COMPUTER ENGINEERING

CEN481 - INTRODUCTION TO DATA MINING

ALGORITHM : K-Nearest Neighbour

Ali Can SARIBOĞA

2019556055

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, StandardScaler

### DRIVE CONNECTION

# Reading dataset
data = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/dataMining/Acoustic Features.csv')

data.head()
```

OUTPUT:

	Class	_RMSenergy_Mean	_Lowenergy_Mean	_Fluctuation_Mean	_Tempo_Mean	_MFCC_Mean_1	_MFCC_Mean_2	_MFCC_Mean_3	_MFCC_Mean_4	_MFCC_Mean_5	...
0	relax	0.052	0.591	9.136	130.043	3.997	0.363	0.887	0.078	0.221	...
1	relax	0.125	0.439	6.680	142.240	4.058	0.516	0.785	0.397	0.556	...
2	relax	0.046	0.639	10.578	188.154	2.775	0.903	0.502	0.329	0.287	...
3	relax	0.135	0.603	10.442	65.991	2.841	1.552	0.612	0.351	0.011	...
4	relax	0.066	0.591	9.769	88.890	3.217	0.228	0.814	0.096	0.434	...

5 rows × 51 columns

_Chromagram_Mean_9	_Chromagram_Mean_10	_Chromagram_Mean_11	_Chromagram_Mean_12	_HarmonicChangeDetectionFunction_Mean	_HarmonicChangeDetectionFunction_Std
0.426	1.000	0.008	0.101	0.316	0.261
0.002	1.000	0.000	0.984	0.285	0.211
0.184	0.746	0.016	1.000	0.413	0.299
0.038	1.000	0.161	0.757	0.422	0.265
0.004	0.404	1.000	0.001	0.345	0.261

_HarmonicChangeDetectionFunction_Slope	_HarmonicChangeDetectionFunction_PeriodFreq	_HarmonicChangeDetectionFunction_PeriodAmp	_HarmonicChangeDetectionFunction_PeriodEntropy
0.018	1.035	0.593	0.970
-0.082	3.364	0.702	0.967
0.134	1.682	0.692	0.963
0.042	0.354	0.743	0.968
0.089	0.748	0.674	0.957

```
data.info()
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 51 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   Class                                         400 non-null    object
 1   _RMSenergy_Mean                               400 non-null    float64
 2   _Lowenergy_Mean                               400 non-null    float64
 3   _Fluctuation_Mean                             400 non-null    float64
 4   _Tempo_Mean                                   400 non-null    float64
 5   _MFCC_Mean_1                                  400 non-null    float64
 6   _MFCC_Mean_2                                  400 non-null    float64
 7   _MFCC_Mean_3                                  400 non-null    float64
 8   _MFCC_Mean_4                                  400 non-null    float64
 9   _MFCC_Mean_5                                  400 non-null    float64
 10  _MFCC_Mean_6                                  400 non-null    float64
 11  _MFCC_Mean_7                                  400 non-null    float64
 12  _MFCC_Mean_8                                  400 non-null    float64
 13  _MFCC_Mean_9                                  400 non-null    float64
 14  _MFCC_Mean_10                                 400 non-null    float64
 15  _MFCC_Mean_11                                 400 non-null    float64
 16  _MFCC_Mean_12                                 400 non-null    float64
 17  _MFCC_Mean_13                                 400 non-null    float64
 18  _Roughness_Mean                               400 non-null    float64
 19  _Roughness_Slope                              400 non-null    float64
 20  _Zero-crossingrate_Mean                       400 non-null    float64
 21  _AttackTime_Mean                              400 non-null    float64
 22  _AttackTime_Slope                             400 non-null    float64
 23  _Rolloff_Mean                                 400 non-null    float64
 24  _Eventdensity_Mean                            400 non-null    float64
 25  _Pulseclarity_Mean                            400 non-null    float64
 26  _Brightness_Mean                              400 non-null    float64
 27  _Spectralcentroid_Mean                        400 non-null    float64
 28  _Spectralspread_Mean                          400 non-null    float64
 29  _Spectralskewness_Mean                        400 non-null    float64
 30  _Spectralkurtosis_Mean                        400 non-null    float64
 31  _Spectralflatness_Mean                        400 non-null    float64
 32  _EntropyofSpectrum_Mean                       400 non-null    float64
 33  _Chromagram_Mean_1                            400 non-null    float64
 34  _Chromagram_Mean_2                            400 non-null    float64
 35  _Chromagram_Mean_3                            400 non-null    float64
 36  _Chromagram_Mean_4                            400 non-null    float64
 37  _Chromagram_Mean_5                            400 non-null    float64
 38  _Chromagram_Mean_6                            400 non-null    float64
 39  _Chromagram_Mean_7                            400 non-null    float64
 40  _Chromagram_Mean_8                            400 non-null    float64
 41  _Chromagram_Mean_9                            400 non-null    float64
 42  _Chromagram_Mean_10                           400 non-null    float64
 43  _Chromagram_Mean_11                           400 non-null    float64
 44  _Chromagram_Mean_12                           400 non-null    float64
 45  _HarmonicChangeDetectionFunction_Mean         400 non-null    float64
 46  _HarmonicChangeDetectionFunction_Std          400 non-null    float64
 47  _HarmonicChangeDetectionFunction_Slope        400 non-null    float64
 48  _HarmonicChangeDetectionFunction_PeriodFreq   400 non-null    float64
 49  _HarmonicChangeDetectionFunction_PeriodAmp    400 non-null    float64
 50  _HarmonicChangeDetectionFunction_PeriodEntropy  400 non-null  float64
dtypes: float64(50), object(1)
memory usage: 159.5+ KB
```

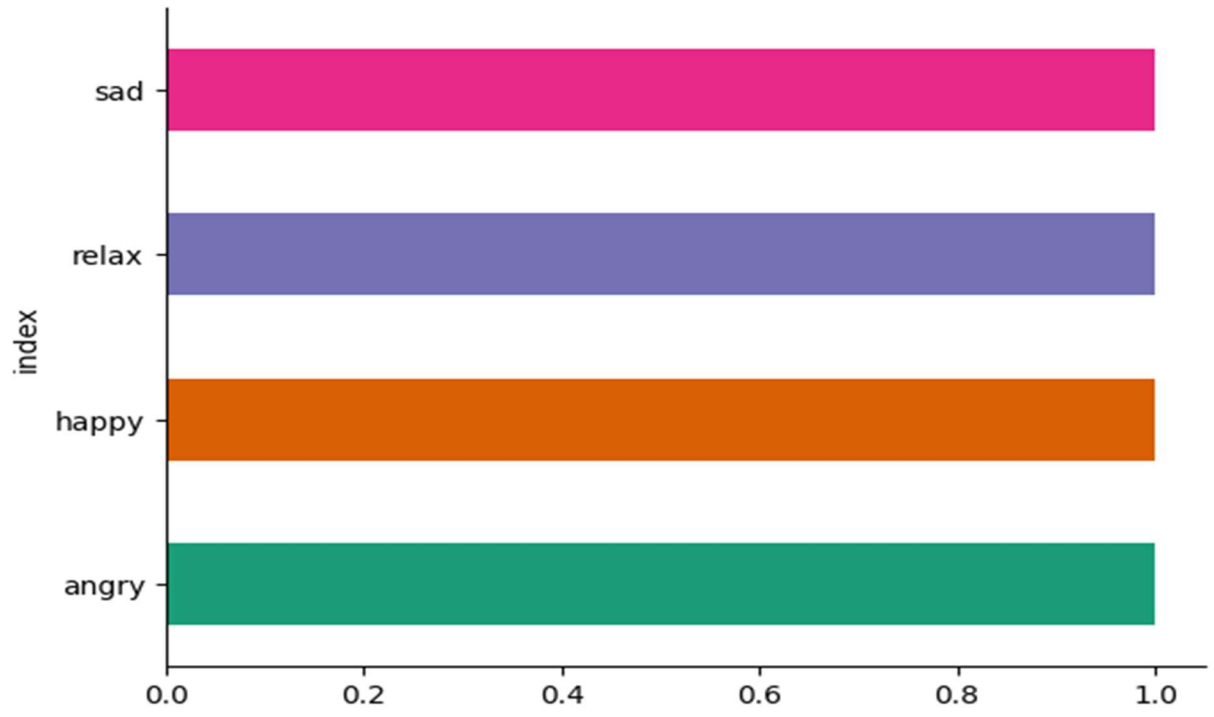
```
# Number of the data in each class
song_types = data["Class"].value_counts()
song_types_df = pd.DataFrame(song_types)
song_types_df = song_types.reset_index(level = 0)
song_types_df
```

OUTPUT:

	index	Class
0	relax	100
1	happy	100
2	sad	100
3	angry	100

```
# Graph - Number of the data in each class
song_types_df.groupby('index').size().plot(kind='barh',
color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```

OUTPUT:




```
# Controlling null value
data.isnull().any().sum()
```

OUTPUT:

0

```
# Distribution percentage of data according to classes
data.Class.value_counts(normalize=True)
```

OUTPUT:

relax 0.25
happy 0.25
sad 0.25
angry 0.25
Name: Class, dtype: float64

```
# About attributes
data.describe().T
```

OUTPUT:

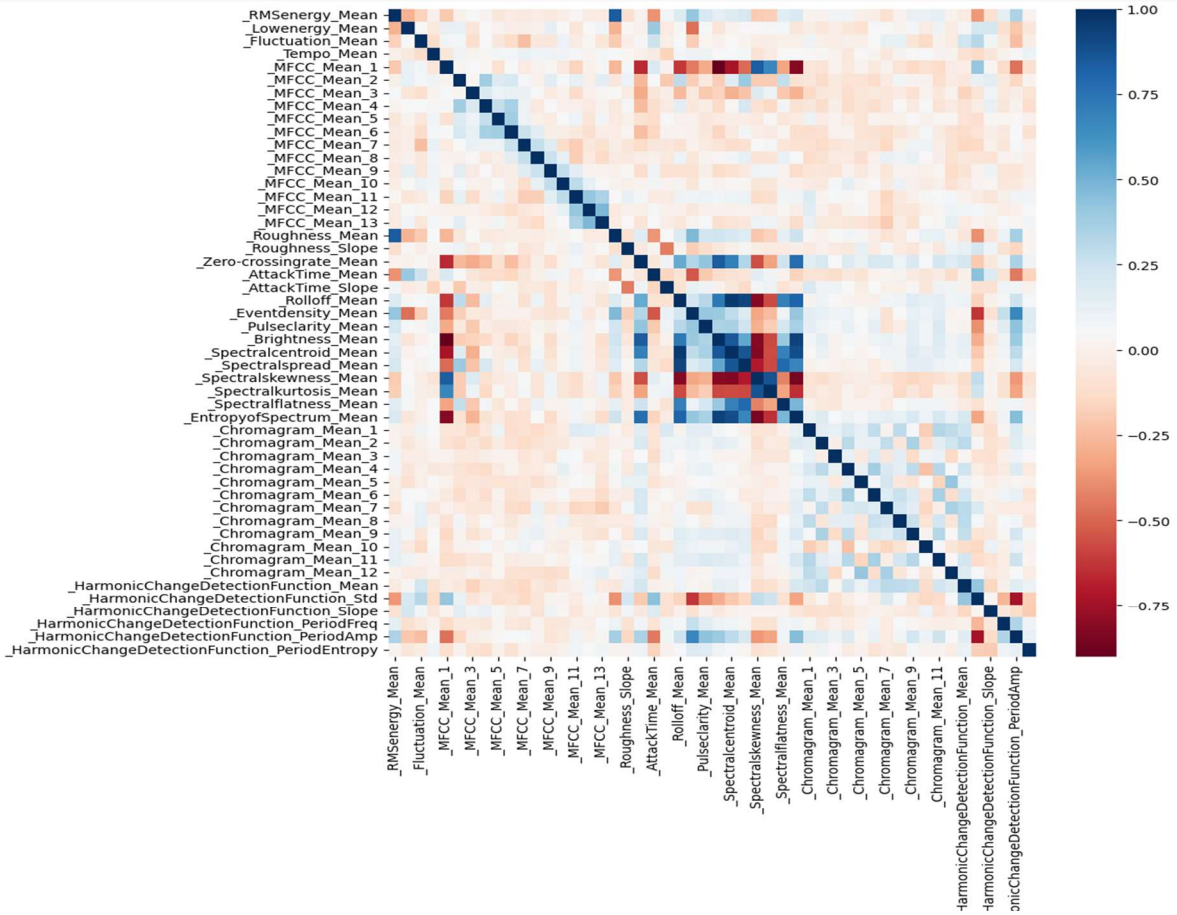
	count	mean	std	min	25%	50%	75%	max
_RMSenergy_Mean	400.0	0.134650	0.064368	0.010	0.08500	0.1280	0.17400	0.431
_Lowenergy_Mean	400.0	0.553605	0.050750	0.302	0.52300	0.5530	0.58325	0.703
_Fluctuation_Mean	400.0	7.145932	2.280145	3.580	5.85950	6.7340	7.82350	23.475
_Tempo_Mean	400.0	123.682020	34.234344	48.284	101.49025	120.1325	148.98625	195.026
_MFCC_Mean_1	400.0	2.456422	0.799262	0.323	1.94850	2.3895	2.86025	5.996
_MFCC_Mean_2	400.0	0.071890	0.537865	-3.484	-0.26275	0.0685	0.41325	1.937
_MFCC_Mean_3	400.0	0.488065	0.294607	-0.870	0.28125	0.4645	0.68600	1.622
_MFCC_Mean_4	400.0	0.030465	0.275839	-1.636	-0.11700	0.0445	0.19825	1.126
_MFCC_Mean_5	400.0	0.178897	0.195230	-0.494	0.06125	0.1810	0.28850	1.055
_MFCC_Mean_6	400.0	0.038307	0.203754	-0.916	-0.07825	0.0495	0.15125	0.799
_MFCC_Mean_7	400.0	0.059943	0.180982	-0.936	-0.04125	0.0720	0.17225	0.571
_MFCC_Mean_8	400.0	0.043467	0.165184	-0.744	-0.04925	0.0395	0.13000	0.728
_MFCC_Mean_9	400.0	0.023010	0.159239	-0.621	-0.07100	0.0165	0.12300	0.539
_MFCC_Mean_10	400.0	0.027793	0.152235	-0.544	-0.05925	0.0315	0.12600	0.510
_MFCC_Mean_11	400.0	0.028798	0.136156	-0.487	-0.04400	0.0370	0.11400	0.494
_MFCC_Mean_12	400.0	0.016667	0.128528	-0.418	-0.05600	0.0225	0.09450	0.355
_MFCC_Mean_13	400.0	0.024118	0.133470	-0.620	-0.04550	0.0390	0.10125	0.536
_Roughness_Mean	400.0	527.681365	521.218943	0.941	169.18875	367.5780	734.37250	3899.847
_Roughness_Slope	400.0	0.072038	0.174301	-0.525	-0.02700	0.0680	0.17400	0.584
_Zero-crossingrate_Mean	400.0	997.252315	524.895867	149.490	592.27500	893.4910	1303.49275	3147.907
_AttackTime_Mean	400.0	0.031305	0.016801	0.010	0.02300	0.0270	0.03300	0.165
_AttackTime_Slope	400.0	-0.002890	0.149920	-0.465	-0.09400	0.0075	0.08900	0.599
_Rolloff_Mean	400.0	5691.069637	2293.401839	887.151	3933.55275	5648.6280	7355.88625	11508.298
_Eventdensity_Mean	400.0	2.784820	1.326889	0.234	1.73700	2.7730	3.69250	7.952
_Pulseclarity_Mean	400.0	0.249387	0.155335	0.011	0.12775	0.2180	0.32725	0.856
_Brightness_Mean	400.0	0.434158	0.131517	0.053	0.35250	0.4480	0.52725	0.737
_Spectralcentroid_Mean	400.0	2581.167267	863.520318	606.524	1981.55775	2547.6780	3182.56975	5326.379
_Spectralspread_Mean	400.0	3082.394695	767.648035	814.817	2506.76850	3150.9490	3684.32525	4721.479
_Spectralskewness_Mean	400.0	1.870035	0.881635	0.390	1.32725	1.6870	2.18250	7.855
_Spectralkurtosis_Mean	400.0	7.348953	8.621386	1.930	3.88150	5.2160	7.84900	121.996
_Spectralflatness_Mean	400.0	0.048523	0.026492	0.006	0.02900	0.0470	0.06200	0.209
_EntropyofSpectrum_Mean	400.0	0.872607	0.037260	0.740	0.85300	0.8790	0.89900	0.942
_Chromagram_Mean_1	400.0	0.352560	0.323071	0.000	0.05700	0.2735	0.55125	1.000
_Chromagram_Mean_2	400.0	0.253035	0.287694	0.000	0.01850	0.1420	0.39525	1.000

_Chromagram_Mean_3	400.0	0.365098	0.324570	0.000	0.07975	0.2885	0.57650	1.000
_Chromagram_Mean_4	400.0	0.208295	0.253623	0.000	0.01700	0.1050	0.31500	1.000
_Chromagram_Mean_5	400.0	0.350412	0.303521	0.000	0.08975	0.2710	0.53575	1.000
_Chromagram_Mean_6	400.0	0.263880	0.292692	0.000	0.01975	0.1440	0.45050	1.000
_Chromagram_Mean_7	400.0	0.242797	0.275796	0.000	0.02600	0.1410	0.36500	1.000
_Chromagram_Mean_8	400.0	0.391873	0.330826	0.000	0.10200	0.2955	0.63550	1.000
_Chromagram_Mean_9	400.0	0.354632	0.334976	0.000	0.06675	0.2470	0.61200	1.000
_Chromagram_Mean_10	400.0	0.590975	0.357981	0.000	0.26450	0.6120	1.00000	1.000
_Chromagram_Mean_11	400.0	0.342340	0.315808	0.000	0.05950	0.2470	0.56525	1.000
_Chromagram_Mean_12	400.0	0.385620	0.348117	0.000	0.06075	0.2965	0.67075	1.000
_HarmonicChangeDetectionFunction_Mean	400.0	0.328213	0.055520	0.112	0.29075	0.3330	0.36725	0.488
_HarmonicChangeDetectionFunction_Std	400.0	0.192997	0.047092	0.060	0.16000	0.1900	0.22600	0.340
_HarmonicChangeDetectionFunction_Slope	400.0	-0.000157	0.104743	-0.285	-0.05800	-0.0020	0.06325	0.442
_HarmonicChangeDetectionFunction_PeriodFreq	400.0	1.762288	0.930352	0.187	0.96100	1.6820	2.24300	4.486
_HarmonicChangeDetectionFunction_PeriodAmp	400.0	0.769690	0.072107	0.530	0.72500	0.7860	0.82400	0.908
_HarmonicChangeDetectionFunction_PeriodEntropy	400.0	0.966712	0.003841	0.939	0.96500	0.9670	0.96900	0.977

```
# Correlation
data_lr_corr = data.select_dtypes('number').corr()

plt.figure(figsize=(10,10))
sns.heatmap(data_lr_corr,cmap='RdBu')
```

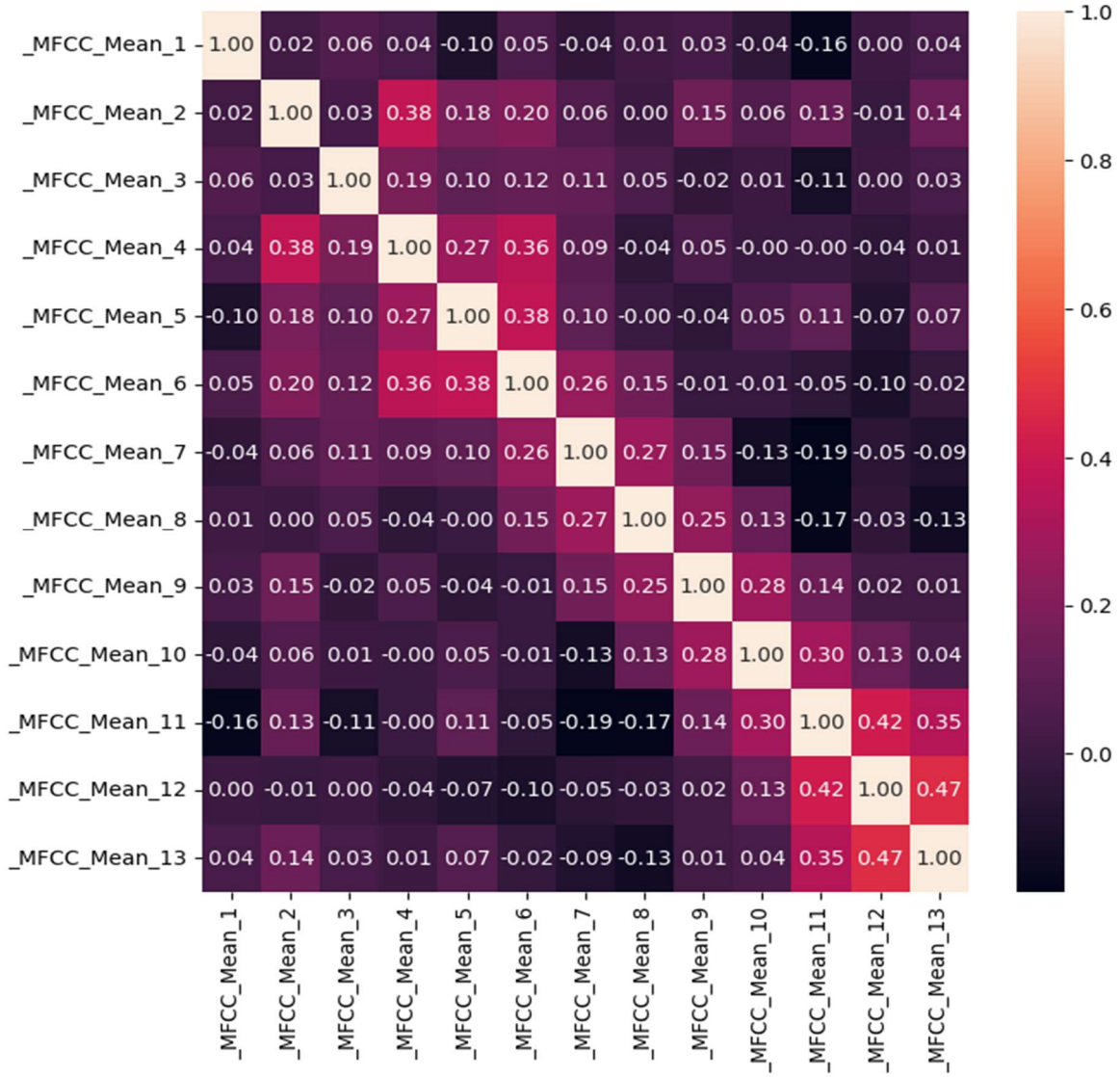
OUTPUT:



```
plt.figure(figsize=(8,8))
sns.heatmap(data.filter(like='MFCC').corr(),annot=True,fmt = '.2f')
```

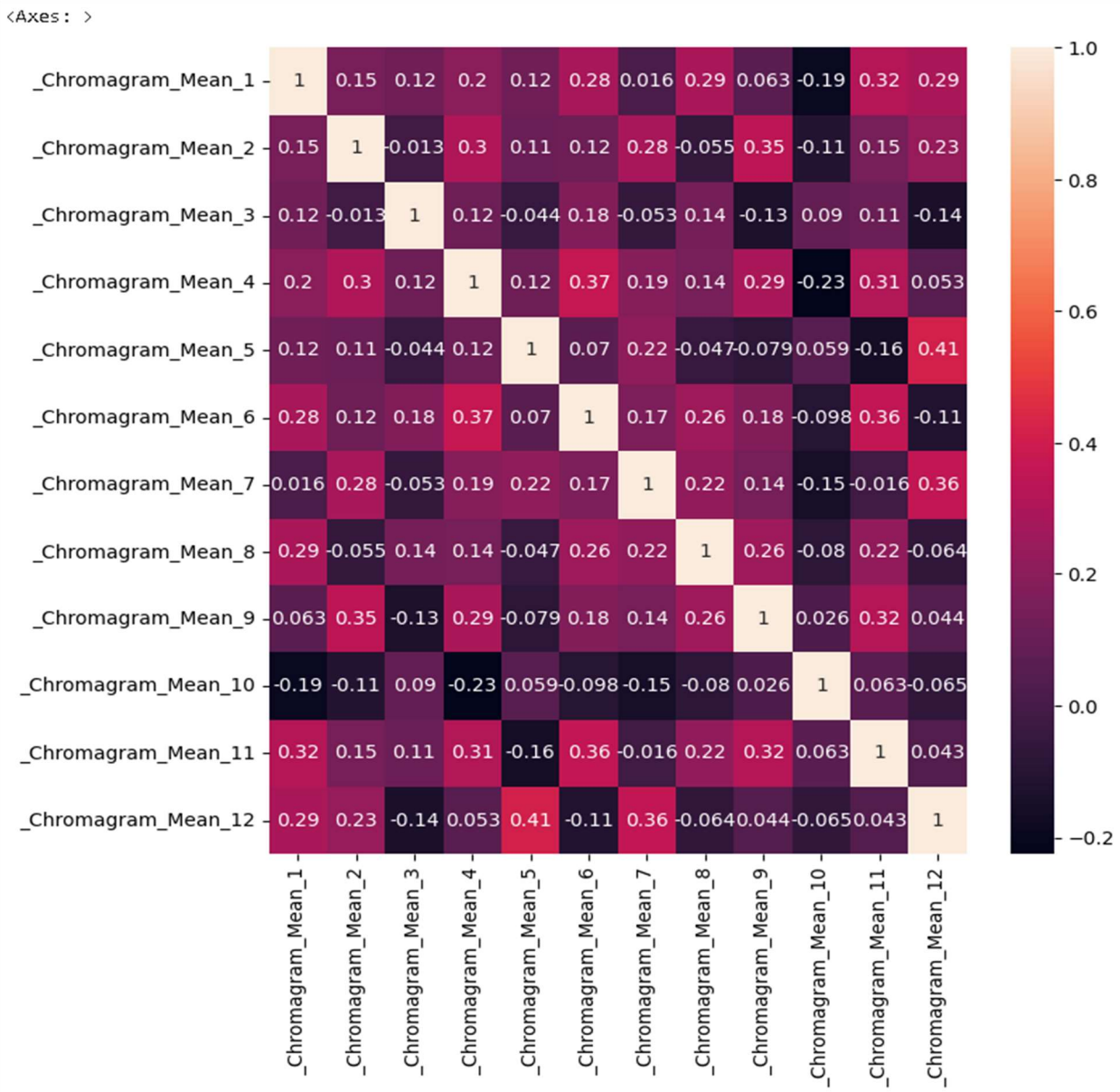
OUTPUT:

```
<Axes: >
```



```
plt.figure(figsize=(8,8))
sns.heatmap(data.filter(like='Chroma').corr(),annot=True)
```

OUTPUT:




```
# Importing knn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, train_test_split,
StratifiedShuffleSplit
from sklearn.metrics import classification_report,
accuracy_score,precision_recall_fscore_support, recall_score,f1_score,
ConfusionMatrixDisplay, confusion_matrix
import time
```

```
# Separating features and labels
X = data.drop('Class', axis = 1)
y = data.loc[:,'Class']

# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    shuffle=True,
                                                    random_state = 20,
                                                    stratify=y)
# Label encoder
le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.fit_transform(y_test)

# Standart scaler
mm = StandardScaler()
X_train = mm.fit_transform(X_train)
X_test = mm.transform(X_test)

def get_metric(model,y_test,y_pred):
    """create a series frame showing the accuracy, precision, f1 and recall
scores"""
    metric ={}
    metric ['accuracy'] = accuracy_score(y_test,y_pred)
    precision,recall,f_score,support =
precision_recall_fscore_support(y_test,y_pred,average='weighted')
    metric['precision'], metric ['recall'] ,metric ['f1']=
precision,recall,f_score
    metric ['train score'] = model.score(X_train,y_train)
    return pd.Series(metric)

# Selecting best k value for the model
score ={}

for n in range(2,50):
    knn_ = KNeighborsClassifier(n).fit(X_train,y_train)
    score[n] = knn_.score(X_test,y_test)
k_ = max(score, key = score.get)
k_
```

OUTPUT:
10

```
# KNN Model
start_time = time.time()
knn = KNeighborsClassifier(n_neighbors=k_)
knn.fit(X_train,y_train)
prediction_duration = time.time() - start_time
```

```
knn.score(X_test,y_test)

# Knn parameters for cross validation
param_grid = {
    'n_neighbors':list(range(2,50)),
    'metric' : ['euclidean','minkowski']
}
```

OUTPUT:
0.7625

```
from sklearn.model_selection import KFold, StratifiedKFold
# Cross Validation
start_time = time.time()
knn_grid = GridSearchCV(KNeighborsClassifier(),
                        param_grid=param_grid,
                        scoring='accuracy',
                        cv=(StratifiedKFold(n_splits = 10,
                                            shuffle = True,
                                            random_state = 20)))
knn_grid.fit(X_train,y_train)
train_pred = knn_grid.predict(X_train)
test_pred = knn_grid.predict(X_test)
cv_duration = time.time() - start_time

knn_grid.score(X_test,y_test)
```

OUTPUT:
0.7125

```
knn_grid.best_params_
```

OUTPUT:
{'metric': 'euclidean', 'n_neighbors': 19}

```
# Prediction
start_time = time.time()
y_pred_cv = knn_grid.predict(X_test)
prediction_duration = time.time() - start_time

def get_metrics2(y_true, y_pred, duration):
    accuracy = accuracy_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred, average='weighted')
    precision, _, f1, _ = precision_recall_fscore_support(y_true, y_pred,
average='weighted')
    return accuracy, recall, precision, f1, duration
```

```
# Measurement of train and test
train_accuracy, train_recall, train_precision, train_f1, _ =
get_metrics2(y_train, train_pred, 0)
test_accuracy, test_recall, test_precision, test_f1, _ = get_metrics2(y_test,
test_pred, prediction_duration)

# Comparison between Knn model and Knn cross validation
print("KNN AND KNN_GRID (Cross Validation)\n")
pd.concat([get_metric(knn,y_test,knn.predict(X_test)),
           get_metric(knn_grid,y_test,knn_grid.predict(X_test))],
          axis = 1).rename(columns={0:'knn',1:'knn_grid'})
```

OUTPUT:

KNN AND KNN_GRID (Cross Validation)

	knn	knn_grid
accuracy	0.762500	0.712500
precision	0.767474	0.714510
recall	0.762500	0.712500
f1	0.755296	0.700123
train score	0.775000	0.771875

```
# Performances
print("PERFORMANCE OF TEST AND TRAIN\n")
result_df = pd.DataFrame({
    'Metric': ['Accuracy', 'Recall', 'Precision', 'F1 Score', 'Prediction
Duration'],
    'Train': [train_accuracy, train_recall, train_precision, train_f1, 0],
    'Test': [test_accuracy, test_recall, test_precision, test_f1,
prediction_duration]
})
print(result_df)
```

OUTPUT:

PERFORMANCE OF TEST AND TRAIN

	Metric	Train	Test
0	Accuracy	0.771875	0.712500
1	Recall	0.771875	0.712500
2	Precision	0.781736	0.714510
3	F1 Score	0.761081	0.700123
4	Prediction Duration	0.000000	0.008998

```
# Results
print("MODEL(knn_grid) RESULTS\n")
print("Best parameters:", knn_grid.best_params_)
print("Average score:", knn_grid.best_score_)
print(f"Training time:
{knn_grid.cv_results_['mean_fit_time'][knn_grid.best_index_]:.4f} second")
print(f"Prediction time: {prediction_duration:.4f} second")
print(f"Cross-validation time:
{knn_grid.cv_results_['mean_score_time'][knn_grid.best_index_]:.4f} second")
```

OUTPUT:

```
MODEL(knn_grid) RESULTS

Best parameters: {'metric': 'euclidean', 'n_neighbors': 19}
Average score: 0.734375
Training time: 0.0007 second
Prediction time: 0.0090 second
Cross-validation time: 0.0043 second
```

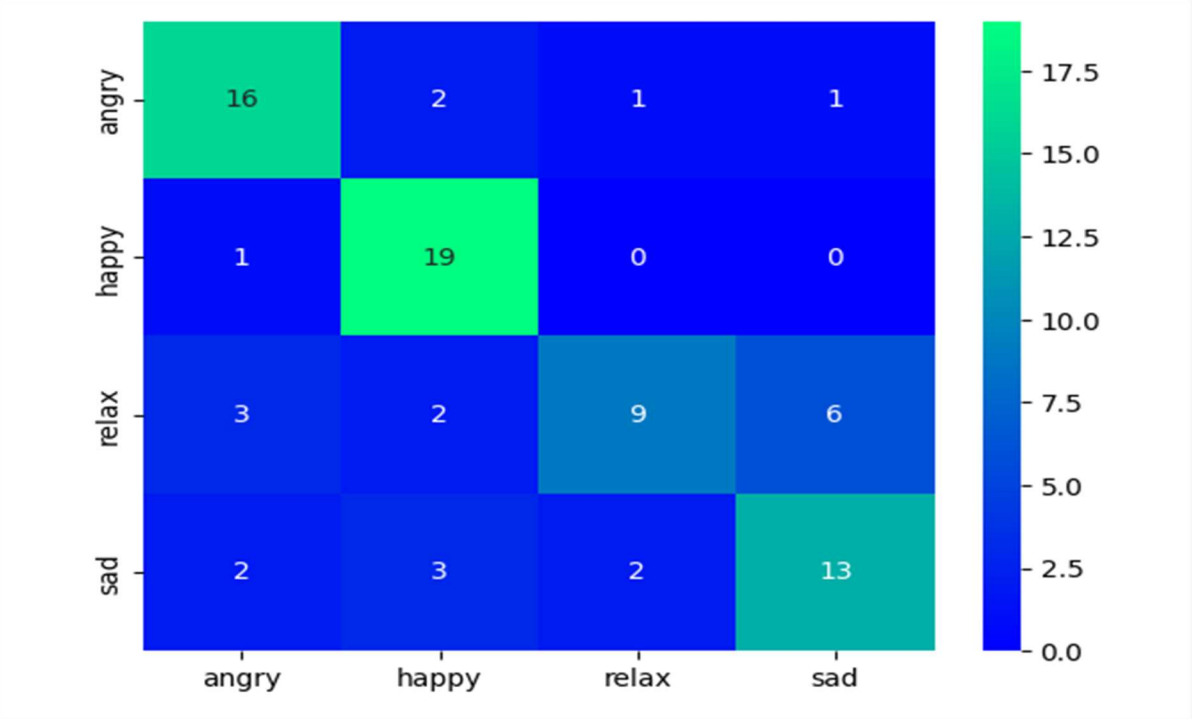
```
conf = confusion_matrix(y_test,y_pred_cv)
print("Confusion Matrix:")
print(conf)
```

OUTPUT:

```
Confusion Matrix:
[[16  2  1  1]
 [ 1 19  0  0]
 [ 3  2  9  6]
 [ 2  3  2 13]]
```

```
# Creating heatmap with test data
ax = sns.heatmap(conf,
                 xticklabels=le.classes_,
                 yticklabels = le.classes_,
                 annot= True,
                 cmap = 'winter')
```

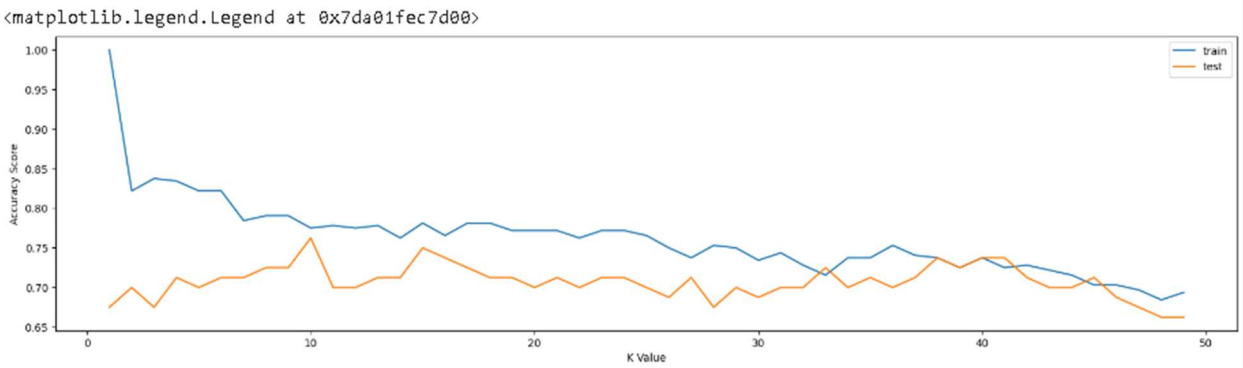
OUTPUT:




```
# Variation of train and test accuracy according to k value
uzunluk = range(1,50)
error1= []
error2= []
for k in uzunluk:
    classifier= KNeighborsClassifier(n_neighbors=k)
    classifier.fit(X_train,y_train)
    error1.append(classifier.score(X_train, y_train))
    error2.append(classifier.score(X_test, y_test))

plt.figure(figsize=(20,5))
plt.plot(uzunluk,error1,label="train")
plt.plot(uzunluk,error2,label="test")
plt.xlabel('K Value')
plt.ylabel('Accuracy Score')
plt.legend()
```

OUTPUT:




```
# Tags and values
labels = ['Validation Accuracy', 'Model Accuracy']
values = [(knn_grid.score(X_test,y_test)), (knn.score(X_test,y_test))]
# Creating bar chart
plt.figure(figsize=(6, 8))
plt.bar(labels, values, color=['blue', 'green'])
# Show values
for i, value in enumerate(values):
    plt.text(i, value + 0.01, f"{value:.4f}", ha='center', va='bottom',
fontsize=12)
# Axes and head
plt.xlabel('Metrics')
plt.ylabel('Accuracy')
plt.title('Validation Accuracy vs Model Accuracy')
plt.show()
```

OUTPUT: