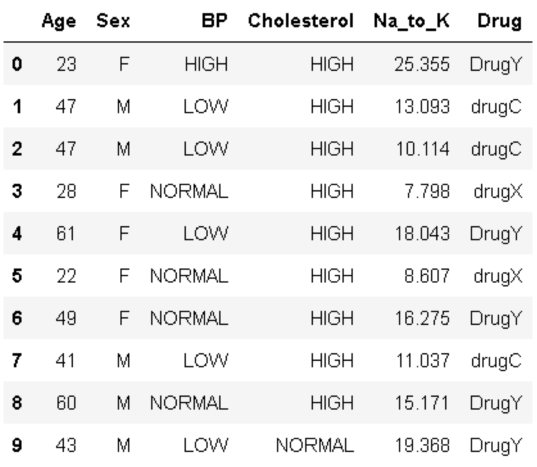
Data Colletion and Exploratory

Dataset information:-

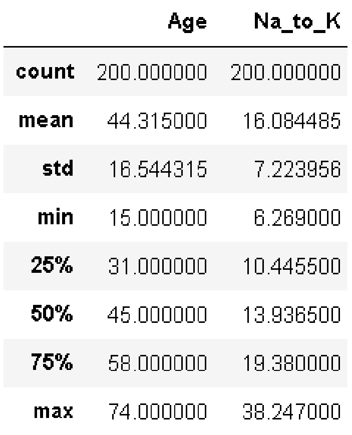
- Name: Drug Classification
- Link: https://www.kaggle.com/prathamtripathi/drug-classification
- File name: drug200.csv
- Target: Drug type (DrugY, drugC, drugX, drugA, drugB)
- Features: Age, Sex, Blood Pressure Levels (BP), Cholesterol Levels (Cholesterol), Sodium to Potassium Ratio (Na_to_K)
- Pandas DataFrame



	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

- Number of rows and columns

```
(200, 6)
```



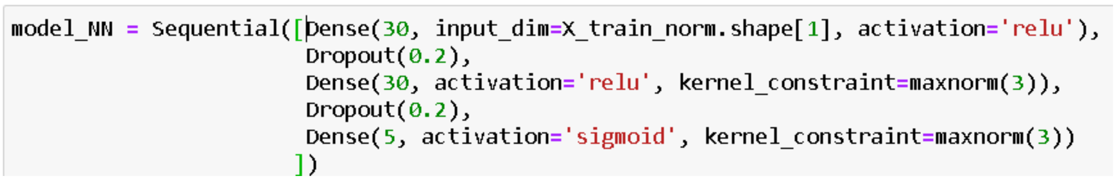
	Age	Na_to_K
count	200.000000	200.000000
mean	44.315000	16.084485
std	16.544315	7.223956
min	15.000000	6.269000
25%	31.000000	10.445500
50%	45.000000	13.936500
75%	58.000000	19.380000
max	74.000000	38.247000

Above figure shows the descriptive statistics on numeric columns. The average age of drug users to be 44 years old, and the age range to be 15 to 74 years old. The average ratio of Sodium to Potassium is 16.08:1, and the range to be 6.269:1 to 38.247:1.

Train test split

Before training of the model, the categorized features and target will be encoded similar to the OneHotEncoder from scikit-learn library. By dummifying the categorized columns into binary columns. By having a 80% of training set with a 20% of test set. Hence, normalization on the features to ensure the numeric columns have a common scale, while reduce the error in measurement.

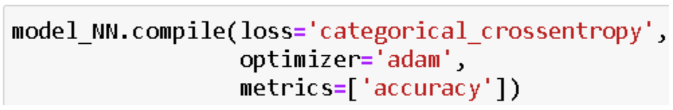
Model (Neural Network)



```
model_NN = Sequential([Dense(30, input_dim=X_train_norm.shape[1], activation='relu'),
                       Dropout(0.2),
                       Dense(30, activation='relu', kernel_constraint=maxnorm(3)),
                       Dropout(0.2),
                       Dense(5, activation='sigmoid', kernel_constraint=maxnorm(3))
                      ])
```

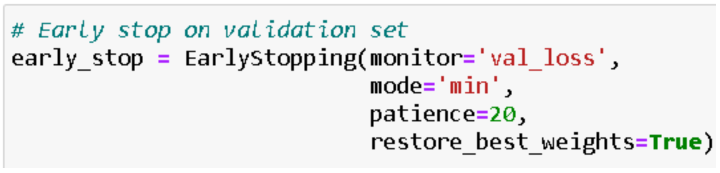
A Sequential model from Keras is created to define a sequence of layers for neural network training. Each of the lines represents the layers for the network architecture. At the beginning, we must ensure the *input_dim*, input layer is set to the number of input feature. *Dense* is the fully connected layers, by specifying the number of neurons in the first layer, and specify its *activation*, activation argument. The rectified linear unit activation function (ReLU) is used in the first both hidden layers as it trains faster than using sigmoid function, and the "Softmax" function as for the multiclass classification problem in the output layer. However, both hidden layers have 30 nodes respectively.

Two dropout regularization are implemented between two hidden layers. A dropout rate of 20% is used as a weight constraint on these layers. For example, 6 inputs will be randomly excluded from each update cycle for 20% dropout rate from 30 connecting nodes. A small dropout value provides a better performance and show a good results.



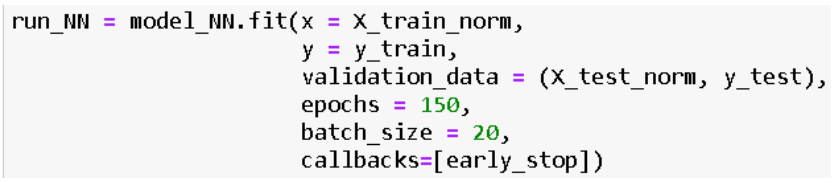
```
model_NN.compile(loss='categorical_crossentropy',
                 optimizer='adam',
                 metrics=['accuracy'])
```

By compiling the defined model, some additional properties required to be specified for the network training. Since the target is multiclass, we set the *loss* argument to be "categorical_crossentropy". The select the efficient stochastic gradient descent optimizer "adam". In the training, the classifcation problem is focusing on the accuracy metrics.



```
# Early stop on validation set
early_stop = EarlyStopping(monitor='val_loss',
                           mode='min',
                           patience=20,
                           restore_best_weights=True)
```

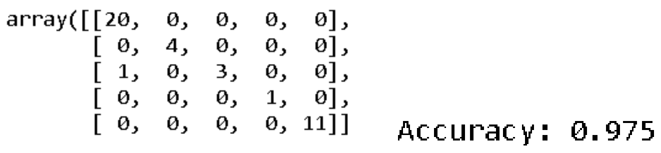
Before starts to execute the model, the early stopping will be used to prevent the validation accuracy starts to decrease during the training. The hyperparamters simply monitor the loss on the validation loss on the 20 epochs before the loss starts to increase. Otherwise, it will cause overfitting.



```
run_NN = model_NN.fit(x = X_train_norm,
                      y = y_train,
                      validation_data = (X_test_norm, y_test),
                      epochs = 150,
                      batch_size = 20,
                      callbacks=[early_stop])
```

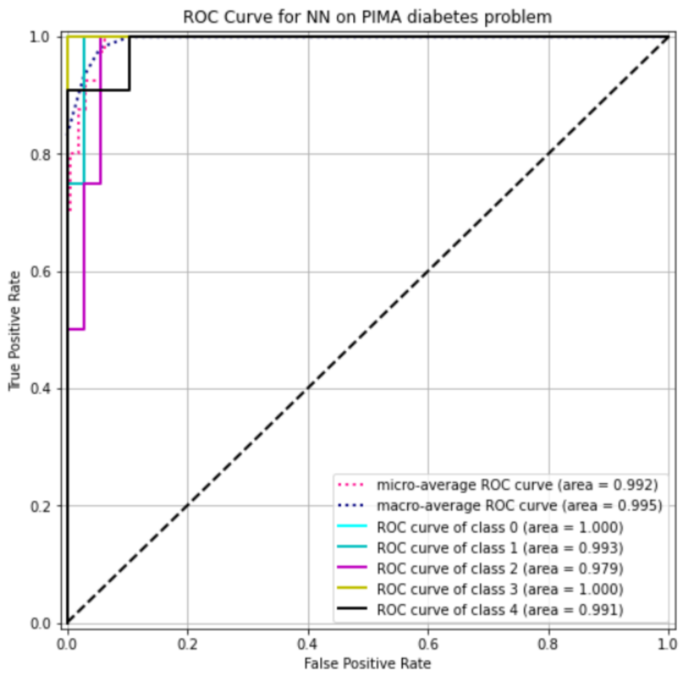
The training process will be executed with a selected 20 batches on each of the 150 epochs. The validation data will be based on the test set of the training.

Evaluation Metrics (Neural Network)

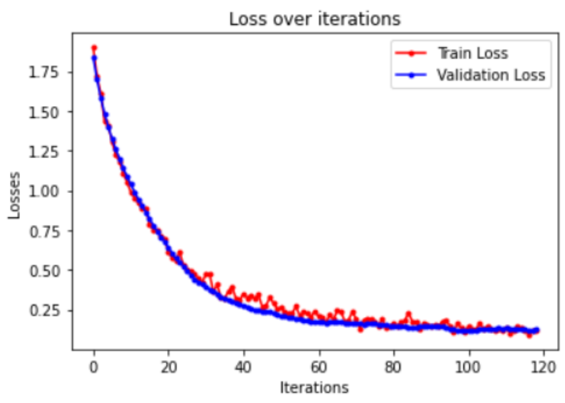


```
array([[20,  0,  0,  0,  0],
       [ 0,  4,  0,  0,  0],
       [ 1,  0,  3,  0,  0],
       [ 0,  0,  0,  1,  0],
       [ 0,  0,  0,  0, 11]])    Accuracy: 0.975
```

The confusion matrix shows the summary of prediction results on each classes based on the predicted values to its actual values. Thus, an accuracy of 0.975 to the predictions that were correct for the following training.



The receiver operating characteristic curve is shown as the performance of the neural network model for each of the classes in the target, type of drugs. From the above figure, each of the classes are having high accuracy with the area under the curve more than 0.9.

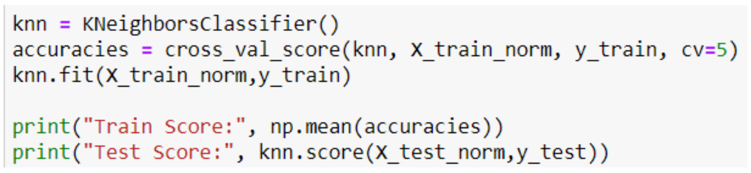


In the figure of loss over iterations, the losses decreases exponentially as the iterations continue. The training is stopped right before the difference between validation loss and train loss getting bigger, where it would cause overfitting.

Table below shows the average accuracy values based on hidden layers in 10 observations.

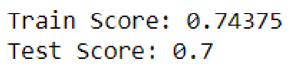
Dense (hidden layers)	Mean accuracy (10 obsevations)
10, 10	0.930
30, 30	0.975
100, 100	0.945

Model (K-Nearest Neighbors)



```
knn = KNeighborsClassifier()
accuracies = cross_val_score(knn, X_train_norm, y_train, cv=5)
knn.fit(X_train_norm,y_train)

print("Train Score:", np.mean(accuracies))
print("Test Score:", knn.score(X_test_norm,y_test))
```

First, in order to train and test our model using cross-validation, we will use the ‘cross_val_score’ function with a cross-validation value of 5. We train the new model by using the ‘fit’ function and pass in our training data as parameters to fit our model to the training data. The train score is 0.74375, it is the mean of the accuracies got from the cross-validation score. We will use the ‘score’ function and pass in our test input and target data to see how well our model predictions match up to the actual results. Our model has an accuracy of approximately 70%.

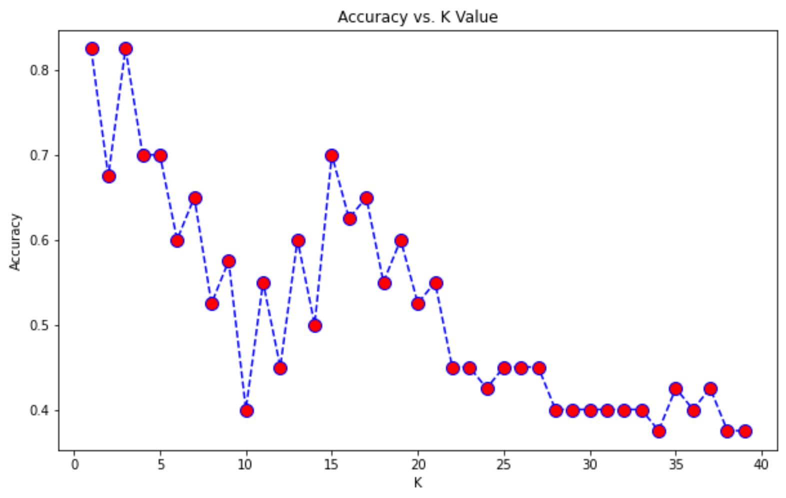


```
Train Score: 0.74375
Test Score: 0.7
```

```
# Get predefined hyperparameters
grid = {'n_neighbors' : np.arange(1,120),
        'p' : np.arange(1,3),
        'weights' : ['uniform','distance']
       }

knn = KNeighborsClassifier(algorithm = "auto")
knn_cv = GridSearchCV(knn,grid,cv=5)
knn_cv.fit(X_train_norm,y_train)

print("Hyperparameters:", knn_cv.best_params_)
print("Train Score:", knn_cv.best_score_)
print("Test Score:", knn_cv.score(X_test_norm, y_test))

acc = []
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors = i).fit(X_train_norm,y_train)
    y_pred = knn.predict(X_test_norm)
    acc.append(accuracy_score(y_test, y_pred))

# Plot the K values for KNN model with its accuracy
plt.figure(figsize=(10,6))
plt.plot(range(1,40),acc,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('Accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
print("Maximum accuracy:",max(acc),"at K =",acc.index(max(acc))+1)
```

To get a parameter for the KNN model, we used the ‘GridSearchCV’ function to compute the best hyperparameters, best train score and their best test score. By plotting the graph accuracy vs K value, we get the maximum accuracy of 0.825 at K=1.

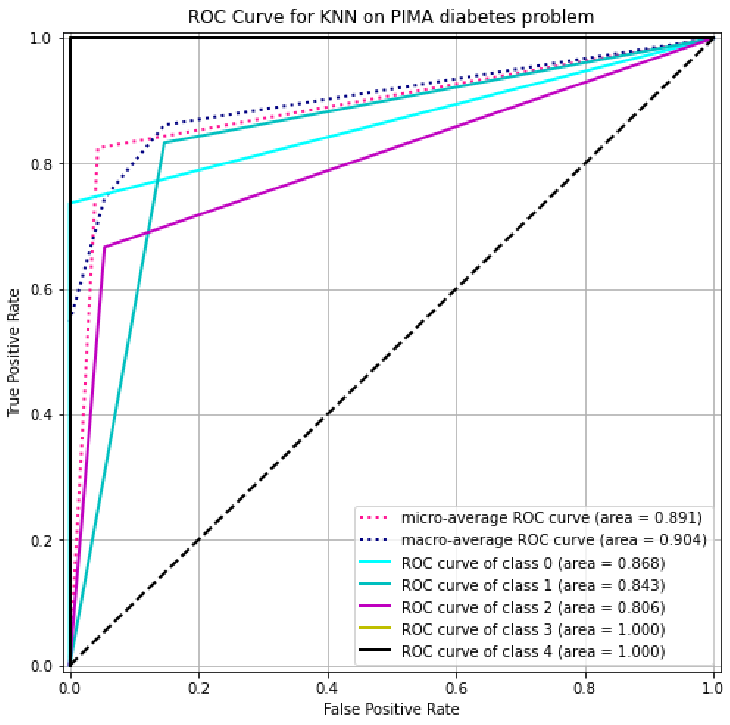
```
Hyperparameters: {'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
Train Score: 0.8625
Test Score: 0.85
Maximum accuracy: 0.825 at K = 1
```



Evaluation Metrics (K-Nearest Neighbors)

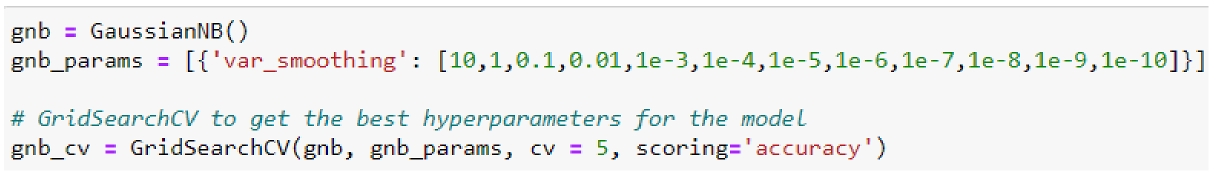
```
[[14  4  1  0  0]
 [ 0  5  1  0  0]
 [ 0  1  2  0  0]
 [ 0  0  0  4  0]
 [ 0  0  0  0  8]]

Accuracy: 0.825
```

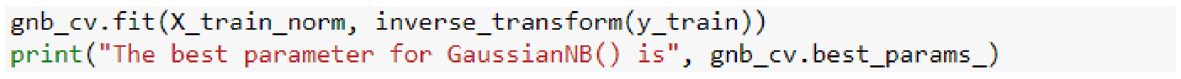
The confusion matrix shows the summary of prediction results on each class based on the predicted values to its actual values. Thus, an accuracy of 0.825 to the predictions that were correct for the following training.

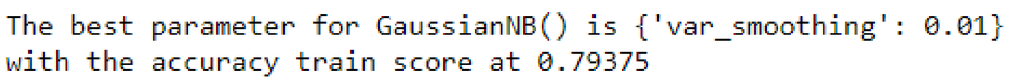


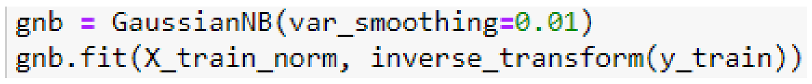
The receiver operating characteristic curve is shown as the performance of the K-Nearest Neighbors model for each of the classes in the target, type of drugs. From the above figure, each of the classes are having outstanding accuracy of the test with the area under the curve more than 0.8.

Model (Gaussian Naïve Bayes)


```
gnb = GaussianNB()
gnb_params = [{'var_smoothing': [10,1,0.1,0.01,1e-3,1e-4,1e-5,1e-6,1e-7,1e-8,1e-9,1e-10]}]

# GridSearchCV to get the best hyperparameters for the model
gnb_cv = GridSearchCV(gnb, gnb_params, cv = 5, scoring='accuracy')
```


```
gnb_cv.fit(X_train_norm, inverse_transform(y_train))
print("The best parameter for GaussianNB() is", gnb_cv.best_params_)
```


```
The best parameter for GaussianNB() is {'var_smoothing': 0.01}
with the accuracy train score at 0.79375
```


```
gnb = GaussianNB(var_smoothing=0.01)
gnb.fit(X_train_norm, inverse_transform(y_train))
```

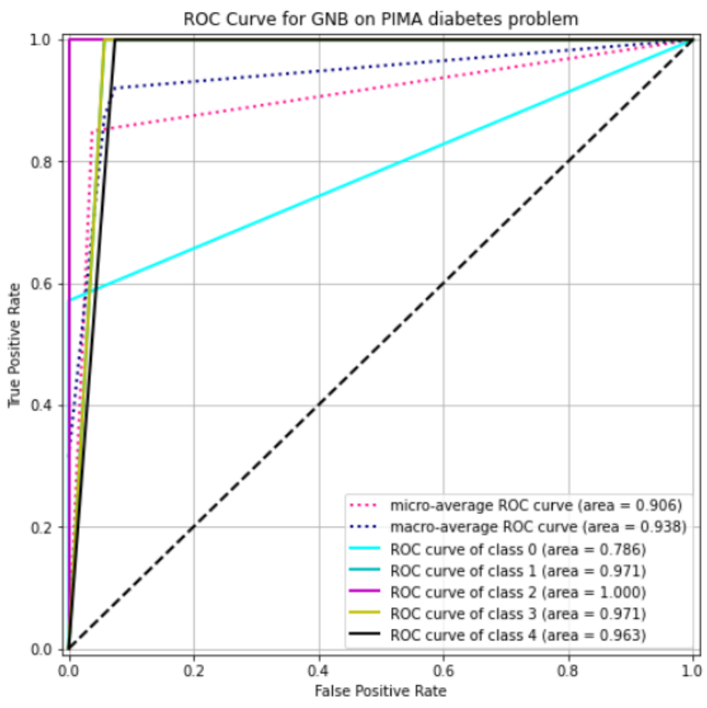
For GNB method, we used sklearn.naive_bayes.GaussianNB function

We adjust the variance distribution of GNB to get train model with best accuracy and got to var_smoothing = 0.01. The parameter is inserted to the GNB and train set is trained.

Evaluation Metrics (Gaussian Naive-Bayer)

```
[[ 8  2  0  2  2]
 [ 0  5  0  0  0]
 [ 0  0  3  0  0]
 [ 0  0  0  5  0]
 [ 0  0  0  0 13]]

Accuracy: 0.850
```

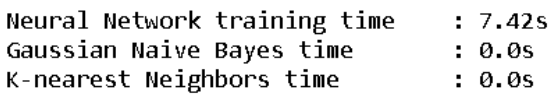
The confusion matrix shows the summary of prediction results on each class based on the predicted values to its actual values. Thus, an accuracy of 0.850 to the predictions that were correct for the following training.



The receiver operating characteristic curve is shown as the performance of the GNB model for each of the classes in the target, type of drugs. From the above figure, each of the classes are having accuracy of over 0.9 except age class with the test result of 0.78 area under the curve.

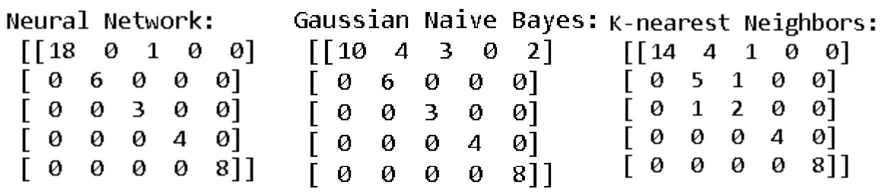
Summary

In conclusion, the evaluation will allow us to compare the difference between these models. Below figure shows the runtime of each model



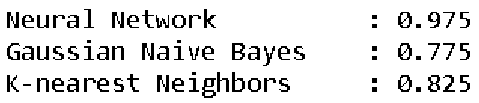
```
Neural Network training time    : 7.42s
Gaussian Naive Bayes time       : 0.0s
K-nearest Neighbors time        : 0.0s
```

Due to the training of neural networks will learn multiple times in each epoch, just to enchance the accuracy in fitting of model, and it takes longer time compared to another two model that runs immediately.



```
Neural Network:        Gaussian Naive Bayes:  K-nearest Neighbors:
 [[18  0  1  0  0]      [[10  4  3  0  2]      [[14  4  1  0  0]
 [ 0  6  0  0  0]       [ 0  6  0  0  0]       [ 0  5  1  0  0]
 [ 0  0  3  0  0]       [ 0  0  3  0  0]       [ 0  1  2  0  0]
 [ 0  0  0  4  0]       [ 0  0  0  4  0]       [ 0  0  0  4  0]
 [ 0  0  0  0  8]]      [ 0  0  0  0  8]]      [ 0  0  0  0  8]]
```

Above figure shows the confusion matrix of each model for the performance on a multiclass classification. Every model can classify last two classes perfectly, while the first class to be the most false positive error. As the result, it may be required more features to train this underfitting of model. In the second and third class, neural network and Gaussian naïve bayes models are able to classify correctly, but not for k-nearest neighbors. However, it may due to the overfitting of model, since K=1 is being used for the highest accuracy in the training.



```
Neural Network          : 0.975
Gaussian Naive Bayes    : 0.775
K-nearest Neighbors     : 0.825
```

Above figure shows the accuracies score for the proposed models. The neural network has the highest accuracy score of 0.975, that outperforms Gaussian naïve bayes and k-nearest neighbors models with the score of 0.775 and 0.825 simultaneously.