```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import os
import random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import cv2
import csv
```

```
labels = ['glasses','no_glasses']
img_size = 200
data = []
def get_training_data(data_dir):
  for label in labels:
    path=os.path.join (data_dir, label)
    class_num = labels.index(label)
    print(class_num)
    for img in os.listdir (path):
      try:
        img_arr = cv2.imread(os.path. join (path, img), cv2.COLOR_BAYER_GB2RGB)
        # print(img_arr.shape)
        resized_arr = cv2.resize(img_arr, (img_size, img_size))
        data.append ([resized_arr, class_num])
      except Exception as e:
        print(e)
  return np.array(data)
train = get_training_data('/content/drive/MyDrive/FaceGlassDetection/Images')
print(data)
```

```
         [107, 120, 129]],
         [106, 118, 127]],

        [[182, 167, 175],
         [181, 166, 174],
         [189, 174, 182],
         ...,
         [107, 121, 127],
```

```
for label in labels:
  print(labels.index(label))
```

```
0
1
```

```
x=[]
y=[]
for i,j in data:
  x.append(i)
  y.append(j)
print(y)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
x
```

```
        ...]
```

y

```
         0,
         0,
         0,
         0,
         0,
         0,
         0,
         0,
         0,
         0,
         0,
         0,
         0,
         0,
         0,
         0,
         0,
         ...]
```

```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.25, random_state = 47)
```

```
xtrain
```

```
                 ...,
                 [118, 124, 123],
                 [122, 129, 126],
                 [123, 130, 125]],

                [[100, 108, 108],
                 [ 99, 108, 108],
                 [ 99, 107, 107],
                 ...,
                 [122, 128, 127],
                 [125, 132, 129],
                 [130, 137, 131]],

                ...,

                [[ 89,  90,  80],
                 [ 93,  94,  85],
                 [ 93,  94,  85],
                 ...,
                 [111,  88,  72],
                 [124, 100,  84],
                 [118,  95,  79]],

                [[ 82,  83,  72],
                 [ 82,  83,  72],
                 [ 78,  79,  69],
                 ...,
                 [124, 103,  83],
                 [124, 103,  82],
                 [121,  99,  79]],

                [[ 67,  65,  65],
                 [ 60,  58,  58],
                 [ 60,  58,  57],
                 ...,
                 [130, 109,  87],
                 [125, 103,  82],
                 [121, 100,  78]]], dtype=uint8),
         ...]
```

```
print(np.array(xtrain).shape)
```

```
    (1710, 200, 200, 3)
```

```
print(np.array(ytest).shape)
```

```
(571,)
```

```
x=np.array(x).reshape(2281,120000)
x
```

```
array([[ 85, 161, 160, ...,  41,  60,  64],
       [152, 170, 177, ..., 160, 176, 179],
       [ 24,  67,  82, ..., 182, 192, 196],
       ...,
       [110, 150, 138, ..., 111, 132, 166],
       [ 70,  95, 111, ...,  67,  83,  94],
       [ 79,  96, 109, ..., 105, 119, 125]], dtype=uint8)
```

```
x1=np.array(x).shape
```

```
x1
```

```
(2281, 120000)
```

```
x
```

```
array([[ 85, 161, 160, ...,  41,  60,  64],
       [152, 170, 177, ..., 160, 176, 179],
       [ 24,  67,  82, ..., 182, 192, 196],
       ...,
       [110, 150, 138, ..., 111, 132, 166],
       [ 70,  95, 111, ...,  67,  83,  94],
       [ 79,  96, 109, ..., 105, 119, 125]], dtype=uint8)
```

```
y1=np.array(y).shape
y1
```

```
(2281,)
```

```
d=np.array(xtrain).reshape(1710,120000)
d
```

```
     array([[  5,  18,  34, ...,   6,  16,  43],
            [221, 226, 225, ..., 225, 229, 223],
            [176, 209, 210, ..., 111,  94,  82],
            ...,
            [239, 230, 226, ...,  67,  48,  45],
            [ 94, 102, 115, ...,  28,  29,  51],
            [ 13,  30,  21, ..., 153, 122,  93]], dtype=uint8)
```

```
e=np.array(xtest).reshape(571,120000)
e
```

```
     array([[150, 165, 167, ..., 130, 142, 145],
            [107, 112, 113, ...,  93,  94,  98],
            [149, 164, 166, ..., 144, 168, 216],
            ...,
            [ 51,  88, 101, ...,  85, 103, 120],
            [ 36,  74,  59, ..., 193, 187, 190],
            [ 68, 100, 109, ...,  62,  81,  88]], dtype=uint8)
```

```
print(np.asarray(d.shape))
```

```
     [  1710 120000]
```

```
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
d= sc_x.fit_transform(d)#normalizing
e = sc_x.transform(e)
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(random_state = 0)
```

```
model.fit(d, ytrain)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: 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:
```

```
y_pred=model.predict(e)
y_pred
```

```
array([1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0])
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(ytest,y_pred)
print("Confusion matrix:\n",cm)
```

```
Confusion matrix:
 [[304   8]
 [  1 258]]
```

```
from sklearn.metrics import accuracy_score
print("Accuracy:",accuracy_score(ytest,y_pred))
```

```
Accuracy: 0.9842381786339754
```

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=12)#k value
```

```
knn.fit(d,ytrain)
```

```
▾       KNeighborsClassifier
KNeighborsClassifier(n_neighbors=12)
```

```
y_pred = model.predict(e)
y_pred
```

```
array([1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0])
```

```
print("predicted value for training value",knn.score(d,ytrain))
print("predicted value for testing value",knn.score(e,ytest))
print("Overall Accuracy:",knn.score(sc_x.transform(x),y))
```

```
predicted value for training value 0.8228070175438597
predicted value for testing value 0.7565674255691769
Overall Accuracy: 0.8062253397632617
```

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

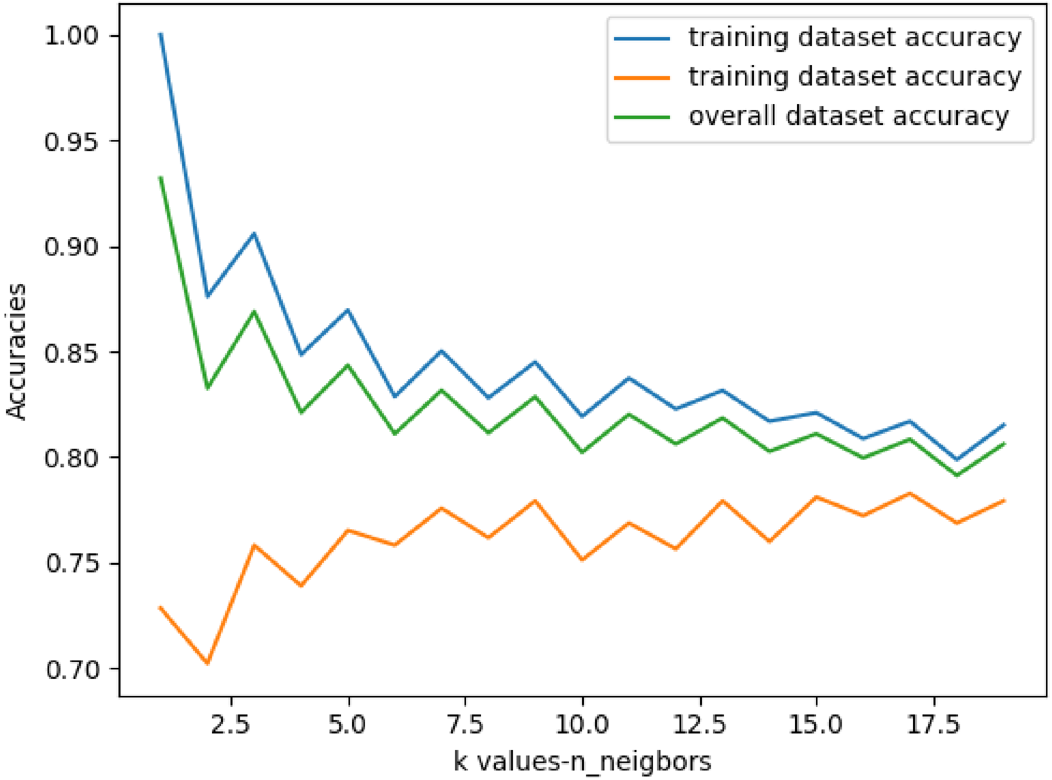
```
y_pred
```

```
array([1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0])
```

```
from sklearn.metrics import confusion_matrix
knns=confusion_matrix(ytest,y_pred)
print("Confusion matrix:\n",knns)
```

```
Confusion matrix:
 [[287  25]
 [114 145]]
```

```
#this code is useful to find best k value using graphs
neighbors=np.arange(1,20)
train_accuracy=np.empty(len(neighbors))
test_accuracy=np.empty(len(neighbors))
overall_accuracy=np.empty(len(neighbors))
#loop over k values
for i,k in enumerate(neighbors):
  knn=KNeighborsClassifier(n_neighbors=k)
  knn.fit(d,ytrain)
  #compute the training and testing accuracy of ML model
  train_accuracy[i]=knn.score(d,ytrain)
  test_accuracy[i]=knn.score(e,ytest)

  #overall score
  overall_accuracy[i]=knn.score(sc_x.transform(x),y)

import matplotlib.pyplot as plt
plt.plot(neighbors,train_accuracy,label="training dataset accuracy")
plt.plot(neighbors,test_accuracy,label="training dataset accuracy")
plt.plot(neighbors,overall_accuracy,label="overall dataset accuracy")
plt.legend()
plt.xlabel('k values-n_neigbors')
plt.ylabel('Accuracies')
plt.show()
```



```
from sklearn import svm
SVM= svm.SVC()
```

```
SVM.fit(d, ytrain)
```

```
▾ SVC
SVC()
```

```
print("Training Accuracy",SVM.score(d,ytrain))
print("Testing Accuracy",SVM.score(e,ytest))
print("Overall Accuracy:",SVM.score(sc_x.transform(x),y))
```

```
Training Accuracy 0.9953216374269006
Testing Accuracy 0.9737302977232924
Overall Accuracy: 0.9899167032003507
```

```
y_pred=SVM.predict(e)
y_pred
```

```
array([1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0])
```

```
from sklearn.metrics import confusion_matrix
SVMS=confusion_matrix(ytest,y_pred)
print("Confusion matrix:\n",SVMS)
```

```
Confusion matrix:
 [[300  12]
 [  3 256]]
```

```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
```

```
dtc.fit(d,ytrain)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
print("Training Accuracy",dtc.score(d,ytrain))
print("Testing Accuracy",dtc.score(e,ytest))
print("Overall Accuracy:",dtc.score(sc_x.transform(x),y))
```

```
Training Accuracy 1.0
Testing Accuracy 0.9316987740805605
Overall Accuracy: 0.9829022358614643
```

```
y_pred=dtc.predict(e)
y_pred
```

```
array([1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0,
       1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0,
       1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0])
```

```
from sklearn.metrics import confusion_matrix
dtcs=confusion_matrix(ytest,y_pred)
print("Confusion matrix:\n",dtcs)
```

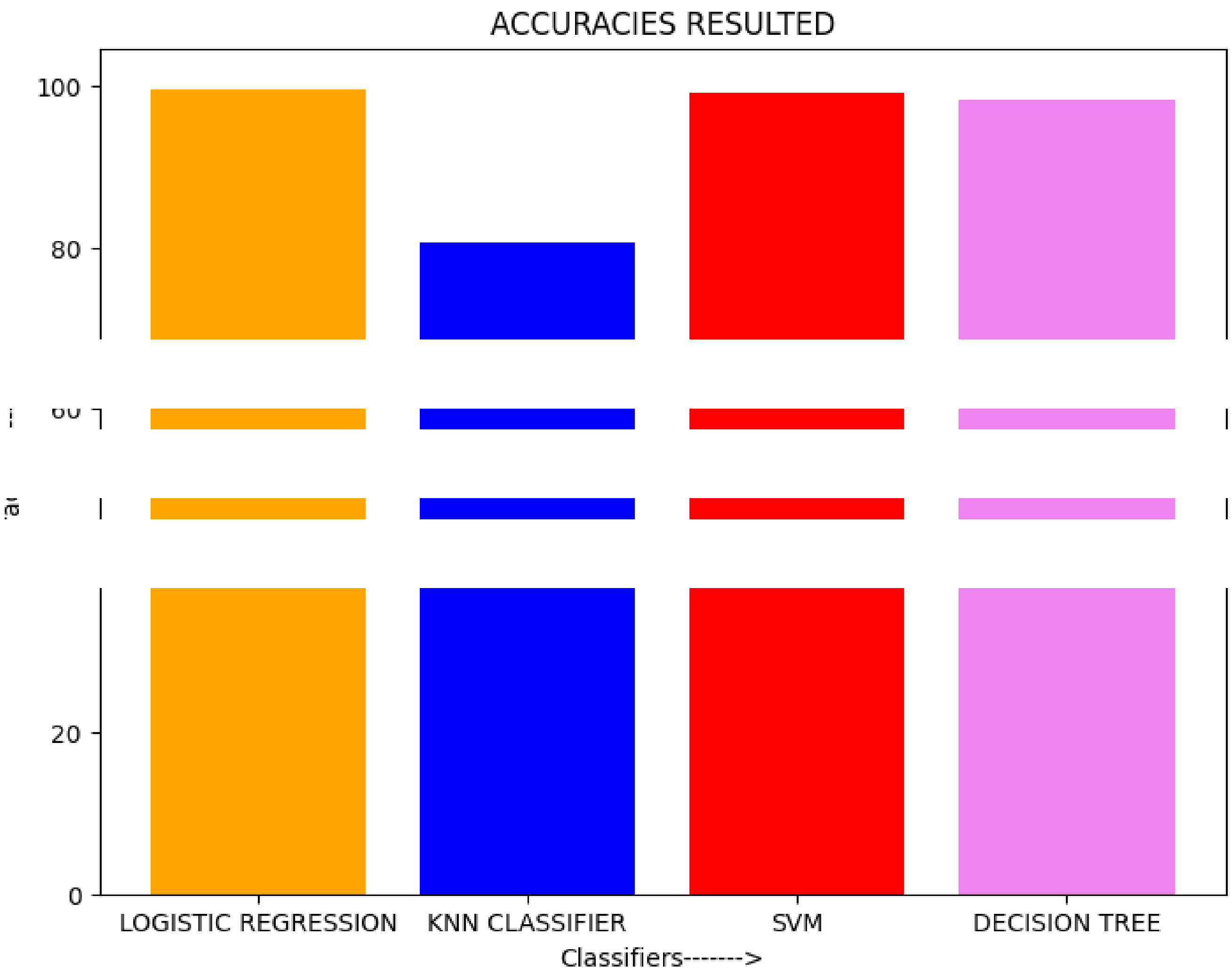
```
Confusion matrix:
 [[298  14]
 [ 25 234]]
```

```
from sklearn.metrics import accuracy_score
accuracy_model = accuracy_score(y,model.predict(sc_x.transform(x)))
print("Logistic regression:",accuracy_model)
accuracy_knn = accuracy_score(y,knn.predict(sc_x.transform(x)))
print("KNN:",accuracy_knn)
accuracy_SVM = accuracy_score(y,SVM.predict(sc_x.transform(x)))
print("Support vector machine:",accuracy_SVM)
accuracy_dtc = accuracy_score(y,dtc.predict(sc_x.transform(x)))
print("Descision tree:",accuracy_dtc)
```

```
Logistic regression: 0.9960543621218764
KNN: 0.8062253397632617
Support vector machine: 0.9899167032003507
Descision tree: 0.9829022358614643
```

```
import matplotlib.pyplot as plt

fig = plt.figure()

ax = fig.add_axes([0,0,1,1])

algo = ['LOGISTIC REGRESSION','KNN CLASSIFIER','SVM','DECISION TREE']
```

```
accuracy = [accuracy_model*100,accuracy_knn*100,accuracy_SVM*100,accuracy_dtc*100]

ax.bar(algo[0],accuracy[0],color = 'orange')

ax.bar(algo[1],accuracy[1],color = 'b')

ax.bar(algo[2],accuracy[2],color = 'r')


ax.bar(algo[3],accuracy[3],color = 'violet')


plt.xlabel('Classifiers------->')

plt.ylabel('Accuracies-------->')

plt.title('ACCURACIES RESULTED')

plt.show()
```



Colab paid products - Cancel contracts here

✓ 0s completed at 2:39 PM