Predicting the chronic kidney disease using machine learning

Kidney Disease Prediction - Problem statement 12 - Group 186

Group No Name Student Email id % Contribution \ 186 Sivarajan N 2021FC04989@wilp.bits-pilani.ac.in Equal (100%) \ 186 Sindhu C 2021FC04993@wilp.bits-pilani.ac.in Equal (100%) \ 186 Manibalan S 2021fc04442@wilp.bits-pilani.ac.in None (0%)

The purpose of this notebook is to get you started to solve this problem.

The main requirements are listed below which follows a standard Data Science Project:

- Presteps

1. Download Dataset from Google Drive - https://drive.google.com/file/d/1NykVFA1f5oGXZ5JlrBGXPBJrnfRncREh/view?usp=sharing
2. Import the required libraries

- I - Data Visualization & Exploration

    1. Print 2 rows for sanity check to identify all the features present in the dataset and if the target matches with them.
    2. Comment on class imbalance with appropriate visualization method.
    3. Provide appropriate visualizations to get an insight about the dataset.
    4. Do the correlational analysis on the dataset. Provide a visualization for the same. Justify the answer by answering this - Will this correlational analysis have effect on feature selection that we will perform in the next step?
- II - Data Preprocessing, Feature Engineering & Cleaning

    1. Do the appropriate pre-processing of the data like identifying NULL or Missing Values if any, handling of outliers if present in the dataset, skewed data etc. Mention the pre-processing steps performed in the markdown cell.
    2. Apply appropriate feature engineering techniques for them. Apply the feature transformation techniques like Standardization, Normalization, etc. Apply the appropriate transformations depending upon the structure and the complexity of the dataset. Provide proper justification.
- III - Model Building

    1. Split the dataset into training and test sets. Justify the choice of split. Experiment with different split to get the final split. Justify the method chosen.
    2. Build Model Development using Decision Tree and KNN. Identify the best parameter

and justify your answer.

- IV - Validation, Performance Evaluation, Testing

1. Do the prediction for the test data and display the results for the inference. Calculate all the evaluation metrics and choose best for the model chosen.
2. Compare the model in tabulated form. [1M] Justify your comment. Answer without justification will not be awarded marks

Takeaways from the blog

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

In [1]:
```
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
import missingno as msno

%matplotlib inline
```

In [2]:
```
df = pd.read_csv('kidney_disease.csv')
data = df
data.head()
```

Out[2]:

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

5 rows × 26 columns

In [3]:
```
df.shape
```

Out[3]:
```
(400, 26)
```

In [4]:
```
data.head()
```

Out[4]:

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

5 rows × 26 columns

Data Preprocessing

In [5]:
```
df.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    float64
 2   bp              388 non-null    float64
 3   sg              353 non-null    float64
 4   al              354 non-null    float64
 5   su              351 non-null    float64
 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    float64
 11  bu              381 non-null    float64
 12  sc              383 non-null    float64
 13  sod             313 non-null    float64
 14  pot             312 non-null    float64
 15  hemo            348 non-null    float64
 16  pcv             330 non-null    object 
 17  wc              295 non-null    object 
 18  rc              270 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  classification  400 non-null    object 
dtypes: float64(11), int64(1), object(14)
memory usage: 81.4+ KB
```

In [6]:
```
df.describe()
```

Out[6]:

	id	age	bp	sg	al	su	bgr	bu

	id	age	bp	sg	al	su	bgr	bu
count	400.000000	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.00000
mean	199.500000	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.42572
std	115.614301	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.50300
min	0.000000	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.50000
25%	99.750000	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.00000
50%	199.500000	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.00000
75%	299.250000	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.00000

In [7]:
```
df.isna().sum()
```

Out[7]:
```
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              70
wc              105
rc              130
htn               2
dm                2
cad               2
appet             1
pe                1
ane               1
classification    0
dtype: int64
```

Correlation matrix & Matrix Visualisation

In [8]:
```
df.corr()
```

Out[8]:

	id	age	bp	sg	al	su	bgr	bu	s
id	1.000000	-0.185308	-0.245744	0.642156	-0.541993	-0.283416	-0.338673	-0.307175	-0.26868
age	-0.185308	1.000000	0.159480	-0.191096	0.122091	0.220866	0.244992	0.196985	0.13253
bp	-0.245744	0.159480	1.000000	-0.218836	0.160689	0.222576	0.160193	0.188517	0.14622

	id	age	bp	sg	al	su	bgr	bu	s
sg	0.642156	-0.191096	-0.218836	1.000000	-0.469760	-0.296234	-0.374710	-0.314295	-0.36147
al	-0.541993	0.122091	0.160689	-0.469760	1.000000	0.269305	0.379464	0.453528	0.39919
su	-0.283416	0.220866	0.222576	-0.296234	0.269305	1.000000	0.717827	0.168583	0.22324
bgr	-0.338673	0.244992	0.160193	-0.374710	0.379464	0.717827	1.000000	0.143322	0.11487
bu	-0.307175	0.196985	0.188517	-0.314295	0.453528	0.168583	0.143322	1.000000	0.58636
sc	-0.268683	0.132531	0.146222	-0.361473	0.399198	0.223244	0.114875	0.586368	1.00000
sod	0.364251	-0.100046	-0.116422	0.412190	-0.459896	-0.131776	-0.267848	-0.323054	-0.69015

In [9]:
```
# Let's find out how many of each class are

df['classification'].value_counts()
# from below output we ca draw our inference that this is close to "imbalanced datase
```

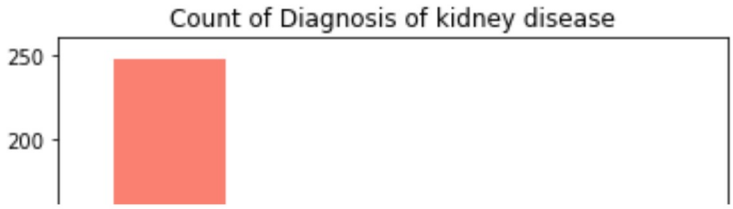
Out[9]:
```
ckd       248
notckd    150
ckd\t       2
Name: classification, dtype: int64
```

In [10]:
```
#Representation of Target variable in Percentage

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

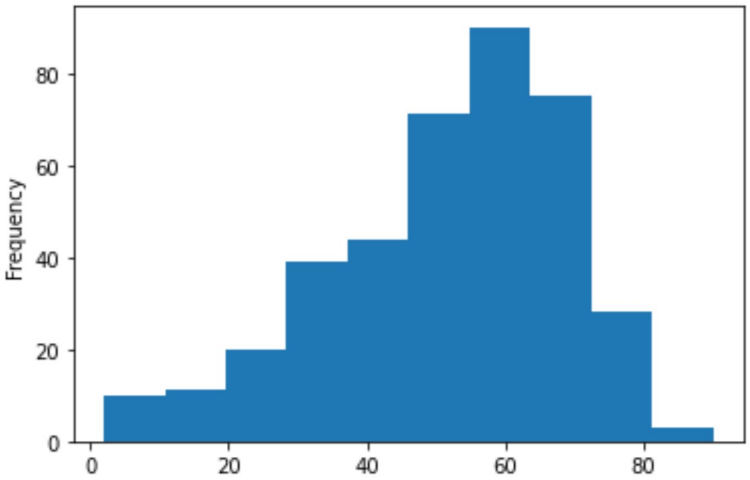
```
Percentage of Patients Haven't Heart Disease: 0.00%
Percentage of Patients Have Heart Disease: 0.00%
```

In [11]:
```
#Understanding the balancing of the data visually

df['classification'].value_counts().plot(kind='bar',color=['salmon','lightblue'],titl
```

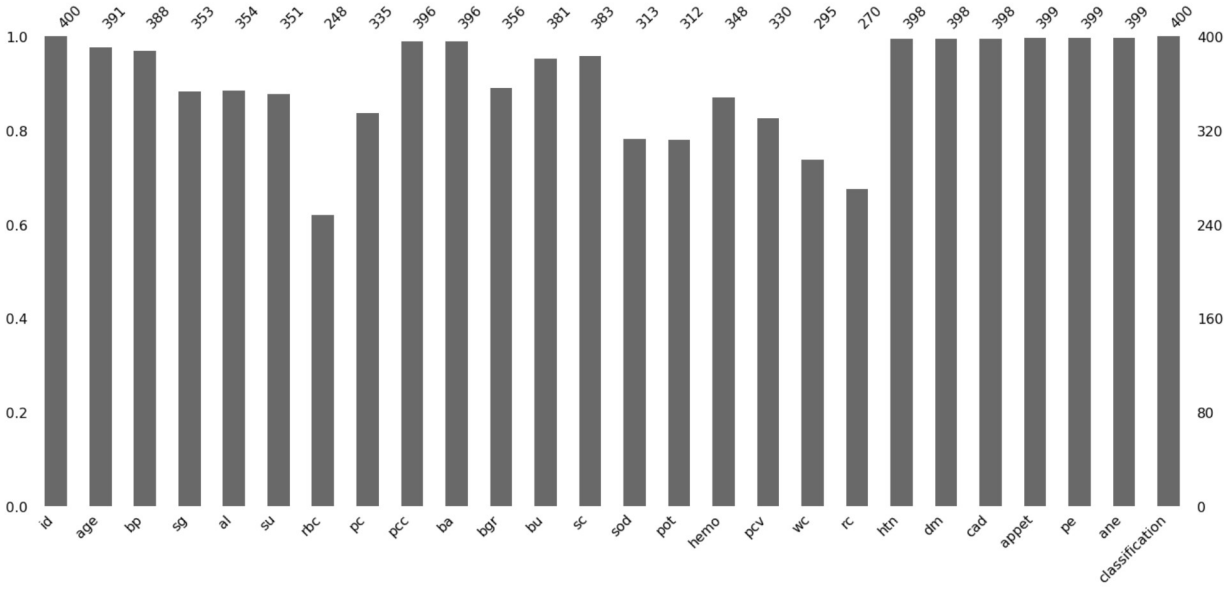
```
C:\Users\ns_ra\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.py:240: Ru
ntimeWarning: Glyph 9 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\ns_ra\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.py:203: Ru
ntimeWarning: Glyph 9 missing from current font.
  font.set_text(s, 0, flags=flags)
```



In [12]:
```
#Check the distribution of the age column with a histogram

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



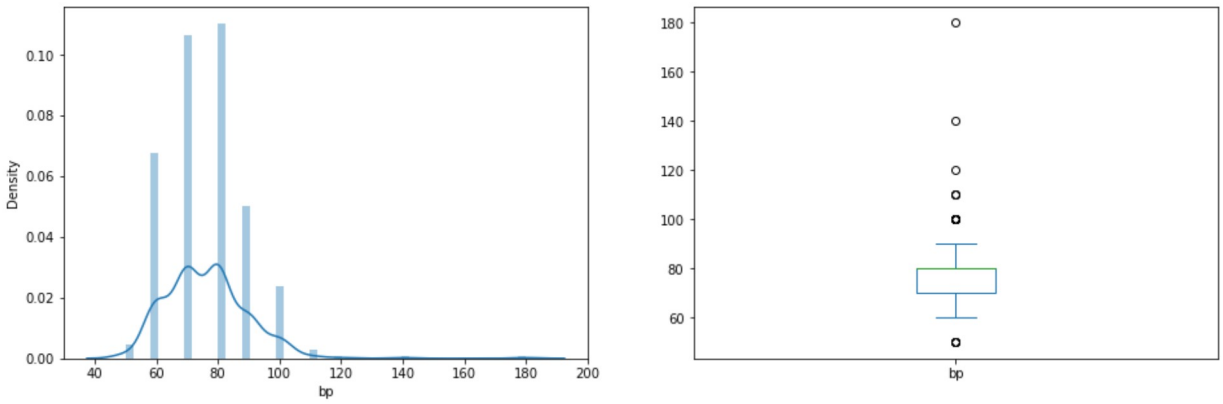
In [13]:
```
# Here we are plotting the graph to see the null values in the datset
p = msno.bar(data)
```



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

In [14]:
```
plt.subplot(121), sns.distplot(data['bp'])
plt.subplot(122), data['bp'].plot.box(figsize=(16,5))
plt.show()
```

```
C:\Users\ns_ra\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. Plea
se adapt your code to use either `displot` (a figure-level function with similar flex
ibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



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

In [15]:
```
data['classification'] = data['classification'].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})
```

In [16]:
```
data['classification'].value_counts()
```

Out[16]:
```
1.0    248
0.0    150
Name: classification, dtype: int64
```

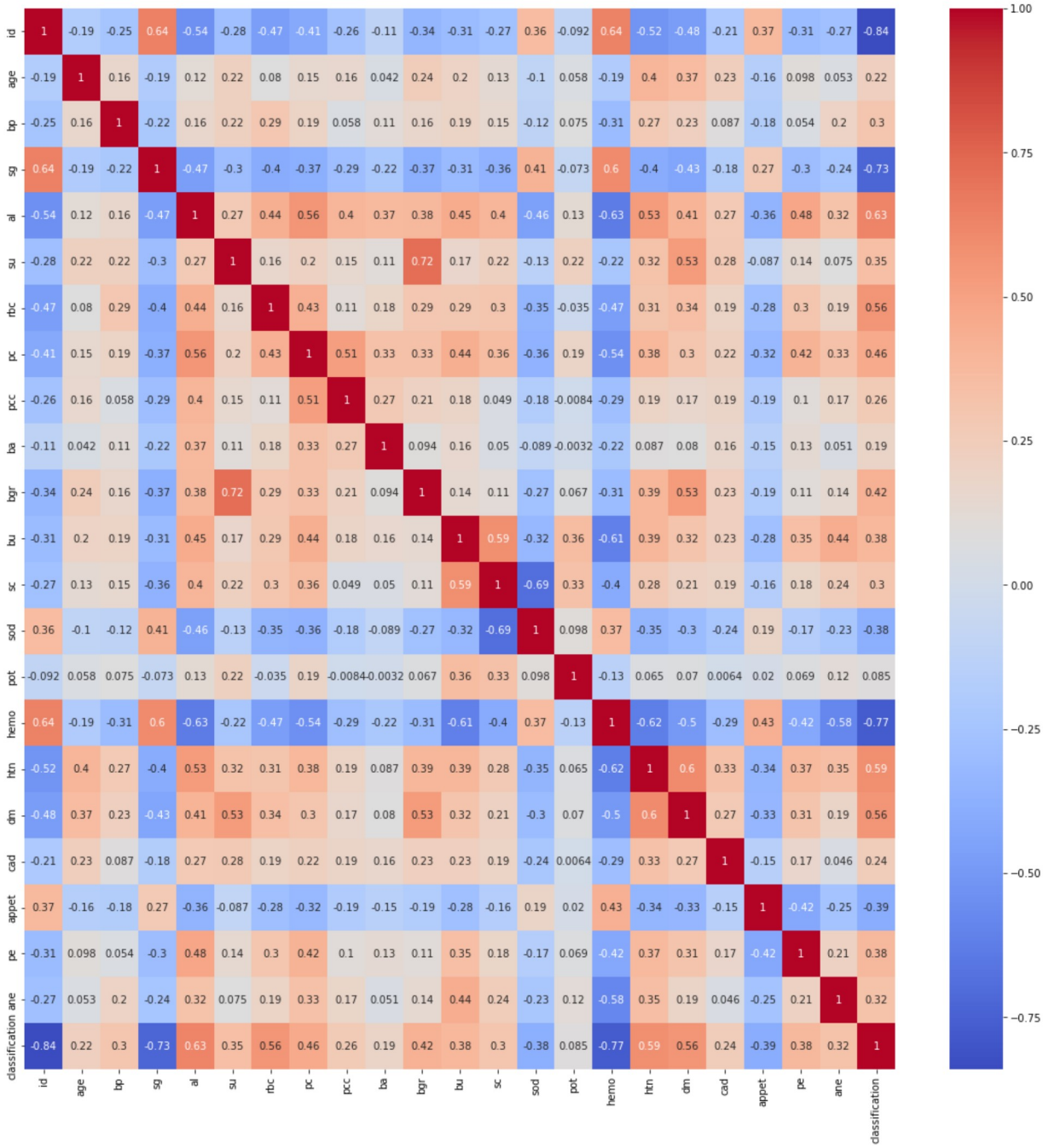
Here I'm mentioning some reasons which will effect you kidney in a drastically bad way.

- Diabetes - blood sugar or diabetes mellitus
- High blood pressure - BP
- Heart and blood vessel (cardiovascular) disease
- Smoking
- Obesity
- Being African-American, Native American or Asian-American
- Family history of kidney disease

- Abnormal kidney structure
- Older age - age

In [17]:
```
plt.figure(figsize = (19,19))
sns.heatmap(data.corr(), annot = True, cmap = 'coolwarm') # looking for strong correl
```

Out[17]: <AxesSubplot:>



EDA

In [18]:
```
data.shape
```

Out[18]: (400, 26)

In [19]:
```
data.columns
```

Out[19]:
```
Index(['id', 'age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr',
       'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad',
       'appet', 'pe', 'ane', 'classification'],
      dtype='object')
```

In [20]:
```
data.isnull().sum()
```

Out[20]:
```
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                70
wc                105
rc                130
htn                 2
dm                  8
cad                 4
appet               1
pe                  1
ane                 1
classification      2
dtype: int64
```

In [21]:
```
data.shape[0], data.dropna().shape[0]
```

Out[21]:
```
(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 to keep them, when we will drop those NA valus 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 caseif we provide very less data to our machine learnig 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 that whether the person is suffering about that disease or not then one thing we should keep in my mind that

the model evaluation should have the least false postive errors.

In [22]:
```
data.dropna(inplace=True)
```

In [23]:
```
data.shape
```

Out[23]:
```
(158, 26)
```

Modeling

Logistic Regression

In [24]:
```
from sklearn.linear_model import LogisticRegression
```

In [25]:
```
logreg = LogisticRegression()
```

In [26]:
```
X = data.iloc[:,:-1]
y = data['classification']
```

In [27]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, stratify = y, shuffle = True
```

In [28]:
```
logreg.fit(X_train,y_train)
```

Out[28]:
```
LogisticRegression()
```

In [29]:
```
#Training score

logreg.score(X_train,y_train)
```

Out[29]:
```
1.0
```

In [30]:
```
#Testing accuracy

logreg.score(X_test,y_test)
```

Out[30]:
```
1.0
```

In [31]:
```
test_pred = logreg.predict(X_test)
train_pred = logreg.predict(X_train)
```

In [32]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [33]:
```
print('Train Accuracy: ', accuracy_score(y_train, train_pred))
print('Test Accuracy: ', accuracy_score(y_test, test_pred))
```

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

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.

In [34]:
```
pd.DataFrame(logreg.coef_, columns=X.columns)
```

Out[34]:

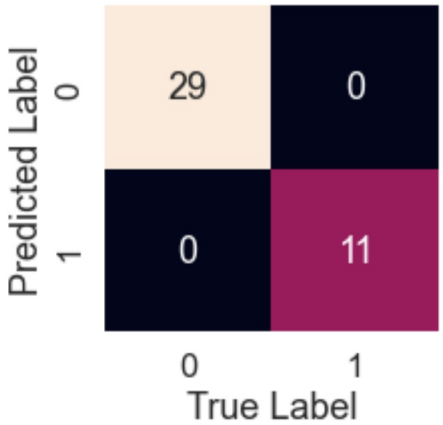
	id	age	bp	sg	al	su	rbc	pc	pcc	ba
0	-0.145556	0.046977	0.071853	0.000221	0.014903	0.004834	0.002736	0.003751	0.000865	0.003376

1 rows × 25 columns

Confusion Matrix

In [35]:
```
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=True,cbar=False)
    plt.xlabel("True Label")
    plt.ylabel("Predicted Label")
```

In [36]:
```
log_pred = logreg.predict(X_test)
plot_conf_mat(y_test, log_pred)
```



In [37]:
```
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}')
```

```
True Neg: 29
False Pos: 0
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 does'nt performs well.

In [38]:
```
df["classification"].value_counts()
```

Out[38]:
```
0.0    115
1.0     43
Name: classification, dtype: int64
```

In [39]:
```
balanced_df = pd.concat([df[df["classification"] == 0], df[df["classification"] == 1
balanced_df.reset_index(drop=True, inplace=True)
```

In [40]:
```
balanced_df["classification"].value_counts()
```

Out[40]:
```
0.0    115
1.0    115
Name: classification, dtype: int64
```

In [41]:
```
X = balanced_df.drop("classification", axis=1)
y = balanced_df["classification"]
```

In [42]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state = 42)
```

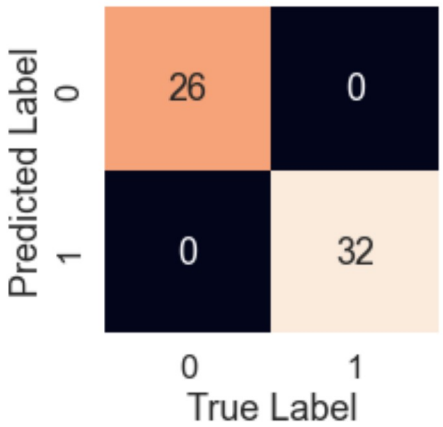
In [43]:
```
ss = StandardScaler()
ss.fit(X_train)
X_train = ss.transform(X_train)
X_test = ss.transform(X_test)
```

In [44]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [45]:
```
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)
```

Out[45]:
```
1.0
```

In [46]:
```
knn_pred = model.predict(X_test)
plot_conf_mat(y_test, knn_pred)
```



In [47]:
```
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}')
```

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

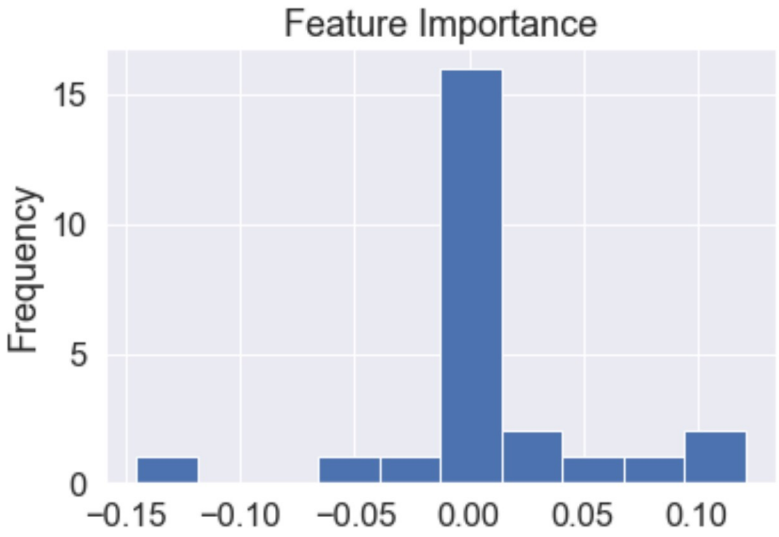
Feature Importance

In [48]:
```
# These coef's tell how much and in what way did each one of it contribute to predict

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

#This is a type of Model driven Exploratory data analysis
```

Out[48]:
```
{'id': -0.14555649821287847,
 'age': 0.046977447201340554,
 'bp': 0.07185295580588096,
 'sg': 0.00022137779797815776,
 'al': 0.014902994149124795,
 'su': 0.004833966042174984,
 'rbc': 0.002735735371008298,
 'pc': 0.003751074765138827,
 'pcc': 0.0008648575839959371,
 'ba': 0.0033760465354954147,
 'bgr': 0.09721892706295206,
 'bu': 0.12078920490806655,
 'sc': 0.017279766792557525,
 'sod': 0.010400238132221663,
 'pot': 0.0009375977914780069,
 'hemo': -0.019924787689860245,
 'pcv': -0.061160028136913715,
 'wc': 0.000846448953045207,
 'rc': -0.004255646775764316,
 'htn': 0.0028593444202493694,
 'dm': 0.0027842042584409923,
 'cad': 0.00011238732038776178,
 'appet': -4.75758605348144e-05,
 'pe': 0.0024114163534211482,
 'ane': 0.000838270769296362}
```

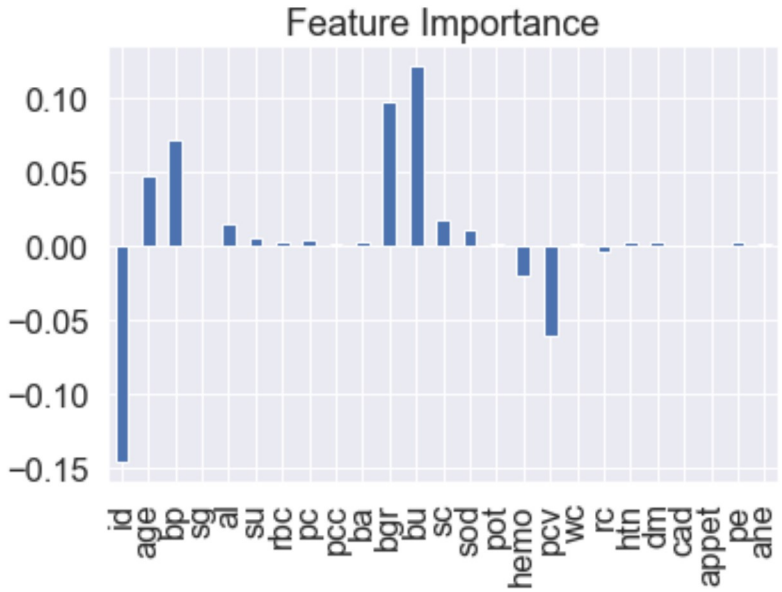
In [49]:
```
#Visualize feature importance

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

Out[49]: <AxesSubplot:title={'center':'Feature Importance'}, ylabel='Frequency'>



In [50]:
```
#Visualize feature importance

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

Out[50]: <AxesSubplot:title={'center':'Feature Importance'}>



Saving model

In [51]:
```
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)
```

Out[51]:
```
array([1., 0., 0., 1., 0., 1., 0., 1., 1., 1., 1., 0., 0., 1., 1., 1., 1.,
       0., 1., 1., 0., 1., 1., 1., 1., 0., 1., 0., 0., 0., 1., 0., 1., 0.,
       0., 1., 1., 1., 0., 1., 0., 0., 1., 0., 0., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 0., 0., 1., 1.])
```

Another method

In [52]:
```
# Now with open function we will save the kidney disease predcition model in write mo
with open('kidney_disease_prediction.pkl', 'wb') as files:
    pickle.dump(logreg, files)
```

In [53]:
```
 # load saved model
with open('kidney_disease_prediction.pkl' , 'rb') as f:
    model = pickle.load(f)
```

In [54]:
```
# check prediction
# lr.predict([[63,1,3,145,233,1,0,150,0,2.3,0,0,1]]) # similar

logreg.predict(X_test)
```

Out[54]:
```
array([1., 0., 0., 1., 0., 1., 0., 1., 1., 1., 1., 0., 0., 1., 1., 1., 1.,
       0., 1., 1., 0., 1., 1., 1., 1., 0., 1., 0., 0., 0., 1., 0., 1., 0.,
       0., 1., 1., 1., 0., 1., 0., 0., 1., 0., 0., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 0., 0., 1., 1.])
```

In [55]:
```
df.head()
```

Out[55]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe
3	3	48.0	70.0	1.005	4.0	0.0	0.0	1.0	1.0	0.0	...	32	6700	3.9	1.0	0.0	0.0	0.0	1.0
9	9	53.0	90.0	1.020	2.0	0.0	1.0	1.0	1.0	0.0	...	29	12100	3.7	1.0	1.0	0.0	0.0	0.0
11	11	63.0	70.0	1.010	3.0	0.0	1.0	1.0	1.0	0.0	...	32	4500	3.8	1.0	1.0	0.0	0.0	1.0
14	14	68.0	80.0	1.010	3.0	2.0	0.0	1.0	1.0	1.0	...	16	11000	2.6	1.0	1.0	1.0	0.0	1.0
20	20	61.0	80.0	1.015	2.0	0.0	1.0	1.0	0.0	0.0	...	24	9200	3.2	1.0	1.0	1.0	0.0	1.0

5 rows × 26 columns

In [56]:
```
# putting datapoints in the model it will either return 0 or 1 i.e. person having chr
logreg.predict([[0,48,80,1.02,1,0,1,0,1,1,2,0,0,1,1,0,1,1,2,0,0,1,0,1,2]])
```

Out[56]:
```
array([1.])
```

Summary

In this article I have used various machine learning model to classify that weather the patient will have chronic kidney problems or not based on the data of patients.

KNN required class balancing, scaling, and model tuning to perform with 100% accuracy, while Logistic Regression was 100% accurate without tuning (note: still had to stratify the train test split).

Logisitic regression in my case seems to be a better model as it provide 100% accuracy yet it used a least resources as it even does'nt required the model tuning.