Prepared by:
Muhammad Bilal Iqbal - F2019054047
Syed Hurr Zaidi – F2019054045
Syed Mohsin Kazmi - F2019054048
Mugheera Khan- F2019054013
Ahsan Ali- F2019054061

Final Project
Machine Learning
Section: B
Prepared for: Ms. Amna Altaf

Importing Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix, classification_report
from mpl_toolkits import mplot3d
from sklearn.multiclass import OneVsOneClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import BernoulliNB
```

Reading & Analyzing Data

```
df = pd.read_csv("/content/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

```
df.describe()
```

	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

```
df.isnull().sum()
```

```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

```
df.isna().sum()
```

```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

- No missing value
- 6 columns
- 200 rows

Variable Description

- Age: Age of patient
- Sex: Gender of patient
- BP: Blood pressure of patient
- Cholesterol: Cholesterol of patient
- Na_to_K: Sodium to Potassium Ratio in Blood
- Drug: Drug Type

```
df.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
```

- float64(1): Na_to_K
- int64(1): Age
- object(4): Sex, BP, Cholesterol, Drug

Univariate Variable Analysis

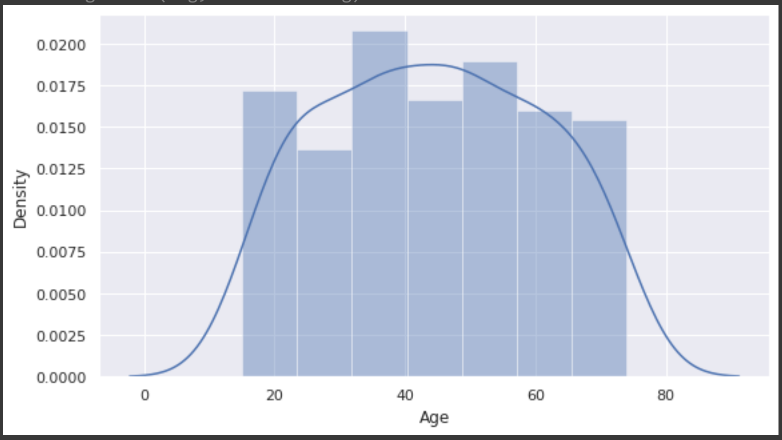
Age Variable

```
print("Max Age:", df.Age.max())
print("Min Age:", df.Age.min())
```

```
Max Age: 74
Min Age: 15
```

```
# Age distribution
plt.figure(figsize = (9,5))
sns.distplot(df.Age)
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be re
  warnings.warn(msg, FutureWarning)
```



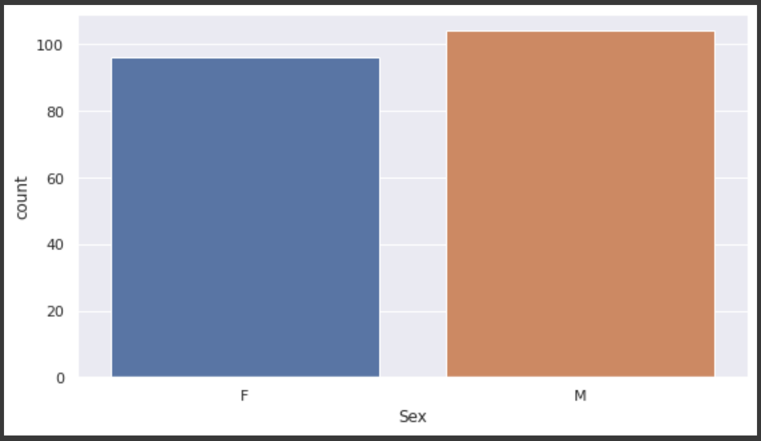
- Age range is between 15 and 74.

Sex Variable

```
df.Sex.value_counts()
```

```
M    104
F     96
Name: Sex, dtype: int64
```

```
# Sex Distribution
plt.figure(figsize=(9,5))
sns.countplot(x = df.Sex)
plt.show()
```



- The ratio of gender seems balanced in the data
- This is a categorical variable. It would be better if we apply label encoder to avoid any error during model implementation.

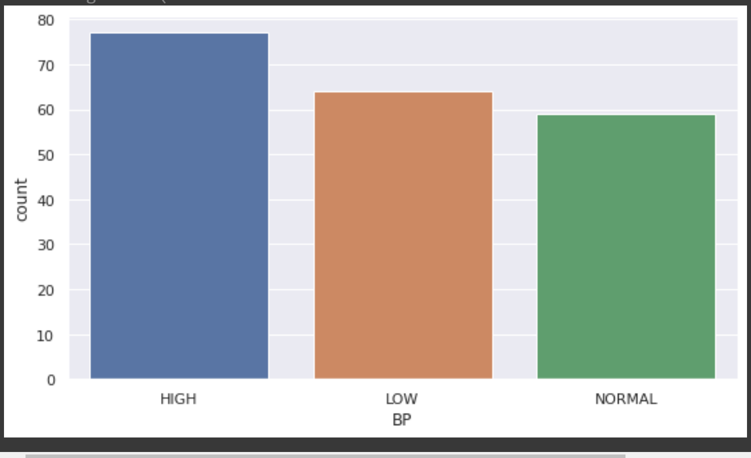
BP Variable

```
df.BP.value_counts()
```

```
HIGH      77
LOW       64
NORMAL    59
Name: BP, dtype: int64
```

```
plt.figure(figsize = (9,5))
sns.countplot(df.BP)
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From
  warnings.warn(
```



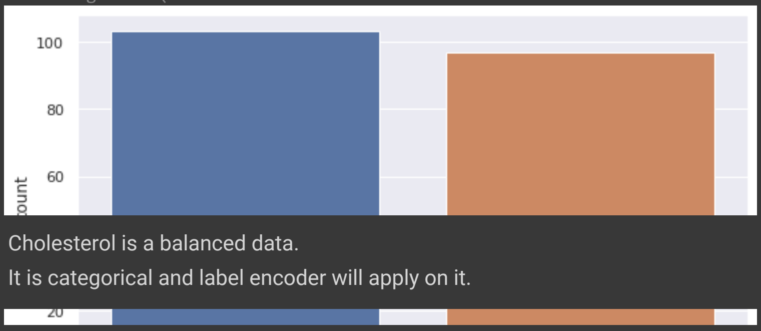
Cholesterol Variable

```
df.Cholesterol.value_counts()
```

```
HIGH      103
NORMAL     97
Name: Cholesterol, dtype: int64
```

```
plt.figure(figsize = (9,5))
sns.countplot(df.Cholesterol)
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From
  warnings.warn(
```



- Cholesterol is a balanced data.
- It is categorical and label encoder will apply on it.

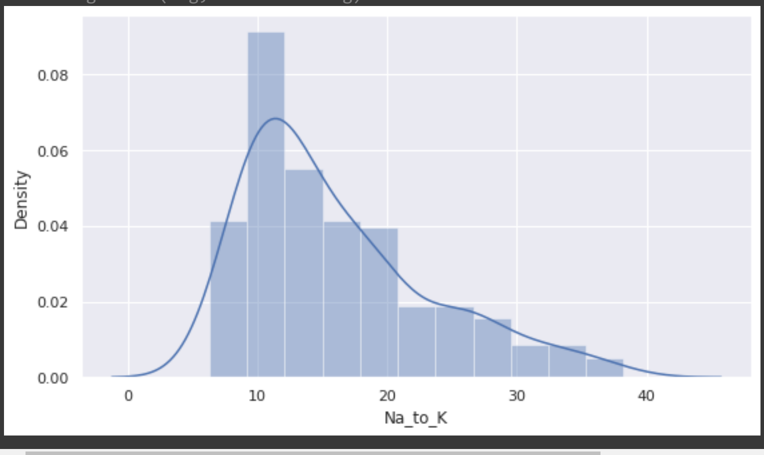
Na_to_K Variable

```
print("Max Na_to_K:",df.Na_to_K.max())
print("Min Na_to_K:",df.Na_to_K.min())
print("Mean Na_to_K:",df.Na_to_K.mean())
```

```
Max Na_to_K: 38.247
Min Na_to_K: 6.269
Mean Na_to_K: 16.084485
```

```
plt.figure(figsize = (9,5))
sns.distplot(df.Na_to_K)
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be re
  warnings.warn(msg, FutureWarning)
```



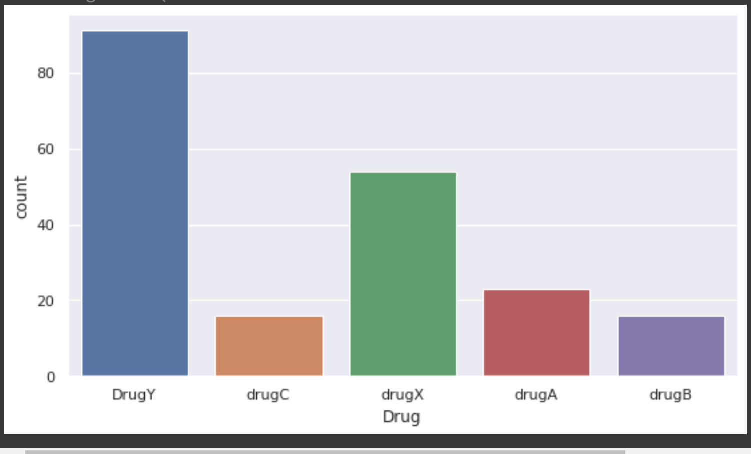
Drug Variable

```
df.Drug.value_counts()
```

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

```
plt.figure(figsize = (9,5))
sns.countplot(df.Drug)
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From
  warnings.warn(
```



- Drug is target column and you can see that it is unbalanced dataset. Using K Fold cross-validation would be better for reliable results.

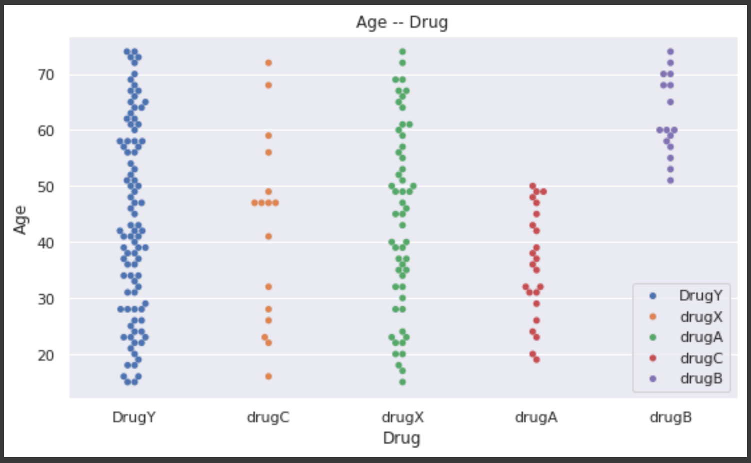
Basic Data Analysis

- Age -- Drug
- Sex -- Drug

- BP -- Drug
- Cholesterol -- Drug

Age -- Drug

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



```
print("Minimum Age of DrugB",df.Age[df.Drug == "drugB"].min())
print("Maximum Age of DrugA",df.Age[df.Drug == "drugA"].max())
```

```
Minimum Age of DrugB 51
Maximum Age of DrugA 50
```

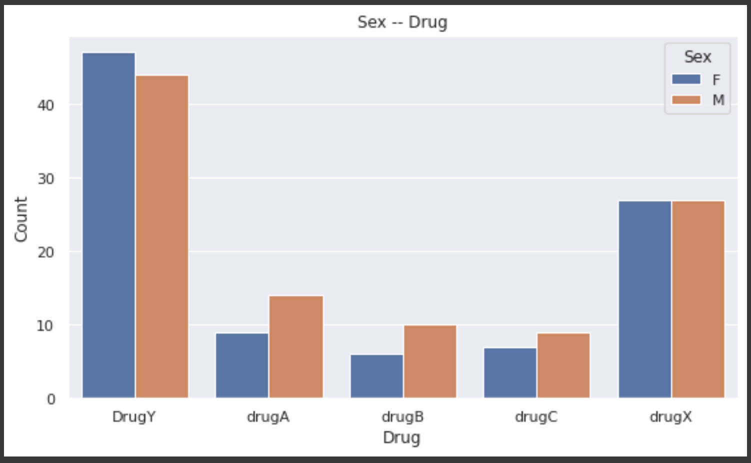
- DrugB is taken only by older than 51 years old.
- DrugA is taken only by younger than 50 years old.

Sex -- Drug

```
df_Sex_Drug = df.groupby(["Drug","Sex"]).size().reset_index(name = "Count")
df_Sex_Drug
```

	Drug	Sex	Count
0	DrugY	F	47
1	DrugY	M	44
2	drugA	F	9
3	drugA	M	14
4	drugB	F	6
5	drugB	M	10
6	drugC	F	7
7	drugC	M	9
8	drugX	F	27
9	drugX	M	27

```
plt.figure(figsize = (9,5))
sns.barplot(x = "Drug",y="Count", hue = "Sex",data = df_Sex_Drug)
plt.title("Sex -- Drug")
plt.show()
```



- Male people get drugA, drugB and drugC more than male people.
- Female people get DrugY more than female people.
- drugX seems equal for male and female people.

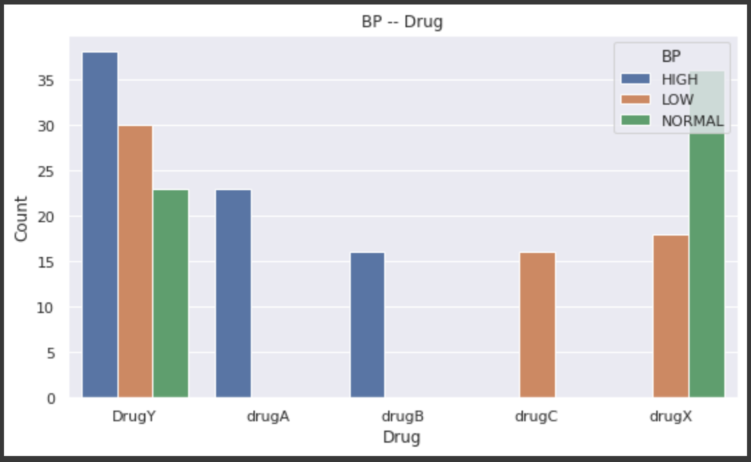
- According to this graph, Sex feature is not an important feature for classification.

BP -- Drug

```
df_BP_Drug = df.groupby(["Drug","BP"]).size().reset_index(name = "Count")
df_BP_Drug
```

	Drug	BP	Count
0	DrugY	HIGH	38
1	DrugY	LOW	30
2	DrugY	NORMAL	23
3	drugA	HIGH	23
4	drugB	HIGH	16
5	drugC	LOW	16
6	drugX	LOW	18
7	drugX	NORMAL	36

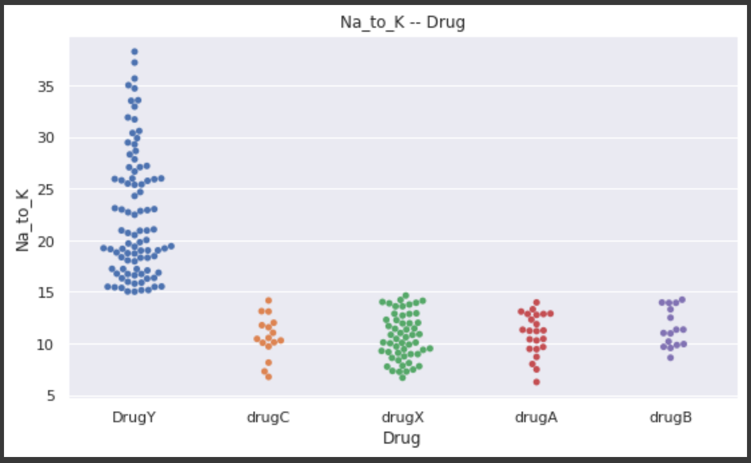
```
plt.figure(figsize = (9,5))
sns.barplot(x = "Drug",y="Count", hue = "BP",data = df_BP_Drug)
plt.title("BP -- Drug")
plt.show()
```



- drugA and drugB are got only by people who have HIGH blood pressure.
- drugC is got by people who have LOW blood pressure.
- drugX is got by people who have HIGH blood pressure.
- BP is an important feature for classification.

Na_to_K -- Drug

```
plt.figure(figsize = (9,5))
sns.swarmplot(x = "Drug", y = "Na_to_K",data = df)
plt.title("Na_to_K -- Drug")
plt.show()
```



```
print("Minimum Na_to_K for DrugY:",df.Na_to_K[df.Drug == "DrugY"].min())
```

```
Minimum Na_to_K for DrugY: 15.015
```

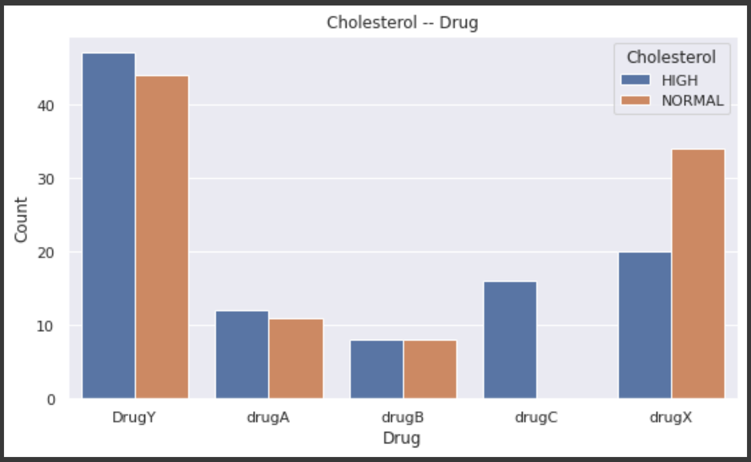
- People who have Na_to_K ratio is bigger than 15, get DrugY.
- We can create a new feature from here.

Cholesterol -- Drug

```
df_CH_Drug = df.groupby(["Drug","Cholesterol"]).size().reset_index(name = "Count")
df_CH_Drug
```

	Drug	Cholesterol	Count
0	DrugY	HIGH	47
1	DrugY	NORMAL	44
2	drugA	HIGH	12
3	drugA	NORMAL	11
4	drugB	HIGH	8
5	drugB	NORMAL	8
6	drugC	HIGH	16
7	drugX	HIGH	20
8	drugX	NORMAL	34

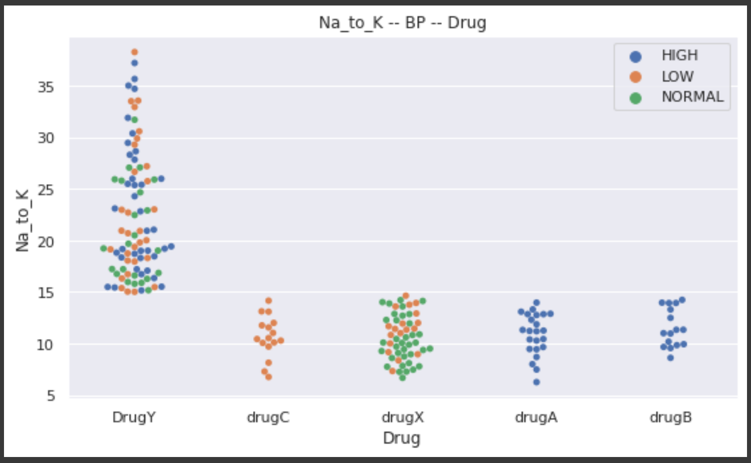
```
plt.figure(figsize = (9,5))
sns.barplot(x = "Drug",y="Count", hue = "Cholesterol",data = df_CH_Drug)
plt.title("Cholesterol -- Drug")
plt.show()
```



- drugC is got by people who have HIGH cholesterol.
- Cholesterol is an important feature to classify drugC

Na_to_K -- BP -- Drug

```
plt.figure(figsize = (9,5))
sns.swarmplot(x = "Drug", y = "Na_to_K",hue="BP",data = df)
plt.legend()
plt.title("Na_to_K -- BP -- Drug")
plt.show()
```



- If people have HIGH blood pressure and Na_to_K ratio is lower than 15 , they get drugA and drugB only.
- If people have LOW blood pressure and Na_to_K ratio is lower than 15 , they get drugC only.

Preparing Data and Feature Engineering

Create New Features

Na_to_K_Bigger_Than_15

If Na_to_K is bigger than 15, it is always drugY.

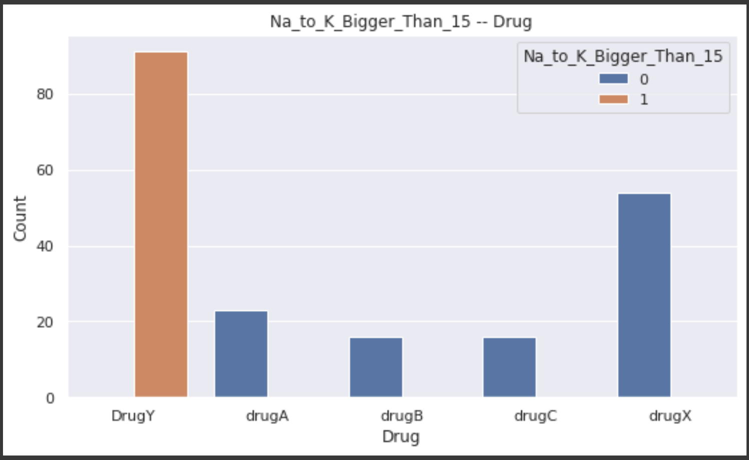
```
df['Na_to_K_Bigger_Than_15'] = [1 if i >=15.015 else 0 for i in df.Na_to_K]
df.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug	Na_to_K_Bigger_Than_15
0	23	F	HIGH	HIGH	25.355	DrugY	1
1	47	M	LOW	HIGH	13.093	drugC	0
2	47	M	LOW	HIGH	10.114	drugC	0
3	28	F	NORMAL	HIGH	7.798	drugX	0
4	61	F	LOW	HIGH	18.043	DrugY	1

```
df_NaK15 = df.groupby(["Drug","Na_to_K_Bigger_Than_15"]).size().reset_index(name = "Count")
df_NaK15
```

	Drug	Na_to_K_Bigger_Than_15	Count
0	DrugY	1	91
1	drugA	0	23
2	drugB	0	16
3	drugC	0	16
4	drugX	0	54

```
plt.figure(figsize = (9,5))
sns.barplot(x = "Drug",y="Count", hue = "Na_to_K_Bigger_Than_15",data = df_NaK15)
plt.title("Na_to_K_Bigger_Than_15 -- Drug")
plt.show()
```



- Na_to_K_Bigger_Than_15 feature will be important feature to drugY classification.

Label Encoding

We will convert from object to int64

- Sex
- BP
- Cholesterol
- Na_to_K
- Na_to_K_Bigger_Than_15

```
from sklearn.preprocessing import LabelEncoder

def label_encoder(y):
    le = LabelEncoder()
    df[y] = le.fit_transform(df[y])
```

```
label_list = ["Sex","BP","Cholesterol","Na_to_K","Na_to_K_Bigger_Than_15","Drug"]

for l in label_list:
    label_encoder(l)
```

```
df.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug	Na_to_K_Bigger_Than_15
0	23	0	0	0	167	0	1
1	47	1	1	0	89	3	0
2	47	1	1	0	43	3	0
3	28	0	2	0	10	4	0
4	61	0	1	0	133	0	1

Train Test Split

```
from sklearn.model_selection import train_test_split

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

```
y = df.Drug

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2, random_state = 42, shuffle = True)

y_train = y_train.values.reshape(-1,1)
y_test = y_test.values.reshape(-1,1)

print("x_train shape:",x_train.shape)
print("x_test shape:",x_test.shape)
print("y_train shape:",y_train.shape)
print("y_test shape:",y_test.shape)
```

```
x_train shape: (160, 6)
x_test shape: (40, 6)
y_train shape: (160, 1)
y_test shape: (40, 1)
```

Data was splitted as 80% train data and 20% test data.

KNN Classifier

To find best score of KNN model, We will try different value of n_neighbors, p, and weights parameters.

Default Parameters

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
knnpred=knn.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test, knnpred))
```

```
Accuracy: 0.65
/usr/local/lib/python3.8/dist-packages/sklearn/neighbors/_classification.py:198: DataConversionWarning: A column-vector y was passed wh
  return self._fit(X, y)
```

```
print (metrics.classification_report(y_test, knnpred))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        15
           1       0.38      0.50      0.43         6
           2       0.50      0.67      0.57         3
           3       0.50      0.20      0.29         5
           4       0.50      0.45      0.48        11

    accuracy                           0.65        40
   macro avg       0.56      0.56      0.55        40
weighted avg       0.65      0.65      0.64        40
```

Random Forest

To find best score of Random Forest model, we will try different value of n_estimators and criterion parameters.

Default Parameters

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state = 42)
rfc.fit(x_train,y_train)
rf_pred = rfc.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test, rf_pred))
```

```
Accuracy: 0.975
<ipython-input-40-f188aa6b6aaf>:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  rfc.fit(x_train,y_train)
```

```
print (metrics.classification_report(y_test, rf_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         3
           3       1.00      0.80      0.89         5
           4       0.92      1.00      0.96        11

    accuracy                           0.97        40
   macro avg       0.98      0.96      0.97        40
weighted avg       0.98      0.97      0.97        40
```

SVM Classifier

To find best score of SVM model, we will try different value of C, kernel, degree and gamma parameters. The easy way to do this is GridSearchCV method.

Default Parameters

```
from sklearn.svm import SVC
svc = SVC(random_state = 42)
svc.fit(x_train,y_train)

prediction=svc.predict(x_test)
print('The accuracy of the SVM is:',metrics.accuracy_score(prediction,y_test))
```

```
The accuracy of the SVM is: 0.65
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d a
  y = column_or_1d(y, warn=True)
```

```
print (metrics.classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        15
           1       0.00      0.00      0.00         6
           2       0.00      0.00      0.00         3
           3       0.00      0.00      0.00         5
           4       0.46      1.00      0.63        11

    accuracy                           0.65        40
   macro avg       0.28      0.40      0.32        40
weighted avg       0.48      0.65      0.54        40

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
```

Logistic Regression

```
model = LogisticRegression()
model.fit(x_train, y_train)
y_predicted = model.predict(x_test)
print('The accuracy of the Logistic Regression is',metrics.accuracy_score(y_predicted,y_test))
```

```
The accuracy of the Logistic Regression is 0.85
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d a
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
print(classification_report(y_test,y_predicted ))
```

```
              precision    recall  f1-score   support

           0       0.88      1.00      0.94        15
           1       1.00      0.83      0.91         6
           2       1.00      1.00      1.00         3
           3       0.00      0.00      0.00         5
           4       0.73      1.00      0.85        11

    accuracy                           0.85        40
   macro avg       0.72      0.77      0.74        40
weighted avg       0.76      0.85      0.80        40

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
```

Naive Bayes

```
Nb= BernoulliNB()
Nb.fit(x_train, y_train)
y_predict_nb = Nb.predict(x_test)
nbs=metrics.accuracy_score(y_predict_nb,y_test)
print('The accuracy of the Naive Bayes is',metrics.accuracy_score(y_predict_nb,y_test))
```

```
The accuracy of the Naive Bayes is 0.8
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d a
  y = column_or_1d(y, warn=True)
```

```
print(classification_report(y_test,y_predict_nb ))
```

```
              precision    recall  f1-score   support
```

```
           0       1.00      1.00      1.00        15
           1       0.67      1.00      0.80         6
           2       0.00      0.00      0.00         3
           3       0.00      0.00      0.00         5
           4       0.69      1.00      0.81        11

    accuracy                           0.80        40
   macro avg       0.47      0.60      0.52        40
weighted avg       0.66      0.80      0.72        40

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
```

Conclusion

```
print('Naive Bayes:          ',nbs)
print('KNN:                  ',metrics.accuracy_score(y_test, knnpred))
print('Logistic Regression:  ',metrics.accuracy_score(y_predicted,y_test))
print('SVM:                  ',metrics.accuracy_score(prediction,y_test))
print('Random Forest:        ',metrics.accuracy_score(y_test, rf_pred))
```

```
Naive Bayes:           0.8
KNN:                   0.65
Logistic Regression:   0.85
SVM:                   0.65
Random Forest:         0.975
```

✓ 0s completed at 7:32 PM