importing the libraries

In [143]:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
from numpy.random import RandomState
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import mean_squared_error, accuracy_score, confu
```

Loading the dataset

In [144]:
```
# Load the dataset
data = pd.read_csv('kidney_disease.csv').drop('id', axis=1)
data.dtypes
```

Out[144]:
```
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
```

Pre-processing the data

In [145]:

```
# Convert categorical variables to numerical
le = preprocessing.LabelEncoder()
categorical_cols = ["rbc","pc","pcc","ba","pcv","wc","rc","htn","dm",
for col in categorical_cols:
    data[col] = le.fit_transform(data[col])
    print(le.classes_)
```

```
['abnormal' 'normal' nan]
['abnormal' 'normal' nan]
['notpresent' 'present' nan]
['notpresent' 'present' nan]
['\t43' '\t?' '14' '15' '16' '17' '18' '19' '20' '21' '22' '23' '24' '2
5'
 '26' '27' '28' '29' '30' '31' '32' '33' '34' '35' '36' '37' '38' '39'
 '40' '41' '42' '43' '44' '45' '46' '47' '48' '49' '50' '51' '52' '53'
 '54' '9' nan]
['\t6200' '\t8400' '\t?' '10200' '10300' '10400' '10500' '10700' '10800'
 '10900' '11000' '11200' '11300' '11400' '11500' '11800' '11900' '12000'
 '12100' '12200' '12300' '12400' '12500' '12700' '12800' '13200' '13600'
 '14600' '14900' '15200' '15700' '16300' '16700' '18900' '19100' '21600'
 '2200' '2600' '26400' '3800' '4100' '4200' '4300' '4500' '4700' '4900'
 '5000' '5100' '5200' '5300' '5400' '5500' '5600' '5700' '5800' '5900'
 '6000' '6200' '6300' '6400' '6500' '6600' '6700' '6800' '6900' '7000'
 '7100' '7200' '7300' '7400' '7500' '7700' '7800' '7900' '8000' '8100'
 '8200' '8300' '8400' '8500' '8600' '8800' '9000' '9100' '9200' '9300'
 '9400' '9500' '9600' '9700' '9800' '9900' nan]
['\t?' '2.1' '2.3' '2.4' '2.5' '2.6' '2.7' '2.8' '2.9' '3' '3.1' '3.2'
 '3.3' '3.4' '3.5' '3.6' '3.7' '3.8' '3.9' '4' '4.1' '4.2' '4.3' '4.4'
 '4.5' '4.6' '4.7' '4.8' '4.9' '5' '5.1' '5.2' '5.3' '5.4' '5.5' '5.6'
 '5.7' '5.8' '5.9' '6' '6.1' '6.2' '6.3' '6.4' '6.5' '8' nan]
['no' 'yes' nan]
['\tno' '\tyes' ' yes' 'no' 'yes' nan]
['\tno' 'no' 'yes' nan]
['good' 'poor' nan]
['no' 'yes' nan]
['no' 'yes' nan]
['ckd' 'ckd\t' 'notckd']
```

In [146]:
```
1 data.dtypes
```

Out[146]:
```
age               float64
bp                float64
sg                float64
al                float64
su                float64
rbc                 int32
pc                  int32
pcc                 int32
ba                  int32
bgr               float64
bu                float64
sc                float64
sod               float64
pot               float64
hemo              float64
pcv                 int32
wc                  int32
rc                  int32
htn                 int32
dm                  int32
cad                 int32
appet               int32
pe                  int32
ane                 int32
classification      int32
dtype: object
```

In [147]:
```
1 data.head(5)
```

Out[147]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	ap
0	48.0	80.0	1.020	1.0	0.0	2	1	0	0	121.0	...	32	72	31	1	4	1	
1	7.0	50.0	1.020	4.0	0.0	2	1	0	0	NaN	...	26	56	46	0	3	1	
2	62.0	80.0	1.010	2.0	3.0	1	1	0	0	423.0	...	19	70	46	0	4	1	
3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	117.0	...	20	62	18	1	3	1	
4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	106.0	...	23	68	25	0	3	1	

5 rows × 25 columns

In [148]:
```
# filling null values
for col in data.columns:
    data[col].fillna(data[col].median(),inplace=True)
data.head(5)
```

Out[148]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	ap
0	48.0	80.0	1.020	1.0	0.0	2	1	0	0	121.0	...	32	72	31	1	4	1	
1	7.0	50.0	1.020	4.0	0.0	2	1	0	0	121.0	...	26	56	46	0	3	1	
2	62.0	80.0	1.010	2.0	3.0	1	1	0	0	423.0	...	19	70	46	0	4	1	
3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	117.0	...	20	62	18	1	3	1	
4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	106.0	...	23	68	25	0	3	1	

5 rows × 25 columns

In [149]:
```
data.describe()
```

Out[149]:

	age	bp	sg	al	su	rbc	pc
count	400.000000	400.000000	400.000000	400.00000	400.000000	400.000000	400.000000
mean	51.562500	76.575000	1.017712	0.90000	0.395000	1.262500	0.972500
std	16.982996	13.489785	0.005434	1.31313	1.040038	0.655491	0.593823
min	2.000000	50.000000	1.005000	0.00000	0.000000	0.000000	0.000000
25%	42.000000	70.000000	1.015000	0.00000	0.000000	1.000000	1.000000
50%	55.000000	80.000000	1.020000	0.00000	0.000000	1.000000	1.000000
75%	64.000000	80.000000	1.020000	2.00000	0.000000	2.000000	1.000000
max	90.000000	180.000000	1.025000	5.00000	5.000000	2.000000	2.000000

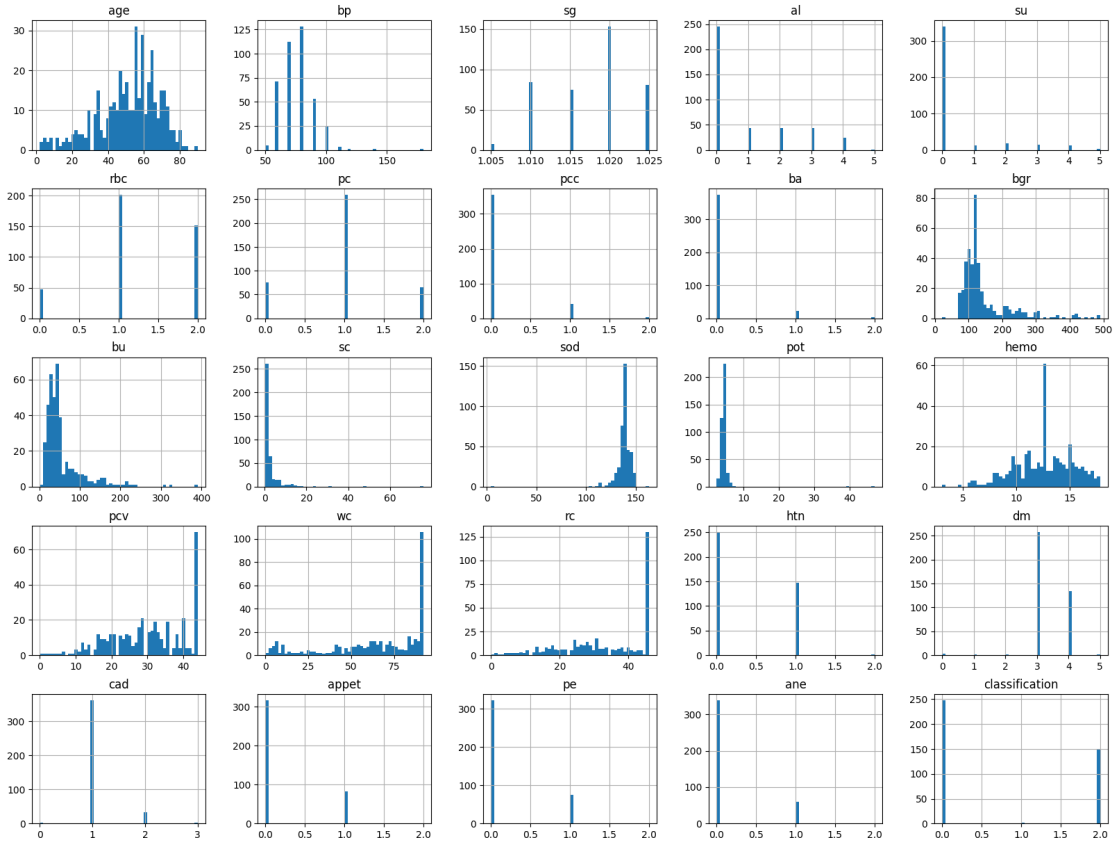
8 rows × 25 columns

In [150]:
```
# checking whether are any null values
data.isnull().sum()
```

Out[150]:
```
age               0
bp                0
sg                0
al                0
su                0
rbc               0
pc                0
pcc               0
ba                0
bgr               0
bu                0
sc                0
sod               0
pot               0
hemo              0
pcv               0
wc                0
rc                0
htn               0
dm                0
cad               0
appet             0
pe                0
ane               0
classification    0
dtype: int64
```

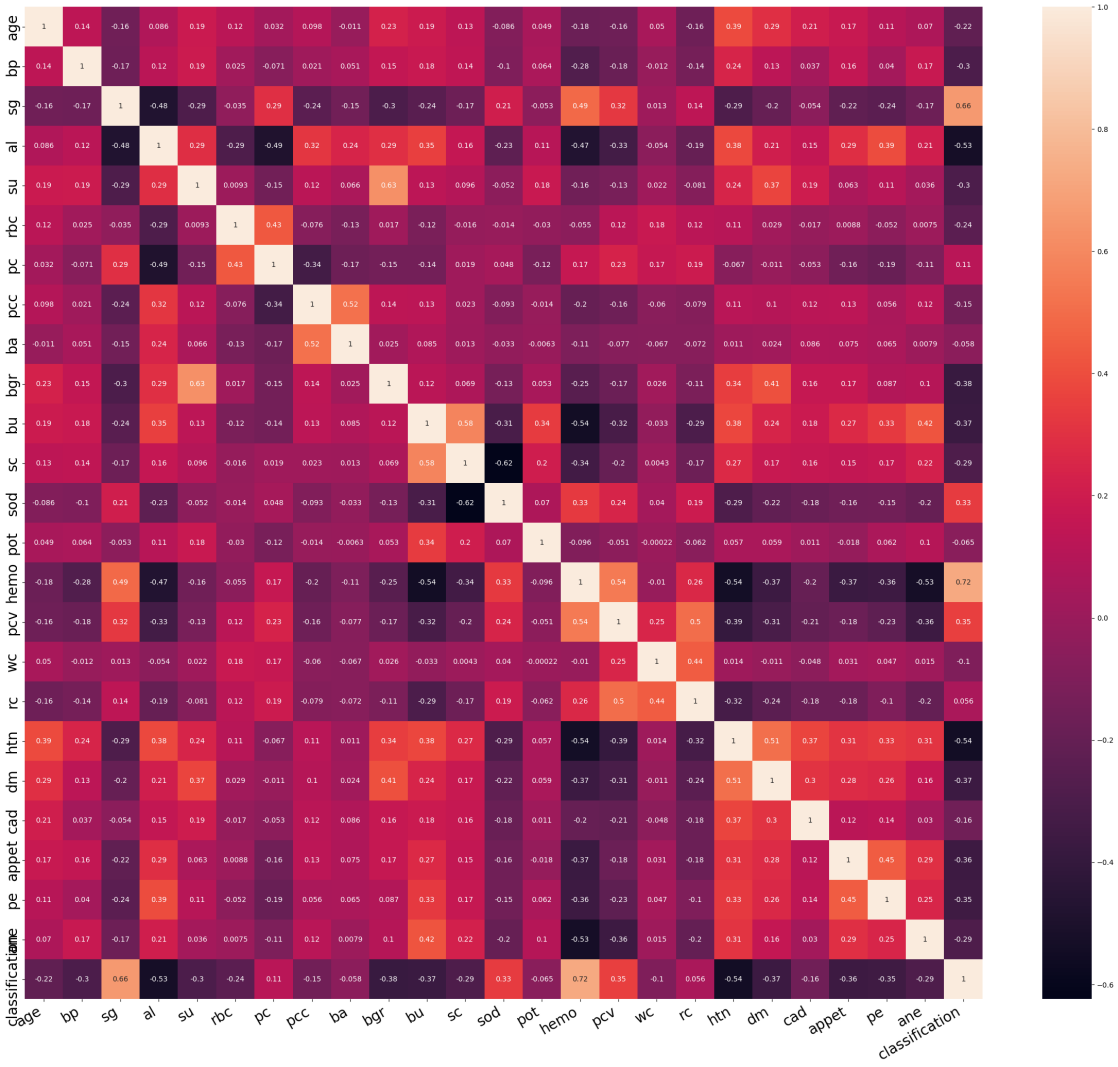
In [151]:

```
1 data.hist(bins=50,figsize=(20,15))
2 plt.show()
```



In [152]:

```
correlation_figure, correlation_axis = plt.subplots(figsize = (30,25))
corr_mtrx = data.corr()
correlation_axis = sns.heatmap(corr_mtrx, annot= True)

plt.xticks(rotation = 30, horizontalalignment = 'right', fontsize = 2
plt.yticks(fontsize = 20)
plt.show()
```



In [153]:

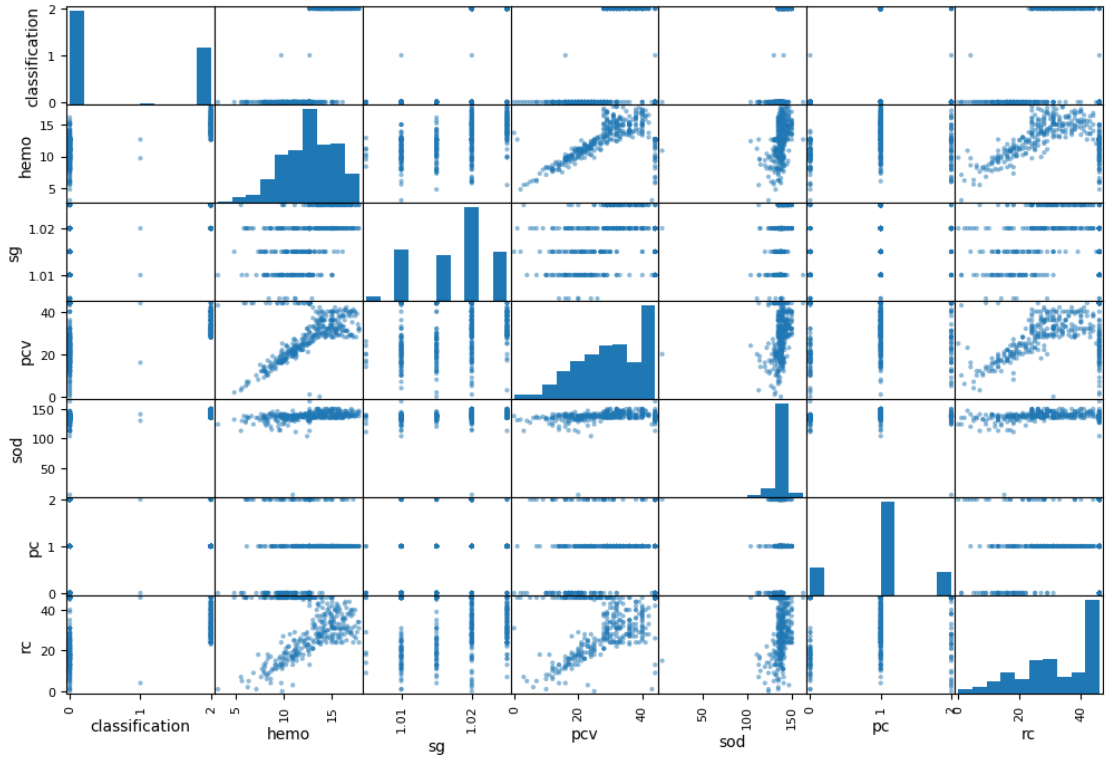
```
corr_matrix = data.corr()
corr_matrix['classification'].sort_values(ascending=False)
```

Out[153]:

```
classification    1.000000
hemo              0.724742
sg                0.657810
pcv               0.349749
sod               0.334153
pc                0.114674
rc                0.055589
ba               -0.057511
pot              -0.065448
wc               -0.102709
pcc              -0.149153
cad              -0.155347
age              -0.222985
rbc              -0.238042
sc               -0.292050
ane              -0.293151
bp               -0.296613
su               -0.296919
pe               -0.352622
appet            -0.359114
dm               -0.366016
bu               -0.371536
bgr              -0.378495
al               -0.531885
htn              -0.543271
Name: classification, dtype: float64
```

In [154]:
```
attributes = ['classification','hemo','sg','pcv','sod','pc','rc']
scatter_matrix(data[attributes],figsize=(12,8))
```

Out[154]:
```
array([[<AxesSubplot: xlabel='classification', ylabel='classification'>,
        <AxesSubplot: xlabel='hemo', ylabel='classification'>,
        <AxesSubplot: xlabel='sg', ylabel='classification'>,
        <AxesSubplot: xlabel='pcv', ylabel='classification'>,
        <AxesSubplot: xlabel='sod', ylabel='classification'>,
        <AxesSubplot: xlabel='pc', ylabel='classification'>,
        <AxesSubplot: xlabel='rc', ylabel='classification'>],
       [<AxesSubplot: xlabel='classification', ylabel='hemo'>,
        <AxesSubplot: xlabel='hemo', ylabel='hemo'>,
        <AxesSubplot: xlabel='sg', ylabel='hemo'>,
        <AxesSubplot: xlabel='pcv', ylabel='hemo'>,
        <AxesSubplot: xlabel='sod', ylabel='hemo'>,
        <AxesSubplot: xlabel='pc', ylabel='hemo'>,
        <AxesSubplot: xlabel='rc', ylabel='hemo'>],
       [<AxesSubplot: xlabel='classification', ylabel='sg'>,
        <AxesSubplot: xlabel='hemo', ylabel='sg'>,
        <AxesSubplot: xlabel='sg', ylabel='sg'>,
        <AxesSubplot: xlabel='pcv', ylabel='sg'>,
        <AxesSubplot: xlabel='sod', ylabel='sg'>,
        <AxesSubplot: xlabel='pc', ylabel='sg'>,
        <AxesSubplot: xlabel='rc', ylabel='sg'>],
       [<AxesSubplot: xlabel='classification', ylabel='pcv'>,
        <AxesSubplot: xlabel='hemo', ylabel='pcv'>,
        <AxesSubplot: xlabel='sg', ylabel='pcv'>,
        <AxesSubplot: xlabel='pcv', ylabel='pcv'>,
        <AxesSubplot: xlabel='sod', ylabel='pcv'>,
        <AxesSubplot: xlabel='pc', ylabel='pcv'>,
        <AxesSubplot: xlabel='rc', ylabel='pcv'>],
       [<AxesSubplot: xlabel='classification', ylabel='sod'>,
        <AxesSubplot: xlabel='hemo', ylabel='sod'>,
        <AxesSubplot: xlabel='sg', ylabel='sod'>,
        <AxesSubplot: xlabel='pcv', ylabel='sod'>,
        <AxesSubplot: xlabel='sod', ylabel='sod'>,
        <AxesSubplot: xlabel='pc', ylabel='sod'>,
        <AxesSubplot: xlabel='rc', ylabel='sod'>],
       [<AxesSubplot: xlabel='classification', ylabel='pc'>,
        <AxesSubplot: xlabel='hemo', ylabel='pc'>,
        <AxesSubplot: xlabel='sg', ylabel='pc'>,
        <AxesSubplot: xlabel='pcv', ylabel='pc'>,
        <AxesSubplot: xlabel='sod', ylabel='pc'>,
        <AxesSubplot: xlabel='pc', ylabel='pc'>,
        <AxesSubplot: xlabel='rc', ylabel='pc'>],
       [<AxesSubplot: xlabel='classification', ylabel='rc'>,
        <AxesSubplot: xlabel='hemo', ylabel='rc'>,
        <AxesSubplot: xlabel='sg', ylabel='rc'>,
        <AxesSubplot: xlabel='pcv', ylabel='rc'>,
        <AxesSubplot: xlabel='sod', ylabel='rc'>,
        <AxesSubplot: xlabel='pc', ylabel='rc'>,
        <AxesSubplot: xlabel='rc', ylabel='rc'>]], dtype=object)
```



Spliting entire data into Train and Test Segments

In [155]:
```
# Spliting entire data into Train and Test Segments
rng = RandomState()

train = data.sample(frac=0.7, random_state=rng)
test = data.loc[~data.index.isin(train.index)]
```

In [156]:
```
train.head(5)
```

Out[156]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad
307	47.0	60.0	1.020	0.0	0.0	1	1	0	0	137.0	...	32	73	24	0	3	1
185	4.0	80.0	1.020	1.0	0.0	2	1	0	0	99.0	...	22	2	46	0	3	1
201	64.0	70.0	1.020	0.0	0.0	2	2	0	0	113.0	...	9	92	46	1	4	2
318	61.0	70.0	1.025	0.0	0.0	1	1	0	0	120.0	...	40	65	32	0	3	1
356	34.0	70.0	1.025	0.0	0.0	1	1	0	0	87.0	...	35	69	40	0	3	1

5 rows × 25 columns

In [157]:
```
test.head(5)
```

Out[157]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	a
4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	106.0	...	23	68	25	0	3	1	
6	68.0	70.0	1.010	0.0	0.0	2	1	0	0	100.0	...	24	92	46	0	3	1	
7	24.0	80.0	1.015	2.0	4.0	1	0	0	0	410.0	...	32	64	29	0	4	1	
9	53.0	90.0	1.020	2.0	0.0	0	0	1	0	70.0	...	17	18	16	1	4	1	
12	68.0	70.0	1.015	3.0	1.0	2	1	1	0	208.0	...	16	19	13	1	4	2	

5 rows × 25 columns

In [158]:
```
data = train[attributes]
data.head(5)
```

Out[158]:

	classification	hemo	sg	pcv	sod	pc	rc
307	2	13.6	1.020	32	150.0	1	24
185	0	12.0	1.020	22	138.0	1	46
201	0	7.9	1.020	9	137.0	2	46
318	2	17.4	1.025	40	137.0	1	32
356	2	17.1	1.025	35	144.0	1	40

In [159]:
```
label = data['classification']
data = data.drop('classification',axis=1)
data.head(5)
```

Out[159]:

	hemo	sg	pcv	sod	pc	rc
307	13.6	1.020	32	150.0	1	24
185	12.0	1.020	22	138.0	1	46
201	7.9	1.020	9	137.0	2	46
318	17.4	1.025	40	137.0	1	32
356	17.1	1.025	35	144.0	1	40

In [160]:
```
test.to_csv('test.csv')
```

Split the data into train and test sets

In [161]:
```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(data, label, test
```

In [162]:
```
def display_results(y_test,y_pred):
    # Print the Confusion Matrix and slice it into four pieces
    cm = confusion_matrix(y_test,y_pred)
    # visualize confusion matrix with seaborn heatmap
    cm_matrix = pd.DataFrame(data=cm)
    print("Model Accuracy:",accuracy_score(y_test,y_pred))
    sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

implementing the Machine Learning Algorithms

1. Linear Regression

In [163]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
# Linear Regression
reg_model = LinearRegression()
reg_model.fit(X_train, y_train)
reg_pred = reg_model.predict(X_test)
```

In [164]:
```
# Calculate metrics
mse = mean_squared_error(y_test, reg_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, reg_pred)
r2 = r2_score(y_test, reg_pred)

# Print metrics
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R²):", r2)
```

```
Mean Squared Error (MSE): 0.3737331558245581
Root Mean Squared Error (RMSE): 0.6113371866855133
Mean Absolute Error (MAE): 0.5001356132694907
R-squared (R²): 0.5832051292084587
```

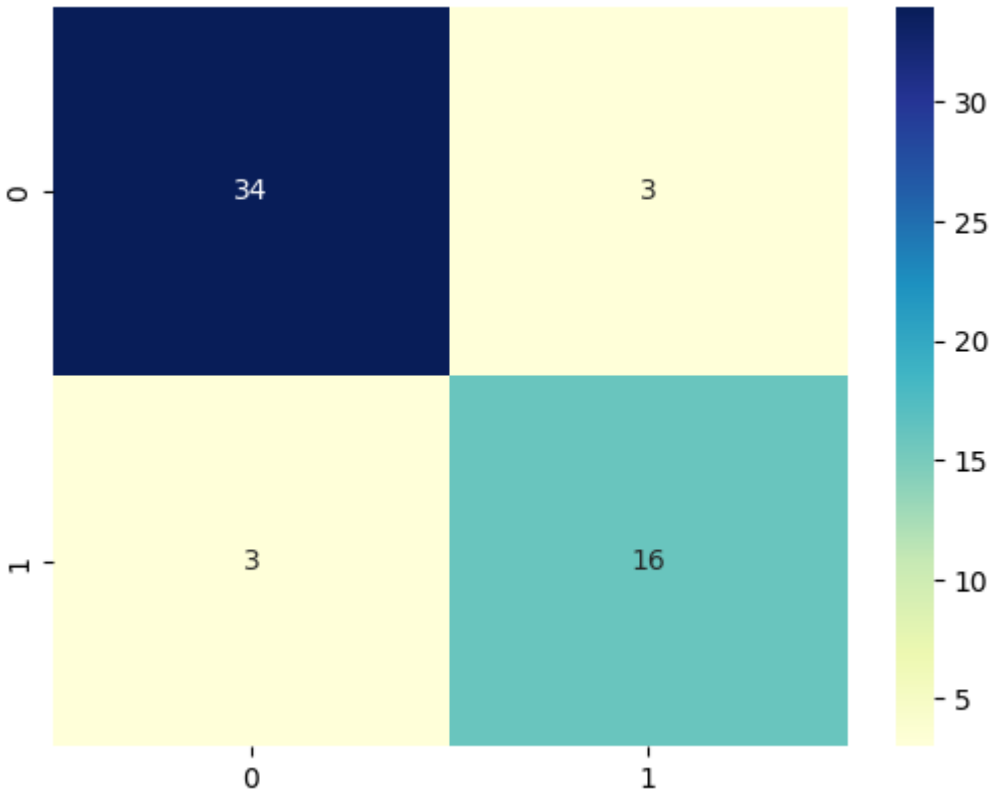
2. Logistic Regression

In [165]:
```
# Logistic Regression
clf_model = LogisticRegression(random_state=0, max_iter=2000, solver=
clf_model.fit(X_train, y_train)
clf_pred = clf_model.predict(X_test)
# Calculate accuracy and display results
clf_accuracy = accuracy_score(y_test, clf_pred)
print("Logistic Regression Accuracy:", clf_accuracy)
print(classification_report(y_test, clf_pred))
```

```
Logistic Regression Accuracy: 0.8928571428571429
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        37
           2       0.84      0.84      0.84        19

    accuracy                           0.89        56
   macro avg       0.88      0.88      0.88        56
weighted avg       0.89      0.89      0.89        56
```

In [166]:
```
display_results(y_test,clf_pred)
```

```
Model Accuracy: 0.8928571428571429
```



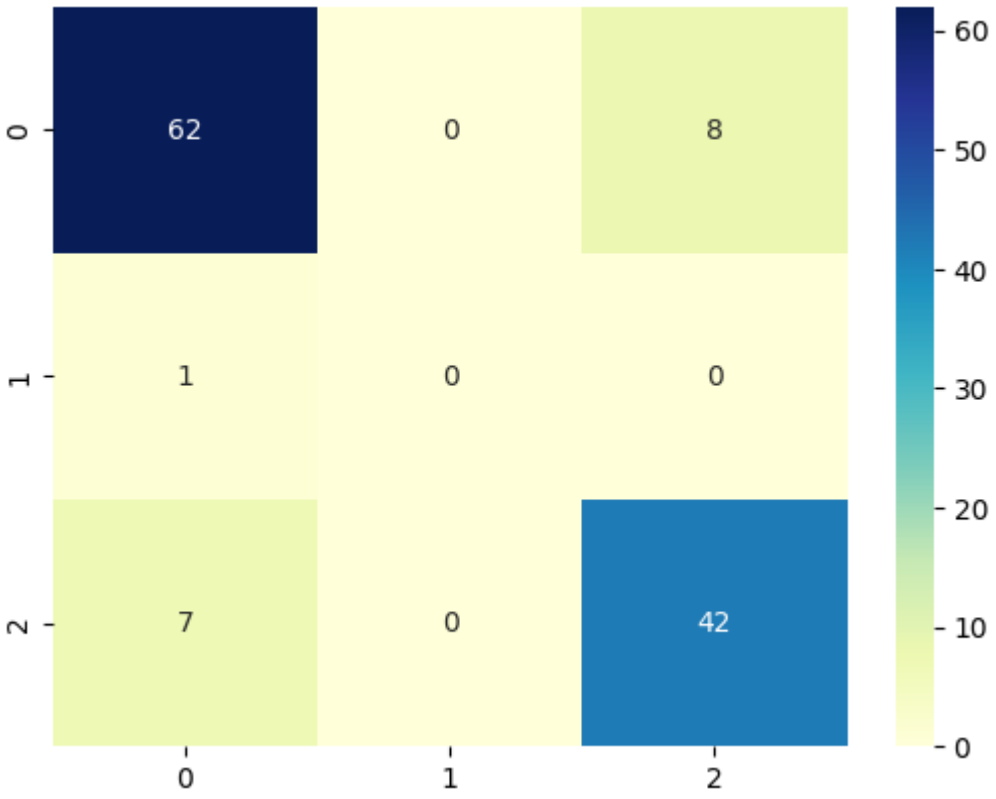
In [167]:
```
#tesing on test split
test_label = test['classification']
test_data = test[attributes].drop('classification',axis=1)
test_data.head(5)
```

Out[167]:

	hemo	sg	pcv	sod	pc	rc
4	11.6	1.010	23	138.0	1	25
6	12.4	1.010	24	104.0	1	46
7	12.4	1.015	32	138.0	0	29
9	9.5	1.020	17	114.0	0	16
12	9.7	1.015	16	138.0	1	13

In [168]:
```
predict = clf_model.predict(test_data)
display_results(test_label,predict)
```

```
Model Accuracy: 0.8666666666666667
```



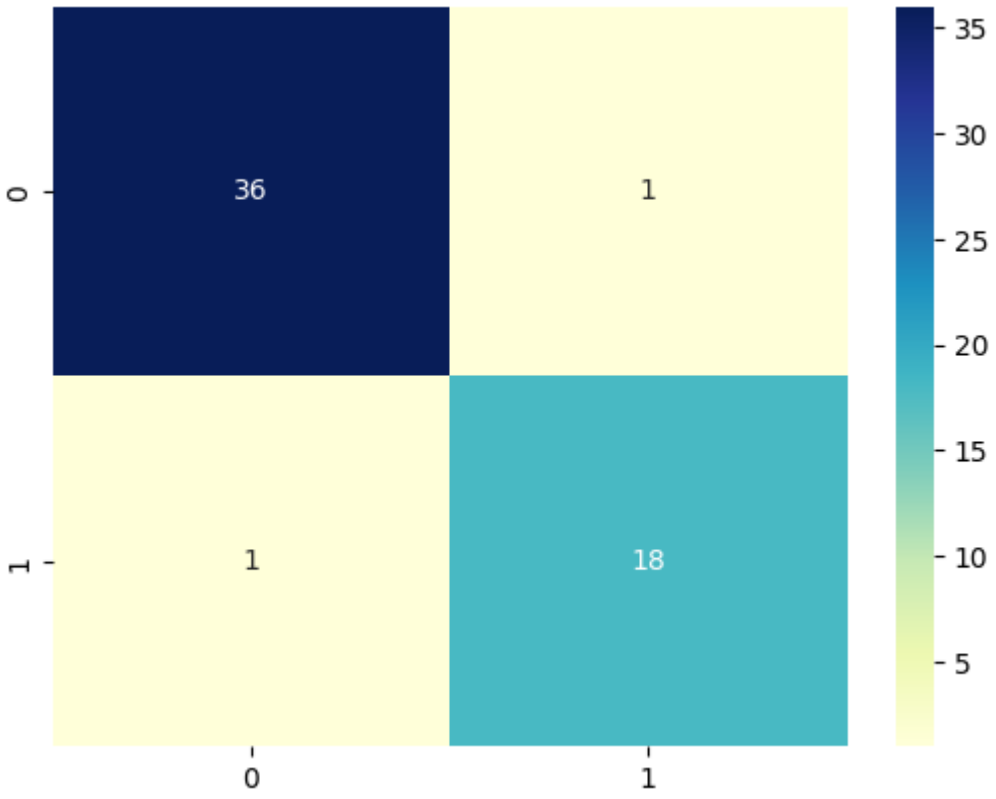
3. Navie Bayes

In [169]:
```
# Naive Bayes
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_pred = nb_model.predict(X_test)
# Calculate accuracy and display results
nb_accuracy = accuracy_score(y_test, nb_pred)
print("Naive Bayes Accuracy:", nb_accuracy)
print(classification_report(y_test, nb_pred))
```

```
Naive Bayes Accuracy: 0.9642857142857143
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        37
           2       0.95      0.95      0.95        19

    accuracy                           0.96        56
   macro avg       0.96      0.96      0.96        56
weighted avg       0.96      0.96      0.96        56
```

In [170]:
```
display_results(y_test,nb_pred)
```

```
Model Accuracy: 0.9642857142857143
```



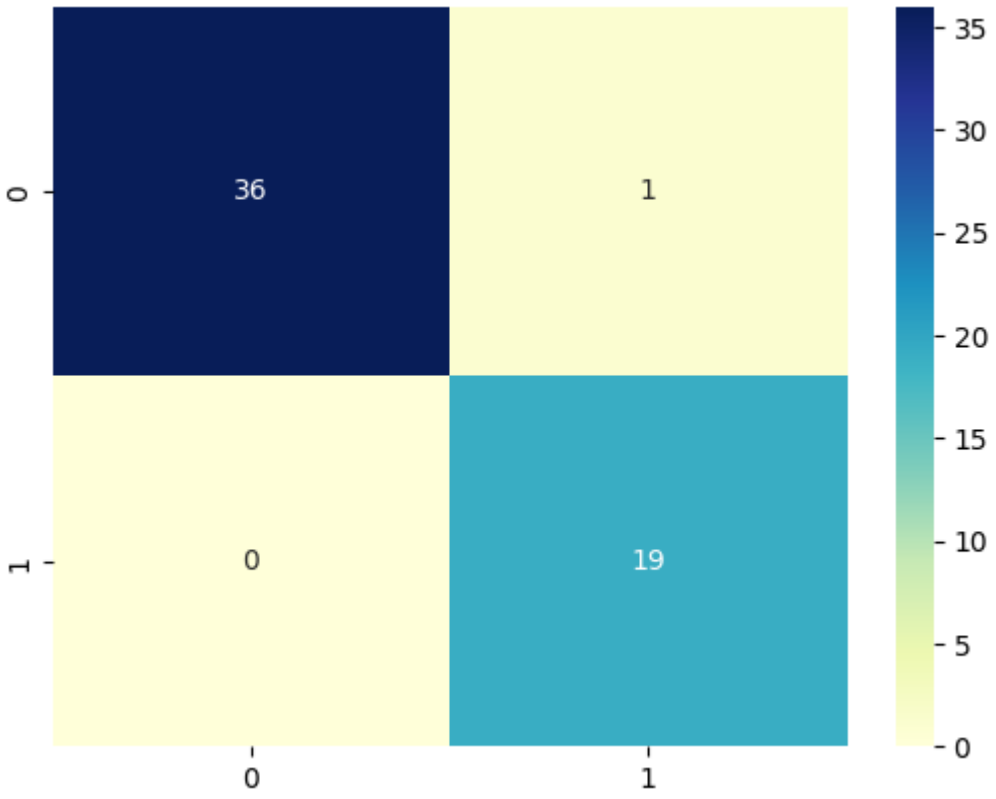
4. Random Forest

In [171]:
```
# Random Forest
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)
# Calculate accuracy and display results
rf_accuracy = accuracy_score(y_test, rf_pred)
print("Random Forest Accuracy:", rf_accuracy)
print(classification_report(y_test, rf_pred))
```

```
Random Forest Accuracy: 0.9821428571428571
              precision    recall  f1-score   support

           0       1.00      0.97      0.99        37
           2       0.95      1.00      0.97        19

    accuracy                           0.98        56
   macro avg       0.97      0.99      0.98        56
weighted avg       0.98      0.98      0.98        56
```

In [172]:
```
display_results(y_test,rf_pred)
```

```
Model Accuracy: 0.9821428571428571
```



5. Support Vector Machine (SVM)

In [173]:

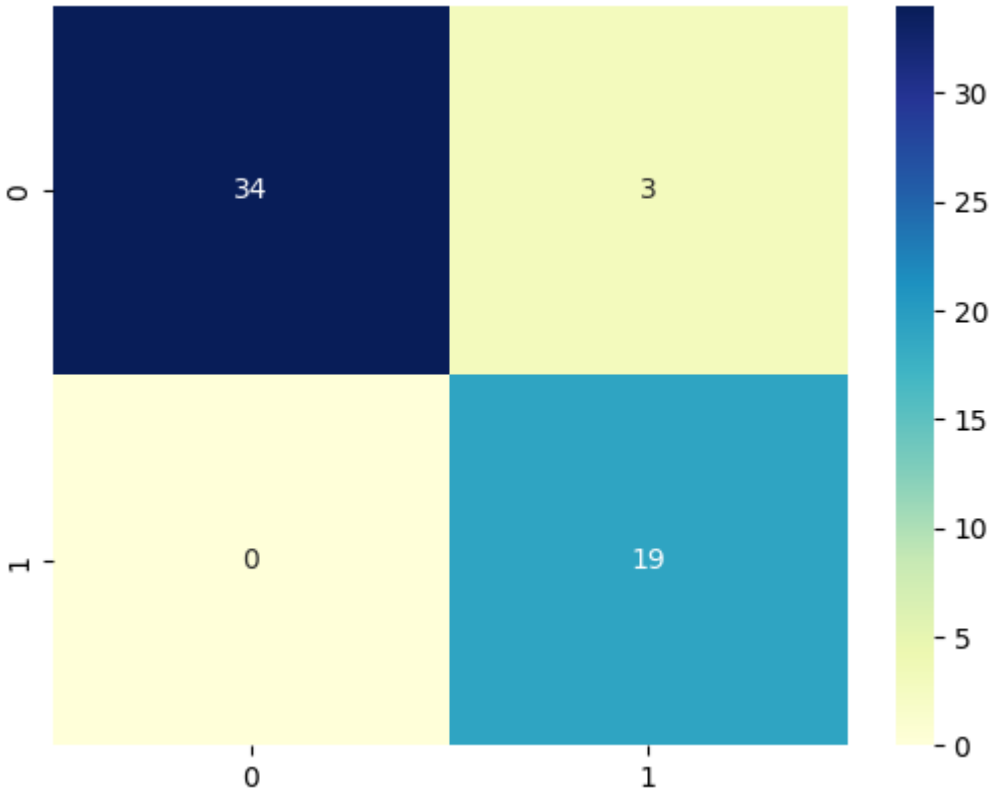
```
from sklearn.svm import SVC
from sklearn import svm

svm_model = svm.SVC(kernel='linear')
svm_model.fit(X_train,y_train)
svm_model_score = round(svm_model.score(X_train, y_train) * 100, 2)
svm_pred = svm_model.predict(X_test)
# Calculate accuracy and display results
svm_accuracy = accuracy_score(y_test, svm_pred)
print("Support Vector Machine Accuracy:",svm_accuracy)
print(classification_report(y_test, svm_pred))
```

```
Support Vector Machine Accuracy: 0.9464285714285714
              precision    recall  f1-score   support

           0       1.00      0.92      0.96        37
           2       0.86      1.00      0.93        19

    accuracy                           0.95        56
   macro avg       0.93      0.96      0.94        56
weighted avg       0.95      0.95      0.95        56
```

In [174]:

```
display_results(y_test,svm_pred)
```

```
Model Accuracy: 0.9464285714285714
```



Comparing all the models

In [175]:

```
# Comparing all the models
models = pd.DataFrame({
    'Model': [ 'Linear Regression', 'Logistic Regression', 'Naive Bay
    'Model Accuracy': [r2, clf_accuracy, nb_accuracy, rf_accuracy, sv
models.sort_values(by='Model Accuracy', ascending=False)
```

Out[175]:

	Model	Model Accuracy
3	Random Forest	0.982143
2	Naive Bayes	0.964286
4	SVM Model	0.946429
1	Logistic Regression	0.892857
0	Linear Regression	0.583205

In [ ]:

```
1
```