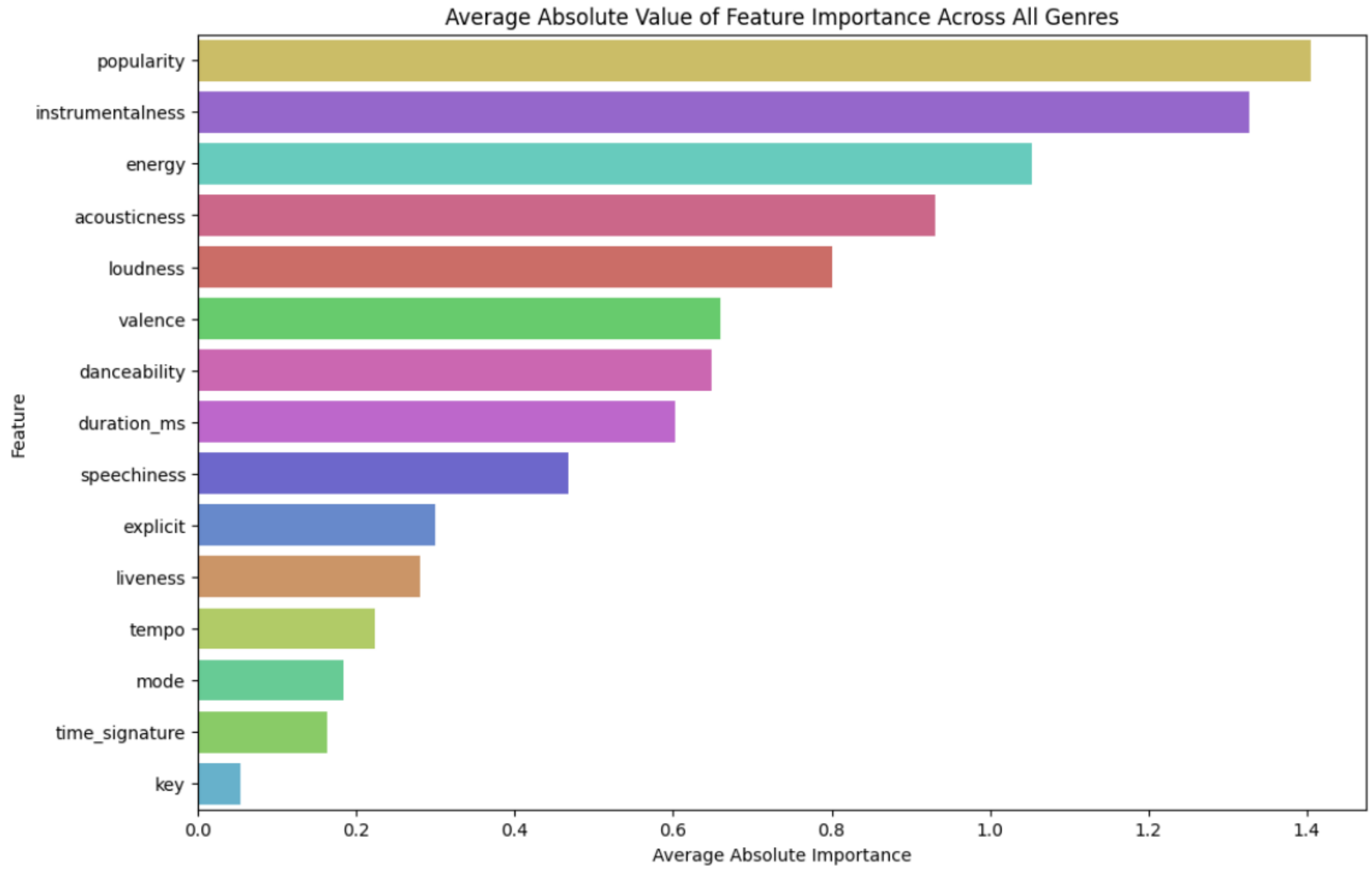


V. KNN, Decision Trees, and Random Forest

KNN

We can fit a KNN model to the Spotify data by encoding the categorical response variable and using split data into 80/20 training and testing.

In [31]:

```python
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.preprocessing import LabelEncoder

df_knn = df.copy()
label_encoder = LabelEncoder()

df_knn['track_genre'] = label_encoder.fit_transform(df_knn['track_genre'])
X_train, X_test, y_train, y_test = train_test_split(df_knn.drop(['track_genre'], axis=1), df_knn.track_genre, test_size=0.2)
X_train['explicit'] = X_train['explicit'].astype(int)
X_test['explicit'] = X_test['explicit'].astype(int)

k = 14
knn = KNeighborsRegressor(n_neighbors=k)
knn.fit(X_train, y_train)

knn_train_pred = knn.predict(X_train)
mse = mean_squared_error(y_train, knn_train_pred)
rmse = sqrt(mse)
print(f"rMSE: {rmse}")

knn_score = knn.score(X_test, y_test)
print(f"kNN Accuracy: {knn_score}")
```

```
rMSE: 4.155109436772109
kNN Accuracy: 0.3839119863538719
```

Cross Validation:

In [32]:

```python
from sklearn.model_selection import GridSearchCV

param_grid = {'n_neighbors': range(1, 26)}
grid_search = GridSearchCV(KNeighborsRegressor(), param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Get best k
best_k = grid_search.best_params_['n_neighbors']
print(f"Optimal k: {best_k}")

# Train the model with the optimal k
knn_optimal = KNeighborsRegressor(n_neighbors=best_k)
knn_optimal.fit(X_train, y_train)

# Predict and evaluate
knn_train_pred_optimal = knn_optimal.predict(X_train)
mse_optimal = mean_squared_error(y_train, knn_train_pred_optimal)
rmse_optimal = sqrt(mse_optimal)
print(f"Optimal rMSE: {rmse_optimal}")

knn_test_pred_optimal = knn_optimal.predict(X_test)
knn_score_optimal = knn_optimal.score(X_test, y_test)
print(f"Optimal kNN Accuracy: {knn_score_optimal}")
```

```
Optimal k: 11
Optimal rMSE: 4.083738010674413
Optimal kNN Accuracy: 0.37882312215287484
```

Decision Tree

The usefulness of a Decision tree was explored in the following section. We set some parameters, such as a max depth of 20 splits, and a minimum sample in each of 20. Gini is used as a criterion. We ended up with a 53% overall accuracy.

In [33]:

```
import plotly.figure_factory as ff
import plotly.graph_objects as go
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance
from sklearn import tree
from joblib import Parallel, delayed
from itertools import product
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image

X_train, X_test, y_train, y_test = train_test_split(df_knn.drop(['track_genre'], axis=1),
                                                    df_knn.track_genre, test_size=0.3)

music_tree = DecisionTreeClassifier(criterion='gini', min_samples_split=20, max_depth=20)
music_tree.fit(X_train, y_train)

music_tree_pred = music_tree.predict(X_test)

print("Confusion Matrix: \n", confusion_matrix(y_test, music_tree_pred))
print("Accuracy: \n", metrics.accuracy_score(y_test, music_tree_pred))
print(classification_report(y_test, music_tree_pred))
```

```
Confusion Matrix:
 [[132   2  11  13   4   6   0  20   6   5  15  12   9  11   5  14   6   8
   26   1]
 [  7 188   3   2   0   0  26  10  37   3   0   7   0   0   1   0   0   0
    2   1]
 [  6   1 104  37   2  14   1   5   2   3  16   3  32  44   8   4   4   4
    2   6]
 [  7   3  26 174   4   0   2   2   4  10   7   3  16   6   2   3   9   6
   10  21]
 [  7   0   5   0 161  19   0   2   0   0   2   1   0   2  18   0  38  40
    0   0]
 [  4   1  16   0  17 158   1   2   0   0   9   1   5   8  34   1   7   9
    0   0]
 [  1  32   0   0   0   0 246   4   6  12   0   1   2   0   0   0   0   0
    0   1]
 [ 15   5   5   3   5   5   2 192  16   6   4  21  11   3   0   1   0   1
    0   0]
 [  8  33   7   3   0   2   1  20 176   4   4  11   8   2   5   1   3   1
    0   1]
 [  7   6   2  10   0   0   7   7   0 221  10   0   7   2   0   1   0   0
   20   6]
 [ 16   0  19   5  14  19   0   3   0   4  93   0   9  21  28  47   5  10
    4   0]
 [  8  10  10   3   2   3   4  30  27   1   3 178  15   5   0   1   6   0
    0   1]
 [  8   1  27  23  11  13   1   4   9   5  21  14 109   9   8   3  18   2
    3  12]
 [  7   0  78   9   5  18   1   7   2   1  30   5  29 100   8  17   0   7
    0   6]
 [  7   1  15   1  33  23   0   2   0   2  28   2   8   9  94   0  58  18
    4   0]
 [ 14   0   4  10   3   2   1   2   1   2  60   0   6  24   1 176   0   3
    0   2]
 [  1   0   7   1  30  15   0   2   0   2   8   3  14   4  69   0 119  19
    1   1]
 [ 10   0  10   1  23   9   0   2   0   0  15   0   1  14  18   3   9 165
    0   0]
 [ 13   0   4   7   0   0   0   3   0  28   9   0   5   0   6   0   2   0
  219   4]
 [  2   4  14  30   1   1   0   2   4   6   1   2   7   1   1   0   0   0
    9 205]]
Accuracy:
 0.5361616836479038
              precision    recall  f1-score   support

           0       0.47      0.43      0.45       306
           1       0.66      0.66      0.66       287
           2       0.28      0.35      0.31       298
           3       0.52      0.55      0.54       315
           4       0.51      0.55      0.53       295
           5       0.51      0.58      0.54       273
           6       0.84      0.81      0.82       305
           7       0.60      0.65      0.62       295
           8       0.61      0.61      0.61       290
           9       0.70      0.72      0.71       306
          10       0.28      0.31      0.29       297
```

```
          11       0.67      0.58      0.62       307
          12       0.37      0.36      0.37       301
          13       0.38      0.30      0.34       330
          14       0.31      0.31      0.31       305
          15       0.65      0.57      0.60       311
          16       0.42      0.40      0.41       296
          17       0.56      0.59      0.58       280
          18       0.73      0.73      0.73       300
          19       0.76      0.71      0.73       290

    accuracy                           0.54      5987
   macro avg       0.54      0.54      0.54      5987
weighted avg       0.54      0.54      0.54      5987
```

In [ ]:

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'criterion': ['gini'],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

dt = DecisionTreeClassifier(random_state=123)


grid_search = GridSearchCV(
    estimator=dt,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    verbose=1,
    n_jobs=-1
)

# Perform grid search
grid_search.fit(X_train, y_train)

# Print best parameters and best score
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)

# Use the best model to predict on the test set
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Evaluate the model
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Hyperparameters: {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10}
Best Cross-Validation Accuracy: 0.5552691935380942
              precision    recall  f1-score   support

           0       0.51      0.47      0.49       287
           1       0.70      0.65      0.67       298
           2       0.31      0.28      0.29       298
           3       0.60      0.44      0.51       315
           4       0.58      0.59      0.59       315
           5       0.51      0.58      0.54       288
           6       0.86      0.85      0.85       294
           7       0.65      0.64      0.64       291
           8       0.63      0.58      0.60       302
           9       0.76      0.71      0.74       302
          10       0.29      0.30      0.30       297
          11       0.63      0.62      0.62       291
          12       0.41      0.34      0.37       293
          13       0.34      0.49      0.40       299
          14       0.33      0.36      0.34       298
          15       0.62      0.77      0.69       296
          16       0.51      0.48      0.49       337
          17       0.55      0.62      0.59       303
          18       0.70      0.73      0.71       294
          19       0.81      0.64      0.72       289

    accuracy                           0.56      5987
   macro avg       0.56      0.56      0.56      5987
weighted avg       0.56      0.56      0.56      5987
```

In [ ]:
```
cv_score_dt = cross_val_score(dt, X_test, y_test, cv=5, scoring='accuracy')
cv_score_dt
```

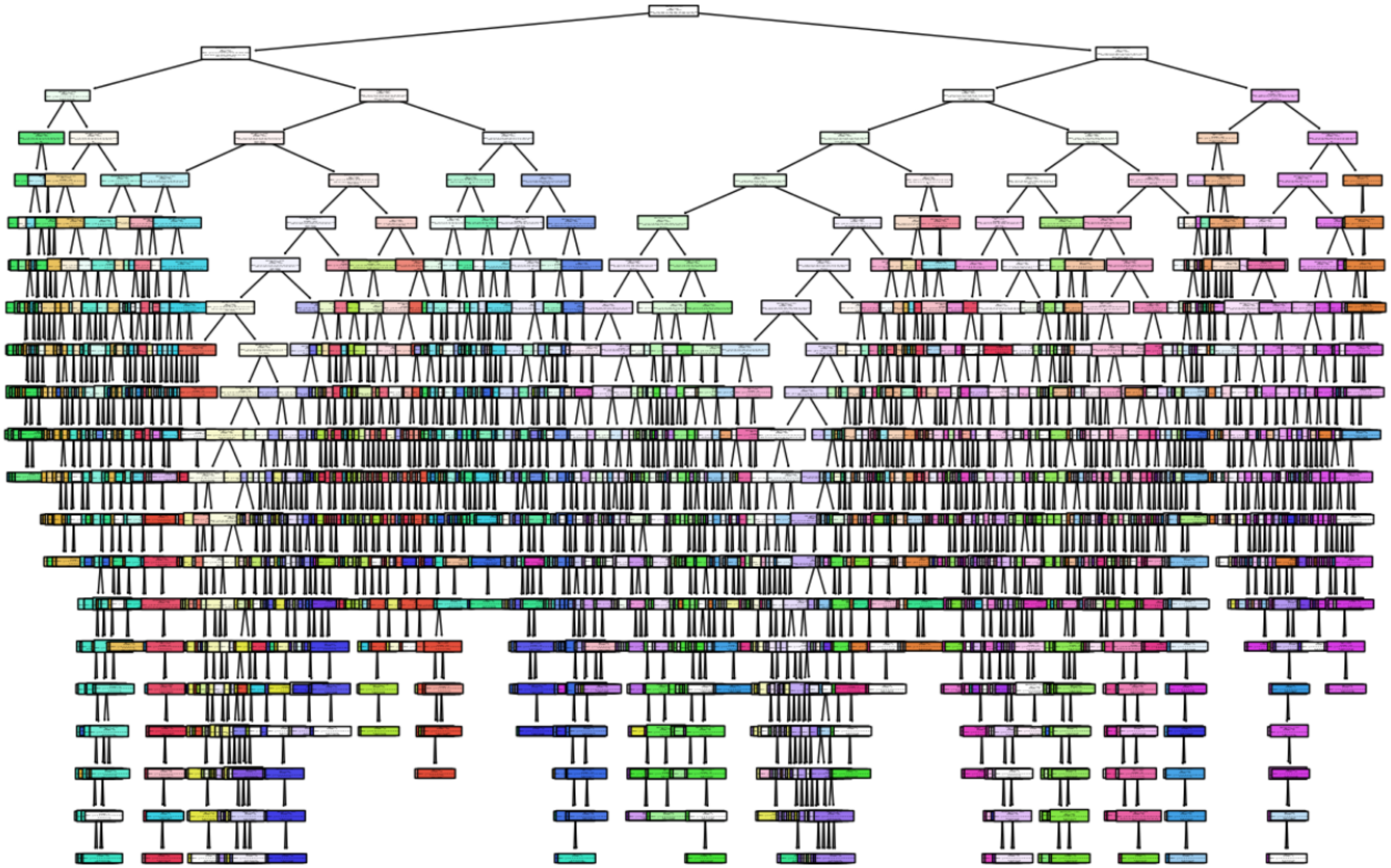
Out[ ]:
```
array([0.4933222 , 0.47495826, 0.46616541, 0.50459482, 0.48788638])
```

In [ ]:
```
np.mean(cv_score_dt)
```

Out[ ]:
```
0.4853854167974193
```

The following plot shows the process of the decision tree. The different labels are color coded in each of the boxes, with the strength of the predictions represented by the transparency of the color.

In [ ]:
```
plt.figure(figsize=(15,10))
plot_tree(music_tree, filled=True, feature_names=list(X_train.columns), class_names=genres, rounded=True)
plt.show()
```



Random Forest

When fitting the Random Forest model, we can see some similarity in the feature importance to the Logistic Regression.

In [34]:
```
music_rf = RandomForestClassifier(random_state=123)
music_rf.fit(X_train, y_train)

music_rf_pred = music_rf.predict_proba(X_test)

music_rf_importance = pd.DataFrame({
    'feature': X_train.columns,
    'importance': music_rf.feature_importances_
}).sort_values('importance', ascending=False).reset_index().drop('index', axis=1)
music_rf_importance
```

Out[34]:

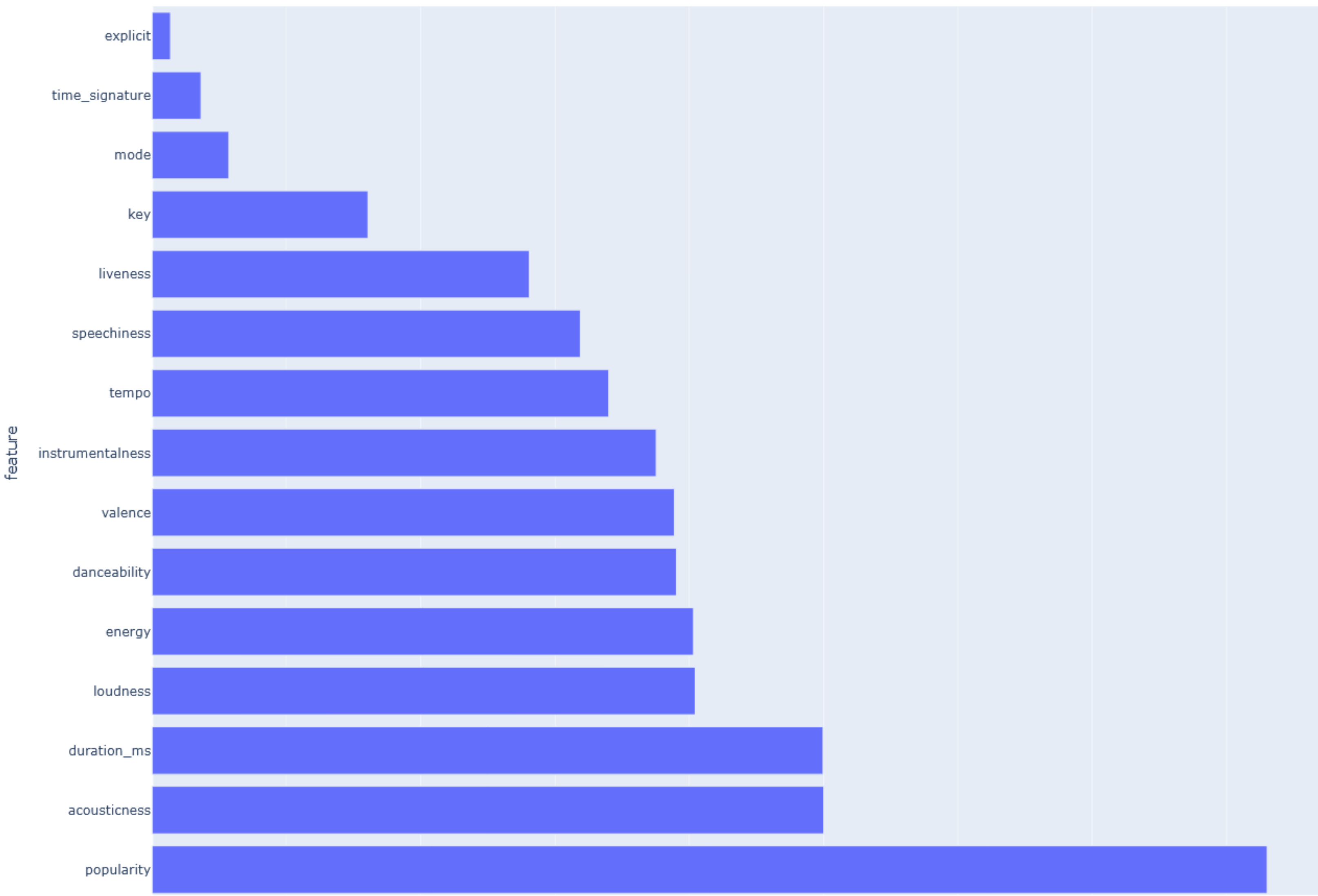
	feature	importance
0	popularity	0.166057
1	acousticness	0.100013
2	duration_ms	0.099923
3	loudness	0.080884
4	energy	0.080626
5	danceability	0.078105
6	valence	0.077792
7	instrumentalness	0.075068
8	tempo	0.068005
9	speechiness	0.063770
10	liveness	0.056186
11	key	0.032148
12	mode	0.011425
13	time_signature	0.007267
14	explicit	0.002732

The Random Forest feature importance is represented in the bar plot shown below.

In [35]:
```
import plotly.express as px
import plotly.offline as pyo
pyo.init_notebook_mode(connected=True)

px.bar(music_rf_importance, y='feature', x='importance',
       orientation='h', title='Impurity Importance for Random Forest', height=1000)
```

Impurity Importance for Random Forest



In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report


rf = RandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [2, 4],
    'max_features': ['sqrt', 'log2'],
}

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           cv=5, scoring='accuracy', verbose=2, n_jobs=-1)

grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)

y_pred = grid_search.best_estimator_.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
Fitting 5 folds for each of 144 candidates, totalling 720 fits
```

```
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarning:

invalid value encountered in cast
```

```
Best Parameters: {'max_depth': 30, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}
Best Score: 0.6642326749484407
              precision    recall  f1-score   support

           0       0.62      0.53      0.57       287
           1       0.75      0.75      0.75       298
           2       0.49      0.38      0.43       298
           3       0.77      0.62      0.69       315
           4       0.63      0.68      0.65       315
           5       0.57      0.72      0.64       288
           6       0.88      0.89      0.89       294
           7       0.82      0.76      0.79       291
           8       0.70      0.70      0.70       302
           9       0.84      0.81      0.82       302
          10       0.46      0.38      0.41       297
          11       0.73      0.74      0.74       291
          12       0.53      0.53      0.53       293
          13       0.45      0.38      0.41       299
          14       0.37      0.41      0.38       298
          15       0.65      0.84      0.74       296
          16       0.54      0.54      0.54       337
          17       0.65      0.67      0.66       303
          18       0.80      0.93      0.86       294
          19       0.81      0.86      0.83       289

    accuracy                           0.65      5987
   macro avg       0.65      0.66      0.65      5987
weighted avg       0.65      0.65      0.65      5987
```

In [ ]:

```
cv_score_rf = cross_val_score(rf, X_test, y_test, cv=5, scoring='accuracy')
cv_score_rf
```

Out[ ]:

```
array([0.63856427, 0.61936561, 0.63074353, 0.64828739, 0.63324979])
```

In [ ]:

```
np.mean(cv_score_rf)
```

Out[ ]:

```
0.63404211697859
```

VI. PCA and Clustering

Principal Component Analysis is a technique for dimensionality reduction when a data set is very high-dimensional (i.e., contains many features). The goal is to simplify the data set while retaining as much information or variability in the data as possible.

Clustering is an unsupervised machine learning technique that creates groups in the data. For this clustering analysis, we attempt to cluster based on time signature.

Clustering is susceptible to a phenomenon known as the curse of dimensionality, in which data set is so high-dimensional and complex that clustering is difficult to perform and largely inaccurate, as the more complex a data set is, the less meaningful distance metrics become. Thus, reducing the dimension of the data set may aid in clustering effectiveness. Our approach involves performing k-means clustering on the data, then performing principal component analysis to create a transformed (simpler) data set, performing k-means clustering again on the new PCA-transformed data, and finally comparing evaluation metrics for the two clustering schemes.