In [30]:

```
# import needed libraries
import pandas as pd
import numpy as np
```

Task1 (1)

Load the row data into Pandas with the meta information being removed.

In [31]:

```
# read data from csv file
df = pd.read_csv('ckd-dataset-v2.csv')
df = df.drop([0,1]) #remove descriptive row and null row (remove the meta informatic
```

View the data

In [32]:

```
df.head()#view data
```

Out[32]:

	bp (Diastolic)	bp limit	sg	al	class	rbc	su	pc	pcc	ba	...	htn	dm	cad	appet	pe	ane
2	0	0	1.019 - 1.021	1 - 1	ckd	0	< 0	0	0	0	...	0	0	0	0	0	0
3	0	0	1.009 - 1.011	< 0	ckd	0	< 0	0	0	0	...	0	0	0	0	0	0
4	0	0	1.009 - 1.011	≥ 4	ckd	1	< 0	1	0	1	...	0	0	0	1	0	0
5	1	1	1.009 - 1.011	3 - 3	ckd	0	< 0	0	0	0	...	0	0	0	0	0	0
6	0	0	1.015 - 1.017	< 0	ckd	0	< 0	0	0	0	...	0	1	0	1	1	0

5 rows × 29 columns

Task 1 (2)

Apply ordinal encoding to all columns contains numerical bins. For instance, in the "age" column, 0 for "<12", 1 for "12-20", 2 for "20-27", etc.

Printing all unique values for each row in the dataset

In [33]:

```
for names in df.columns: #print all unique values of each row to create encoding map
    print(names, end = ' -> ')
    print(df[names].unique())
```

```
bp (Diastolic) -> ['0' '1']
bp limit -> ['0' '1' '2']
sg -> ['1.019 - 1.021' '1.009 - 1.011' '1.015 - 1.017' '≥ 1.023' '< 1.
007']
al -> ['1 - 1' '< 0' '≥ 4' '3 - 3' '2 - 2']
class -> ['ckd' 'notckd']
rbc -> ['0' '1']
su -> ['< 0' '4 - 4' '2 - 2' '3 - 4' '1 - 2' '≥ 4']
pc -> ['0' '1']
pcc -> ['0' '1']
ba -> ['0' '1']
bgr -> ['< 112' '112 - 154' '154 - 196' '406 - 448' '238 - 280' '196 -
238'
 '≥ 448' '280 - 322' '364 - 406' '322 - 364']
bu -> ['< 48.1' '48.1 - 86.2' '200.5 - 238.6' '124.3 - 162.4' '86.2 -
124.3'
 '162.4 - 200.5' '≥ 352.9' '238.6 - 276.7']
sod -> ['138 - 143' '133 - 138' '123 - 128' '143 - 148' '148 - 153' '<
118'
 '128 - 133' '118 - 123' '≥ 158']
sc -> ['< 3.65' '3.65 - 6.8' '16.25 - 19.4' '6.8 - 9.95' '13.1 - 16.2
5'
 '9.95 - 13.1' '≥ 28.85']
pot -> ['< 7.31' '≥ 42.59' '7.31 - 11.72' '38.18 - 42.59']
hemo -> ['11.3 - 12.6' '8.7 - 10' '13.9 - 15.2' '≥ 16.5' '10 - 11.3'
'7.4 - 8.7'
 '12.6 - 13.9' '15.2 - 16.5' '< 6.1' '6.1 - 7.4']
pcv -> ['33.5 - 37.4' '29.6 - 33.5' '41.3 - 45.2' '37.4 - 41.3' '≥ 49.
1'
 '21.8 - 25.7' '45.2 - 49.1' '< 17.9' '25.7 - 29.6' '17.9 - 21.8']
rbcc -> ['4.46 - 5.05' '5.05 - 5.64' '3.28 - 3.87' '3.87 - 4.46' '6.23
- 6.82'
 '5.64 - 6.23' '2.69 - 3.28' '< 2.69' '≥ 7.41']
wbcc -> ['7360 - 9740' '12120 - 14500' '14500 - 16880' '4980 - 7360'
'< 4980'
 '9740 - 12120' '16880 - 19260' '≥ 24020' '19260 - 21640']
htn -> ['0' '1']
dm -> ['0' '1']
cad -> ['0' '1']
appet -> ['0' '1']
pe -> ['0' '1']
ane -> ['0' '1']
grf -> ['≥ 227.944' '127.281 - 152.446' '102.115 - 127.281' '177.612 -
202.778'
 '26.6175 - 51.7832' '51.7832 - 76.949' '76.949 - 102.115'
 '152.446 - 177.612' '202.778 - 227.944' '< 26.6175' ' p ']
stage -> ['s1' 's4' 's3' 's2' 's5']
affected -> ['1' '0']
age -> ['< 12' '12 - 20' '20 - 27' '27 - 35' '35 - 43' '43 - 51' '51 -
59'
 '59 - 66' '66 - 74' '≥ 74']
```

Notice that grf consists of numerical bins except for 1 value which is 'p', therefore the row needs to be cleaned up.

In [34]:

```
df = df[df.grf != ' p ']#removing the row with p.
```

In [35]:

```
#running unique values for dataframe again.
for names in df.columns: #print all unique values of each row to create encoding map
    print(names, end = ' -> ')
    print(df[names].unique())# note that p is now missing from grf
```

```
bp (Diastolic) -> ['0' '1']
bp limit -> ['0' '1' '2']
sg -> ['1.019 - 1.021' '1.009 - 1.011' '1.015 - 1.017' '≥ 1.023' '< 1.
007']
al -> ['1 - 1' '< 0' '≥ 4' '3 - 3' '2 - 2']
class -> ['ckd' 'notckd']
rbc -> ['0' '1']
su -> ['< 0' '4 - 4' '2 - 2' '3 - 4' '1 - 2' '≥ 4']
pc -> ['0' '1']
pcc -> ['0' '1']
ba -> ['0' '1']
bgr -> ['< 112' '112 - 154' '154 - 196' '406 - 448' '238 - 280' '196 -
238'
 '≥ 448' '280 - 322' '364 - 406' '322 - 364']
bu -> ['< 48.1' '48.1 - 86.2' '200.5 - 238.6' '124.3 - 162.4' '86.2 -
124.3'
 '162.4 - 200.5' '≥ 352.9' '238.6 - 276.7']
sod -> ['138 - 143' '133 - 138' '123 - 128' '143 - 148' '148 - 153' '<
118'
 '128 - 133' '118 - 123' '≥ 158']
sc -> ['< 3.65' '3.65 - 6.8' '16.25 - 19.4' '6.8 - 9.95' '13.1 - 16.2
5'
 '9.95 - 13.1' '≥ 28.85']
pot -> ['< 7.31' '≥ 42.59' '7.31 - 11.72' '38.18 - 42.59']
hemo -> ['11.3 - 12.6' '8.7 - 10' '13.9 - 15.2' '≥ 16.5' '10 - 11.3'
'7.4 - 8.7'
 '12.6 - 13.9' '15.2 - 16.5' '< 6.1' '6.1 - 7.4']
pcv -> ['33.5 - 37.4' '29.6 - 33.5' '41.3 - 45.2' '37.4 - 41.3' '≥ 49.
1'
 '21.8 - 25.7' '45.2 - 49.1' '< 17.9' '25.7 - 29.6' '17.9 - 21.8']
rbcc -> ['4.46 - 5.05' '5.05 - 5.64' '3.28 - 3.87' '3.87 - 4.46' '6.23
- 6.82'
 '5.64 - 6.23' '2.69 - 3.28' '< 2.69' '≥ 7.41']
wbcc -> ['7360 - 9740' '12120 - 14500' '14500 - 16880' '4980 - 7360'
'< 4980'
 '9740 - 12120' '16880 - 19260' '≥ 24020' '19260 - 21640']
htn -> ['0' '1']
dm -> ['0' '1']
cad -> ['0' '1']
appet -> ['0' '1']
pe -> ['0' '1']
ane -> ['0' '1']
grf -> ['≥ 227.944' '127.281 - 152.446' '102.115 - 127.281' '177.612 -
202.778'
 '26.6175 - 51.7832' '51.7832 - 76.949' '76.949 - 102.115'
 '152.446 - 177.612' '202.778 - 227.944' '< 26.6175']
stage -> ['s1' 's4' 's3' 's2' 's5']
affected -> ['1' '0']
age -> ['< 12' '12 - 20' '20 - 27' '27 - 35' '35 - 43' '43 - 51' '51 -
59'
 '59 - 66' '66 - 74' '≥ 74']
```

We can observe that columns sg,al,al,su,bgr,bu,sod,sc,pot,hemo,pcv,rbcc,wbcc,grf,age needs to be encoded.

We created a map for each column in accordance to the numerical bin it represents to preserve the relationship between attributes

In [36]:

```
#Apply ordinal encoding to all columns contains numerical bins. For instance, in the
#for “<12”, 1 for “12-20”, 2 for “20-27”, etc.
encoding_map = {"age" : {'< 12':0, '12 - 20':1, '20 - 27':2, '27 - 35':3, '35 - 43':
       '51 - 59':6, '59 - 66':7, '66 - 74':8, '≥ 74':9},
               "grf" : {'≥ 227.944':9 ,'127.281 - 152.446':5 ,'102.115 - 127.281':4
:7 ,'26.6175 - 51.7832':1, '51.7832 - 76.949':2, '76.949 - 102.115':3,
 '152.446 - 177.612':6,'202.778 - 227.944':8, '< 26.6175':0},
               'wbcc':{'7360 - 9740':2, '12120 - 14500':4, '14500 - 16880':5 ,'4980
 '9740 - 12120':3, '16880 - 19260':6, '≥ 24020':8 ,'19260 - 21640':7},
               'rbcc':{'4.46 - 5.05':4, '5.05 - 5.64':5, '3.28 - 3.87':2, '3.87 - 4.
 '5.64 - 6.23':6 ,'2.69 - 3.28':1 ,'< 2.69':0 ,'≥ 7.41':8},
               'bgr' : {'< 112' : 0, '112 - 154' : 1, '154 - 196' : 2, '406 - 448' :
'bu' : {'< 48.1' : 0, '48.1 - 86.2' : 1, '200.5 - 238.6' : 5, '124.3 - 162.4' : 3, '
'sod' : {'138 - 143' : 5, '133 - 138' : 4, '123 - 128' : 2, '143 - 148' : 6, '148 -
'sc' : {'< 3.65' : 0, '3.65 - 6.8' : 1, '16.25 - 19.4' : 5, '6.8 - 9.95' : 2, '13.1
'pot' : {'< 7.31' : 0, '≥ 42.59' : 3, '7.31 - 11.72' : 1 , '38.18 - 42.59' : 2},
'hemo' : {'11.3 - 12.6' : 5, '8.7 - 10' : 3, '13.9 - 15.2' : 7, '≥ 16.5' : 9, '10 -
'pcv' : {'33.5 - 37.4' : 5, '29.6 - 33.5' : 4, '41.3 - 45.2' : 7, '37.4 - 41.3' : 6,
'su' : {'< 0' : 0, '4 - 4' : 4, '2 - 2' : 2, '3 - 4' : 3, '1 - 2' : 1, '≥ 4' : 5},
'sg' : {'1.019 - 1.021' : 3, '1.009 - 1.011' : 1, '1.015 - 1.017' : 2, '≥ 1.023' : 4
'al' : {'1 - 1' : 1, '< 0' : 0, '≥ 4' : 4, '3 - 3' : 3, '2 - 2' : 2}}
encoded_df = df.replace(encoding_map)
#running unique values for dataframe again.
for names in encoded_df.columns: #print all unique values of each row to verify enco
    print(names, end = ' -> ')
    print(encoded_df[names].unique())# note that all numerical bins are encoded
```

```
bp (Diastolic) -> ['0' '1']
bp limit -> ['0' '1' '2']
sg -> [3 1 2 4 0]
al -> [1 0 4 3 2]
class -> ['ckd' 'notckd']
rbc -> ['0' '1']
su -> [0 4 2 3 1 5]
pc -> ['0' '1']
pcc -> ['0' '1']
ba -> ['0' '1']
bgr -> [0 1 2 8 4 3 9 5 7 6]
bu -> [0 1 5 3 2 4 7 6]
sod -> [5 4 2 6 7 0 3 1 8]
sc -> [0 1 5 2 4 3 6]
pot -> [0 3 1 2]
hemo -> [5 3 7 9 4 2 6 8 0 1]
pcv -> [5 4 7 6 9 2 8 0 3 1]
rbcc -> [4 5 2 3 7 6 1 0 8]
wbcc -> [2 4 5 1 0 3 6 8 7]
htn -> ['0' '1']
dm -> ['0' '1']
cad -> ['0' '1']
appet -> ['0' '1']
pe -> ['0' '1']
ane -> ['0' '1']
grf -> [9 5 4 7 1 2 3 6 8 0]
stage -> ['s1' 's4' 's3' 's2' 's5']
affected -> ['1' '0']
age -> [0 1 2 3 4 5 6 7 8 9]
```

Task 1 (3)

Apply one-hot-encoding to the “stage” column.

In [37]:

```
#first view unique stages
encoded_df["stage"].unique()
```

Out[37]:

```
array(['s1', 's4', 's3', 's2', 's5'], dtype=object)
```

We can observe that the only possible values for stage is s1,s2,s3 and s4

In [38]:

```
#using pd.get_dummies to one hot encode the stage column
dummy = pd.get_dummies(encoded_df.stage,prefix='Stage')
dummy.head()
```

Out[38]:

	Stage_s1	Stage_s2	Stage_s3	Stage_s4	Stage_s5
2	1	0	0	0	0
3	1	0	0	0	0
4	1	0	0	0	0
5	1	0	0	0	0
6	1	0	0	0	0

Creating a dummy table for one hot encoding

In [39]:

```
#join dummy dataframe to original data frame
new_df = encoded_df.join(dummy)
new_df
```

Out[39]:

	bp (Diastolic)	bp limit	sg	al	class	rbc	su	pc	pcc	ba	...	ane	grf	stage	affected	age
2	0	0	3	1	ckd	0	0	0	0	0	...	0	9	s1	1	0
3	0	0	1	0	ckd	0	0	0	0	0	...	0	9	s1	1	0
4	0	0	1	4	ckd	1	0	1	0	1	...	0	5	s1	1	0
5	1	1	1	3	ckd	0	0	0	0	0	...	0	5	s1	1	0
6	0	0	2	0	ckd	0	0	0	0	0	...	0	5	s1	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
197	1	2	3	0	ckd	0	0	0	0	0	...	1	1	s3	1	9
198	0	0	3	0	ckd	0	0	0	0	0	...	1	0	s4	1	9
199	1	1	4	0	notckd	0	0	0	0	0	...	0	2	s2	0	9
200	1	1	4	0	notckd	0	0	0	0	0	...	0	4	s1	0	9
201	1	1	1	2	ckd	0	2	0	0	0	...	0	0	s4	1	9

199 rows × 34 columns

We can observe that the dummy table is now appended to the end of the data frame

In [40]:

```
new_df = new_df.drop(columns="stage") #dropping original stage column
```

Dropping the stage column from the dataframe as the information has been encoded in the new columns

Task 1 (4)

Find out and rank the correlations between the "class" column and all other columns

In [41]:

```
#Get information of all the data types of the dataset
new_df.info()
#some binary values are still object data types as they are wrapped in '0' and '1'
#class column is also in ckd and notckd which needs to be encoded to 0 and 1
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 199 entries, 2 to 201
Data columns (total 33 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   bp (Diastolic)  199 non-null    object
 1   bp limit        199 non-null    object
 2   sg              199 non-null    int64 
 3   al              199 non-null    int64 
 4   class           199 non-null    object
 5   rbc             199 non-null    object
 6   su              199 non-null    int64 
 7   pc              199 non-null    object
 8   pcc             199 non-null    object
 9   ba              199 non-null    object
 10  bgr             199 non-null    int64 
 11  bu              199 non-null    int64 
 12  sod             199 non-null    int64 
 13  sc              199 non-null    int64 
 14  pot             199 non-null    int64 
 15  hemo            199 non-null    int64 
 16  pcv             199 non-null    int64 
 17  rbcc            199 non-null    int64 
 18  wbcc            199 non-null    int64 
 19  htn             199 non-null    object
 20  dm              199 non-null    object
 21  cad             199 non-null    object
 22  appet           199 non-null    object
 23  pe              199 non-null    object
 24  ane             199 non-null    object
 25  grf             199 non-null    int64 
 26  affected        199 non-null    object
 27  age             199 non-null    int64 
 28  Stage_s1        199 non-null    uint8 
 29  Stage_s2        199 non-null    uint8 
 30  Stage_s3        199 non-null    uint8 
 31  Stage_s4        199 non-null    uint8 
 32  Stage_s5        199 non-null    uint8 
dtypes: int64(14), object(14), uint8(5)
memory usage: 54.2+ KB
```

As you can observe, the dataframe still consists of not numerical attributes in the objecct data type columms, therefore we need to convert the str values representing integers into true integers

In [42]:

```
encode_class = {"class":{'ckd':1,'notckd':0}}
new_df = new_df.replace(encode_class)#change ckd and notckd to 1 and 0 respectively
new_df
```

Out[42]:

	bp (Diastolic)	bp limit	sg	al	class	rbc	su	pc	pcc	ba	...	pe	ane	grf	affected	age	Sta
2	0	0	3	1	1	0	0	0	0	0	...	0	0	9	1	0	
3	0	0	1	0	1	0	0	0	0	0	...	0	0	9	1	0	
4	0	0	1	4	1	1	0	1	0	1	...	0	0	5	1	0	
5	1	1	1	3	1	0	0	0	0	0	...	0	0	5	1	0	
6	0	0	2	0	1	0	0	0	0	0	...	1	0	5	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
197	1	2	3	0	1	0	0	0	0	0	...	0	1	1	1	9	
198	0	0	3	0	1	0	0	0	0	0	...	0	1	0	1	9	
199	1	1	4	0	0	0	0	0	0	0	...	0	0	2	0	9	
200	1	1	4	0	0	0	0	0	0	0	...	0	0	4	0	9	
201	1	1	1	2	1	0	2	0	0	0	...	0	0	0	1	9	

199 rows × 33 columns

Converting the class attribute to 1 or 0

In [43]:

```
new_df = new_df.astype(int)
new_df.info()#converting all strings numericals to string
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 199 entries, 2 to 201
Data columns (total 33 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   bp (Diastolic)  199 non-null    int64
 1   bp limit        199 non-null    int64
 2   sg              199 non-null    int64
 3   al              199 non-null    int64
 4   class           199 non-null    int64
 5   rbc             199 non-null    int64
 6   su              199 non-null    int64
 7   pc              199 non-null    int64
 8   pcc             199 non-null    int64
 9   ba              199 non-null    int64
 10  bgr             199 non-null    int64
 11  bu              199 non-null    int64
 12  sod             199 non-null    int64
 13  sc              199 non-null    int64
 14  pot             199 non-null    int64
 15  hemo            199 non-null    int64
 16  pcv             199 non-null    int64
 17  rbcc            199 non-null    int64
 18  wbcc            199 non-null    int64
 19  htn             199 non-null    int64
 20  dm              199 non-null    int64
 21  cad             199 non-null    int64
 22  appet           199 non-null    int64
 23  pe              199 non-null    int64
 24  ane             199 non-null    int64
 25  grf             199 non-null    int64
 26  affected        199 non-null    int64
 27  age             199 non-null    int64
 28  Stage_s1        199 non-null    int64
 29  Stage_s2        199 non-null    int64
 30  Stage_s3        199 non-null    int64
 31  Stage_s4        199 non-null    int64
 32  Stage_s5        199 non-null    int64
dtypes: int64(33)
memory usage: 61.0 KB
```

Converting all strings to ints

In [44]:

```
cormat = new_df.corr()
cormat = cormat.abs()
cormat['class'].sort_values(ascending=False)
#sort the class correlation matrix after transforming p value to absolute value
#ranking correlation by ascending order
```

Out[44]:

```
affected          1.000000
class             1.000000
hemo              0.765089
pcv               0.699531
sg                0.657702
grf               0.637083
rbcc              0.606591
htn               0.604532
Stage_s1          0.598828
al                0.558429
dm                0.548551
sod               0.495498
bu                0.424668
pc                0.407015
appet             0.377656
bgr               0.366404
pe                0.347838
Stage_s5          0.341792
ane               0.329595
Stage_s3          0.323438
Stage_s4          0.307020
sc                0.301996
pcc               0.298320
su                0.286902
rbc               0.285404
Stage_s2          0.283929
bp limit          0.273247
cad               0.265454
wbcc              0.235190
age               0.232851
ba                0.182130
bp (Diastolic)    0.098866
pot               0.086149
Name: class, dtype: float64
```

As we can observe from the correlation matrix, affected is exactly correlated to the class column with a p value of 1. Other notable values such as hemo, pcv, sg, grf and rbcc have medium to strong correlation to the class attribute, hinting that those features are good indicators of predicting Chronic Kidney Disease