

## Main Objective:

The primary objective of this analysis is to develop a predictive model that can determine the most appropriate drug for a future patient suffering from a specific illness. Each patient in the dataset has been treated with one of five medications: Drug A, Drug B, Drug C, Drug X, or Drug Y. By analyzing the relationships between the patients' features—such as age, sex, blood pressure, Na\_to\_K and cholesterol levels—and their respective drug responses, we aim to build a model that can accurately recommend the best medication for new patients based on their individual characteristics. This predictive capability is crucial for optimizing treatment strategies and improving patient outcomes.

- Brief description of the data set you chose, a summary of its attributes, and an outline of what you are trying to accomplish with this analysis.

## Dataset Description:

The dataset I am using in analysis contains medical information for a group of patients, each represented by a row. The features include Age, Sex, Blood Pressure, Na\_to\_K, and Cholesterol levels, with the target variable being the specific drug that each patient responded to. The details of each feature are as follows:

- **Age:** The age of the patient.
- **Sex:** The gender of the patient.
- **Blood Pressure:** The blood pressure category of the patient, typically categorized into 'Low', 'Normal', or 'High'.
- **Cholesterol:** The cholesterol level of the patient, categorized as 'Normal' or 'High'.
- **Na\_to\_K:** represents the ratio of sodium (Na) to potassium (K) levels in the patient's body.
- **Drug:** The drug to which the patient responded, with options including Drug A, Drug B, Drug C, Drug X, and Drug Y.

This dataset serves as a valuable resource for predicting which drug might be most effective for a new patient based on their medical profile.

### **Exploration Data Analysis:**

The dataset contains 200 rows and 6 columns, with the following data types for each feature:

Age: int64 (Numerical)

Sex: object (Categorical)

BP (Blood Pressure): object (Categorical)

Cholesterol: object (Categorical)

Na\_to\_K: float64 (Numerical)

Drug: object (Categorical, Target variable)

### **Given the nature of the data:**

**Categorical Features:** The features Sex, BP, and Cholesterol are categorical. These need to be encoded into numerical values to be used effectively in modeling. Common techniques include one-hot encoding or label encoding.

**Target Variable:** The Drug column, which is also categorical, will be converted into numerical values, typically using label encoding, to facilitate model training.

To know if we need to scale the data or not, we have many options. One of them is using **histogram** or we can use **skew()** to find the skew values based on these values we can say if they are skewed or not.

We will be fixing some threshold usually 0.5, if

$\text{abs}(\text{skew val}) > 0.5$  - skewed

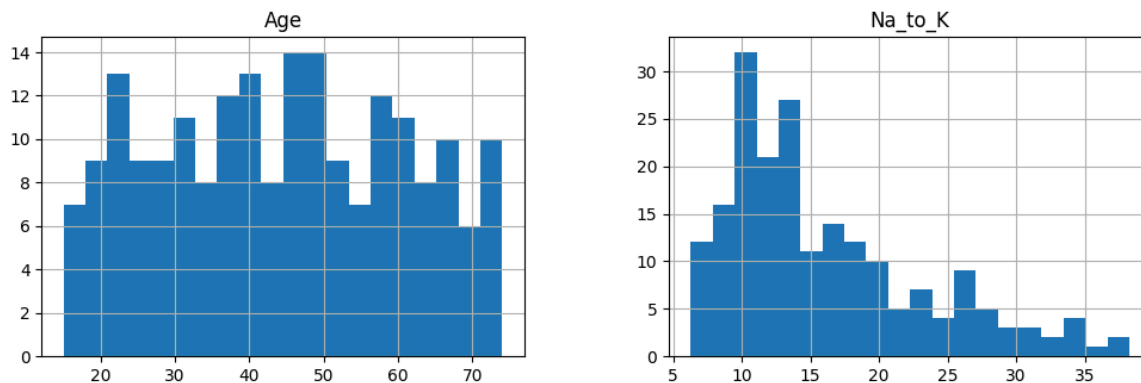
$\text{abs}(\text{skew val}) < 0.5$  - not skewed

```

: drug_df.hist(bins=20,figsize=(12,8))

: array([[<Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Na_to_K'}>],
        [<Axes: title={'center': 'Drug'}>, <Axes: >]], dtype=object)

```



From the image we can see that features 'Age' , 'Na\_to\_K' need to be scaled to ensure all features contribute equally to the model . Standardization (scaling to have mean 0 and variance 1) or Normalization (scaling to a range, typically [0, 1]) are commonly used techniques .

## Data PreProcessing:

Scaled the numerical features , so that all features contribute equally to the model . I used **StandardScaler** to scale the numerical data. For the categorical features, I applied **LabelEncoder** to convert them into numerical values .

We have scaled the numerical features : 'Age' , 'Na\_to\_K'

```

]: numerical_columns=['Age','Na_to_K']
   scaler=StandardScaler()
   drug_df[numerical_columns]=scaler.fit_transform(drug_df[numerical_columns])

```

We have done label encoding on the categorical columns : 'BP' , 'Cholesterol' , 'sex' and the target variable 'Drug' .

```
le_bp.classes_
```

```
array(['HIGH', 'LOW', 'NORMAL'], dtype='<U6')
```

```
drug_df['BP'].value_counts()
```

```
BP
0    77
1    64
2    59
Name: count, dtype: int64
```

```
le_chol.classes_
```

```
array(['HIGH', 'NORMAL'], dtype='<U6')
```

```
drug_df['Cholesterol'].value_counts()
```

```
Cholesterol
0    103
1     97
Name: count, dtype: int64
```

```
le_sex.classes_
```

```
array(['F', 'M'], dtype='<U1')
```

```
drug_df['Sex'].value_counts()
```

```
Sex
1    104
0     96
Name: count, dtype: int64
```

```
le_y.classes_
```

```
array(['drugA', 'drugB', 'drugC', 'drugX', 'drugY'], dtype=object)
```

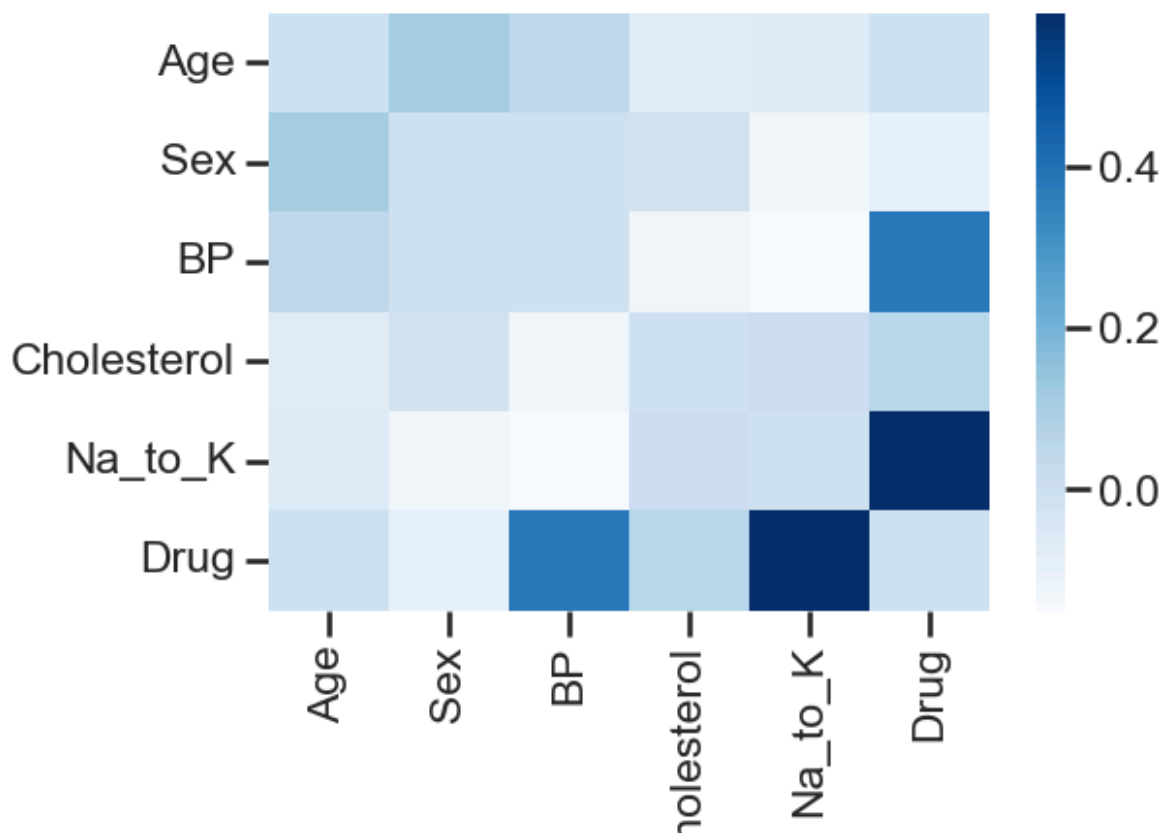
```
drug_df['Drug'].value_counts()
```

```
Drug
4     91
3     54
0     23
2     16
1     16
Name: count, dtype: int64
```

Additionally, I computed the correlation matrix using Pearson correlation (`corr()`) to analyze the relationships between the features. The correlation coefficients revealed that the `Drug` and `Na_to_K` features are positively correlated, indicating a potential relationship that might influence the model's predictions.

```
fig,ax=plt.subplots(figsize=(6,4))  
sns.heatmap(correlation_matrix,cmap='Blues')
```

<Axes: >



```
)]: correlation_matrix['Drug'].sort_values() |
```

```
)]: Sex          -0.098573  
     Age          -0.004828  
     Drug          0.000000  
     Cholesterol  0.055629  
     BP           0.372868  
     Na_to_K      0.589120  
     Name: Drug, dtype: float64
```

From the above image we can see that 'Na\_to\_K', 'BP' are positively correlated with 'Drug' .

Next, the dataset was split into two subsets: training data and testing data. The training data is used to train the model, while the testing data evaluates the model's performance on unseen data. To ensure that the distribution of the target variable remains consistent between the training and testing sets, the `stratify` attribute was applied in the `train_test_split` function.. This technique is particularly important in classification tasks as it preserves the class proportions in both subsets, leading to a more reliable assessment of the model's accuracy. By maintaining the same ratio of data, we ensure that the model is exposed to a representative sample during training and is tested against a realistic scenario, ultimately leading to a more robust and generalizable model.

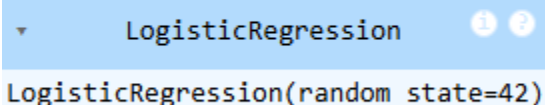
## Model Building and Evaluation:

### Logistic Regression:

LR is a commonly used baseline model in classification tasks due to its simplicity and interpretability. It provides a good starting point to understand the relationship between the features and the target variable .

By starting with LR, we can easily compare its performance with more complex models (like decision trees, k-NN, and stacking classifiers). This comparison helps in understanding whether the added complexity of other models significantly improves performance.

```
lr=LogisticRegression(random_state=42)
lr.fit(X_train,Y_train)
```



```
LogisticRegression(random_state=42)
```

```

Logistic Regression :
              precision    recall  f1-score   support

     0         0.64         1.00         0.78         7
     1         1.00         0.80         0.89         5
     2         0.00         0.00         0.00         5
     3         0.80         1.00         0.89        16
     4         1.00         0.93         0.96        27

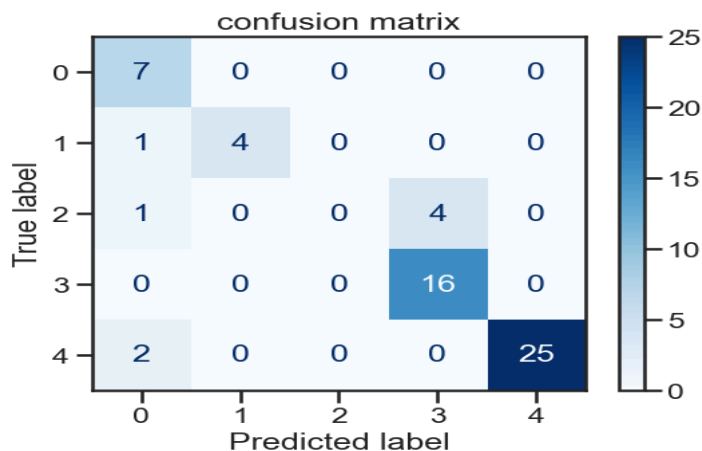
 accuracy          0.87         60
 macro avg         0.69         0.75         0.70         60
 weighted avg      0.82         0.87         0.83         60

```

We have fitted the training data to LR without any penalty or we can say regularization . We have printed the classification report , and we have found that the precision , recall for 'drugC' is 0 . The accuracy score is around 0.86 only

Confusion matrix :

We have also plotted the confusion matrix , from that we can say that drugC is being misclassified .



So now we have added a penalty to Lr which is L1(Lasso) and L2(Ridge).

we can see that after doing regularization there's an increase in accuracy\_score .

```

Logistic Regression (penalty l1):
              precision    recall  f1-score   support

     0         1.00      0.86      0.92         7
     1         1.00      0.60      0.75         5
     2         0.50      1.00      0.67         5
     3         1.00      0.88      0.93        16
     4         1.00      1.00      1.00        27

 accuracy          0.92         60
 macro avg          0.90      0.87      0.85         60
 weighted avg       0.96      0.92      0.92         60

```

```

Logistic Regression (penalty l2):
              precision    recall  f1-score   support

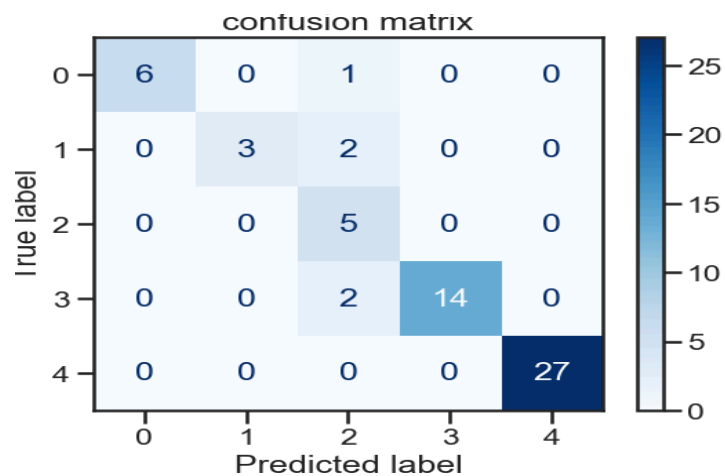
     0         1.00      0.86      0.92         7
     1         1.00      0.60      0.75         5
     2         0.50      1.00      0.67         5
     3         1.00      0.88      0.93        16
     4         1.00      1.00      1.00        27

 accuracy          0.92         60
 macro avg          0.90      0.87      0.85         60
 weighted avg       0.96      0.92      0.92         60

```

And confusion matrix after adding penalty :

Even with the addition of a penalty, some misclassifications remain. However, the model performs better in classifying DrugC compared to Logistic Regression.





## KNearestNeighbors(KNN):

In the KNN model, classification is based on the proximity of data points to their nearest neighbors. We utilized GridSearchCV to identify the optimal hyperparameters, ensuring the model's best performance.

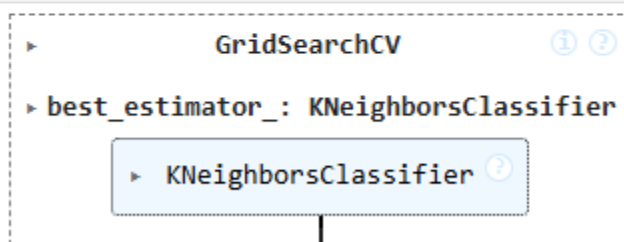
The hyper parameters are:

N\_neighbors : no of neighbors to check

Distance metric : 'Euclidean' , 'Manhattan'

```
knn=KNeighborsClassifier()  
param_grid={'n_neighbors':[i+1 for i in range(20)], 'weights':['distance']}  
gr_knn=GridSearchCV(knn,param_grid=param_grid,scoring='accuracy')
```

```
gr_knn.fit(X_train,Y_train)
```



After printing the classification report, we observed that while this model classifies better than the basic Logistic Regression model, it does not outperform the Logistic Regression model with a penalty.

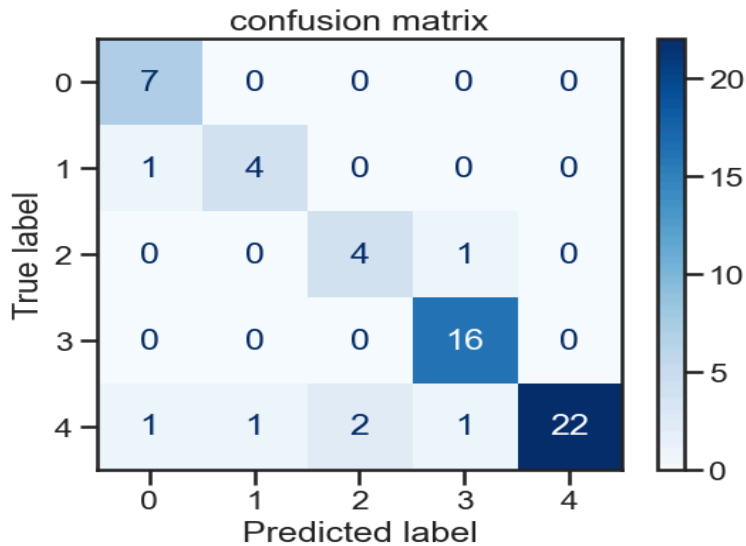
```
y_pred_gr_knn=gr_knn.predict(X_test)  
print(classification_report(Y_test,y_pred_gr_knn))
```

	precision	recall	f1-score	support
0	0.78	1.00	0.88	7
1	0.80	0.80	0.80	5
2	0.67	0.80	0.73	5
3	0.89	1.00	0.94	16
4	1.00	0.81	0.90	27
accuracy			0.88	60
macro avg	0.83	0.88	0.85	60
weighted avg	0.90	0.88	0.88	60

Confusion matrix :

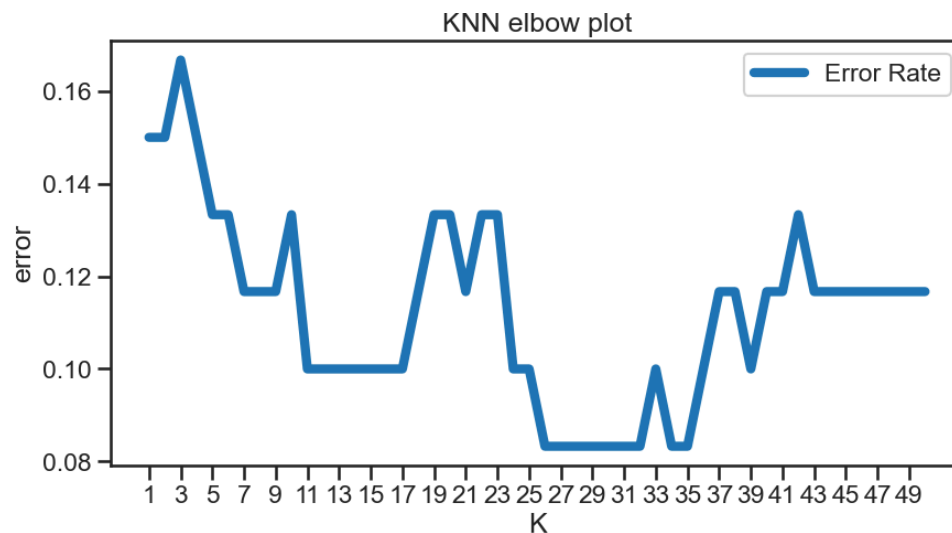
The confusion matrix indicates that the classes are being classified more accurately compared to the basic Logistic Regression model. However, when compared to the Logistic Regression model with a penalty, there are still more misclassifications with the KNN model.

```
plot_confusion_matrix(Y_test,y_pred_gr_knn)
```



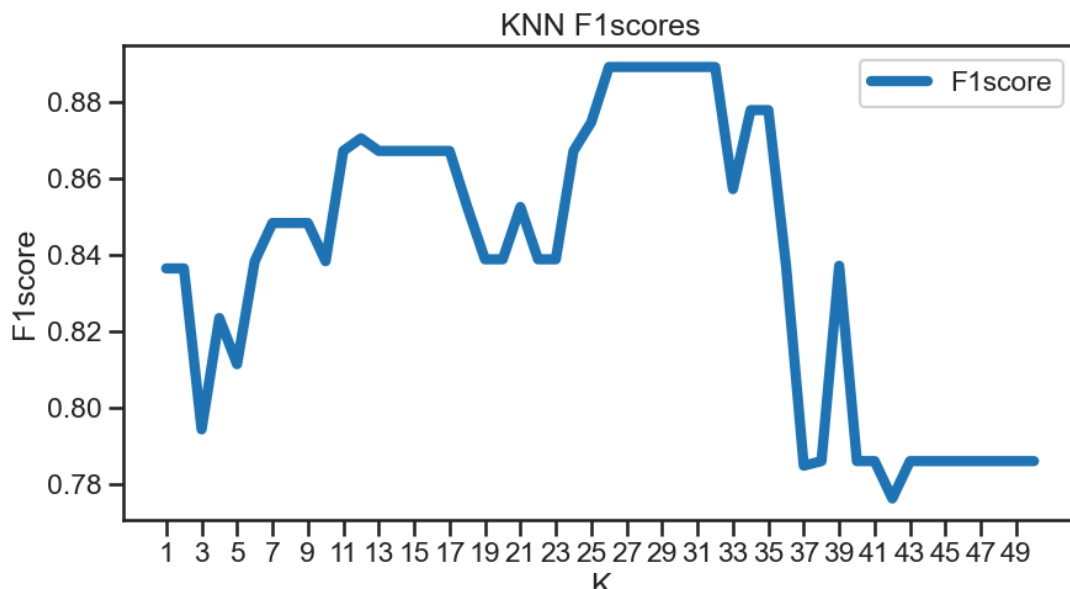
KNN Elbow Curve :

Below, the value of k is varied from 1 to 50, and it is observed that the error rate is minimized at k=25.



F1 scores vs K:

It was observed that the F1 score is highest when  $k=25$ .



### Decision Tree:

The Decision Tree (DT) model was chosen for its interpretability and ability to handle both numerical and categorical data effectively. The model splits the data based on feature values, creating branches that represent decision paths leading to the final prediction. Hyperparameter tuning, such as adjusting the maximum depth and minimum samples per leaf, was performed to avoid overfitting and improve the model's generalization to unseen data.

This makes DT an effective choice for understanding the decision-making process and identifying the most influential features in predicting the target variable. They provide a clear and intuitive way to make decisions based on data by modeling the relationships between different variables.

The hyper parameters are:

max\_depth : maximum depth of the tree

max\_features: max no of features the model should consider when splitting the node

```
] param_grid={'max_depth':range(1,DT.tree_.max_depth,2),'max_features':range(1,len(DT.feature_importances_),2)}
gr_dt=GridSearchCV(DecisionTreeClassifier(random_state=42),param_grid=param_grid,scoring='accuracy',n_jobs=-1)
```

```
] gr_dt.fit(X_train,Y_train)
```

```
] ▶ GridSearchCV ⓘ ?
  ▶ best_estimator_: DecisionTreeClassifier
    ▶ DecisionTreeClassifier ?
```

```
] gr_dt.best_params_
```

```
] {'max_depth': 3, 'max_features': 1}
```

```
print('Decision Tree :')
get_accuracy(X_train,Y_train,X_test,Y_test,gr_dt)
```

```
Decision Tree :
{'train_accuracy': 0.7, 'test_accuracy': 0.7}
```

We observe that both the training and testing accuracy scores are low, and the classification report indicates poor performance. Decision trees are prone to overfitting, so it is important to implement pruning or perform hyperparameter tuning to address this issue.

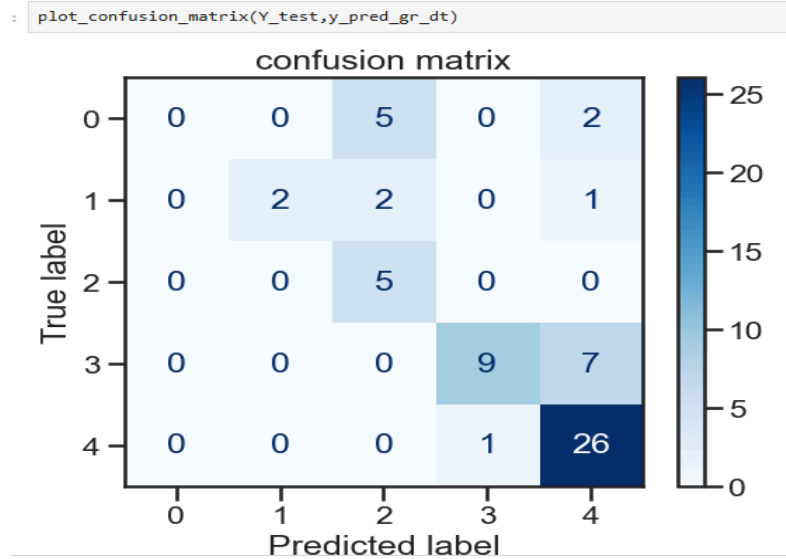
```
19]: y_pred_gr_dt=gr_dt.predict(X_test)
    print(classification_report(Y_test,y_pred_gr_dt))
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	7
1	1.00	0.40	0.57	5
2	0.42	1.00	0.59	5
3	0.90	0.56	0.69	16
4	0.72	0.96	0.83	27
accuracy			0.70	60
macro avg	0.61	0.59	0.54	60
weighted avg	0.68	0.70	0.65	60

Confusion matrix:

From the classification report, we can see that drugs A and X are highly misclassified, along with several other classes. The performance is worse

compared to logistic regression.



## Random Forest:

Random Forest is an ensemble learning method that uses decision trees as base classifiers. By aggregating the predictions from multiple decision trees, it aims to improve accuracy and reduce overfitting. Unlike a standard bagging classifier, Random Forest introduces randomness by selecting a subset of features for splitting at each node.

While a single decision tree tends to overfit on this dataset, Random Forest mitigates this risk by averaging the results from multiple trees. Additionally, by introducing randomness in feature selection at each split, Random Forest further reduces the likelihood of overfitting, leading to a more robust and reliable model .

```
: RF=RandomForestClassifier(random_state=42)

: param_grid={'n_estimators':[2*(n+1) for n in range(20)], 'max_depth':[2*(i+1) for i in range(10)], 'max_features':['sqrt','log2']}

: gr_rf=GridSearchCV(estimator=RF,param_grid=param_grid,scoring='accuracy')
: gr_rf.fit(X_train,Y_train)

: 
  > GridSearchCV ① ②
  > best_estimator_: RandomForestClassifier
    > RandomForestClassifier ③

: gr_rf.best_score_

: np.float64(1.0)
```

```

: gr_rf.best_params_

: {'max_depth': 6, 'max_features': 'sqrt', 'n_estimators': 38}

: print('Random Forest:')
  get_accuracy(X_train,Y_train,X_test,Y_test,gr_rf)

Random Forest:
: {'train_accuracy': 1.0, 'test_accuracy': 0.9833333333333333}

```

From the above, we can see that it has a high accuracy score on both the training and testing data.

Classification report:

From the classification report, we can see that all classes have high precision and recall scores. The accuracy score is 0.98, which indicates excellent performance .

```

: y_pred_gr_rf=gr_rf.predict(X_test)
  print(classification_report(Y_test,y_pred_gr_rf))

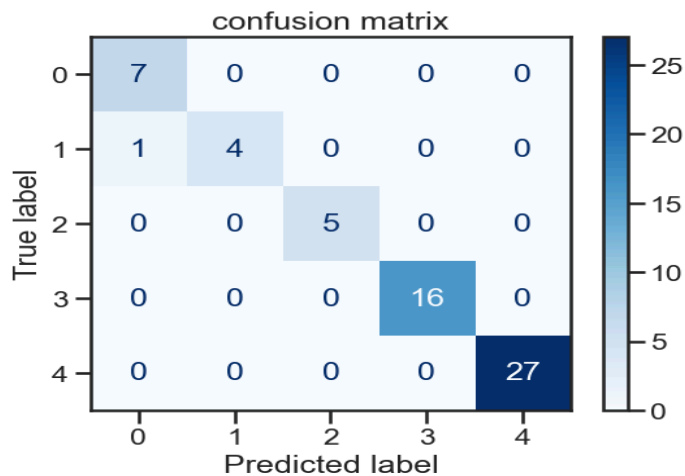
```

	precision	recall	f1-score	support
0	0.88	1.00	0.93	7
1	1.00	0.80	0.89	5
2	1.00	1.00	1.00	5
3	1.00	1.00	1.00	16
4	1.00	1.00	1.00	27
accuracy			0.98	60
macro avg	0.97	0.96	0.96	60
weighted avg	0.99	0.98	0.98	60

Confusion matrix :

From the confusion matrix, we can see that the classes are classified correctly.

```
: plot_confusion_matrix(Y_test,y_pred_gr_rf)
```



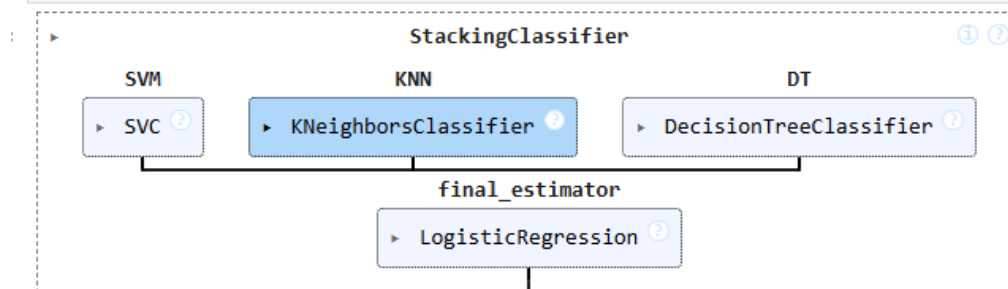
## Stacking Classifier:

A stacking classifier is an ensemble learning technique that combines multiple individual models to create a more robust and accurate predictive model. The key idea behind stacking is to leverage the strengths of individual models to improve overall performance .

Here I have used 'SVM' , 'KNN','DT' as base classifiers and 'LogisticRegression' as final estimator .

```
: from sklearn.ensemble import StackingClassifier
estimators=[('SVM',SVC(random_state=42)),('KNN',KNeighborsClassifier()),('DT',DecisionTreeClassifier())]
stack_class=StackingClassifier(estimators=estimators,final_estimator=LogisticRegression())
```

```
: stack_class.fit(X_train,Y_train)
```



```
: y_pred_stack=stack_class.predict(X_test)
```

```

: print('stacking classifier:')
  get_accuracy(X_train,Y_train,X_test,Y_test,stack_class)

stacking classifier:
: {'train_accuracy': 1.0, 'test_accuracy': 0.9666666666666667}

```

From above we can see that the accuracy score is 1 for training data and testing data is also nearly 1 .

Classification report:

```

y_pred_stack=stack_class.predict(X_test)
print(classification_report(Y_test,y_pred_stack))

```

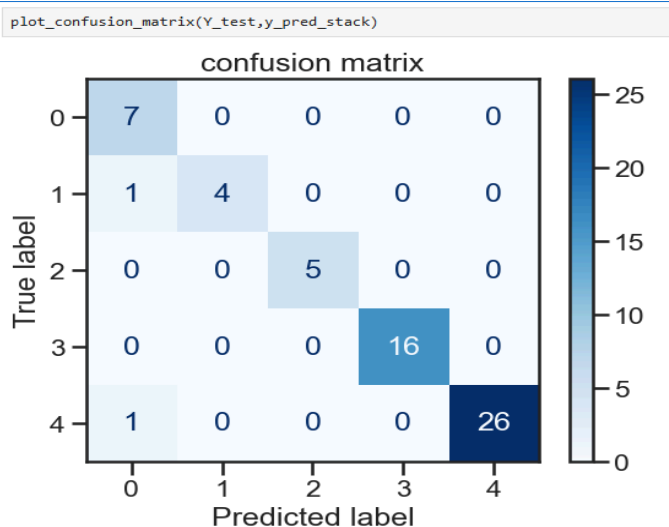
	precision	recall	f1-score	support
0	0.78	1.00	0.88	7
1	1.00	0.80	0.89	5
2	1.00	1.00	1.00	5
3	1.00	1.00	1.00	16
4	1.00	0.96	0.98	27
accuracy			0.97	60
macro avg	0.96	0.95	0.95	60
weighted avg	0.97	0.97	0.97	60

From the classification report, we can see that the precision for drug A is low, although the overall accuracy score is 0.98.

Confusion matrix:

One of the rows is being predicted as drug A instead of drug B, and another row is misclassified as drug A when it should be predicted as drug Y.





Based on the performance analysis of the different classifier models, **Random Forest** emerges as the recommended final model.

Random Forest achieved the highest accuracy score of 0.98 on both the training and testing datasets. This indicates that it consistently performs well in classifying the target variable

The classification report for Random Forest shows high precision and recall across all classes, including **drugC**, which was a challenge for the other models. This suggests that Random Forest effectively handles class imbalances.

Unlike the Decision Tree model, which suffered from overfitting, Random Forest mitigates this risk by averaging predictions from multiple trees and introducing randomness in feature selection. This makes it more generalizable to unseen data.

While Random Forest is less interpretable compared to individual models like Decision Trees, its superior performance in terms of accuracy and handling of class imbalances outweighs this drawback. The model's performance can be understood through feature importance scores, which provide insights into which features are most influential in the predictions.

The **Stacking Classifier** demonstrated high accuracy and was effective in combining multiple models to improve performance. However, it encountered issues with misclassifying other classes as **drugA**.

## Summary:

**Logistic Regression** : provided a solid baseline but had limited performance without regularization. Adding L1 and L2 regularization improved results but did not match the robustness of Random Forest.

**KNN** : The K-Nearest Neighbors (KNN) model is easy to interpret, understand, and implement. However, it misclassified **drugY**, has a lower accuracy score compared to Random Forest, and exhibits low precision for some classes.

**Decision Tree** : The Decision Tree model is interpretable due to its hierarchical structure, but it shows low accuracy on both training and testing data, with very low precision and recall scores. Decision trees are prone to overfitting .

**Random forest** : Aggregating the results of multiple decision trees and introducing randomness at each split helps address the limitations of individual decision trees. This approach is where Random Forest comes into the picture, combining the predictions of numerous decision trees to improve accuracy and reduce overfitting. By using the Random Forest model, we achieved high accuracy, precision, and recall scores..

Stacking combines predictions from multiple models to leverage their individual strengths, often resulting in better overall performance than any single model. It can improve accuracy and robustness by aggregating diverse learning patterns. By using a diverse set of base models, stacking can reduce the risk of overfitting and improve generalization to unseen data. The final estimator learns to make the best use of the base models' outputs, leading to more reliable predictions.

We could use stacking classifiers for this task, but it misclassified the **drugA** class. Therefore, I recommend Random Forest as the better model.

## Future enhancements:

- Enhancing feature engineering to potentially improve model performance. Considering adding or creating new features that could provide additional insights or better represent the underlying patterns in the data .

- Conducting detailed analysis of feature importance using models like random forest or gradient boosting.
- Comparing these models with other advanced ensemble methods.
- Implementing more extensive cross-validation techniques to ensure the robustness and reliability of the models.