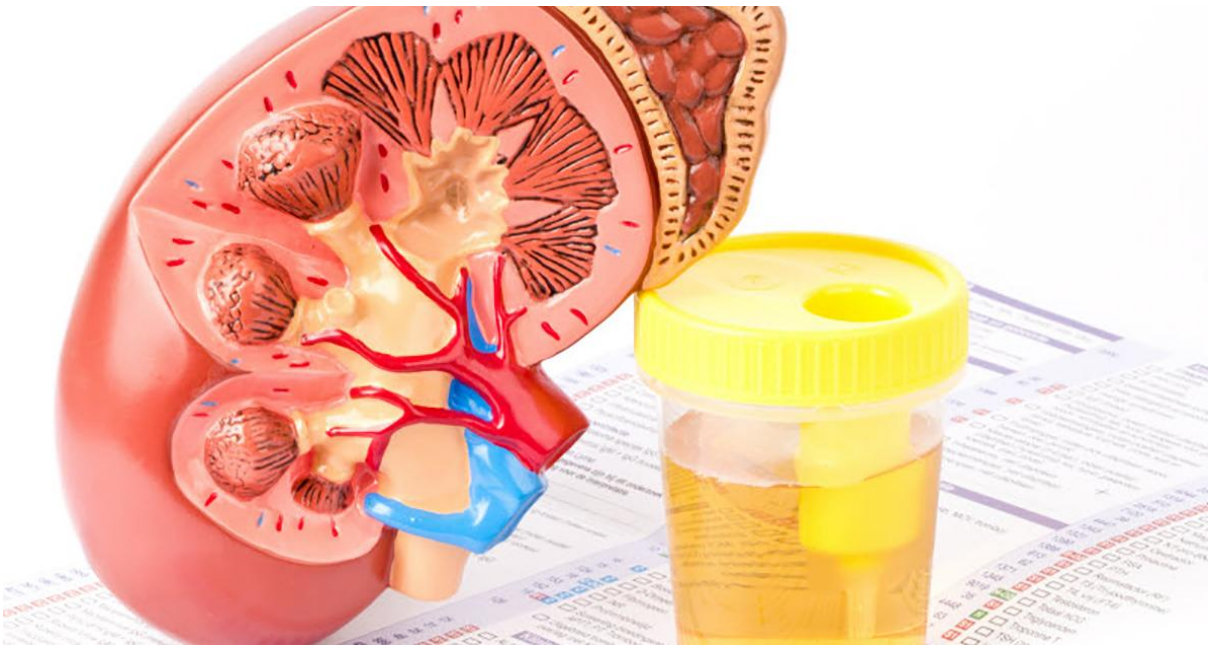


health care kidney chronic

Topics that will be covered are:

1. Data preprocessing
2. Exploratory data analysis
3. Model building
4. Saving the model

Let’s get started!

Importing necessary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV, train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
import scipy.stats as stats
import seaborn as sns

%matplotlib inline
```

read our dataset

```
df = pd.read_csv('kidney.csv')
data = df
data.head()
```

Output:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	...	44.0	7800.0	5.2	yes	yes	no	good	no	no	ckd
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	...	38.0	6000.0	NaN	no	no	no	good	no	no	ckd
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	...	31.0	7500.0	NaN	no	yes	no	poor	no	yes	ckd
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	...	32.0	6700.0	3.9	yes	no	no	poor	yes	yes	ckd
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	...	35.0	7300.0	4.6	no	no	no	good	no	no	ckd

shape of our dataset

```
data.shape
```

Output:

```
(400, 25)
```

Data Preprocessing

```
df.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   age     391 non-null    float64
 1   bp      388 non-null    float64
 2   sg      353 non-null    float64
 3   al      354 non-null    float64
 4   su      351 non-null    float64
 5   rbc     248 non-null    object 
 6   pc      335 non-null    object 
 7   pcc     396 non-null    object 
 8   ba      396 non-null    object 
 9   bgr     356 non-null    float64
 10  bu      381 non-null    float64
 11  sc      383 non-null    float64
 12  sod     313 non-null    float64
 13  pot     312 non-null    float64
 14  hemo    348 non-null    float64
 15  pcv     329 non-null    float64
 16  wbcc    294 non-null    float64
 17  rbcc    269 non-null    float64
 18  htn     398 non-null    object 
 19  dm      398 non-null    object 
 20  cad     398 non-null    object 
 21  appet   399 non-null    object 
 22  pe      399 non-null    object 
 23  ane     399 non-null    object 
 24  class   400 non-null    object 
dtypes: float64(14), object(11)
memory usage: 78.2+ KB
```

our data statistically

```
df.describe()
```

Output:

	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo	pcv
count	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000	313.000000	312.000000	348.000000	329.000000
mean	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454	137.528754	4.627244	12.526437	38.884498
std	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.741126	10.408752	3.193904	2.912587	8.990105
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000	4.500000	2.500000	3.100000	9.000000
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.900000	135.000000	3.800000	10.300000	32.000000
50%	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.300000	138.000000	4.400000	12.650000	40.000000
75%	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.800000	142.000000	4.900000	15.000000	45.000000
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000	163.000000	47.000000	17.800000	54.000000

total count of null values that every feature holds

```
df.isna().sum()
```

Output:

```
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
```

Correlation matrix & Matrix Visualisation

```
df.corr()
```

Output:

	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo	pcv	wbcc	rbcc
age	1.000000	0.159480	-0.191096	0.122091	0.220866	0.244992	0.196985	0.132531	-0.100046	0.058377	-0.192928	-0.242119	0.118339	-0.268896
bp	0.159480	1.000000	-0.218836	0.160689	0.222576	0.160193	0.188517	0.146222	-0.116422	0.075151	-0.306540	-0.326319	0.029753	-0.261936
sg	-0.191096	-0.218836	1.000000	-0.469760	-0.296234	-0.374710	-0.314295	-0.361473	0.412190	-0.072787	0.602582	0.603560	-0.236215	0.579476
al	0.122091	0.160689	-0.469760	1.000000	0.269305	0.379464	0.453528	0.399198	-0.459896	0.129038	-0.634632	-0.611891	0.231989	-0.566437
su	0.220866	0.222576	-0.296234	0.269305	1.000000	0.717827	0.168583	0.223244	-0.131776	0.219450	-0.224775	-0.239189	0.184893	-0.237448
bgr	0.244992	0.160193	-0.374710	0.379464	0.717827	1.000000	0.143322	0.114875	-0.267848	0.066966	-0.306189	-0.301385	0.150015	-0.281541
bu	0.196985	0.188517	-0.314295	0.453528	0.168583	0.143322	1.000000	0.586368	-0.323054	0.357049	-0.610360	-0.607621	0.050462	-0.579087
sc	0.132531	0.146222	-0.361473	0.399198	0.223244	0.114875	0.586368	1.000000	-0.690158	0.326107	-0.401670	-0.404193	-0.006390	-0.400852
sod	-0.100046	-0.116422	0.412190	-0.459896	-0.131776	-0.267848	-0.323054	-0.690158	1.000000	0.097887	0.365183	0.376914	0.007277	0.344873
pot	0.058377	0.075151	-0.072787	0.129038	0.219450	0.066966	0.357049	0.326107	0.097887	1.000000	-0.133746	-0.163182	-0.105576	-0.158309
hemo	-0.192928	-0.306540	0.602582	-0.634632	-0.224775	-0.306189	-0.610360	-0.401670	0.365183	-0.133746	1.000000	0.895382	-0.169413	0.798880
pcv	-0.242119	-0.326319	0.603560	-0.611891	-0.239189	-0.301385	-0.607621	-0.404193	0.376914	-0.163182	0.895382	1.000000	-0.197022	0.791625
wbcc	0.118339	0.029753	-0.236215	0.231989	0.184893	0.150015	0.050462	-0.006390	0.007277	-0.105576	-0.169413	-0.197022	1.000000	-0.158163
rbcc	-0.268896	-0.261936	0.579476	-0.566437	-0.237448	-0.281541	-0.579087	-0.400852	0.344873	-0.158309	0.798880	0.791625	-0.158163	1.000000

find out how many of each class are:

```
df['class'].value_counts()
```

Output:

```
ckd       250
notckd    150
Name: class, dtype: int64
```

Representation of Target variable in Percentage

```
countNoDisease = len(df[df['class'] == 0])
countHaveDisease = len(df[df['class'] == 1])
print("Percentage of Patients Haven't Heart Disease:
{:.2f}%".format((countNoDisease /
(len(df['class']))*100)))
print("Percentage of Patients Have Heart Disease:
{:.2f}%".format((countHaveDisease /
(len(df['class']))*100)))
```

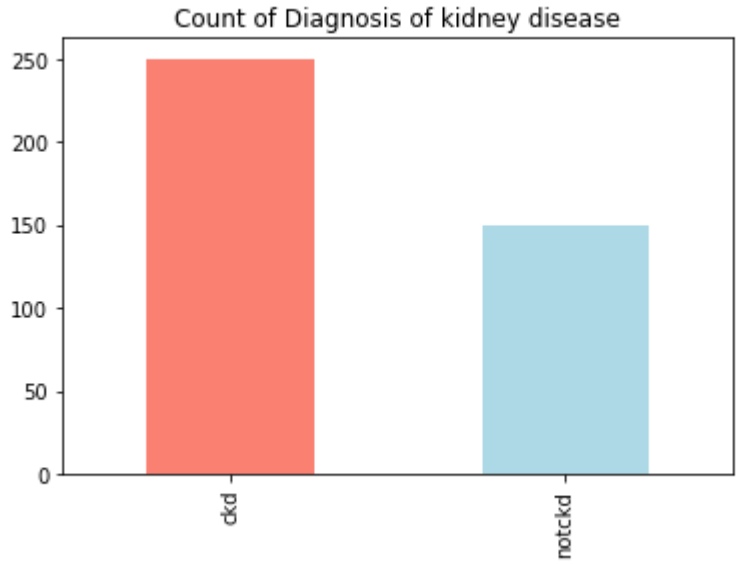
Output:

```
Percentage of Patients Haven't Kidney Disease: 0.00%
Percentage of Patients Have Kidney Disease: 0.00%
```

Understanding the balancing of the data visually

```
df['class'].value_counts().plot(kind='bar',color=['salm
on','lightblue'],title="Count of Diagnosis of kidney
disease")
```

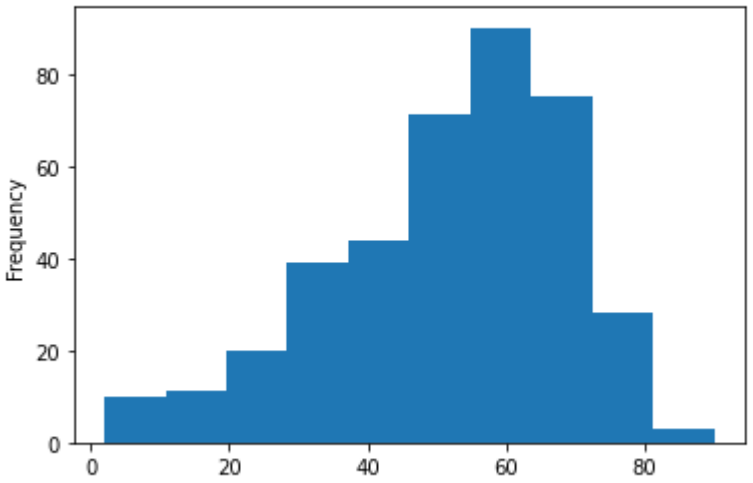
Output:



checking the distribution of the age column

```
df['age'].plot(kind='hist')
```

Output:

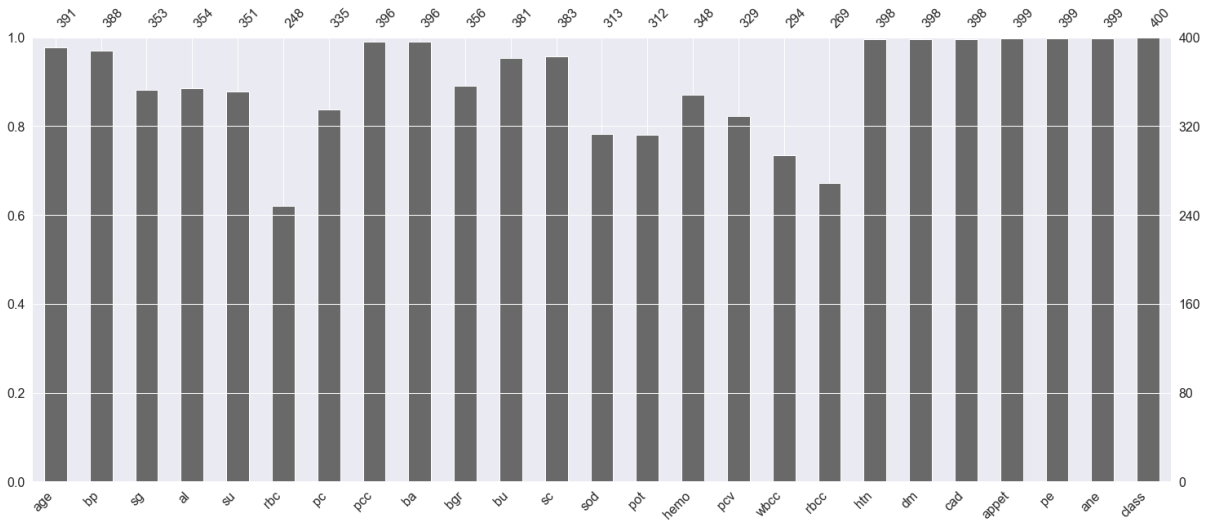


Inference: Herewith the help of histogram we can see that 50-60 age group people are widely spread in this dataset.

Here we are plotting the graph to see the null values in the dataset.

```
p = msno.bar(data)
```

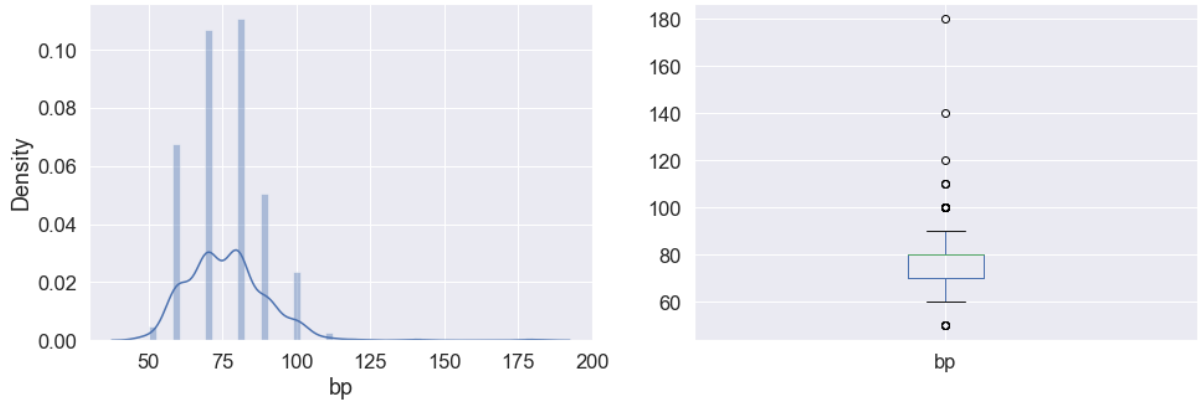
Output:



Inference: Here any features which have not touched the 400 mark at the top are having null values.

```
plt.subplot(121), sns.distplot(data['bp'])
plt.subplot(122), data['bp'].plot.box(figsize=(16,5))
plt.show()
```

Output:



Inference: Here in the above graph we can see the distribution of blood pressure and also in the subplot it is visible that the bp column has some outliers in it.

Now we will convert the categorical values(object) to categorical values(int)

```
data['class'] = data['class'].map({'ckd':1,'notckd':0})
data['htn'] = data['htn'].map({'yes':1,'no':0})
data['dm'] = data['dm'].map({'yes':1,'no':0})
data['cad'] = data['cad'].map({'yes':1,'no':0})
data['appet'] = data['appet'].map({'good':1,'poor':0})
data['ane'] = data['ane'].map({'yes':1,'no':0})
data['pe'] = data['pe'].map({'yes':1,'no':0})
data['ba'] =
data['ba'].map({'present':1,'notpresent':0})
data['pcc'] =
data['pcc'].map({'present':1,'notpresent':0})
data['pc'] = data['pc'].map({'abnormal':1,'normal':0})
data['rbc'] =
data['rbc'].map({'abnormal':1,'normal':0})
data['class'].value_counts()
```

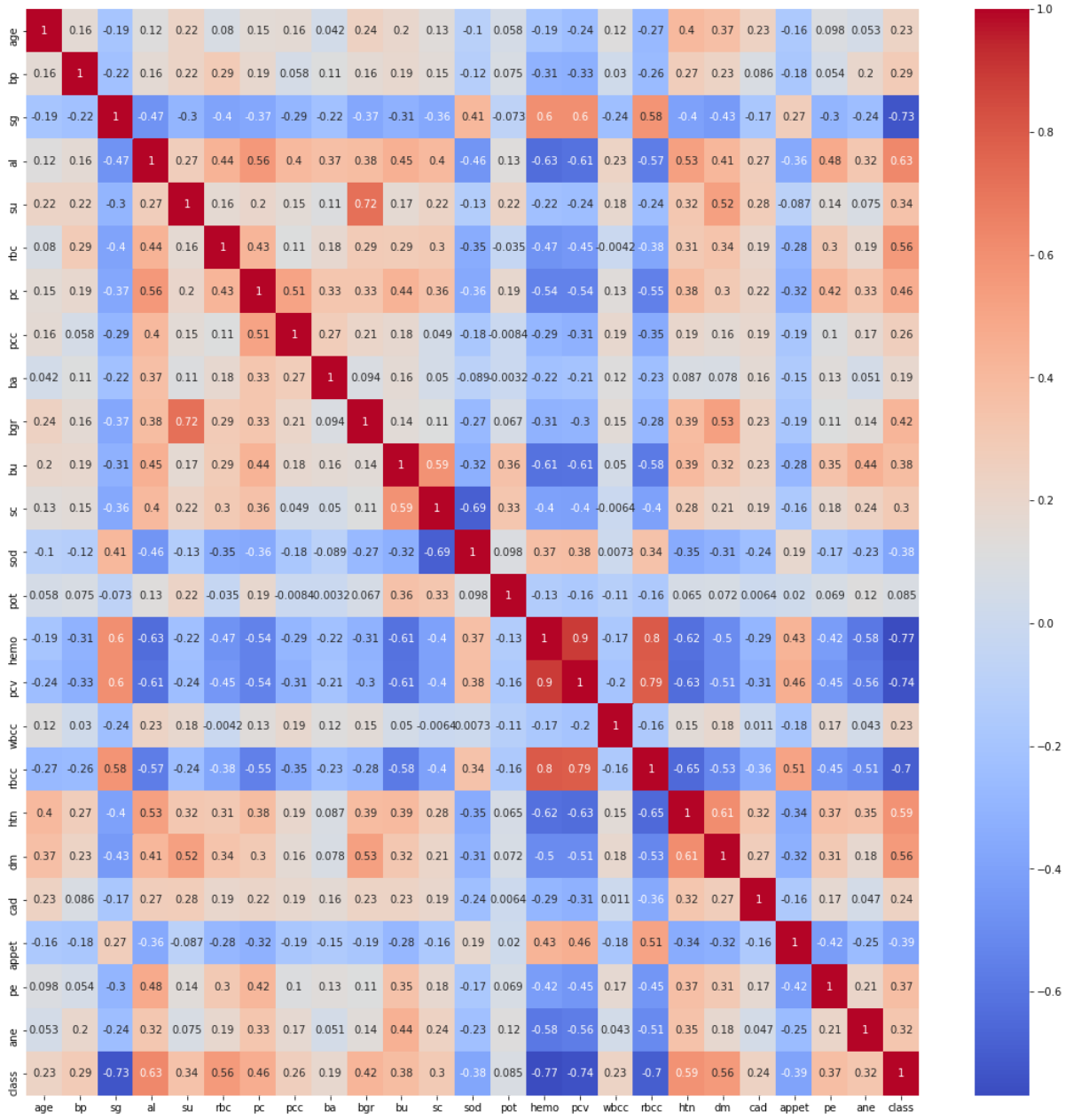
Output:

```
1    250
0    150
Name: class, dtype: int64
```

Finding the Correlation between the plots

```
plt.figure(figsize = (19,19))
sns.heatmap(data.corr(), annot = True, cmap =
'coolwarm') # looking for strong correlations with
"class" row
```

Output:



Exploratory data analysis (EDA)

Let’s see the shape of the dataset again after analysis

```
data.shape
```

Output:

```
(400, 25)
```

Now let’s see the final columns present in our dataset.

```
data.columns
```

Output:

```
Index(['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc',
'pcc', 'ba', 'bgr', 'bu',
       'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbcc',
'rbcc', 'htn', 'dm', 'cad',
       'appet', 'pe', 'ane', 'class'],
      dtype='object')
```

Now dropping the null values.

```
data.shape[0], data.dropna().shape[0]
```

Output:

```
(400, 158)
```

Here from the above output, we can see that there are 158 null values in the dataset. Now here we are left with two choices that we could either drop all the null values or keep them when we will drop that NA values so we should understand that our dataset is not that large and if we drop those null values then it would be even smaller in that case if we provide very fewer data to our machine learning model then the performance would be very less also we yet don't know that these null values are related to some other features in the dataset.

So for this time I'll keep these values and see how the model will perform in this dataset.

Also when we are working on some healthcare project where we will be predicting whether the person is suffering from that disease or not then one thing we should keep in my mind is that the model evaluation should have the least false positive errors.

```
data.dropna(inplace=True)
data.shape
```

Output:

```
(158, 25)
```

Model Building

1. Logistic regression

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()

X = data.iloc[:,:-1]
y = data['class']

X_train, X_test, y_train, y_test =
train_test_split(X,y, stratify = y, shuffle = True)

logreg.fit(X_train,y_train)
```

Output:

```
LogisticRegression()
```

Training score

```
logreg.score(X_train,y_train)
```

Output:

```
1.0
```

Testing accuracy

```
logreg.score(X_test,y_test)
```

Output:

```
0.975
```

Printing the training and testing accuracy

```
from sklearn.metrics import accuracy_score,
confusion_matrix

print('Train Accuracy: ', accuracy_score(y_train,
train_pred))
print('Test Accuracy: ', accuracy_score(y_test,
test_pred))
```

Output:

```
Train Accuracy:  1.0
Test Accuracy:  0.975
```

The cell below shows the coefficients for each variable.

(example on reading the coefficients from a Logistic Regression: a one-unit increase in age makes an individual about e^0.14 time as likely to have CKD, while a one-unit increase in blood pressure makes an individual about e^-0.07 times as likely to have CKD.

```
pd.DataFrame(logreg.coef_, columns=X.columns)
```

Output:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	hemo	pcv	wbcc	rbcc
0	0.28582	-0.132118	0.002671	0.309953	0.010789	0.019856	0.091069	0.003106	0.006829	0.414045	...	-0.282868	-0.62111	0.001157	-0.141783

1 rows × 24 columns

Confusion Matrix

```
sns.set(font_scale=1.5)

def plot_conf_mat(y_test,y_preds):
    """
    This function will be heloing in plotting the
confusion matrix by using seaborn
```

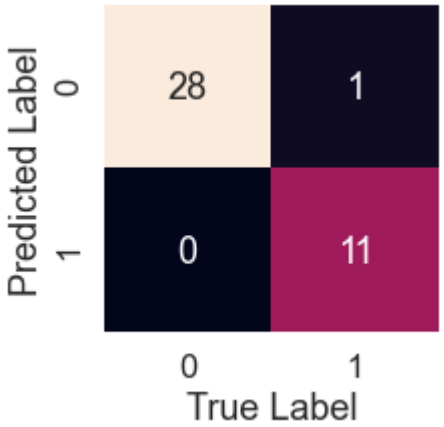
```
    """

    fig,ax=plt.subplots(figsize=(3,3))

ax=sns.heatmap(confusion_matrix(y_test,y_preds),annot=T
rue,cbar=False)
    plt.xlabel("True Label")
    plt.ylabel("Predicted Label")


log_pred = logreg.predict(X_test)
plot_conf_mat(y_test, log_pred)
```

Output:




```
tn, fp, fn, tp = confusion_matrix(y_test,
test_pred).ravel()

print(f'True Neg: {tn}')
print(f'False Pos: {fp}')
print(f'False Neg: {fn}')
print(f'True Pos: {tp}')
```

Output:

```
True Neg: 28
False Pos: 1
False Neg: 0
True Pos: 11
```

K-Nearest Neighbors Classifier

It is a good practice to first balance the class well before using the KNN, as we know that in the case of unbalanced classes KNN doesn't perform well.

```
df["class"].value_counts()
```

Output:

```
0    115
1     43
Name: class, dtype: int64
```

Now let's CONCATENATE our class variables together.

```
balanced_df = pd.concat([df[df["class"] == 0],
df[df["class"] == 1].sample(n = 115, replace = True)],
axis = 0)
balanced_df.reset_index(drop=True, inplace=True)
balanced_df["class"].value_counts()
```

Output:

```
1    115
0    115
Name: class, dtype: int64
```

Now let's scale down the data

```
ss = StandardScaler()
ss.fit(X_train)
X_train = ss.transform(X_train)
X_test = ss.transform(X_test)
```

Now we will tune the KNN model for better accuracy

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()

params = {
    "n_neighbors":[3,5,7,9],
    "weights":["uniform","distance"],
    "algorithm":["ball_tree","kd_tree","brute"],
    "leaf_size":[25,30,35],
    "p":[1,2]
}

gs = GridSearchCV(knn, param_grid=params)
model = gs.fit(X_train,y_train)
preds = model.predict(X_test)
accuracy_score(y_test, preds)
```

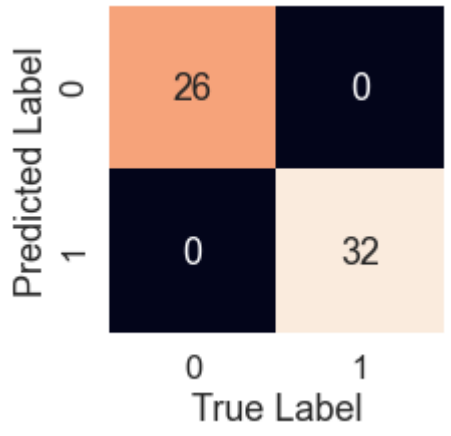
Output:

```
1.0
```

Confusion matrix for KNN model

```
knn_pred = model.predict(X_test)
plot_conf_mat(y_test, knn_pred)
```

Output:



```
tn, fp, fn, tp = confusion_matrix(y_test,
preds).ravel()

print(f'True Neg: {tn}')
print(f'False Pos: {fp}')
print(f'False Neg: {fn}')
print(f'True Pos: {tp}')
```

Output:

```
True Neg: 26
False Pos: 0
False Neg: 0
True Pos: 32
```

Feature Importance

```
feature_dict=dict(zip(df.columns,list(logreg.coef_[0]))
)
feature_dict
```

Here we will get the coefficient from the features which will tell the weightage of each feature.

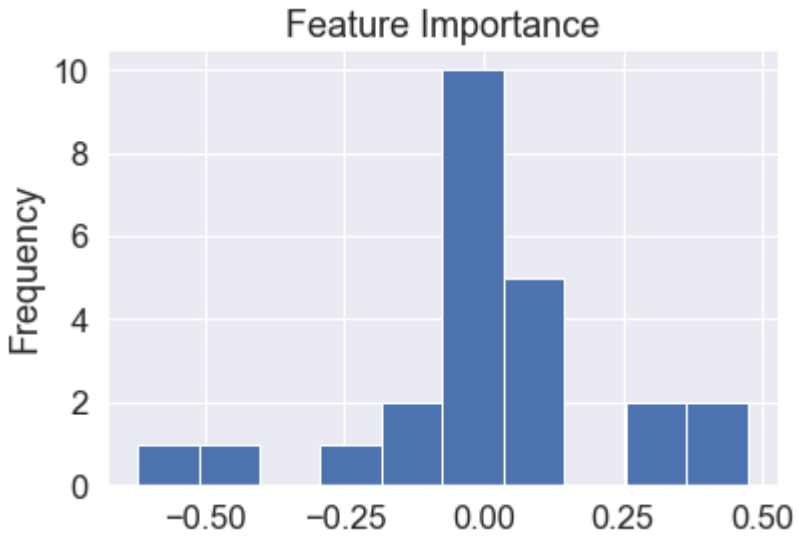
Output:

```
{'age': 0.2858203378727209,
 'bp': -0.13211767022170745,
 'sg': 0.0026714534270341444,
 'al': 0.3099528545093234,
 'su': 0.010788785962584712,
 'rbc': 0.01985635073828642,
 'pc': 0.09106895589320387,
 'pcc': 0.003106393240262293,
 'ba': 0.006828878616469952,
 'bgr': 0.4140451203997997,
 'bu': 0.47262371944289844,
 'sc': 0.12893875993072498,
 'sod': -0.4419699201228987,
 'pot': 0.05909714695858163,
 'hemo': -0.28286805186344094,
 'pcv': -0.6211104727832718,
 'wbcc': 0.001157338688486265,
 'rbcc': -0.1417833283935927,
 'htn': 0.08881269207443204,
 'dm': 0.08689401433102413,
 'cad': 0.0018059681932075433,
 'appet': -0.004481530609769657,
 'pe': 0.0051126943850270425,
 'ane': 0.00349950552414094}
```

Visualize feature importance

```
feature_df=pd.DataFrame(feature_dict,index=[0])
feature_df.T.plot(kind="hist",legend=False,title="Featu
re Importance")
```

Output:

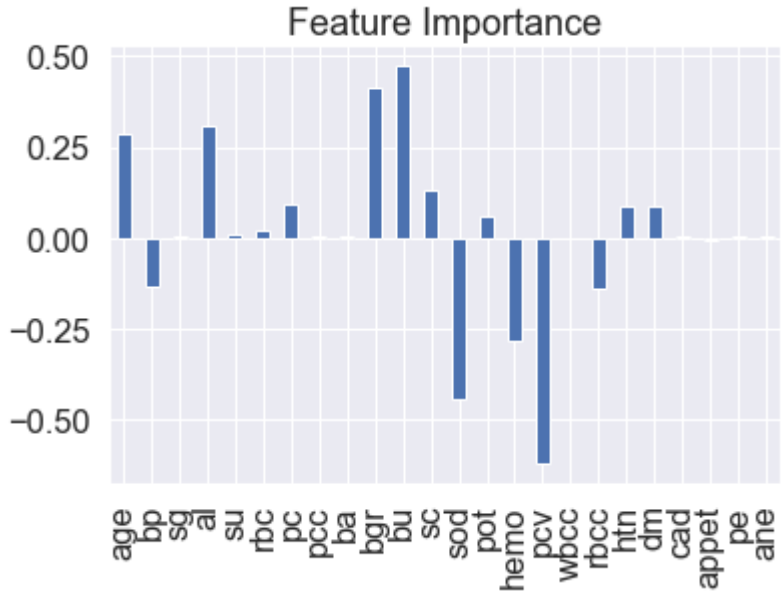


Visualize feature importance – Transpose

```
feature_df=pd.DataFrame(feature_dict,index=[0])
```

```
feature_df.T.plot(kind="bar",legend=False,title="Featur
e Importance")
```

Output:



Saving the model

```
import pickle

# Now with the help of pickle model we will be saving
the trained model
saved_model = pickle.dumps(logreg)

# Load the pickled model
logreg_from_pickle = pickle.loads(saved_model)

# Now here we will load the model
logreg_from_pickle.predict(X_test)
```

Output:

```
array([1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1], dtype=int64)
```