MACHINE LEARNING LAB
EXPERIMENT – 5

ANKITH MOTHA
21BAI1284

k-NN Clustering

Dataset: Heart Attack Dataset

Input Variables: age, sex, cp, trtbps, chol, fbs, restecg, thalachh, exng, oldpeak, slp, caa, thall.

Target Variable: Output

About this dataset

- Age : Age of the patient
- Sex : Sex of the patient
- exang: exercise induced angina (1 = yes; 0 = no)
- ca: number of major vessels (0-3)
- cp : Chest Pain type chest pain type
  - Value 1: typical angina
  - Value 2: atypical angina
  - Value 3: non-anginal pain
  - Value 4: asymptomatic
- trtbps : resting blood pressure (in mm Hg)
- chol : cholestoral in mg/dl fetched via BMI sensor
- fbs : (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- rest_ecg : resting electrocardiographic results
  - Value 0: normal
  - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
  - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria.
- thalach : maximum heart rate achieved
- target :
  - 0 - less chance of heart attack;
  - 1 - more chance of heart attack

Overview:

The k-nearest neighbours (KNN) algorithm is a non-parametric, lazy learning algorithm that is used for both classification and regression. It works by finding the k most similar instances in the training set to a new instance, and then assigning the new instance to the class that is most common among the k nearest neighbours.

The k value is a hyperparameter that must be chosen by the user. A higher value of k will give more weight to the more distant neighbours, while a lower value of k will give more weight to the closer neighbours. The optimal value of k will depend on the specific dataset and the desired accuracy.

KNN is a simple algorithm to understand and implement, and it is often used as a baseline algorithm for comparison with other machine learning algorithms. However, it can be computationally expensive to calculate the distances between all pairs of instances in the training set, and it can be sensitive to noise in the data.

Here are the steps on how KNN works:

1. Choose the k value. The k value is a hyperparameter that must be chosen by the user. A higher value of k will give more weight to the more distant neighbours, while a lower value of k will give more weight to the closer neighbours. The optimal value of k will depend on the specific dataset and the desired accuracy.
2. Find the k nearest neighbours. For each new instance, find the k instances in the training set that are most similar to the new instance. This can be done using any distance metric, such as the Euclidean distance or the Manhattan distance.
3. Predict the class of the new instance. The class of the new instance is predicted by the majority vote of its k nearest neighbours. For example, if 3 of the k nearest neighbours are of class A and 2 are of class B, then the new instance is predicted to be of class A.

Here are some of the advantages of KNN:

- It is a simple algorithm to understand and implement.
- It is a non-parametric algorithm, which means that it does not make any assumptions about the underlying distribution of the data.
- It can be used for both classification and regression problems.
- It is a robust algorithm that is not sensitive to noise in the data.

Here are some of the disadvantages of KNN:

- It can be computationally expensive to calculate the distances between all pairs of instances in the training set.
- It can be sensitive to the choice of the hyperparameter k.
- It can be slow to predict new instances, especially for large datasets.

Overall, KNN is a simple and powerful machine learning algorithm that can be used for a variety of tasks. It is a good choice for beginners who are learning about machine learning, and it can also be used as a baseline algorithm for comparison with other machine learning algorithms.

1. Workflow of KNN

Initially, Let's consider 20 samples from the training set of our heart disease dataset and let's apply the KNN algorithm with K-neighbours = 3 on a testing sample.

Test Sample:

Input: [0.47916667, 1., 1., 0.22641509, 0.1369863, 0., 0., 0.90839695, 0., 0., 0.5, 0., 0.33333333]

Expected Result: 1

Training Data for workflow from Training set

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

212

+ Code + Markdown

```
# Using random sampling technique to select 20 samples from the training set to demonstrate the workflow
from random import sample
workflow_indices = sample(list(np.arange(0,X_train.shape[0])),10)
X_work = X_train[workflow_indices]
y_work = y_train[workflow_indices]

pd.DataFrame(X_work, columns=X_cols), pd.DataFrame(y_work, columns=['output'])
```

```
(        age  sex        cp    trtbps      chol  fbs  restecg  thalachh  exng  \
 0  0.312500  1.0  0.333333  0.245283  0.214612  0.0      0.5  0.755725   0.0
 1  0.645833  1.0  0.000000  0.481132  0.356164  0.0      0.0  0.541985   1.0
 2  0.541667  1.0  0.000000  0.622642  0.372146  0.0      0.0  0.564885   1.0
 3  0.770833  0.0  0.666667  0.490566  0.347032  0.0      0.0  0.618321   0.0
 4  0.604167  0.0  0.333333  0.396226  0.440639  1.0      0.0  0.618321   0.0
 5  0.500000  0.0  0.000000  0.415094  0.246575  0.0      0.0  0.679389   0.0
 6  0.520833  1.0  0.666667  0.528302  0.242009  0.0      0.0  0.717557   0.0
 7  0.395833  1.0  0.666667  0.283019  0.294521  1.0      0.5  0.793893   0.0
 8  0.687500  0.0  0.666667  0.339623  0.312785  0.0      0.5  0.198473   0.0
 9  0.395833  0.0  0.666667  0.339623  0.340183  0.0      0.5  0.519084   0.0

     oldpeak  slp   caa     thall
 0  0.000000  1.0  0.00  0.666667
 1  0.451613  0.5  0.50  1.000000
 2  0.129032  0.5  0.25  1.000000
 3  0.000000  0.5  0.25  0.666667
 4  0.000000  1.0  0.50  0.666667
 5  0.000000  1.0  0.00  0.666667
 6  0.258065  1.0  0.00  1.000000
 7  0.000000  1.0  0.50  0.666667
 8  0.193548  0.5  0.25  1.000000
 9  0.032258  1.0  0.00  0.666667  ,
    output
 0       1
 1       0
 2       0
 3       1
 4       0
 5       1
 6       1
 7       1
 8       0
 9       1)
```

```python
class KNN_classifier:
    def __init__(self,k_neighbours):
        self.k_neighbours=k_neighbours

    def minkowski(self,x_tr,x_tt,p=2):  # P=2 refers to euclidean distance
        return (sum([(abs(x_tr[i]-x_tt[i]))**p for i in range(len(x_tr))]))**(1/p)

    def fit(self,X,y):
        self.X = X
        self.y = y

    def predict(self,x_test,p=2):
        distances = []
        for train_id in range(len(self.X)):
            try:
                if len(x_test) != len(self.X[train_id]):
                    raise Exception
                distances.append([self.minkowski(x_test,self.X[train_id],p), self.y[train_id]])
            except:
                print("Length of train data and prediction data not matching")
                break

        distances.sort(key = lambda tup: tup[0])
        freq = np.bincount(np.array(distances, dtype=np.int64)[:self.k_neighbours,1])
        return(np.argmax(freq))
```

As we pass the testing sample as input to the classifier for prediction,

[0.47916667, 1., 1., 0.22641509, 0.1369863, 0., 0., 0.90839695, 0., 0., 0.5, 0., 0.33333333]

We first calculate the distances of the input test vector with the other training data

	distance	label
0	1.054999	1
1	1.781391	0
2	1.690502	0
3	1.252131	1
4	1.810073	0
5	1.568948	1
6	1.006218	1
7	1.421372	1
8	1.580073	0
9	1.391358	1

The distances of the test sample with all the other 10 training samples is calculated using Euclidean distance,

$$d = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}\,.$$

Where x(i) is a feature value of test sample and y(i) is the corresponding feature value of the train sample. "n" is the total number of features in the input.

Next, sort the distances in ascending order and chose the '**k**' nearest neighbours of the testing data based on the sorted distance**.**

	distance	label
0	1.006218	1
1	1.054999	1
2	1.252131	1
3	1.391358	1
4	1.421372	1
5	1.568948	1
6	1.580073	0
7	1.690502	0
8	1.781391	0
9	1.810073	0

It can be seen that the distances have been sorted in ascending order.

We need to now choose the k nearest neighbours, i.e, the first 3 training points which are closest to test point.

Output:

[[1.00621776, 1.]

[1.0549994, 1.]

[1.25213052, 1.]]

Using the principle of **MAX VOTING**, we will choose the labels which are more frequently appearing among the **k nearest neighbours**. This max label will be assigned to the test data.

Output:

[0 3]

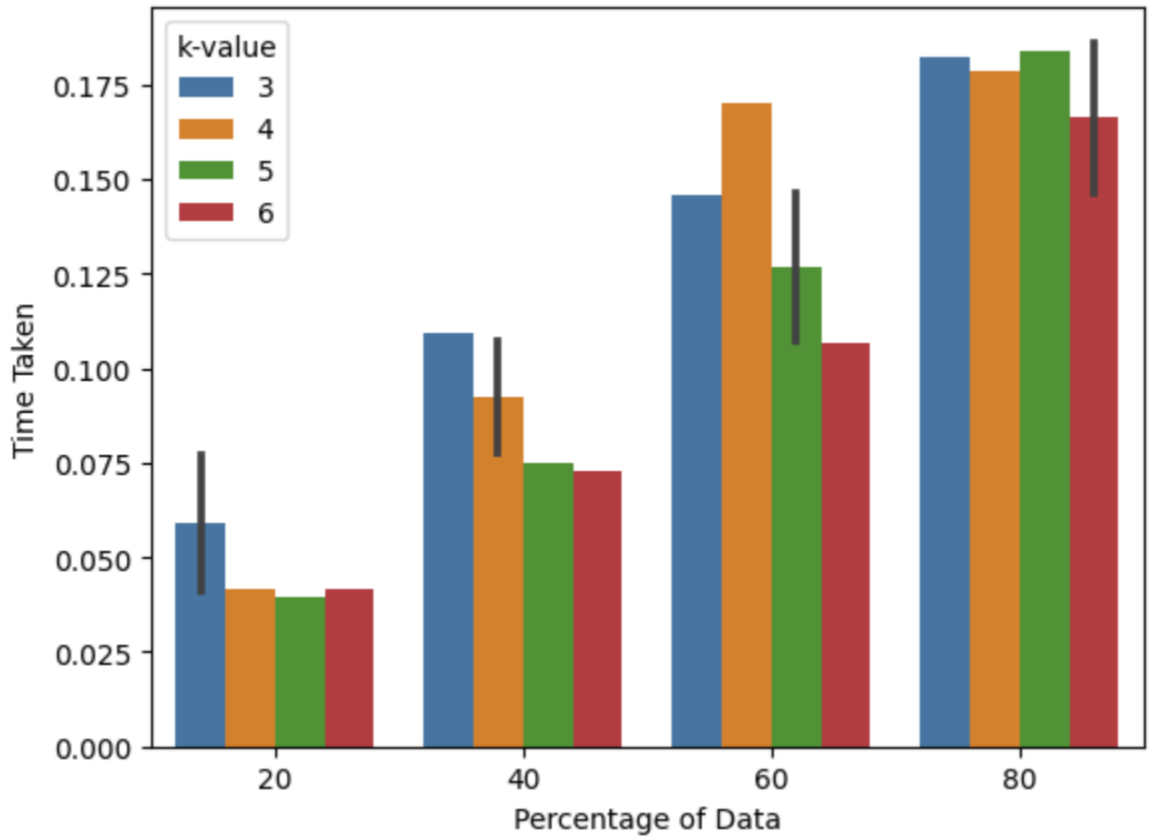
The above results conveys the meaning that there are 3 tuples with Label 1 as answer, and 0 tuples with Label 0. Hence by max voting, the test data will get 1 as the label.

On comparing the true label and predicted label, we find that both are the same **(Label: 1).**

2. Comparative Analysis

1&2) Time Analysis for Train Data vs K along with 30% Test Data(in secs):

Size of Training Data \ K Neighbours	K = 3	K = 4	K = 5	K = 6
20%	0.041011325	0.041398121	0.039613125	0.041703245
40%	0.077000163	0.077383058	0.074809444	0.072773152
60%	0.109130311	0.107329887	0.107174437	0.106888516
80%	0.145766027	0.169836267	0.146234051	0.146342354
100%	0.182187114	0.178715758	0.183862835	0.186098923



3) Testing Data with Different k-values:

Experiment	Precision	Recall	F1-Score	Accuracy
K = 3	0.7778	0.8571	0.8155	0.7912
K = 4	0.8261	0.7755	0.8	0.7912
K = 5	0.7885	0.8367	0.8119	0.7912
K = 6	0.8163	0.8163	0.8163	0.8022

```
Accuracy: 0.7912087912087912
              precision    recall  f1-score   support

           0       0.81      0.71      0.76        42
           1       0.78      0.86      0.82        49

    accuracy                           0.79        91
   macro avg       0.79      0.79      0.79        91
weighted avg       0.79      0.79      0.79        91

(0.7777777777777778, 0.8571428571428571, 0.8155339805825242, None)
```

```
Accuracy: 0.7912087912087912
              precision    recall  f1-score   support

           0       0.76      0.81      0.78        42
           1       0.83      0.78      0.80        49

    accuracy                           0.79        91
   macro avg       0.79      0.79      0.79        91
weighted avg       0.79      0.79      0.79        91

(0.8260869565217391, 0.7755102040816326, 0.8, None)
```

```
Accuracy: 0.7912087912087912
              precision    recall  f1-score   support

           0       0.79      0.74      0.77        42
           1       0.79      0.84      0.81        49

    accuracy                           0.79        91
   macro avg       0.79      0.79      0.79        91
weighted avg       0.79      0.79      0.79        91

(0.7884615384615384, 0.8367346938775511, 0.8118811881188118, None)

Accuracy: 0.8021978021978022
              precision    recall  f1-score   support

           0       0.79      0.79      0.79        42
           1       0.82      0.82      0.82        49

    accuracy                           0.80        91
   macro avg       0.80      0.80      0.80        91
weighted avg       0.80      0.80      0.80        91

(0.8163265306122449, 0.8163265306122449, 0.8163265306122449, None)
```

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

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [ ]:
```
df = pd.read_csv("/kaggle/input/heart-attack-analysis-prediction-dataset/heart.c
df
```

Out[ ]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	..
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2

303 rows × 14 columns

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh',
       'exng', 'oldpeak', 'slp', 'caa', 'thall', 'output'],
      dtype='object')
```

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trtbps    303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalachh  303 non-null    int64  
 8   exng      303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64  
 11  caa       303 non-null    int64  
 12  thall     303 non-null    int64  
 13  output    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [ ]:

```
df.describe()
```

Out[ ]:

	age	sex	cp	trtbps	chol	fbs	reste
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000(
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528(
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525{
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000(
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000(
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000(
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000(
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000(

In [ ]:

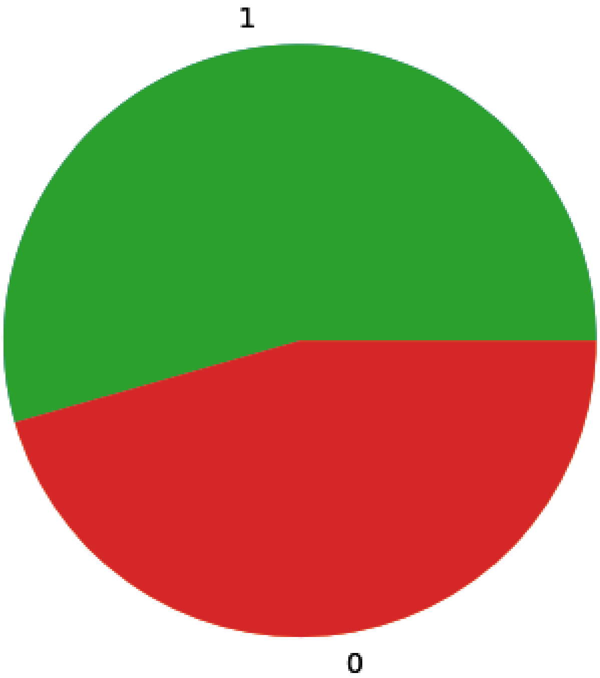
```
# Showing the frequency of categorical data in each feature of the dataset
for i in df.columns:
    if len(df[i].value_counts()) <= 10:
        print(i,":")
        print(df[i].value_counts())
        print()
```

```
sex :
1    207
0     96
Name: sex, dtype: int64

cp :
0    143
2     87
1     50
3     23
Name: cp, dtype: int64

fbs :
0    258
1     45
Name: fbs, dtype: int64

restecg :
1    152
0    147
2      4
Name: restecg, dtype: int64

exng :
0    204
1     99
Name: exng, dtype: int64

slp :
2    142
1    140
0     21
Name: slp, dtype: int64

caa :
0    175
1     65
2     38
3     20
4      5
Name: caa, dtype: int64

thall :
2    166
3    117
1     18
0      2
Name: thall, dtype: int64

output :
1    165
0    138
Name: output, dtype: int64
```

In [ ]:

```
plt.pie(df['output'].value_counts(), labels=pd.unique(df['output']))
plt.pie(df['output'].value_counts(), labels=pd.unique(df['output']))
```

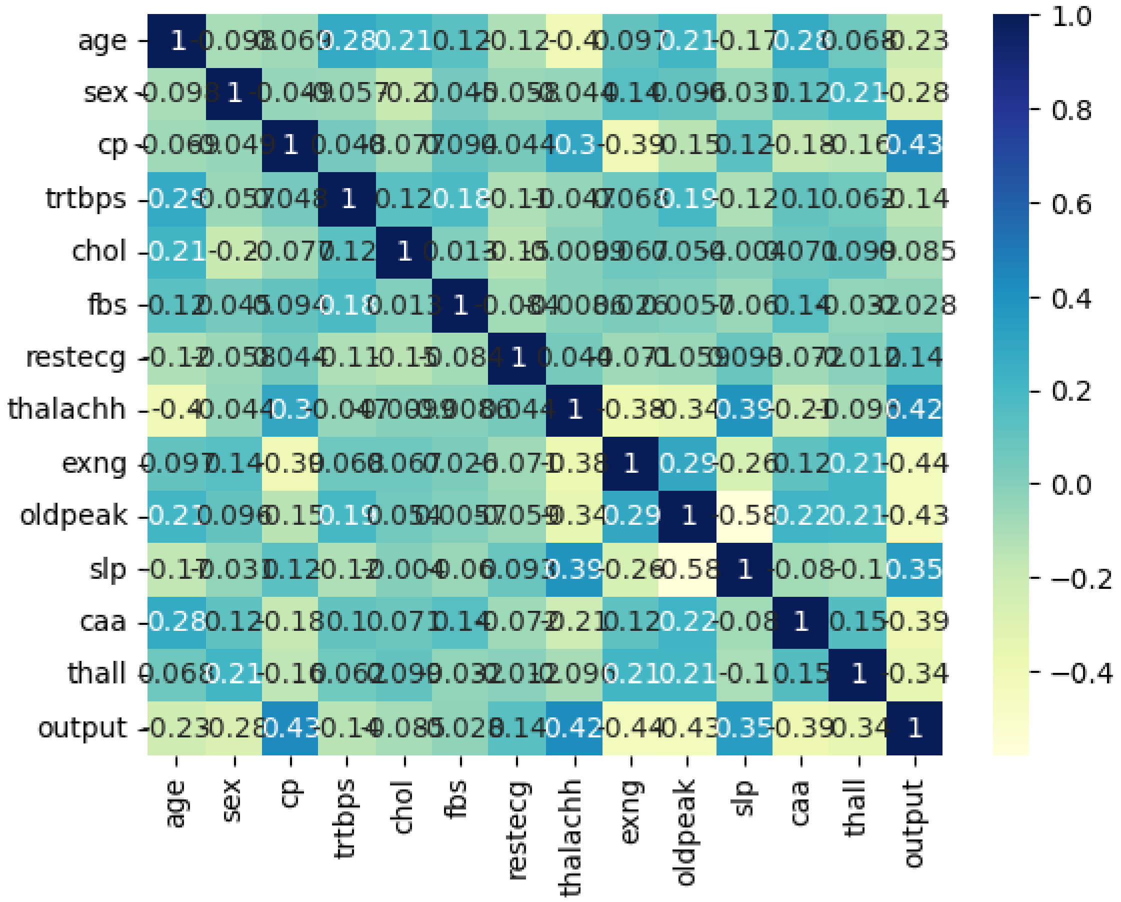
Out[ ]:
```
([<matplotlib.patches.Wedge at 0x7a2ed0aff580>,
  <matplotlib.patches.Wedge at 0x7a2ed0e70df0>],
 [Text(-0.1534669293828939, 1.0892418930548835, '1'),
  Text(0.1534670313650796, -1.0892418786862677, '0')])
```



In [ ]:
```python
y = df['output']
X = df.drop(["output"], axis=1)
X,y
```

```
Out[ ]: (     age  sex  cp  trtbps  chol  fbs  restecg  thalachh  exng  oldpeak  slp  \
 0     63    1   3     145   233    1        0       150     0      2.3    0   
 1     37    1   2     130   250    0        1       187     0      3.5    0   
 2     41    0   1     130   204    0        0       172     0      1.4    2   
 3     56    1   1     120   236    0        1       178     0      0.8    2   
 4     57    0   0     120   354    0        1       163     1      0.6    2   
 ..   ...  ...  ..     ...   ...  ...      ...       ...   ...      ...  ...   
 298   57    0   0     140   241    0        1       123     1      0.2    1   
 299   45    1   3     110   264    0        1       132     0      1.2    1   
 300   68    1   0     144   193    1        1       141     0      3.4    1   
 301   57    1   0     130   131    0        1       115     1      1.2    1   
 302   57    0   1     130   236    0        0       174     0      0.0    1   

      caa  thall  
 0      0      1  
 1      0      2  
 2      0      2  
 3      0      2  
 4      0      2  
 ..   ...    ...  
 298    0      3  
 299    0      3  
 300    2      3  
 301    1      3  
 302    1      2  

 [303 rows x 13 columns],
 0      1
 1      1
 2      1
 3      1
 4      1
       ..
 298    0
 299    0
 300    0
 301    0
 302    0
 Name: output, Length: 303, dtype: int64)
```

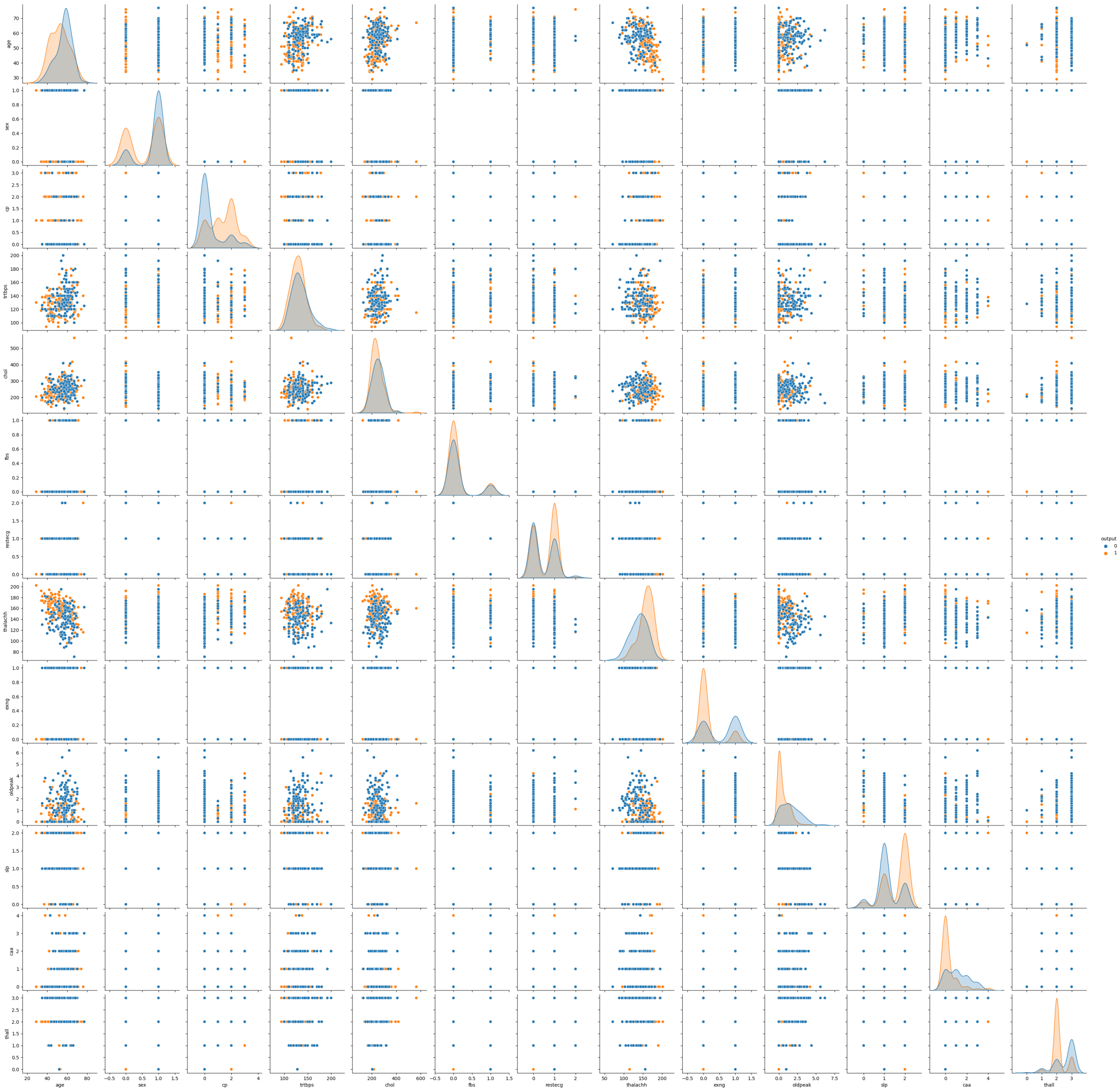
```
In [ ]: dataplot = sns.heatmap(df.corr(), cmap="YlGnBu", annot=True)
```



The correlation above shows that there aren't any highly correlated features in the dataset

In [ ]:
```
sns.pairplot(df,hue="output")
```

Out[ ]: `<seaborn.axisgrid.PairGrid at 0x7a2ece66bb50>`



In [ ]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_cols = X.columns
X = scaler.fit_transform(X)

X = pd.DataFrame(X,columns = X_cols)
X
```

Out[ ]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpea
0	0.708333	1.0	1.000000	0.481132	0.244292	1.0	0.0	0.603053	0.0	0.37096
1	0.166667	1.0	0.666667	0.339623	0.283105	0.0	0.5	0.885496	0.0	0.56451
2	0.250000	0.0	0.333333	0.339623	0.178082	0.0	0.0	0.770992	0.0	0.2258C
3	0.562500	1.0	0.333333	0.245283	0.251142	0.0	0.5	0.816794	0.0	0.12903
4	0.583333	0.0	0.000000	0.245283	0.520548	0.0	0.5	0.702290	1.0	0.09677
...	...	...	...	...	...	...	...	...	...	
298	0.583333	0.0	0.000000	0.433962	0.262557	0.0	0.5	0.396947	1.0	0.03225
299	0.333333	1.0	1.000000	0.150943	0.315068	0.0	0.5	0.465649	0.0	0.19354
300	0.812500	1.0	0.000000	0.471698	0.152968	1.0	0.5	0.534351	0.0	0.54838
301	0.583333	1.0	0.000000	0.339623	0.011416	0.0	0.5	0.335878	1.0	0.19354
302	0.583333	0.0	0.333333	0.339623	0.251142	0.0	0.0	0.786260	0.0	0.0000C

303 rows × 13 columns

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X.to_numpy(),y.to_numpy(),test_
X_train.shape[0]
```

Out[ ]: 212

In [ ]:

```
# Using random sampling technique to select different sizes of training data
from random import sample
perc = 100
samples= int(X_train.shape[0]*(perc/100))
workflow_indices = sample(list(np.arange(0,X_train.shape[0])),samples)
X_work = X_train[workflow_indices]
y_work = y_train[workflow_indices]

samples
```

Out[ ]: 212

In [ ]:

```
class KNN_classifier:
    def __init__(self,k_neighbours):
        self.k_neighbours=k_neighbours

    def minkowski(self,x_tr,x_tt,p=2):  # P=2 refers to euclidean distance
        return (sum([(abs(x_tr[i]-x_tt[i]))**p for i in range(len(x_tr))]))**(1/

    def fit(self,X,y):
        self.X = X
        self.y = y

    def predict(self,x_test,p=2):
        distances = []
        for train_id in range(len(self.X)):
```

```
            try:
                if len(x_test) != len(self.X[train_id]):
                    raise Exception
                distances.append([self.minkowski(x_test,self.X[train_id],p), sel
            except:
                print("Length of train data and prediction data not matching")
                break

        distances.sort(key = lambda tup: tup[0])
        freq = np.bincount(np.array(distances, dtype=np.int64)[:self.k_neighbour
        return(np.argmax(freq))
```

In [ ]:
```
from time import process_time

for k in range(3,7):
    t1 = process_time()
    classifier = KNN_classifier(k)
    classifier.fit(X_work,y_work)

    # Predicting the Test set results
    y_test_pred = []
    for i in X_test:
        y_test_pred.append(classifier.predict(i))

    t2 = process_time()

    print(f"Time taken for training(train data) and predicting(test data) {k}:",

#     from sklearn.metrics import classification_report

#     acc = sum([1 if y_test_pred[i] == y_test[i] else 0 for i in range(len(y_te
#     print("Accuracy:",acc)
#     print(classification_report(y_test,y_test_pred))
```

```
Time taken for training(train data) and predicting(test data) 3: 0.36430781500000
364
Time taken for training(train data) and predicting(test data) 4: 0.34091543600000
307
Time taken for training(train data) and predicting(test data) 5: 0.34207887899999
89
Time taken for training(train data) and predicting(test data) 6: 0.34221773699999
86
```

In [ ]:
```
time_analysis_k = [0.041011325,0.041398121,0.039613125,0.041703245,
0.077000163,0.077383058,0.074809444,0.072773152,
0.109130311,0.107329887,0.107174437,0.106888516,
0.145766027,0.169836267,0.146234051,0.146342354,
0.182187114,0.178715758,0.183862835,0.186098923]

perc = [20,20,20,20,20,40,40,40,40,40,60,60,60,60,60,80,80,80,80,80,100,100,100,
k_vals = [3,4,5,6]*5
# dt = np.concatenate([perc,time_analysis_k,],axis=1)
time_analysis = pd.DataFrame(list(zip(perc, time_analysis_k, k_vals)), columns=[
time_analysis
```

Out[ ]:

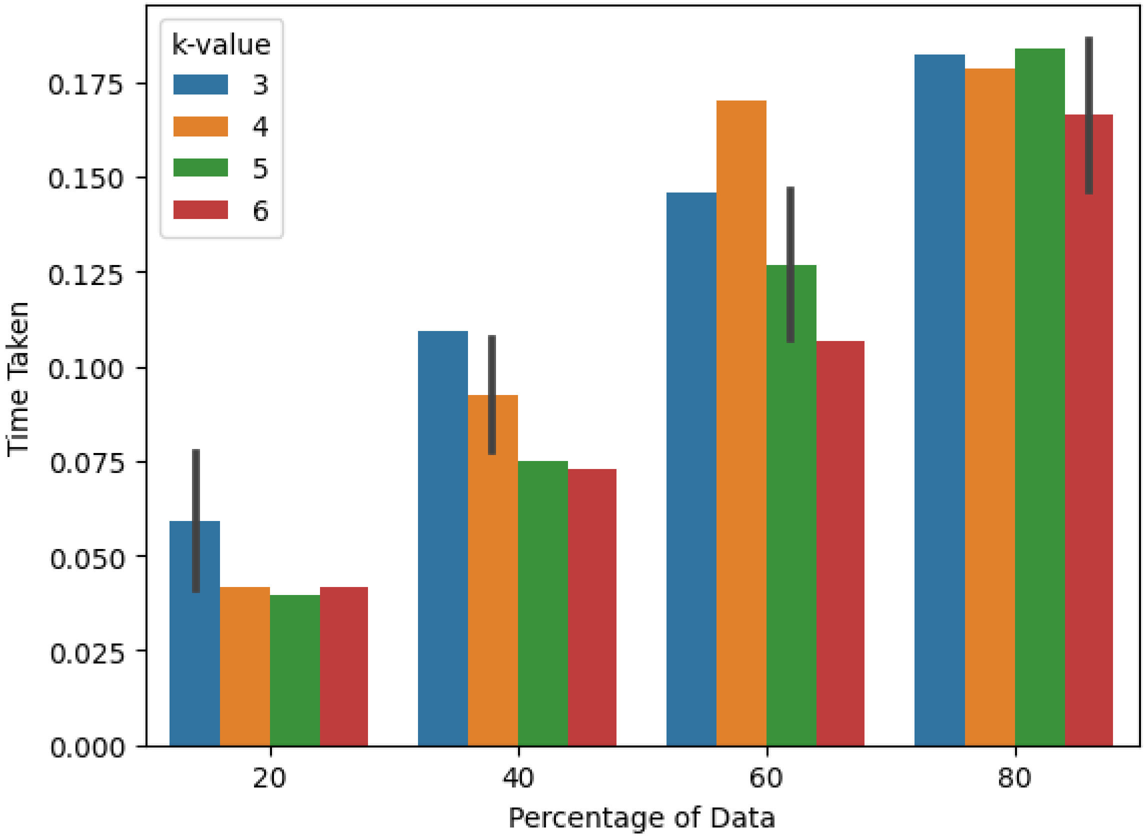
	perc	time	k-value
0	20	0.041011	3
1	20	0.041398	4
2	20	0.039613	5
3	20	0.041703	6
4	20	0.077000	3
5	40	0.077383	4
6	40	0.074809	5
7	40	0.072773	6
8	40	0.109130	3
9	40	0.107330	4
10	60	0.107174	5
11	60	0.106889	6
12	60	0.145766	3
13	60	0.169836	4
14	60	0.146234	5
15	80	0.146342	6
16	80	0.182187	3
17	80	0.178716	4
18	80	0.183863	5
19	80	0.186099	6

In [ ]:

```
sns.barplot(x = 'perc',y = 'time', hue="k-value", data=time_analysis)
plt.xlabel("Percentage of Data")
plt.ylabel("Time Taken")
```

Out[ ]:

```
Text(0, 0.5, 'Time Taken')
```



In [ ]:
```
for k in range(3,7):
    classifier = KNN_classifier(k)
    classifier.fit(X_train,y_train)

    # Predicting the Test set results
    y_test_pred = []
    for i in X_test:
        y_test_pred.append(classifier.predict(i))

    from sklearn.metrics import classification_report
    from sklearn.metrics import precision_recall_fscore_support

    acc = sum([1 if y_test_pred[i] == y_test[i] else 0 for i in range(len(y_test
    print("Accuracy:",acc)

    print(classification_report(y_test,y_test_pred))

    print(precision_recall_fscore_support(y_test,y_test_pred, average='binary'))
```

```
Accuracy: 0.7912087912087912
              precision    recall  f1-score   support

           0       0.81      0.71      0.76        42
           1       0.78      0.86      0.82        49

    accuracy                           0.79        91
   macro avg       0.79      0.79      0.79        91
weighted avg       0.79      0.79      0.79        91

(0.7777777777777778, 0.8571428571428571, 0.8155339805825242, None)
Accuracy: 0.7912087912087912
              precision    recall  f1-score   support

           0       0.76      0.81      0.78        42
           1       0.83      0.78      0.80        49

    accuracy                           0.79        91
   macro avg       0.79      0.79      0.79        91
weighted avg       0.79      0.79      0.79        91

(0.8260869565217391, 0.7755102040816326, 0.8, None)
Accuracy: 0.7912087912087912
              precision    recall  f1-score   support

           0       0.79      0.74      0.77        42
           1       0.79      0.84      0.81        49

    accuracy                           0.79        91
   macro avg       0.79      0.79      0.79        91
weighted avg       0.79      0.79      0.79        91

(0.7884615384615384, 0.8367346938775511, 0.8118811881188118, None)
Accuracy: 0.8021978021978022
              precision    recall  f1-score   support

           0       0.79      0.79      0.79        42
           1       0.82      0.82      0.82        49

    accuracy                           0.80        91
   macro avg       0.80      0.80      0.80        91
weighted avg       0.80      0.80      0.80        91

(0.8163265306122449, 0.8163265306122449, 0.8163265306122449, None)
```

In [ ]: