References

1. Dataset:

   https://www.kaggle.com/purumalgi/music-genre-classification?select=test.csv

2. Reference Papers:
   1. Covariance theory & importance :
      1. http://www.netmba.com/statistics/covariance/
   2. PCA theory :
      1. http://www.iro.umontreal.ca/~pift6080/H09/documents/papers/pca_tutorial.pdf
      2. https://builtin.com/data-science/step-step-explanation-principal-component-analysis

Problem Formulation (3 marks)

Objective: To identify different Music genres based on the given features

Dataset details:

- No. of rows: 17,996
- No. of Columns: 16
- No. of Class: 11
- Dataset is published by machinehack.com for one of its hackathons and later posted in kaggle.
- Method of data collection is unknown

Assumptions:

- From the link mentioned for dataset, only "train.csv" was considered for solving
  - "test.csv" was intentionally ignored, since the class-feature isn't present and therefore, true-positives , true-negatives , accuracy and other few terms cannot be calculated.
  - "Submission.csv" - is meant for hackathon submission purposes

Problem Formulation (3 marks)

Assumptions:

- Features – “Track Name” and “Artist Name” were ignored while computing, since they are strings and requires NLP for solving.

  Also, with these features in dataset, the distance between two data points cannot be calculated as all the features-values must be of type – float
- Missing data were filled with mean of rest of the corresponding data

Link to full code mentioned in slides :

https://drive.google.com/drive/folders/165woKCAcqn4amEGqTr50VZwJxJ8gG7rP?usp=sharing

Feature Description (2 marks)

- Artist name – Name of the composer/singer (ignored)
- Track name – Name of the track (ignored)
- Popularity – Reach of the track among track
- Danceability – Peppiness of track
- Energy – Peppiness of track
- Key – Scale in which the track is composed
- Loudness – Amount of sound in track
- Mode – Musical term related with sounds
- Speechiness – Amount of lyrics
- Acousticness – Amount of acoustic instruments used
- Instrumentalness – Amount of total instruments used
- Liveness – Engaging factor in song
- Valence – Musical positiveness conveyed by track
- Tempo – Fastness of the track
- Duration in milliseconds – time duration of track
- Time_signature – No of beats per bar
- Class – One among the genres mentioned

Common in all methods/calculations

```
In [1]: #authors: Agash Uthayasuriyan & Kavvin UV
        #objective: To find optimal k value
        #input: Dataset
        #output: Accuracy
        import pandas as pd #data analysis toolkit
        import matplotlib.pyplot as plt # for plotting graphs
        import numpy as np # for high level computations
        %matplotlib inline
```

```
In [2]: from sklearn.preprocessing import StandardScaler # standardization of values
        from sklearn.preprocessing import MinMaxScaler # Normalization of values
        from sklearn.model_selection import train_test_split # to split data
        from sklearn.neighbors import KNeighborsClassifier #KNN classifier
        from sklearn.metrics import confusion_matrix,accuracy_score # to get confusion matrix and accuracy
        from sklearn.model_selection import cross_val_score # to perform evaluation and cross-validation
```

```
In [3]: data_set = pd.read_csv("dataset.csv") # dataset_input
```

```
In [4]: data_set=data_set.drop(['Track Name','Artist Name'], axis = 1) # dropping of columns as mentioned
        data_set=data_set.fillna(data_set.mean()) # mean for missing data
```

```
In [5]: data_set = np.round(data_set, decimals=2) # rouding all values in dataset to 2 decimal places
        data_set.head() # first 5 values in dataset
```

Out[5]:

	Popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_in min/ms	ti
0	60.0	0.85	0.56	1.0	-4.96	1	0.05	0.02	0.18	0.08	0.90	134.07	234596.0	
1	54.0	0.38	0.81	3.0	-7.23	1	0.04	0.00	0.00	0.10	0.57	116.45	251733.0	
2	35.0	0.43	0.61	6.0	-8.33	1	0.05	0.49	0.00	0.39	0.79	147.68	109667.0	
3	66.0	0.85	0.60	10.0	-6.53	0	0.06	0.02	0.18	0.12	0.57	107.03	173968.0	
4	53.0	0.17	0.98	2.0	-4.28	1	0.22	0.00	0.02	0.17	0.09	199.06	229960.0	

Knn classifier (5 marks)

- Distance metric used for computation is Minkowski distance (default_metric)
- Splitting of dataset into testing and training; one_train,one_test , two_train , two_test with 70% for training and 30% for testing
- Cross validation: Re-sampling procedure used to evaluate a model
  - cv set to 5 (generally considered 5 or 10 based on data)
  - In the first iteration, the first fold is used to test the model and the rest are used to train the model. In the second iteration, 2nd fold is used as the testing set while the rest serve as the training set. This process is repeated until each fold of the 5 folds have been used as the testing set.
- Extra code after running common code is as follows
- Other information is mentioned in comments of code for better understanding

```
In [6]: dset_modified = data_set.drop('Class',axis=1) # dataset without class feature
```

```
In [7]: data_set_feat = pd.DataFrame(dset_modified,columns=data_set.columns[:-1]) # dataset without class feature
```

```
In [8]: data_set_feat = np.round(data_set_feat, decimals=2) # rouding all values to 2 decimal places
```

```
In [9]: one_train, one_test, two_train, two_test = train_test_split(data_set_feat,data_set['Class'],
                                                                test_size=0.30)
        # test_train split with test size =30% and train size =70%
```

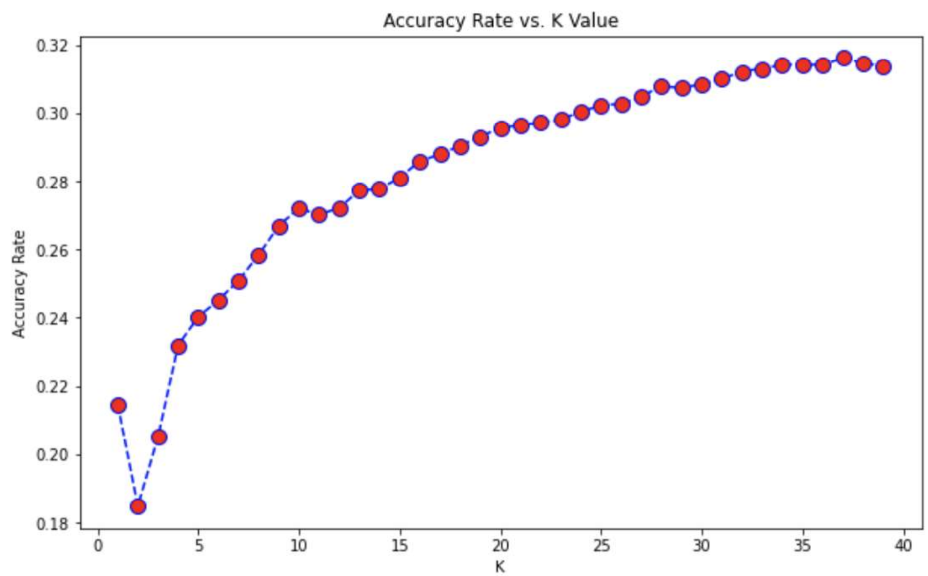
```
In [10]: # Computation of accuracy rates for various neighbour values
         Accurate_rates = []

         for i in range(1,40):

             k_nearest_neighbour = KNeighborsClassifier(n_neighbors=i)
             final_score=cross_val_score(k_nearest_neighbour,data_set_feat,data_set['Class'],cv=5)
             Accurate_rates.append(final_score.mean())
```

```
In [11]: # plot
         plt.figure(figsize=(10,6))

         plt.plot(range(1,40),Accurate_rates,color='blue', linestyle='dashed', marker='o',
                  markerfacecolor='red', markersize=10)
         plt.title('Accuracy Rate vs. K Value')
         plt.xlabel('K')
         plt.ylabel('Accuracy Rate')
```



```
In [12]: max_index = Accurate_rates.index(max(Accurate_rates)) # Best case identifier

k_nearest_neighbour = KNeighborsClassifier(n_neighbors=max_index)

k_nearest_neighbour.fit(one_train,two_train)
prediction = k_nearest_neighbour.predict(one_test)

print('For K=',max_index)
print('Confusion matrix:')
print('\n')
print(confusion_matrix(two_test,prediction))  # Confusion Matrix
print('\n')
print('Accuracy rate: ',round(accuracy_score(two_test,prediction),2)*100,'%')
# Accuracy rate
```

```
For K= 36
Confusion matrix:


[[  99    0    0   34   24    0    0   15    0   12    0]
 [   0    2    3    0    0   15   26    0   13   41  335]
 [   0    0    4    0    0   15   22    0   26   37  287]
 [  32    0    0   53   10    0    0    6    0    2    0]
 [  44    0    0    2   48    0    0    7    0    6    0]
 [   0    7    0    0    0   39   38    0   13   53  275]
 [   0    5    5    0    0   40   67    0   29   72  595]
 [  40    0    0   13    1    0    0  130    0    1    0]
 [   0    2    1    0    0    9   46    0   52   21  414]
 [  40    1    3    9   16   36   56    1   14  118  474]
 [  14    7    3    5   26   45  117    1   67  112 1046]]


Accuracy rate:  31.0 %
```

- Therefore, for the given data the maximum accuracy using K-nearest neighbors method was found as 31% (BEST CASE) for k= 36 neighbors
- The corresponding confusion matrix has been printed.

Knn classifier (5 marks)

- For a different K value :
  - Same accuracy rate has been obtained as in graph

```
In [13]: t = 30 # Random K value

         k_nearest_neighbour = KNeighborsClassifier(n_neighbors=t)

         k_nearest_neighbour.fit(one_train,two_train)
         prediction = k_nearest_neighbour.predict(one_test)

         print('For K=',t)
         print('Confusion matrix:')
         print('\n')
         print(confusion_matrix(two_test,prediction))  # Confusion Matrix
         print('\n')
         print('Accuracy rate: ',round(accuracy_score(two_test,prediction),
         # Accuracy rate
```

```
For K= 30
Confusion matrix:


[[106   0   0  35  22   0   0  12   0   9   0]
 [  0   6   2   0   0  14  42   0  18  42 311]
 [  0   4   7   0   0   8  29   0  30  38 275]
 [ 30   0   0  56   8   0   0   6   0   3   0]
 [ 44   0   0   1  48   0   0   7   0   7   0]
 [  0   6   1   0   0  34  57   0  14  57 256]
 [  0   8   5   0   0  32  79   0  38  89 562]
 [ 38   0   0  12   1   0   0 133   0   1   0]
 [  0   2   5   0   0   6  52   0  49  28 403]
 [ 38   9   5   9  16  32  73   1  16 141 428]
 [ 16  11   8   6  26  54 155   0  84 138 945]]


Accuracy rate:  30.0 %
```

- Inference:
  - Minkowski Distance uses both Manhattan and Euclidean distance in a generalized form for calculation

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

  - For various values of K the accuracy rates changes and through plotting all the values, the best case was found
  - In addition, the accuracy rates for other K values can be inferred from graph
  - Confusion matrix which formulates predicted vs actual values, Sensitivity & specificity was found to be low which has in-turn resulted in less accuracy rate

Normalization (5 marks)

Extra code after running common code:

```
In [5]: scaled = MinMaxScaler() #function MinMax scaler for normalising values

In [6]: scaled.fit(data_set.drop('Class',axis=1)) # dropping class-feature

Out[6]: MinMaxScaler()

In [7]: dset_modified = scaled.transform(data_set.drop('Class',axis=1)) #dropping class-feature

In [8]: data_set_feat = pd.DataFrame(dset_modified,columns=data_set.columns[:-1]) #dropping class-feature

In [9]: data_set_feat = np.round(data_set_feat, decimals=2) #rounding all values to 2 decimals
        data_set_feat.head() #dataset_after_normalization

Out[9]:
```

	Popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_in min/ms	time_signature
0	0.60	0.85	0.56	0.0	0.85	1.0	0.03	0.02	0.18	0.07	0.91	0.55	0.16	0.75
1	0.54	0.35	0.81	0.2	0.79	1.0	0.02	0.00	0.00	0.09	0.57	0.46	0.17	0.75
2	0.34	0.40	0.61	0.5	0.77	1.0	0.03	0.49	0.00	0.39	0.79	0.63	0.07	0.75
3	0.66	0.85	0.60	0.9	0.81	0.0	0.04	0.02	0.18	0.11	0.57	0.41	0.12	0.75
4	0.53	0.12	0.97	0.1	0.86	1.0	0.21	0.00	0.02	0.16	0.08	0.90	0.16	0.75

```
In [10]: one_train, one_test, two_train, two_test = train_test_split(data_set_feat,data_set['Class'],
                                                                     test_size=0.30)
```

```
In [11]: Accurate_rates = []

         for i in range(1,40):

             k_nearest_neighbour = KNeighborsClassifier(n_neighbors=i)
             final_score=cross_val_score(k_nearest_neighbour,data_set_feat,data_set['Class'],cv=5)
             Accurate_rates.append(final_score.mean())
```

```
In [12]: plt.figure(figsize=(10,6))

         plt.plot(range(1,40),Accurate_rates,color='blue', linestyle='dashed', marker='o',
                  markerfacecolor='red', markersize=10)
         plt.title('Accuracy Rate vs. K Value')
         plt.xlabel('K')
         plt.ylabel('Accuracy Rate')

Out[12]: Text(0, 0.5, 'Accuracy Rate')
```