Step 1: Import Data & Libraries

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.linear_model import LinearRegression, Ridge, RidgeCV, Lasso, LassoCV
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
```

In [ ]:

```
#Read in data
featDetails = 'complete_feature_1000.xlsx'
featDF = pd.read_excel(featDetails)
```

In [ ]:

```
#select subset of data
data = featDF.drop('id',axis=1)

#view first six rows of data
data[0:6]
```

Out[ ]:

	genrel_hip_hop	genrelabstract_hip_hop	genrelacoustic_pop	genreladult_standards	genrelaesthetic_rap	genrelafrofuturism
0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0

6 rows × 545 columns

Step 2: Perform the appropriate train test split. The original dataset has 544 predictor variables and one target variable (track_pop)

In [ ]:

```
#X = data[['danceability','energy','loudness','speechiness','acousticness','instrumentaln
ess','liveness','valence','tempo']] #all other features
X = data.drop('track_pop',axis=1)

y = data[['track_pop']] #population metric
```

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
)
```

Step 3: Standardize Features

Before running PCR, we standardize each original feature to be on the same scale before generating the principal components, as principal component computation is based on variance. Otherwise features with larger

ranges would be misconstrued as having higher variances and disproportionately affect the PCs generated. This would be done with something like the scale() method from sklearn.preprocessing to standardize the original features, i.e., center to the mean and scale to unit variance.

In [ ]:

```
# Run standardization on X variables
X_train_scaled, X_test_scaled = scale(X_train), scale(X_test)
X_train_scaled
```

Out[ ]:

```
array([[-0.05005248,  0.        , -0.04850675, ..., -0.29738086,
        -0.73178456,  0.73178456],
       [-0.05005248,  0.        , -0.04850675, ..., -0.29738086,
        -0.73178456,  0.73178456],
       [-0.05005248,  0.        , -0.04850675, ..., -0.29738086,
         1.36652242, -1.36652242],
       ...,
       [-0.05005248,  0.        , -0.04850675, ..., -0.29738086,
        -0.73178456,  0.73178456],
       [-0.05005248,  0.        , -0.04850675, ..., -0.29738086,
         1.36652242, -1.36652242],
       [-0.05005248,  0.        , -0.04850675, ..., -0.29738086,
        -0.73178456,  0.73178456]])
```

Step 4 — Run Baseline Regression Models

To evaluate the performance of the PCR model,we ran three baseline models (Standard Linear Regression, Lasso Regression, and Ridge Regression) for benchmarks to allow comparison, and saved the RMSE scores. In particular, we want to save the following:

i. 9-fold cross-validation RMSE in the training set

ii. Prediction RMSE on the test set

In [ ]:

```
# Define cross-validation folds
cv = KFold(n_splits=9, shuffle=True, random_state=42)
```

Linear Regression - No Regularization

In [ ]:

```
# Linear Regression
lin_reg = LinearRegression().fit(X_train_scaled, y_train)
lr_score_train = -1 * cross_val_score(lin_reg, X_train_scaled, y_train, cv=cv, scoring='
neg_root_mean_squared_error').mean()
lr_score_test = mean_squared_error(y_test, lin_reg.predict(X_test_scaled), squared=False
)
```

Lasso Regression - L1 Regularization

In [ ]:

```
# Lasso Regression
lasso_reg = LassoCV().fit(X_train_scaled, y_train)
lasso_score_train = -1 * cross_val_score(lasso_reg, X_train_scaled, y_train, cv=cv, scor
ing='neg_root_mean_squared_error').mean()
lasso_score_test = mean_squared_error(y_test, lasso_reg.predict(X_test_scaled), squared=
False)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:1571:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:1571:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
```

```
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0003997260956063453, tolerance: 0.00027787768651690327
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.00040823682051893684, tolerance: 0.00027787768651690327
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.00038377737625983066, tolerance: 0.00027787768651690327
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:1571:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0003938538498068356, tolerance: 0.0002838684807256238б
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0007593423852627623, tolerance: 0.0002838684807256238б
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:1571:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:1571:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:1571:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:1571:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:1571:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0002977504674053222, tolerance: 0.00027553450464692925
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0003583442866237352, tolerance: 0.0002829420553224218З
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:1571:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0007128612397473422, tolerance: 0.0002828398810021958
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0003779889543868986, tolerance: 0.0002828398810021958
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0007257775536455568, tolerance: 0.0002828398810021958
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0007365073073600792, tolerance: 0.0002828398810021958
```

```
terations. Duality gap: 0.0007365073073600792, tolerance: 0.0002828398810021958
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0007063499134996043, tolerance: 0.0002828398810021958
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0005747554002247801, tolerance: 0.0002828398810021958
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0006323640811882747, tolerance: 0.0002795673756321509
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0006397451332085247, tolerance: 0.0002795673756321509
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0004689026940059015 7, tolerance: 0.0002795673756321509
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0007385671822319 75, tolerance: 0.0002795673756321509
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:644: C
onvergenceWarning: Objective did not converge. You might want to increase the number of i
terations. Duality gap: 0.0007132198117862 92, tolerance: 0.0002795673756321509
  positive,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:1571:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Ridge Regression - L2 Regularization

In [ ]:

```
# Ridge Regression
ridge_reg = RidgeCV().fit(X_train_scaled, y_train)
ridge_score_train = -1 * cross_val_score(ridge_reg, X_train_scaled, y_train, cv=cv, scor
ing='neg_root_mean_squared_error').mean()
ridge_score_test = mean_squared_error(y_test, ridge_reg.predict(X_test_scaled), squared=
False)
```

Step 5 — Generate Principal Components

The principal components for the original features can be generated using PCA() from sklearn.decomposition.

In [ ]:

```
# Generate all the principal components
pca = PCA()  # Default n_components = min(n_samples, n_features)
X_train_pc = pca.fit_transform(X_train)
X_train_pc.shape
```

Out[ ]:

```
(800, 544)
```

In [ ]:

```
# View explained variance ratio for each principal component
pca.explained_variance_ratio_
```

Out[ ]:

```
array([9.01915129e-02, 4.23893735e-02, 3.21713970e-02, 2.65782870e-02,
       2.48582093e-02, 2.26747681e-02, 2.16818901e-02, 2.02629492e-02,
       1.89413125e-02, 1.87472730e-02, 1.81523148e-02, 1.70950428e-02,
       1.65100290e-02, 1.59319437e-02, 1.47936621e-02, 1.37829168e-02,
```

1.30139778e-02, 1.26926071e-02, 1.17406539e-02, 1.14123019e-02,
1.11717686e-02, 1.06706512e-02, 9.83329762e-03, 9.32882369e-03,
9.17526962e-03, 8.86955131e-03, 8.19040098e-03, 7.90980520e-03,
7.43375177e-03, 7.25713599e-03, 7.17451995e-03, 6.96876431e-03,
6.70554690e-03, 6.40707266e-03, 6.29614660e-03, 6.08788447e-03,
6.01589642e-03, 5.71513436e-03, 5.43249634e-03, 5.17541546e-03,
5.13129305e-03, 4.99034240e-03, 4.89010572e-03, 4.75953426e-03,
4.63646184e-03, 4.61226560e-03, 4.44068388e-03, 4.37652776e-03,
4.20081302e-03, 4.08238406e-03, 4.03192804e-03, 3.98761147e-03,
3.95495261e-03, 3.83679706e-03, 3.68409932e-03, 3.65888350e-03,
3.62764404e-03, 3.53109516e-03, 3.49371078e-03, 3.38631893e-03,
3.31721562e-03, 3.30086492e-03, 3.21209320e-03, 3.11113480e-03,
3.09614231e-03, 3.06656370e-03, 3.01603844e-03, 2.98225100e-03,
2.94675504e-03, 2.93017028e-03, 2.88399440e-03, 2.84486058e-03,
2.79289399e-03, 2.72826931e-03, 2.69108532e-03, 2.67867848e-03,
2.62324666e-03, 2.59645560e-03, 2.56737876e-03, 2.50161360e-03,
2.48324368e-03, 2.44193605e-03, 2.38134212e-03, 2.37091090e-03,
2.33137261e-03, 2.31831192e-03, 2.28055666e-03, 2.25822039e-03,
2.25643404e-03, 2.19468954e-03, 2.18283497e-03, 2.17218610e-03,
2.14221366e-03, 2.12528134e-03, 2.11297746e-03, 2.08371020e-03,
2.06645884e-03, 2.04622261e-03, 2.01011556e-03, 1.99945430e-03,
1.96510997e-03, 1.96177499e-03, 1.91561423e-03, 1.90306176e-03,
1.88597328e-03, 1.87164702e-03, 1.85839755e-03, 1.83754632e-03,
1.82066797e-03, 1.80071239e-03, 1.79123180e-03, 1.77796407e-03,
1.74419395e-03, 1.73347330e-03, 1.70860225e-03, 1.68218234e-03,
1.66766818e-03, 1.66394049e-03, 1.63663433e-03, 1.61450253e-03,
1.61152581e-03, 1.59390616e-03, 1.56754839e-03, 1.55343991e-03,
1.54436987e-03, 1.52866584e-03, 1.52360869e-03, 1.49938877e-03,
1.49228272e-03, 1.46327701e-03, 1.45655155e-03, 1.43733234e-03,
1.42576392e-03, 1.40514721e-03, 1.39771460e-03, 1.39398052e-03,
1.37876335e-03, 1.36852629e-03, 1.35203548e-03, 1.34379198e-03,
1.33432580e-03, 1.32947173e-03, 1.31971276e-03, 1.30900970e-03,
1.28356993e-03, 1.28282337e-03, 1.27803054e-03, 1.25631403e-03,
1.24138176e-03, 1.22431481e-03, 1.22268997e-03, 1.20952415e-03,
1.18923565e-03, 1.18271808e-03, 1.17464591e-03, 1.15048817e-03,
1.13994385e-03, 1.13677422e-03, 1.11992474e-03, 1.11053204e-03,
1.08905136e-03, 1.08579029e-03, 1.07558348e-03, 1.06707273e-03,
1.05768841e-03, 1.05021972e-03, 1.03924853e-03, 1.03309971e-03,
1.02238574e-03, 1.01779390e-03, 1.00791785e-03, 9.99728197e-04,
9.91245516e-04, 9.89291866e-04, 9.81739215e-04, 9.73432820e-04,
9.66955890e-04, 9.63578285e-04, 9.60097143e-04, 9.54995716e-04,
9.51033035e-04, 9.50412660e-04, 9.47307877e-04, 9.44887371e-04,
9.44301390e-04, 9.40488490e-04, 9.33554119e-04, 9.30325761e-04,
9.28362986e-04, 9.23739587e-04, 9.21797217e-04, 9.14213245e-04,
9.11414134e-04, 9.06021695e-04, 8.95906792e-04, 8.89647744e-04,
8.79828534e-04, 8.70562936e-04, 8.65793810e-04, 8.56337833e-04,
8.50802612e-04, 8.43471739e-04, 8.36639750e-04, 8.27755795e-04,
8.23181859e-04, 8.14468767e-04, 8.12685051e-04, 8.01942621e-04,
7.93884703e-04, 7.93304167e-04, 7.86257503e-04, 7.78349443e-04,
7.75719181e-04, 7.65841752e-04, 7.55294091e-04, 7.50315634e-04,
7.42358385e-04, 7.36172947e-04, 7.23179971e-04, 7.20299434e-04,
7.10289593e-04, 7.09243764e-04, 7.07844834e-04, 6.98796102e-04,
6.91288047e-04, 6.83728747e-04, 6.75523530e-04, 6.67768434e-04,
6.60751041e-04, 6.55687082e-04, 6.51505464e-04, 6.45753171e-04,
6.40822083e-04, 6.33993234e-04, 6.28266617e-04, 6.21091668e-04,
6.15851566e-04, 6.01389725e-04, 5.95402323e-04, 5.89178587e-04,
5.87574968e-04, 5.82667799e-04, 5.78143504e-04, 5.73611180e-04,
5.70057875e-04, 5.64378783e-04, 5.55960479e-04, 5.49960140e-04,
5.48522572e-04, 5.41871539e-04, 5.38853780e-04, 5.34289178e-04,
5.31399059e-04, 5.22464931e-04, 5.18871273e-04, 5.15750763e-04,
5.07500661e-04, 5.06338022e-04, 4.93729411e-04, 4.90150731e-04,
4.84257367e-04, 4.82548074e-04, 4.74984270e-04, 4.72465542e-04,
4.66954357e-04, 4.63403467e-04, 4.58589348e-04, 4.55300019e-04,
4.51455145e-04, 4.50017489e-04, 4.37519328e-04, 4.33673455e-04,
4.29479625e-04, 4.26172745e-04, 4.22419392e-04, 4.19571522e-04,
4.10917448e-04, 4.08140115e-04, 4.02074932e-04, 3.95792443e-04,
3.92309071e-04, 3.87879749e-04, 3.83229352e-04, 3.79806979e-04,
3.74612974e-04, 3.73071393e-04, 3.64845847e-04, 3.64270225e-04,
3.58293438e-04, 3.54722806e-04, 3.52246635e-04, 3.50709387e-04,
3.45825548e-04, 3.43230001e-04, 3.41377672e-04, 3.33906258e-04,
3.29634322e-04, 3.26150819e-04, 3.19745947e-04, 3.15530926e-04,

```
        2.91951289e-04, 2.89281330e-04, 2.88095707e-04, 2.83728368e-04,
        2.82304012e-04, 2.80919777e-04, 2.73460599e-04, 2.69994093e-04,
        2.67171095e-04, 2.65956454e-04, 2.63207061e-04, 2.57458502e-04,
        2.55254451e-04, 2.50344766e-04, 2.49801677e-04, 2.46623756e-04,
        2.44114958e-04, 2.39059063e-04, 2.34008383e-04, 2.30048425e-04,
        2.24420604e-04, 2.23769030e-04, 2.23194790e-04, 2.17164497e-04,
        2.11446315e-04, 2.07301236e-04, 2.06212657e-04, 2.02698928e-04,
        2.02565714e-04, 1.99322886e-04, 1.94491730e-04, 1.93242467e-04,
        1.89337052e-04, 1.86835534e-04, 1.82135106e-04, 1.78787080e-04,
        1.74027033e-04, 1.72379044e-04, 1.69473647e-04, 1.67537342e-04,
        1.64627033e-04, 1.62756290e-04, 1.60165293e-04, 1.58876525e-04,
        1.56860620e-04, 1.50481808e-04, 1.47340816e-04, 1.39977522e-04,
        1.39521067e-04, 1.35701747e-04, 1.32709040e-04, 1.30320498e-04,
        1.26632729e-04, 1.25123622e-04, 1.22905949e-04, 1.19956130e-04,
        1.17388365e-04, 1.16522444e-04, 1.11771209e-04, 1.08318804e-04,
        1.07541847e-04, 1.04109499e-04, 9.95087280e-05, 9.83662030e-05,
        9.57936885e-05, 9.01701671e-05, 8.69490535e-05, 8.42555254e-05,
        8.18681104e-05, 7.92904919e-05, 7.59744112e-05, 7.34630363e-05,
        7.30226189e-05, 6.36121512e-05, 6.05066591e-05, 5.88135037e-05,
        5.49917808e-05, 5.22388498e-05, 4.83018969e-05, 4.78082434e-05,
        4.63067413e-05, 4.57556357e-05, 4.29247927e-05, 3.84671468e-05,
        3.59643648e-05, 3.47279952e-05, 3.24834800e-05, 3.07671925e-05,
        2.85403066e-05, 2.74285704e-05, 2.36705651e-05, 2.32169413e-05,
        2.05406157e-05, 2.04110973e-05, 1.85558409e-05, 1.69281510e-05,
        1.59689287e-05, 1.43874166e-05, 1.27717362e-05, 9.55920102e-06,
        8.08272902e-06, 2.11016796e-32, 2.89946163e-33, 1.77854620e-33,
        1.65010484e-33, 1.21123063e-33, 9.80534481e-34, 8.64093546e-34,
        6.16083777e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 5.55717400e-34, 5.55717400e-34, 5.55717400e-34,
        5.55717400e-34, 2.75792443e-34, 2.22303710e-34, 2.03517735e-34])
```

In [ ]:

```
# View first 5 rows of all principal components
pd.DataFrame(pca.components_.T).loc[:4,:]
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...
[illegible]	5.726224e-	[illegible]	3.002828e-	3.962011e-	[illegible]	[illegible]	8.203966e-	[illegible]	[illegible]	3.532256e-	

0	08	1.516632e-03	08	04	0.005841	9.046791e-05	08	1.466289e-07	1.220259e-08	04	...
1	4.440892e-16	7.632783e-17	-2.775558e-17	2.428613e-17	-0.000000	-2.927346e-17	-1.387779e-16	7.632783e-17	8.326673e-17	-2.515349e-17	...
2	-3.420392e-03	2.524021e-03	-4.811170e-03	4.373686e-03	-0.009845	1.453716e-03	1.459842e-02	-5.674016e-03	-3.555305e-03	-5.841350e-03	...
3	-8.098723e-04	-1.992931e-03	5.406789e-03	-3.607296e-03	-0.005888	-1.531853e-03	6.801605e-03	4.604724e-04	-3.672619e-04	-8.830091e-03	...
4	2.172856e-03	-9.316633e-04	6.703280e-06	-8.780955e-04	-0.001454	1.320051e-04	-2.155210e-03	-1.003185e-03	3.586232e-05	6.063938e-04	...

5 rows × 544 columns

Step 6 — Determine the Number of Principal Components

The number of principal components (k) is typically determined by cross-validation and visual analysis.

The value k is essentially a hyperparameter that we need to tune. We iterate over an increasing number of principal components to include in regression modeling and assess the resulting RMSE scores.

In [ ]:

```
# Loop through different count of principal components for linear regression
lin_reg = LinearRegression()
rmse_list = []
for i in range(1, X_train_pc.shape[1]+1):
    rmse_score = -1 * cross_val_score(lin_reg,
                                      X_train_pc[:,:i], # Use first k principal componen
ts
                                      y_train,
                                      cv=cv,
                                      scoring='neg_root_mean_squared_error').mean()
    rmse_list.append(rmse_score)
```

The next task is to inspect the plot of training set cross-validation RMSE vs. the number of principal components used:

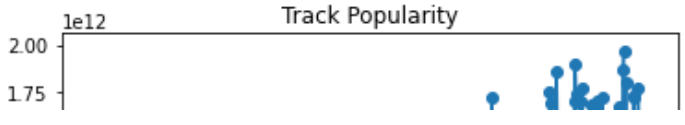
We see that the training set performance of PCR worsens significantly (i.e., RMSE increases) when too many principal components are considered, in line with what we expect.

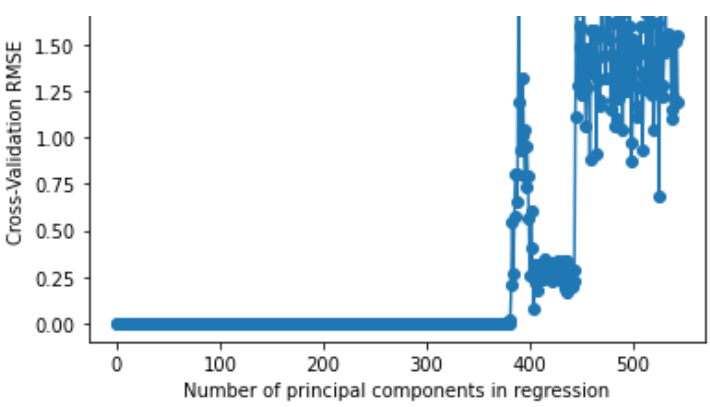
The green line is the RMSE benchmark from the baseline standard linear regression model using all original features.

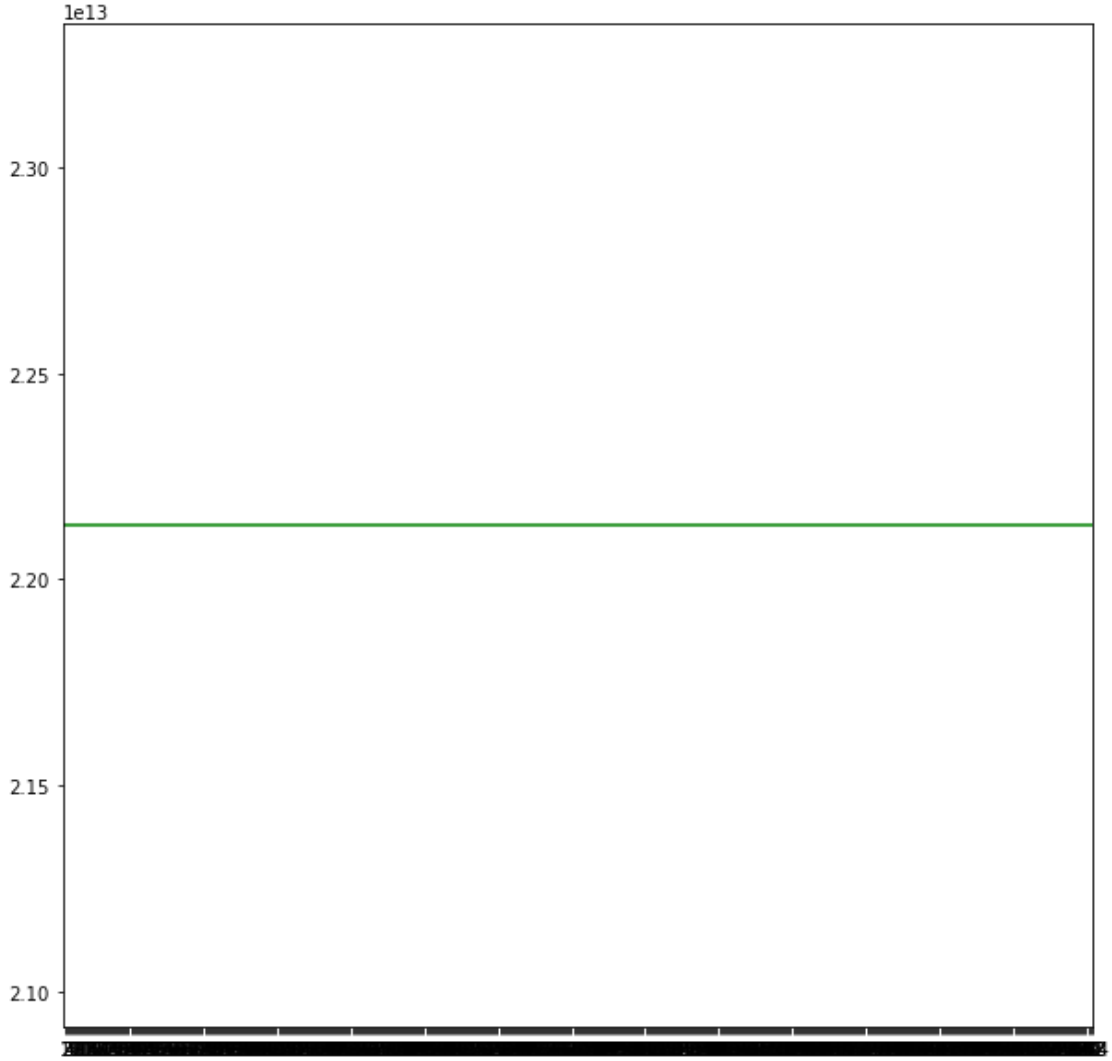
The plot shows that the lowest cross-validation RMSE (minimum point in the plot) occurs when there are M=9 principal components. The RMSE at M=9 is, in fact, significantly below the green line.

In [ ]:

```
# Plot RMSE vs count of principal components used
plt.plot(rmse_list, '-o')
plt.xlabel('Number of principal components in regression')
plt.ylabel('Cross-Validation RMSE')
#plt.yscale('log')
plt.title('Track Popularity')
plt.figure(figsize=(10,10))
plt.xlim(xmin=-1);
plt.xticks(np.arange(X_train_pc.shape[1]), np.arange(1, X_train_pc.shape[1]+1))
plt.axhline(y=lr_score_train, color='g', linestyle='-');
```

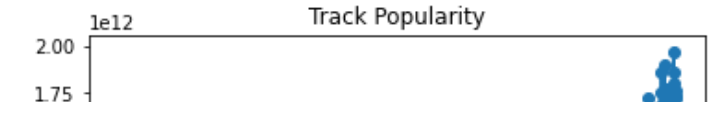


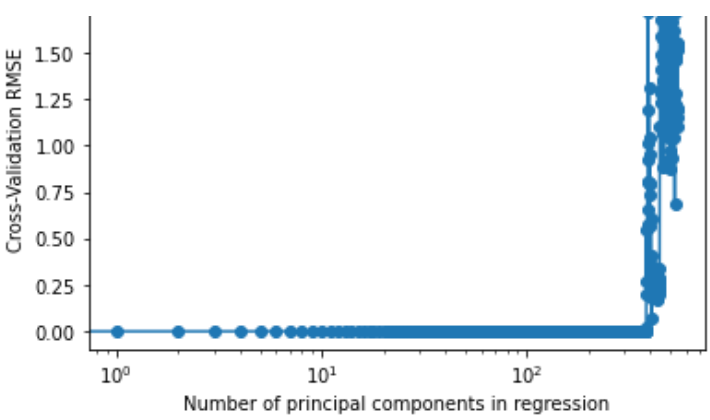


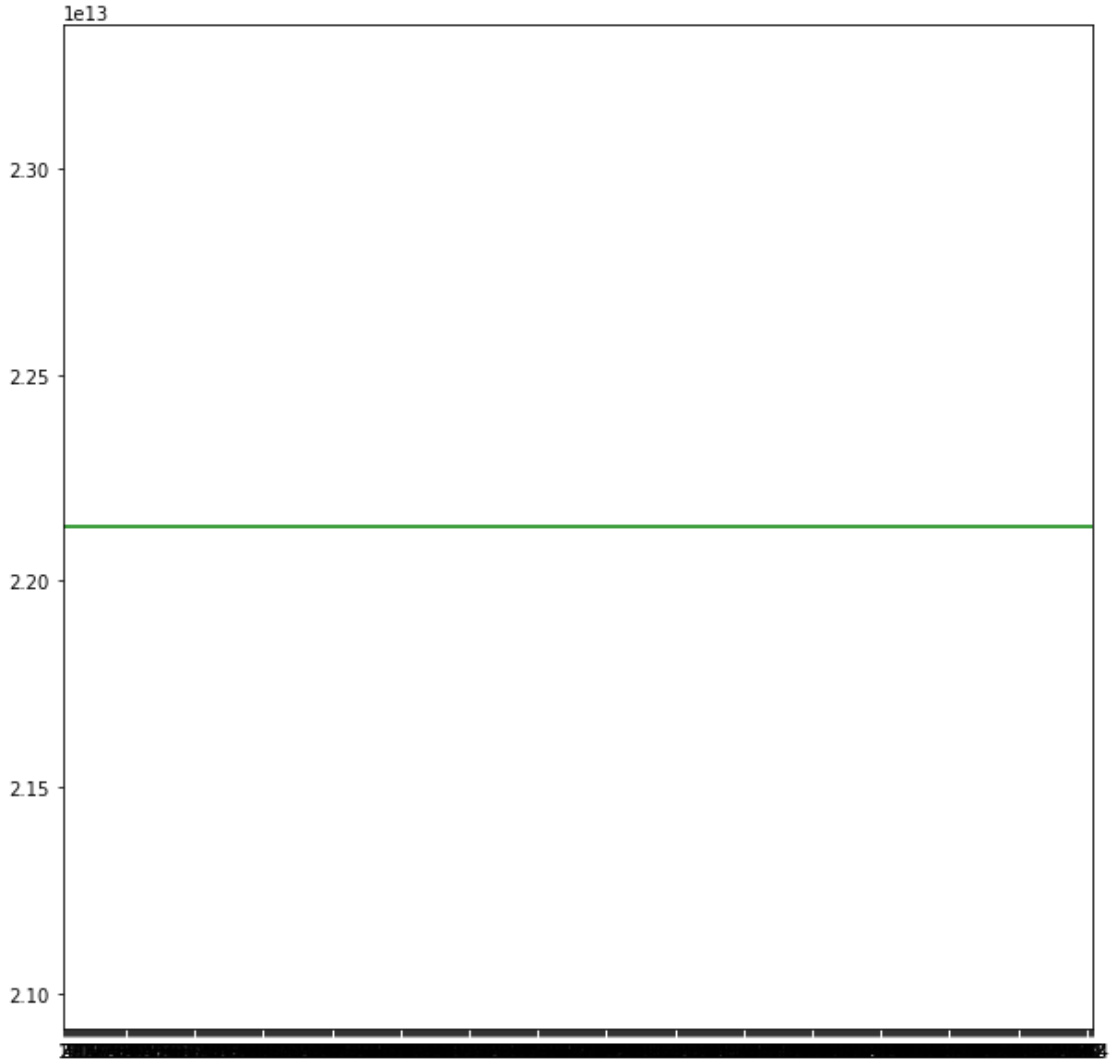


In [ ]:

```
# Plot RMSE vs count of principal components used with log scaled on x-axis
plt.plot(rmse_list, '-o')
plt.xlabel('Number of principal components in regression')
plt.ylabel('Cross-Validation RMSE')
plt.xscale('log')
#plt.yscale('log')
plt.title('Track Popularity')
fig = plt.figure(figsize=(10,10))
plt.xlim([200,400])
#plt.xlim(xmin=-1);
plt.xticks(np.arange(X_train_pc.shape[1]), np.arange(1, X_train_pc.shape[1]+1))
plt.axhline(y=lr_score_train, color='g', linestyle='-');
#plt.xscale('log')
#spacing = 0.25
#fig.subplots_adjust(bottom=spacing)
#plt.show()
```







In [ ]:

```
# Visually determine optimal number of principal components
best_pc_num = 125
```

Step 7: Perform Predictions with PCR model on Training Set

In [ ]:

```
# Train model on training set
lin_reg_pc = LinearRegression().fit(X_train_pc[:,:best_pc_num], y_train)
```

In [ ]:

```
# Get R2 score
lin_reg_pc.score(X_train_pc[:,:best_pc_num], y_train)
```

Out[ ]:

```
0.31037089874966184
```

In [ ]:

```
pcr_score_train = -1 * cross_val_score(lin_reg_pc,
                                       X_train_pc[:,:best_pc_num],
                                       y_train,
                                       cv=cv,
                                       scoring='neg_root_mean_squared_error').mean()
pcr_score_train
```

Out[ ]:

```
0.06899625129940946
```

In [ ]:

```
# Get principal components of test set
X_test_pc = pca.transform(X_test_scaled)[:,:best_pc_num]
X_test_pc.shape
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but PCA was fitted with feature names
  "X does not have valid feature names, but"
```

Out[ ]:

```
(200, 125)
```

In [ ]:

```
# Predict on test data
preds = lin_reg_pc.predict(X_test_pc)
pcr_score_test = mean_squared_error(y_test, preds, squared=False)
pcr_score_test
```

Out[ ]:

```
0.5243132545173128
```

Step 8: Evaluate and Interpret Results The following table displays the RMSE results from the various models, including the two PCR models trained using a different number of principal components.

In [ ]:

```
train_metrics = np.array([round(lr_score_train,3),
                          round(lasso_score_train,3),
                          round(ridge_score_train,3),
                          round(pcr_score_train,3)])
train_metrics = pd.DataFrame(train_metrics, columns=['RMSE (Train Set)'])
train_metrics.index = ['Linear Regression',
                       'Lasso Regression',
                       'Ridge Regression',
                       f'PCR ({best_pc_num} components)']
train_metrics
```

Out[ ]:

	RMSE (Train Set)
Linear Regression	2.213167e+13
Lasso Regression	6.500000e-02
Ridge Regression	7.800000e-02
PCR (125 components)	6.900000e-02

In [ ]:

```
test_metrics = np.array([round(lr_score_test,3),
                         round(lasso_score_test,3),
                         round(ridge_score_test,3),
                         round(pcr_score_test,3)])
test_metrics = pd.DataFrame(test_metrics, columns=['RMSE (Test Set)'])
```

```
test_metrics.index = ['Linear Regression',
                      'Lasso Regression',
                      'Ridge Regression',
                      f'PCR ({best_pc_num} components)']
test_metrics
```

Out[ ]:

	RMSE (Test Set)
Linear Regression	1.162224e+13
Lasso Regression	6.200000e-02
Ridge Regression	7.400000e-02
PCR (125 components)	5.240000e-01

Simulation with 110 PCs

In [ ]:

```

```

In [ ]:

```
# Visually determine optimal number of principal components
best_pc_num = 110
```

Step 7: Perform Predictions with PCR model on Training Set

In [ ]:

```
# Train model on training set
lin_reg_pc = LinearRegression().fit(X_train_pc[:,:best_pc_num], y_train)
```

In [ ]:

```
# Get R2 score
lin_reg_pc.score(X_train_pc[:,:best_pc_num], y_train)
```

Out[ ]:

```
0.2945451806882573
```

In [ ]:

```
pcr_score_train = -1 * cross_val_score(lin_reg_pc,
                                       X_train_pc[:,:best_pc_num],
                                       y_train,
                                       cv=cv,
                                       scoring='neg_root_mean_squared_error').mean()
pcr_score_train
```

Out[ ]:

```
0.06812026310133992
```

In [ ]:

```
# Get principal components of test set
X_test_pc = pca.transform(X_test_scaled)[:,:best_pc_num]
X_test_pc.shape
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have
valid feature names, but PCA was fitted with feature names
  "X does not have valid feature names, but"
```

Out[ ]:

```
(200, 110)
```

In [ ]:

```
# Predict on test data
preds = lin_reg_pc.predict(X_test_pc)
pcr_score_test = mean_squared_error(y_test, preds, squared=False)
pcr_score_test
```

Out[ ]:

```
0.503543611219579
```

Step 8: Evaluate and Interpret Results The following table displays the RMSE results from the various models, including the two PCR models trained using a different number of principal components.

In [ ]:

```
train_metrics = np.array([round(lr_score_train,3),
                          round(lasso_score_train,3),
                          round(ridge_score_train,3),
                          round(pcr_score_train,3)])
train_metrics = pd.DataFrame(train_metrics, columns=['RMSE (Train Set)'])
train_metrics.index = ['Linear Regression',
                       'Lasso Regression',
                       'Ridge Regression',
                       f'PCR ({best_pc_num} components)']
train_metrics
```

Out[ ]:

	RMSE (Train Set)
Linear Regression	2.213167e+13
Lasso Regression	6.500000e-02
Ridge Regression	7.800000e-02
PCR (110 components)	6.800000e-02

In [ ]:

```
test_metrics = np.array([round(lr_score_test,3),
                         round(lasso_score_test,3),
                         round(ridge_score_test,3),
                         round(pcr_score_test,3)])
test_metrics = pd.DataFrame(test_metrics, columns=['RMSE (Test Set)'])
test_metrics.index = ['Linear Regression',
                      'Lasso Regression',
                      'Ridge Regression',
                      f'PCR ({best_pc_num} components)']
test_metrics
```

Out[ ]:

	RMSE (Test Set)
Linear Regression	1.162224e+13
Lasso Regression	6.200000e-02
Ridge Regression	7.400000e-02
PCR (110 components)	5.040000e-01

Simulation with 9 PCs

In [ ]:

```
# Visually determine optimal number of principal components
best_pc_num = 9
```

Step 7: Perform Predictions with PCR model on Training Set

In [ ]:

```
# Train model on training set
lin_reg_pc = LinearRegression().fit(X_train_pc[:,:best_pc_num], y_train)
```

In [ ]:

```
# Get R2 score
lin_reg_pc.score(X_train_pc[:,:best_pc_num], y_train)
```

Out[ ]:

```
0.09803570154554087
```

In [ ]:

```
pcr_score_train = -1 * cross_val_score(lin_reg_pc,
                                       X_train_pc[:,:best_pc_num],
                                       y_train,
                                       cv=cv,
                                       scoring='neg_root_mean_squared_error').mean()
pcr_score_train
```

Out[ ]:

```
0.06710259247931293
```

In [ ]:

```
# Get principal components of test set
X_test_pc = pca.transform(X_test_scaled)[:,:best_pc_num]
X_test_pc.shape
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but PCA was fitted with feature names
  "X does not have valid feature names, but"
```

Out[ ]:

```
(200, 9)
```

In [ ]:

```
# Predict on test data
preds = lin_reg_pc.predict(X_test_pc)
pcr_score_test = mean_squared_error(y_test, preds, squared=False)
pcr_score_test
```

Out[ ]:

```
0.18865560825347916
```

Step 8: Evaluate and Interpret Results The following table displays the RMSE results from the various models, including the two PCR models trained using a different number of principal components.

In [ ]:

```
train_metrics = np.array([round(lr_score_train,3),
                          round(lasso_score_train,3),
                          round(ridge_score_train,3),
                          round(pcr_score_train,3)])
train_metrics = pd.DataFrame(train_metrics, columns=['RMSE (Train Set)'])
train_metrics.index = ['Linear Regression',
                       'Lasso Regression',
                       'Ridge Regression',
                       f'PCR ({best_pc_num} components)']
train_metrics
```

Out[ ]:

	RMSE (Train Set)
Linear Regression	2.213167e+13
Lasso Regression	6.500000e-02
Ridge Regression	7.800000e-02
PCR (9 components)	6.700000e-02

In [ ]:

```
test_metrics = np.array([round(lr_score_test,3),
                         round(lasso_score_test,3),
                         round(ridge_score_test,3),
                         round(pcr_score_test,3)])
test_metrics = pd.DataFrame(test_metrics, columns=['RMSE (Test Set)'])
test_metrics.index = ['Linear Regression',
                      'Lasso Regression',
                      'Ridge Regression',
                      f'PCR ({best_pc_num} components)']
test_metrics
```

Out[ ]:

	RMSE (Test Set)
Linear Regression	1.162224e+13
Lasso Regression	6.200000e-02
Ridge Regression	7.400000e-02
PCR (9 components)	1.890000e-01

Observations

We performed PCR with varying number of principal components (125, 110, and 9, respectively) to obtain RMSE results comparing benchmark models to PCR models trained using a different number of principal components.

We see that regardless of the number of principal components, PCR performed significantly better than standard linear regression. This makes sense considering all the PCR models utilized only a subset of all the original features.

We see that the PCR models performed on par with the other 2 baseline regression models based on the training set, which were trained on all the original features. It should be noted the PCR with 9 principal components, however, performed significantly better than all 3 baseline regression models on the test set.

This makes sense considering points made in the introduction for part 2, where we noted that the majority of original features were dummy variables that we felt would bear an insignificant impact to the target variable result compared to certain audio features. It seems added more components beyond the salient features begins to negatively affect the model.

It is worth remembering that PCR performs better in situations where the first few principal components capture most of the variation in the predictors and the relationship with the target.

Conclusion

In conclusion, our analysis showed that the RMSE remained at its lowest when only a few principal components were selected to be predictor variables for the regression model.

As demonstrated by the fact that the PCR with 9 principal components seemed the strongest model, PCR helps to reduce overfitting and, in line with theory, led to better performance than a standard linear regression model trained on all the original features.

PCR is also known to more noticeably improve performance when data has many featuers and significant multicollinearity, which happens to be the case with our song features data set.

multicollinearity, which happens to be the case with our song features data set.

Certain limitations of PCR worth noting: PCR is NOT a feature selection method because the resulting principal components (i) requires all the original features (ii) utilized all the original features to create principal components (as linear combinations). This also means the principal components cannot really be explained.

Upon completing this project, we've also considered opportunities for improvement. The following are some items that we would do differently:

- **Improve combination of datasets. Upon pulling the datasets from the Spotify API, we combined two datasets. The first being only song data, and the second being song feature data. The combining of datasets were contingent on the track ID and this caused significant lag due to the massive size of datasets.**
- **Smaller issue but the format of the original dataset was in a JSON file. Would have preferred to have exported in CSV.**
- **Lastly, as an improvement to the recommendation system, we could analyze and recommend playlists & songs based off Spotipy feature data.**

Sources Used

https://www.statology.org/principal-components-regression-in-python/

https://towardsdatascience.com/extracting-song-data-from-the-spotify-api-using-python-b1e79388d50

https://towardsdatascience.com/part-iii-building-a-song-recommendation-system-with-spotify-cf76b52705e7
https://github.com/enjuichang/PracticalDataScience-ENCA/tree/main/notebooks