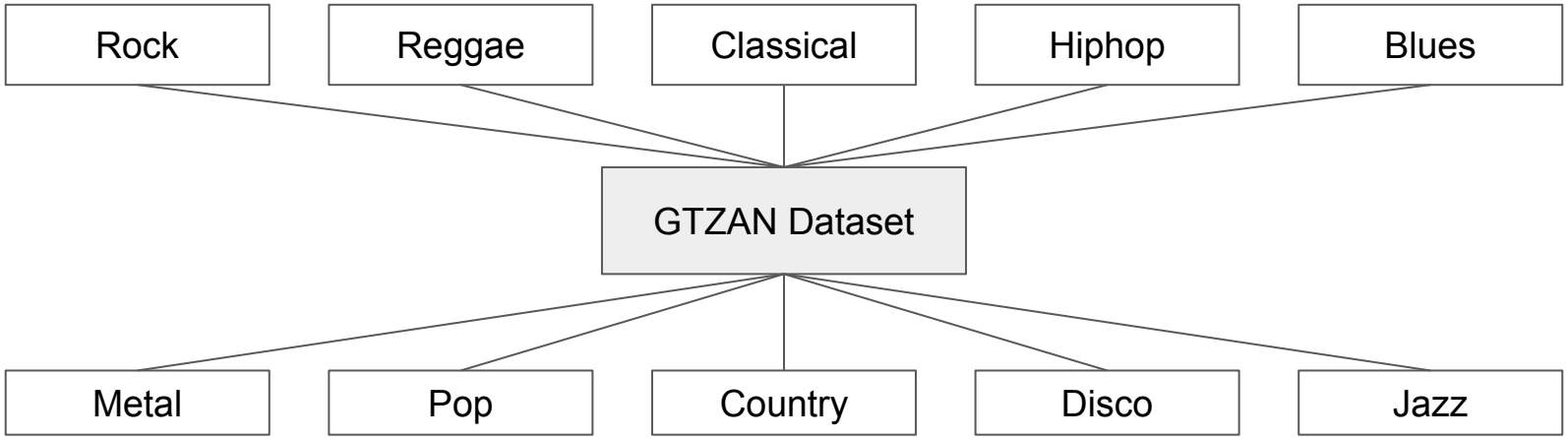
General Assembly DSI - Capstone Project
Music Genre Classification

Joh Akaishi
4 Dec 2020

1. Problem Statement
2. EDA & Preprocessing
3. Audio Features
4. Spectrograms
5. Conclusions & Future Expansion

➢ The GTZAN dataset consists of 10 different genres of music:



➢ Multiclass classification using (2 independent approaches):
- Success metric: accuracy

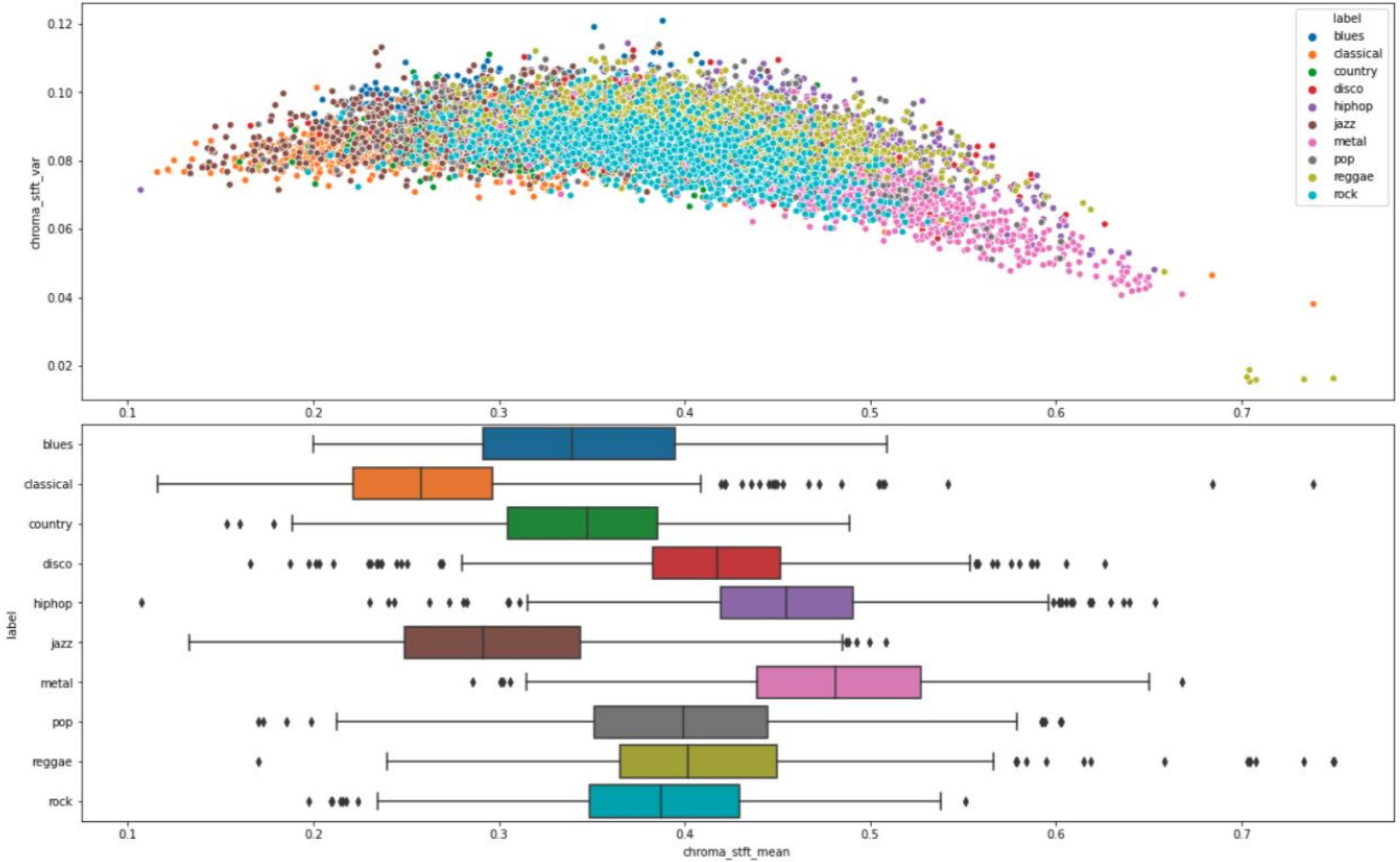
1. Problem Statement
2. EDA & Preprocessing
3. Audio Features
4. Spectrograms
5. Conclusions & Future Expansion

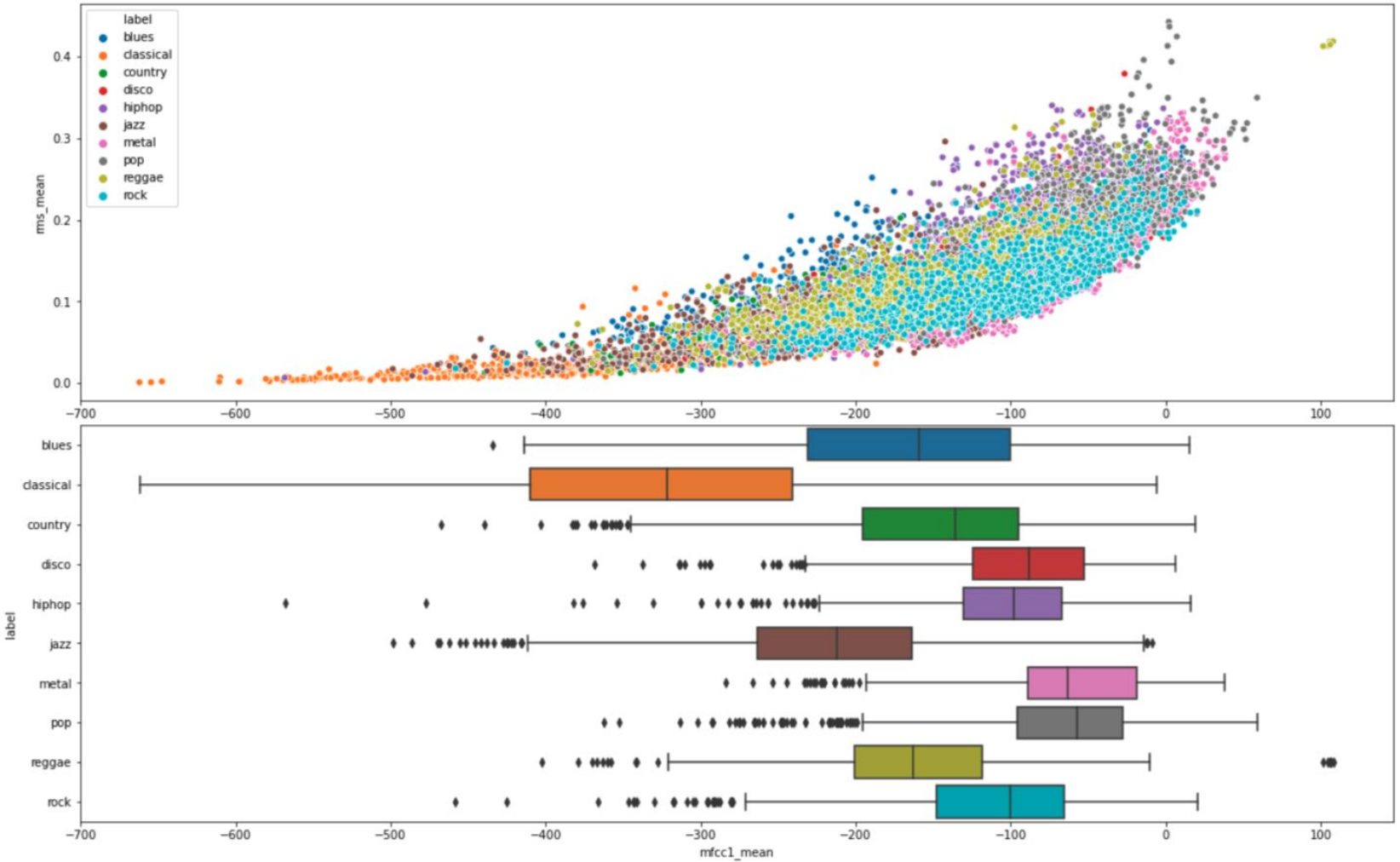
- Original dataset shape: 9990 rows x 60 cols.
  - But consisted of some duplicate rows, some across different labels of genre.

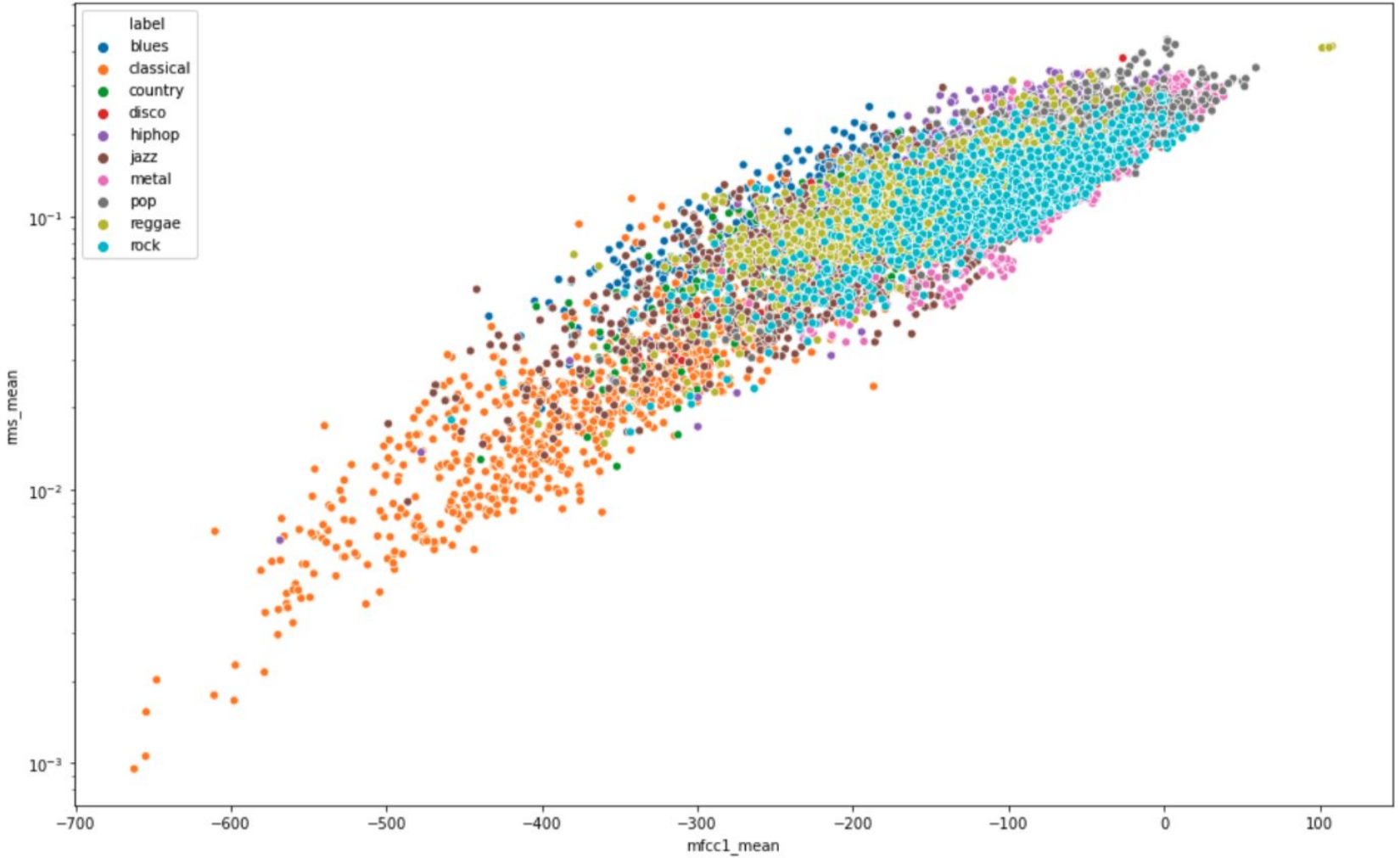
	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var	spectral_bandwidth_m
4445	hiphop.00045.3.wav	66149	0.304791	0.102734	0.178455	0.004849	2506.040096	2.041799e+06	2723.751
4385	hiphop.00039.3.wav	66149	0.304791	0.102734	0.178455	0.004849	2506.040096	2.041799e+06	2723.751
4444	hiphop.00045.2.wav	66149	0.315420	0.102082	0.148629	0.002391	2551.680954	1.562116e+06	2892.195
4384	hiphop.00039.2.wav	66149	0.315420	0.102082	0.148629	0.002391	2551.680954	1.562116e+06	2892.195
6623	metal.00063.1.wav	66149	0.322256	0.089340	0.101789	0.000377	2426.551665	7.442035e+04	2008.064
...	...	...	...	...	...	...	...	...	
6607	metal.00061.5.wav	66149	0.545779	0.069771	0.102126	0.000638	2836.894862	3.061083e+05	2247.514
9156	rock.00016.4.wav	66149	0.580280	0.051524	0.155180	0.000769	2499.091446	7.744979e+04	2253.036
6576	metal.00058.4.wav	66149	0.580280	0.051524	0.155180	0.000769	2499.091446	7.744979e+04	2253.036
6609	metal.00061.7.wav	66149	0.592884	0.053974	0.087314	0.000399	2728.993235	9.829380e+04	2311.647
6399	metal.00040.7.wav	66149	0.592884	0.053974	0.087314	0.000399	2728.993235	9.829380e+04	2311.647

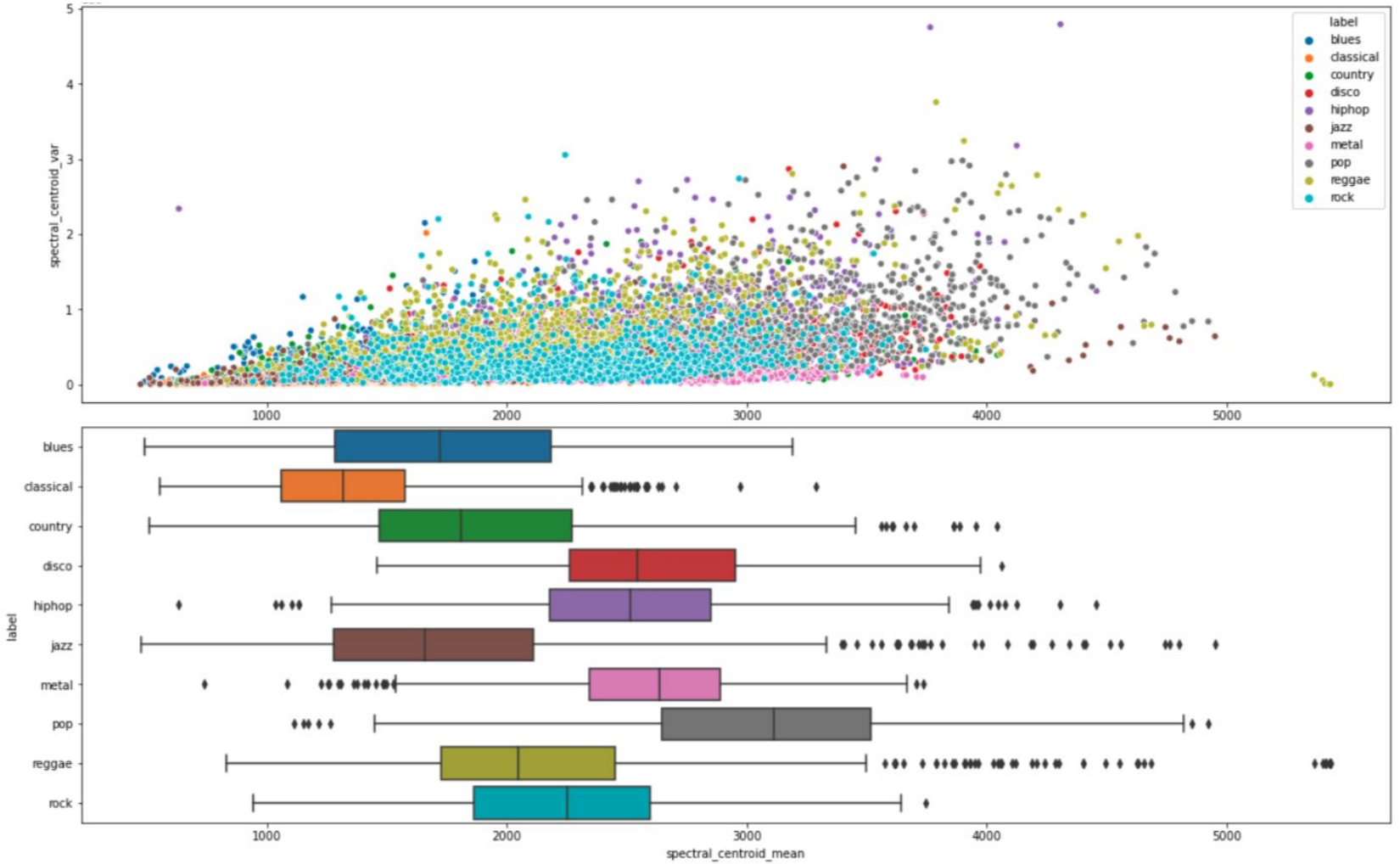
286 rows × 60 columns

- Resulting dataset shape: 9847 rows x 60 cols (after dropping duplicates).

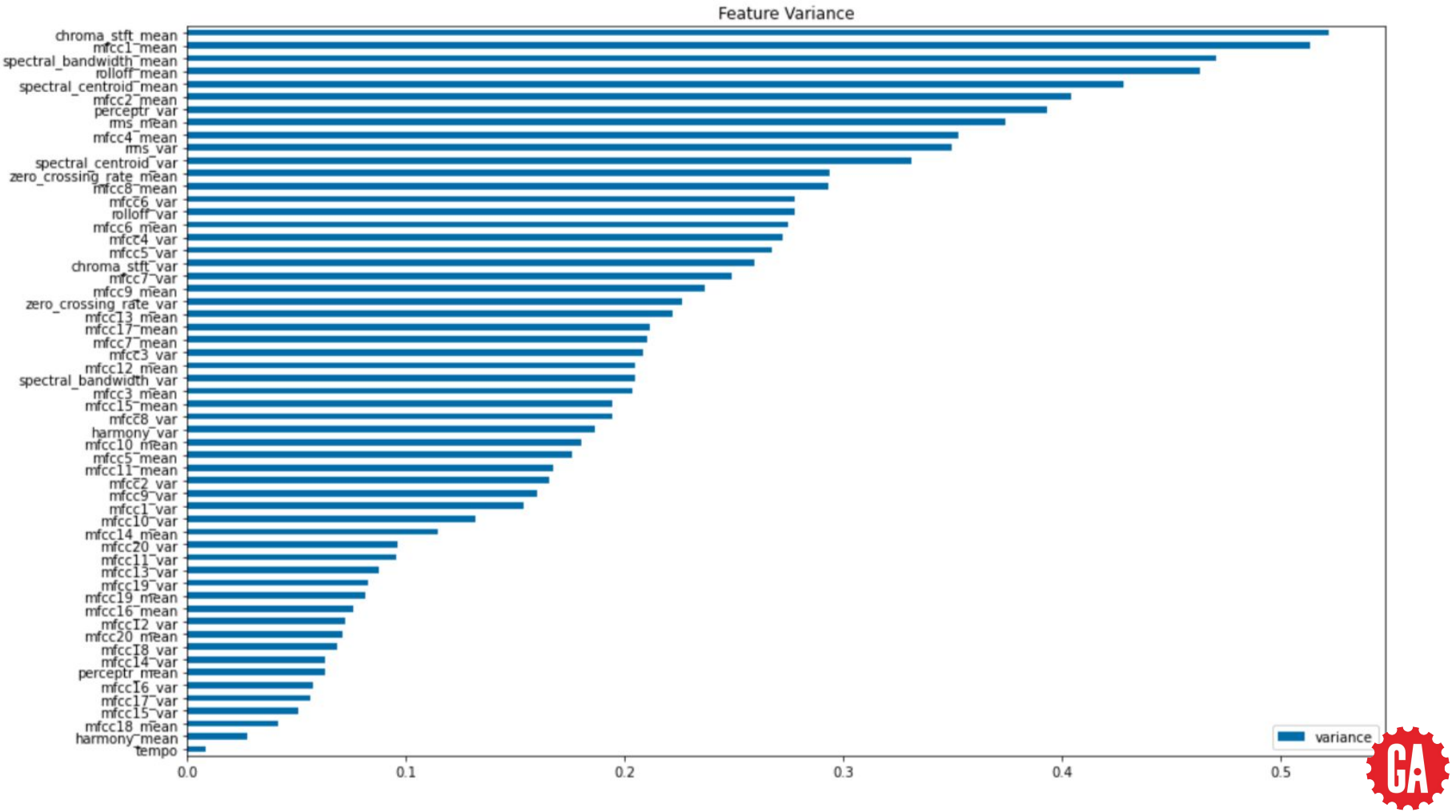

label
blues
classical
country
disco
hiphop
jazz
metal
pop
reggae
rock
chroma_stft_var
0.12
0.10
0.08
0.06
0.04
0.02
0.1
0.2
0.3
0.4
0.5
0.6
0.7
label
chroma_stft_mean


label
blues
classical
country
disco
hiphop
jazz
metal
pop
reggae
rock
rms_mean
0.0
0.1
0.2
0.3
0.4
−700
−600
−500
−400
−300
−200
−100
0
100
label
mfcc1_mean


label
blues
classical
country
disco
hiphop
jazz
metal
pop
reggae
rock
10^{-1}
10^{-2}
10^{-3}
rms_mean
−700
−600
−500
−400
−300
−200
−100
0
100
mfcc1_mean


spectral_centroid_var
label
blues
classical
country
disco
hiphop
jazz
metal
pop
reggae
rock
spectral_centroid_mean

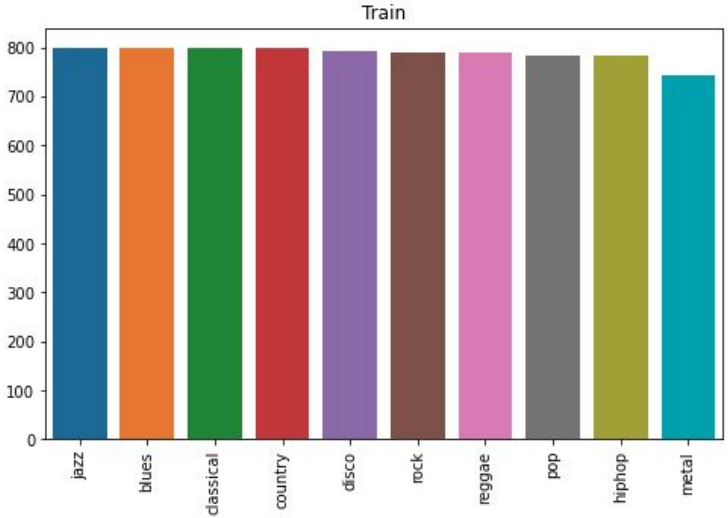

label
blues
classical
country
disco
hiphop
jazz
metal
pop
reggae
rock
10^6
10^5
10^4
10^3
spectral_centroid_var
10^3
spectral_centroid_mean

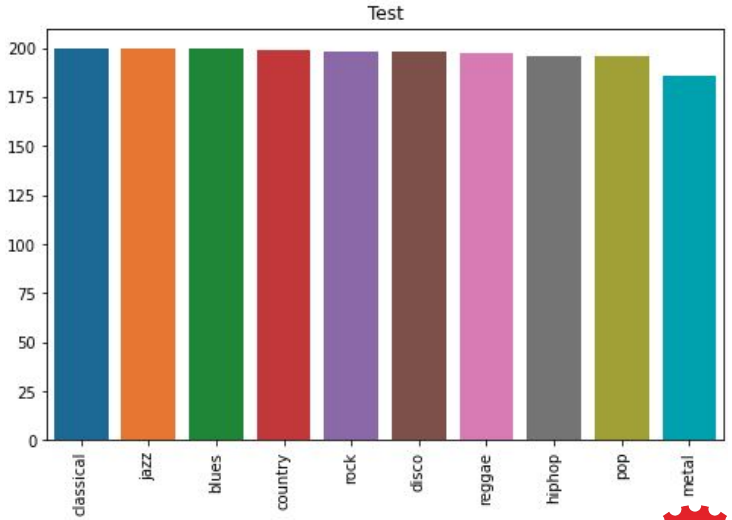

Feature Variance
chroma_stft_mean
mfcc1_mean
spectral_bandwidth_mean
rolloff_mean
spectral_centroid_mean
mfcc2_mean
perceptr_var
rms_mean
mfcc4_mean
rms_var
spectral_centroid_var
zero_crossing_rate_mean
mfcc8_mean
mfcc6_var
rolloff_var
mfcc6_mean
mfcc4_var
mfcc5_var
chroma_stft_var
mfcc7_var
mfcc9_mean
zero_crossing_rate_var
mfcc13_mean
mfcc17_mean
mfcc7_mean
mfcc3_var
mfcc12_mean
spectral_bandwidth_var
mfcc3_mean
mfcc15_mean
mfcc8_var
harmony_var
mfcc10_mean
mfcc5_mean
mfcc11_mean
mfcc2_var
mfcc9_var
mfcc1_var
mfcc10_var
mfcc14_mean
mfcc20_var
mfcc11_var
mfcc13_var
mfcc19_var
mfcc19_mean
mfcc16_mean
mfcc12_var
mfcc20_mean
mfcc18_var
mfcc14_var
perceptr_mean
mfcc16_var
mfcc17_var
mfcc15_var
mfcc18_mean
harmony_mean
tempo
0.0
0.1
0.2
0.3
0.4
0.5
variance

1. Problem Statement
2. EDA & Preprocessing
3. Audio Features
4. Spectrograms
5. Conclusions & Future Expansion

- Dataset was prepared for modelling:
  - Set target variable and features.
  - Train test split: 0.8 training set, 0.2 test set, while stratifying on the genre to ensure a balanced split between train and test.
  - Standardization using StandardScaler: fit_transform on training set and transform on test set.
  - Calculate baseline: ~0.101523

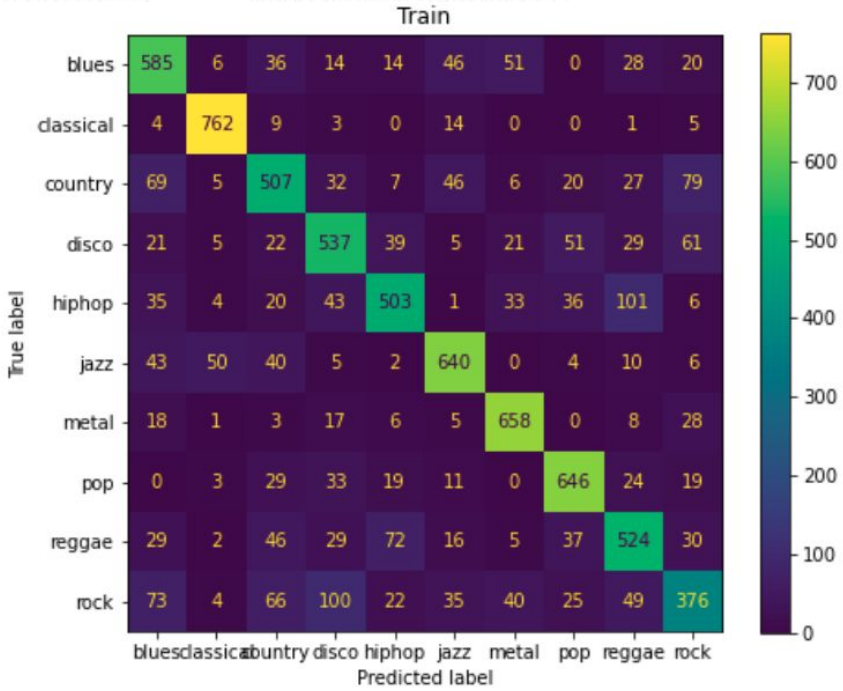
```
Baseline:

classical    0.101523
jazz         0.101523
blues        0.101523
country      0.101015
rock         0.100508
disco        0.100508
reggae       0.100000
hiphop       0.099492
pop          0.099492
metal        0.094416
```

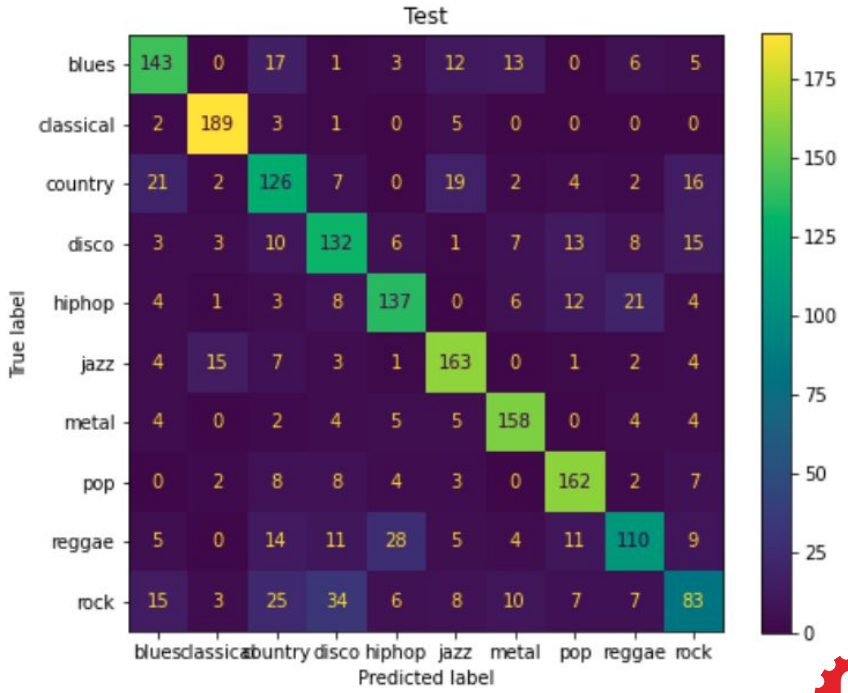




- First model fit to find a benchmark accuracy score: logistic regression with no regularisation.
  - Cross-validated mean: ~0.71

```
LogReg (no regularisation)

Train score:      0.7284499174812746
CV mean:          0.7064845701393925
Test score:       0.7121827411167513
```





- Classification reports for logistic regression model.
  - Model is scoring quite highly for classical already.
  - Room for improvement for other genres (eg country, reggae, rock).

Classification report: train set

	precision	recall	f1-score	support
blues	0.6670	0.7312	0.6977	800
classical	0.9050	0.9549	0.9293	798
country	0.6517	0.6353	0.6434	798
disco	0.6605	0.6789	0.6696	791
hiphop	0.7354	0.6432	0.6862	782
jazz	0.7814	0.8000	0.7906	800
metal	0.8084	0.8844	0.8447	744
pop	0.7888	0.8240	0.8060	784
reggae	0.6542	0.6633	0.6587	790
rock	0.5968	0.4759	0.5296	790
accuracy			0.7284	7877
macro avg	0.7249	0.7291	0.7256	7877
weighted avg	0.7245	0.7284	0.7250	7877

Classification report: test set

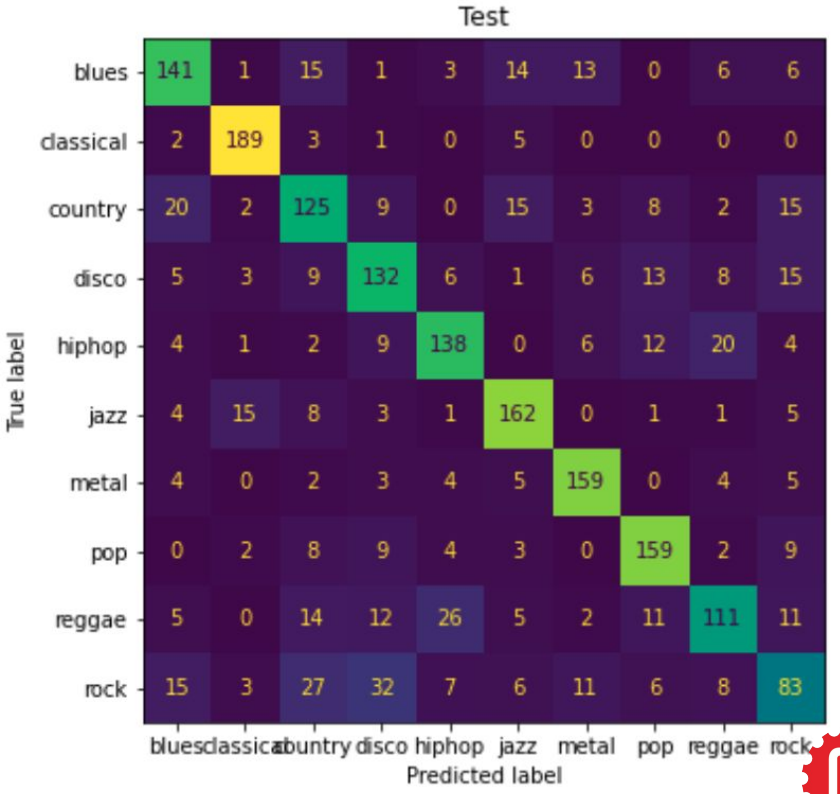
	precision	recall	f1-score	support
blues	0.7114	0.7150	0.7132	200
classical	0.8791	0.9450	0.9108	200
country	0.5860	0.6332	0.6087	199
disco	0.6316	0.6667	0.6486	198
hiphop	0.7211	0.6990	0.7098	196
jazz	0.7376	0.8150	0.7743	200
metal	0.7900	0.8495	0.8187	186
pop	0.7714	0.8265	0.7980	196
reggae	0.6790	0.5584	0.6128	197
rock	0.5646	0.4192	0.4812	198
accuracy			0.7122	1970
macro avg	0.7072	0.7127	0.7076	1970
weighted avg	0.7068	0.7122	0.7071	1970

Model	Parameters	Train Score	Test Score	CV Mean
Logistic Regression	Default	0.7284	0.7122	0.7065
Logistic Regression	Ridge regularisation	0.7304	0.7102	0.7091
Logistic Regression	Lasso regularisation	0.7309	0.7152	0.7084

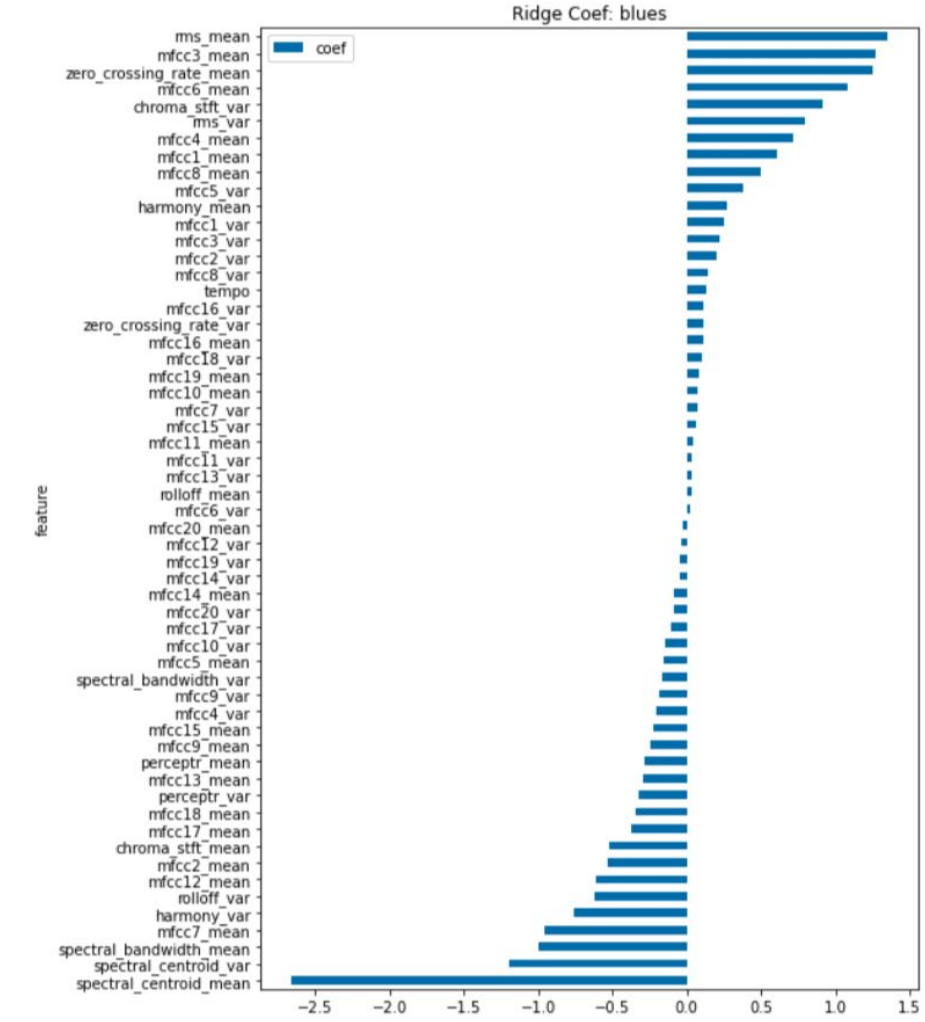
LogReg (Ridge)

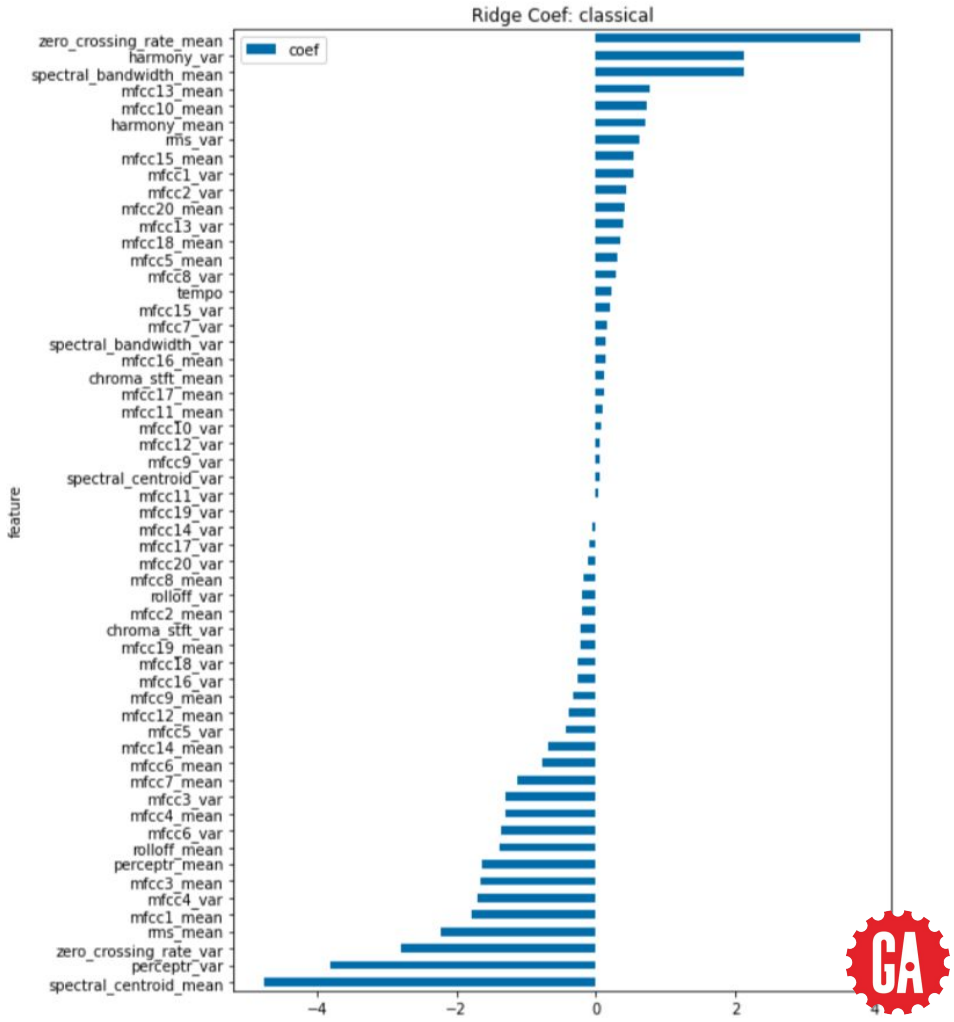
Classification report: test set

	precision	recall	f1-score	support
blues	0.7050	0.7050	0.7050	200
classical	0.8750	0.9450	0.9087	200
country	0.5869	0.6281	0.6068	199
disco	0.6256	0.6667	0.6455	198
hiphop	0.7302	0.7041	0.7169	196
jazz	0.7500	0.8100	0.7788	200
metal	0.7950	0.8548	0.8238	186
pop	0.7571	0.8112	0.7833	196
reggae	0.6852	0.5635	0.6184	197
rock	0.5425	0.4192	0.4729	198
accuracy			0.7102	1970
macro avg	0.7052	0.7108	0.7060	1970
weighted avg	0.7048	0.7102	0.7055	1970



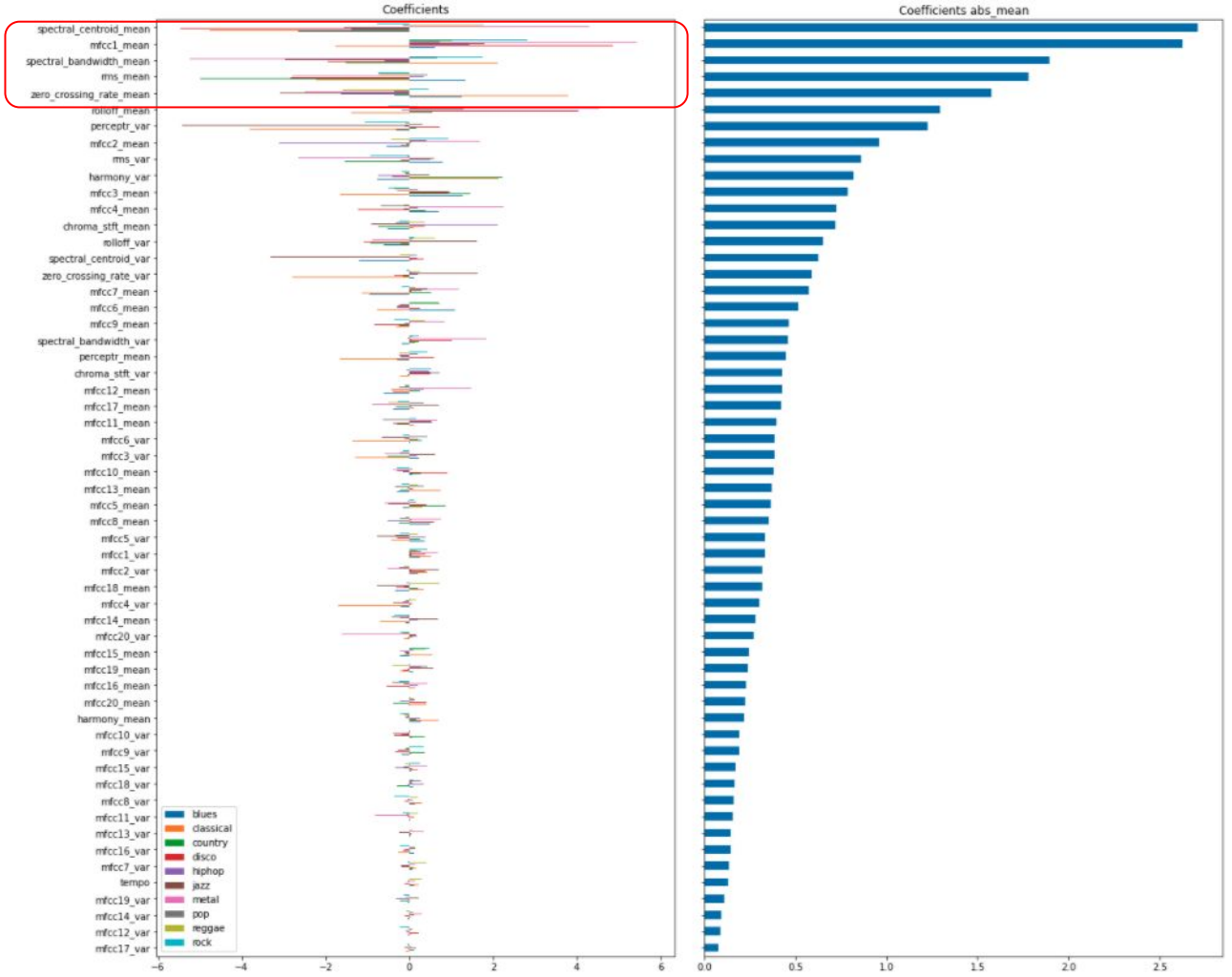
LogReg (Ridge): *model coefficients for two example classes (blues and classical)*



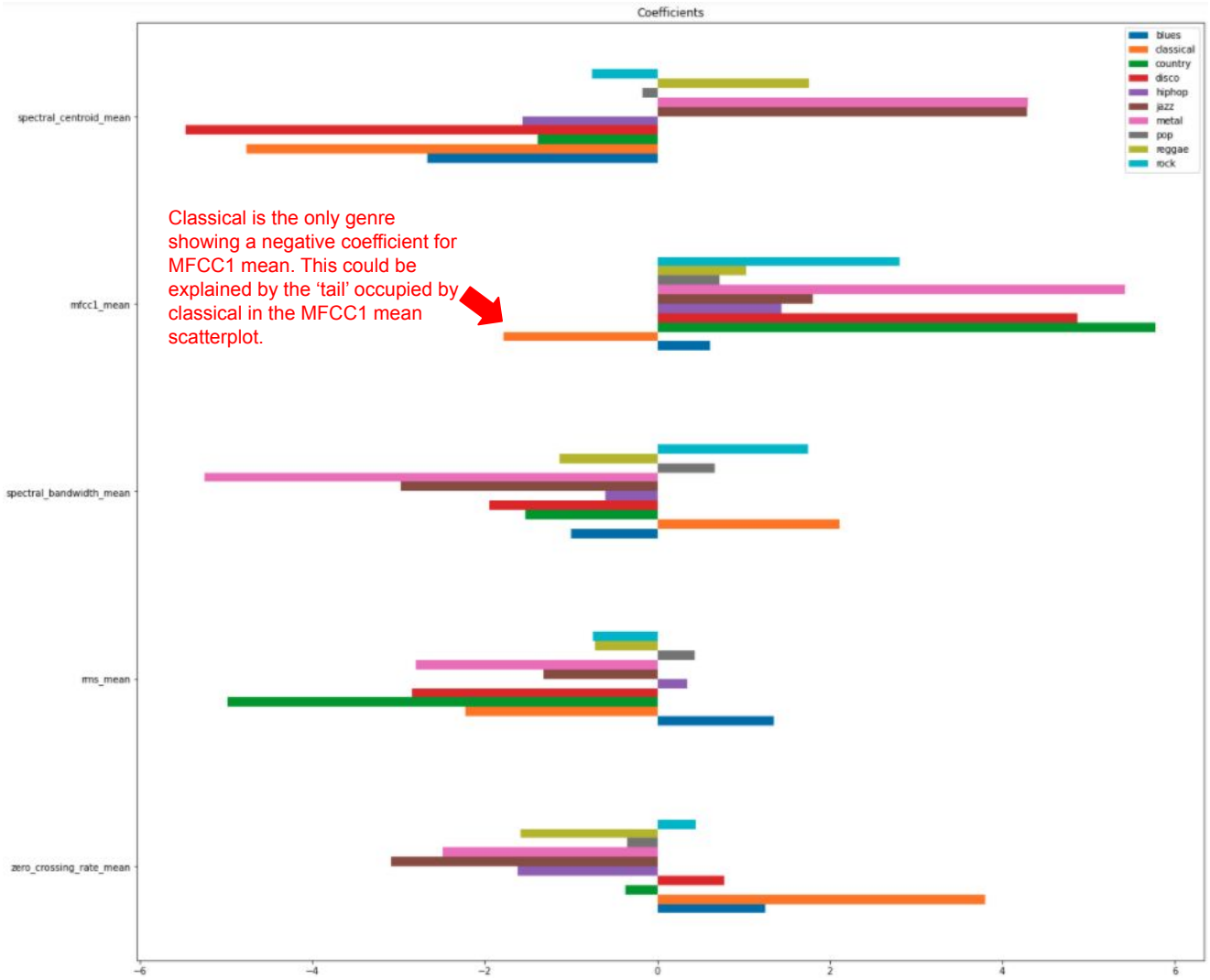


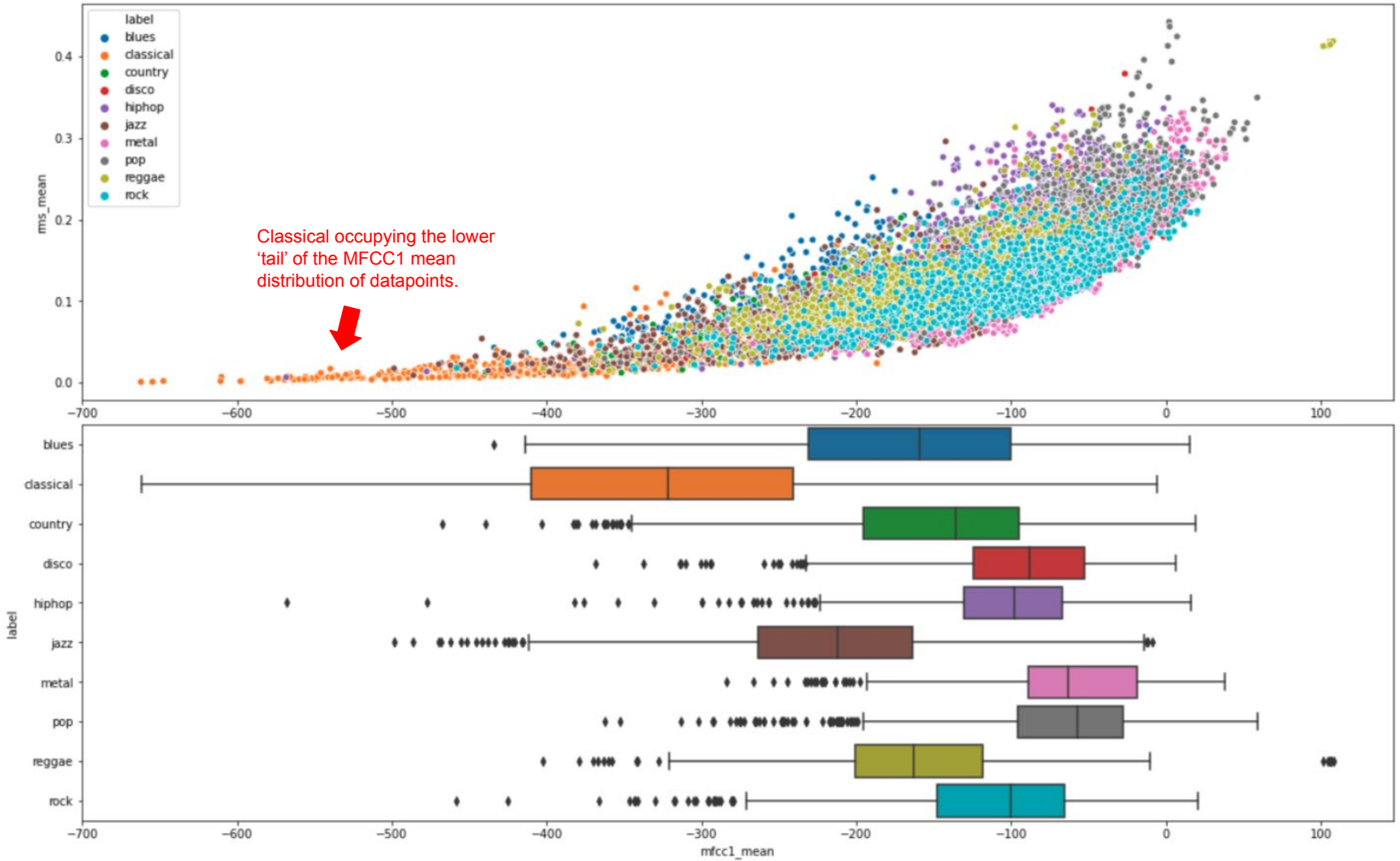
LogReg (Ridge): *features ordered by the mean of absolute coefficient values (accounting for pos./neg. depending on model class)*

Top five shown in next slide

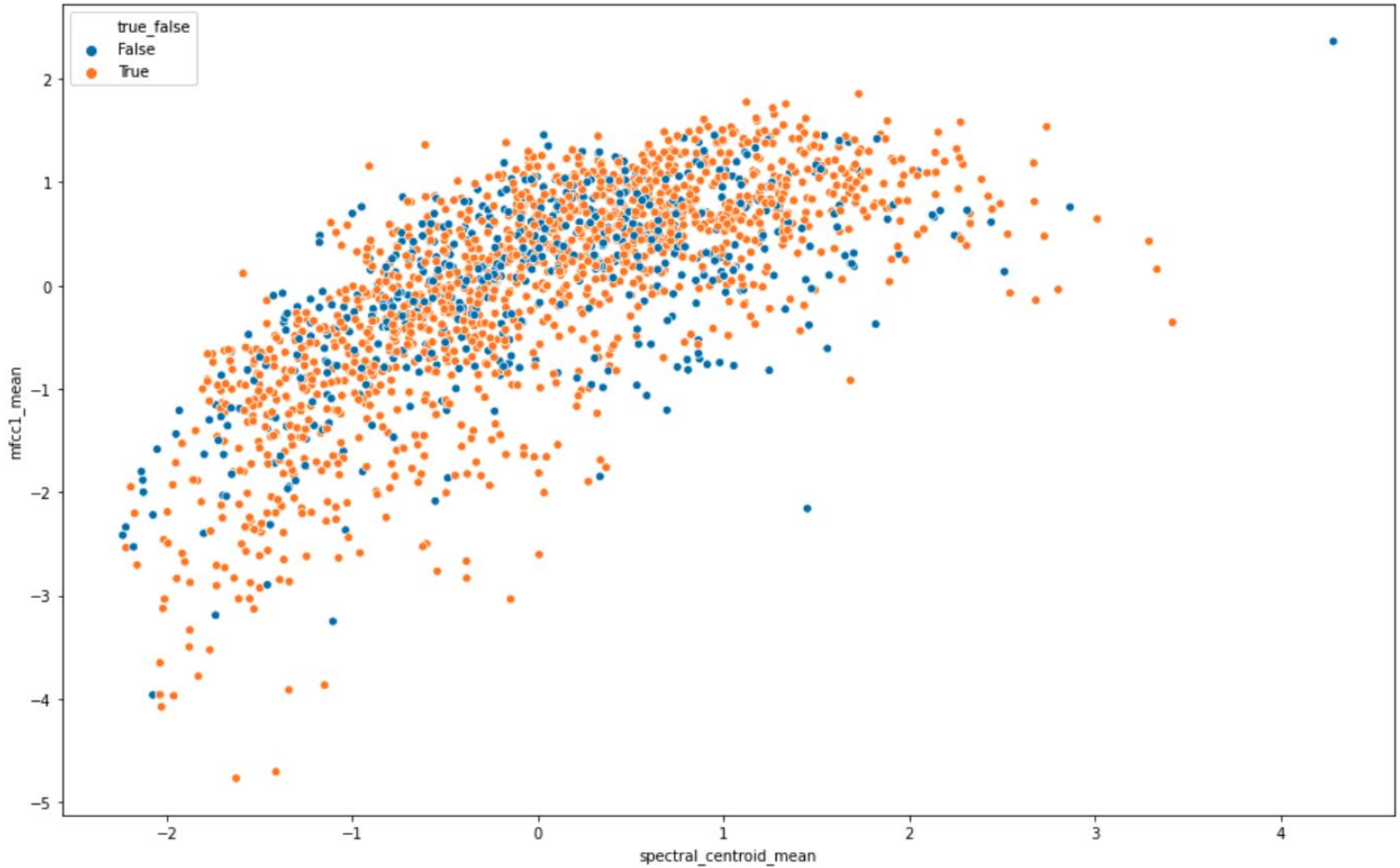


LogReg (Ridge)




label
blues
classical
country
disco
hiphop
jazz
metal
pop
reggae
rock
rms_mean
Classical occupying the lower 'tail' of the MFCC1 mean distribution of datapoints.
label
mfcc1_mean

- Visual representation of predicted values, on example feature plane.
  - correctly predicted & incorrectly predicted

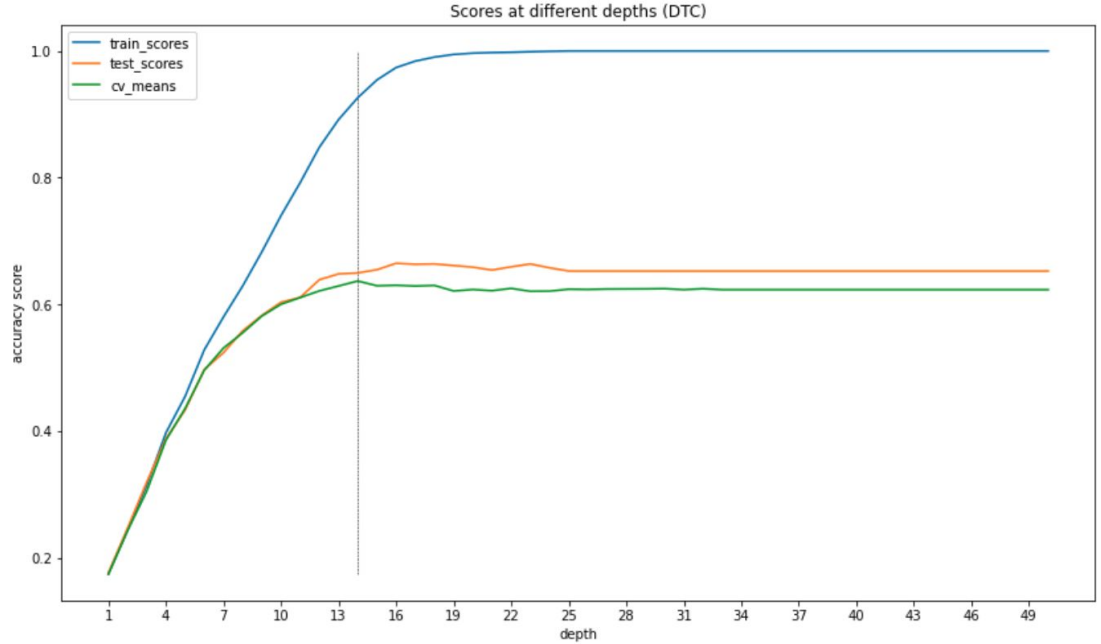


DecisionTreeClassifier / RandomForestClassifier

Model	Parameters	Train Score	Test Score	CV Mean
DTC	Default	1.0000	0.6609	0.6227
DTC (Grid)	{'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 14, 'splitter': 'best'}	0.9839	0.6558	0.6521
DTC (Bagging)		1.0000	0.8345	0.8338
RFC	{'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 29, 'n_estimators': 1500}	1.0000	0.8746	0.8666

DTC: Tree depth

```
CV means max: 0.6370443961002337
Depth: 14
```



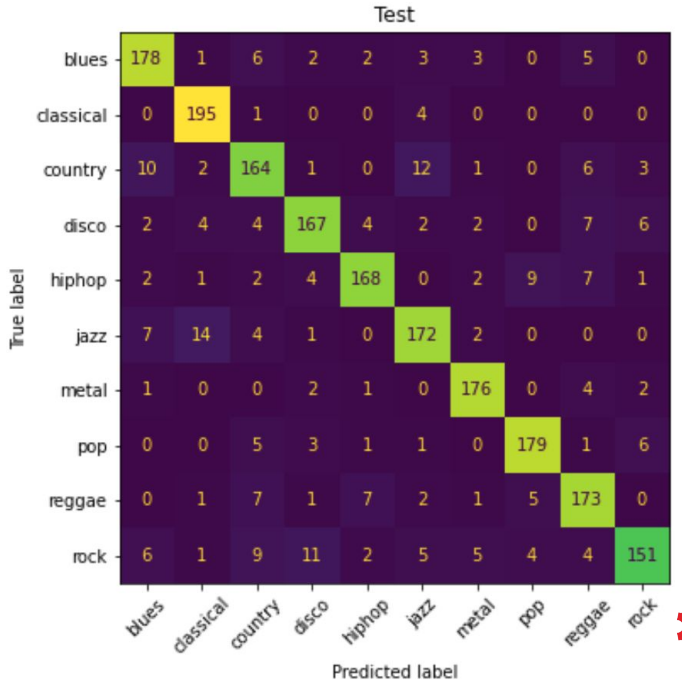
DTC / RFC

Model	Parameters	Train Score	Test Score	CV Mean
DTC	Default	1.0000	0.6609	0.6227
DTC (Grid)	{'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 14, 'splitter': 'best'}	0.9839	0.6558	0.6521
DTC (Bagging)		1.0000	0.8345	0.8338
RFC	{'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 29, 'n_estimators': 1500}	1.0000	0.8746	0.8666

RFC:

Classification report: test set

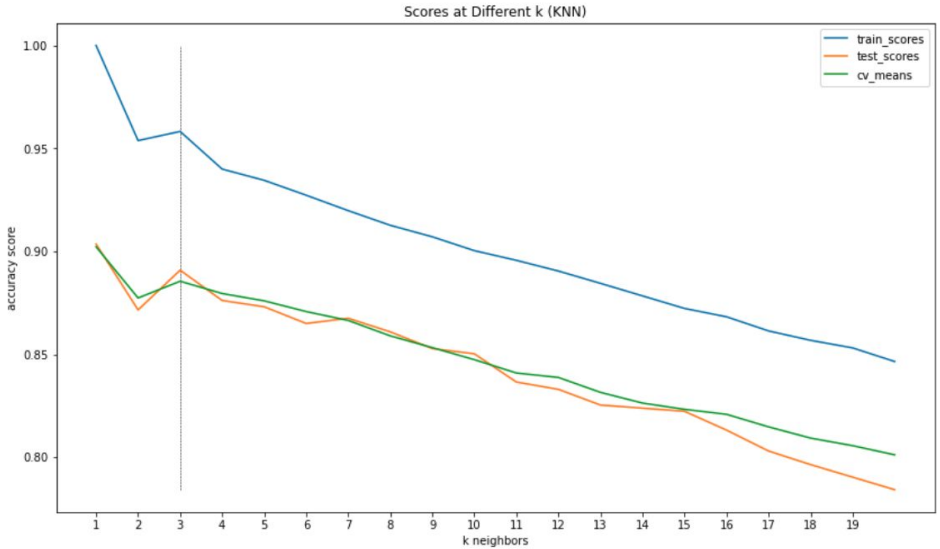
	precision	recall	f1-score	support
blues	0.8641	0.8900	0.8768	200
classical	0.8904	0.9750	0.9308	200
country	0.8119	0.8241	0.8180	199
disco	0.8698	0.8434	0.8564	198
hiphop	0.9081	0.8571	0.8819	196
jazz	0.8557	0.8600	0.8579	200
metal	0.9167	0.9462	0.9312	186
pop	0.9086	0.9133	0.9109	196
reggae	0.8357	0.8782	0.8564	197
rock	0.8935	0.7626	0.8229	198
accuracy			0.8746	1970
macro avg	0.8755	0.8750	0.8743	1970
weighted avg	0.8751	0.8746	0.8740	1970

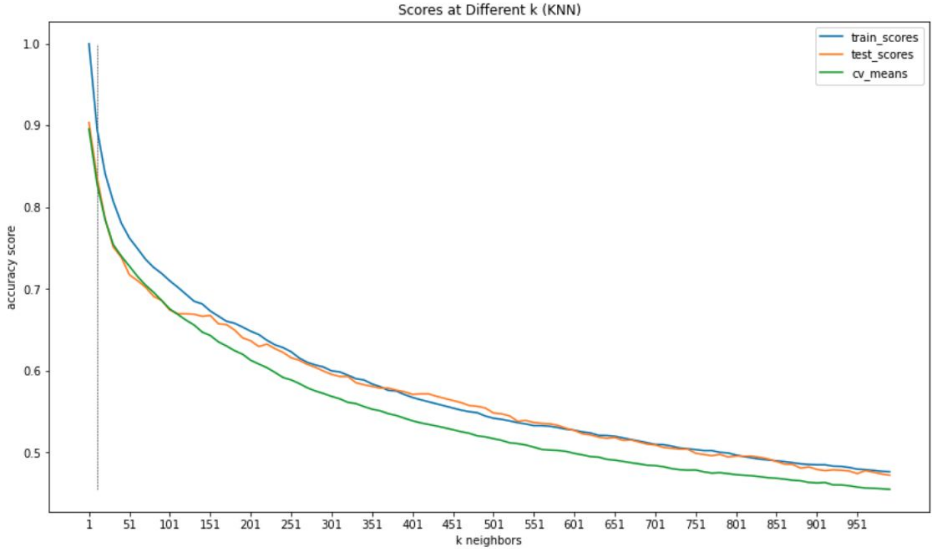


KNN

Model	Parameters	Train Score	Test Score	CV Mean
KNN	n_neighbors: 3	0.9562	0.8854	0.8890
KNN (Grid)	{'algorithm': 'auto', 'n_neighbors': 1, 'weights': 'uniform'}	1.0000	0.9036	0.8955

KNN: k neighbours





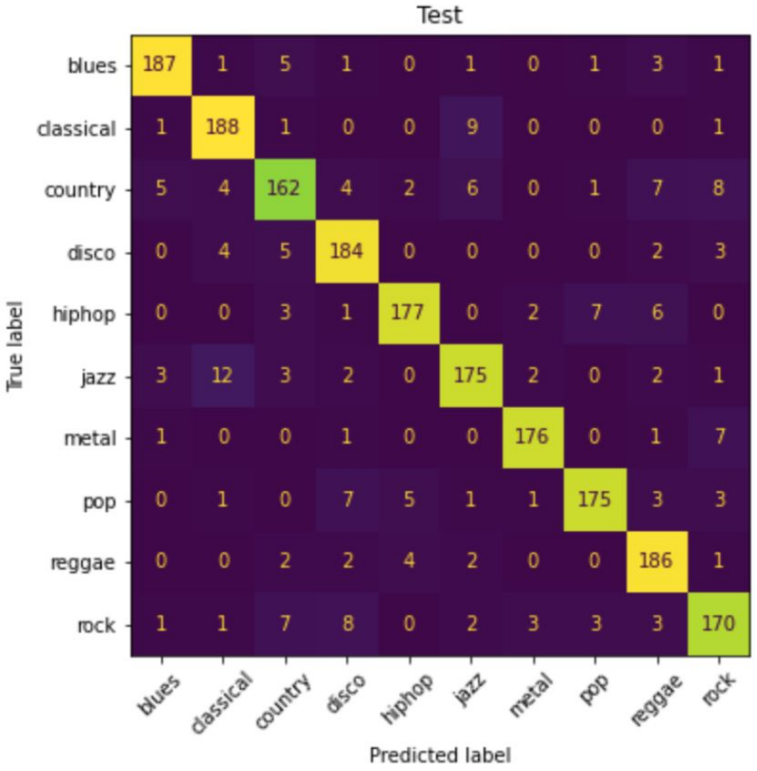
KNN

Model	Parameters	Train Score	Test Score	CV Mean
KNN	n_neighbors: 3	0.9562	0.8854	0.8890
KNN (Grid)	{'algorithm': 'auto', 'n_neighbors': 1, 'weights': 'uniform'}	1.0000	0.9036	0.8955

KNN: 1 neighbour

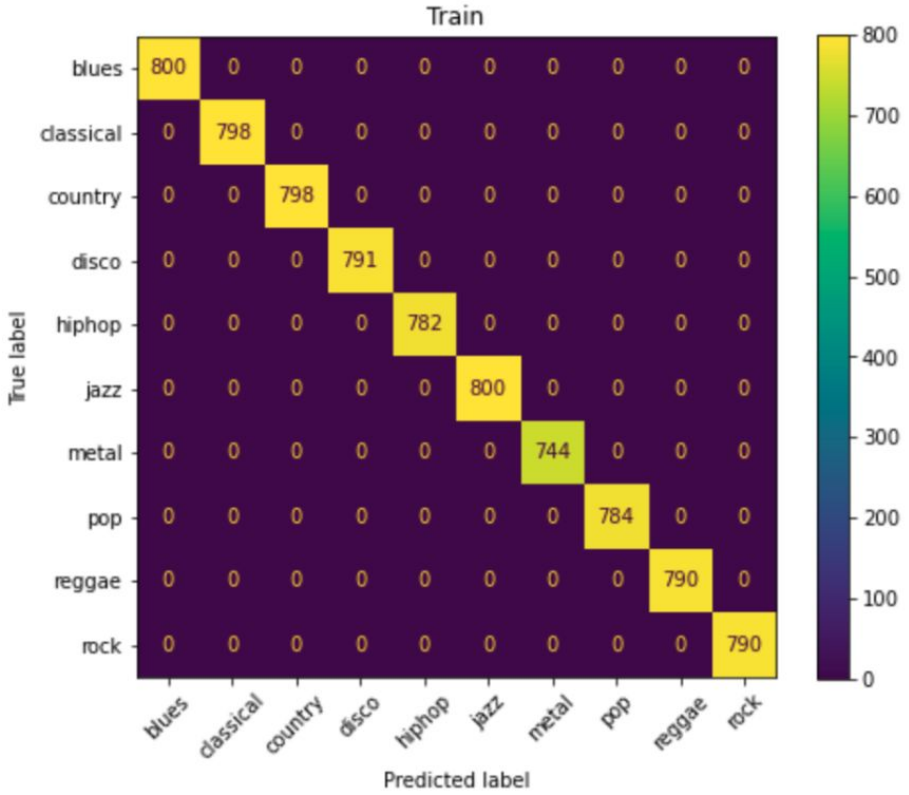
Classification report: test set

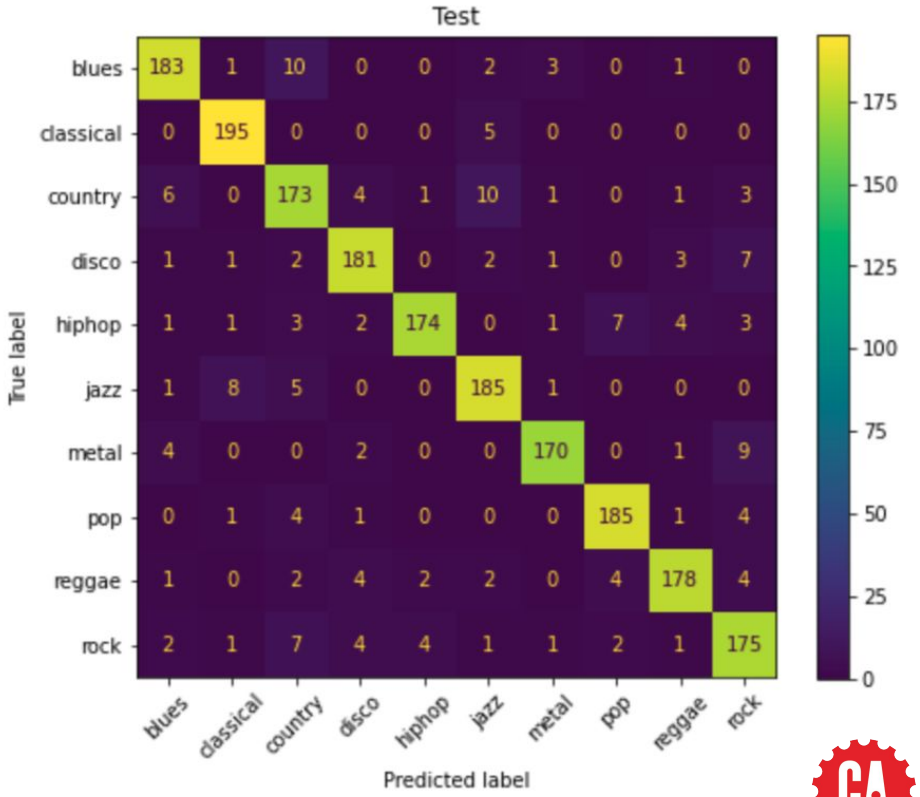
	precision	recall	f1-score	support
blues	0.9444	0.9350	0.9397	200
classical	0.8910	0.9400	0.9148	200
country	0.8617	0.8141	0.8372	199
disco	0.8762	0.9293	0.9020	198
hiphop	0.9415	0.9031	0.9219	196
jazz	0.8929	0.8750	0.8838	200
metal	0.9565	0.9462	0.9514	186
pop	0.9358	0.8929	0.9138	196
reggae	0.8732	0.9442	0.9073	197
rock	0.8718	0.8586	0.8651	198
accuracy			0.9036	1970
macro avg	0.9045	0.9038	0.9037	1970
weighted avg	0.9041	0.9036	0.9034	1970



LightGBM

Model	Parameters	Train Score	Test Score	CV Mean
LightGBM	num_leaves=10, max_depth=10, learning_rate=0.1, n_estimators=1000, class_weight='balanced'	1.0000	0.9132	0.9087





LightGBM

Model	Parameters	Train Score	Test Score	CV Mean
LightGBM	num_leaves=10, max_depth=10, learning_rate=0.1, n_estimators=1000, class_weight='balanced'	1.0000	0.9132	0.9087

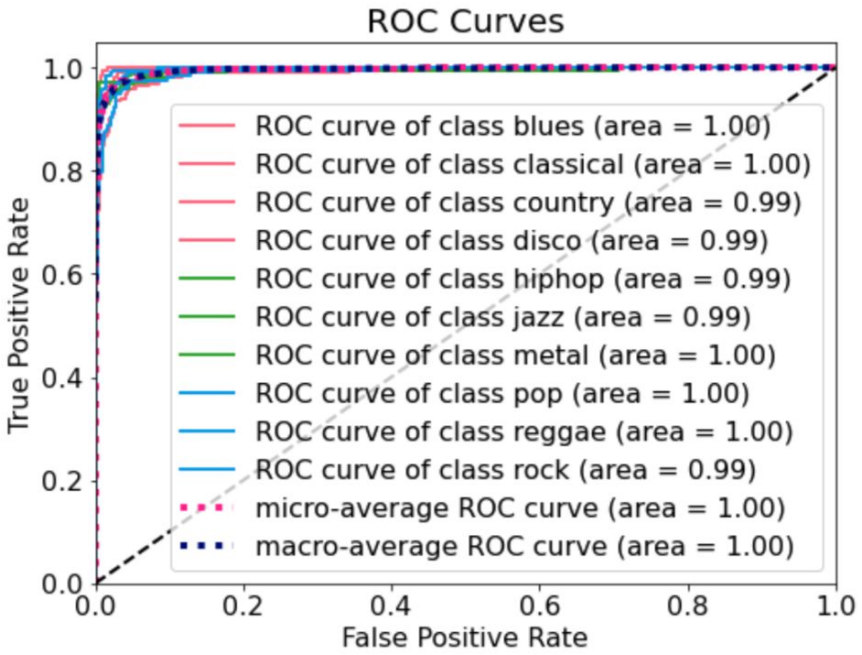
Classification report: train set

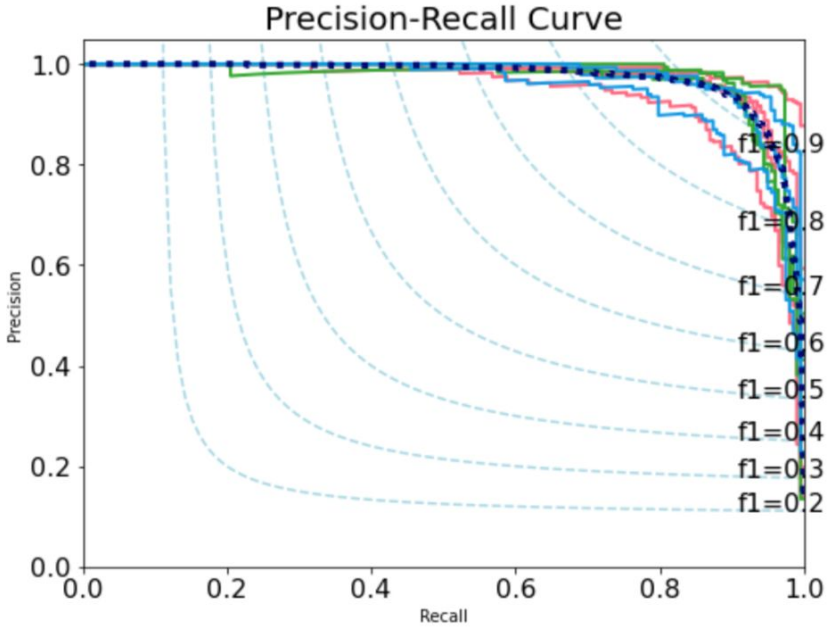
	precision	recall	f1-score	support
blues	1.0000	1.0000	1.0000	800
classical	1.0000	1.0000	1.0000	798
country	1.0000	1.0000	1.0000	798
disco	1.0000	1.0000	1.0000	791
hiphop	1.0000	1.0000	1.0000	782
jazz	1.0000	1.0000	1.0000	800
metal	1.0000	1.0000	1.0000	744
pop	1.0000	1.0000	1.0000	784
reggae	1.0000	1.0000	1.0000	790
rock	1.0000	1.0000	1.0000	790
accuracy			1.0000	7877
macro avg	1.0000	1.0000	1.0000	7877
weighted avg	1.0000	1.0000	1.0000	7877

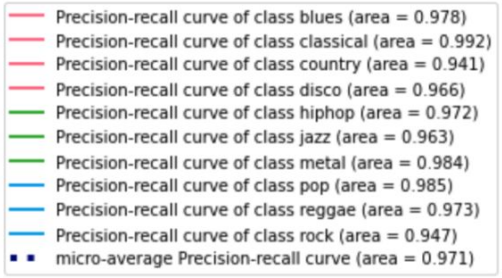
Classification report: test set

	precision	recall	f1-score	support
blues	0.9196	0.9150	0.9173	200
classical	0.9375	0.9750	0.9559	200
country	0.8398	0.8693	0.8543	199
disco	0.9141	0.9141	0.9141	198
hiphop	0.9613	0.8878	0.9231	196
jazz	0.8937	0.9250	0.9091	200
metal	0.9551	0.9140	0.9341	186
pop	0.9343	0.9439	0.9391	196
reggae	0.9368	0.9036	0.9199	197
rock	0.8537	0.8838	0.8685	198
accuracy			0.9132	1970
macro avg	0.9146	0.9131	0.9135	1970
weighted avg	0.9142	0.9132	0.9134	1970

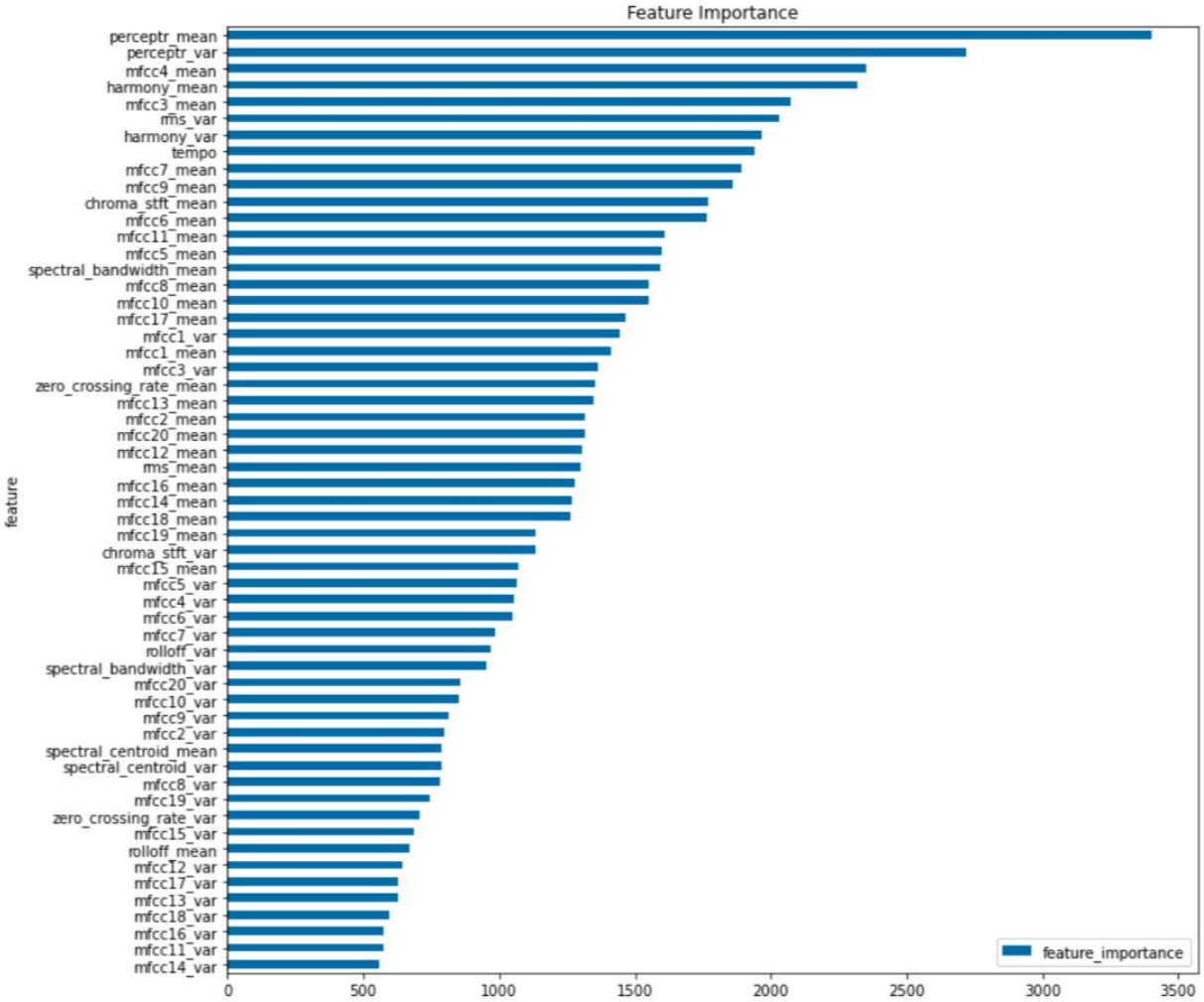
LightGBM


ROC Curves
True Positive Rate
False Positive Rate
ROC curve of class blues (area = 1.00)
ROC curve of class classical (area = 1.00)
ROC curve of class country (area = 0.99)
ROC curve of class disco (area = 0.99)
ROC curve of class hiphop (area = 0.99)
ROC curve of class jazz (area = 0.99)
ROC curve of class metal (area = 1.00)
ROC curve of class pop (area = 1.00)
ROC curve of class reggae (area = 1.00)
ROC curve of class rock (area = 0.99)
micro-average ROC curve (area = 1.00)
macro-average ROC curve (area = 1.00)


Precision-Recall Curve
Precision
Recall
f1=0.9
f1=0.8
f1=0.7
f1=0.6
f1=0.5
f1=0.4
f1=0.3
f1=0.2


Precision-recall curve of class blues (area = 0.978)
Precision-recall curve of class classical (area = 0.992)
Precision-recall curve of class country (area = 0.941)
Precision-recall curve of class disco (area = 0.966)
Precision-recall curve of class hiphop (area = 0.972)
Precision-recall curve of class jazz (area = 0.963)
Precision-recall curve of class metal (area = 0.984)
Precision-recall curve of class pop (area = 0.985)
Precision-recall curve of class reggae (area = 0.973)
Precision-recall curve of class rock (area = 0.947)
micro-average Precision-recall curve (area = 0.971)

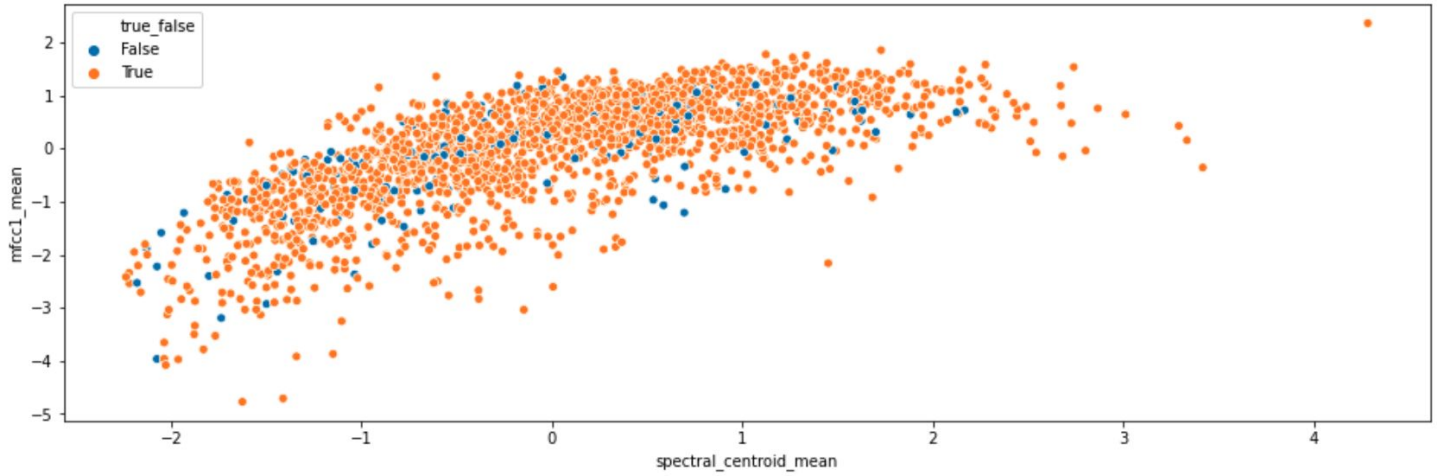
LightGBM


Feature Importance
feature
perceptr_mean
perceptr_var
mfcc4_mean
harmony_mean
mfcc3_mean
rms_var
harmony_var
tempo
mfcc7_mean
mfcc9_mean
chroma_stft_mean
mfcc6_mean
mfcc11_mean
mfcc5_mean
spectral_bandwidth_mean
mfcc8_mean
mfcc10_mean
mfcc17_mean
mfcc1_var
mfcc1_mean
mfcc3_var
zero_crossing_rate_mean
mfcc13_mean
mfcc2_mean
mfcc20_mean
mfcc12_mean
rms_mean
mfcc16_mean
mfcc14_mean
mfcc18_mean
mfcc19_mean
chroma_stft_var
mfcc15_mean
mfcc5_var
mfcc4_var
mfcc6_var
mfcc7_var
rolloff_var
spectral_bandwidth_var
mfcc20_var
mfcc10_var
mfcc9_var
mfcc2_var
spectral_centroid_mean
spectral_centroid_var
mfcc8_var
mfcc19_var
zero_crossing_rate_var
mfcc15_var
rolloff_mean
mfcc12_var
mfcc17_var
mfcc13_var
mfcc18_var
mfcc16_var
mfcc11_var
mfcc14_var
0
500
1000
1500
2000
2500
3000
3500
feature_importance

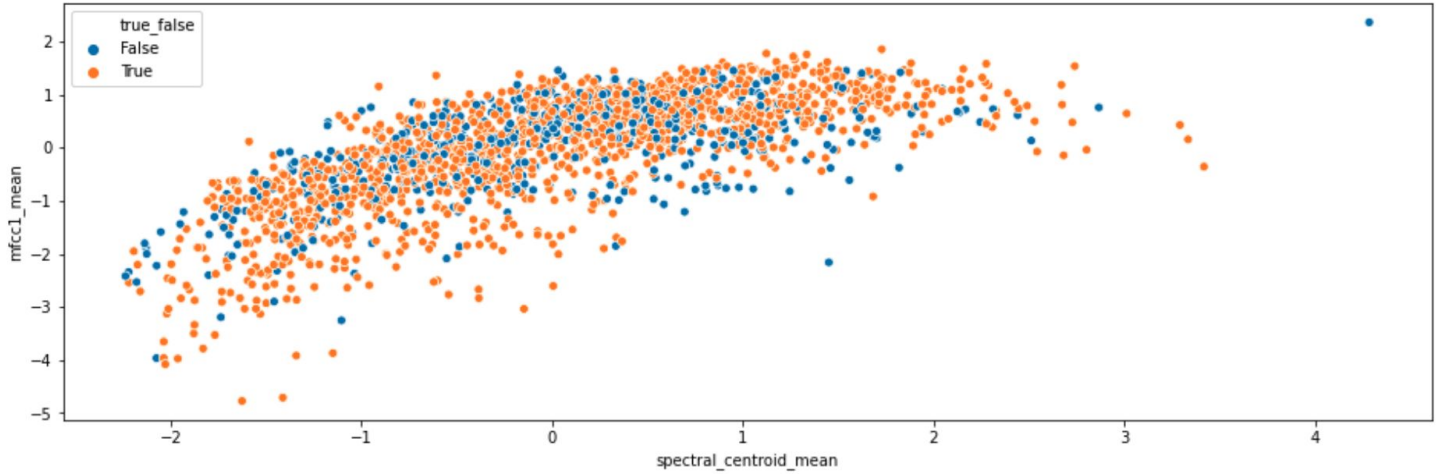
LightGBM

Visual comparison of prediction results between LightGBM and logistic regression, on example feature plane.

LightGBM:



Logistic Regression (Ridge):



1. Problem Statement
2. EDA & Preprocessing
3. Audio Features
4. Spectrograms
5. Conclusions & Future Expansion

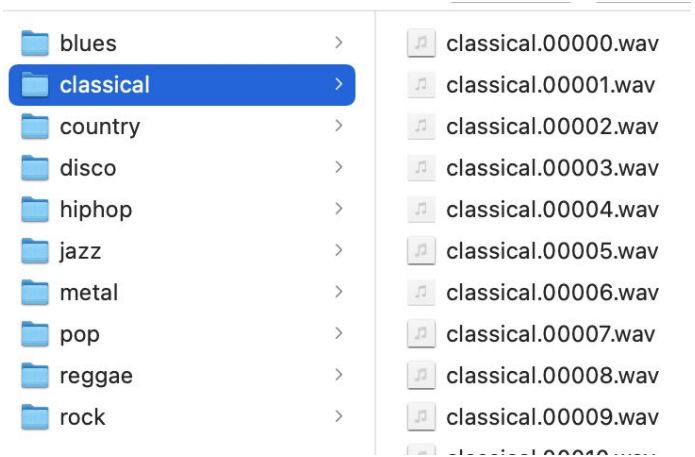
Generating the spectrograms

Additional libraries used: Librosa, Scikit-image

Generating the spectrograms

Additional libraries used: Librosa, Scikit-image

100 WAV files
x 10 genres

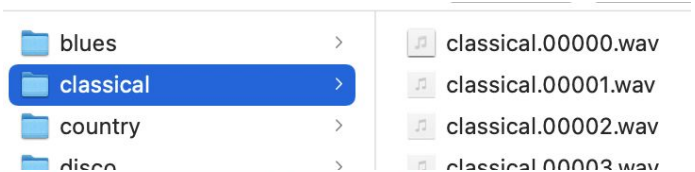


Generating the spectrograms

Additional libraries used: Librosa, Scikit-image

100 WAV files
x 10 genres

Read in WAV files




```
# iterate through wav files, process spectrograms with librosa, save images to another dire
for genre in genres:
    for num in filenums:
        filename_wav = genre+'.'+num

        filepath = f'../gtzan/Data/genres_original/{genre}/{filename_wav}'
#         print(filepath)
        print(f'{genre}{num[:-4]}_spec.png')

        try:
            # loading audio
            y, sr = librosa.load(filepath)

            #Mel Spectogram
            S = librosa.feature.melspectrogram(y, sr=sr, n_mels=128)

            # convert to log scale (dB), using peak power (max) as reference
            log_S = librosa.power_to_db(S, ref=np.max)

            plt.figure(figsize=(12,4))

            # display spectogram on a mel scale
            # sample rate and hop length parameters used to render time axis
            librosa.display.specshow(log_S, sr=sr)#, x_axis='time', y_axis='mel')

            # write to disk
            plt.savefig(f'../gtzan/lib_images/lib_specgrams/{genre}{num[:-4]}_spec.png')

        except:
            continue
```

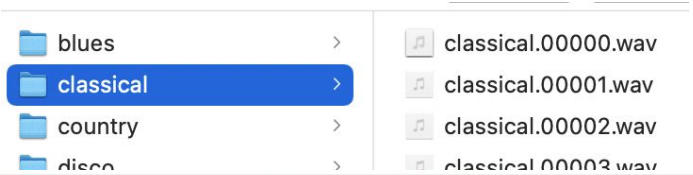
Generating the spectrograms

Additional libraries used: Librosa, Scikit-image

100 WAV files
x 10 genres

Read in WAV files

Generate Mel Freq Spectrograms (Librosa)




```
# iterate through wav files, process spectrograms with librosa, save images to another dire
for genre in genres:
    for num in filenums:
        filename_wav = genre+'.'+num

        filepath = f'../gtzan/Data/genres_original/{genre}/{filename_wav}'
#         print(filepath)
        print(f'{genre}{num[:-4]}_spec.png')

        try:
            # loading audio
            y, sr = librosa.load(filepath)

            #Mel Spectogram
            S = librosa.feature.melspectrogram(y, sr=sr, n_mels=128)

            # convert to log scale (dB), using peak power (max) as reference
            log_S = librosa.power_to_db(S, ref=np.max)

            plt.figure(figsize=(12,4))

            # display spectogram on a mel scale
            # sample rate and hop length parameters used to render time axis
            librosa.display.specshow(log_S, sr=sr)#, x_axis='time', y_axis='mel')

            # write to disk
            plt.savefig(f'../gtzan/lib_images/lib_specgrams/{genre}{num[:-4]}_spec.png')

        except:
            continue
```

Generating the spectrograms

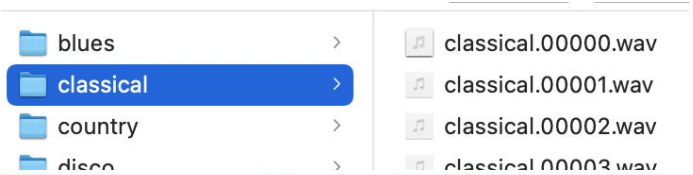
Additional libraries used: Librosa, Scikit-image

100 WAV files
x 10 genres

Read in WAV files

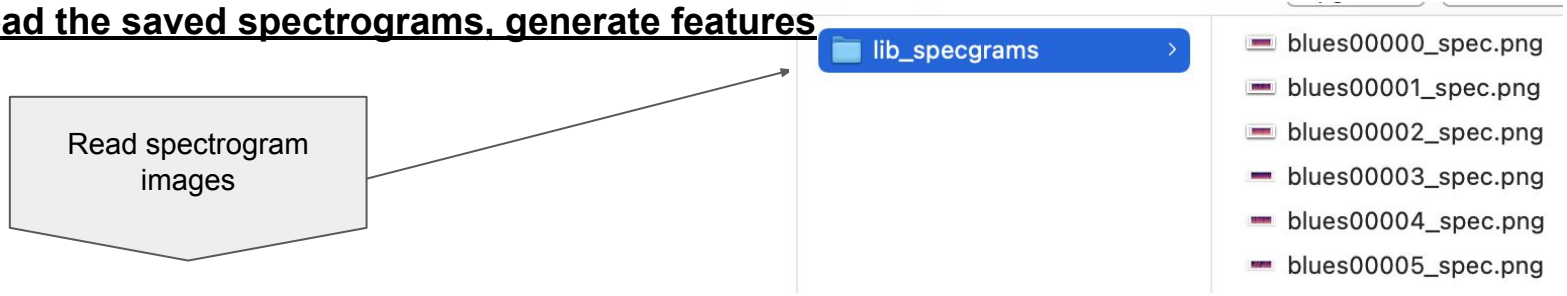
Generate Mel Freq Spectrograms (Librosa)

Save Spectrogram images to disk



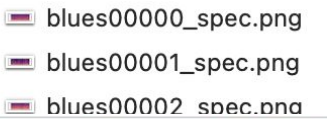

```
# iterate through wav files, process spectrograms with librosa, save images to another dire
for genre in genres:
    for num in filenums:
        filename_wav = genre+'.'+num

        filepath = f'../gtzan/Data/genres_original/{genre}/{filename_wav}'
#        print(filepath)
        print(f'{genre}{num[:-4]}_spec.png')

        try:
            # loading audio
            y, sr = librosa.load(filepath)

            #Mel Spectogram
            S = librosa.feature.melspectrogram(y, sr=sr, n_mels=128)

            # convert to log scale (dB), using peak power (max) as reference
            log_S = librosa.power_to_db(S, ref=np.max)

            plt.figure(figsize=(12,4))

            # display spectogram on a mel scale
            # sample rate and hop length parameters used to render time axis
            librosa.display.specshow(log_S, sr=sr)#, x_axis='time', y_axis='mel')

            # write to disk
            plt.savefig(f'../gtzan/lib_images/lib_specgrams/{genre}{num[:-4]}_spec.png')

        except:
            continue
```

Read the saved spectrograms, generate features



Read the saved spectrograms, generate features

Read spectrogram images




```python
# populate images_dict with filename and accompanying np array, representing the associated image
images_dict = {}

# cycle through spectrograms, add to images_dict
for genre in genres:
    for num in filenums:
        filepath = f'../gtzan/lib_images/lib_specgrams/{genre}{num[:-4]}_spec.png'

        try:
            images_dict[f'{genre}{num[:-4]}'] = io.imread(filepath)
        except:
            continue

# images_dict will have following fomat:
# {'hiphop00000': array([[[255, 255, 255, 255],
#          [255, 255, 255, 255],
#          ...,
#          [255, 255, 255, 255],
#          [255, 255, 255, 255],
#          [255, 255, 255, 255]],
```

Read the saved spectrograms, generate features

Read spectrogram images

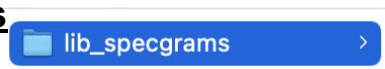
Populate dictionary
{filename genre : 3D array}

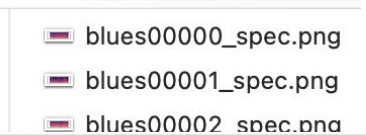

```python
# populate images_dict with filename and accompanying np array, representing the associated image
images_dict = {}

# cycle through spectrograms, add to images_dict
for genre in genres:
    for num in filenums:
        filepath = f'../gtzan/lib_images/lib_specgrams/{genre}{num[:-4]}_spec.png'

        try:
            images_dict[f'{genre}{num[:-4]}'] = io.imread(filepath)
        except:
            continue

# images_dict will have following fomat:
# {'hiphop00000': array([[[255, 255, 255, 255],
#          [255, 255, 255, 255],
#          [255, 255, 255, 255],
#          ...,
#          [255, 255, 255, 255],
#          [255, 255, 255, 255],
#          [255, 255, 255, 255]],
```

Read the saved spectrograms, generate features

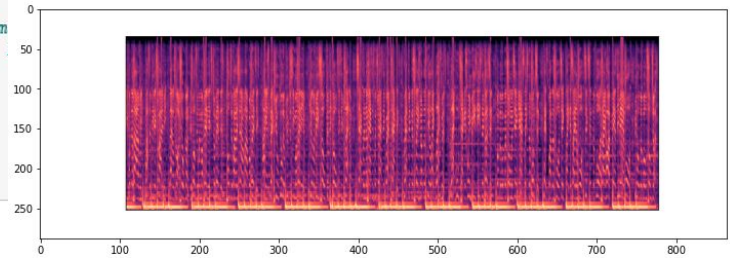
Read spectrogram images

Populate dictionary
{filename genre : 3D array}






```python
# populate images_dict with filename and accompanying np array, representing the associated image
images_dict = {}

# cycle through spectrograms, add to images_dict
for genre in genres:
    for num in filenums:
        filepath = f'../gtzan/lib_images/lib_specgrams/{genre}{num[:-4]}_spec.png'

        try:
            images_dict[f'{genre}{num[:-4]}'] = io.imread(filepath)
        except:
            continue

# images_dict will have following fom
# {'hiphop00000': array([[[255, 255,
#          [255, 255, 255, 255],
#          [255, 255, 255, 255],
#          ...,
#          [255, 255, 255, 255],
#          [255, 255, 255, 255],
#          [255, 255, 255, 255]],
```

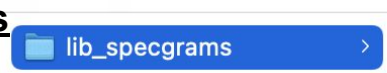


Read the saved spectrograms, generate features

Read spectrogram images

Populate dictionary
{filename genre : 3D array}

Crop white borders




```python
# populate images_dict with filename and accompanying np array, representing the associated image
images_dict = {}

# cycle through spectrograms, add to images_dict
for genre in genres:
    for num in filenums:
        filepath = f'../gtzan/lib_images/lib_specgrams/{genre}{num[:-4]}_spec.png'

        try:
            images_dict[f'{genre}{num[:-4]}'] = io.imread(filepath)
        except:
            continue

# images_dict will have following fom
```

```python
# crop white borders out of each image and "standardize"
cropped_dict = {}

for genre in genres:
    for num in filenums:
        try: # jazz00054 was missing. Could be others too. And "standardize" (divide 255)
            cropped_dict[genre+num[:-4]] = images_dict[genre+num[:-4]][35:253,108:779] / 255
        except:
            continue

# cropped_dict will have following format:
#     {'hiphop00000': array([[[0.01176471, 0.00392157, 0.02745098, 1.        ],
#          [0.        , 0.        , 0.        , 1.        ],
#          [0.        , 0.        , 0.        , 1.        ],
#          ...,
#          [0.        , 0.        , 0.        , 1.        ],
#          [0.        , 0.        , 0.        , 1.        ],
#          [0.0745098 , 0.0745098 , 0.0745098 , 1.        ]],
```

Read the saved spectrograms, generate features

Read spectrogram images

Populate dictionary
{filename genre : 3D array}

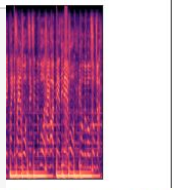
Crop white borders

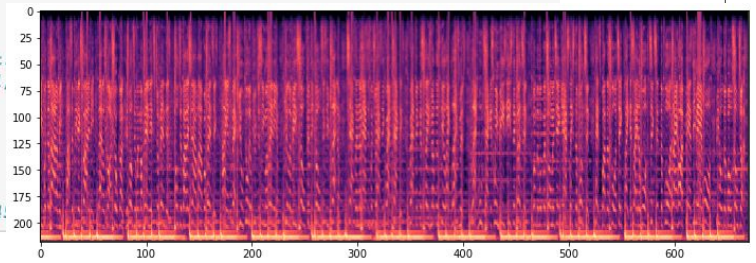
lib_specgrams

blues00000_spec.png
blues00001_spec.png
blues00002_spec.png

```python
# populate images_dict with filename and accompanying np array, representing the associated image
images_dict = {}

# cycle through spectrograms, add to images_dict
for genre in genres:
    for num in filenums:
        filepath = f'../gtzan/lib_images/lib_specgrams/{genre}{num[:-4]}_spec.png'

        try:
            images_dict[f'{genre}{num[:-4]}'] = io.imread(filepath)
        except:
            continue

# images_dict will have following fom
```

```python
# crop white borders out of each image and "standardize"
cropped_dict = {}

for genre in genres:
    for num in filenums:
        try: # jazz00054 was missing. Could be others too. And "standardize" (divide 255)
            cropped_dict[genre+num[:-4]] = images_dict[genre+num[:-4]][35:253,108:779] / 255
        except:
            continue

# cropped_dict will have following format
#     {'hiphop00000': array([[[0.01176471,
#          [0.        , 0.        , 0.
#          [0.        , 0.        , 0.
#          ...,
#          [0.        , 0.        , 0.
#          [0.        , 0.        , 0.
#          [0.0745098 , 0.0745098 , 0.074
```





Read the saved spectrograms, generate features

Flatten 3D array to 1D

```
# flatten all 3D arrays (images) to 1D
flat_dict = {}

for key in cropped_dict:
#     print(key+'_flat')
    flat_dict[key+'_flat'] = cropped_dict[key].ravel() # .ravel(order='F')

df_flat = pd.DataFrame(flat_dict).T

print(df_flat.shape)
display(df_flat.head(3))
display(df_flat.tail(3))
```

Resulting dataframe shape:

- 999 observations.
- 585112 columns: corresponds to number of pixels when flattened.

```
(999, 585112)
```

	0	1	2	3	4	5	6	7	8	9	...	585102	585103	585104	585105	585106	585107	585108	585109	5851
hiphop00000_flat	0.011765	0.003922	0.027451	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.2	1.0	0.2	0.2	0.2	1.0	0.07451	0.07451	0.074
hiphop00001_flat	0.003922	0.003922	0.015686	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.2	1.0	0.2	0.2	0.2	1.0	0.07451	0.07451	0.074
hiphop00002_flat	0.000000	0.000000	0.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.2	1.0	0.2	0.2	0.2	1.0	0.07451	0.07451	0.074

3 rows × 585112 columns

	0	1	2	3	4	5	6	7	8	9	...	585102	585103	585104	585105	585106	585107	585108	585109	585
country00097_flat	0.000000	0.000000	0.007843	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.2	1.0	0.2	0.2	0.2	1.0	0.07451	0.07451	0.07
country00098_flat	0.007843	0.003922	0.023529	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.2	1.0	0.2	0.2	0.2	1.0	0.07451	0.07451	0.07
country00099_flat	0.003922	0.003922	0.019608	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.2	1.0	0.2	0.2	0.2	1.0	0.07451	0.07451	0.07

3 rows × 585112 columns

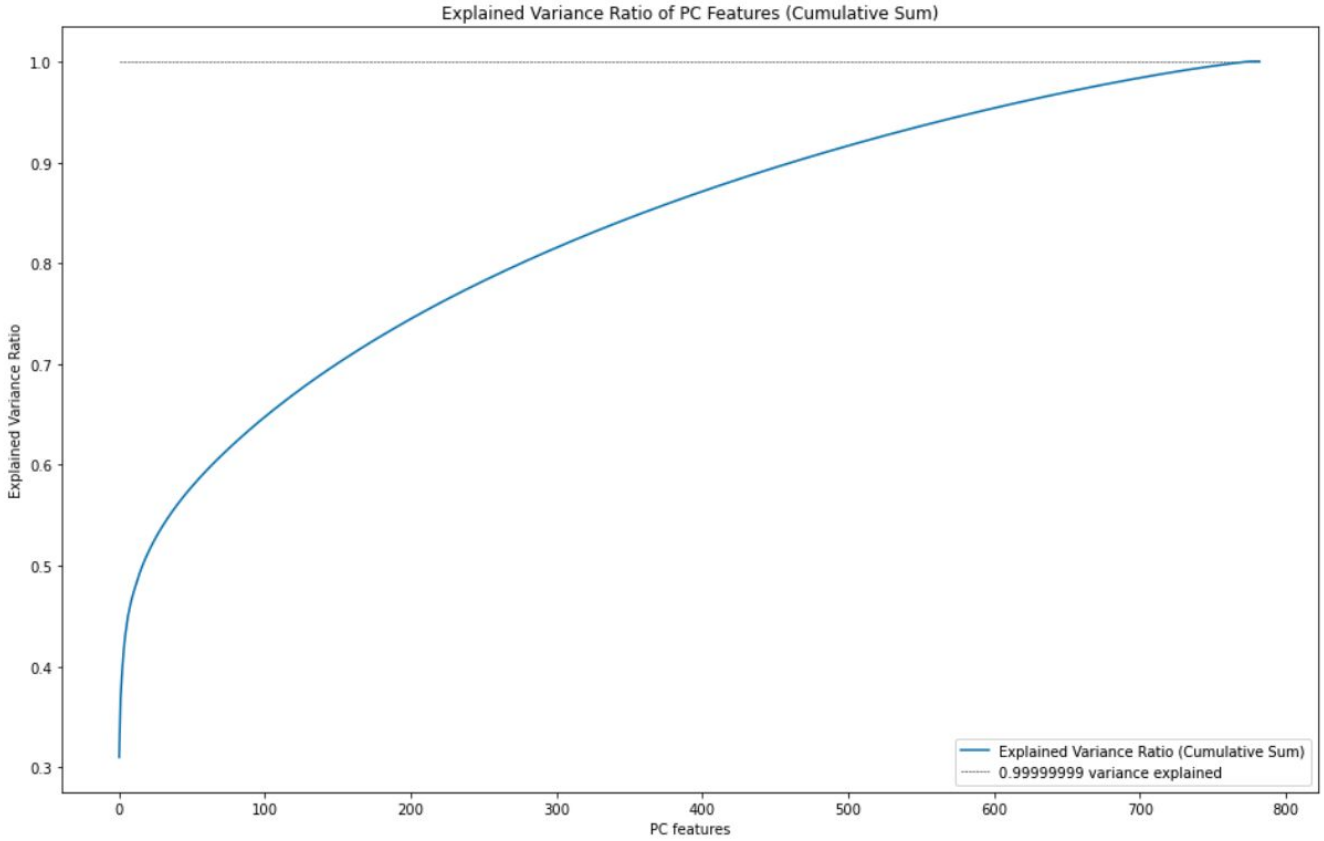
Read the saved spectrograms, generate features

Dimension Reduction (PC)

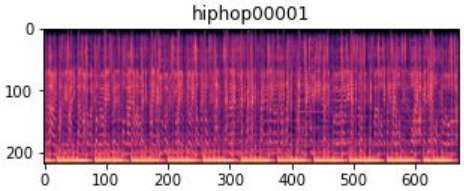
```
X_train_pc shape: (793, 783)
```

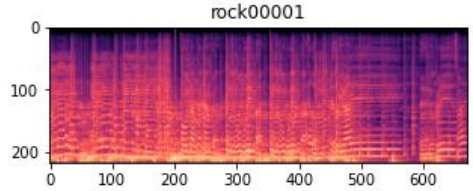
After decomposing using principle component analysis.

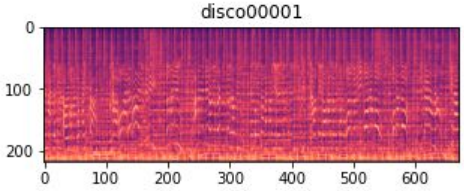
- 793 observations (in train set).
- 783 principal components to maintain nearly all explained variance within dataset.

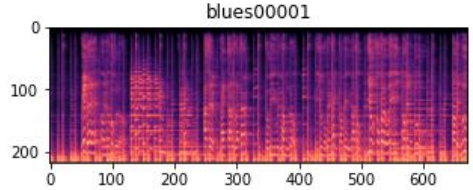


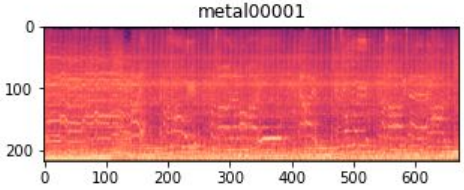
Examples of Spectrograms

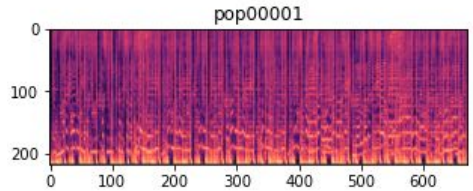


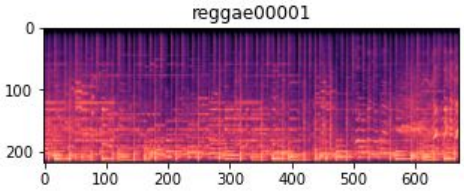


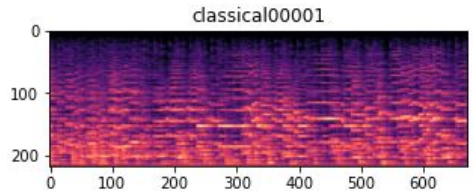


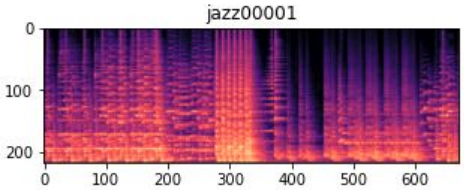


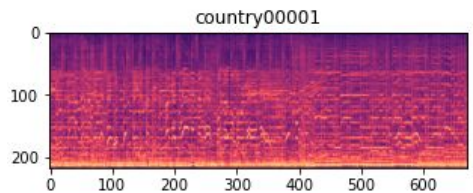












Modelling on Spectrograms:

Model	Parameters	Train Score	Test Score	CV Mean
RFC	{'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 10, 'n_estimators': 500,}	0.9987	0.4800	0.4843

Modelling on Spectrograms:

Model	Parameters	Train Score	Test Score	CV Mean
RFC	{'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 10, 'n_estimators': 500,}	0.9987	0.4800	0.4843

Using a RFC to generate ‘feature’ (pixel) importance of images on MNIST dataset (an example):

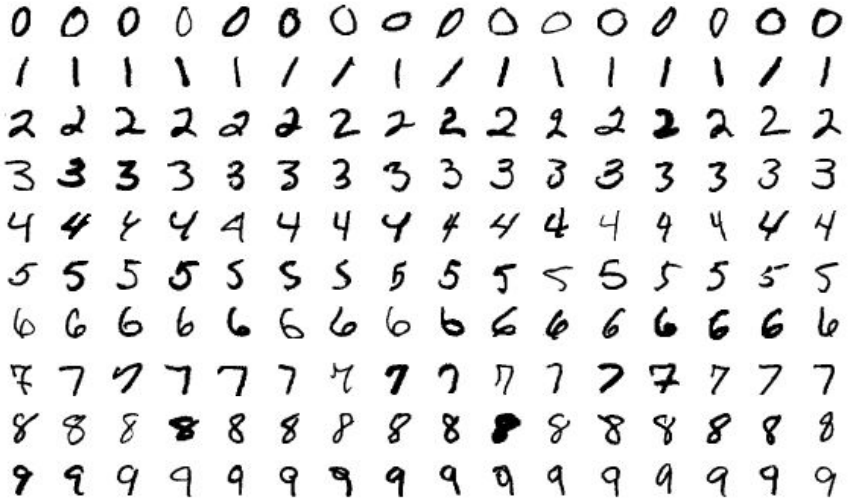
MNIST Dataset

Modelling on Spectrograms:

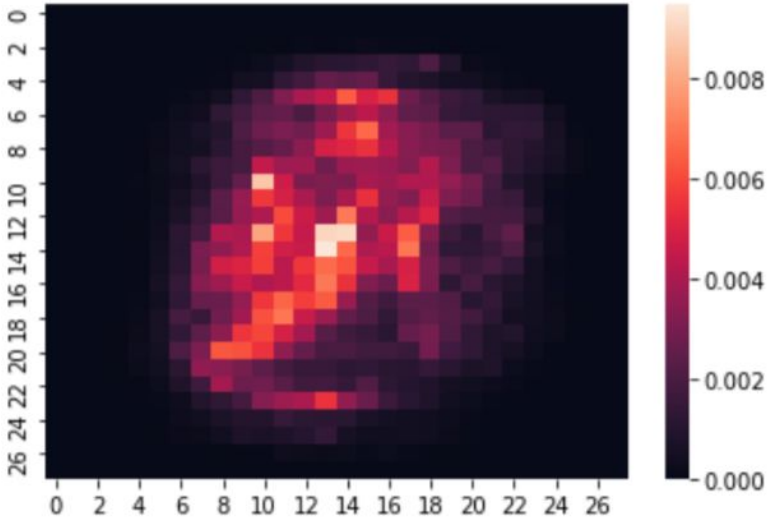
Model	Parameters	Train Score	Test Score	CV Mean
RFC	{'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 10, 'n_estimators': 500,}	0.9987	0.4800	0.4843

Using a RFC to generate ‘feature’ (pixel) importance of images on MNIST dataset (an example):

MNIST Dataset



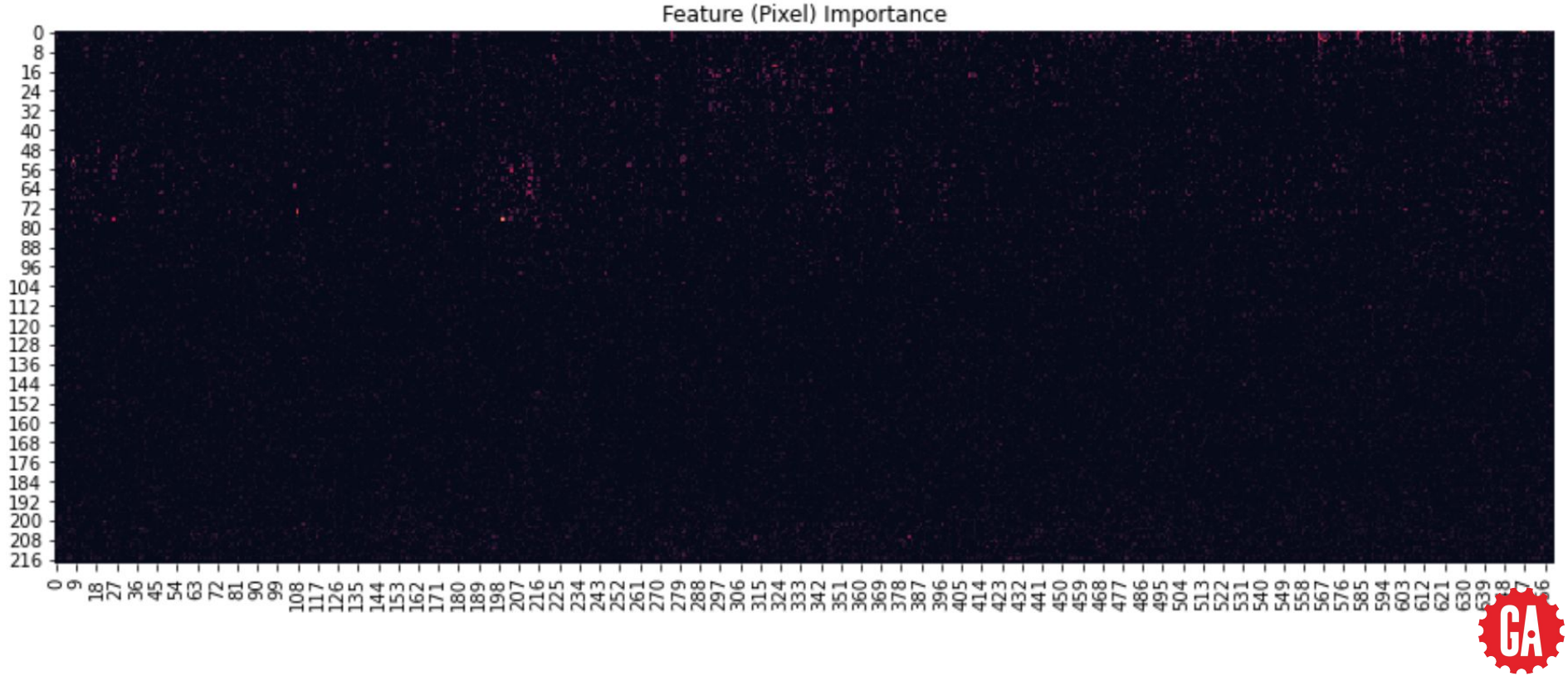
RFC Feature Importance



Modelling on Spectrograms:

Model	Parameters	Train Score	Test Score	CV Mean
RFC	{'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 10, 'n_estimators': 500,}	0.9987	0.4800	0.4843

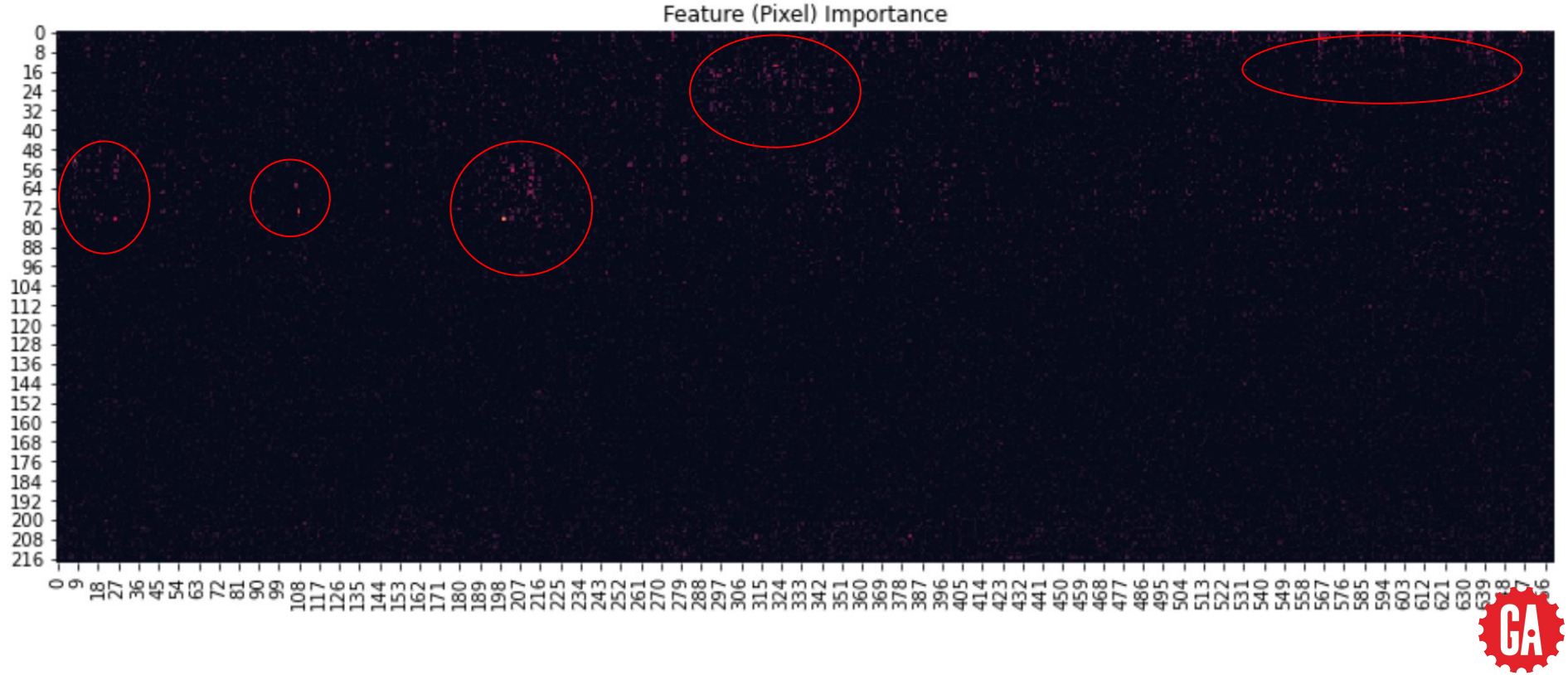
Spectrogram equivalent:



Modelling on Spectrograms:

Model	Parameters	Train Score	Test Score	CV Mean
RFC	{'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 10, 'n_estimators': 500,}	0.9987	0.4800	0.4843

Spectrogram equivalent:

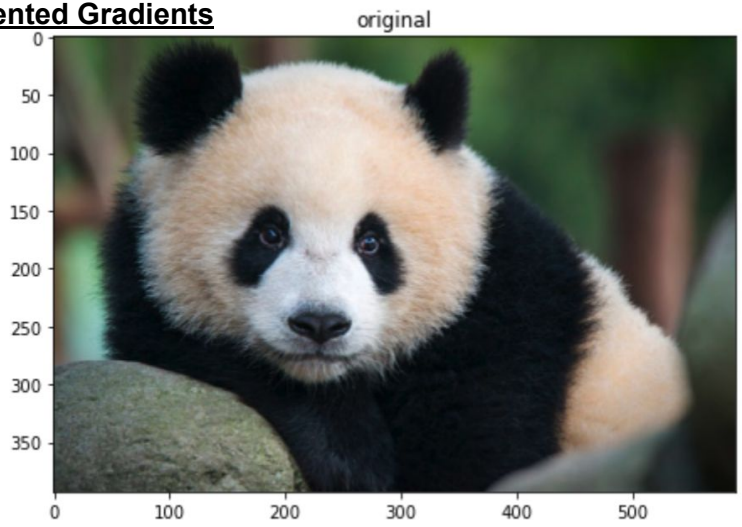


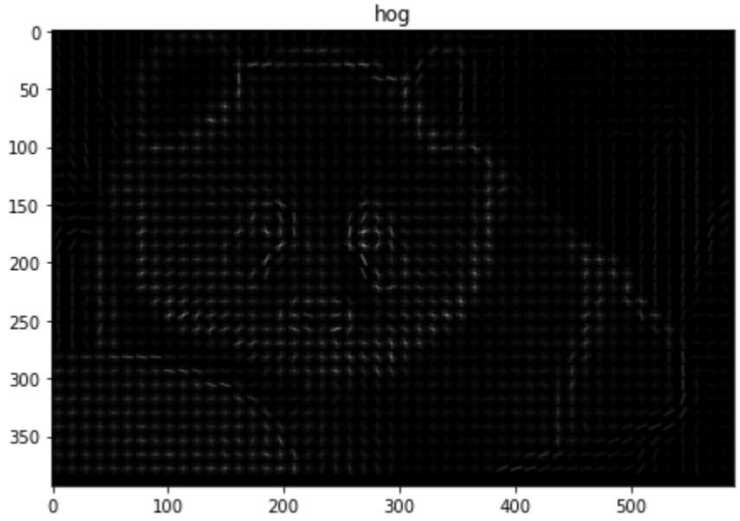
Histogram of Oriented Gradients: commonly used in object detection

Eg) Pandas… 😉

Histogram of Oriented Gradients

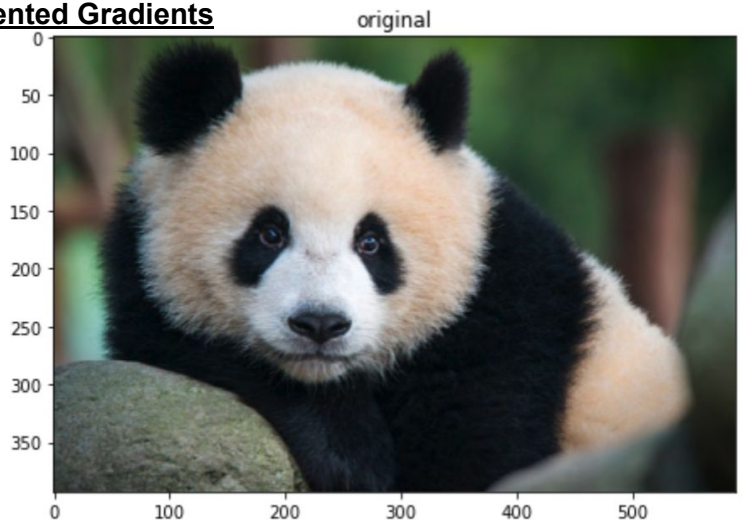
Eg) Pandas… 😉

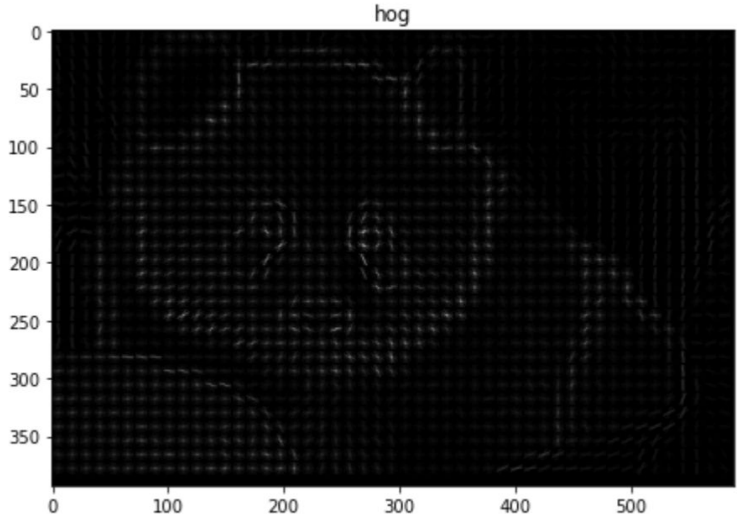




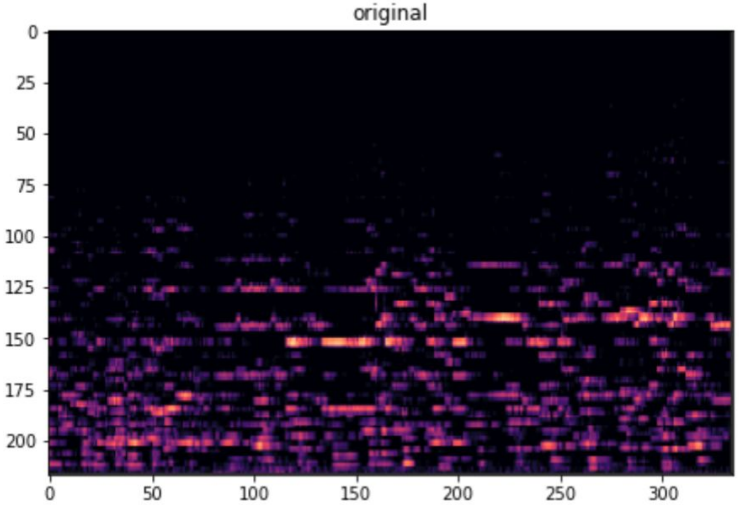
Histogram of Oriented Gradients

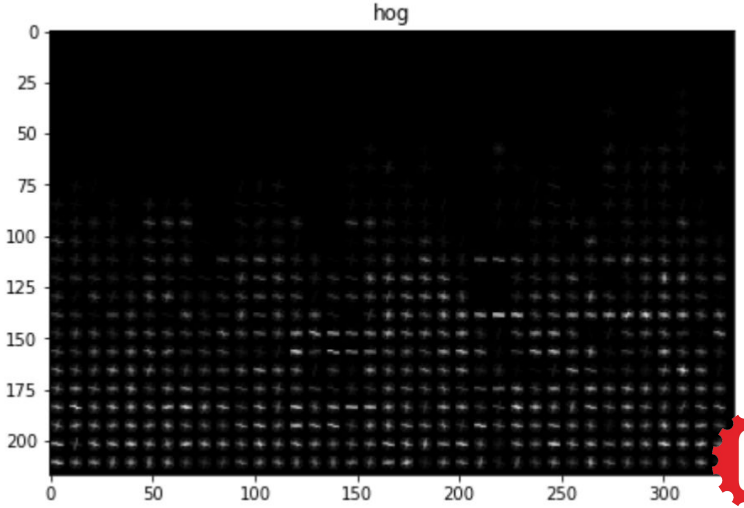
Eg) Pandas… 😉





Spectrogram equivalent





Model Performance on HOG spectrograms (metric: accuracy)

Model	Parameters	Train Score	Test Score	CV Mean	Comments
LogisticRegression	default	0.9987	0.6350	0.5682	HOG (9x9) PC
Logistic Regression	Ridge	0.9987	0.5250	0.4906	HOG (9x9) PC
LogisticRegression	default	0.9987	0.6500	**0.5945**	HOG (12x12) PC
LightGBM	default	1.0000	0.5450	0.4668	HOG (12x12) PC
LightGBM	{'class_weight': None, 'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 1000, 'num_leaves': 50}	1.0000	0.5300	0.4793	HOG (12x12) PC

- Increasing the ‘pixels per cell’ size for hog features to 12x12 (from 9x9) improved accuracy score.
- But overall achieved accuracy was still not as high as modelling on the the original audio features.

1. Problem Statement
2. EDA & Preprocessing
3. Audio Features
4. Spectrograms
5. Conclusions & Future Expansion

1. **Conclusions**
   - kNN and LightGBM performed the best out of all models tested.
   - Modelling on spectrograms did not perform as well as on original audio signal features.

2. **Shortcomings**
   - Small dataset (999 spectrograms); more needed.
   - Pixel "unravelling" is not the best way; CNN?

3. **Future Expansion**
   - Implement polynomial features to try to improve Logistic Regression score.
   - Apply similar technique to my own music collection, to see how the model performs on imbalanced classes.

Thanks for listening/reading :)