```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

df=pd.read_csv(r'C:¥Users¥Gokul¥Downloads¥chronickidneydisease (1).csv')
```

Exploring Dataset

```
print("The dataset shape is {}".format(df.shape))
```

The dataset shape is (400, 26)

The dataset shape is (400, 26)

```
df.head()
```

	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

```
df.columns

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')

df.dtypes

id                  int64
age               float64
bp                float64
sg                float64
al                float64
su                float64
rbc                object
pc                 object
pcc                object
ba                 object
bgr               float64
bu                float64
sc                float64
sod               float64
pot               float64
hemo              float64
pcv                object
wc                 object
rc                 object
```

```
htn               object
dm                object
cad               object
appet             object
pe                object
ane               object
classification    object
dtype: object
```

df.info()

```
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

df.describe().T

	count	mean	std	min	25%	50%	75%	max
id	400.0	199.500000	115.614301	0.000	99.75	199.50	299.25	399.000
age	391.0	51.483376	17.169714	2.000	42.00	55.00	64.50	90.000
bp	388.0	76.469072	13.683637	50.000	70.00	80.00	80.00	180.000
sg	353.0	1.017408	0.005717	1.005	1.01	1.02	1.02	1.025
al	354.0	1.016949	1.352679	0.000	0.00	0.00	2.00	5.000
su	351.0	0.450142	1.099191	0.000	0.00	0.00	0.00	5.000
bgr	356.0	148.036517	79.281714	22.000	99.00	121.00	163.00	490.000

bu	381.0	57.425722	50.503006	1.500	27.00	42.00	66.00	391.000
sc	383.0	3.072454	5.741126	0.400	0.90	1.30	2.80	76.000
sod	313.0	137.528754	10.408752	4.500	135.00	138.00	142.00	163.000
pot	312.0	4.627244	3.193904	2.500	3.80	4.40	4.90	47.000
hemo	348.0	12.526437	2.912587	3.100	10.30	12.65	15.00	17.800

```
for col in df:
    unique=df[col].value_counts()
    print(unique,"¥n========================= ¥n")
```

```
0      1
263    1
273    1
272    1
271    1
      ..
130    1
129    1
128    1
127    1
399    1
Name: id, Length: 400, dtype: int64
========================


60.0    19
65.0    17
48.0    12
```

55.0 12

50.0 12

..

83.0 1

27.0 1

14.0 1

81.0 1

79.0 1

Name: age, Length: 76, dtype: int64

========================

80.0 116

70.0 112

60.0 71

90.0 53

100.0 25

50.0 5

110.0 3

140.0 1

180.0 1

120.0 1

Name: bp, dtype: int64

========================

1.020 106

```
1.010    84
1.025    81
1.015    75
1.005     7
Name: sg, dtype: int64
========================


0.0    199
1.0     44
2.0     43
3.0     43
4.0     24
5.0      1
Name: al, dtype: int64
========================


0.0    290
2.0     18
3.0     14
4.0     13
1.0     13
5.0      3
Name: su, dtype: int64
========================
```

normal 201

abnormal 47

Name: rbc, dtype: int64

========================

normal 259

abnormal 76

Name: pc, dtype: int64

========================

notpresent 354

present 42

Name: pcc, dtype: int64

========================

notpresent 374

present 22

Name: ba, dtype: int64

========================

99.0 10

93.0 9

100.0 9

107.0 8

131.0 6

```
       ..
288.0    1
182.0    1
84.0     1
256.0    1
226.0    1
Name: bgr, Length: 146, dtype: int64
========================

46.0    15
25.0    13
19.0    11
40.0    10
50.0     9
        ..
176.0    1
145.0    1
92.0     1
322.0    1
186.0    1
Name: bu, Length: 118, dtype: int64
========================

1.2    40
1.1    24
```

```
0.5      23
1.0      23
0.9      22
         ..
3.8       1
12.2      1
9.2       1
13.8      1
0.4       1
Name: sc, Length: 84, dtype: int64
========================

135.0     40
140.0     25
141.0     22
139.0     21
138.0     20
142.0     20
137.0     19
150.0     17
136.0     17
147.0     13
145.0     11
132.0     10
146.0     10
```

131.0	9
144.0	9
133.0	8
130.0	7
134.0	6
143.0	4
124.0	3
127.0	3
122.0	2
113.0	2
120.0	2
125.0	2
128.0	2
114.0	2
126.0	1
163.0	1
115.0	1
129.0	1
4.5	1
104.0	1
111.0	1

Name: sod, dtype: int64

========================

3.5	30

5.0	30
4.9	27
4.7	17
4.8	16
3.9	14
3.8	14
4.1	14
4.2	14
4.0	14
4.4	14
4.5	13
4.3	12
3.7	12
3.6	8
4.6	7
3.4	5
5.2	5
5.3	4
5.7	4
3.2	3
5.5	3
6.3	3
5.4	3
2.9	3
3.3	3

5.6	2
3.0	2
6.5	2
2.5	2
5.9	2
5.8	2
7.6	1
47.0	1
6.6	1
5.1	1
6.4	1
2.8	1
2.7	1
39.0	1

Name: pot, dtype: int64

========================

15.0	16
10.9	8
13.6	7
13.0	7
9.8	7
	..
6.8	1
8.5	1

7.3	1
12.8	1
17.6	1

Name: hemo, Length: 115, dtype: int64

========================

41	21
52	21
44	19
48	19
40	16
43	14
42	13
45	13
32	12
36	12
33	12
50	12
28	12
34	11
37	11
30	9
29	9
35	9
46	9

31	8
24	7
39	7
26	6
38	5
53	4
51	4
49	4
47	4
54	4
25	3
27	3
22	3
19	2
23	2
15	1
21	1
17	1
20	1
¥t43	1
18	1
9	1
14	1
¥t?	1
16	1

```
Name: pcv, dtype: int64
========================

9800     11
6700     10
9200      9
9600      9
7200      9
         ..
19100     1
¥t?       1
12300     1
14900     1
12700     1
Name: wc, Length: 92, dtype: int64
========================

5.2    18
4.5    16
4.9    14
4.7    11
4.8    10
3.9    10
4.6     9
3.4     9
```

5.9	8
5.5	8
6.1	8
5.0	8
3.7	8
5.3	7
5.8	7
5.4	7
3.8	7
5.6	6
4.3	6
4.2	6
3.2	5
4.4	5
5.7	5
6.4	5
5.1	5
6.2	5
6.5	5
4.1	5
3.6	4
6.3	4
6.0	4
4.0	3
3.3	3

```
4        3
3.5      3
2.9      2
3.1      2
2.6      2
2.1      2
2.5      2
2.8      2
3.0      2
2.7      2
5        2
2.3      1
¥t?      1
2.4      1
3        1
8.0      1
Name: rc, dtype: int64
========================


no      251
yes     147
Name: htn, dtype: int64
========================


no        258
```

```
yes          134
¥tno           3
¥tyes          2
  yes          1
Name: dm, dtype: int64
=========================

no          362
yes          34
¥tno          2
Name: cad, dtype: int64
=========================

good       317
poor        82
Name: appet, dtype: int64
=========================

no        323
yes        76
Name: pe, dtype: int64
=========================

no        339
yes        60
```

Name: ane, dtype: int64

========================

ckd 248

notckd 150

ckd¥t 2

Name: classification, dtype: int64

========================

Remove unwanted columns

```
df.drop('id',axis=1,inplace=True)
```

Cleaning the Data values

```
# cleaning 'PCV'

df['pcv']=df['pcv'].apply(lambda x:x if type(x)==type(3.5) else x.replace('¥t43','43').replace('¥t?','Nan'))

# cleaning "WC"

df['wc']=df['wc'].apply(lambda x:x if type(x)==type(3.5) else
x.replace('¥t?','Nan').replace('¥t6200','6200').replace('¥t8400','8400'))

# cleaning "RC"

df['rc']=df['rc'].apply(lambda x:x if type(x)==type(3.5) else x.replace('¥t?','Nan'))

# cleaning "dm"

df['dm']=df['dm'].apply(lambda x:x if type(x)==type(3.5) else
```

```
x.replace('¥tno','no').replace('¥tyes','yes').replace(' yes','yes'))


# cleaning "CAD"
df['cad']=df['cad'].apply(lambda x:x if type(x)==type(3.5) else x.replace('¥tno','no'))


# cleaning "Classification"
df['classification']=df['classification'].apply(lambda x:x if type(x)==type(3.5) else x.replace('ckd¥t','ckd'))
mistyped=[['pcv','rc','wc']]
for i in mistyped:
    df[i]=df[i].astype('float')
#    define categoricsl features
cat_cols=list(df.select_dtypes('object'))
cat_cols
['rbc',
 'pc',
 'pcc',
 'ba',
 'htn',
 'dm',
 'cad',
 'appet',
 'pe',
 'ane',
 'classification']
# define numeric features
```

```
num_cols=list(df.select_dtypes(['int64','float64']))

num_cols

['age',
 'bp',
 'sg',
 'al',
 'su',
 'bgr',
 'bu',
 'sc',
 'sod',
 'pot',
 'hemo',
 'pcv',
 'wc',
 'rc']

# Checking missing/Nan values

df.isnull().sum().sort_values(ascending=False)

rbc       152
rc        131
wc        106
pot        88
sod        87
pcv        71
pc         65
```

```
hemo              52
su                49
sg                47
al                46
bgr               44
bu                19
sc                17
bp                12
age                9
ba                 4
pcc                4
htn                2
dm                 2
cad                2
appet              1
pe                 1
ane                1
classification     0
dtype: int64
```

```
# Let's impute Nan Values with median in numeric features
for col in num_cols:
    df[col]=df[col].fillna(df[col].median())
# let's impute categorical features with most frequent value
df['rbc'].fillna('normal',inplace=True)
df['pc'].fillna('normal',inplace=True)
```

```
df['pcc'].fillna('notpresent',inplace=True)

df['ba'].fillna('notpresent',inplace=True)

df['htn'].fillna('no',inplace=True)

df['dm'].fillna('no',inplace=True)

df['cad'].fillna('no',inplace=True)

df['appet'].fillna('good',inplace=True)

df['pe'].fillna('no',inplace=True)

df['ane'].fillna('no',inplace=True)

df.isna().sum().sort_values(ascending=False)

age        0
pot        0
ane        0
pe         0
appet      0
cad        0
dm         0
htn        0
rc         0
wc         0
pcv        0
hemo       0
sod        0
bp         0
sc         0
bu         0
```

```
bgr               0
ba                0
pcc               0
pc                0
rbc               0
su                0
al                0
sg                0
classification    0
dtype: int64
```

```
# Encode classification
df['classification']=df['classification'].map({'ckd':1,'notckd':0})
attr_count=df['classification'].value_counts()
attr_label=df['classification'].value_counts().index

# plot
fig,ax=plt.subplots(figsize=(14,6))
ax.pie(attr_count,explode=(0.1,0),labels=attr_label,autopct='%.2f%%',startangle=90)
ax.set_title("Classification ",fontsize=15)
plt.show()

fig,ax=plt.subplots(figsize=(7,70),ncols=1,nrows=14)

i=0
for col in num_cols:
```

```
        sns.kdeplot(x=df[col],fill=True,alpha=1,ax=ax[i])
        ax[i].set_xlabel(' ')
        ax[i].set_ylabel(' ')
        ax[i].set_title(col,fontsize=21)
        i=i+1
plt.show()


# check skewness of the distribution
skew=[]
for col in num_cols:
        skew.append(round(df[col].skew(),3))
num_dist=pd.DataFrame({'features':num_cols,'skewness':skew})
num_dist
```

	features	skewness
0	age	-0.689
1	bp	1.602
2	sg	-0.333
3	al	1.180
4	su	2.700
5	bgr	2.204
6	bu	2.724
7	sc	7.666
8	sod	-7.929
9	pot	13.133
10	hemo	-0.377

11	pcv	-0.549
12	wc	2.002
13	rc	-0.330

```
plt.figure(figsize=(16,8))
plt.title('Correlation between All Numerical Features',size=15)

# create mask
mask=np.triu(np.ones_like(df.corr()))

# create colormap
colormap=sns.color_palette('Blues')
# plot heatmap
sns.heatmap(df.corr(),annot=True,cmap=colormap,mask=mask)
plt.show()

df.drop('pcv',axis=1,inplace=True)
num_cols.remove('pcv')
```

Target Relationship

```
tg_num_corr=[]

for col in num_cols:
    tg_num_corr.append(df[col].corr(df['classification']))

# create as DataFrame
```

```
tg_num_df=pd.DataFrame({'numerical_predictor':num_cols,'correlation_w_target':tg_num_corr})

# sort the DataFrmae by the absolute vaue of their correlation coefficient,descending
tg_num_df=tg_num_df.sort_values(by='correlation_w_target',ascending=False).reset_index(drop=True)

tg_num_df
```

	numerical_predictor	correlation_w_target
0	al	0.531562
1	bgr	0.379321
2	bu	0.369393
3	su	0.294555
4	bp	0.293693
5	sc	0.291245
6	age	0.227842
7	wc	0.177571
8	pot	0.065218
9	sod	-0.334900
10	rc	-0.566163
11	sg	-0.659504
12	hemo	-0.726368

```
# display as figure
plt.figure(figsize=(7,5))
sns.barplot(x=tg_num_df['correlation_w_target'],y=tg_num_df['numerical_predictor'],color='#a2c9f4')
plt.xlabel('Correlation Coefficient')
plt.title('Numerical-Target Relationship',fontsize=12)
```

```
plt.show()


# set the figure
fig,ax=plt.subplots(ncols=1,nrows=14,figsize=(7,70))
i=0
for col in num_cols:
    sns.boxplot(data=df,x=col,ax=ax[i],palette='pastel')
    ax[i].set_title(col,fontsize=14)
    i=i+1


plt.show()
```

Encoding

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

Normalization

```
# scaling with MinMaxScaler

from sklearn.preprocessing import StandardScaler,MinMaxScaler

mm_scaler=MinMaxScaler()

df[num_cols]=mm_scaler.fit_transform(df[num_cols])
```

Model Building

```
from sklearn.model_selection import train_test_split

x=df.drop('classification',axis=1)

y=df['classification']


X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)

print("X_train size {} , X_test size {}".format(X_train.shape,X_test.shape))

X_train size (320, 23) , X_test size (80, 23)

# Using GridSearchCV we find the best algorithm to this problem

from sklearn.model_selection import ShuffleSplit,GridSearchCV,StratifiedKFold

from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

# Crete a function to find the best algo. for this problem

def find_best_model(x,y):


models={'Logistic_regression':{'model':LogisticRegression(solver='liblinear',penalty='l2',multi_class='aut
o'),'parameter':{'C':[1,4,8]}},


'decision_tree':{'model':DecisionTreeClassifier(splitter='best'),'parameter':{'criterion':['gini','entropy'],'m
```

```
ax_depth':[5,7,13,15]}},
            'svm':{'model':SVC(gamma='auto'),'parameter':{'kernel':['sigmoid','linear'],'C':[1,5,10,15]}},
'random_forest':{'model':RandomForestClassifier(criterion='gini'),'parameter':{'max_depth':[5,10,15],'n_
estimators':[1,3,5]}}}
    scores=[]
    cv_shuffle=StratifiedKFold(n_splits=10)

    for model_name,model_params in models.items():
gs=GridSearchCV(model_params['model'],model_params['parameter'],cv=cv_shuffle,return_train_score
=False)
        gs.fit(x,y)
scores.append({'model':model_name,'best_parameters':gs.best_params_,'score':gs.best_score_})
    return pd.DataFrame(scores,columns=['model','best_parameters','score'])
find_best_model(X_train,y_train)
```

	model	best_parameters	score
0	Logistic_regression	{'C': 4}	0.975000
1	decision_tree	{'criterion': 'entropy', 'max_depth': 7}	0.981250
2	svm	{'C': 5, 'kernel': 'linear'}	0.978125
3	random_forest	{'max_depth': 15, 'n_estimators': 5}	0.993750

```
# Using cross_val_score for gaining average accuracy
from sklearn.model_selection import cross_val_score
score=cross_val_score(RandomForestClassifier(max_depth=15,n_estimators=5),X_train,y_train,cv=10)
print("Average Accuracy Score {}".format(score.mean()))
```

Average Accuracy Score 0.98125

```
# Creating Random Forest model

rf=RandomForestClassifier(max_depth=5,n_estimators=5)

rf.fit(X_train,y_train)

RandomForestClassifier(max_depth=5, n_estimators=5)
```

Model Evaluation

```
# Creating a confusion matrix

from sklearn.metrics import confusion_matrix,classification_report,accuracy_score

y_pred=rf.predict(X_test)

cm=confusion_matrix(y_pred,y_test)

cm

array([[28,    4],

       [ 0, 48]], dtype=int64)

# Plotting the confusion matrix

plt.figure(figsize=(10,7))

p = sns.heatmap(cm, annot=True, cmap="Blues", fmt='g')

plt.title('Confusion matrix for RandomForest Model - Test Set')

plt.xlabel('Predicted Values')

plt.ylabel('Actual Values')

plt.show()


# Accuracy score

score=round(accuracy_score(y_test,y_pred),3)

print("Accuracy on the Test set: {}".format(score))

Accuracy on the Test set: 0.95
```

```
# Classification report
print(classification_report(y_test,y_pred))
              precision    recall  f1-score   support

           0       0.88      1.00      0.93        28
           1       1.00      0.92      0.96        52

    accuracy                           0.95        80
   macro avg       0.94      0.96      0.95        80
weighted avg       0.96      0.95      0.95        80

# Creating a confusion matrix for training set
y_train_pred=rf.predict(X_train)
cm=confusion_matrix(y_train,y_train_pred)
cm
array([[121,   1],
       [  3, 195]], dtype=int64)
# Accuracy score
score=round(accuracy_score(y_train,y_train_pred),3)
print("Accuracy on training set: {}".format(score))
Accuracy on training set: 0.988
print(classification_report(y_train,y_train_pred))
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       122
```

1	0.99	0.98	0.99	198
accuracy			0.99	320
macro avg	0.99	0.99	0.99	320
weighted avg	0.99	0.99	0.99	320

Feature Selection

```
# Top 10 Features
feature_scores=pd.DataFrame(rf.feature_importances_,columns=['Score'],index=X_train.columns).sort_values(by='Score',ascending=False)
top10_feature = feature_scores.nlargest(n=10, columns=['Score'])

plt.figure(figsize=(14,6))
g = sns.barplot(x=top10_feature.index, y=top10_feature['Score'])
p = plt.title('Top 10 Features with Random Forest')
p = plt.xlabel('Feature name')
p = plt.ylabel('Random Forest score')
p = g.set_xticklabels(g.get_xticklabels(), horizontalalignment='right')
```

Prediction

```
X_train=X_train[['hemo','rc','sg','al','sc','htn','sod','bp','wc','age']]
X_test=X_test[['hemo','rc','sg','al','sc','htn','sod','bp','wc','age']]
rf.fit(X_train,y_train)
RandomForestClassifier(max_depth=5, n_estimators=5)
```

```
# Prediction 1

# input parameter : Hemoglobin(hemo), Red Blood Cells(rc), Specific Gravity(sg), Albumin(al), Searum Creatinite(sc),

# Hypertension(htn), Sodium(sod), Blood Pressure(bp), White Blood Cells(wc), Age

prediction = rf.predict([[67.4,7.2,0.99,4,17.0,1,160.6,87,22089,36]])[0]

if prediction:

    print('Oops! You have Chronic Kidney Disease.')

else:

    print("Great! You don't have Chronic Kidney Disease.")

Oops! You have Chronic Kidney Disease.

print(prediction)

1

import pickle

pickle.dump(rf,open("model.pkl","wb"))
```