Classification Assignment

Problem Statement & Requirement:

A requirement from the Hospital, Management asked us to create a predictive model which will predict the Chronic Kidney Disease (CKD) based on the several parameters. The Client has provided the dataset of the same.

1.) Identify your problem statement

We can find the problem statement by using 3 stage method

I. Data provided is numerically - Machine Learning
II. Input & Output Very clear - Supervised Learning
III. Output will be in categorical - Classification

2.) Basic info about the dataset (Total number of rows, columns)

Rows - 399
Columns - 25

3.) Mention the Pre-Processing method if you're doing any (like converting string to number – nominal data)

We used "ONE HOT ENCODING" to converting the string to number for the following column

pc_normal pcc_present ba_present htn_yes dm_yes cad_yes appet_yes pe_yes ane_yes classification_yes

4.) Good evaluation metric is

Random forest the Accuracy is 0.99

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```python
dataset = pd.read_csv('CKD.csv')
dataset
```

Out[2]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	
0	2.000000	76.459948	c	3.0	0.0	normal	abnormal	notpresent	notpresent	1
1	3.000000	76.459948	c	2.0	0.0	normal	normal	notpresent	notpresent	1
2	4.000000	76.459948	a	1.0	0.0	normal	normal	notpresent	notpresent	
3	5.000000	76.459948	d	1.0	0.0	normal	normal	notpresent	notpresent	1
4	5.000000	50.000000	c	0.0	0.0	normal	normal	notpresent	notpresent	1
...	...	...	...	...	...	...	...	...	...	
394	51.492308	70.000000	a	0.0	0.0	normal	normal	notpresent	notpresent	2
395	51.492308	70.000000	c	0.0	2.0	normal	normal	notpresent	notpresent	2
396	51.492308	70.000000	c	3.0	0.0	normal	normal	notpresent	notpresent	1
397	51.492308	90.000000	a	0.0	0.0	normal	normal	notpresent	notpresent	2
398	51.492308	80.000000	a	0.0	0.0	normal	normal	notpresent	notpresent	1

399 rows × 25 columns

In [3]:
```python
dataset=pd.get_dummies(dataset,drop_first=True)
dataset
```

Out[3]:

	age	bp	al	su	bgr	bu	sc	sod
0	2.000000	76.459948	3.0	0.0	148.112676	57.482105	3.077356	137.528754
1	3.000000	76.459948	2.0	0.0	148.112676	22.000000	0.700000	137.528754
2	4.000000	76.459948	1.0	0.0	99.000000	23.000000	0.600000	138.000000
3	5.000000	76.459948	1.0	0.0	148.112676	16.000000	0.700000	138.000000
4	5.000000	50.000000	0.0	0.0	148.112676	25.000000	0.600000	137.528754
...	...	...	...	...	...	...	...	...
394	51.492308	70.000000	0.0	0.0	219.000000	36.000000	1.300000	139.000000
395	51.492308	70.000000	0.0	2.0	220.000000	68.000000	2.800000	137.528754
396	51.492308	70.000000	3.0	0.0	110.000000	115.000000	6.000000	134.000000
397	51.492308	90.000000	0.0	0.0	207.000000	80.000000	6.800000	142.000000
398	51.492308	80.000000	0.0	0.0	100.000000	49.000000	1.000000	140.000000

399 rows × 28 columns

In [4]:
```
dataset.columns
```

Out[4]:
```
Index(['age', 'bp', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hrmo', 'pc
v',
       'wc', 'rc', 'sg_b', 'sg_c', 'sg_d', 'sg_e', 'rbc_normal', 'pc_normal',
       'pcc_present', 'ba_present', 'htn_yes', 'dm_yes', 'cad_yes',
       'appet_yes', 'pe_yes', 'ane_yes', 'classification_yes'],
      dtype='object')
```

In [5]:
```
indep=dataset[['age', 'bp', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hrmo
       'wc', 'rc', 'sg_b', 'sg_c', 'sg_d', 'sg_e', 'rbc_normal', 'pc_normal',
       'pcc_present', 'ba_present', 'htn_yes', 'dm_yes', 'cad_yes',
       'appet_yes', 'pe_yes', 'ane_yes']]
indep
```

Out[5]:

	age	bp	al	su	bgr	bu	sc	sod
0	2.000000	76.459948	3.0	0.0	148.112676	57.482105	3.077356	137.528754
1	3.000000	76.459948	2.0	0.0	148.112676	22.000000	0.700000	137.528754
2	4.000000	76.459948	1.0	0.0	99.000000	23.000000	0.600000	138.000000
3	5.000000	76.459948	1.0	0.0	148.112676	16.000000	0.700000	138.000000
4	5.000000	50.000000	0.0	0.0	148.112676	25.000000	0.600000	137.528754
...	...	...	...	...	...	...	...	...
394	51.492308	70.000000	0.0	0.0	219.000000	36.000000	1.300000	139.000000
395	51.492308	70.000000	0.0	2.0	220.000000	68.000000	2.800000	137.528754
396	51.492308	70.000000	3.0	0.0	110.000000	115.000000	6.000000	134.000000
397	51.492308	90.000000	0.0	0.0	207.000000	80.000000	6.800000	142.000000
398	51.492308	80.000000	0.0	0.0	100.000000	49.000000	1.000000	140.000000

399 rows × 27 columns

In [6]:
```
dep=dataset['classification_yes'].value_counts()
dep
```

Out[6]:
```
1    249
0    150
Name: classification_yes, dtype: int64
```

In [7]:
```
dep=dataset['classification_yes']
dep
```

Out[7]:
```
0      1
1      1
2      1
3      1
4      1
      ..
394    1
395    1
396    1
397    1
398    0
Name: classification_yes, Length: 399, dtype: uint8
```

In [8]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(indep, dep, test_size = 0.
```

In [9]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_= sc.fit_transform(X_train)
X_test_= sc.transform(X_test)
```

In [10]:
```python
from sklearn.ensemble import RandomForestClassifier
#https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-paramet
```

In [11]:
```python
from sklearn.model_selection import GridSearchCV

param_grid = {'criterion':['gini','entropy'],
              'max_features': ['auto','sqrt','log2'],
              'n_estimators':[10,100]}



grid = GridSearchCV(RandomForestClassifier(), param_grid, refit = True, verbos

# fitting the model for grid search
grid.fit(X_train_, y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

Out[11]:
```
GridSearchCV(estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': [10, 100]},
             scoring='f1', verbose=3)
```

In [12]:
```python
# print best parameter after tuning
#print(grid.best_params_)

re=grid.cv_results_
grid_predictions = grid.predict(X_test_)


from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, grid_predictions)


from sklearn.metrics import classification_report
clf_report = classification_report(y_test, grid_predictions)
```

In [13]:
```python
from sklearn.metrics import f1_score
f1_macro=f1_score(y_test,grid_predictions,average='weighted')
print("The f1_macro value for best parameter {}:".format(grid.best_params_),f1
```

```
The f1_macro value for best parameter {'criterion': 'entropy', 'max_features':
'log2', 'n_estimators': 10}: 0.9916844900066377
```

In [14]:
```python
print("The confusion Matrix:\n",cm)
```

```
The confusion Matrix:
 [[45  0]
 [ 1 74]]
```

In [15]:
```python
print("The report:\n",clf_report)
```

```
The report:
               precision    recall  f1-score   support

           0       0.98      1.00      0.99        45
           1       1.00      0.99      0.99        75

    accuracy                           0.99       120
   macro avg       0.99      0.99      0.99       120
weighted avg       0.99      0.99      0.99       120
```

In [16]:
```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test,grid.predict_proba(X_test)[:,1])
```

```
C:\Users\user\Anaconda3\lib\site-packages\sklearn\base.py:444: UserWarning: X h
as feature names, but RandomForestClassifier was fitted without feature names
  f"X has feature names, but {self.__class__.__name__} was fitted without"
```

Out[16]: 0.76

In [17]:
```
table=pd.DataFrame.from_dict(re)
table
```

Out[17]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterio
0	0.021159	0.002828	0.005167	0.001655	gi
1	0.183381	0.032052	0.026656	0.004864	gi
2	0.024977	0.004100	0.004559	0.000712	gi
3	0.208134	0.010818	0.029623	0.002795	gi
4	0.024019	0.001252	0.005918	0.000200	gi
5	0.270043	0.015502	0.037649	0.005358	gi
6	0.028137	0.001464	0.005966	0.000106	entrop
7	0.315721	0.031694	0.036067	0.000687	entrop
8	0.042569	0.001531	0.010681	0.002382	entrop
9	0.337607	0.020517	0.038013	0.001252	entrop
10	0.036254	0.004131	0.008329	0.000881	entrop
11	0.297709	0.004842	0.033924	0.004193	entrop

In [ ]:

```
age=float(input("Age:"))
bP=float(input("BP:"))
al=float(input("AL:"))
su=float(input("SU:"))
rbc_normal=int(input("RBC_NORMAL:"))
pc_normal=int(input("PC_NORMAL:"))
pcc_present=float(input("PCC_PREASENT:"))
ba_present=float(input("BA_PRESENT:"))
bgr=float(input("BGR:"))
pcv=float(input("PCV:"))
wc=float(input("WC:"))
rc=float(input("RC:"))
htn_yes=int(input("HTN_YES:"))
dm_yes=int(input("DM_YES:"))
cad_yes=int(input("CAD_YES:"))
appet_yes=int(input("APPET_YES:"))
pe_yes=int(input("PE_YES:"))
ane_yes=int(input("ANE_YES:"))
```

In [ ]:

```
Future_Prediction=grid.predict([[age,bP,al,su,bgr,bu,sc,sod,pot,hrmo,pcv,wc,rc
print("Future_Prediction={}".format(Future_Prediction))
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: