YILDIZ TECHNICAL UNIVERSITY
FACULTY OF MECHANICAL ENGINEERING
DEPARTMENT OF INDUSTRIAL ENGINEERING

Advisor
Prof. Dr. ALEV TAŞKIN

END4991 INDUSTRIAL ENGINEERING DESIGN II
FINAL REPORT

İlyas Uğur Vural 19061069
Özlem Yüksel Bilir 19061070
Nihal Diler 19069606
Özgür Gümüş 19061062
Melis Kamacıoğlu 18061034

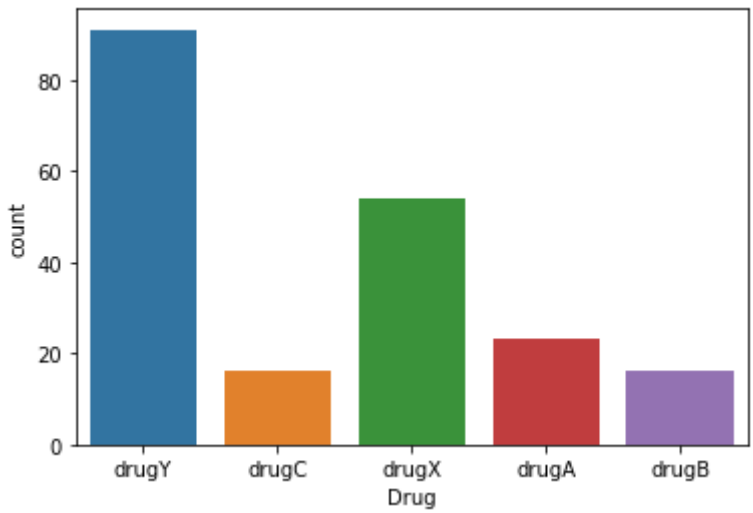
2022-2023 Fall

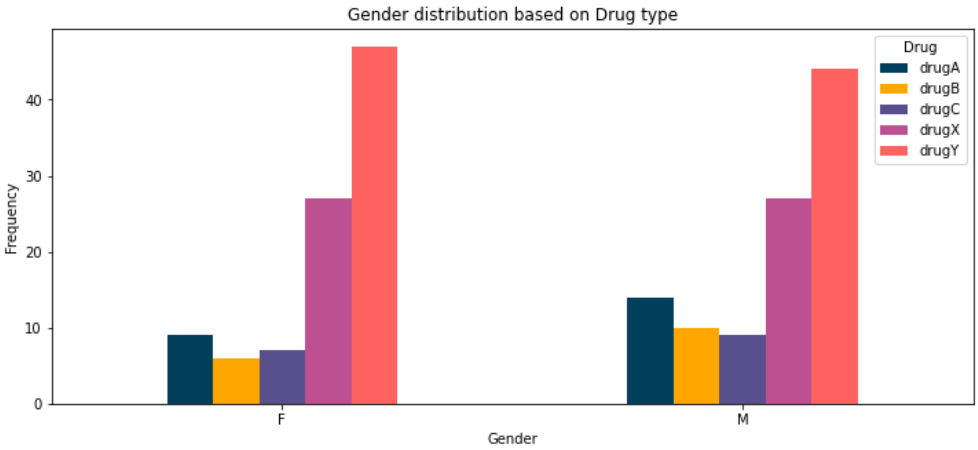
CONTENT

1.INTRODUCTION

Dataset: The dataset with 200 rows contains information that effects drug type such as Age, Sex, BP, Cholesterol levels, Na to Potassium Ratio. Age shows the age of the patient, sex shows the gender. BP is for blood pressure and it has three values; high, normal and low. Na to potassiom rate is the rate of Sodium to Potassiom in patient's blood.

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY





2.KNN

2.1 Description of the KNN methodology

KNN stands for "k-nearest neighbors". It is a type of supervised machine learning algorithm used for classification and regression. Given a new observation, it finds the k-number of training examples that are closest to it in feature space, and then it assigns the class label based on the majority class among those k-nearest examples. It's a simple yet powerful algorithm that can be used for both classification and regression problems.

2.2 Implementation of KNN in Python

1- Collect and preprocess the data: This step involves collecting the data that will be used to train and test the algorithm. The data may need to be cleaned and preprocessed to ensure that it is in a format that can be used by the algorithm.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split,  cross_val_score, StratifiedKFold, KFold, RepeatedKFold, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
```

```
df = pd.read_csv("drug200.csv")
df.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	DrugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	DrugY

2- Assigning Age, Sex, BP, Cholesterol and Na_to_K columns to X and Drug column to Y.

```
X = df[["Age","Sex","BP","Cholesterol","Na_to_K"]]
X.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K
0	23	F	HIGH	HIGH	25.355
1	47	M	LOW	HIGH	13.093
2	47	M	LOW	HIGH	10.114
3	28	F	NORMAL	HIGH	7.798
4	61	F	LOW	HIGH	18.043

```
y = df['Drug']
y.head()
```

```
0    DrugY
1    drugC
2    drugC
3    drugX
4    DrugY
Name: Drug, dtype: object
```

3- Split the data into training and test sets: The data is typically split into two sets, one for training and one for testing. The training set is used to train the algorithm, while the test set is used to evaluate the performance of the trained model. Then StandardScaler is used to standardize the features of a dataset by removing the mean and scaling to unit variance.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size = 0.3)
```

```
X_train = pd.get_dummies(X_train)
X_test = pd.get_dummies(X_test)
X_train.head()
```

	Age	Na_to_K	Sex_F	Sex_M	BP_HIGH	BP_LOW	BP_NORMAL	Cholesterol_HIGH	Cholesterol_NORMAL
131	52	32.922	0	1	0	1	0	0	1
96	58	38.247	1	0	0	1	0	1	0
181	59	13.884	1	0	0	0	1	1	0
19	32	25.974	1	0	1	0	0	0	1
153	72	14.642	1	0	0	1	0	0	1

```
sc=StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

4- Define the distance metric: The distance metric is a function that is used to calculate the distance between two observations. Common distance metrics include Euclidean distance, Manhattan distance and Minkowski distance. After trying this 3 distances it has been seen that manhattan is the best distance metric out of them.

After choosing the metric knn score is printed for train and test sets. It's seen that knn score is 0.857 for train and 0.85 for test set.

Make predictions: Given a new observation, the KNN algorithm finds the k-nearest training examples to it in feature space, and then it assigns the class label based on the majority class among those k-nearest examples.

There is a prediction example for 52 year old male with BP levels are low, normal cholesterol and 32.922 Na_to_K which preditcs DrugY

```
knn = KNeighborsClassifier(n_neighbors=30, metric='manhattan')
knn.fit(X_train,y_train)
```

```
KNeighborsClassifier(metric='manhattan', n_neighbors=30)
```

```
print(knn.score(X_train, y_train),knn.score(X_test, y_test))
```

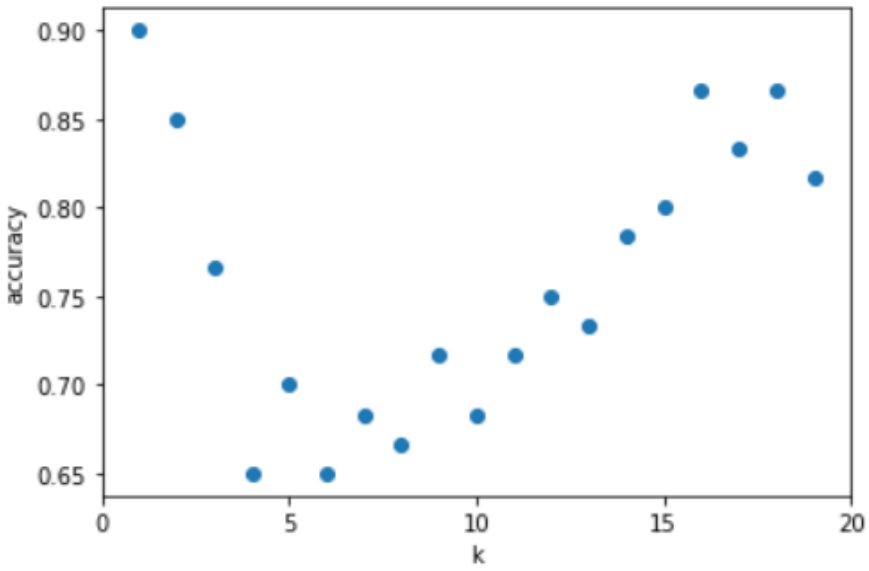
```
0.8571428571428571 0.85
```

```
y_pred = knn.predict(X_test)
```

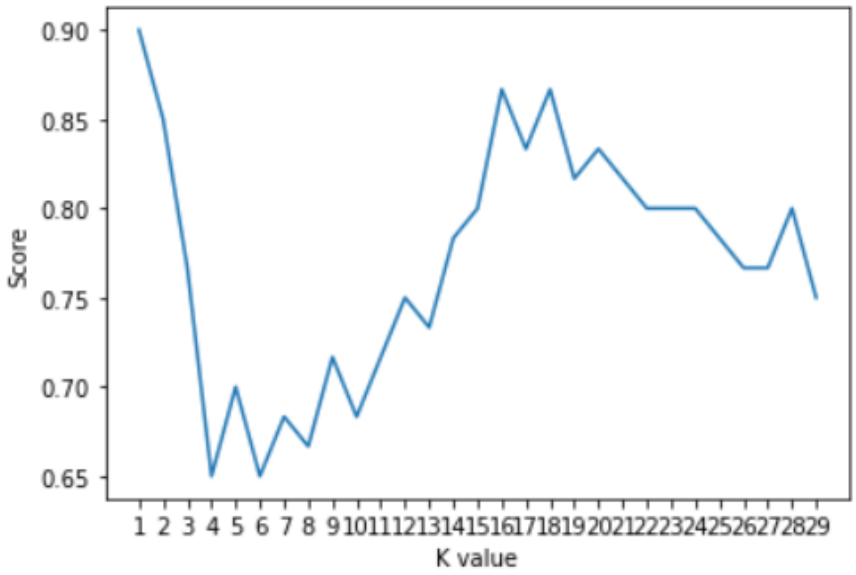
```
### Use the trained k-NN classifier model to classify new, previously unseen objects
knn_prediction = knn.predict([[52, 32.922, 0, 1, 0, 1, 0, 0, 1]])

knn_prediction[0]
```

```
'DrugY'
```

5- The sensitivity of k-NN classification accuracy to the choice of the 'k' parameter value is determined and displayed in the graph below.



6- The max KNN Acccuracy on every ‘k’ value is determined and displayed in the graph below.




```
KNN Acc Max 90.00%
```

7- A classification report is used for displaying performance evaluation metric in machine learning that is used to evaluate the performance of a classification algorithm. It provides a summary of the performance of the algorithm on a test set by computing several evaluation metrics such as precision, recall, f1-score, and support.

The classification report contains the following metrics:
Precision: It is the ratio of correctly predicted positive observations to the total predicted positive observations. It measures the ability of the classifier not to label as positive a sample that is negative.
Recall: It is the ratio of correctly predicted positive observations to the all observations in actual class. It measures the ability of the classifier to find all the positive observations.
F1-Score: It is the harmonic mean of precision and recall. The F1-Score reaches its best value at 1 (perfect precision and recall) and worst at 0.
Support: It is the number of samples of the true response that lie in that class.

The confusion matrix is used to show the number of correct predictions and incorrect predictions made by the classifier. It is typically represented as a table, with the true class labels on one axis and the predicted class labels on the other axis.
A confusion matrix allows to understand how well the classifier is doing by providing a clear picture of how many observations are being correctly classified and how many are not. It also helps to identify common mistakes made by the classifier.

```
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

       DrugY       0.75      0.80      0.77        30
       drugA       1.00      0.80      0.89         5
       drugB       0.50      0.33      0.40         3
       drugC       1.00      1.00      1.00         4
       drugX       0.72      0.72      0.72        18

    accuracy                           0.77        60
   macro avg       0.79      0.73      0.76        60
weighted avg       0.77      0.77      0.76        60

[[24  0  1  0  5]
 [ 1  4  0  0  0]
 [ 2  0  1  0  0]
 [ 0  0  0  4  0]
 [ 5  0  0  0 13]]
```

8- ROC AUC (Receiver Operating Characteristic - Area Under the Curve) is a performance evaluation metric for binary classification problems. It is implemented for measure of the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity) of a classifier.
The predict_proba() method is used to generate the predicted class probabilities for the test data.
Then, the roc_auc_score() method is provided by the scikit-learn library to calculate the AUC-ROC score. The method takes two arguments: the true labels of the test set and the predicted class probabilities generated by the classifier.
The predict_proba() method is used to generate the predicted class probabilities for the test data.
As we can see below ROC AUC score for our multiclass classification is 0.964

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc
y_probs = knn.predict_proba(X_test)
roc_auc = roc_auc_score(y_test, y_probs, multi_class='ovr')
print("Overall ROC AUC: {:.3f}".format(roc_auc))
```

```
Overall ROC AUC: 0.964
```

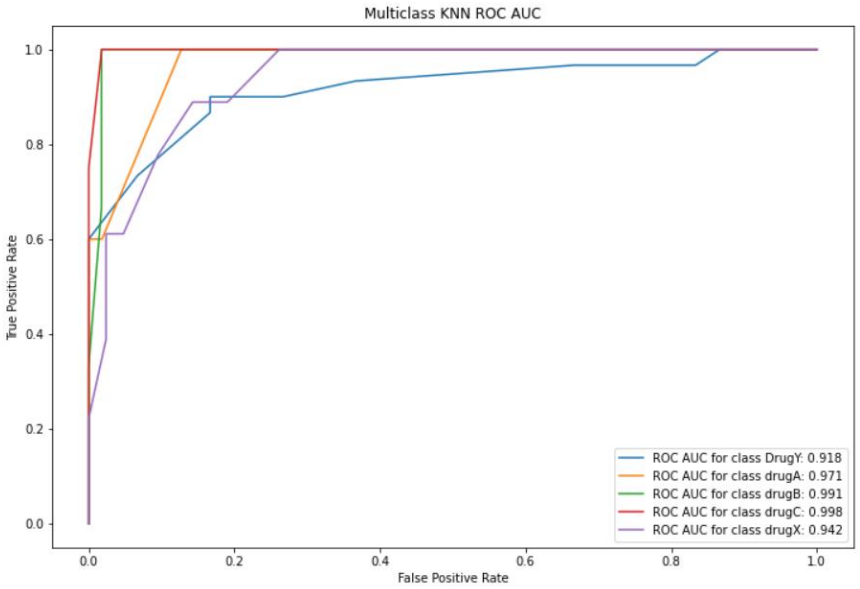
9- The ROC curve is a graphical representation of the performance of a classifier, where the true positive rate is plotted on the y-axis and the false positive rate is plotted on the x-axis. The ideal classifier would have a ROC curve that hugs the top left corner of the plot, indicating that it has a high true positive rate and a low false positive rate. The area under the ROC curve (AUC) is a single number that summarizes the performance of a classifier across all possible threshold
Used the roc_curve() method provided by the scikit-learn library to calculate the true positive rate and false positive rate for each class.
Plot the ROC curve for each class by plotting the true positive rate against the false positive rate. Use different colors for each class to distinguish them.

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer

lb = LabelBinarizer()
y_test_bin = lb.fit_transform(y_test)
```

```
y_test_labels = lb.classes_
```

```
plt.figure(figsize=(12,8))
for i in range(y_test_bin.shape[1]):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_probs[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label='ROC AUC for class {}: {:.3f}'.format(y_test_labels[i], roc_auc))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass KNN ROC AUC')
plt.legend()
plt.show()
```

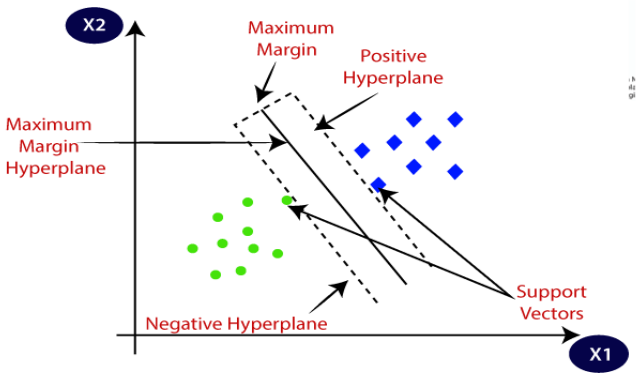
10- The plot shows the trade-off between the true positive rate and the false positive rate for each class. The closer the ROC curve is to the top left corner of the plot, the better the classifier is at distinguishing between the positive and negative classes. The AUC score is a single number that summarizes the performance of the classifier across all possible thresholds. A score of 1 represents a perfect classifier, and a score of 0.5 represents a classifier that performs no better than random guessing.



3. SVM

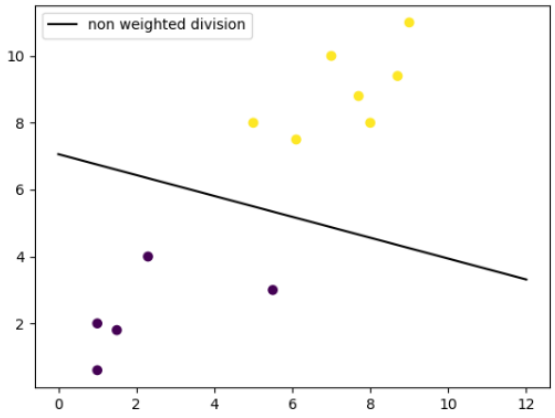
3.1 Description of the SVM methodology

SVM stands for Support Vector Machine, it is a type of supervised learning algorithm that can be used for classification or regression tasks. The main idea behind SVM is to find a hyperplane (a line or a plane in high-dimensional space) that separates different classes of data points with the largest margin possible. This hyperplane is called the decision boundary. SVM algorithm is useful in high dimensional spaces and also effective in cases where the number of dimensions is greater than the number of samples.



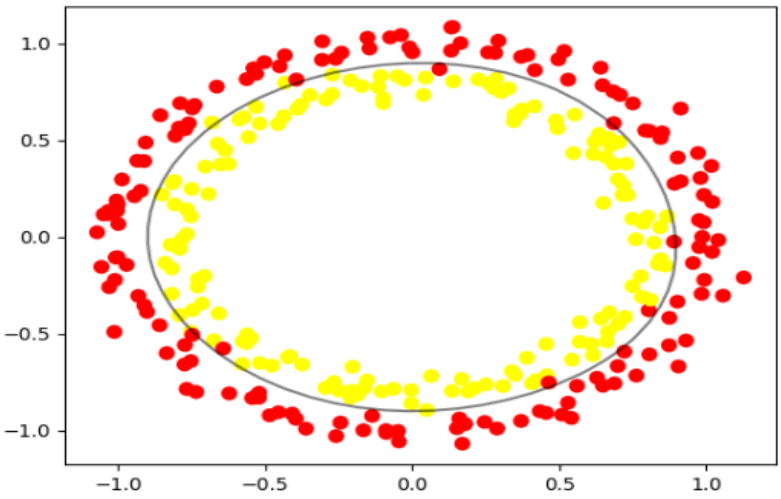
There are two main types of SVM:

Linear SVM: Linear SVM is used for linear classification problems, where the data can be separated by a straight line or a plane. A linear SVM finds the hyperplane that separates the different classes of data points with the largest margin.



linear SVM

Non-Linear SVM: Non-Linear SVM is used for non-linear classification problems, where the data cannot be separated by a straight line or a plane. Non-Linear SVM uses a technique called the kernel trick, which maps the input data into a higher-dimensional space where it becomes linearly separable. Some examples of Non-linear kernel functions include polynomial, radial basis function (RBF) and sigmoid.



non-linear SVM using RBF kernel

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

Support Vectors: The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

3.2 Implementation of SVM in Python

1. Data preparation: The first step is to gather and prepare the data that will be used to train the SVM model. This includes cleaning and preprocessing the data, and possibly transforming it into a format that can be used by the algorithm.

2. Choosing a kernel: SVM uses a kernel trick to transform the input data into a higher dimensional space where it can be separated by a hyperplane. Different types of kernels can be used, such as linear, polynomial, and radial basis function (RBF). The choice of kernel will depend on the specific characteristics of the data.

3. Training the model: Once the data is prepared and the kernel is chosen, the SVM algorithm can be trained on the data. This involves finding the optimal hyperplane that separates the different classes of data points with the largest margin.

4. Making predictions: After the SVM model is trained, it can be used to make predictions on new data. This is done by passing the new data through the same kernel used during training and then determining which class the new data point belongs to based on which side of the decision boundary it falls on.

5. Model Evaluation: After the predictions are made, the model performance is evaluated using metrics like accuracy, precision, recall, and F1-score. If the performance is not satisfactory, the process is repeated by tweaking the parameters or trying different kernel functions.

- Benefits of SVM:

1. SVM can handle high dimensional data effectively, even when the number of dimensions is greater than the number of samples.
2. SVM can be used for both classification and regression tasks.
3. SVM can effectively handle non-linear decision boundaries using kernel tricks.
4. SVM is less prone to overfitting compared to some other algorithms.

- Disadvantages of SVM:

1. SVM is sensitive to the choice of kernel function and the parameters of the kernel function.
2. SVM can be computationally intensive and may not scale well to very large datasets.
3. SVM does not provide probability estimates for the class, which can be useful in certain applications.
4. SVM can be sensitive to the presence of outliers and noise in the data.
5. SVM does not perform well when the data is too large.
6. SVM does not perform well when the number of features is much greater than the number of samples.

Step 1:

```
In [63]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
```

Firstly, we import necessary library as numpy, pandas and matplotlib.pyplot libraries.

Step 2:

```
In [64]: df_drug = pd.read_csv("drug200.csv")
```

```
In [65]: df_drug.head()
```

Out[65]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	DrugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	DrugY

We upload the dataset to read. Then, we check the head of data to analyze our data features.

Step 3:

```
In [66]: print(df_drug.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          200 non-null    int64  
 1   Sex          200 non-null    object 
 2   BP           200 non-null    object 
 3   Cholesterol  200 non-null    object 
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object 
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
None
```

We print data info with using .info() function to check our features type and whether they are null or not.

```
In [67]: df_drug.Drug.value_counts()
```

```
Out[67]: DrugY    91
drugX    54
drugA    23
drugC    16
drugB    16
Name: Drug, dtype: int64
```

Also, we check number of drug type.

Step 4:

```
In [71]: df_drug.describe()
```

Out[71]:

	Age	Na_to_K
count	200.000000	200.000000
mean	44.315000	16.084485
std	16.544315	7.223956
min	15.000000	6.269000
25%	31.000000	10.445500
50%	45.000000	13.936500
75%	58.000000	19.380000
max	74.000000	38.247000

We check statistical values like count, mean, standart deviation, min, max for Age and Na_to_K features to analyze these features.

Step 5:

```
In [72]: skewAge = df_drug.Age.skew(axis = 0, skipna = True)
         print('Age skewness: ', skewAge)
```

```
Age skewness:  0.03030835703000607
```

```
In [73]: skewNatoK = df_drug.Na_to_K.skew(axis = 0, skipna = True)
         print('Na to K skewness: ', skewNatoK)
```

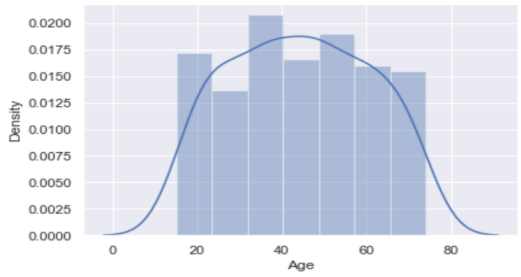
```
Na to K skewness:  1.039341186028881
```

To check skewness of Age and Na_to_K , we use .skew() funciton.

Step 6: To check distribution of Age and Na_to_K, we use sns.distplot() function.

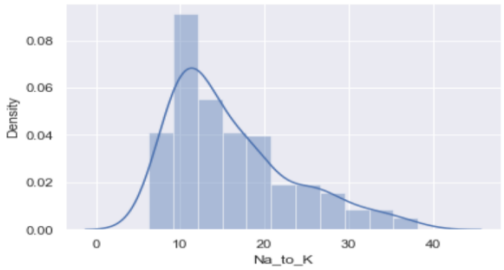
```
In [139]: sns.distplot(df_drug['Age']);
```

```
C:\Users\melis\Anaconda3\lib\site-packages\seaborn\distribu
d will be removed in a future version. Please adapt your c
xibility) or `histplot` (an axes-level function for histog
  warnings.warn(msg, FutureWarning)
```




```
In [140]: sns.distplot(df_drug['Na_to_K']);
```

```
C:\Users\melis\Anaconda3\lib\site-packages\seaborn\distribu
d will be removed in a future version. Please adapt your c
xibility) or `histplot` (an axes-level function for histog
  warnings.warn(msg, FutureWarning)
```



Step 7: We import some librarirs to make classification and confusion matrix. A confusion matrix is a table that is used to define the performance of a classification model. It gives a detailed breakdown of correct and incorrect predictions made by the model. The rows of the matrix represent the actual values, while the columns represent the predicted values.

A typical confusion matrix will have four entries: true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN). These terms are defined as follows:

True Positives (TP): These are the cases where the model correctly predicted the positive class.

False Positives (FP): These are the cases where the model predicted the positive class but the actual class is negative.

True Negatives (TN): These are the cases where the model correctly predicted the negative class.

False Negatives (FN): These are the cases where the model predicted the negative class but the actual class is positive.

Using the values in the confusion matrix, various performance metrics such as accuracy, precision, recall, and F1-score can be computed.

It is important to note that confusion matrix is commonly used to evaluate binary classifiers, but it can be used to evaluate the performance of multi-class classifiers as well by creating multiple binary classifiers, one for each class.

After that, we drop the Drug column to assign x values. Drug column is our y values so, we assign this value as y. Then we import model selection library to split data to train and test. To split data, train_test_split() function is used with test size = 0.3.

```
In [199]: from sklearn.metrics import confusion_matrix
          from sklearn.metrics import classification_report

In [200]: X = df_drug.drop(["Drug"], axis=1)
          y = df_drug["Drug"]

          from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

Step 8: To make categorical values to numeric, pd.get_dummies() function is used and to check data with .head().

```
In [202]: X_train = pd.get_dummies(X_train)
          X_test = pd.get_dummies(X_test)

In [203]: X_train.head()
```

Out[203]:

	Sex_F	Sex_M	BP_HIGH	BP_LOW	BP_NORMAL	Cholesterol_HIGH	Cholesterol_NORMAL	Age_binned_<20s	Age_binned_20s	Age_binned_30s	Age_binn
131	0	1	0	1	0	0	1	0	0	0	
96	1	0	0	1	0	1	0	0	0	0	
181	1	0	0	0	1	1	0	0	0	0	
19	1	0	1	0	0	0	1	0	0	1	
153	1	0	0	1	0	0	1	0	0	0	

```
In [204]: X_test.head()
```

Out[204]:

	Sex_F	Sex_M	BP_HIGH	BP_LOW	BP_NORMAL	Cholesterol_HIGH	Cholesterol_NORMAL	Age_binned_<20s	Age_binned_20s	Age_binned_30s	Age_binn
18	0	1	0	1	0	1	0	0	1	0	
170	1	0	0	0	1	1	0	0	1	0	
107	0	1	0	1	0	1	0	0	0	0	
98	0	1	1	0	0	0	1	0	1	0	
177	0	1	0	0	1	1	0	0	1	0	

Step 9: First of all, MinMaxScaler is imported. MinMaxScaler is a preprocessing technique used to scale the features of a dataset to a specific range, usually [0,1]. It is a way to normalize the features so that they are on the same scale, which can be useful for certain machine learning algorithms that are sensitive to the scale of the input features.

The MinMaxScaler method scales the feature values by subtracting the minimum value of the feature and dividing by the range (max - min). In Python, MinMaxScaler is a class provided by the scikit-learn library that can be used to scale the features of a dataset to a specific range, usually [0, 1].

After that, it is called and used to X_train and X_test.

```
In [254]: from sklearn.preprocessing import MinMaxScaler
          scaler = MinMaxScaler()
          X_train = scaler.fit_transform(X_train)
          X_test = scaler.transform(X_test)
```

Step 10: SVM model is set with kernel = linear and calculate precision, accuracy, recall, f1-score and support values.

Accuracy is a metric used to evaluate the performance of a classification model. It is defined as the ratio of the number of correct predictions to the total number of predictions made by the model.

Accuracy can be calculated using the following formula:

Accuracy = (number of correct predictions) / (total number of predictions)

It's important to note that accuracy should not be used as the sole metric of evaluation, as it can be misleading in certain scenarios, such as imbalanced datasets. Other evaluation metrics such as precision, recall, F1-score, and AUC-ROC should also be considered.

Precision, recall, and F1-score are all metrics used to evaluate the performance of a classification model.

Precision: Precision is a metric that measures the proportion of true positive predictions made by a model among all positive predictions made by the model. It is calculated using the following formula:

Precision = (True Positives) / (True Positives + False Positives)

Recall (Sensitivity or True Positive Rate): Recall is a metric that measures the proportion of true positive predictions made by a model among all actual positive examples. It is calculated using the following formula:

Recall = (True Positives) / (True Positives + False Negatives)

F1-score: F1-score is a metric that is a harmonic mean of precision and recall. It tries to balance the trade-off between precision and recall by taking into account both true positives and false positives. It is calculated using the following formula:

F1-score = 2 * (Precision * Recall) / (Precision + Recall)

It is important to note that precision and recall are not always in trade-off with each other and a good model should have a balance of both. In some cases, high precision is more important, such as in a medical test where a false positive result could cause unnecessary anxiety or further testing. In other cases, high recall is more important, such as in a spam filter where it is more important to capture all spam emails and not let any go through. F1-score is a metric that balances the trade-off between precision and recall.

In [255]:
```
from sklearn.svm import SVC
SVCclassifier1= SVC(kernel='linear', max_iter=251, probability=True)
SVCclassifier1.fit(X_train, y_train)

y_pred = SVCclassifier1.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
SVCAcc = accuracy_score(y_pred,y_test)
print('SVC accuracy is: {:.2f}%'.format(SVCAcc*100))
```

```
              precision    recall  f1-score   support

           1       0.82      1.00      0.90        18
           2       1.00      0.70      0.82        30
           3       0.71      1.00      0.83         5
           4       0.75      1.00      0.86         3
           5       0.67      1.00      0.80         4

    accuracy                           0.85        60
   macro avg       0.79      0.94      0.84        60
weighted avg       0.89      0.85      0.85        60

[[18  0  0  0  0]
 [ 4 21  2  1  2]
 [ 0  0  5  0  0]
 [ 0  0  0  3  0]
 [ 0  0  0  0  4]]
SVC accuracy is: 85.00%
```

Step 11: To calculate roc_auc_score, define function is created. The ROC AUC (Receiver Operating Characteristic - Area Under the Curve) score is a metric used to evaluate the performance of binary classification models. It is a measure of the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity) of a classification model at different threshold settings.

The ROC curve is a plot of the true positive rate (sensitivity) against the false positive rate (1-specificity) for different threshold settings. The AUC (Area Under the Curve) of the ROC curve is a measure of how well the model is able to distinguish between the positive and negative classes. A model with a higher AUC score is considered to be a better classifier than one with a lower AUC score.

In [256]:
```
from sklearn import metrics
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelBinarizer
```

In [257]:
```
def multiclass_roc_auc_score(y_test, y_pred, average="macro"):
    lb = LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    y_pred = lb.transform(y_pred)
    return roc_auc_score(y_test, y_pred, average=average)
```

Step 12: To calculate roc_auc_score, define function is called.

In [258]:
```
#AUC-ROC
multiclass_roc_auc_score(y_test, y_pred)
```

Out[258]: 0.9515140123034861

Step 13: To calculate best SVM accuracy and AUC-ROC value, we set the model with kernel = linear, poly, rbf and sigmoid. Also we change max_iteraiton number to check whether there is a any change in accuracy.

In [261]:
```
from sklearn.svm import SVC
SVCclassifier3 = SVC(kernel='poly', max_iter=251, probability=True)
SVCclassifier3.fit(X_train, y_train)

y_pred = SVCclassifier3.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
SVCAcc = accuracy_score(y_pred,y_test)
print('SVC accuracy is: {:.2f}%'.format(SVCAcc*100))
```

```
[[17  1  0  0  0]
 [ 4 21  2  1  2]
 [ 0  1  4  0  0]
 [ 0  0  0  3  0]
 [ 0  0  0  0  4]]
SVC accuracy is: 81.67%
```

```
In [262]: #AUC-ROC
          multiclass_roc_auc_score(y_test, y_pred)

Out[262]: 0.9192917900812638
```

Conclusion: After implementing 6 SVM models, model1 SVM with kernel = kernel turned out to be the best with %85 accuracy score and 0.9515140123034861 AUC-ROC score. Moreover max iteation number does not affect the scores.

4. DECISION TREE CLASSIFIER

4.1 Description of the Decision Tree Methodology

A decision tree classifier is a supervised machine learning algorithm that is used to classify data by creating a flowchart-like tree structure. The internal nodes in the tree represent features or attributes, the branches represent decision rules, and the leaf nodes represent the outcome. The purpose of a decision tree classifier is to develop a model that can accurately predict the class label of an input example based on multiple input features.

The decision tree classifier algorithm begins by creating a root node and then it repeatedly divides the data into smaller subsets based on the selected attribute and its value. This process of dividing the data into subsets continues until the leaf nodes are reached. The leaf nodes in the tree represent the final class label, and the path from the root node to the leaf node represents the classification rules.

The decision tree classifier algorithm employs the concepts of entropy and information gain to decide where to split the data. Entropy is used to evaluate the impurities in the input set, while information gain measures the reduction in entropy after the set is divided on a specific attribute. The algorithm chooses the attribute with the highest information gain as the split point and applies the same process on each subset.

In brief, a decision tree classifier is a supervised machine learning technique that is utilized for classifying data. It creates a tree-like structure where internal nodes represent features, branches represent decision rules, and leaf nodes represent the outcome. The algorithm follows a greedy approach where it selects the locally optimal choice at each step with the goal of finding a globally optimal solution.

4.2 Implementation of Decision Tree Classifier in Python

Importing libraries that will be used in this notebook.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Following the import of libraries, the dataset will also import for use.

```
drug = pd.read_csv("drug200.csv")
```

Displaying the first ten rows of the dataset using the head() function.

```
drug.head(10)
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	DrugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	DrugY
5	22	F	NORMAL	HIGH	8.607	drugX
6	49	F	NORMAL	HIGH	16.275	DrugY
7	41	M	LOW	HIGH	11.037	drugC
8	60	M	NORMAL	HIGH	15.171	DrugY
9	43	M	LOW	NORMAL	19.368	DrugY

info.() function provides information about the DataFrame's columns, including the data type and number of non-null values.

```
drug.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          200 non-null    int64  
 1   Sex          200 non-null    object 
 2   BP           200 non-null    object 
 3   Cholesterol  200 non-null    object 
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object 
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

Input variables are defined as "X" and output variables as "y". Using the .drop() function to extract the "Drug" column from the input data.

After the inputs and outputs are determined, the data is separated for training and testing.

```
X = drug.drop(["Drug"], axis=1)
y = drug["Drug"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

One-hot encoding is applied to the outputs.

```
X_train = pd.get_dummies(X_train)
X_test = pd.get_dummies(X_test)
```

The decision tree classifier algorithm is applied to the data set. The accuracy scores are evaluated by varying the parameters of the algorithm.

```
from sklearn.tree import DecisionTreeClassifier
dtr3= DecisionTreeClassifier(criterion='entropy', splitter='best', max_depth=None, min_samples_split=2,max_leaf_nodes=40, random_state = 0)
dtr3.fit(X_train, y_train)

y_pred = dtr3.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
dtascore = accuracy_score(y_pred,y_test)
print('Decision Tree accuracy is: {:.2f}%'.format(dtascore*100))
```

The results of the parameter giving the highest accuracy result are given below.

```
                precision    recall  f1-score   support

         DrugY       0.95      0.67      0.78        30
         drugA       0.56      1.00      0.71         5
         drugB       0.75      1.00      0.86         3
         drugC       0.67      1.00      0.80         4
         drugX       0.85      0.94      0.89        18

      accuracy                           0.82        60
     macro avg       0.75      0.92      0.81        60
  weighted avg       0.86      0.82      0.82        60

[[20  4  1  2  3]
 [ 0  5  0  0  0]
 [ 0  0  3  0  0]
 [ 0  0  0  4  0]
 [ 1  0  0  0 17]]
Decision Tree accuracy is: 81.67%
```

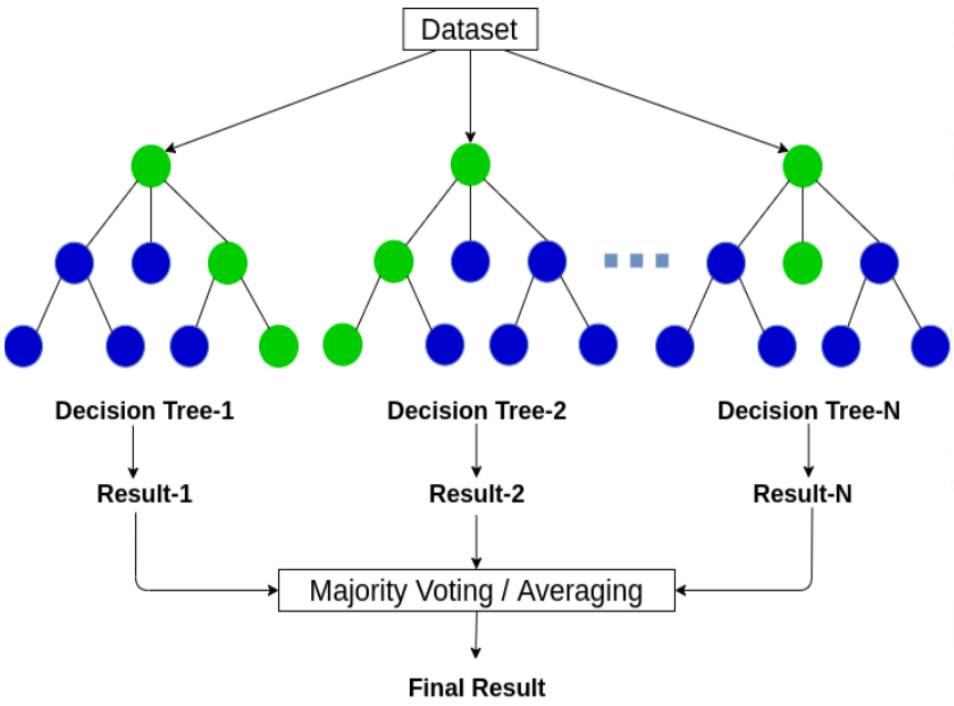
The Receiver Operating Characteristic (ROC) Area Under the Curve (AUC) Score is a commonly used evaluation metric for classification problems. ROC-AUC score is not influenced by the imbalance between the classes and it's a suitable metric to employ in case of a significant class imbalance. The higher the model's AUC, the better the classification model.

```
def multiclass_roc_auc_score(y_test, y_pred, average='macro'):
    lb=LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    y_pred = lb.transform(y_pred)
    return roc_auc_score(y_test, y_pred, average=average)

print("ROC AUC Score is: ", multiclass_roc_auc_score(y_test, y_pred))
```

```
ROC AUC Score is:  0.9281806789701527
```

4.3 Random Forest Classifier

Random Forest Classifier is a method of ensemble learning, which is used for classification problems. It creates several decision trees during the training phase and selects the class label that is the most common among the predictions of all the trees. The algorithm combines the results of multiple decision trees to increase both accuracy and stability of the model. The term "random" in the name of the algorithm refers to the random selection of features at each split in the decision trees and the use of random samples of training data to build the decision trees. This randomness reduces the correlation between the individual decision trees, resulting in an improved overall performance of the model.



4.4 Implementation of Random Forest Classifier in Python

When applying the random forest algorithm, selecting the appropriate parameters is crucial for improving the accuracy of the model.

n_estimators

The n_estimators parameter in the Random Forest algorithm determines the quantity of decision trees in the forest. Increasing the number of trees will generally improve the model's performance as it allows the model to extract more information from the training data. However, it also increases the of overfitting. It is essential to find the optimal determining the value of n_estimators. As a general rule, a higher number of trees leads to better performance, but it also increases the training time.

Criterion

The criterion parameter in the Random Forest algorithm is used to determine the method of measuring the quality of the splits. One of the options for this parameter is the 'gini' index, which assesses the impurity of the splits of the nodes.

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=100, criterion='gini', max_leaf_nodes=30, random_state=0)
rfc.fit(X_train, y_train)

y_pred = rfc.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
rfcas = accuracy_score(y_pred,y_test)
print('Random Forest accuracy is: {:.2f}%'.format(rfcas*100))
```

```
                precision    recall  f1-score   support

         DrugY       0.95      0.70      0.81        30
         drugA       0.67      0.80      0.73         5
         drugB       0.75      1.00      0.86         3
         drugC       0.67      1.00      0.80         4
         drugX       0.82      1.00      0.90        18

      accuracy                           0.83        60
     macro avg       0.77      0.90      0.82        60
  weighted avg       0.86      0.83      0.83        60

[[21  2  1  2  4]
 [ 1  4  0  0  0]
 [ 0  0  3  0  0]
 [ 0  0  0  4  0]
 [ 0  0  0  0 18]]
Random Forest accuracy is: 83.33%
```

```
def multiclass_roc_auc_score(y_test, y_pred, average='macro'):
    lb=LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    y_pred = lb.transform(y_pred)
    return roc_auc_score(y_test, y_pred, average=average)

print("ROC AUC Score is: ", multiclass_roc_auc_score(y_test, y_pred))
```

```
ROC AUC Score is:  0.9281806789701527
```

4.5 Application results

As seen in the example, random forest classifier usually results in better performance than decision tree classifier. The reason for this result is that the random forest classifier combines multiple decision trees to improve the accuracy and stability of the model. Additionally, when making predictions, the random forest classifier averages the predictions made by the majority vote or all decision trees. By combining the predictions of multiple decision trees, it can reduce the problem of overfitting when using a single decision tree. As a result, it reduces fitting and variance.

5. NAIVE BAYES CLASSIFIER

5.1 Description of the Naive Bayes Methodology

In machine learning and statistics; classification is a supervised learning approach that the computer program learns from given data input and then uses that learning to classify new observations. These datasets can be bi-class (like whether a person is male or female, determining whether an email is spam or not) or multi-class.

Bayes theorem is found by Thomas Bayes which Naive Bayes Classifier is based on.

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem.

Bayes Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Using Bayes theorem, we can find the probability of A happening, given that B has occurred. Here, B is the evidence and A is the hypothesis. The assumption made here is that the predictors/features are independent. That is presence of one particular feature does not affect the other. Hence it is called naive.

The class with the highest probability is considered as the most likely class. This is also known as the Maximum A Posteriori (MAP).

The MAP for a hypothesis with 2 events A and B is

MAP (A)

= max (P (A | B))

= max (P (B | A) * P (A))/P (B)

= max (P (B | A) * P (A))

Here, P (B) is evidence probability. It is used to normalize the result. It remains the same, So, removing it would not affect the result.

Naive Bayes Classifier assumes that all the features are unrelated to each other. Presence or absence of a feature does not influence the presence or absence of any other feature.

In real world datasets, we test a hypothesis given multiple evidence on features. So, the calculations become quite complicated. To simplify the work, the feature independence approach is used to uncouple multiple evidence and treat each as an independent one.

Example

Consider the problem of playing golf. The dataset is represented as below.

	OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY GOLF
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes
5	Sunny	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Rainy	Mild	High	False	No
8	Rainy	Cool	Normal	False	Yes
9	Sunny	Mild	Normal	False	Yes
10	Rainy	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Sunny	Mild	High	True	No

We classify whether the day is suitable for playing golf, given the features of the day. The columns represent these features and the rows represent individual entries. If we take the first row of the dataset, we can observe that is not suitable for playing golf if the outlook is rainy, temperature is hot, humidity is high and it is not windy. We make two assumptions here, one as stated above we consider that these predictors are independent. That is, if the temperature is hot, it does not necessarily mean that the humidity is high. Another assumption made here is that all the predictors have an equal effect on the outcome. That is, the day being windy does not have more importance in deciding to play golf or not.

According to this example, Bayes theorem can be rewritten as:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

The variable y is the class variable(play golf), which represents if it is suitable to play golf or not given the conditions. Variable X represent the parameters/features.

X is given as,

$$X = (x_1, x_2, x_3, ..., x_n)$$

Here x_1,x_2….x_n represent the features, i.e they can be mapped to outlook, temperature, humidity and windy. By substituting for X and expanding using the chain rule we get,

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

Now, you can obtain the values for each by looking at the dataset and substitute them into the equation. For all entries in the dataset, the denominator does not change, it remain static. Therefore, the denominator can be removed and a proportionality can be introduced.

$$P(y|x_1, ..., x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

In our case, the class variable(y) has only two outcomes, yes or no. There could be cases where the classification could be multivariate. Therefore, we need to find the class y with maximum probability.

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

Using the above function, we can obtain the class, given the predictors.

Types of Naive Bayes Classifier:

Multinomial Naive Bayes:

This is mostly used for document classification problem, i.e whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

Bernoulli Naive Bayes:

This is similar to the multinomial naive bayes but the predictors are boolean variables. The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not.

Gaussian Naive Bayes:

When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.

Since the way the values are present in the dataset changes, the formula for conditional probability changes to,

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

5.2 Implementation of Naive Bayes Classifier in Python

We start implementing with importing the necessary libraries for our classification.

Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

And our dataset,

Importing the dataset

```
df = pd.read_csv("drug_data.csv")
```

```
df.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY

Input variables are defined as “X” and output variable, drug, is defined as “y”. That’s why we use the .drop() function to drop “Drug” column from our input data “X”.

And after that, we split the data into test and train. Test size is 30% of our whole data.

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

```
X = df.drop(["Drug"], axis=1)
y = df["Drug"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

We need to use one hot encoding to categorial columns in order to use these values in our calculations, such as “Age, Sex, BP, Cholesterol”. Without One hot encoding, functions wouldn’t understand their text value and couldn’t make any operations.

One-hot encoding

```
X_train = pd.get_dummies(X_train)
X_test = pd.get_dummies(X_test)
```

Now, our data is ready to train with Naive Bayes model.

```
from sklearn.naive_bayes import CategoricalNB
NBclassifier1 = CategoricalNB()
NBclassifier1.fit(X_train, y_train)

y_pred = NBclassifier1.predict(X_test)
```

“y_pred” is the parameter that contains our predicted values with using train data.

Finally, let’s see how well our model performs. To see that, We are going to print accuracy score, confusion matrix, and Roc Auc Score.

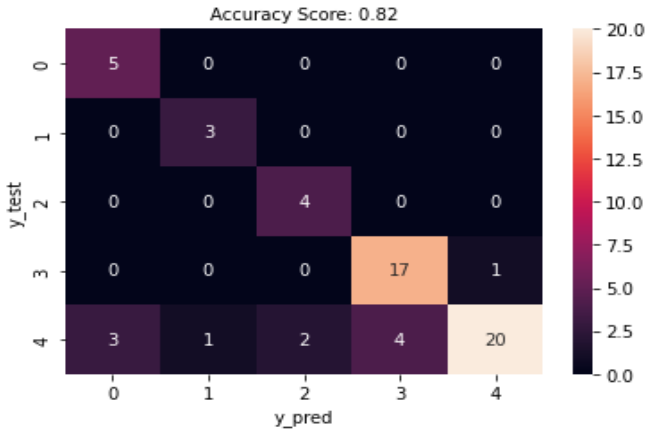
Accuracy,

```
from sklearn.naive_bayes import CategoricalNB
NBclassifier1 = CategoricalNB()
NBclassifier1.fit(X_train, y_train)

y_pred = NBclassifier1.predict(X_test)

from sklearn.metrics import accuracy_score
NBAcc1 = accuracy_score(y_pred,y_test)
print('Naive Bayes accuracy is: {:.2f}%'.format(NBAcc1*100)
```

```
Naive Bayes accuracy is: 81.67%
```

Confusion Matrix,

```
def plot_confusion_matrix(y_test, y_pred):
    acc = round(accuracy_score(y_test, y_pred), 2)
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt=".0f")
    plt.xlabel('y_pred')
    plt.ylabel('y_test')
    plt.title('Accuracy Score: {0}'.format(acc), size=10)
    plt.rcParams["figure.figsize"] = (15,10)
    plt.show()

plot_confusion_matrix(y_test, y_pred)
```



Roc Auc Score,

```
from sklearn import metrics
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelBinarizer
```

```
def multiclass_roc_auc_score(y_test, y_pred, average='macro'):
    lb=LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    y_pred = lb.transform(y_pred)
    return roc_auc_score(y_test, y_pred, average=average)

print("ROC AUC Score is: ", multiclass_roc_auc_score(y_test, y_pred))
```

```
ROC AUC Score is:  0.937473608263082
```

6. CLUSTERING

6.1 Description Of Clustering

Clustering or cluster analysis is a machine learning technique, which groups the unlabelled dataset. It can be defined as "Task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups."
Shape, size, color, behavior, etc. in unlabeled dataset. it finds some similar patterns and divides them according to the presence and absence of these similar patterns.

It is an unsupervised learning method, hence no supervision is provided to the algorithm, and it deals with the unlabeled dataset.
After applying this clustering technique, each cluster or group is given a cluster ID. The ML system can use this identity to simplify the processing of large and complex datasets.
The clustering technique is commonly used for statistical data analysis.
Clustering is somewhere similar to the classification algorithm, but the difference is the type of dataset we use. In classification we work with the labeled data set, while in clustering we work with the unlabeled data set.

Hierarchical clustering is based on the concept that nearby objects are more related than objects that are farther away. These algorithms provide a hierarchy of clusters that at certain distances are merged.
In centroid/partitioning clustering, clusters are represented by a central vector, which may not necessarily be a member of the dataset. This is an optimization problem: finding the number of centroids or the value of K and assigning the objects to nearby cluster centers. One of the most widely used centroid-based clustering algorithms is K-Means.

There are two methods to choose the correct value of K: Elbow and Silhouette.

The Elbow method picks the range of values and takes the best among them. It calculates the Within Cluster Sum of Square(WCSS) for different values of K. It calculates the sum of squared points and calculates the average distance.

The Silhouette score/coefficient(SC) is calculated using average intra-cluster distance(m) and an average of the nearest cluster distance(n) for each sample.

Gaussian mixture model (GMM) is one of the types of distribution-based clustering. These clustering approaches assume data is composed of distributions, such as Gaussian distributions. GMM can be used to find clusters in the same way as K-Means. The probability that a point belongs to the distribution's center decreases as the distance from the distribution center increases.

6.2 Implementation of Clustering

Type 1 (Silhouette Score)

I will use the Silhouette score to cluster the data and measure their success performance.
First, we'll start with installing all the libraries we need. If you are using Pycharm, the Plotly library must be set up with the terminal.
->Pip install plotly

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_samples, silhouette_score
from yellowbrick.cluster import SilhouetteVisualizer
from sklearn import decomposition
import plotly.graph_objects as go
```

We download to dataset in our project. Then reflecting the data on the screen.

```
17      veriler =pd.read_csv('drug200.csv')
18
19      veriler.info()
```

Output:

```
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          200 non-null    int64
 1   Sex          200 non-null    object
 2   BP           200 non-null    object
 3   Cholesterol  200 non-null    object
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object
```

In this part(Figure 4), first we encoded all the data to convert them turn different types into same type. The Drug Column needs to be dropped because this data set basically using for classification, so drug is the dependent variable. We do not need to dependent variable for clustering. The Main idea is understand the relationship between independent variables in clustering. After the encoding process , all relevant data are scaled.
In addition, the Train and test part was established and scaled.

```
21  def label_encoder(y):
22      le = LabelEncoder()
23      veriler[y] = le.fit_transform(veriler[y])
24
25
26  label_list = ["Sex","BP","Cholesterol","Na_to_K","Drug"]
27
28  for l in label_list:
29      label_encoder(l)
30
31  X, y = veriler.drop(['Drug'], axis=1), veriler['Drug']
32  X = veriler.iloc[:,5:].values
33  X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.2, random_state=101)
34
35  scaler = MinMaxScaler()
36  scaler = StandardScaler()
37
38  X_train_scaled = scaler.fit_transform(X_train)
39  X_test_scaled = scaler.transform(X_test)
40
41  y_train = y_train.to_frame()
42  y_test = y_test.to_frame()
43
44  one_hot = OneHotEncoder()
45  y_train_hot = one_hot.fit_transform(y_train).todense()
46  y_test_hot = one_hot.transform(y_test).todense()
```

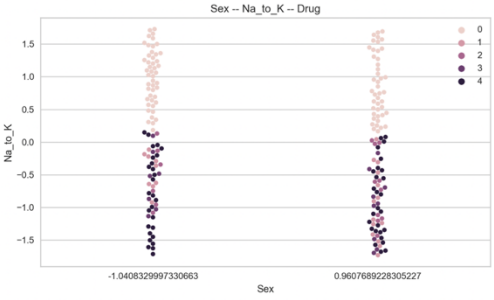
We scale the empty column that was belong the “Drug” before. Then we show the relationship between Drug and other variables.

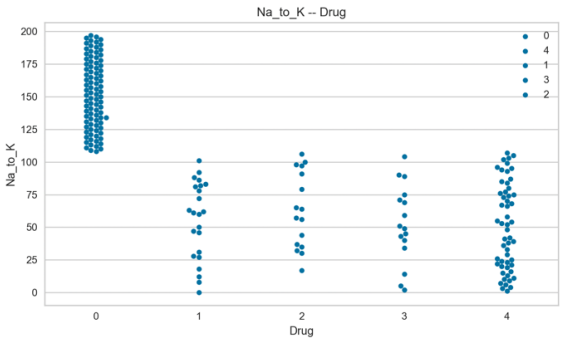
```
def scale_a_column(data, label_column='Drug'):
    scaler = StandardScaler()
    df1 = data.drop([label_column], axis=1)
    res = scaler.fit_transform(df1)
    df1 = pd.DataFrame(res, columns=df1.columns)
    df1[label_column] = data[label_column]
    return df1

scaled_data = scale_a_column(veriler)

plt.figure(figsize=(9,5))
sns.swarmplot(x="Drug", y="Na_to_K", data=veriler)
plt.legend(veriler.Drug.value_counts().index)
plt.title("Na_to_K -- Drug")
plt.show()

def plot_2d(data, x, y, figsize=(9, 5)):
    plt.figure(figsize=figsize)
    sns.swarmplot(x=x, y=y, hue="Drug",data=data)
    plt.legend()
    plt.title(f"{x} -- {y} -- Drug")
    plt.show()
```

Plot_2d is used to visualize data.

```
plot_2d(scaled_data,          'Sex',          'Na_to_K')
plot_2d(scaled_data,          'BP',           'Na_to_K')
plot_2d(scaled_data,          'Cholesterol',  'Na_to_K')
plot_2d(scaled_data, 'Cholesterol', 'BP')
```

Output :





The main part has begun. First we defined our “def run_kmeans” function, 300 iteration is enough for this example. This function also allows us to find the optimal number of clusters for clustering. Na to k, age, BP, Cholestrol variables enter the loop to find optimum n numbers to clustering. We also scaled the X_Kmeans. One of the most critical aspects of clustering is selecting the correct value of K. Randomly selecting K might not be a favorable choice. In this example we’ll be using silhouette score to choose the value of K.

```
def run_kmean(X_scaled, y, n_clusters=14):
    kmeans = KMeans(
        init="random",
        n_clusters=n_clusters,
        n_init=10,
        max_iter=300,
        random_state=42)

    X_kmeans = X_scaled.copy()
    #X_kmeans['Na_to_K'] = X_kmeans['Na_to_K'] * 10
    #X_kmeans['Age'] = X_kmeans['Age'] * 5
    #X_kmeans['BP'] = X_kmeans['BP'] * 100
    #X_kmeans['Cholesterol'] = X_kmeans['Cholesterol'] * 1000

    kmeans.fit(X_kmeans)

    X_kmeans = X_scaled.copy()
    X_kmeans['cluster'] = kmeans.labels_
    print(X_kmeans)
    for i in range(0, X_kmeans.shape[1]):
        for j in range(i+1, X_kmeans.shape[1]):
            for k in range(j+1, X_kmeans.shape[1]-1):
                break
                plot_one(X_kmeans, X_kmeans.columns[i], X_kmeans.columns[j], X_kmeans.columns[k], 'cluster')
    #plot_one(X_kmeans, 'BP', 'Na_to_K', 'Cholesterol', 'cluster')
    return X_kmeans
```

Gaussian Mixture Model or Mixture of Gaussian as it is sometimes called, is not so much a model as it is a probability distribution. It is a universally used model for generative unsupervised learning or clustering. Gaussian Mixture models are used for representing Normally Distributed subpopulations within an overall population. K-means calculates distance and GM calculates weights. The k-means algorithm gives you a hard assignment or a soft assignment. In a lot of cases we just want that hard assignment. Sometimes we want the maximum probability like: This is going to be 70% likely that it's a part of this class. That's why we incorporate the standard deviation.

```
def run_em(X_scaled, y):
    X_GM = X_scaled.copy()
    gm = GaussianMixture(n_components=14, random_state=0).fit(X_GM)
    labels = gm.predict(X_GM)

    X_GM = X_scaled.copy()
    X_GM['cluster'] = labels
    print(X_GM)

from sklearn import random_projection

def run_random_project(X_scaled, y):
    transformer = random_projection.GaussianRandomProjection(n_components=X_scaled.shape[1] -1)
    X_rp = transformer.fit_transform(X_scaled)
    X_rp = pd.DataFrame(X_rp)
    X_rp.columns = [f'col{i+1}' for i in range(X_rp.shape[1])]

    data_rp_for_plot=pd.concat([X_rp, y], axis=1)

    for i in range(1, data_rp_for_plot.shape[1]):
        for j in range(i+1, data_rp_for_plot.shape[1]):
            for k in range(j+1, data_rp_for_plot.shape[1]):
                break
                plot_one(data_rp_for_plot, f'col{i}', f'col{j}', f'col{k}', 'Drug')
    return X_rp
```

In this section, we see which data belongs to which cluster.Decomposition is a statistical job that involves breaking down Time Series data into many components or identifying seasonality and trend from a series of data. The following are the components' definitions: The average value in the series is called the level. Principal Component Analysis (PCA) is a linear dimensionality reduction technique that can be used to extract information from a high-dimensional space. PCA tries to preserve key parts with more variation in the data and remove non-essential parts with less variation.

Inertia measures how well a dataset was clustered by K-Means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster. A good model is one with low inertia and a low number of clusters (K).

```
for i in range(n_start,n_max):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(x_pca)
    inertia_list[i] = kmeans.inertia_
    scores.append(silhouette_score(x_pca, kmeans.labels_))

n_max_shift = 2  # find maximum after this index of score
n_clusters = np.argmax(scores[n_max_shift:])+(n_start+n_max_shift)# it's my upgrade
plt.plot(range(0,n_max),inertia_list,'-o')
plt.xlabel('Number of cluster')
plt.axvline(x=n_clusters, color='blue', linestyle='--')
plt.ylabel('Inertia')
plt.show()

plt.plot(range(2,n_max), scores);
plt.title('Results KMeans')
plt.xlabel('n_clusters');
plt.axvline(x=n_clusters, color='blue', linestyle='--')
plt.ylabel('Silhouette Score');
plt.show()
```

As a result of these codes, it tries the number of clusters one by one, extracts the silhouette score and turns it into a table.

```
fig, ax = plt.subplots(2, 2, figsize=(15,8))
for i in [2, 3,4]:

    km = KMeans(n_clusters=i, init='k-means++', n_init=10, max_iter=100, random_state=42)
    q, mod = divmod(i, 2)

    visualizer = SilhouetteVisualizer(km, colors='yellowbrick', ax=ax[q-1][mod])
    visualizer.fit(X)

plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.title('Silhouetter Score')
plt.show()
```

Finally, we print the numerical version of our score.

```
score= silhouette_score(X,km.labels_,metric='euclidean')
print('Silhouetter Score: %.3f' % score)

x_sc_clusters =km.predict(X)
x_sc_clusters_centers =km.cluster_centers_

print(x_sc_clusters)
```

6.3 Application Results of Type - 1

- The value of the silhouette coefficient is between [-1, 1].
- A score of 1 denotes the best meaning that the data point i is very compact within the cluster to which it belongs and far away from the other clusters.
- The worst value is -1. Values near 0 denote overlapping clusters.

So our final result has acceptable value.

```
Silhouetter Score: 0.917
```

Type 2 (Elbow Method)

We just remove the silhouette score from the same code template and add the elbow part.

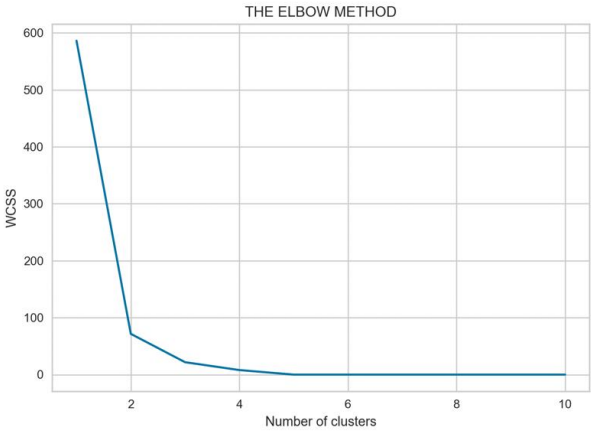
The elbow method runs k-means clustering (kmeans number of clusters) on the dataset for a range of values of k (say 1 to 10). For each of the K values, we calculate average distances to the centroid across all data points. Plot these points and find the point where the average distance from the centre falls suddenly ("Elbow"). We can use the Elbow method to work out what type of data we are looking at. It consists in the interpretation of a line plot with an elbow shape. The number of clusters is were the elbow bends, and the y axis is the Within Clusters Sum of Squares.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3,init='k-means++')
kmeans.fit(X)
print(kmeans.cluster_centers_)
sonuclar =[]

for i in range (1,11):
    kmeans= KMeans(n_clusters=i,init='k-means++',random_state=123)
    kmeans.fit(X)
    sonuclar.append(kmeans.inertia_)

plt.plot(range(1,11),sonuclar)
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.title('THE ELBOW METHOD')
plt.show()

x_sc_clusters =kmeans.predict(X)
x_sc_clusters_centers =kmeans.cluster_centers_
dist = [np.linalg.norm(x - y) for x, y in zip(X, x_sc_clusters_centers[x_sc_clusters])]

print('Elbow küme merkezine uzaklık durumu')
print(dist)
```

6.4 Application Results of Type - 2

From the above visualization, we can see that the optimal number of clusters should be around 2. But visualizing the data alone cannot always give the right answer. Hence we demonstrate the following steps.

```
Elbow küme merkezine uzaklık durumu
[0.0, 0.0, 0.0, 8.881784197001252e-16, 0.0, 8.881784197001252e-16, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.881784197001252e-16, 0.0,
```



For the given data, we conclude that the optimal number of clusters for the data is 3. To determine the value of k at the "elbow" ie the point after which the distortion/inertia start decreasing in a linear fashion, consider the following data set.

7. REFERENCES

https://ai-pool.com/a/s/random-forests-understanding

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

https://towardsdatascience.com/decision-tree-classifier-explained-in-real-life-picking-a-vacation-destination-6226b2b60575

https://www.kaggle.com/code/caesarmario/drug-classification-w-various-ml-models/notebook

https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

https://www.kaggle.com/code/prashant111/naive-bayes-classifier-in-python

https://scikit-learn.org/stable/modules/svm.html

https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/