```
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.

In [36]:
```
import plotly.express as px
import plotly.graph_objects as go
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score, silhouette_samples, rand_score, adjusted_rand_score
```

In [37]:
```
ss = preprocessing.StandardScaler()
df_clustering = df_raw.drop(columns='track_genre')
# Create a standardized version of the data for modeling purposes after EDA
num_cols = df_clustering.columns.values.tolist()
num_cols.remove('time_signature')
df_clustering[num_cols] = ss.fit_transform(df_clustering.drop(columns='time_signature'))
```

In [38]:
```
set(df_clustering['time_signature'])
```

Out[38]:
```
{0, 1, 3, 4, 5}
```

In [39]:
```
kmeans = KMeans(n_clusters=5)
y_kmeans = kmeans.fit_predict(df_clustering.drop(columns='time_signature'))
```

```
/Users/peiyuanlee/miniforge3/envs/myenv/lib/python3.11/site-packages/sklearn/cluster/_kmeans.py:1412: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
```

In [40]:
```
for i in np.arange(0, len(kmeans.labels_)):
  if kmeans.labels_[i] > 1:
    kmeans.labels_[i] += 1
set(kmeans.labels_)
```

Out[40]:
```
{0, 1, 3, 4, 5}
```

In [41]:
```
print(silhouette_score(df_clustering.drop(columns='time_signature'), kmeans.labels_))
```

```
0.12619622421023385
```

In [42]:
```
pca_U, pca_d, pca_V = np.linalg.svd(df_clustering.drop(columns='time_signature'))
```

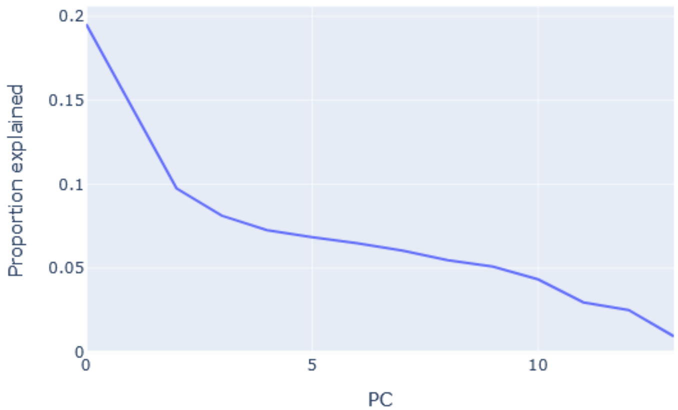
In [43]:
```
prop_var = np.square(pca_d) / sum(np.square(pca_d))
scree_data = pd.DataFrame(
{"PC": 1 + np.arange(0, prop_var.shape[0]),
"variability_explained": prop_var.round(4),
"cumulative_variability_explained": prop_var.cumsum().round(4)
})
scree_data.head(20)
```

Out[43]:

	PC	variability_explained	cumulative_variability_explained
0	1	0.1954	0.1954
1	2	0.1461	0.3416
2	3	0.0975	0.4391
3	4	0.0813	0.5204
4	5	0.0726	0.5930
5	6	0.0684	0.6614
6	7	0.0648	0.7263
7	8	0.0604	0.7866
8	9	0.0548	0.8415
9	10	0.0510	0.8925
10	11	0.0433	0.9358
11	12	0.0296	0.9655
12	13	0.0251	0.9906
13	14	0.0094	1.0000

In [44]:

```
px.line(x=np.arange(14),
y=scree_data.iloc[range(14), :].loc[:, 'variability_explained'],
labels={"x": "PC",
"y": "Proportion explained"},
width=600, height=400)
```



We attempted to perform Principal Component Analysis on the data to prepare for clustering based on time signature. The scree plot indicates that the first two principal components capture about 34.16% of the variability in the data, and after that, each principal component makes a small, consistent contribution. Unfortunately, keeping only the first two principal components would simply result in a data set that does not capture nearly enough information from the original data to be usable. Moreover, if we want to retain most of the information in the original data, let's say 90%, then we would need to keep the first ten principal components, which is not a very successful dimensionality reduction down from fourteen original features.

In [45]:
```
X_train_pca = np.dot(df_clustering.drop(columns='time_signature'), pca_V[np.arange(0, 9)].T)
X_train_pca = pd.DataFrame(X_train_pca, columns=['PC' + str(x) for x in np.arange(1, 10)])
X_train_pca.head()
```

Out[45]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9
0	-1.737953	-1.237693	-0.573628	-1.434923	0.417737	-0.251574	-0.329893	-0.705392	-0.509406
1	-1.502498	-0.722715	-1.620502	0.727191	-0.105492	-0.296720	-1.382797	0.192409	-0.210325
2	-1.223136	-1.176942	-0.947351	0.804374	0.866419	-0.695013	-1.072830	-0.055739	0.416886
3	-1.750299	-0.736756	-0.589327	-1.595235	0.480411	-0.535357	0.041964	-0.623377	-0.235042
4	-1.099484	-1.289466	-0.236192	-0.447316	1.276546	0.314724	-0.978155	-1.587110	-0.371327

In [46]:
```
kmeans_new = KMeans(n_clusters=5)
y_kmeans = kmeans_new.fit_predict(X_train_pca)
```

```
/Users/peiyuanlee/miniforge3/envs/myenv/lib/python3.11/site-packages/sklearn/cluster/_kmeans.py:1412: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
```

In [47]:
```
for i in np.arange(0, len(kmeans_new.labels_)):
  if kmeans_new.labels_[i] > 1:
    kmeans_new.labels_[i] += 1
set(kmeans_new.labels_)
```

Out[47]:
```
{0, 1, 3, 4, 5}
```

In [48]:
```
print(silhouette_score(X_train_pca, kmeans_new.labels_))
```

```
0.14164406496997287
```

Keeping the first nine principal components and clustering with the new PCA transformed data set, we obtain only a slightly higher silhouette score of 0.1296, compared to a score of 0.1098 before performing PCA. Silhouette score is a measure of how tightly and distinctly the data is clustered, where 1 is tightly clustered and 0 is loosely (and indistinctively) clustered. It is one measure of effectiveness for a clustering algorithm. The calculated silhouette scores for both clustering schemes suggests that the effects of PCA are minimal for this data set, so we will not employ it in the main analysis of our classification of track genres.

VI. Neural Networks

In [ ]:
```
df_NN = df.copy()
df_NN.head()
```

Out[ ]:

	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	track_genre
5000	2.352090	0.252651	-0.181294	0.063329	0.812652	0.215398	1.212446	-1.349109	-0.285892	-1.094908	-0.541600	-0.382670	0.301861	0.178699	0.241705	anime
5001	2.479707	0.127650	-0.181294	-0.576031	1.159506	-0.912980	1.222452	0.741230	-0.347826	-1.131754	-0.119504	0.341987	-0.256109	-1.062736	0.241705	anime
5002	3.117792	-0.366398	-0.181294	0.282539	1.187097	-1.195074	0.657329	0.741230	0.416500	-1.125528	-0.542039	-0.721608	-0.622775	-0.718561	0.241705	anime
5003	2.479707	0.108868	-0.181294	-0.137612	0.982138	0.497493	1.206533	-1.349109	0.151871	-0.982160	-0.542049	-0.678954	-0.463355	0.372413	0.241705	anime
5004	1.650196	0.208297	-0.181294	0.976702	0.524921	-1.477168	0.640500	-1.349109	-0.366125	-1.079190	-0.540688	-0.593647	0.214180	0.146523	0.241705	anime

From the feature importance derived from random forests, we select the top 10 features to reduce noise from irrelevant features. To optimize model performance, we designed three feedforward neural network architectures with 4, 6, and 10 layers, incorporating dropout and batch normalization layers to mitigate overfitting and stabilize training.

In [ ]:
```
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
X = df_NN.drop(['track_genre', 'mode', 'explicit', 'liveness', 'key', 'time_signature'], axis = 1)
y = LabelEncoder().fit_transform(df_NN['track_genre'])

X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=2000, random_state=123, stratify=y)

X_train, X_valid, y_train, y_valid = train_test_split(
    X_temp, y_temp, test_size=1000, random_state=123, stratify=y_temp)

# Convert to categorical (onehot encoding matrices)
y_train = to_categorical(y_train, num_classes=20)
y_valid = to_categorical(y_valid, num_classes=20)
y_test = to_categorical(y_test, num_classes=20)
```