

Machine learning

Q-7. Imagine you have a dataset where you need to predict the Genres of Music using an Unsupervised algorithm and you need to find the accuracy of the model, built-in docker, and use some library to display that in frontend

Dataset This is the Dataset You can use this dataset for this question.

In [2]:

```
1 ## Required Libraries
2 import numpy as np # Linear algebra
3 import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
4 from pandas.plotting import scatter_matrix
5 import seaborn as sns
6 import matplotlib.pyplot as plt
```

In [3]:

```
1 ## Loading the Dataset
2 data_1 = pd.read_csv('Downloads/archive (6)/data.csv')
3 data_2 = pd.read_csv('Downloads/archive (6)/data_2genre.csv')
4
5 data = pd.concat([data_1, data_2])
```

In [4]:

```
1 ## Checking top 5 row of Dataset
2 data.head()
```

Out[4]:

	filename	tempo	beats	chroma_stft	rmse	spectral_centroid	spectral_band
0	blues.00081.au	103.359375	50	0.380260	0.248262	2116.942959	1956.6
1	blues.00022.au	95.703125	44	0.306451	0.113475	1156.070496	1497.6
2	blues.00031.au	151.999081	75	0.253487	0.151571	1331.073970	1973.6
3	blues.00012.au	184.570312	91	0.269320	0.119072	1361.045467	1567.8
4	blues.00056.au	161.499023	74	0.391059	0.137728	1811.076084	2052.3

5 rows × 30 columns



Here is a list of the genres in our dataframe, along with their counts:

In [5]:

```
1 data['label'].value_counts()
```

Out[5]:

```
blues      100
classical  100
country    100
disco      100
hiphop     100
jazz       100
metal      100
pop        100
reggae     100
rock       100
1          100
2          100
Name: label, dtype: int64
```

It looks like there are some weird numerical values. By looking at the data I see that "1" corresponds to "pop" and the "2" corresponds to "classical". Let's change those.

In [6]:

```
1 data['label'] = data['label'].replace(to_replace={1: 'pop', 2: 'classical'})
```

Now we can see the true value counts

In [7]:

```
1 data['label'].value_counts()
```

Out[7]:

```
classical  200
pop        200
blues      100
country    100
disco      100
hiphop     100
jazz       100
metal      100
reggae     100
rock       100
Name: label, dtype: int64
```

Next, I'll do some exploratory data analysis to see what kind of relationships we have between our features.

Tempo Distribution

Here we can see the differences in tempo distribution between some different genres.

Most have fairly normal distributions with peaks around 100 BPM.

Disco is distinctly different, with a peak closer to 150 BPM.

Classical and Jazz are the most diverse, having less prominent peaks and a wider spread of tempos.

In [8]:

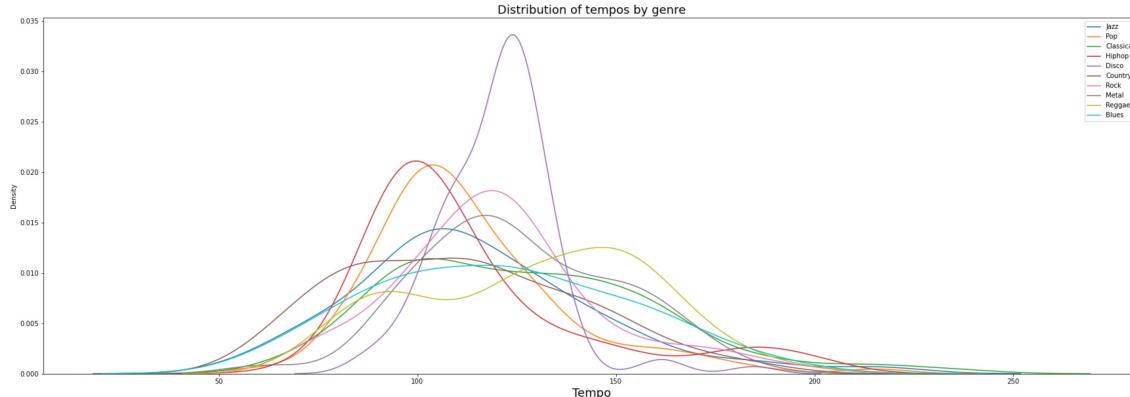
```

1 plt.figure(figsize=(30,10))
2
3 sns.kdeplot(data=data.loc[data['label']=='jazz', 'tempo'], label="Jazz")
4 sns.kdeplot(data=data.loc[data['label']=='pop', 'tempo'], label="Pop")
5 sns.kdeplot(data=data.loc[data['label']=='classical', 'tempo'], label="Classical")
6 sns.kdeplot(data=data.loc[data['label']=='hiphop', 'tempo'], label="Hiphop")
7 sns.kdeplot(data=data.loc[data['label']=='disco', 'tempo'], label="Disco")
8 sns.kdeplot(data=data.loc[data['label']=='country', 'tempo'], label="Country")
9 sns.kdeplot(data=data.loc[data['label']=='rock', 'tempo'], label="Rock")
10 sns.kdeplot(data=data.loc[data['label']=='metal', 'tempo'], label="Metal")
11 sns.kdeplot(data=data.loc[data['label']=='reggae', 'tempo'], label="Reggae")
12 sns.kdeplot(data=data.loc[data['label']=='blues', 'tempo'], label="Blues")
13
14 plt.title("Distribution of tempos by genre", fontsize = 18)
15
16 plt.xlabel("Tempo", fontsize = 18)
17
18 plt.legend()

```

Out[8]:

<matplotlib.legend.Legend at 0x220e33cb130>



Tempo Mean

However, if we look at the average tempo of each we can see that they are all very similar.

In [9]:

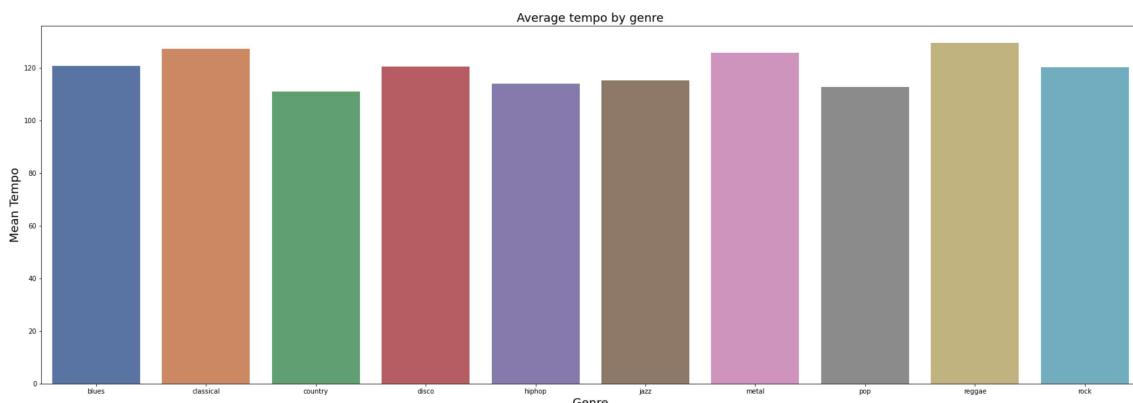
```

1 plt.figure(figsize=(30,10))
2
3 genres = data['label'].unique()
4
5 tempos = [ data[data['label']==x].tempo.mean() for x in genres ]
6
7 sns.barplot(x=genres, y=tempos, palette="deep")
8
9 plt.title("Average tempo by genre", fontsize = 18)
10
11 plt.xlabel('Genre', fontsize = 18)
12 plt.ylabel('Mean Tempo', fontsize = 18)

```

Out[9]:

Text(0, 0.5, 'Mean Tempo')



Now I am going to look at some of the less intuitive features in the dataset.

These features are more technical. I had to do some research to understand their meanings and implications, so I will explain them below.

1. Spectral Centroid

Spectral centroid is the average of frequencies weighted by amplitude, so a high spectral centroid implies that higher frequencies have higher amplitudes, or are more prominent, in this sample.

Spectral Centroid Distribution

Songs in the classical, jazz, and country genres seem to trend toward lower spectral centroids, while pop, disco, hiphop, and metal songs tend to have higher centroids. It's possible that high spectral centroids could be correlated with catchy songs that grab your attention with high frequencies, while low spectral centroids correlate with low-toned, more relaxed music that is more common in classical, jazz, and country.

The classical and metal genres both have fairly low variance, implying that they are less diverse in terms of spectral centroids.

The distributions of classical and metal have very little overlap. We could discern between these two genres fairly accurately even if we only used this feature.

In [10]:

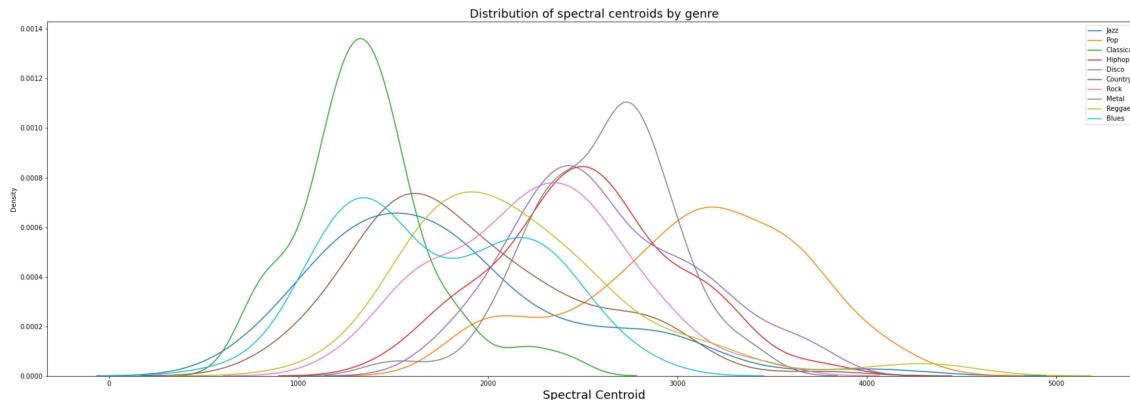
```

1 plt.figure(figsize=(30,10))
2
3 sns.kdeplot(data=data.loc[data['label']=='jazz', 'spectral_centroid'], label="Jazz")
4 sns.kdeplot(data=data.loc[data['label']=='pop', 'spectral_centroid'], label="Pop")
5 sns.kdeplot(data=data.loc[data['label']=='classical', 'spectral_centroid'], label="Classical")
6 sns.kdeplot(data=data.loc[data['label']=='hiphop', 'spectral_centroid'], label="Hip Hop")
7 sns.kdeplot(data=data.loc[data['label']=='disco', 'spectral_centroid'], label="Disco")
8 sns.kdeplot(data=data.loc[data['label']=='country', 'spectral_centroid'], label="Country")
9 sns.kdeplot(data=data.loc[data['label']=='rock', 'spectral_centroid'], label="Rock")
10 sns.kdeplot(data=data.loc[data['label']=='metal', 'spectral_centroid'], label="Metal")
11 sns.kdeplot(data=data.loc[data['label']=='reggae', 'spectral_centroid'], label="Reggae")
12 sns.kdeplot(data=data.loc[data['label']=='blues', 'spectral_centroid'], label="Blues")
13
14 plt.title("Distribution of spectral centroids by genre", fontsize = 18)
15
16 plt.xlabel("Spectral Centroid", fontsize = 18)
17
18 plt.legend()

```

Out[10]:

<matplotlib.legend.Legend at 0x220e55d07f0>



Spectral Centroid Mean

As can be seen below, there is much more variance in the means of the spectral centroids than there was for tempo.

In [11]:

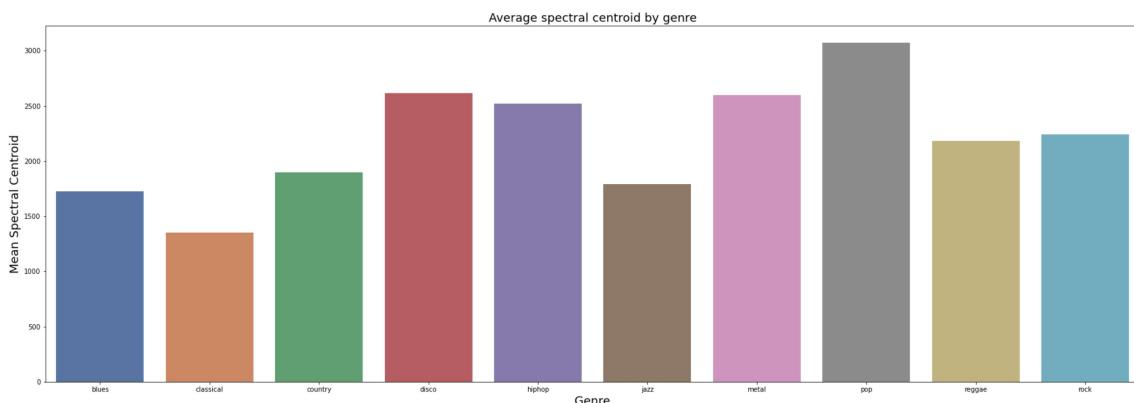
```

1 plt.figure(figsize=(30,10))
2
3 genres = data['label'].unique()
4
5 spectral_centroids = [ data[data['label']==x].spectral_centroid.mean() for x in gen
6
7 sns.barplot(x=genres, y=spectral_centroids, palette="deep")
8
9 plt.title("Average spectral centroid by genre", fontsize = 18)
10
11 plt.xlabel('Genre', fontsize = 18)
12 plt.ylabel('Mean Spectral Centroid', fontsize = 18)

```

Out[11]:

Text(0, 0.5, 'Mean Spectral Centroid')



2. Spectral Bandwidth

Spectral bandwidth is the width of the frequency band for which the frequencies are at least half of the maximum amplitude. Basically, it shows us how wide the range of prominent frequencies is.

Spectral Bandwidth Distribution

- Interestingly, there are three very distinct peaks in this graph: classical, metal, and pop. Their distributions have relatively low variance, and they have little overlap with each other, meaning that this feature will be useful in distinguishing them.
- Most classical songs have a smaller spectral bandwidth. This could be due to many classical songs being played by a single instrument, such as piano, limiting the tonal range.
- Pop songs tend to have higher bandwidths. This may be because most pop songs include multiple instruments and vocal parts.

In [12]:

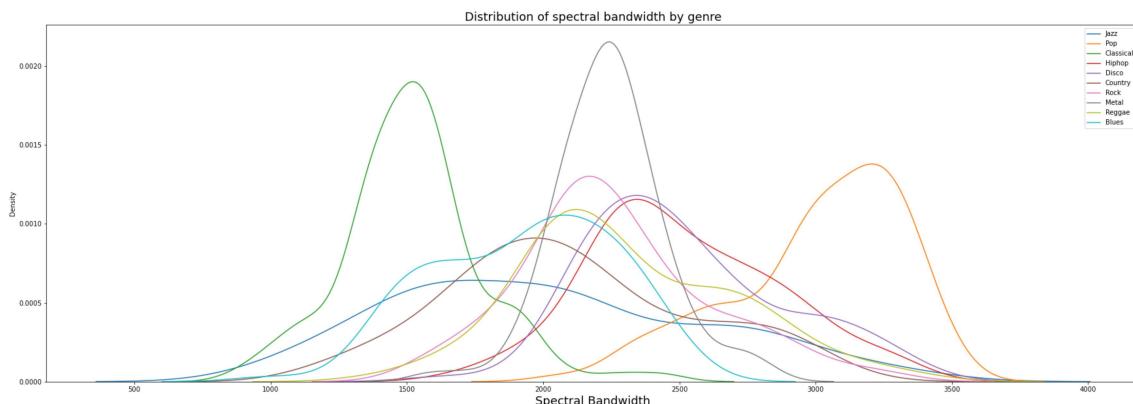
```

1 plt.figure(figsize=(30,10))
2
3 sns.kdeplot(data=data.loc[data['label']=='jazz', 'spectral_bandwidth'], label="Jazz")
4 sns.kdeplot(data=data.loc[data['label']=='pop', 'spectral_bandwidth'], label="Pop")
5 sns.kdeplot(data=data.loc[data['label']=='classical', 'spectral_bandwidth'], label="Classical")
6 sns.kdeplot(data=data.loc[data['label']=='hiphop', 'spectral_bandwidth'], label="Hip Hop")
7 sns.kdeplot(data=data.loc[data['label']=='disco', 'spectral_bandwidth'], label="Disco")
8 sns.kdeplot(data=data.loc[data['label']=='country', 'spectral_bandwidth'], label="Country")
9 sns.kdeplot(data=data.loc[data['label']=='rock', 'spectral_bandwidth'], label="Rock")
10 sns.kdeplot(data=data.loc[data['label']=='metal', 'spectral_bandwidth'], label="Metal")
11 sns.kdeplot(data=data.loc[data['label']=='reggae', 'spectral_bandwidth'], label="Reggae")
12 sns.kdeplot(data=data.loc[data['label']=='blues', 'spectral_bandwidth'], label="Blues")
13
14 plt.title("Distribution of spectral bandwidth by genre", fontsize = 18)
15
16 plt.xlabel("Spectral Bandwidth", fontsize = 18)
17
18 plt.legend()

```

Out[12]:

<matplotlib.legend.Legend at 0x220e641c970>



Spectral Bandwidth Mean

The spectral bandwidth means look very similar to the spectral centroid means. This may indicate some kind of correlation.

In [13]:

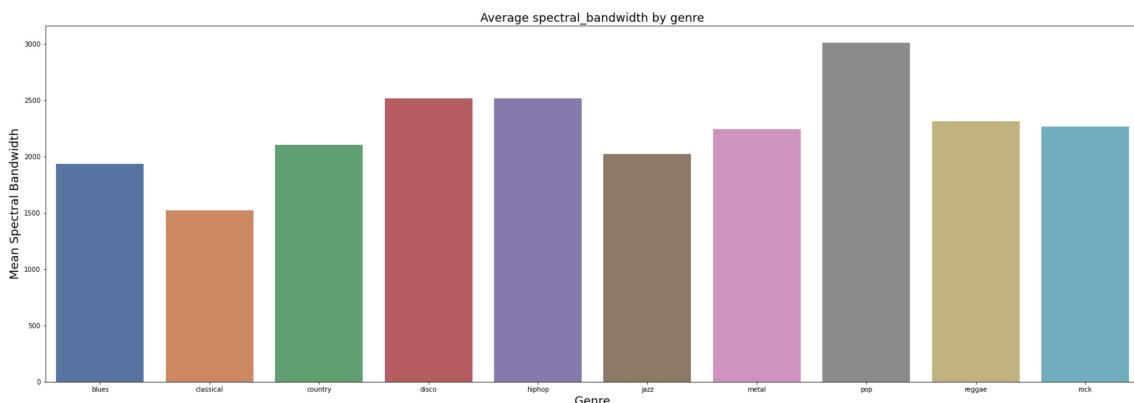
```

1 plt.figure(figsize=(30,10))
2
3 genres = data['label'].unique()
4
5 spectral_bandwidths = [ data[data['label']==x].spectral_bandwidth.mean() for x in g
6
7 sns.barplot(x=genres, y=spectral_bandwidths, palette="deep")
8
9 plt.title("Average spectral_bandwidth by genre", fontsize = 18)
10
11 plt.xlabel('Genre', fontsize = 18)
12 plt.ylabel('Mean Spectral Bandwidth', fontsize = 18)

```

Out[13]:

Text(0, 0.5, 'Mean Spectral Bandwidth')



3. Rolloff

Rolloff is a term typically used to describe filters. It describes the steepness of the transition from the stop band to the pass band (the stop band includes the blocked frequencies, while the pass band includes the audible frequencies). A higher rolloff might indicate music that has less overtones (peripheral frequencies with lower amplitude), or that sounds more "crisp" and clean.

Rolloff Distribution

The rolloff distributions looks almost the same as the spectral bandwidth distributions. This very likely indicates a correlation between the two.

Pop, disco, hiphop, and metal have high rolloff. This seems to support my theory about "crisp" sounding music.

In [14]:

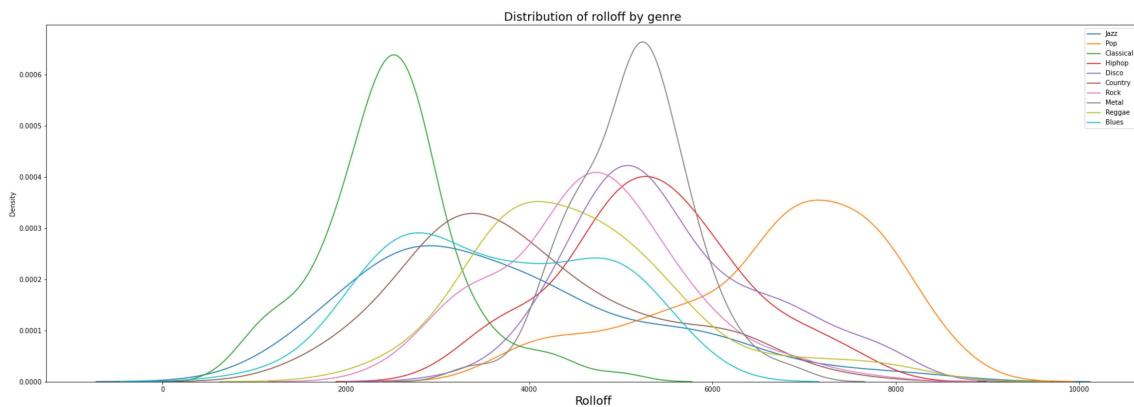
```

1 plt.figure(figsize=(30,10))
2
3 sns.kdeplot(data=data.loc[data['label']=='jazz', 'rolloff'], label="Jazz")
4 sns.kdeplot(data=data.loc[data['label']=='pop', 'rolloff'], label="Pop")
5 sns.kdeplot(data=data.loc[data['label']=='classical', 'rolloff'], label="Classical")
6 sns.kdeplot(data=data.loc[data['label']=='hiphop', 'rolloff'], label="Hiphop")
7 sns.kdeplot(data=data.loc[data['label']=='disco', 'rolloff'], label="Disco")
8 sns.kdeplot(data=data.loc[data['label']=='country', 'rolloff'], label="Country")
9 sns.kdeplot(data=data.loc[data['label']=='rock', 'rolloff'], label="Rock")
10 sns.kdeplot(data=data.loc[data['label']=='metal', 'rolloff'], label="Metal")
11 sns.kdeplot(data=data.loc[data['label']=='reggae', 'rolloff'], label="Reggae")
12 sns.kdeplot(data=data.loc[data['label']=='blues', 'rolloff'], label="Blues")
13 plt.title("Distribution of rolloff by genre", fontsize = 18)
14
15 plt.xlabel("Rolloff", fontsize = 18)
16
17 plt.legend()

```

Out[14]:

<matplotlib.legend.Legend at 0x220e69e2fa0>



Rolloff Means

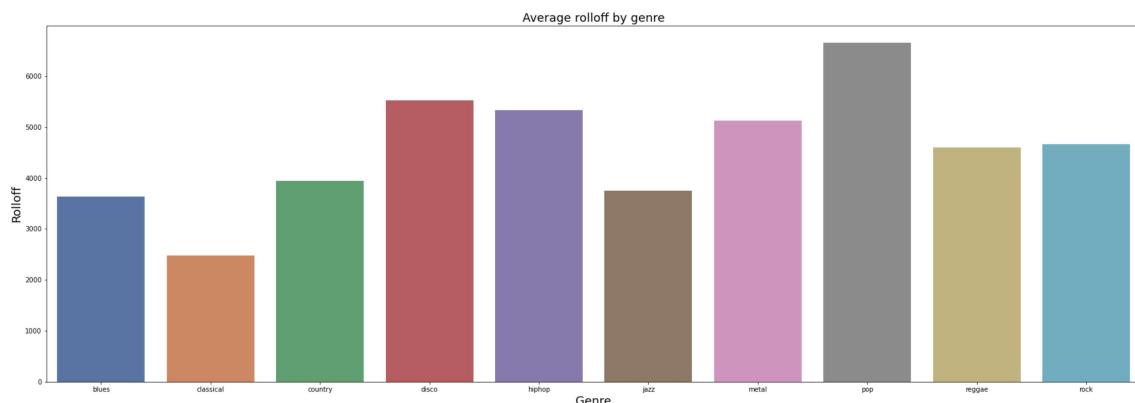
There is a lot of variance in the means of the rolloff. It also closely resembles the means of the spectral bandwidth.

In [15]:

```
1 plt.figure(figsize=(30,10))
2
3 genres = data['label'].unique()
4
5 rolloffs = [ data[data['label']==x].rolloff.mean() for x in genres ]
6
7 sns.barplot(x=genres, y=rolloffs, palette="deep")
8
9 plt.title("Average rolloff by genre", fontsize = 18)
10
11 plt.xlabel('Genre', fontsize = 18)
12 plt.ylabel('Rolloff', fontsize = 18)
```

Out[15]:

Text(0, 0.5, 'Rolloff')



Feature Correlations

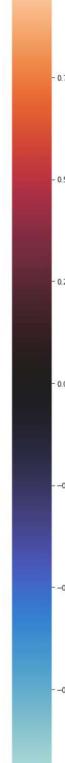
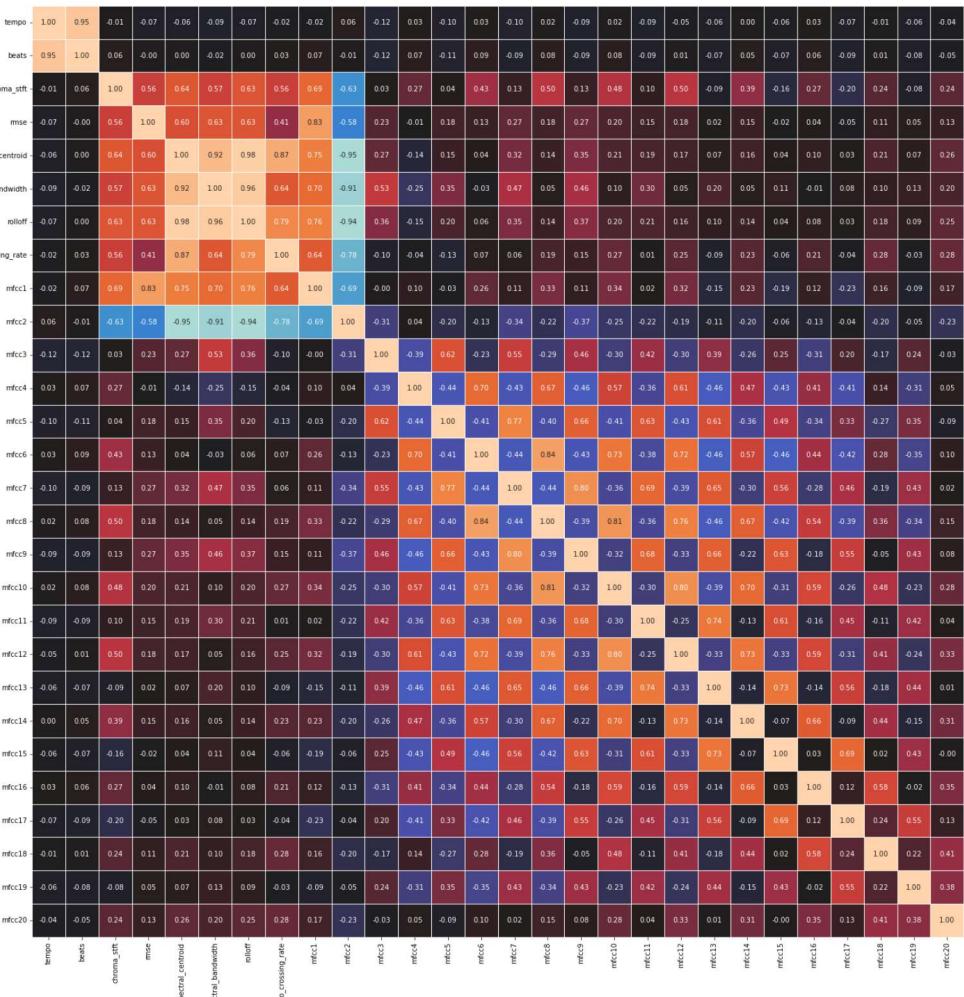
This heatmap shows the correlations between all of the features. This quantifies how close they are to a perfect linear relationship.

In [16]:

```

1 correlations = data.corr()
2
3 fig, ax = plt.subplots(figsize=(30,30))
4 sns.heatmap(correlations, vmax=1.0, center=0, fmt='.2f',
5               square=True, linewidths=.5, annot=True, cbar_kws={"shrink": .70})
6 plt.show();

```



- This filtered list more clearly shows the features with the strongest positive correlations.
- Here we can see that rolloff is strongly correlated with both spectral centroid and spectral bandwidth.
- I am not sure what the difference between tempo and beats is, but there seems to be some minor discrepancy.
- We can also see that there is also a fairly strong correlation between spectral bandwidth and centroid
- I would have expected the correlation between zero crossing rate and spectral centroid to be higher, since they are both dependent on frequency.

In [17]:

```

1 c = data.corr()
2
3 s = c.unstack()
4 so = s.sort_values(kind="quicksort")
5
6 print(so[745:-28])

```

```

mfcc8          mfcc6          0.843818
zero_crossing_rate  spectral_centroid  0.874095
spectral_centroid  zero_crossing_rate  0.874095
spectral_bandwidth spectral_centroid  0.920961
spectral_centroid  spectral_bandwidth 0.920961
beats           tempo          0.953903
tempo           beats          0.953903
rolloff          spectral_bandwidth 0.964335
spectral_bandwidth rolloff        0.964335
spectral_centroid  rolloff        0.982622
rolloff          spectral_centroid  0.982622
dtype: float64

```

This one shows the features with the strongest negative correlations.

mfcc2 (the second coefficient of the Mel-frequency cepstrum, a mathematical representation of the sound) has a strong negative correlation with centroid, rolloff, and bandwidth.

In [18]:

```

1 c = data.corr()
2
3 s = c.unstack()
4 so = s.sort_values(kind="quicksort")
5
6 print(so[:10])

```

```

spectral_centroid  mfcc2          -0.946137
mfcc2              spectral_centroid -0.946137
                    rolloff        -0.940227
rolloff            mfcc2          -0.940227
mfcc2              spectral_bandwidth -0.906566
spectral_bandwidth mfcc2          -0.906566
zero_crossing_rate mfcc2          -0.775487
mfcc2              zero_crossing_rate -0.775487
mfcc1              mfcc2          -0.689646
mfcc2              mfcc1          -0.689646
dtype: float64

```

Scatter Plots

These scatter plots effectively visualize the relationships between the highly correlated variables.

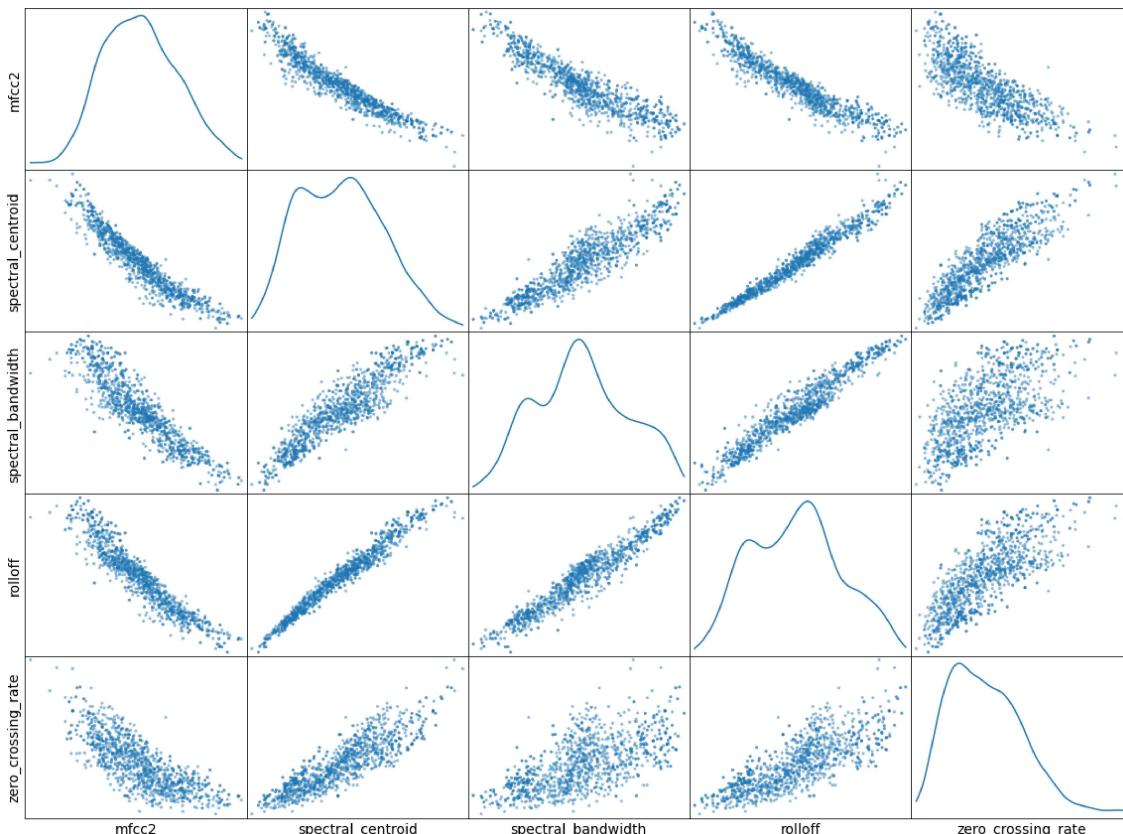
Most notably we can see that some variables have negative non-linear relationships.

In [19]:

```

1 attributes = ["mfcc2", "spectral_centroid",
2                 "spectral_bandwidth", "rolloff", "zero_crossing_rate"]
3 sm = scatter_matrix(data[attributes], figsize=(20, 15), diagonal = "kde");
4
5 #Hide all ticks
6 [s.set_xticks(()) for s in sm.reshape(-1)];
7 [s.set_yticks(()) for s in sm.reshape(-1)];
8
9 for x in sm.ravel():
10     x.set_xlabel(x.get_xlabel(), fontsize = 14)
11     x.set_ylabel(x.get_ylabel(), fontsize = 14)

```



Preprocessing and Feature Selection

Since all the data is numerical and we have no NaN values (shown below) preprocessing should be easy.

In [20]:

```
1 data.isna().sum()
```

Out[20]:

```
filename          0
tempo            0
beats            0
chroma_stft     0
rmse             0
spectral_centroid 0
spectral_bandwidth 0
rolloff          0
zero_crossing_rate 0
mfcc1            0
mfcc2            0
mfcc3            0
mfcc4            0
mfcc5            0
mfcc6            0
mfcc7            0
mfcc8            0
mfcc9            0
mfcc10           0
mfcc11           0
mfcc12           0
mfcc13           0
mfcc14           0
mfcc15           0
mfcc16           0
mfcc17           0
mfcc18           0
mfcc19           0
mfcc20           0
label            0
dtype: int64
```

First we should train and evaluate a model including all the features, and then one with some features removed to see which method is preferable.

In [21]:

```
1 from sklearn.model_selection import train_test_split
2 from xgboost.sklearn import XGBClassifier
3 from sklearn.ensemble import RandomForestClassifier
4 from sklearn.metrics import classification_report, accuracy_score
5 from sklearn.preprocessing import LabelEncoder
6 from sklearn.model_selection import KFold, GridSearchCV
7
8 data = data.drop('filename', axis=1)
9
10 X = data.loc[:, data.columns != 'label']
11 y = data['label']
12
13 label_encoder = LabelEncoder().fit(y)
14 name_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))
15
16 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```


In []:

```
1 rf_model = RandomForestClassifier()
2 xgb_model = XGBClassifier()
3 k_fold = KFold(n_splits=5, random_state=0, shuffle=True)
4
5 rf_params = {
6     'n_estimators': [ i*10 for i in range(15, 30) ],
7     'max_features': ['auto'],
8     'n_jobs': [-1],
9     'random_state': [0]
10 }
11
12 xgb_params = {
13     'max_depth': range (2, 10, 1),
14     'n_estimators': range(60, 220, 40),
15     'learning_rate': [0.1],
16     'n_jobs': [-1],
17     'random_state': [0]
18 }
19
20 rf_grid = GridSearchCV(estimator=rf_model, param_grid=rf_params, cv=k_fold, n_jobs=
21 xgb_grid = GridSearchCV(estimator=xgb_model, param_grid=xgb_params, cv=k_fold, n_jo
22
23 rf_grid.fit(X_train, y_train)
24 xgb_grid.fit(X_train, y_train)
25
26 rf_params_max = rf_grid.best_params_
27 xgb_params_max = xgb_grid.best_params_
28
29 print("RF accuracy:")
30 print(rf_grid.score(X_train, y_train))
31 print("RF params:")
32 print(rf_params_max)
33 print("")
34 print("XGB accuracy:")
35 print(xgb_grid.score(X_train, y_train))
36 print("XGB params:")
37 print(xgb_params_max)
38 print("")
39
40 rf_model = RandomForestClassifier(**rf_params_max)
41 xgb_model = XGBClassifier(**xgb_params_max)
42
43 rf_model.fit(X_train, y_train)
44 xgb_model.fit(X_train, y_train)
45
46 rf_preds = rf_model.predict(X_test)
47 xgb_preds = xgb_model.predict(X_test)
48
49
50 print("RF validation accuracy")
51 print(accuracy_score(y_test, rf_preds))
52 print("")
53 print("Random Forest Classification Report: \n" + classification_report(y_test, rf_
54 print("")
55
56 print("XGB validation accuracy:")
57 print(accuracy_score(y_test, xgb_preds))
58 print("")
59 print("XGB Classification Report: \n" + classification_report(y_test, xgb_preds))
```

```
60 print("")
```

Now we can drop some of the features to see if it improves the model

s expected, removing the highly correlated features had very little effect.

In []:

```
1 # Create models with the parameters found in grid search
2 rf_model = RandomForestClassifier(**rf_params_max)
3 xgb_model = XGBClassifier(**xgb_params_max)
4
5 X_train = X_train.drop(['rolloff', 'mfcc2', 'beats'], axis=1)
6 X_test = X_test.drop(['rolloff', 'mfcc2', 'beats'], axis=1)
7
8 rf_grid = GridSearchCV(estimator=rf_model, param_grid=rf_params, cv=k_fold, n_jobs=-1)
9 xgb_grid = GridSearchCV(estimator=xgb_model, param_grid=xgb_params, cv=k_fold, n_jobs=-1)
10
11 rf_grid.fit(X_train, y_train)
12 xgb_grid.fit(X_train, y_train)
13
14 rf_params_max = rf_grid.best_params_
15 xgb_params_max = xgb_grid.best_params_
16
17 print("RF accuracy:")
18 print(rf_grid.score(X_train, y_train))
19 print("RF params:")
20 print(rf_params_max)
21 print("")
22 print("XGB accuracy:")
23 print(xgb_grid.score(X_train, y_train))
24 print("XGB params:")
25 print(xgb_params_max)
26 print("")
27
28 rf_model = RandomForestClassifier(**rf_params_max)
29 xgb_model = XGBClassifier(**xgb_params_max)
30
31 rf_model.fit(X_train, y_train)
32 xgb_model.fit(X_train, y_train)
33
34 rf_preds = rf_model.predict(X_test)
35 xgb_preds = xgb_model.predict(X_test)
36
37 print("RF validation accuracy")
38 print(accuracy_score(y_test, rf_preds))
39 print("")
40 print("Random Forest Classification Report: \n" + classification_report(y_test, rf_preds))
41 print("")
42
43 print("XGB validation accuracy")
44 print(accuracy_score(y_test, xgb_preds))
45 print("")
46 print("XGB Classification Report: \n" + classification_report(y_test, xgb_preds))
47 print("")
```

Model Summary

We were able to classify about 70% of songs correctly in the test set. XGBoost very slightly outperformed the Random Forest model, in most cases by 1-2%. It could be worth trying additional models or feature engineering techniques in the future to see if performance can be improved.

In []:

```
1
```