In [3]:
```python
import numpy as np
import pandas as pd
import seaborn as sns

from sklearn.feature_selection import r_regression
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn import preprocessing
from matplotlib import pyplot as plt
```

Q3

How are class labels distributed?

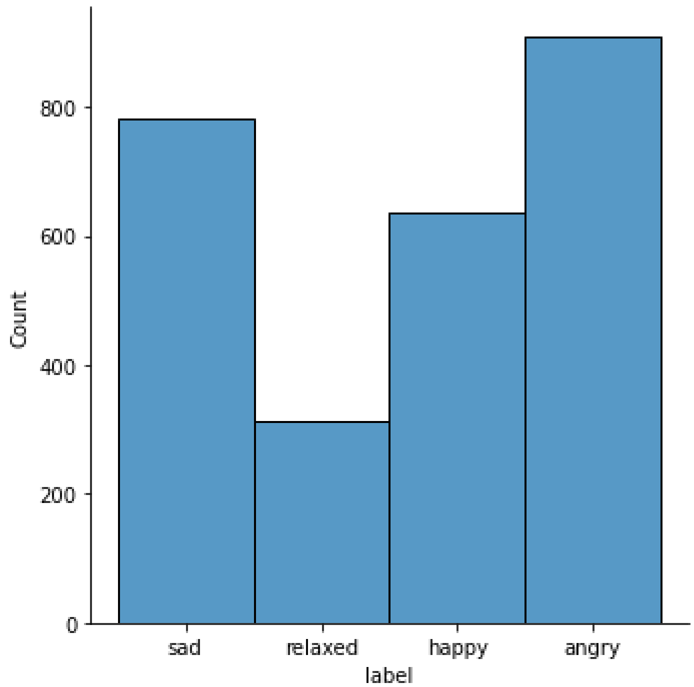
In [4]:
```python
#Load data

raw_data_labeled = pd.read_csv('task_2_annotations_labeled.csv')
aggregated_data_labeled = pd.read_csv('task_2_annotaions_aggregated_labels.cs
raw_data_labeled.head()
```

Out[4]:

	pianist_id	segment_id	annotator_id	arousal	valence	gems_wonder	gems_transcendence
0	1.0	0.0	91.0	1.0	-1.0	2.0	1.0
1	1.0	0.0	19.0	2.0	-1.0	3.0	3.0
2	1.0	0.0	189.0	2.0	0.0	2.0	1.0
3	1.0	0.0	126.0	2.0	2.0	4.0	5.0
4	1.0	0.0	26.0	4.0	2.0	3.0	5.0

In [5]:
```
sns.displot(data=raw_data_labeled, x="label")
```

Out[5]: <seaborn.axisgrid.FacetGrid at 0x17db7f358e0>



Are the classes imbalanced? How much?

We use the entropy to determine how imbalanced the classes are. To get a balance measure between 0 and 1, with 1 being perfectly balanced and 0 being totally unbalanced (i.e. only 1 of k classes in data) we divide the entropy by log k. Since entropy is 0 if there is only one class and log k if the classes are equally distributed, we divide the entropy by log k, to get 0 for a totally unbalanced dataset (1 class) and 1 for a totally balanced dataset.

In [6]:
```python
labels = raw_data_labeled['label']
```

In [7]:
```python
def balance(data, k=2): #k = number of different classes

    if k == 1:
        raise ValueError('k must be >= 2')

    count_per_class = data.value_counts()
    n = len(labels)

    H = 0

    for c_i in count_per_class:
        H += c_i/n * np.log(c_i/n)

    balance = -H/np.log(k)
    return balance
```

In [8]:
```python
balance(labels, k=4)
```

Out[8]:
```
0.954494698684711
```

How are the features distributed?

xxxxxxx TODO! xxxxxxx

Are there any pairs or subsets of features that seem highly correlated or redundant?

In [9]:
```python
#load low level dataset
features = pd.read_csv('data_features.csv')
```

In [10]:
```python
def get_redundant_pairs(df):
    '''Get diagonal and lower triangular pairs of correlation matrix'''
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    return pairs_to_drop

def get_top_abs_correlations(df, n=5, ascending=False):
    au_corr = df.corr().abs().unstack()
    labels_to_drop = get_redundant_pairs(df)
    au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=ascer
    return au_corr[0:n]
```

Covar and top correlation pairs

In [11]:
```
#calculate covariance matrix for features
features_cov = features.iloc[: , 3:].corr() # remove snipped_id etc..

print("Top Absolute Correlations")
print(get_top_abs_correlations(features_cov, 36, ascending=False))
```

```
Top Absolute Correlations
librosa_mfcc_mean_0          librosa_mfcc_pct_50_0           0.999963
librosa_mfcc_mean_2          librosa_mfcc_pct_50_2           0.999928
librosa_mfcc_mean_6          librosa_mfcc_pct_50_6           0.999861
mirtoolbox_roughness_mean    mirtoolbox_roughness_pct_50     0.999811
librosa_mfcc_mean_7          librosa_mfcc_pct_50_7           0.999763
librosa_mfcc_mean_3          librosa_mfcc_pct_50_3           0.999726
librosa_mfcc_mean_5          librosa_mfcc_pct_50_5           0.999591
librosa_mfcc_mean_8          librosa_mfcc_pct_50_8           0.999538
mirtoolbox_dynamics_mean     mirtoolbox_dynamics_pct_50      0.999521
librosa_mfcc_mean_1          librosa_mfcc_pct_50_1           0.999302
librosa_mfcc_mean_10         librosa_mfcc_pct_50_10          0.998873
librosa_mfcc_mean_9          librosa_mfcc_pct_50_9           0.998688
mirtoolbox_dynamics_pct_50   mirtoolbox_dynamics_pct_90      0.998626
mirtoolbox_dynamics_mean     mirtoolbox_dynamics_pct_10      0.998362
                             mirtoolbox_dynamics_pct_90      0.998208
librosa_mfcc_mean_4          librosa_mfcc_pct_50_4           0.997740
librosa_mfcc_mean_11         librosa_mfcc_pct_50_11          0.997708
mirtoolbox_dynamics_pct_10   mirtoolbox_dynamics_pct_50      0.997228
librosa_mfcc_mean_0          librosa_mfcc_pct_90_0           0.996573
librosa_mfcc_pct_50_0        librosa_mfcc_pct_90_0           0.996565
librosa_mfcc_mean_0          librosa_mfcc_pct_10_0           0.996552
librosa_mfcc_pct_10_0        librosa_mfcc_pct_50_0           0.996371
mirtoolbox_novelty_mean      mirtoolbox_novelty_pct_90       0.995871
essentia_strong_peak_mean    essentia_strong_peak_stdev      0.995861
mirtoolbox_dynamics_pct_10   mirtoolbox_dynamics_pct_90      0.993412
librosa_mfcc_mean_2          librosa_mfcc_pct_90_2           0.991686
librosa_mfcc_pct_50_2        librosa_mfcc_pct_90_2           0.991410
librosa_mfcc_pct_90_0        mirtoolbox_dynamics_pct_90      0.991251
librosa_chroma_mean_8        librosa_chroma_pct_50_8         0.990489
librosa_mfcc_pct_90_0        mirtoolbox_dynamics_pct_50      0.990469
                             mirtoolbox_dynamics_mean        0.990421
librosa_chroma_mean_1        librosa_chroma_pct_50_1         0.990366
librosa_mfcc_mean_2          librosa_mfcc_pct_10_2           0.990288
librosa_mfcc_pct_10_2        librosa_mfcc_pct_50_2           0.990180
librosa_chroma_mean_4        librosa_chroma_pct_50_4         0.990151
librosa_chroma_mean_9        librosa_chroma_pct_50_9         0.990106
dtype: float64
```

1) 36 feature pairs have a covariance of higher than 0.99

2) 237 feature pairs have a covariance of higher than 0.9

In [12]:
```
#add labels to snippets in features dataframe
for i, el in aggregated_data_labeled.iterrows():
    features.loc[features['segment_id'] == i, 'label'] = el['label']
```

Q5

Which features seem useful for classification?

Feature importance with random forest

In [15]:

```
#encode labels as numerical values
le = preprocessing.LabelEncoder()
le.fit(features.label)
features['label'] = le.transform(features.label)

X = features.drop(['pianist_id', 'segment_id', 'snippet_id', 'label'], axis=1
y = features.label
```

In [16]:

```
#create split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, rar
```

In [17]:

```
#train random forest
rf = RandomForestRegressor(n_estimators=150)
rf.fit(X_train, y_train)
```

Out[17]:

```
RandomForestRegressor(n_estimators=150)
```

In [18]:

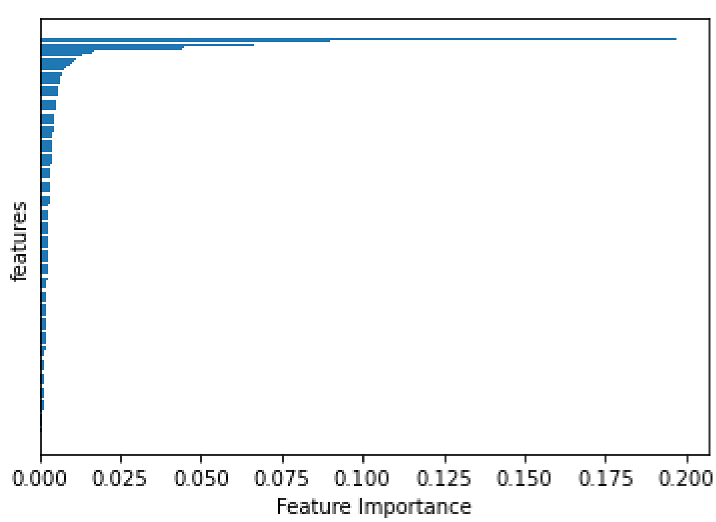
```
sort = rf.feature_importances_.argsort()
```

In [19]:

```
plt.barh(X.columns[sort], rf.feature_importances_[sort])
plt.rcParams["figure.figsize"] = (10,3)

plt.yticks([])
plt.xlabel("Feature Importance")
plt.ylabel('features')
```

Out[19]:

```
Text(0, 0.5, 'features')
```

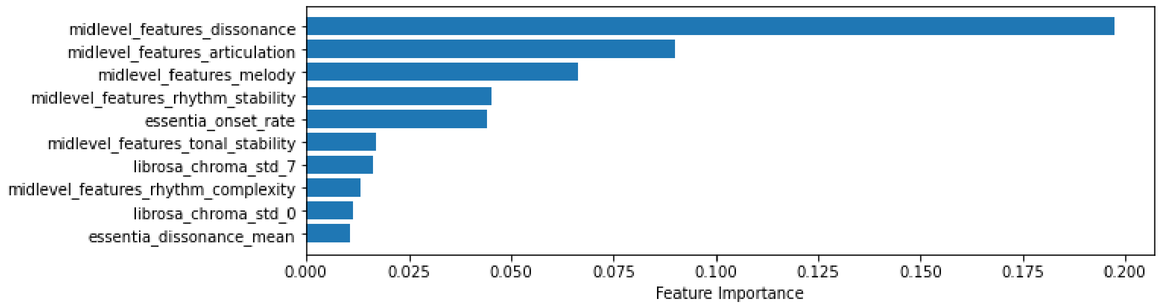


We used a random forest to determine feature importance. On the chart above we can see that most features are not important for classification, and only a hand full are very important.

Let's take a closer look at the most important features.

In [20]:
```
plt.barh(X.columns[sort][-10:], rf.feature_importances_[sort][-10:])

plt.rcParams["figure.figsize"] = (5,5)
plt.xlabel("Feature Importance")
```

Out[20]:
```
Text(0.5, 0, 'Feature Importance')
```



Which features are correlated with the labels?

In [22]:
```
r_sort = r_regression(X, y).argsort()
r_sort
```

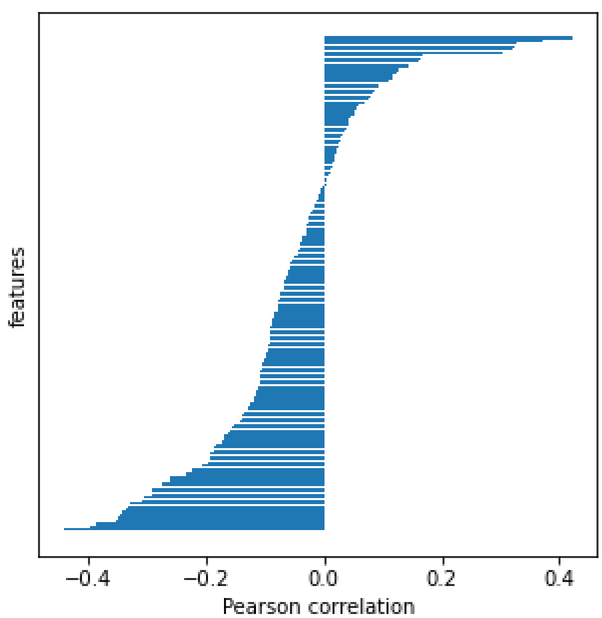
Out[22]:
```
array([140, 138,   0,  80,  76,  79, 149, 148,  78, 145,   3, 147, 143,
       167, 164, 166, 168, 137,  13,  14,  61,  64,  43,  44,  63,  56,
         5,  46,  49,  58,  41,  48,  59,  38,  60,  18,  65,  39,  36,
        11, 146,  84,  81, 165, 123,  50,  12,  75,  57,  83,  85,  25,
        21,  23, 121,  19, 124,  71,  72,  97,   7, 113,   4,  53, 162,
       158, 142,  40,  95,  24, 111,  62, 114,  47, 159, 125, 161,  22,
        74,  92,  16, 163,  73, 115,  45,  68, 109,  91, 106, 108,   1,
       152,  94, 110,  15,  33, 100,  77,  93, 133,  37,   8, 131,  54,
       135, 105,  99, 102, 134,   9, 101,  96, 104,  69, 153, 103,  10,
        28, 118, 132, 107, 119, 116,  66, 127, 144,  42,  87, 160, 120,
        20,  98,  51, 128, 150, 117, 112, 130,  34, 122, 126,  82, 129,
        31, 157,  70,  29,  17, 156, 154,   2,  35,  26, 155,  55, 151,
        32, 139,  67,  30,   6,  52,  27,  90,  89,  86,  88, 141, 136],
      dtype=int64)
```

In [25]:
```
X.columns[r_sort]
```

Out[25]:
```
Index(['midlevel_features_dissonance', 'midlevel_features_rhythm_complexit
y',
       'essentia_dissonance_mean', 'librosa_mfcc_pct_90_0',
       'librosa_mfcc_mean_0', 'librosa_mfcc_pct_50_0',
       'mirtoolbox_dynamics_pct_90', 'mirtoolbox_dynamics_pct_50',
       'librosa_mfcc_pct_10_0', 'mirtoolbox_dynamics_mean',
       ...
       'librosa_chroma_pct_90_2', 'essentia_pitch_salience_stdev',
       'librosa_chroma_std_7', 'librosa_chroma_std_2', 'librosa_mfcc_pct_90
_2',
       'librosa_mfcc_pct_50_2', 'librosa_mfcc_mean_2', 'librosa_mfcc_pct_10
_2',
       'midlevel_features_tonal_stability', 'midlevel_features_melody'],
      dtype='object', length=169)
```

In [31]:
```
# plt.barh(X.columns[r_sort][-10:], r_regression(X, y)[r_sort][-10:])
plt.barh(X.columns[r_sort], r_regression(X, y)[r_sort])

plt.yticks([])
plt.rcParams["figure.figsize"] = (5,5)
plt.xlabel("Pearson correlation")
plt.ylabel("features")
```

Out[31]:
```
Text(0, 0.5, 'features')
```



Most features have a weak correlation below +- 0.3. Only a few features have a medium correlation between +-0.3 and +-0.5

In [ ]:
```

```