Genre Classification Using KNN

November 29, 2021

1 CSCI 166- Genre Classification

by Anthony Marin and Adrian Romero

Introduction: Our goal is to utilize supervised ML and K-Nearest Neighbor algorithm to help us sort music by their respective genre. Challenges we expect to face include sorting songs between similar genres and determining which data provides information and which data acts as noise.

1.1 Import Data

[113]:
```python
from sklearn.neighbors import NearestNeighbors
import pandas as pnda
import plotly.express as pltx
import matplotlib.pyplot as ppt
%matplotlib inline
import numpy as npy
import seaborn as sbn
from sklearn.preprocessing import StandardScaler


alt = pnda.read_csv("dataSets/alternative_music_data.csv")
blues = pnda.read_csv("dataSets/blues_music_data.csv")
hiphop = pnda.read_csv("dataSets/hiphop_music_data.csv")
indie = pnda.read_csv("dataSets/indie_alt_music_data.csv")
metal = pnda.read_csv("dataSets/metal_music_data.csv")
pop = pnda.read_csv("dataSets/pop_music_data.csv")
rock = pnda.read_csv("dataSets/rock_music_data.csv")
```

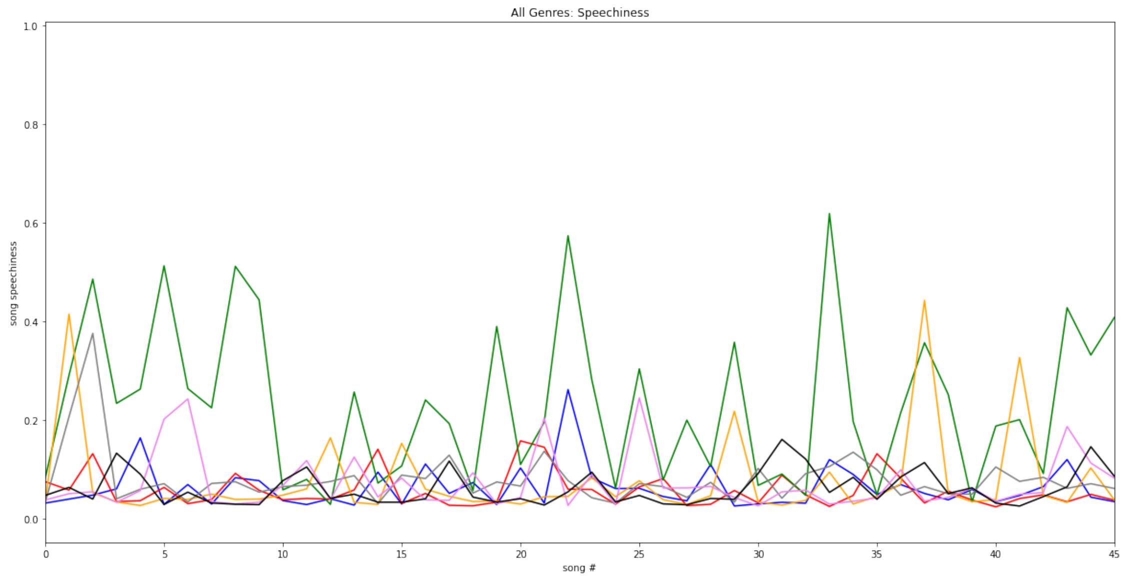
2 Genre Tags:

1. Alternative
2. Blues
3. Hiphop
4. Indie
5. Metal
6. Pop
7. Rock

3 Visualizing Our Data

3.1 Speechiness

[114]:
```
ppt.figure(figsize=(20,10))

aS = npy.array(alt.speechiness)
bS = npy.array(blues.speechiness)
hhS = npy.array(hiphop.speechiness)
iS = npy.array(indie.speechiness)
mS = npy.array(metal.speechiness)
pS = npy.array(pop.speechiness)
rS = npy.array(rock.speechiness)

songRange = list(range(1, 1000))

ppt.xlim([0, 45])


ppt.plot(list(range(0,len(aS))),aS,color='blue')
ppt.plot(list(range(0,len(bS))),bS, color='red')
ppt.plot(list(range(0,len(hhS))),hhS, color='green')
ppt.plot(list(range(0,len(iS))),iS, color='orange')
ppt.plot(list(range(0,len(mS))),mS, color='gray')
ppt.plot(list(range(0,len(pS))),pS, color='violet')
ppt.plot(list(range(0,len(rS))),rS, color='black')


ppt.xlabel("song #")
ppt.ylabel("song speechiness")
ppt.title("All Genres: Speechiness")

ppt.show()
```

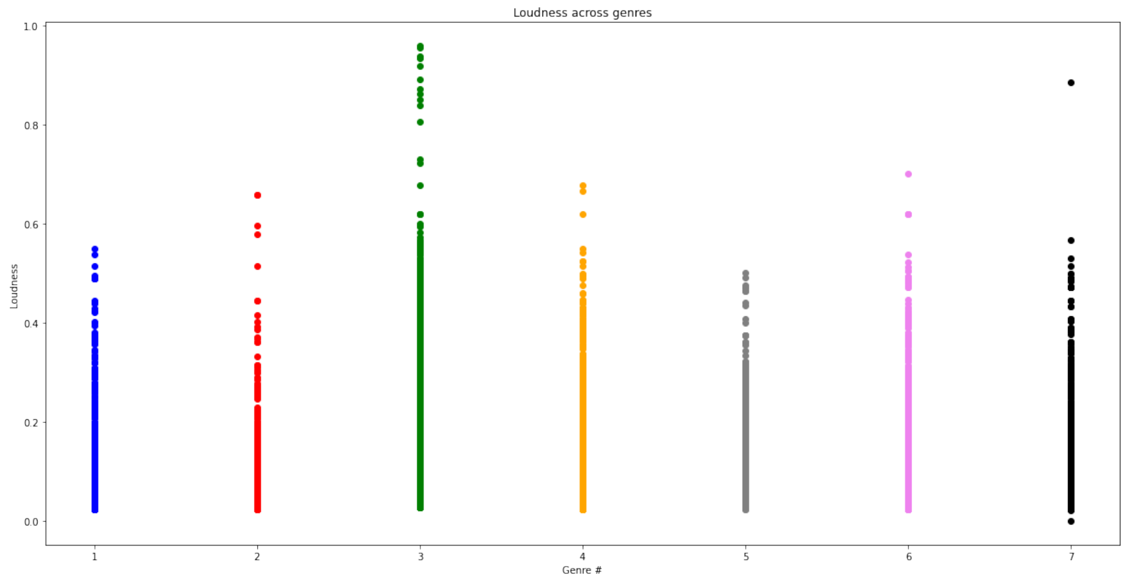


While we can see that hiphop typically rates higher in speechiness, deciphering any patterns in the lower values of speechiness because very difficult.

Let's try to visualize the data little differently.

[115]:
```
ppt.figure(figsize=(20,10))
ppt.scatter(alt.genre_tag,alt.speechiness, color = 'blue')
ppt.scatter(blues.genre_tag,blues.speechiness, color = 'red')
ppt.scatter(hiphop.genre_tag,hiphop.speechiness, color = 'green')
ppt.scatter(indie.genre_tag,indie.speechiness, color = 'orange')
ppt.scatter(metal.genre_tag,metal.speechiness, color = 'gray')
ppt.scatter(pop.genre_tag,pop.speechiness, color = 'violet')
ppt.scatter(rock.genre_tag,rock.speechiness, color = 'black')
ppt.xlabel("Genre #")
ppt.ylabel("Loudness")
ppt.title("Loudness across genres")
```

[115]:
```
Text(0.5, 1.0, 'Loudness across genres')
```

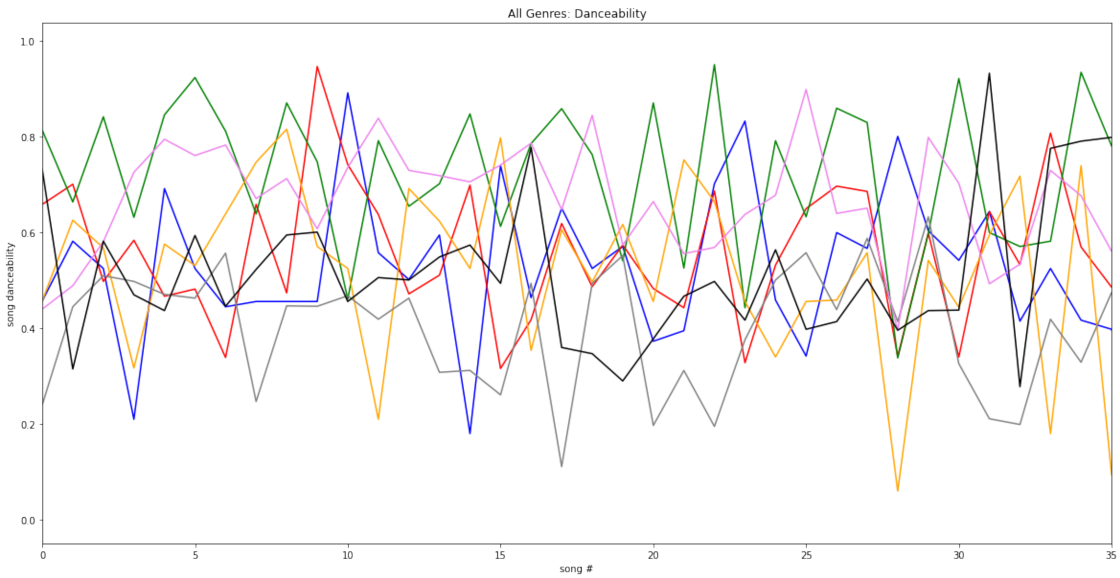


Not much information is gained, but we can confirm that hiphop has high levels of speechiness.

3.2 Danceability

[116]:
```
ppt.figure(figsize=(20,10))

aD = npy.array(alt.danceability)
bD = npy.array(blues.danceability)
hhD = npy.array(hiphop.danceability)
iD = npy.array(indie.danceability)
mD = npy.array(metal.danceability)
pD = npy.array(pop.danceability)
rD = npy.array(rock.danceability)

songRange = list(range(1, 1000))

ppt.xlim([0, 35])


ppt.plot(list(range(0,len(aD))),aD,color='blue')
ppt.plot(list(range(0,len(bD))),bD, color='red')
ppt.plot(list(range(0,len(hhD))),hhD, color='green')
ppt.plot(list(range(0,len(iD))),iD, color='orange')
ppt.plot(list(range(0,len(mD))),mD, color='gray')
ppt.plot(list(range(0,len(pD))),pD, color='violet')
ppt.plot(list(range(0,len(rD))),rD, color='black')
```

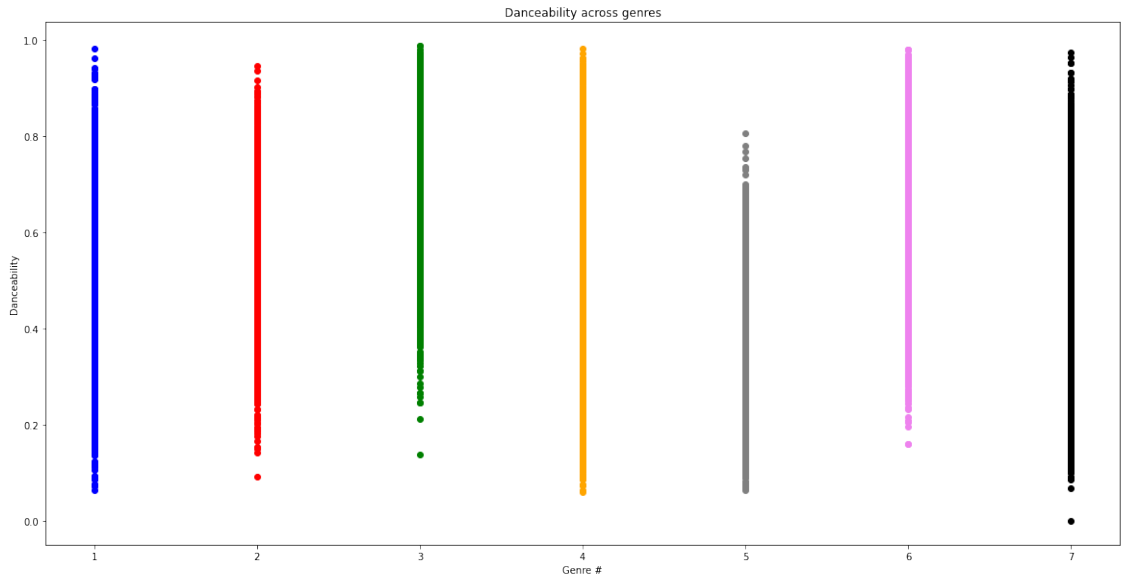
```python
ppt.xlabel("song #")
ppt.ylabel("song danceability")
ppt.title("All Genres: Danceability")

ppt.show()
```



Again, this data isn't very useable. Let's just stick to scatter graphs and see if we can any usefel genre correlations.

[117]:
```python
ppt.figure(figsize=(20,10))
ppt.scatter(alt.genre_tag,alt.danceability, color = 'blue')
ppt.scatter(blues.genre_tag,blues.danceability, color = 'red')
ppt.scatter(hiphop.genre_tag,hiphop.danceability, color = 'green')
ppt.scatter(indie.genre_tag,indie.danceability, color = 'orange')
ppt.scatter(metal.genre_tag,metal.danceability, color = 'gray')
ppt.scatter(pop.genre_tag,pop.danceability, color = 'violet')
ppt.scatter(rock.genre_tag,rock.danceability, color = 'black')
ppt.xlabel("Genre #")
ppt.ylabel("Danceability")
ppt.title("Danceability across genres")
```

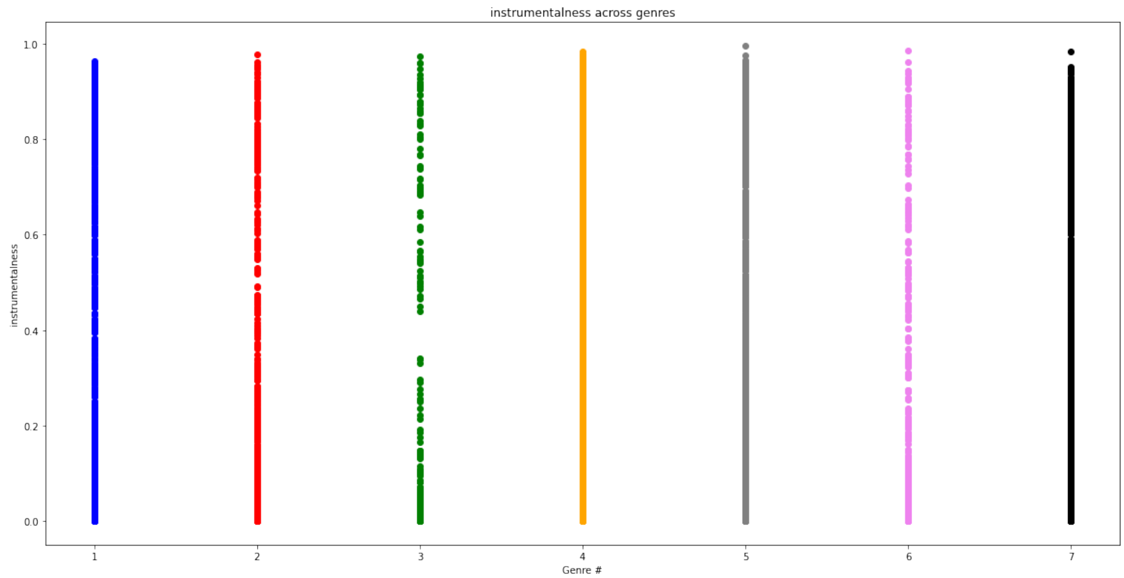
[117]:
```
Text(0.5, 1.0, 'Danceability across genres')
```



3.3 Instrumentalness

[118]:
```
ppt.figure(figsize=(20,10))
ppt.scatter(alt.genre_tag,alt.instrumentalness, color = 'blue')
ppt.scatter(blues.genre_tag,blues.instrumentalness, color = 'red')
ppt.scatter(hiphop.genre_tag,hiphop.instrumentalness, color = 'green')
ppt.scatter(indie.genre_tag,indie.instrumentalness, color = 'orange')
ppt.scatter(metal.genre_tag,metal.instrumentalness, color = 'gray')
ppt.scatter(pop.genre_tag,pop.instrumentalness, color = 'violet')
ppt.scatter(rock.genre_tag,rock.instrumentalness, color = 'black')
ppt.xlabel("Genre #")
ppt.ylabel("instrumentalness")
ppt.title("instrumentalness across genres")
```

[118]:
```
Text(0.5, 1.0, 'instrumentalness across genres')
```

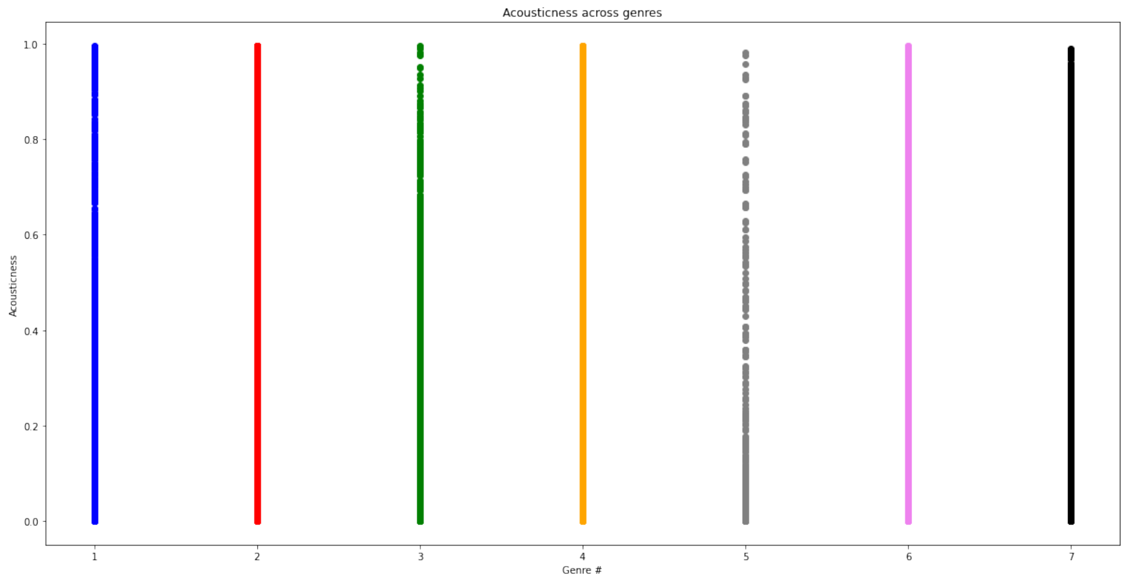


No clear patterns seen here! We can throw out this column!

3.4 Acousticness

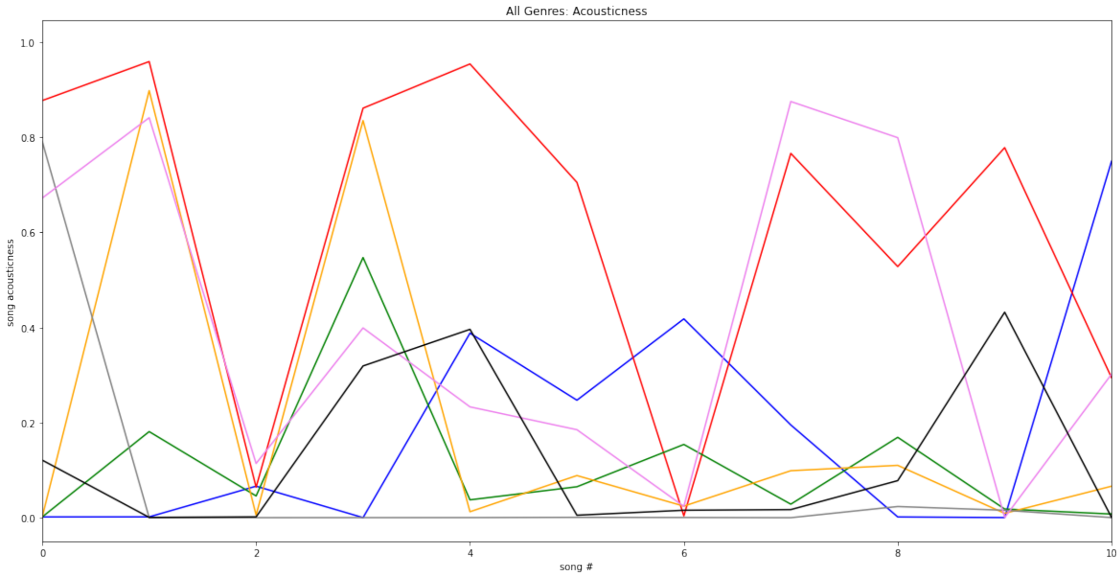
[119]:
```
ppt.figure(figsize=(20,10))
ppt.scatter(alt.genre_tag,alt.acousticness, color = 'blue')
ppt.scatter(blues.genre_tag,blues.acousticness, color = 'red')
ppt.scatter(hiphop.genre_tag,hiphop.acousticness, color = 'green')
ppt.scatter(indie.genre_tag,indie.acousticness, color = 'orange')
ppt.scatter(metal.genre_tag,metal.acousticness, color = 'gray')
ppt.scatter(pop.genre_tag,pop.acousticness, color = 'violet')
ppt.scatter(rock.genre_tag,rock.acousticness, color = 'black')
ppt.xlabel("Genre #")
ppt.ylabel("Acousticness")
ppt.title("Acousticness across genres")
```

[119]:
```
Text(0.5, 1.0, 'Acousticness across genres')
```



[120]:
```
ppt.figure(figsize=(20,10))

aA = npy.array(alt.acousticness)
bA = npy.array(blues.acousticness)
hhA = npy.array(hiphop.acousticness)
iA = npy.array(indie.acousticness)
mA = npy.array(metal.acousticness)
pA = npy.array(pop.acousticness)
rA = npy.array(rock.acousticness)

songRange = list(range(1, 1000))

ppt.xlim([0, 10])


ppt.plot(list(range(0,len(aA))),aA,color='blue')
ppt.plot(list(range(0,len(bA))),bA, color='red')
ppt.plot(list(range(0,len(hhA))),hhA, color='green')
ppt.plot(list(range(0,len(iA))),iA, color='orange')
ppt.plot(list(range(0,len(mA))),mA, color='gray')
ppt.plot(list(range(0,len(pA))),pA, color='violet')
ppt.plot(list(range(0,len(rA))),rA, color='black')


ppt.xlabel("song #")
ppt.ylabel("song acousticness")
ppt.title("All Genres: Acousticness")
```

```
ppt.show()
```



In our line graph, blues and pop seem to have higher levels of acousticness.

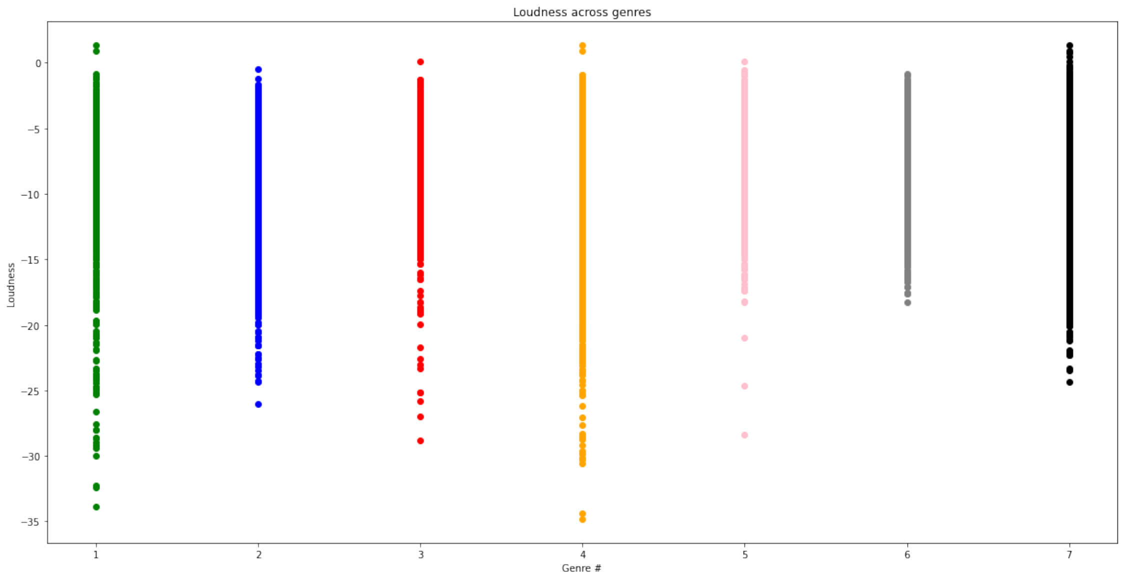
3.5 Loudness

[121]:

```
ppt.figure(figsize=(20,10))
ppt.scatter(alt.genre_tag,alt.loudness, color = 'green')
ppt.scatter(blues.genre_tag,blues.loudness, color = 'blue')
ppt.scatter(hiphop.genre_tag,hiphop.loudness, color = 'red')
ppt.scatter(indie.genre_tag,indie.loudness, color = 'orange')
ppt.scatter(metal.genre_tag,metal.loudness, color = 'pink')
ppt.scatter(pop.genre_tag,pop.loudness, color = 'gray')
ppt.scatter(rock.genre_tag,rock.loudness, color = 'black')
ppt.xlabel("Genre #")
ppt.ylabel("Loudness")
ppt.title("Loudness across genres")
```

[121]:

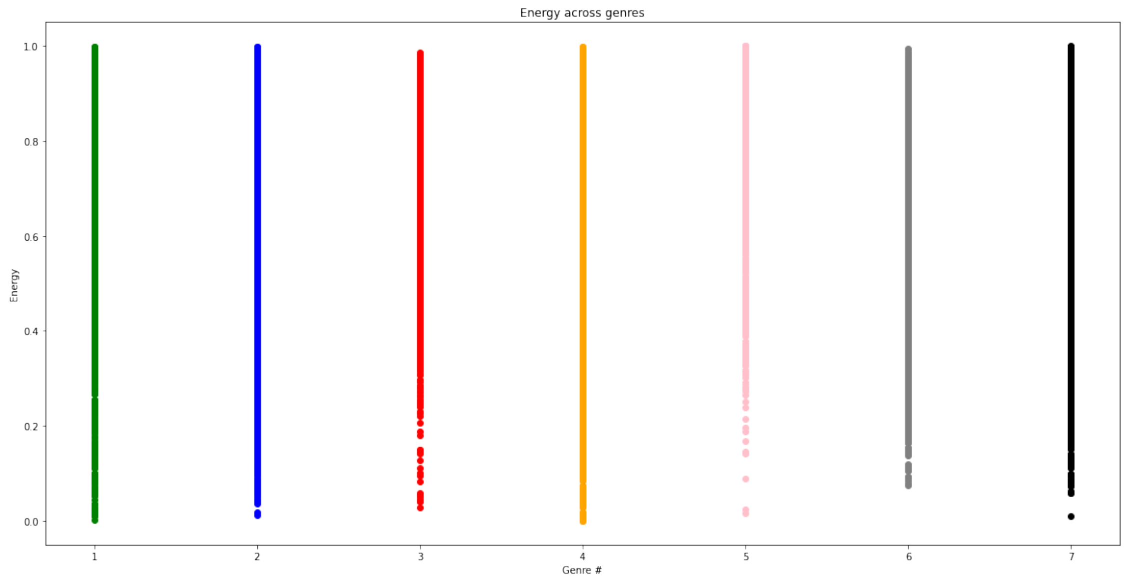
```
Text(0.5, 1.0, 'Loudness across genres')
```



3.6 Energy

[122]:
```
ppt.figure(figsize=(20,10))
ppt.scatter(alt.genre_tag,alt.energy, color = 'green')
ppt.scatter(blues.genre_tag,blues.energy, color = 'blue')
ppt.scatter(hiphop.genre_tag,hiphop.energy, color = 'red')
ppt.scatter(indie.genre_tag,indie.energy, color = 'orange')
ppt.scatter(metal.genre_tag,metal.energy, color = 'pink')
ppt.scatter(pop.genre_tag,pop.energy, color = 'gray')
ppt.scatter(rock.genre_tag,rock.energy, color = 'black')
ppt.xlabel("Genre #")
ppt.ylabel("Energy")
ppt.title("Energy across genres")
```

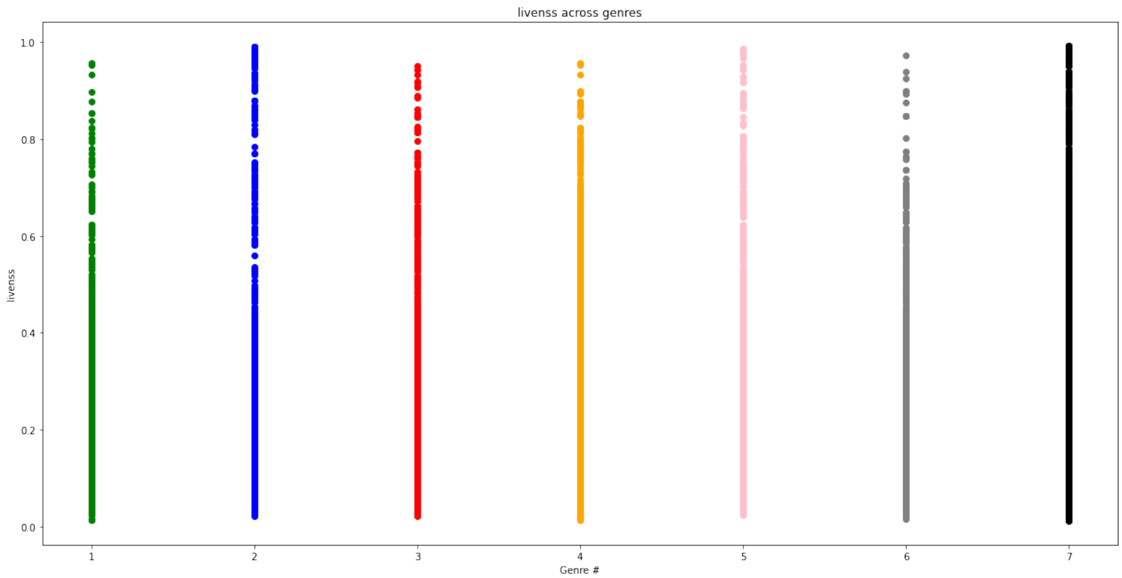
[122]:
```
Text(0.5, 1.0, 'Energy across genres')
```



3.7 Liveness

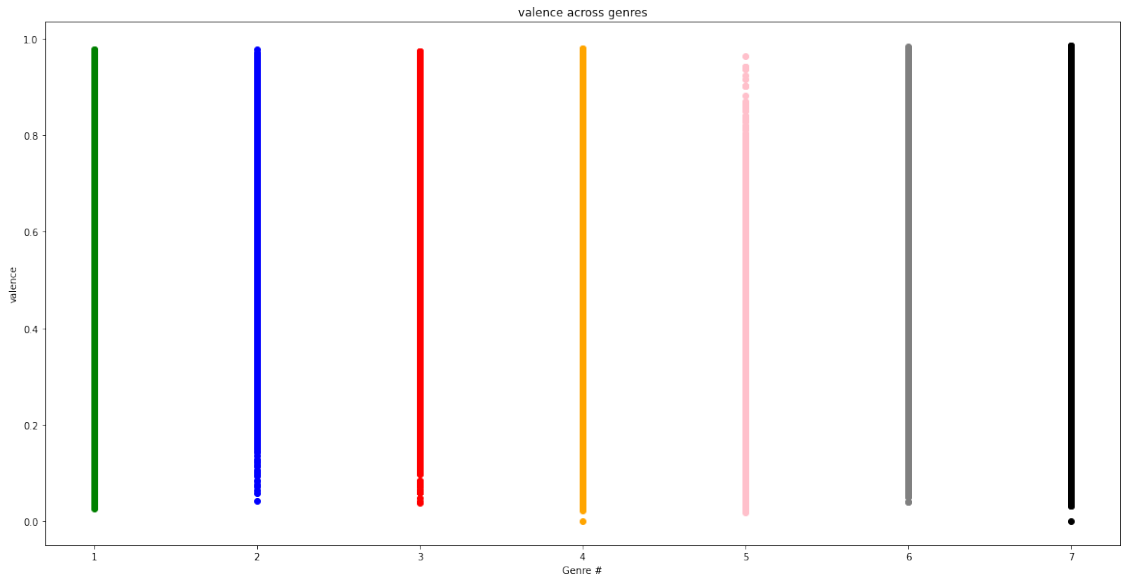
[123]:
```
ppt.figure(figsize=(20,10))
ppt.scatter(alt.genre_tag,alt.liveness, color = 'green')
ppt.scatter(blues.genre_tag,blues.liveness, color = 'blue')
ppt.scatter(hiphop.genre_tag,hiphop.liveness, color = 'red')
ppt.scatter(indie.genre_tag,indie.liveness, color = 'orange')
ppt.scatter(metal.genre_tag,metal.liveness, color = 'pink')
ppt.scatter(pop.genre_tag,pop.liveness, color = 'gray')
ppt.scatter(rock.genre_tag,rock.liveness, color = 'black')
ppt.xlabel("Genre #")
ppt.ylabel("livenss")
ppt.title("livenss across genres")
```

[123]:
```
Text(0.5, 1.0, 'livenss across genres')
```



[124]:
```
ppt.figure(figsize=(20,10))
ppt.scatter(alt.genre_tag,alt.valence, color = 'green')
ppt.scatter(blues.genre_tag,blues.valence, color = 'blue')
ppt.scatter(hiphop.genre_tag,hiphop.valence, color = 'red')
ppt.scatter(indie.genre_tag,indie.valence, color = 'orange')
ppt.scatter(metal.genre_tag,metal.valence, color = 'pink')
ppt.scatter(pop.genre_tag,pop.valence, color = 'gray')
ppt.scatter(rock.genre_tag,rock.valence, color = 'black')
ppt.xlabel("Genre #")
ppt.ylabel("valence")
ppt.title("valence across genres")
```

[124]:
```
Text(0.5, 1.0, 'valence across genres')
```



Speechiness, loudness, danceability, id

4 Training of First Model: All Genres(Speechiness, Loudness, Danceability)

Our first instance with be trained off the speechiness, loudness, and danceability of songs from all genres.

Hypothesis: Due to there being some genres very similar to one another, our first model will likely have a difficult time predicting some of the rock based genres.

[128]:
```
comboD = pnda.read_csv("dataSets/trainingData1/all_genres_SLP.csv")
```

[129]:
```
scaler = StandardScaler()
scaler.fit(comboD.drop('target',axis=1))
scaled_features = scaler.transform(comboD.drop('target',axis = 1))

all_genresSLD_feat = pnda.DataFrame(scaled_features, columns = comboD.columns[:
 ↪-1])
```

[130]:
```
from sklearn.model_selection import train_test_split

x = all_genresSLD_feat
y = comboD['target']

x_train,x_test,y_train,y_test  = train_test_split(x,y,test_size=0.3,␣
 ↪random_state = 30, shuffle = True)
```

[131]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 7, p = 2, metric = 'euclidean')
knn.fit(x_train,y_train)
```

[131]:
```
KNeighborsClassifier(metric='euclidean', n_neighbors=7)
```

[139]:
```
prediction = knn.predict(x_test)
```

4.1 Results of First Instance

[133]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

           1       0.06      0.04      0.05       644
           2       0.25      0.24      0.25       632
           3       0.61      0.73      0.67       820
           4       0.20      0.19      0.19      1274
           5       0.38      0.36      0.37       885
           6       0.37      0.30      0.33      1159
           7       0.45      0.52      0.48      2610

    accuracy                           0.38      8024
   macro avg       0.33      0.34      0.33      8024
weighted avg       0.36      0.38      0.37      8024
```

Tags(1 = alt, 2 = blues, 3 = hiphop, 4 = indie, 5 = metal, 6 = pop, 7 = rock)

The results above show that our KNN method was more likely to correctly identify a hiphop song more than any other genre.

Metal, pop, and rock were all identified ~%40 of the time.

Blues and indie music were identified ~20-25% of the time.

Alternative was only identified 6% of the time.

What happens if we use a more appropriate K value?

Let's find that K-Value!

[ ]:
```
error_rate = []

for i in range(1,60):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train, y_train)
    prediction_i = knn.predict(x_test)
```

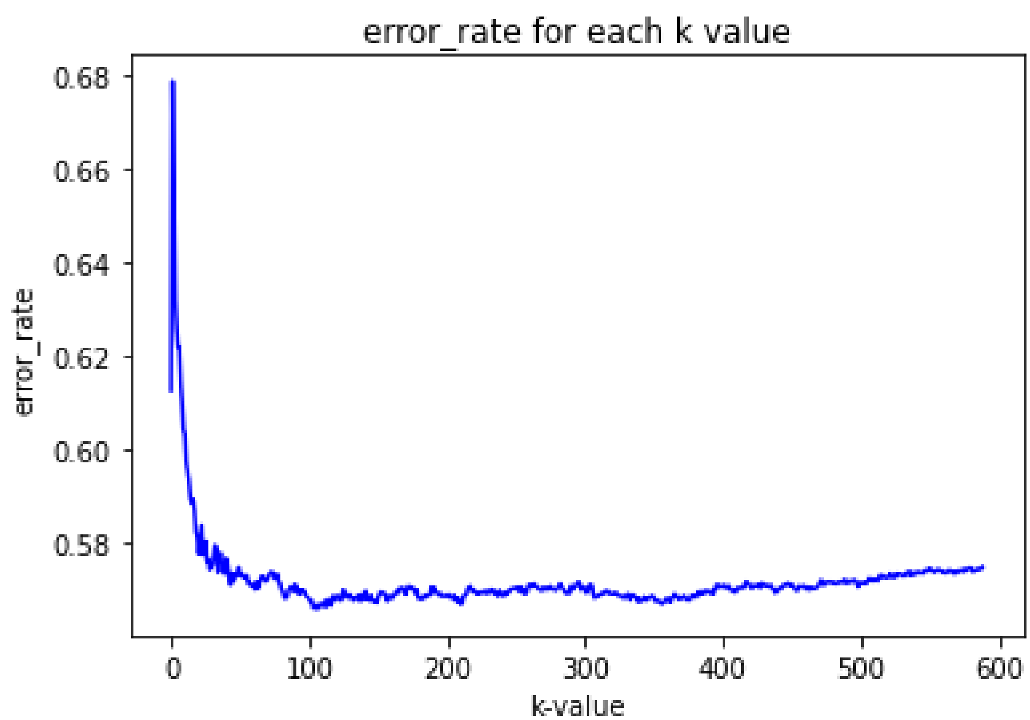
```
    error_rate.append(npy.mean(prediction_i != y_test))
```

[143]:
```
minError = 1000
minPos = 1;
index = 1;
for i in error_rate:

    if (i < minError):
        minError = i
        minPos = index

    index = index+1
print (minPos)
```

107

[144]:
```
ppt.plot(error_rate, color='blue')

ppt.xlabel("k-value")
ppt.ylabel("error_rate")
ppt.title("error_rate for each k value")
```

[144]: Text(0.5, 1.0, 'error_rate for each k value')



[146]:
```python
knn = KNeighborsClassifier(n_neighbors = 59, p = 2, metric = "euclidean")
knn.fit(x_train,y_train)
prediction = knn.predict(x_test)
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

           1       0.50      0.00      0.01       644
           2       0.35      0.10      0.15       632
           3       0.66      0.73      0.69       820
           4       0.24      0.12      0.16      1274
           5       0.46      0.25      0.32       885
           6       0.41      0.30      0.35      1159
           7       0.41      0.79      0.54      2610

    accuracy                           0.43      8024
   macro avg       0.43      0.33      0.32      8024
weighted avg       0.42      0.43      0.37      8024
```

K value = 7

```
                  precision    recall  f1-score   support

           1       0.06      0.04      0.05       644
           2       0.25      0.24      0.25       632
           3       0.61      0.73      0.67       820
           4       0.20      0.19      0.19      1274
           5       0.38      0.36      0.37       885
           6       0.37      0.30      0.33      1159
           7       0.45      0.52      0.48      2610

    accuracy                           0.38      8024
   macro avg       0.33      0.34      0.33      8024
weighted avg       0.36      0.38      0.37      8024
```

K Value = 59

```
                  precision    recall  f1-score   support

           1       0.50      0.00      0.01       644
           2       0.35      0.10      0.15       632
           3       0.66      0.73      0.69       820
           4       0.24      0.12      0.16      1274
           5       0.46      0.25      0.32       885
           6       0.41      0.30      0.35      1159
           7       0.41      0.79      0.54      2610

    accuracy                           0.43      8024
   macro avg       0.43      0.33      0.32      8024
```

```
weighted avg       0.42      0.43      0.37      8024
```

Here we notice a huge uptick in the precision of classifying "alternaive" songs. Previously, our precision was only about 6% for alternative, but now it is 50%. Our averages have also increased by 10%!

5 Training/Testing Second Instance: Hiphop/Rock(Speechiness, Loudness, Danceability)

For the second iteration of training, we will be aiming for success.

Looking back at the first instance's results, we see that rock and rap had the highest precision and recall rates. We will use these two genres in our next instanc.

[156]:
```
hh_rock_SLD = pnda.read_csv("dataSets/trainingData2/hh_rock.csv")
scaler2 = StandardScaler()
scaler2.fit(hh_rock_SLD.drop('target',axis=1))
scaled_features2 = scaler2.transform(hh_rock_SLD.drop('target',axis = 1))

hh_rock_SLD_features = pnda.DataFrame(scaled_features2, columns = hh_rock_SLD.
 ↪columns[:-1])

x2 = hh_rock_SLD_features
y2 = hh_rock_SLD['target']

x_train2,x_test2,y_train2,y_test2  = train_test_split(x2,y2,test_size=0.3,␣
 ↪random_state = 30, shuffle = True)

knn2 = KNeighborsClassifier(n_neighbors = 59, p = 2, metric = 'euclidean')
knn2.fit(x_train2,y_train2)
prediction2 = knn2.predict(x_test2)
```

5.1 Results of Second Instance (K=59)

[144]:
```
print(classification_report(y_test2, prediction2))
```

```
              precision    recall  f1-score   support

           3       0.89      0.78      0.83       780
           7       0.94      0.97      0.95      2619

    accuracy                           0.93      3399
   macro avg       0.92      0.88      0.89      3399
weighted avg       0.93      0.93      0.93      3399
```

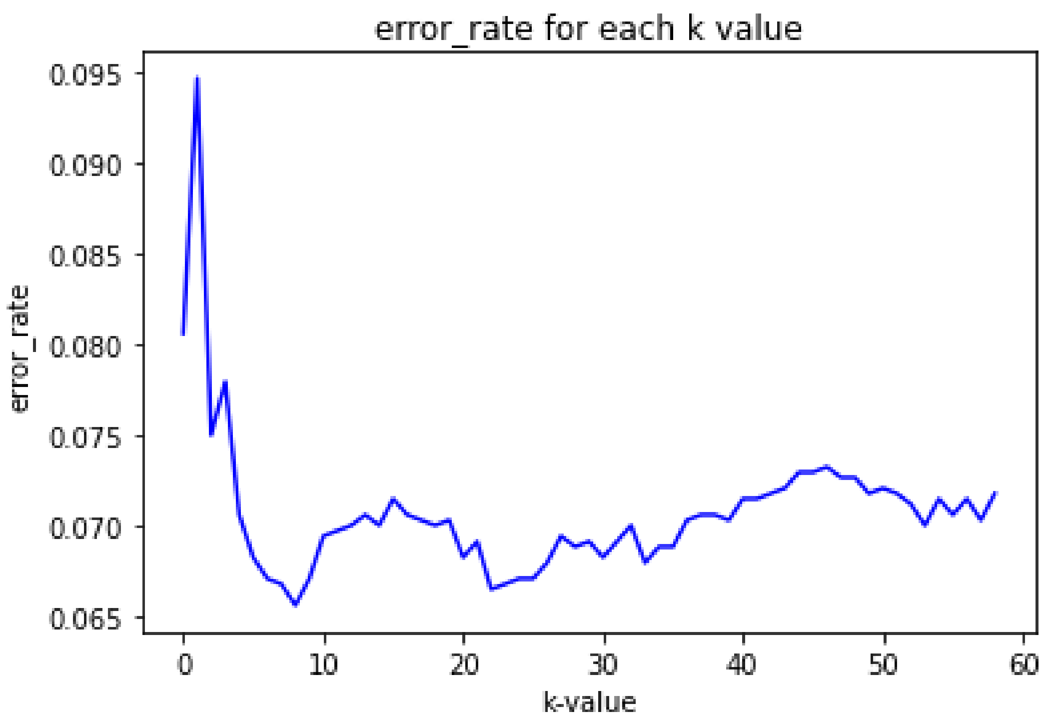
Here we notice a huge improvement in the precision and recall of our model. By just using 2 classes and the same K-value. Let's systematically find our K-value again. ## Refining K-Value for Second Instance

[113]:
```python
error_rate2 = []

for i in range(1,60):  #gather error_rates

    knn2 = KNeighborsClassifier(n_neighbors=i)
    knn2.fit(x_train2, y_train2)
    prediction_i2 = knn2.predict(x_test2)
    error_rate2.append(npy.mean(prediction_i2 != y_test2))
```

```
done
```

[122]:
```python
ppt.plot(error_rate2, color='blue')

ppt.xlabel("k-value")
ppt.ylabel("error_rate")
ppt.title("error_rate for each k value")
```

[122]:
```
Text(0.5, 1.0, 'error_rate for each k value')
```



[114]:
```python
minError = 100
minPos = 1;
index = 1;
for i in error_rate2:
```

```
    if (i < minError):
        minError = i
        minPos = index

    index = index+1
print (minPos)
```

```
9
```

5.2 Second Instance Results (K=9)

[124]:
```
knn2 = KNeighborsClassifier(n_neighbors = 9, p = 2, metric = 'euclidean')
knn2.fit(x_train2,y_train2)
prediction2 = knn2.predict(x_test2)
print(classification_report(y_test2, prediction2))
```

```
              precision    recall  f1-score   support

           3       0.88      0.82      0.85       780
           7       0.95      0.97      0.96      2619

    accuracy                           0.93      3399
   macro avg       0.92      0.89      0.90      3399
weighted avg       0.93      0.93      0.93      3399
```

6 Training Third Model: Blues, Hiphop, Rock (Speechiness, Loudness, Danceability)

[140]:
```python
blues_hh_rock_SLD = pnda.read_csv("dataSets/trainingData3/blues_hh_rock.csv")
scaler3 = StandardScaler()
scaler3.fit(blues_hh_rock_SLD.drop('target',axis=1))
scaled_features3 = scaler3.transform(blues_hh_rock_SLD.drop('target',axis = 1))

blues_hh_rock_SLD_features = pnda.DataFrame(scaled_features3, columns =␣
↪blues_hh_rock_SLD.columns[:-1])

x3 = blues_hh_rock_SLD_features
y3 = blues_hh_rock_SLD['target']

x_train3,x_test3,y_train3,y_test3  = train_test_split(x3,y3,test_size=0.3,␣
↪random_state = 30, shuffle = True)

knn3 = KNeighborsClassifier(n_neighbors = 9, p = 2, metric = 'euclidean')
knn3.fit(x_train3,y_train3)
prediction3 = knn3.predict(x_test3)
prediction3
```

[140]:
```
array([3, 7, 7, …, 7, 7, 3], dtype=int64)
```

6.1 Third Instance Results (K=9)

[25]:
```python
print(classification_report(y_test3, prediction3))
```

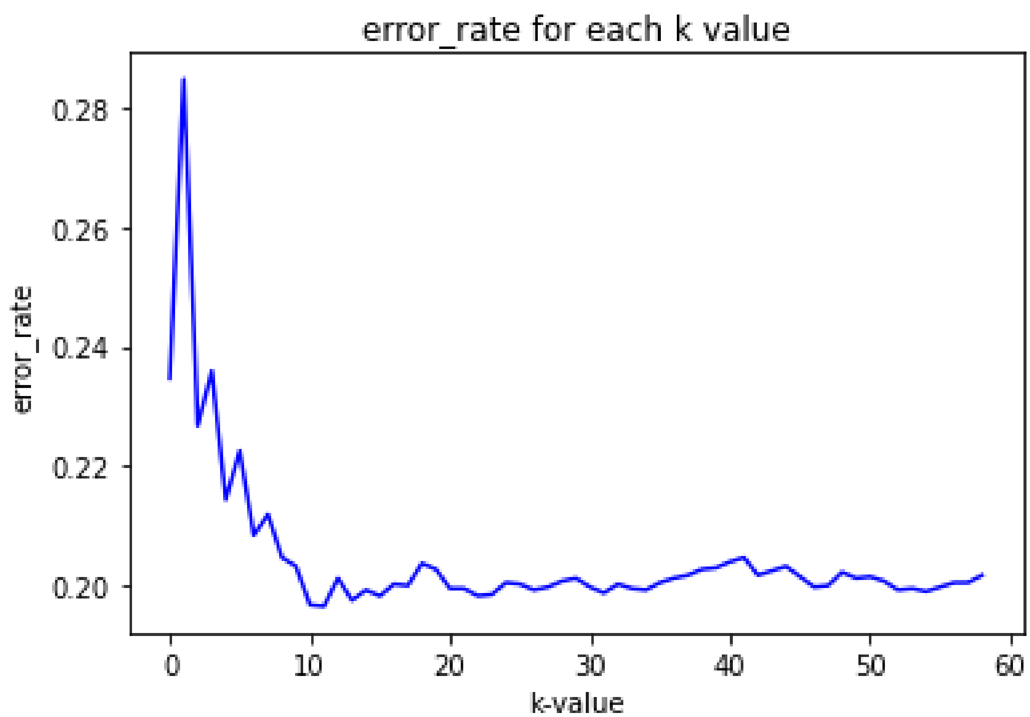
```
              precision    recall  f1-score   support

           2       0.38      0.34      0.36       592
           3       0.84      0.78      0.81       753
           7       0.83      0.87      0.85      2669

    accuracy                           0.77      4014
   macro avg       0.68      0.66      0.67      4014
weighted avg       0.77      0.77      0.77      4014
```

6.2 Refining Third Instance's K-Value

[129]:
```python
error_rate3 = []

for i in range(1,60):  #gather error_rates

    knn3 = KNeighborsClassifier(n_neighbors=i)
    knn3.fit(x_train3, y_train3)
    prediction_i3 = knn3.predict(x_test3)
    error_rate3.append(npy.mean(prediction_i3 != y_test3))
```

[130]:
```
ppt.plot(error_rate3, color='blue')

ppt.xlabel("k-value")
ppt.ylabel("error_rate")
ppt.title("error_rate for each k value")
```

[130]: Text(0.5, 1.0, 'error_rate for each k value')



[132]:
```
minError = 100
minPos = 1;
index = 1;
for i in error_rate3:

    if (i < minError):
        minError = i
        minPos = index

    index = index+1
print (minPos)
```

12

6.3 Results of Third Instance (K=12)

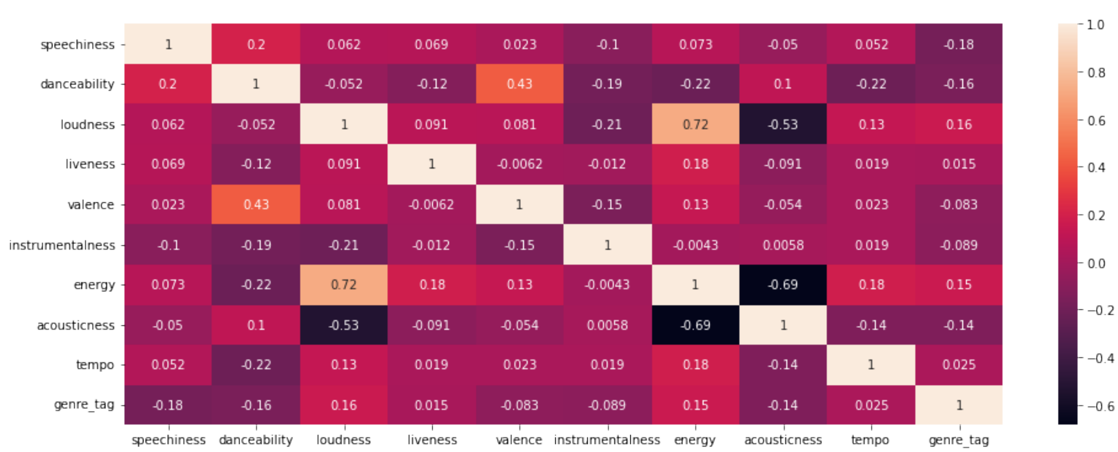
[139]:
```
knn3 = KNeighborsClassifier(n_neighbors = 12, p = 2, metric = 'euclidean')
knn3.fit(x_train3,y_train3)
prediction3 = knn3.predict(x_test3)
print(classification_report(y_test3, prediction3))
```

```
              precision    recall  f1-score   support

           2       0.50      0.22      0.31       592
           3       0.83      0.81      0.82       753
           7       0.82      0.93      0.87      2669

    accuracy                           0.80      4014
   macro avg       0.72      0.65      0.67      4014
weighted avg       0.78      0.80      0.78      4014
```

7 Taking Another Look at our Data

[148]:
```
ppt.figure(figsize = (16, 6))
full_dataset =  pnda.concat([alt,pnda.concat([blues,pnda.concat([hiphop,pnda.
↪concat([indie,pnda.concat([metal,pnda.concat([pop,pnda.
↪concat([rock])])])])])])])
sbn.heatmap(full_dataset[['speechiness','danceability','loudness','liveness',␣
↪'valence','instrumentalness','energy','acousticness','tempo','genre_tag'] ].
↪corr(),annot=True)
```

[148]: <AxesSubplot:>



Song features we missed: energy, and acousticness -> Both have high positive/negative correlations to their genre tags.

8 Instance 4: Adding More Information to our Second Experiment

[181]:
```
hh_rock_SLDEA = pnda.read_csv("dataSets/trainingData4/hh_rock_sldea.csv")
scaler4 = StandardScaler()
scaler4.fit(hh_rock_SLDEA.drop('target',axis=1))
scaled_features4 = scaler4.transform(hh_rock_SLDEA.drop('target',axis = 1))

hh_rock_SLDEA_features = pnda.DataFrame(scaled_features4, columns =␣
↪hh_rock_SLDEA.columns[:-1])

x4 = hh_rock_SLDEA_features
y4 = hh_rock_SLDEA['target']

x_train4,x_test4,y_train4,y_test4  = train_test_split(x4,y4,test_size=0.3,␣
↪random_state = 30, shuffle = True)

knn4 = KNeighborsClassifier(n_neighbors = 59, p = 2, metric = 'euclidean')
knn4.fit(x_train4,y_train4)
prediction4 = knn4.predict(x_test4)
prediction4
```

[181]:
```
array([7, 7, 7, …, 7, 7, 3], dtype=int64)
```

[187]:
```
 knn4 = KNeighborsClassifier(n_neighbors = 59, p = 2, metric = 'euclidean')
knn4.fit(x_train4,y_train4)
prediction4 = knn4.predict(x_test4)
print(classification_report(y_test4, prediction4))
```

```
              precision    recall  f1-score   support

           3       0.91      0.78      0.84       780
           7       0.94      0.98      0.96      2617

    accuracy                           0.93      3397
   macro avg       0.93      0.88      0.90      3397
weighted avg       0.93      0.93      0.93      3397
```

While our original classification results were showing great results, we wonder if we can bump up our precisions by playing testing for a better K-Value

[179]:
```
error_rate4 = []

for i in range(1,70):  #gather error_rates

    knn4 = KNeighborsClassifier(n_neighbors=i)
    knn4.fit(x_train4, y_train4)
    prediction_i4 = knn4.predict(x_test4)
```

```
    error_rate4.append(npy.mean(prediction_i4 != y_test4))
```

[180]:
```
minError = 1000
minPos = 1;
index = 1;
for i in error_rate4:

    if (i < minError):
        minError = i
        minPos = index

    index = index+1
print (minPos)
```

```
12
```

[188]:
```
 knn4 = KNeighborsClassifier(n_neighbors = 12, p = 2, metric = 'euclidean')
knn4.fit(x_train4,y_train4)
prediction4 = knn4.predict(x_test4)
print(classification_report(y_test4, prediction4))
```

```
              precision    recall  f1-score   support

           3       0.90      0.83      0.86       780
           7       0.95      0.97      0.96      2617

    accuracy                           0.94      3397
   macro avg       0.93      0.90      0.91      3397
weighted avg       0.94      0.94      0.94      3397
```

9 Instance 5: Adding more information our third instance

This time we will retry our third experiment while including energy and acouticness in our predictions.

We've now learned that we should calculate the K-Value before running the instance.

[207]:
```
blues_hh_rock_SLDEA = pnda.read_csv("dataSets/trainingData5/blues_hh_rockSLDEA.
 ↪csv")
scaler5 = StandardScaler()
scaler5.fit(blues_hh_rock_SLDEA.drop('target',axis=1))
scaled_features5 = scaler5.transform(blues_hh_rock_SLDEA.drop('target',axis =␣
 ↪1))

blues_hh_rock_SLDEA_features = pnda.DataFrame(scaled_features5, columns =␣
 ↪blues_hh_rock_SLDEA.columns[:-1])
```

```
x5 = blues_hh_rock_SLDEA_features
y5 = blues_hh_rock_SLDEA['target']

x_train5,x_test5,y_train5,y_test5 = train_test_split(x5,y5,test_size=0.3,␣
↪random_state = 30, shuffle = True)
```

[206]:
```
error_rate5 = []

for i in range(1,70):  #gather error_rates

    knn5 = KNeighborsClassifier(n_neighbors=i)
    knn5.fit(x_train5, y_train5)
    prediction_i5 = knn5.predict(x_test5)
    error_rate5.append(npy.mean(prediction_i5 != y_test5))
```

[205]:
```
minError = 1000
minPos = 1;
index = 1;
for i in error_rate5:

    if (i < minError):
        minError = i
        minPos = index

    index = index+1
print (minPos)
```

```
30
```

9.1 Instance 5: Results

[204]:
```
 knn5= KNeighborsClassifier(n_neighbors = 30, p = 2, metric = 'euclidean')
knn5.fit(x_train5,y_train5)
prediction5 = knn5.predict(x_test5)
print(classification_report(y_test5, prediction5))
```

```
              precision    recall  f1-score   support

           2       0.58      0.29      0.38       592
           3       0.87      0.80      0.83       754
           7       0.83      0.94      0.88      2666

    accuracy                           0.82      4012
   macro avg       0.76      0.68      0.70      4012
weighted avg       0.80      0.82      0.80      4012
```

Previous attempt at our third experiment precision recall f1-score support

```
           2       0.50      0.22      0.31       592
           3       0.83      0.81      0.82       753
           7       0.82      0.93      0.87      2669

    accuracy                           0.80      4014
   macro avg       0.72      0.65      0.67      4014
weighted avg       0.78      0.80      0.78      4014
```

Our overall averages for precision and recall (and subsequently f1-score) increased slightly. Maybe energy and speechiness didn't include as much information as we wanted?

Let's give our very first experiment another go!

10 Instance 6: Final Instance

Here we will take everything we've learned about supervised learning, K-Nearest Neighbor, and our dataset to revisit our very first experiment. Let's look at those results again:

```
                   precision    recall  f1-score   support

           1       0.50      0.00      0.01       644
           2       0.35      0.10      0.15       632
           3       0.66      0.73      0.69       820
           4       0.24      0.12      0.16      1274
           5       0.46      0.25      0.32       885
           6       0.41      0.30      0.35      1159
           7       0.41      0.79      0.54      2610

    accuracy                           0.43      8024
   macro avg       0.43      0.33      0.32      8024
weighted avg       0.42      0.43      0.37      8024
```

[231]:
```
all_genres_SLDEA = pnda.read_csv("dataSets/trainingData6/all_genres_SLDEA.csv")

scaler6 = StandardScaler()
scaler6.fit(all_genres_SLDEA.drop('target',axis=1))

scaled_features6 = scaler6.transform(all_genres_SLDEA.drop('target',axis = 1))

all_genres_SLDEA_features = pnda.DataFrame(scaled_features6, columns =␣
 ↪all_genres_SLDEA.columns[:-1])

x6 = all_genres_SLDEA_features
y6 = all_genres_SLDEA['target']

x_train6,x_test6,y_train6,y_test6 = train_test_split(x6,y6,test_size=0.3,␣
 ↪random_state = 30, shuffle = True)
```

[232]:

```python
error_rate6 = []

for i in range(1,70):  #gather error_rates

    knn6 = KNeighborsClassifier(n_neighbors=i)
    knn6.fit(x_train6, y_train6)
    prediction_i6 = knn6.predict(x_test6)
    error_rate6.append(npy.mean(prediction_i6 != y_test6))
```

[233]:

```python
minError = 1000
minPos = 1;
index = 1;
for i in error_rate6:

    if (i < minError):
        minError = i
        minPos = index

    index = index+1
print (minPos)
```

```
33
```

[234]:

```python
 knn6= KNeighborsClassifier(n_neighbors = 33, p = 2, metric = 'euclidean')
knn6.fit(x_train6,y_train6)
prediction6 = knn6.predict(x_test6)
print(classification_report(y_test6, prediction6))
```

```
              precision    recall  f1-score   support

           1       0.11      0.00      0.01       643
           2       0.35      0.16      0.21       632
           3       0.67      0.72      0.70       820
           4       0.31      0.17      0.22      1316
           5       0.51      0.50      0.50       877
           6       0.42      0.46      0.44      1170
           7       0.45      0.70      0.55      2565

    accuracy                           0.46      8023
   macro avg       0.40      0.39      0.38      8023
weighted avg       0.42      0.46      0.42      8023
```

Let’s look back at our first experiment!

```
                precision    recall  f1-score   support

           1       0.50      0.00      0.01       644
           2       0.35      0.10      0.15       632
```

3	0.66	0.73	0.69	820
4	0.24	0.12	0.16	1274
5	0.46	0.25	0.32	885
6	0.41	0.30	0.35	1159
7	0.41	0.79	0.54	2610
accuracy			0.43	8024
macro avg	0.43	0.33	0.32	8024
weighted avg	0.42	0.43	0.37	8024

It seems that in the end, the increased information (energy and acousticness) did not significantly increase our overall averages. In the case of our sixth instance, all of our weighted averages dropped, except our recall rates.

11 Conclusion

By experiment with the Kth-Nearest Neighbor algorithm, supervised machine learning, and our classification problem, we were able to make some very intriguing findings regarding genre-classification.

The aim of our exploration was to learn and use ML and KNN to, with high probabily, sort songs from different genres into their respective genres. While our training and testing against all genres (alt,blues,hiphop,indie,metal,pop,rock) didn't prove to be too successful, we were still able to classify said genres ~41% of the time. When the genres were limited to just hiphop and rock music, we were able to secure a f1-score of 0.93. It is likely that our tests may be been made more difficult by the fact that some of these genres do share similarities in observed features.