = df['classification'].replace({'ckd': 'yes', 'notckd': 'no'})

# Verify cleaning
for column in ['pcv', 'wc', 'rc', 'dm', 'cad', 'classification']:
    print(f"\nUnique values in '{column}':\n", df[column].unique())
```

```
Unique values in 'pcv':
 ['44' '38' '31' '32' '35' '39' '36' '33' '29' '28' nan '16' '24' '37' '30'
 '34' '40' '45' '27' '48' '52' '14' '22' '18' '42' '17' '46' '23' '19'
 '25' '41' '26' '15' '21' '43' '20' '47' '9' '49' '50' '53' '51' '54']

Unique values in 'wc':
 ['7800' '6000' '7500' '6700' '7300' nan '6900' '9600' '12100' '4500'
 '12200' '11000' '3800' '11400' '5300' '9200' '6200' '8300' '8400' '10300'
 '9800' '9100' '7900' '6400' '8600' '18900' '21600' '4300' '8500' '11300'
 '7200' '7700' '14600' '6300' '7100' '11800' '9400' '5500' '5800' '13200'
 '12500' '5600' '7000' '11900' '10400' '10700' '12700' '6800' '6500'
 '13600' '10200' '9000' '14900' '8200' '15200' '5000' '16300' '12400'
 '10500' '4200' '4700' '10900' '8100' '9500' '2200' '12800' '11200'
 '19100' '12300' '16700' '2600' '26400' '8800' '7400' '4900' '8000'
 '12000' '15700' '4100' '5700' '11500' '5400' '10800' '9900' '5200' '5900'
 '9300' '9700' '5100' '6600']

Unique values in 'rc':
 ['5.2' nan '3.9' '4.6' '4.4' '5' '4.0' '3.7' '3.8' '3.4' '2.6' '2.8' '4.3'
 '3.2' '3.6' '4' '4.1' '4.9' '2.5' '4.2' '4.5' '3.1' '4.7' '3.5' '6.0'
 '5.0' '2.1' '5.6' '2.3' '2.9' '2.7' '8.0' '3.3' '3.0' '3' '2.4' '4.8'
 '5.4' '6.1' '6.2' '6.3' '5.1' '5.8' '5.5' '5.3' '6.4' '5.7' '5.9' '6.5']

Unique values in 'dm':
 ['yes' 'no' nan]

Unique values in 'cad':
 ['no' 'yes' nan]

Unique values in 'classification':
 ['yes' 'no']
```

---

2. Mistyped Features:

- **Check Data Types**: Inspect data types of each column.
- **Data Type Correction**: Convert the identified columns (`pcv` , `wc` , `rc`) from `object` to `float64` .

In [54]:
```python
# Check data types
data_overview = pd.DataFrame({
    'Column': df.columns,
    'DataType': df.dtypes.values,
    'UniqueValues': [
        ', '.join(map(str, df[col].unique()[:7])) + ('...' if df[col].nunique() > 10 else '')
        for col in df.columns
    ],
    'UniqueValuesCount': df.nunique().values
})

def highlight_columns(s):
    if s['Column'] in ['pcv', 'wc', 'rc']:
        return ['background-color: peachpuff'] * len(s)
    else:
        return [''] * len(s)

styled_data_overview = data_overview.style.apply(highlight_columns, axis=1)

styled_data_overview
```

Out[54]:

	Column	DataType	UniqueValues	UniqueValuesCount
0	age	float64	48.0, 7.0, 62.0, 51.0, 60.0, 68.0, 24.0...	76
1	bp	float64	80.0, 50.0, 70.0, 90.0, nan, 100.0, 60.0	10
2	sg	float64	1.02, 1.01, 1.005, 1.015, nan, 1.025	5
3	al	float64	1.0, 4.0, 2.0, 3.0, 0.0, nan, 5.0	6
4	su	float64	0.0, 3.0, 4.0, 1.0, nan, 2.0, 5.0	6
5	rbc	object	nan, normal, abnormal	2
6	pc	object	normal, abnormal, nan	2
7	pcc	object	notpresent, present, nan	2
8	ba	object	notpresent, present, nan	2
9	bgr	float64	121.0, nan, 423.0, 117.0, 106.0, 74.0, 100.0...	146
10	bu	float64	36.0, 18.0, 53.0, 56.0, 26.0, 25.0, 54.0...	118
11	sc	float64	1.2, 0.8, 1.8, 3.8, 1.4, 1.1, 24.0...	84
12	sod	float64	nan, 111.0, 142.0, 104.0, 114.0, 131.0, 138.0...	34
13	pot	float64	nan, 2.5, 3.2, 4.0, 3.7, 4.2, 5.8...	40
14	hemo	float64	15.4, 11.3, 9.6, 11.2, 11.6, 12.2, 12.4...	115
15	pcv	object	44, 38, 31, 32, 35, 39, 36...	42
16	wc	object	7800, 6000, 7500, 6700, 7300, nan, 6900...	89
17	rc	object	5.2, nan, 3.9, 4.6, 4.4, 5, 4.0...	48
18	htn	object	yes, no, nan	2
19	dm	object	yes, no, nan	2
20	cad	object	no, yes, nan	2
21	appet	object	good, poor, nan	2
22	pe	object	no, yes, nan	2
23	ane	object	no, yes, nan	2
24	classification	object	yes, no	2

In [55]:
```
# Convert 'object' to 'float64'
df[['pcv', 'wc', 'rc']] = df[['pcv', 'wc', 'rc']].astype('float64')

# Verify correcting
for col, dtype in df[['pcv', 'wc', 'rc']].dtypes.items():
    print(f"{col}: {dtype}")
```

```
pcv: float64
wc: float64
rc: float64
```

---

3. Missing Values:

- **Check Missing Values**: Identify the number of missing values in each column.
- **One-Hot Encoding**: Convert categorical features into binary columns with `drop_first=True`.
- **KNN Imputation**: Address missing values using the `KNNImputer`.

In [56]:
```
pd.DataFrame(df.isnull().sum(), columns=["Missing Values"]).style.bar(color = "salmon")
```

Out[56]:

	Missing Values
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
wc	106
rc	131
htn	2
dm	2
cad	2
appet	1
pe	1
ane	1
classification	0

In [57]:
```
# One-Hot Encoding
df_onehot=pd.get_dummies(df,drop_first=True,prefix_sep=': ')
df_onehot.head()
```

Out[57]:

	age	bp	sg	al	su	bgr	bu	sc	sod	pot	...	pc: normal	pcc: present	ba: present	htn: yes	dm: yes	cad: yes	appet: poor	pe: yes	ane: yes	classification: yes
0	48.0	80.0	1.020	1.0	0.0	121.0	36.0	1.2	NaN	NaN	...	True	False	False	True	True	False	False	False	False	True
1	7.0	50.0	1.020	4.0	0.0	NaN	18.0	0.8	NaN	NaN	...	True	False	False	False	False	False	False	False	False	True
2	62.0	80.0	1.010	2.0	3.0	423.0	53.0	1.8	NaN	NaN	...	True	False	False	False	True	False	True	False	True	True
3	48.0	70.0	1.005	4.0	0.0	117.0	56.0	3.8	111.0	2.5	...	False	True	False	True	False	False	True	True	True	True
4	51.0	80.0	1.010	2.0	0.0	106.0	26.0	1.4	NaN	NaN	...	True	False	False	False	False	False	False	False	False	True

5 rows × 25 columns

In [58]:
```
from sklearn.impute import KNNImputer
```

In [59]:
```
# Impute NaN values with KNN Imputer
imputer = KNNImputer(weights='distance', n_neighbors=8)
df_imputed = pd.DataFrame(imputer.fit_transform(df_onehot), columns=df_onehot.columns)
df_imputed.head()
```

Out[59]:

	age	bp	sg	al	su	bgr	bu	sc	sod	pot	...	pc: normal	pcc: present	ba: present	htn: yes	dm: yes	cad: yes	appet: poor	pe: yes	ane: yes	classification: yes
0	48.0	80.0	1.020	1.0	0.0	121.000000	36.0	1.2	139.802900	4.369414	...	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0
1	7.0	50.0	1.020	4.0	0.0	113.868407	18.0	0.8	137.197465	3.789831	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2	62.0	80.0	1.010	2.0	3.0	423.000000	53.0	1.8	133.359693	4.255775	...	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0
3	48.0	70.0	1.005	4.0	0.0	117.000000	56.0	3.8	111.000000	2.500000	...	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0
4	51.0	80.0	1.010	2.0	0.0	106.000000	26.0	1.4	139.121421	4.080290	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

5 rows × 25 columns

In [60]:
```
# Verify imputation
pd.DataFrame(df_imputed.isnull().sum(), columns=["Missing Values"])
```

Out[60]:

	Missing Values
age	0
bp	0
sg	0
al	0
su	0
bgr	0
bu	0
sc	0
sod	0
pot	0
hemo	0
pcv	0
wc	0
rc	0
rbc: normal	0
pc: normal	0
pcc: present	0
ba: present	0
htn: yes	0
dm: yes	0
cad: yes	0
appet: poor	0
pe: yes	0
ane: yes	0
classification: yes	0

In [61]:
```
# Last check
pd.DataFrame({
    'Column': df_imputed.columns,
    'DataType': df_imputed.dtypes.values,
    'UniqueValues': [df_imputed[col].unique() for col in df_imputed.columns],
    'UniqueValuesCount': df_imputed.nunique().values
})
```

Out[61]:

	Column	DataType	UniqueValues	UniqueValuesCount
0	age	float64	[48.0, 7.0, 62.0, 51.0, 60.0, 68.0, 24.0, 52.0...	84
1	bp	float64	[80.0, 50.0, 70.0, 90.0, 70.05318724012041, 10...	21
2	sg	float64	[1.02, 1.01, 1.005, 1.015, 1.0177123677017632,...	51
3	al	float64	[1.0, 4.0, 2.0, 3.0, 0.0, 0.9385069173670606, ...	51
4	su	float64	[0.0, 3.0, 4.0, 1.0, 0.49923681536713876, 2.0,...	45
5	bgr	float64	[121.0, 113.86840711503898, 423.0, 117.0, 106....	189
6	bu	float64	[36.0, 18.0, 53.0, 56.0, 26.0, 25.0, 54.0, 31....	136
7	sc	float64	[1.2, 0.8, 1.8, 3.8, 1.4, 1.1, 24.0, 1.9, 7.2,...	100
8	sod	float64	[139.8028995177722, 137.19746549701304, 133.35...	120
9	pot	float64	[4.369414395481028, 3.7898307442486243, 4.2557...	127
10	hemo	float64	[15.4, 11.3, 9.6, 11.2, 11.6, 12.2, 12.4, 10.8...	166
11	pcv	float64	[44.0, 38.0, 31.0, 32.0, 35.0, 39.0, 36.0, 33....	112
12	wc	float64	[7800.0, 6000.0, 7500.0, 6700.0, 7300.0, 9414....	194
13	rc	float64	[5.2, 5.238422569399005, 4.193417162341338, 3....	175
14	rbc: normal	float64	[0.0, 1.0]	2
15	pc: normal	float64	[1.0, 0.0]	2
16	pcc: present	float64	[0.0, 1.0]	2
17	ba: present	float64	[0.0, 1.0]	2
18	htn: yes	float64	[1.0, 0.0]	2
19	dm: yes	float64	[1.0, 0.0]	2
20	cad: yes	float64	[0.0, 1.0]	2
21	appet: poor	float64	[0.0, 1.0]	2
22	pe: yes	float64	[0.0, 1.0]	2
23	ane: yes	float64	[0.0, 1.0]	2
24	classification: yes	float64	[1.0, 0.0]	2

In [62]:
```
df_imputed.shape
```

Out[62]:
```
(400, 25)
```

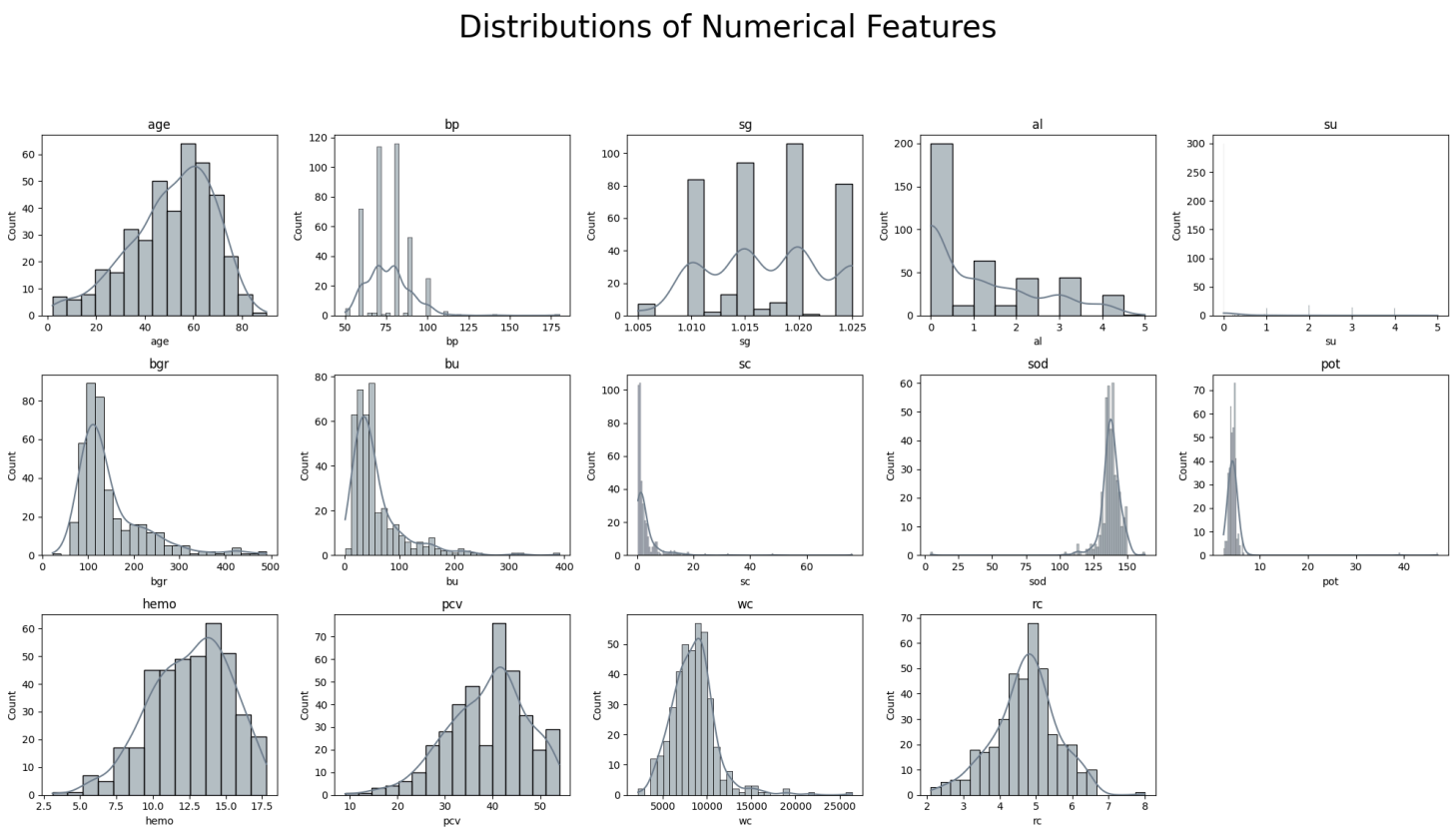
---

4. **Data Transformation**:

- **Check Distributions of Numerical Features**: Visualize their distributions and identify skewness.

- **Skewness Analysis**: Identify highly skewed columns (`su` , `bgr` , `bu` , `sc` , `sod` , `pot` , and `wc`).
- **Try Different Transformations**: `Log` , `QuantileTransformer` with `normal` distribution, and `QuantileTransformer` with `uniform` distribution.
- **Compare Results**:
  - **Normal-Quantile Transformation** was chosen as it reduces skewness to near-zero and preserves natural relationships.
  - Log Transformation was less effective for highly skewed features and non-positive values.
  - Uniform-Quantile Transformation lacks the interpretability of a normal distribution.

In [63]:
```
# Check the distributions of numerical features
numerical_features = df.select_dtypes(include='float64').columns.tolist()
num_features = len(numerical_features)

# Grid-histogram
cols = 5
rows = math.ceil(num_features / cols)

fig, axes = plt.subplots(rows, cols, figsize=(cols * 4, rows * 4))
fig.suptitle('\n\nDistributions of Numerical Features\n', y=1.02, fontsize=30)
axes = axes.flatten()

for i, feature in enumerate(numerical_features):
    sns.histplot(df_imputed[feature], kde=True, ax=axes[i], color="slategray")
    axes[i].set_title(f'{feature}')

# Drop redundant subplots
for j in range(i + 1, rows * cols):
    axes[j].set_visible(False)

plt.tight_layout()
#plt.savefig('transformation_distributions.png', dpi=300)
plt.show()

# Check the skewness of numerical features
skewness_values = df_imputed[numerical_features].skew()

skewness_df = pd.DataFrame({
    "Feature": numerical_features,
    "Skewness": skewness_values.values
})

def highlight_skewness(value):
    if abs(value) > 1.65:
        return 'background-color: lightcoral'

styled_skewness_df = skewness_df.style.applymap(highlight_skewness, subset=['Skewness'])

styled_skewness_df
```



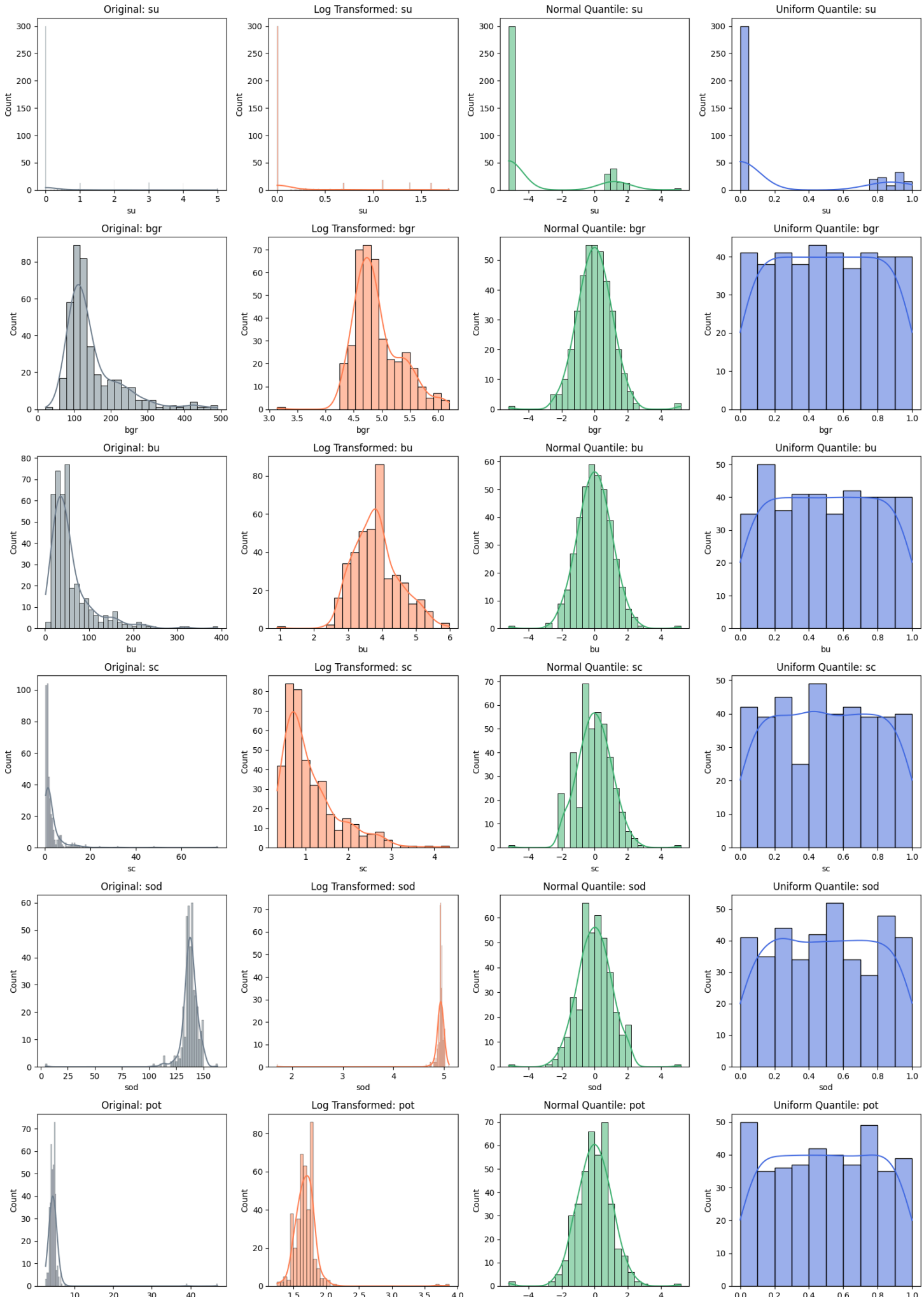
Out[63]:

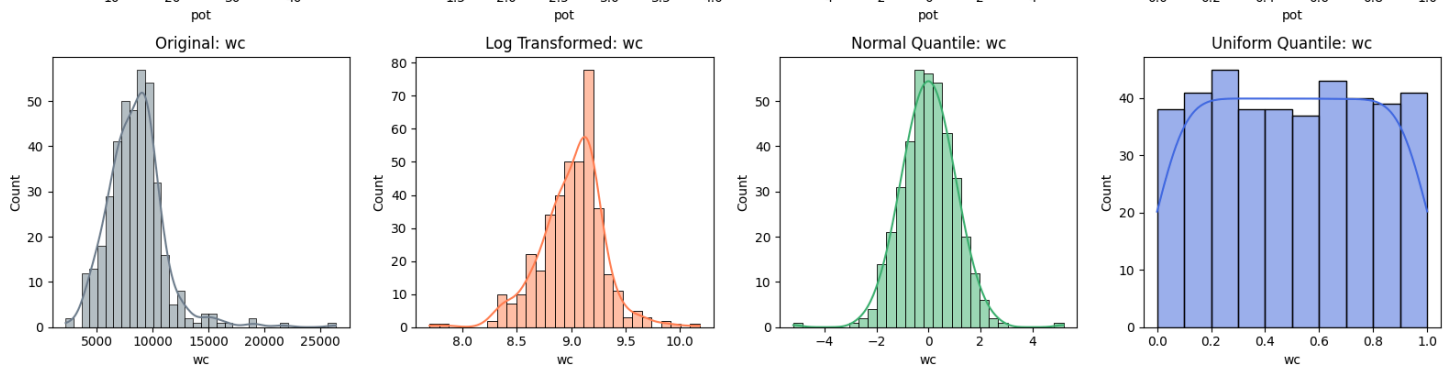
	Feature	Skewness
0	age	-0.623302
1	bp	1.618489
2	sg	-0.054562
3	al	0.980244
4	su	2.502877
5	bgr	1.976905
6	bu	2.626584
7	sc	7.616492
8	sod	-7.648545
9	pot	13.072692
10	hemo	-0.396326
11	pcv	-0.473501
12	wc	1.680153
13	rc	-0.225727

In [64]:
```
from sklearn.preprocessing import QuantileTransformer
```

In [65]:
```
# Specify the features to be transformed
features_to_transform = ['su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'wc']

# Try different transformations
df_log_transformed = df_imputed.copy()
df_normal_quantile = df_imputed.copy()
df_uniform_quantile = df_imputed.copy()

# Log transform
for feature in features_to_transform:
    df_log_transformed[feature] = df_log_transformed[feature].apply(lambda x: np.log(x + 1) if x > 0 else 0)

# QuantileTransformer - Normal distribution
qt_normal = QuantileTransformer(output_distribution='normal', random_state=42)
df_normal_quantile[features_to_transform] = qt_normal.fit_transform(df_imputed[features_to_transform])

# QuantileTransformer - Uniform distribution
qt_uniform = QuantileTransformer(output_distribution='uniform', random_state=42)
df_uniform_quantile[features_to_transform] = qt_uniform.fit_transform(df_imputed[features_to_transform])

# Comparison of different transformations
cols = 4
rows = len(features_to_transform)

fig, axes = plt.subplots(rows, cols, figsize=(cols * 4, rows * 4))
fig.suptitle('\n\nComparison of Different Transformations', y=1.02, fontsize=30)

for i, feature in enumerate(features_to_transform):
    # Original distribution
    sns.histplot(df_imputed[feature], kde=True, ax=axes[i, 0], color="slategray")
    axes[i, 0].set_title(f'Original: {feature}')

    # Log transform distribution
    sns.histplot(df_log_transformed[feature], kde=True, ax=axes[i, 1], color="coral")
    axes[i, 1].set_title(f'Log Transformed: {feature}')

    # Normal QuantileTransformer distribution
    sns.histplot(df_normal_quantile[feature], kde=True, ax=axes[i, 2], color="mediumseagreen")
    axes[i, 2].set_title(f'Normal Quantile: {feature}')

    # Uniform QuantileTransformer distribution
    sns.histplot(df_uniform_quantile[feature], kde=True, ax=axes[i, 3], color="royalblue")
    axes[i, 3].set_title(f'Uniform Quantile: {feature}')

plt.tight_layout()
#plt.savefig('transformation_comparison.png', dpi=300)
plt.show()
```

Comparison of Different Transformations





In [66]:
```
# Determind the best transformer
print("-" * 25)
print("(Before Transformation)")
print("\nOriginal:")
print(df_imputed[features_to_transform].skew().to_frame(name="Skewness"))
print("-" * 25)
print("(After Transformation)")

print("\nLog:")
print(df_log_transformed[features_to_transform].skew().to_frame(name="Skewness"))

print("\nNormal-Quantile:")
print(df_normal_quantile[features_to_transform].skew().to_frame(name="Skewness"))

print("\nUniform-Quantile:")
print(df_uniform_quantile[features_to_transform].skew().to_frame(name="Skewness"))
```

```
-------------------------
(Before Transformation)

Original:
      Skewness
su    2.502877
bgr   1.976905
bu    2.626584
sc    7.616492
sod  -7.648545
pot  13.072692
wc    1.680153
-------------------------
(After Transformation)

Log:
      Skewness
su    1.890152
bgr   0.702522
bu    0.344213
sc    1.553040
sod -17.615337
pot   6.164567
wc   -0.302077

Normal-Quantile:
     Skewness
su   1.242259
bgr  0.235580
bu   0.011175
sc   0.070180
sod -0.048054
pot -0.216396
wc   0.002321

Uniform-Quantile:
     Skewness
su   1.189813
bgr -0.000041
bu  -0.000499
sc   0.002412
sod  0.001557
pot  0.000398
wc   0.000317
```

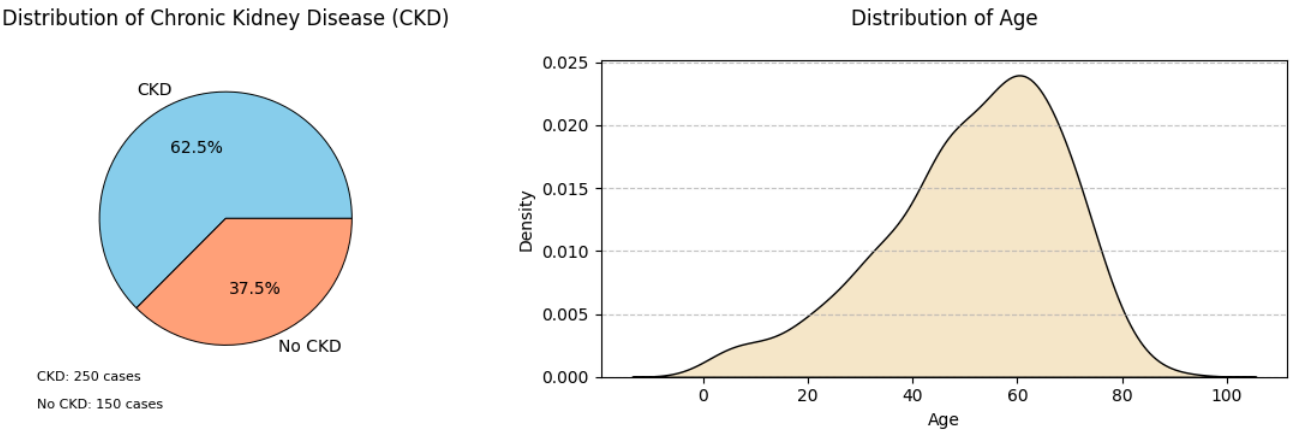
Step 2: Visualizations

1. **Univariate Analysis**:

- **Pie Chart**: The dataset shows a moderate imbalance, with **62.5%** of cases classified as CKD and **37.5%** as No CKD.
- **KDE Plot**: The Age distribution shows a peak around 60 years and a gradual decline after that.

In [67]:
```
# Create a figure with two subplots (1 row, 2 columns)
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# CKD distribution
classification_counts = df['classification'].value_counts()
ckd_labels = {'yes': 'CKD', 'no': 'No CKD'}

axes[0].pie(classification_counts,
            labels=[ckd_labels[label] for label in classification_counts.index],
            autopct='%1.1f%%',
            colors=['skyblue', 'lightsalmon'],
            wedgeprops={'edgecolor': 'black', 'linewidth': 0.7})

axes[0].set_title('\nDistribution of Chronic Kidney Disease (CKD)\n', fontsize=12)

ckd_count = classification_counts.get('yes', 0)
```

```
no_ckd_count = classification_counts.get('no', 0)
axes[0].text(-1.5, -1.5, f"CKD: {ckd_count} cases\n\nNo CKD: {no_ckd_count} cases",
             ha='left', fontsize=8)

# BU distribution
sns.kdeplot(df['age'], ax=axes[1], shade=True, color='wheat', edgecolor='black', alpha=0.7)
axes[1].set_title('\nDistribution of Age\n', fontsize=12)
axes[1].set_xlabel('Age', fontsize=10)
axes[1].set_ylabel('Density', fontsize=10)
axes[1].grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
#plt.savefig('Univariate_Analysis.png', dpi=300, bbox_inches='tight')
plt.show()
```

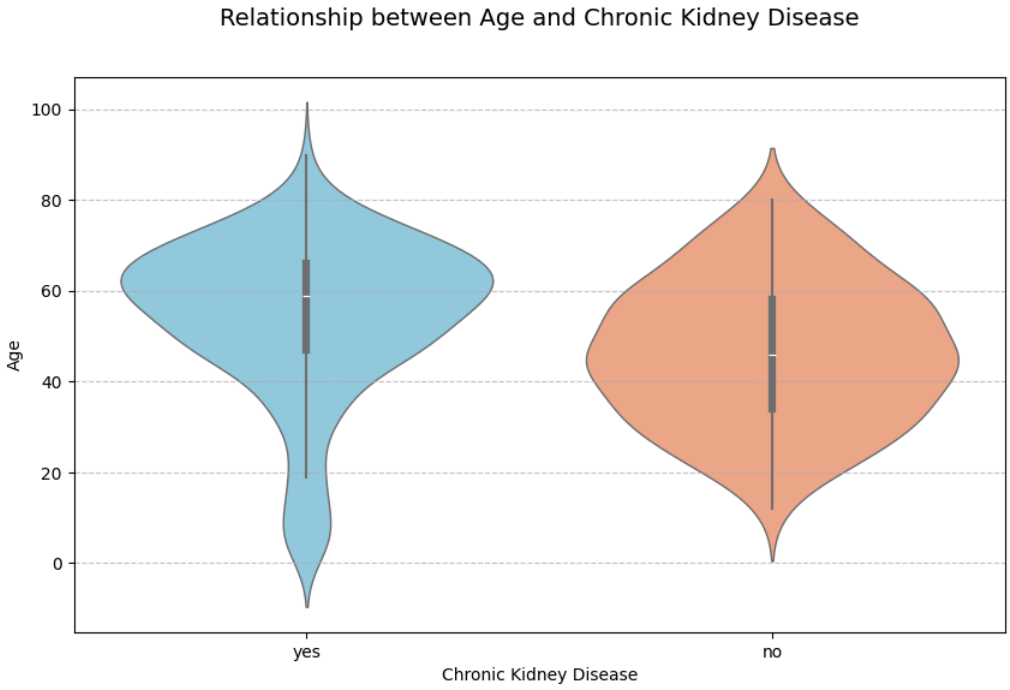


---

2. **Bivariate Analysis**:

- **Violin plot**: The median age is higher for those with Chronic Kidney Disease (CKD), suggesting a correlation between older age and CKD, which aligns with medical knowledge that CKD prevalence increases with age.

In [68]:
```
# Relationship between BU and CKD
color_map = {'yes': 'skyblue', 'no': 'lightsalmon'}

plt.figure(figsize=(10, 6))
sns.violinplot(
    data=df,
    x='classification',
    y='age',
    palette=color_map,
    inner='box',
    linewidth=1
)

plt.title('\nRelationship between Age and Chronic Kidney Disease\n', fontsize=14, pad=15)
plt.xlabel('Chronic Kidney Disease', fontsize=10)
plt.ylabel('Age', fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

#plt.savefig('Bivariate_Analysis.png', dpi=300, bbox_inches='tight')
plt.show()
```

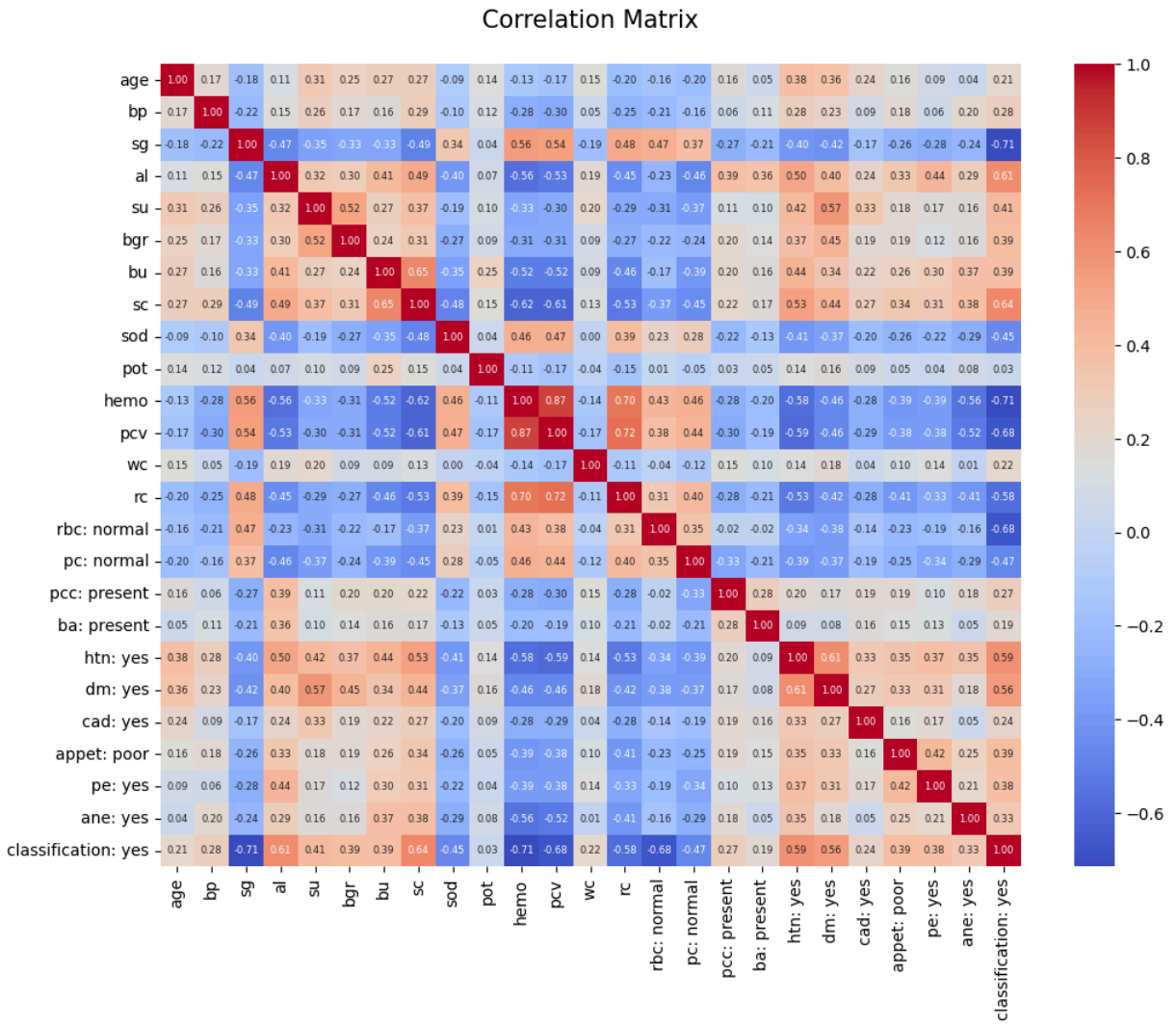


---

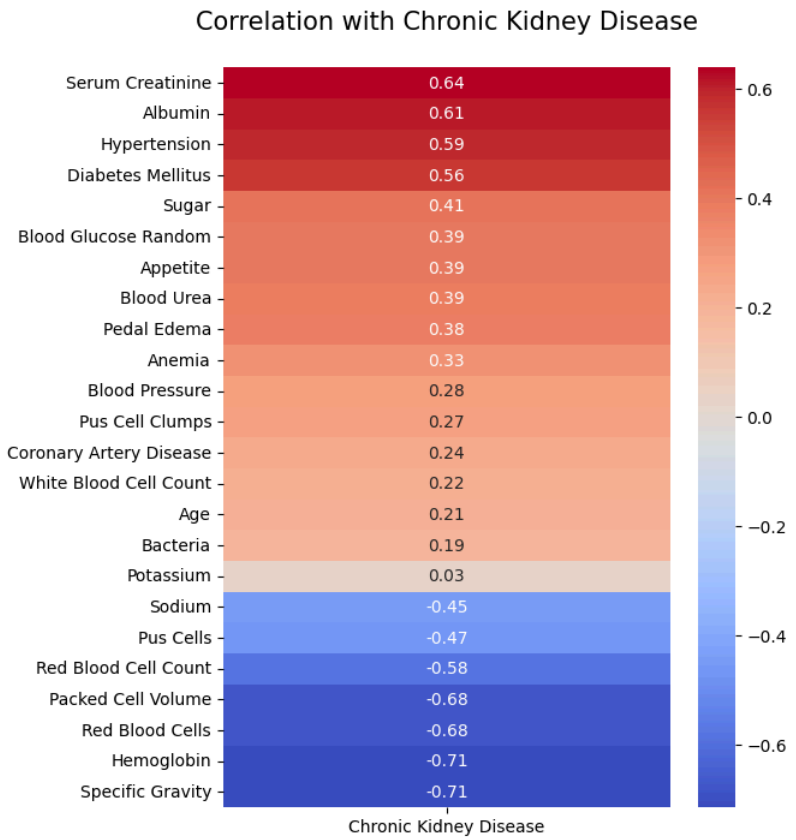
3. **Correlation Analysis**:

- **Correlation Matrix**: `Hemoglobin` and `Packed Cell Volume` are highly correlated (0.87) due to their shared physiological basis. Both features provide complementary and clinically valuable information, making it reasonable to retain both.
- **Target Correlation**:
  - **Strong Positive Correlations**: `Serum Creatinine` (0.64) and `Albumin` (0.61) are key indicators of kidney damage and proteinuria.
  - **Strong Negative Correlations**: `Specific Gravity` (-0.71) and `Hemoglobin` (-0.71) indicate impaired urine concentration and excretory function. `Red Blood Cells` (-0.68) and `Packed Cell Volume` (-0.68) reflect reduced kidney filtration and anemia.

In [69]:
```
# Calculate correlation
correlation_matrix = df_normal_quantile.corr()

# Correlation matrix
plt.figure(figsize=(12, 9))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", annot_kws={"size": 6})
plt.title("\nCorrelation Matrix\n", fontsize=15)

#plt.savefig('correlation_matrix.png', dpi=300, bbox_inches='tight')
plt.show()
```



In [70]:
```
# Convert abbreviations to real names
feature_description=['Age','Blood Pressure','Specific Gravity','Albumin','Sugar','Blood Glucose Random',
                     'Blood Urea','Serum Creatinine','Sodium','Potassium','Hemoglobin',
                     'Packed Cell Volume','White Blood Cell Count','Red Blood Cell Count','Red Blood Cells','Pus Cells',
                     'Pus Cell Clumps','Bacteria','Hypertension','Diabetes Mellitus',
                     'Coronary Artery Disease','Appetite','Pedal Edema','Anemia','Chronic Kidney Disease']

df_normal_quantile.columns=feature_description

# Calculate correlation
correlation_matrix = df_normal_quantile.corr()

# Extract and sort the correlation between features and target variable
target_correlation = (
    correlation_matrix[['Chronic Kidney Disease']]
    .drop(index='Chronic Kidney Disease')
    .sort_values(by='Chronic Kidney Disease', ascending=False)
)

# Correlation matrix with target variable
plt.figure(figsize=(6, 8))
sns.heatmap(target_correlation, annot=True, fmt=".2f", cmap="coolwarm", cbar=True, annot_kws={"size": 10})
plt.title("\nCorrelation with Chronic Kidney Disease\n", fontsize=15)

#plt.savefig('correlation_target.png', dpi=300, bbox_inches='tight')
plt.show()
```



---

4 Models

Step 1: Unsupervised Learning

1. **Dimensionality Reduction**:

- T-SNE:
  - **With Target Variable**: Clearly separated.
  - **Without Target Variable**: Still shows some degree of separability between the two classes. However, the clusters are less distinct compared to when the target variable is included.

- PCA:
  - **PCA Component Analysis**: The first PCA component captures the largest amount of variation in the dataset.
  - **PCA with 1 Component**: Clearly separated.
  - **PCA with 2 Components**: Some overlap.

In [71]:
```
# Separate features and target
X = df_normal_quantile.drop('Chronic Kidney Disease', axis=1)
y = df_normal_quantile['Chronic Kidney Disease']
```

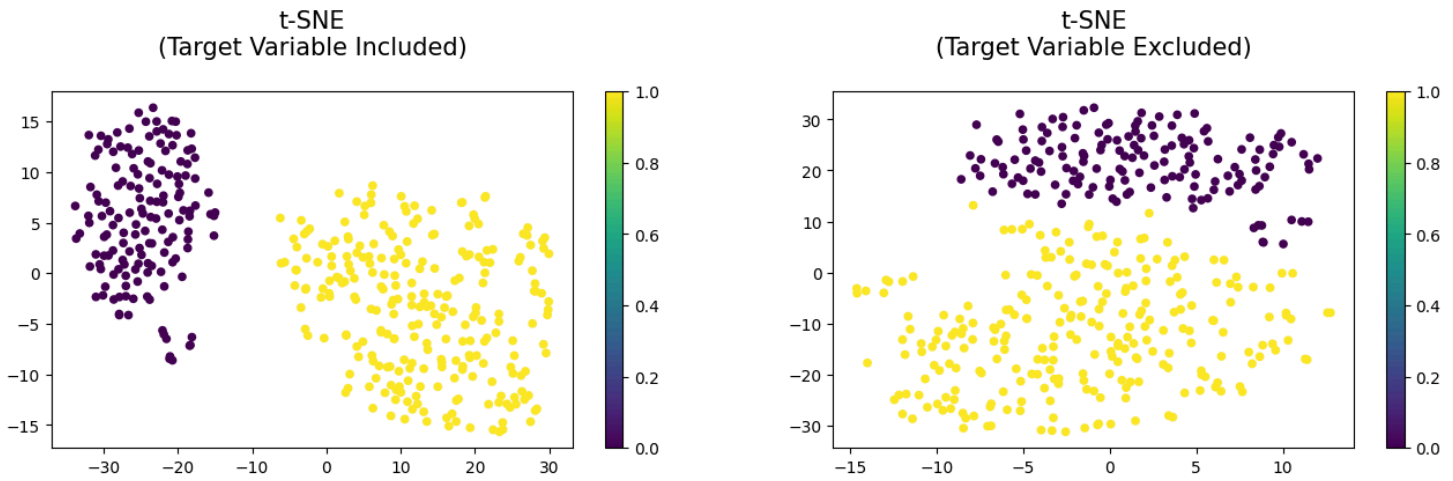
In [72]:
```
from sklearn.preprocessing import StandardScaler
```

In [73]:
```
# Scale the data
ss = StandardScaler()
X_std = ss.fit_transform(X)
df_std = ss.fit_transform(df_normal_quantile)
```

In [74]:
```
from sklearn.manifold import TSNE
```

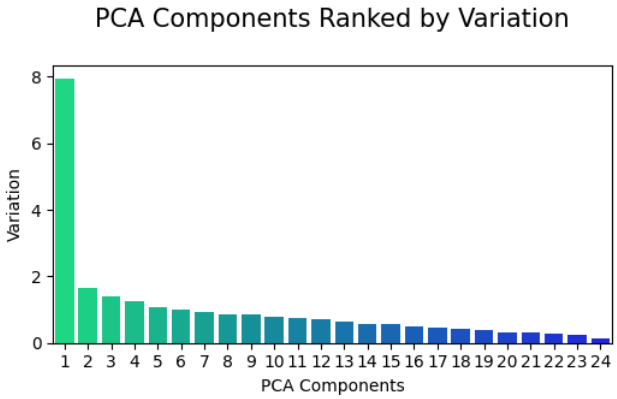
In [75]:
```
# TSNE with target variable
tsne_model = TSNE(n_components=2, random_state=42)
tsne_data_with_target = tsne_model.fit_transform(df_std)

xs_with_target = tsne_data_with_target[:, 0]
ys_with_target = tsne_data_with_target[:, 1]

# TSNE without target variable
tsne_data_no_target = tsne_model.fit_transform(X_std)

xs_no_target = tsne_data_no_target[:, 0]
ys_no_target = tsne_data_no_target[:, 1]

# Plotting both graphs
fig, axes = plt.subplots(1, 2, figsize=(16, 4), facecolor='white')

# Plot with target variable
axes[0].set_facecolor('white')
scatter1 = axes[0].scatter(xs_with_target, ys_with_target,
                           c=pd.get_dummies(y, drop_first=True).values, cmap="viridis", s=20)
axes[0].set_title("\nt-SNE\n(Target Variable Included)\n", fontsize=15)
fig.colorbar(scatter1, ax=axes[0])

# Plot without target variable
axes[1].set_facecolor('white')
scatter2 = axes[1].scatter(xs_no_target, ys_no_target,
                           c=pd.get_dummies(y, drop_first=True).values, cmap="viridis", s=20)
```

```
axes[1].set_title("\nt-SNE\n(Target Variable Excluded)\n", fontsize=15)
fig.colorbar(scatter2, ax=axes[1])

#plt.savefig('tsne.png', dpi=300, bbox_inches='tight')
plt.show()
```

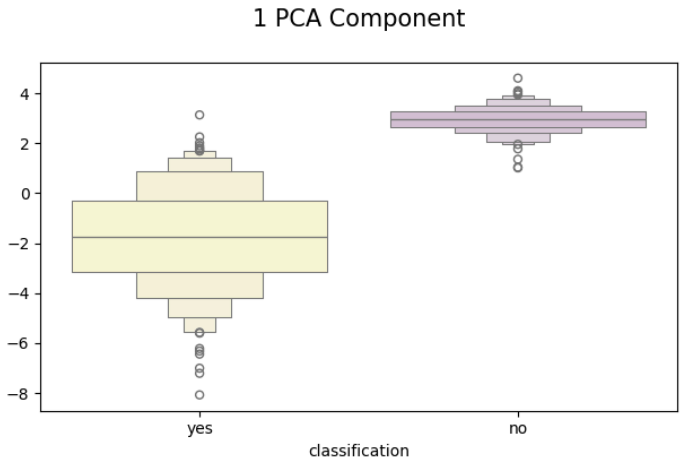


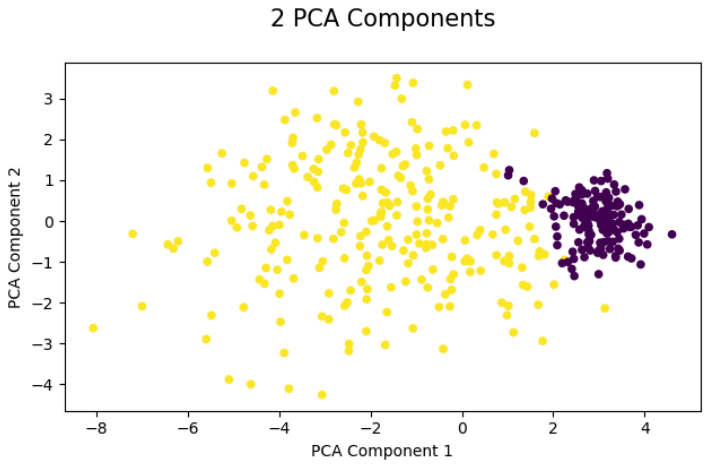
In [76]:
```
from sklearn.decomposition import PCA
```

In [77]:
```
# PCA
pca = PCA()
pca.fit(X_std)
pca_data = pca.fit_transform(X_std)
pca_features = list(range(1, len(pca.explained_variance_) + 1))

fig, ax = plt.subplots(figsize=(6, 3), facecolor='white')
ax.set_facecolor('white')

sns.barplot(x=pca_features, y=pca.explained_variance_, palette="winter_r", ax=ax)
ax.set_ylabel('Variation', fontsize=10)
ax.set_xlabel('PCA Components', fontsize=10)
ax.set_title("\nPCA Components Ranked by Variation\n", fontsize=15)

#plt.savefig('PCA.png', dpi=300, bbox_inches='tight')
plt.show()
```



In [78]:
```
# Box plot for 1 PCA component
pca1_data = pca_data[:, 0]

custom_palette = {"yes": "lemonchiffon", "no": "thistle"}

df['classification'] = df['classification'].astype(str)

fig, axes = plt.subplots(1, 2, figsize=(16, 4), facecolor='white')
axes[0].set_facecolor('white')

sns.boxenplot(y=pca1_data, x=df['classification'], palette=custom_palette, showfliers=True, ax=axes[0])
axes[0].set_title("\n1 PCA Component\n", fontsize=15)

# Scatter plot for 2 PCA components
pca2_data = pca_data[:, :2]
scatter = axes[1].scatter(pca2_data[:, 0], pca2_data[:, 1],
                          c=pd.get_dummies(y, drop_first=True).values, cmap="viridis", s=20)
axes[1].set_facecolor('white')
axes[1].set_title("\n2 PCA Components\n", fontsize=15)
axes[1].set_xlabel('PCA Component 1', fontsize=10)
axes[1].set_ylabel('PCA Component 2', fontsize=10)

#plt.savefig('PCA_comparison.png', dpi=300, bbox_inches='tight')
plt.show()
```





---

2. **Clustering**:

- K-Means:
  - **Elbow Method**: `k=2` is optimal, aligning with the binary nature of CKD classification (yes/no).
  - **Clustering on Training Data**: Our best result is with 5-7 clusters. **(Training Accuracy = 278/280 = 99%)**
  - **Clustering on Test Data**: Setting `n_clusters=6` . **(Test Accuracy = 118/120 = 98%)**

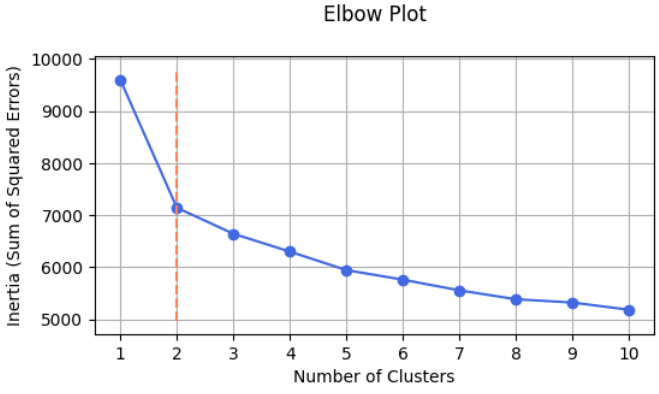
In [79]:
```
from sklearn.model_selection import train_test_split
```

In [80]:
```
# Split the data 70:30
X_train, X_test, y_train, y_test = train_test_split(X_std, y, test_size=0.3, random_state=42)

print("Training set size:", X_train.shape[0])
print("Test set size:", X_test.shape[0])
```

```
Training set size: 280
Test set size: 120
```

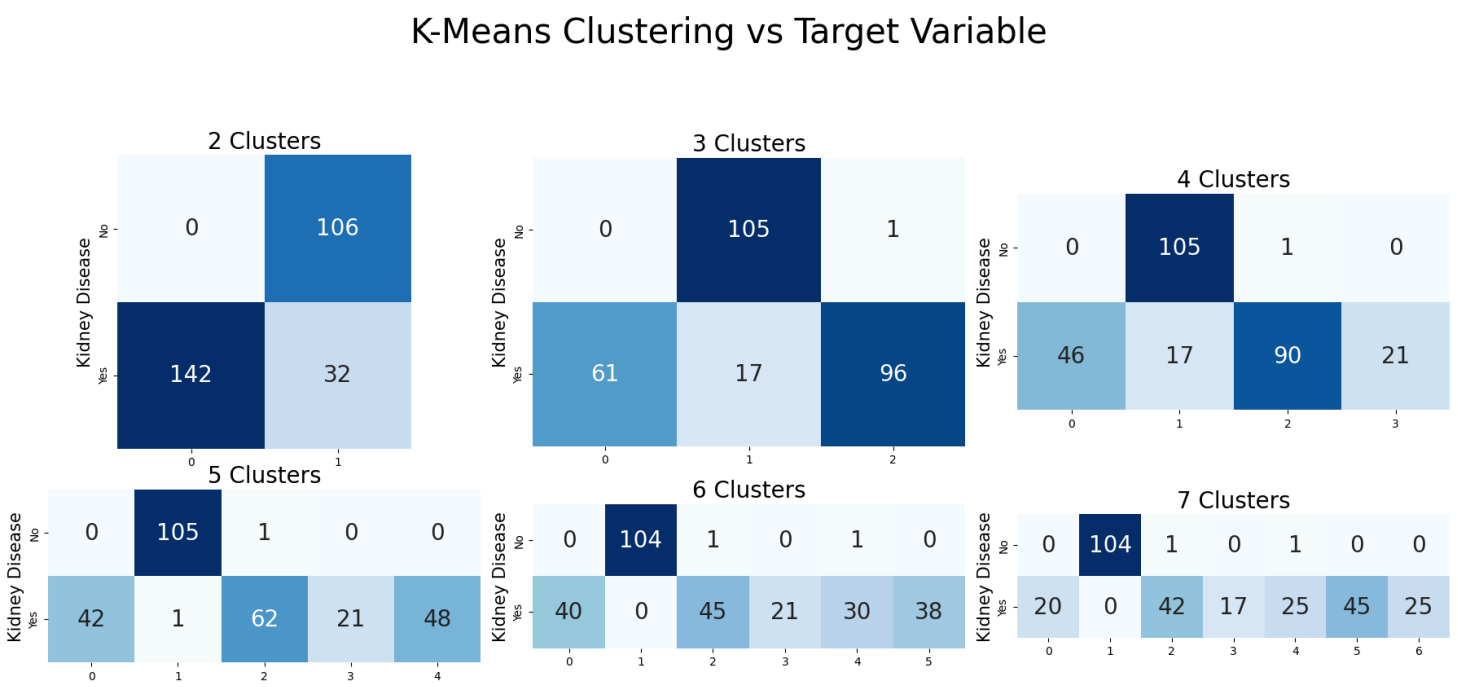
In [81]:
```
from sklearn.cluster import KMeans
```

In [82]:
```
# Elbow Plot
inertia_values = []
cluster_range = range(1, 11)

for k in cluster_range:
    model = KMeans(n_clusters=k, random_state=42)
    model.fit(X_std)
    inertia_values.append(model.inertia_)

inertia_diffs = np.diff(inertia_values)

elbow_point = np.argmin(inertia_diffs[1:]) + 2

plt.figure(figsize=(6, 3))
plt.plot(cluster_range, inertia_values, marker='o', linestyle='-', color='royalblue')
plt.vlines(elbow_point, plt.ylim()[0], plt.ylim()[1], linestyles='dashed', colors='coral')
plt.title("\nElbow Plot\n")
plt.xlabel("Number of Clusters")
plt.ylabel("Inertia (Sum of Squared Errors)")
plt.xticks(cluster_range)
plt.grid(True)

#plt.savefig('clustering_elbow.png', dpi=300, bbox_inches='tight')
plt.show()
```



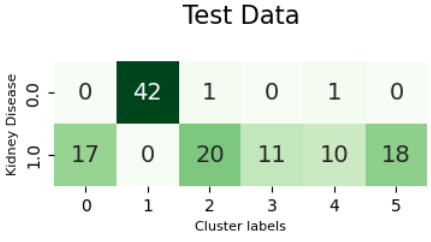
In [83]:
```
# K-Means clustering on training data
n_rows, n_cols = (2, 3)

figure, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(18, 10), facecolor='white')
figure.suptitle('\n\nK-Means Clustering vs Target Variable\n', fontsize=30)

for index, clusters in enumerate(range(2, 8)):

    i, j = (index // n_cols), (index % n_cols)

    model = KMeans(n_clusters=clusters, random_state=42)
```

```
    model.fit(X_train)
    cluster_labels = model.predict(X_train)

    sns.heatmap(pd.crosstab(y_train, cluster_labels),
                ax=axes[i, j],
                cmap='Blues',
                square=True,
                cbar=False,
                annot=True,
                annot_kws={'fontsize':20},
                fmt='d')

    axes[i, j].set_title(f"{clusters} Clusters", fontsize=20)
    axes[i, j].set_ylabel("Kidney Disease", fontsize=15)
    axes[i, j].set_xlabel(None)
    axes[i, j].set_xticklabels(axes[i, j].get_xticklabels(), fontsize=10)
    axes[i, j].set_yticklabels(["No", "Yes"], fontsize=10)

plt.tight_layout(rect=[0, 0, 1, 0.96])
#plt.savefig('clustering_train.png', dpi=300, bbox_inches='tight')
plt.show()
```



In [84]:

```
# K-Means clustering on test data
model=KMeans(n_clusters=6, random_state=42)
model.fit(X_train)

# Predict for test data
cluster_labels=model.predict(X_test)

# Heatmap
fig, ax = plt.subplots(figsize=(4, 3), facecolor='white')
ax.set_facecolor('white')

sns.heatmap(pd.crosstab(y_test,cluster_labels),
                cmap='Greens',
                square='True',
                cbar=False,
                annot=True,
                annot_kws={'fontsize':14},
                fmt='d')

plt.title("\nTest Data\n",fontsize=15)
plt.ylabel("Kidney Disease",fontsize=8)
plt.xlabel("Cluster labels",fontsize=8)

#plt.savefig('clustering_test.png', dpi=300, bbox_inches='tight')
plt.show()
```



Step 2: Supervised Learning

1. **Random Forest**: Using `RandomForestClassifier` with 50 decision trees `n_estimators=50`.
2. **Gradient Boosting**: Using `GradientBoostingClassifier` with 50 decision trees `n_estimators=50`.

3. **Neural Network**: Using `MLPClassifier` with two hidden layers (24 and 12 neurons) `hidden_layer_sizes=(24, 12)` and L2 regularization `alpha=0.001`.

In [85]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
```

In [86]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [87]:
```
# Define models
models = {
    "Random Forest": RandomForestClassifier(n_estimators=50, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(n_estimators=50, random_state=42),
    "Neural Network": MLPClassifier(hidden_layer_sizes=(24, 12), alpha=0.001, random_state=42)
}

# Function to evaluate and display results for each model
def evaluate_models(models, X_train, X_test, y_train, y_test):
    for name, model in models.items():
        # Train the model
        model.fit(X_train, y_train)

        # Predict on the test set
        y_pred = model.predict(X_test)

        # Calculate accuracy
        print("-" * 25)
        accuracy = accuracy_score(y_test, y_pred)
        print(f"Model: {name}")
        print(f"Accuracy: {round(accuracy, 2)}")

        # Confusion matrix
        conf_matrix = confusion_matrix(y_test, y_pred)
        print(f"Confusion Matrix:\n", conf_matrix)

# Run the evaluation function
evaluate_models(models, X_train, X_test, y_train, y_test)
```

```
-------------------------
Model: Random Forest
Accuracy: 0.98
Confusion Matrix:
 [[42  2]
 [ 0 76]]
-------------------------
Model: Gradient Boosting
Accuracy: 0.97
Confusion Matrix:
 [[42  2]
 [ 1 75]]
-------------------------
Model: Neural Network
Accuracy: 1.0
Confusion Matrix:
 [[44  0]
 [ 0 76]]
```

---

5 Results and Analysis

1. **Evaluation**:

- **Performance Comparisons**:
  - **Unsupervised Learning** (KMeans): Perform remarkably well without using labels, showing the dataset's clear and separable structure. **(0.98)**
  - **Supervised Learning** (RF, GB, NN): Leverage labels for optimization, achieving similar or better results **(RF: 0.98, GB: 0.97)**. **NN**, as the most complex model, achieves perfect scores **(1.00)**.
- **Conclusion**: **KMeans** shows impressive performance, highlighting the dataset's inherent structure and clear class separability. **Supervised models** are better suited for classification tasks when labeled data is available, particularly for complex or less distinct datasets. Use KMeans for initial exploration or when labels are unavailable, but supervised methods remain the optimal choice for tasks requiring precision and reliability.

In [88]:
```
# Confusion Matrices
confusion_matrices = {
    "KMeans": np.array([[42, 2], [0, 76]]),
    "RF": np.array([[42, 2], [0, 76]]),
    "GB": np.array([[42, 2], [1, 75]]),
    "NN": np.array([[44, 0], [0, 76]]),
}

# Calculate metrics
results = []
for model, cm in confusion_matrices.items():
    tn, fp, fn, tp = cm.ravel()
    accuracy = (tp + tn) / cm.sum()
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0
    f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
    roc_auc = (tn / (tn + fp) + recall) / 2
    results.append({
        "Model": model,
        "Accuracy": round(accuracy, 3),
        "F1-Score": round(f1, 3),
        "ROC-AUC": round(roc_auc, 3)
    })

# Create a DataFrame
results_df = pd.DataFrame(results)
# results_df
# Highlight the maximum values in each metric
styled_df = results_df.style.highlight_max(subset=['Accuracy', 'F1-Score', 'ROC-AUC'], color='lightgreen') \
    .format(precision=2) \
    .set_table_styles([{
        'selector': 'thead th',
        'props': [('font-weight', 'bold'), ('text-align', 'center')]
    }, {
```

```
        'selector': 'tbody td',
        'props': [('text-align', 'center')]
    }])

styled_df
```

Out[88]:

	Model	Accuracy	F1-Score	ROC-AUC
0	KMeans	0.98	0.99	0.98
1	RF	0.98	0.99	0.98
2	GB	0.97	0.98	0.97
3	NN	1.00	1.00	1.00

---

2. **Misclassified Data Points**:

- **KMeans [1, 67]**: These points may lie near cluster boundaries, making their assignment ambiguous. Also, KMeans assumes spherical clusters and may misclassify points if the true class boundaries are non-linear.
- **Random Forest [1, 67]**: RF misclassifies the same points as KMeans, suggesting that these points may inherently be hard to classify even with label information. These points could be outliers or have feature values that overlap significantly with other classes.
- **Gradient Boosting [1, 24, 67]**: GB misclassifies an additional point (24), compared to RF and KMeans. GB models often overfit on certain samples during iterative boosting, potentially causing it to misclassify 24 if it is an outlier or noisy data point.
- **Neural Network [ ]**: NN achieves perfect classification with no misclassified points, indicating NN's high capacity allows it to fit the data perfectly, capturing non-linear boundaries and separating even challenging points like 1 and 67. However, this could also indicate overfitting, particularly given the small dataset.
- **Conclusion**:
  - **KMeans**: Shows the limitations of clustering when class boundaries are non-linear.
  - **RF and GB**: Struggle with similar points, hinting at potential data characteristics like overlapping features or outliers.
  - **NN**: Perfect classification suggests it captures the data structure fully but might risk overfitting.

In [89]:
```
# Identify misclassified points
# Kmeans
y_test_df = pd.Series(y_test).reset_index(drop=True)
misclassified_indices = y_test_df[(y_test_df == 0) & ((cluster_labels == 4) | (cluster_labels == 2))].index

X_test_df = pd.DataFrame(X_test)

misclassified_points = X_test_df.iloc[misclassified_indices]
clustering_misclassified_points = misclassified_points

# RF, GB, and NN
def evaluate_models_and_misclassifications(models, X_train, X_test, y_train, y_test):
    misclassified_points = {}

    for name, model in models.items():
        # Train the model
        model.fit(X_train, y_train)

        # Predict on the test set
        y_pred = model.predict(X_test)

        # Calculate accuracy
        accuracy = accuracy_score(y_test, y_pred)

        # Identify misclassified points
        y_test_df = pd.Series(y_test).reset_index(drop=True)
        misclassified_indices = y_test_df[y_test_df != y_pred].index
        misclassified_data = y_test_df.iloc[misclassified_indices]
        misclassified_points[name] = misclassified_data

    return misclassified_points

# Run the evaluation function
misclassified_points = evaluate_models_and_misclassifications(models, X_train, X_test, y_train, y_test)

# Compare misclassified data across clustering and classification models
print(f"Clustering Misclassified Data Points (Kmeans): {clustering_misclassified_points.index.values}")
print(f"Classification Misclassified Data Points (RF): {misclassified_points["Random Forest"].index.values}")
print(f"Classification Misclassified Data Points (GB): {misclassified_points["Gradient Boosting"].index.values}")
print(f"Classification Misclassified Data Points (NN): {misclassified_points["Neural Network"].index.values}")
```

```
Clustering Misclassified Data Points (Kmeans): [ 1 67]
Classification Misclassified Data Points (RF): [ 1 67]
Classification Misclassified Data Points (GB): [ 1 24 67]
Classification Misclassified Data Points (NN): []
```

---

6 Discussion and Conclusion

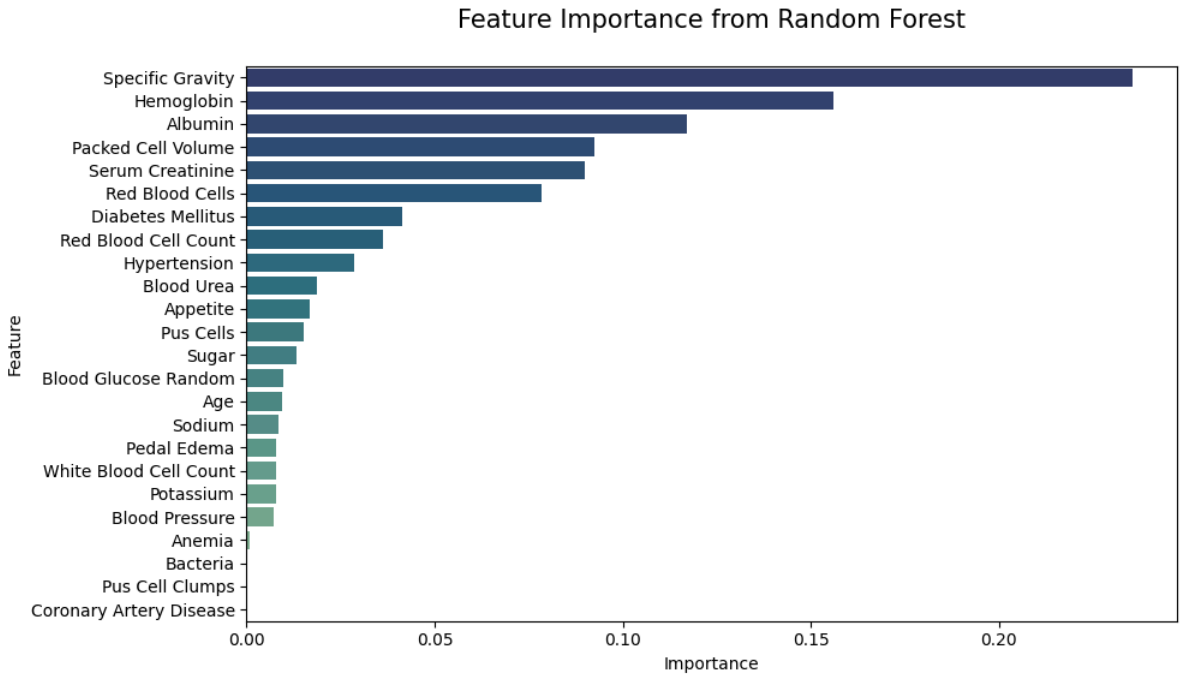
Discussion

1. **Feature Importance**:

- **RF**: Features like `Specific Gravity`, `Hemoglobin`, and `Albumin` are crucial for the RF model to make accurate predictions.
- **PCA**: Features like `Hemoglobin`, `Packed Cell Volume`, and `Serum Creatinine` are significant for explaining the largest variation in the dataset. PCA is unsupervised and does not consider the target variable, therefore, these features do not necessarily have predictive power but represent dominant trends or patterns in the dataset.
- **Correlation**: Features like `Specific Gravity`, `Hemoglobin`, and `Red Blood Cells` have the strongest linear relationships with the target. Correlation measures direct linear relationships, which may not fully capture complex or non-linear dependencies like RF does.
- **Conclusion**:
  - All methods highlight features that are important in some way—predictive **(RF)**, variance-explaining **(PCA)**, or statistically associated **(correlation)**.
  - `Hemoglobin` appear in multiple lists, indicating it is consistently important across different analyses.

- The differences highlight the complementary nature of these methods, emphasizing the importance of multi-perspective analysis for robust insights.

In [90]:
```
# Feature importance from Random Forest
model = RandomForestClassifier(n_estimators=50, random_state=42)
model.fit(X_train, y_train)

importance = model.feature_importances_
feature_importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importance
}).sort_values(by='Importance', ascending=False)

fig, ax = plt.subplots(figsize=(10, 6), facecolor='white')
sns.barplot(x='Importance', y='Feature', data=feature_importance_df, palette="crest_r", ax=ax)
ax.set_title('\nFeature Importance from Random Forest\n', fontsize=15)
ax.set_facecolor("white")
#plt.savefig('Feature_importance.png', dpi=300, bbox_inches='tight')
```



In [91]:
```
# Top three important features from RF
top_features_rf = feature_importance_df.head(3)["Feature"].values

print("-" * 55)
print("Top 3 Importance Features from RF:\n", top_features_rf)

# Top features from the first PCA component
pca = PCA(n_components=1)
pca.fit(X_train)

loadings = pd.DataFrame(pca.components_.T, columns=['PCA Component 1'], index=X.columns)

top_features_pca = loadings['PCA Component 1'].abs().sort_values(ascending=False).head(3).index.tolist()

print("-" * 55)
print("Top Features from PCA Component 1:\n", top_features_pca)

# Top three features from correlation matrix
top_features_corr = (
    target_correlation['Chronic Kidney Disease']
    .abs()
    .sort_values(ascending=False)
    .head(3)
    .index.tolist()
)

print("-" * 55)
print("Top 3 Features from Correlation Matrix:\n", top_features_corr)
```

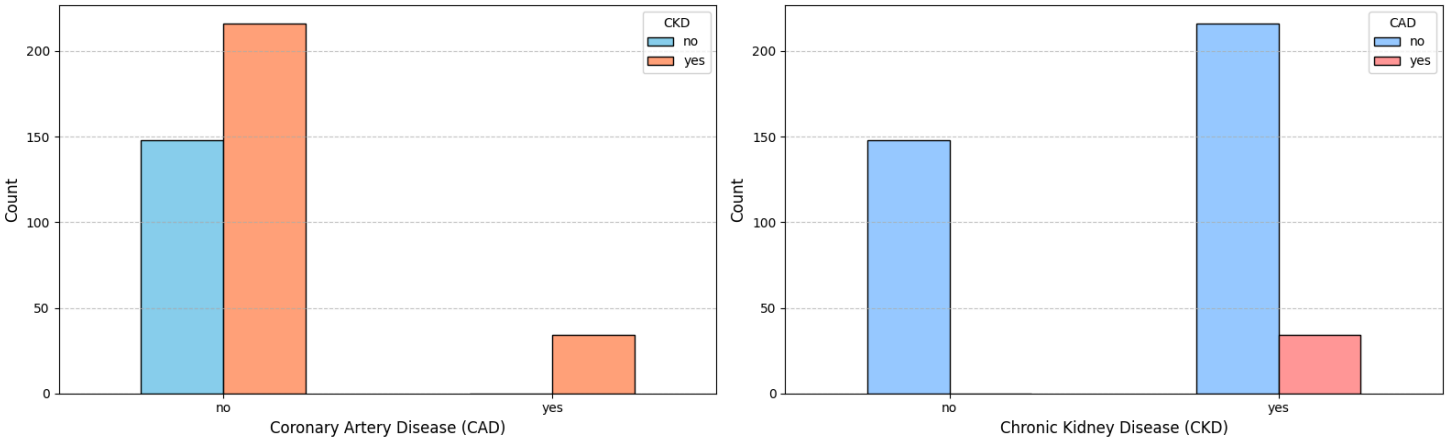
```
-------------------------------------------------------
Top 3 Importance Features from RF:
 ['Specific Gravity' 'Hemoglobin' 'Albumin']
-------------------------------------------------------
Top Features from PCA Component 1:
 ['Hemoglobin', 'Packed Cell Volume', 'Serum Creatinine']
-------------------------------------------------------
Top 3 Features from Correlation Matrix:
 ['Specific Gravity', 'Hemoglobin', 'Red Blood Cells']
```

---

2. Coronary Artery Disease (CAD):

- **Feature Importance Score of 0**: In Random Forest (RF), CAD does not directly contribute to the model's classification decisions, meaning the model relies very little (or not at all) on CAD for its predictions.
- **Domain Knowledge**: In medical knowledge, CAD is often associated with CKD.
- **Data Imbalance**: Since positive samples for CAD (cad="yes") are significantly fewer than negative samples (cad="no"), the model is more likely to rely on features with higher frequency to make decisions. This data imbalance can lead RF to treat CAD as a secondary feature, resulting in a score of 0.
- **Conclusion**: A feature importance score of 0 for CAD does not necessarily mean it is irrelevant; it could be due to data imbalance or feature correlation.

In [92]:
```
# Create subplots for two bar charts
fig, axes = plt.subplots(1, 2, figsize=(16, 5), facecolor='white')

# First plot: x-axis is 'cad', y-axis is count for 'classification'
cad_classification_counts = df.groupby(['cad', 'classification']).size().unstack()
cad_classification_counts.plot(kind='bar',
                               color=['skyblue', 'lightsalmon'],
                               edgecolor='black',
                               ax=axes[0])
axes[0].set_xlabel('Coronary Artery Disease (CAD)', fontsize=12)
axes[0].set_ylabel('Count', fontsize=12)
axes[0].legend(title='CKD', fontsize=10)
axes[0].grid(axis='y', linestyle='--', alpha=0.7)
axes[0].tick_params(axis='x', rotation=0)

# Second plot: x-axis is 'classification', y-axis is count for 'cad'
classification_cad_counts = df.groupby(['classification', 'cad']).size().unstack()
classification_cad_counts.plot(kind='bar',
                               color=['#99CCFF', '#FF9999'],
                               edgecolor='black',
                               ax=axes[1])
axes[1].set_xlabel('Chronic Kidney Disease (CKD)', fontsize=12)
axes[1].set_ylabel('Count', fontsize=12)
axes[1].legend(title='CAD', fontsize=10)
axes[1].grid(axis='y', linestyle='--', alpha=0.7)
axes[1].tick_params(axis='x', rotation=0)

plt.tight_layout()
#plt.savefig('CAD.png', dpi=300, bbox_inches='tight')
plt.show()
```



Conclusion

This project focused on clustering CKD data using KMeans, which effectively captured the dataset's clear structure, achieving 98% accuracy on test data. Dimensionality reduction techniques, such as PCA and t-SNE, revealed distinct separability between classes, further supporting KMeans' results. While supervised models like Neural Networks, Random Forest, and Gradient Boosting provided a performance benchmark, the primary goal was to evaluate the potential of unsupervised learning. This project underscores the value of clustering methods for uncovering patterns in this CKD data without relying on labels.

GitHub Repository Link

https://github.com/d93xup60126/Unsupervised_Learning_CKD_Clustering

---