Imports

```
import warnings
warnings.filterwarnings('ignore')
from plotnine import *

from sklearn.decomposition import PCA
import pandas as pd

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Ridge, Lasso,
ElasticNet
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.mixture import GaussianMixture

from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score, silhouette_samples


from sklearn.model_selection import GridSearchCV
from sklearn import decomposition, datasets
from sklearn.pipeline import Pipeline
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, confusion_matrix,
plot_confusion_matrix, f1_score, recall_score, plot_roc_curve,
precision_score, roc_auc_score


import numpy as np

from sklearn.linear_model import LogisticRegression # Logistic
Regression Model
from sklearn.preprocessing import StandardScaler #Z-score variables
from sklearn.metrics import accuracy_score, confusion_matrix,
plot_confusion_matrix,\
 f1_score, recall_score, plot_roc_curve, precision_score,
roc_auc_score
```

```
data =
pd.read_csv("https://raw.githubusercontent.com/anoushkasarma/DataScien
ceFinal/main/dataset.csv")

data = data.dropna()
```

Q1) Which predictors between duration, energy, loudness, liveness, and tempo determine a higher score of danceability? (Linear Regression Model)

Linear Regression Model:

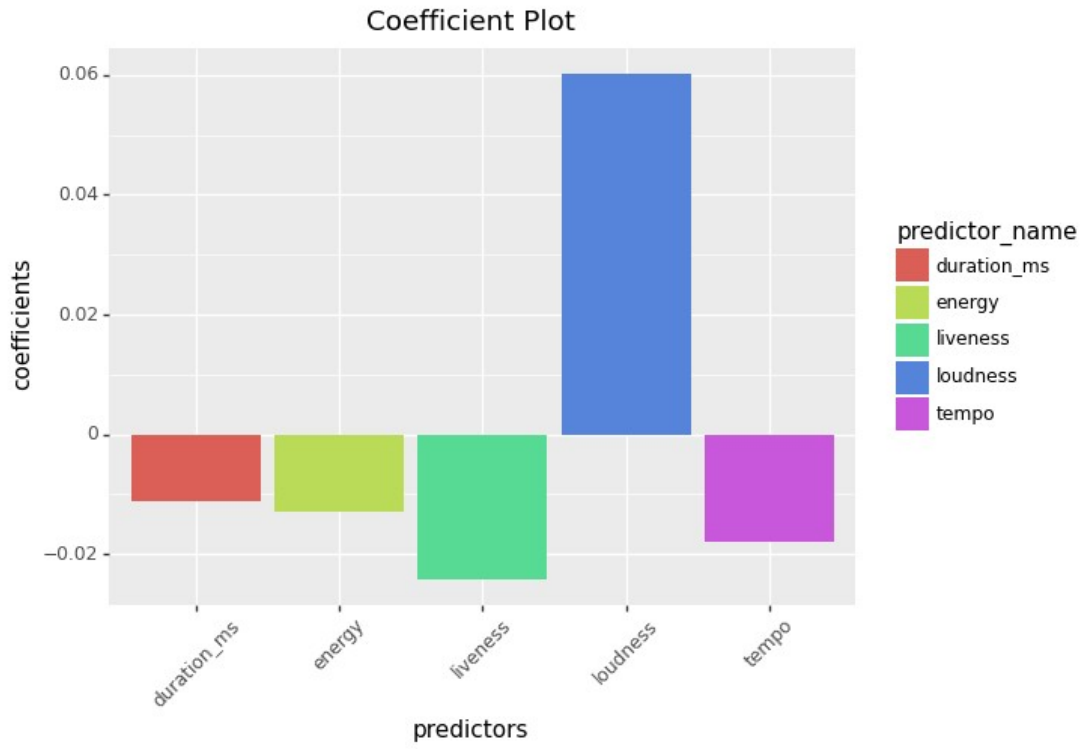
```
predictors = ["duration_ms","energy","liveness","loudness","tempo"]

X_train, X_test, y_train, y_test = train_test_split( data[predictors],

data["danceability"], test_size = 0.2)

z = StandardScaler()

X_train[predictors] = z.fit_transform(X_train[predictors])
X_test[predictors] = z.transform(X_test[predictors])

model = LinearRegression()
model.fit(X_train, y_train)

LinearRegression()

predictions = model.predict(X_test)

r2_test = r2_score(y_test, model.predict(X_test))
r2_train = r2_score(y_train, model.predict(X_train))

print(r2_test)
print(r2_train)

0.11538094647842834
0.10774180000548983

MAE_test = mean_absolute_error(y_test, model.predict(X_test))
MAE_train = mean_absolute_error(y_train, model.predict(X_train))
print(MAE_test)
print(MAE_train)

0.13270828452432362
0.13234695572881935
```

Plots/Graphs:

```
#coefficient plot
coef = pd.DataFrame({"coefficients": model.coef_,"predictor_name":
predictors})
coef
```

```
   coefficients predictor_name
0     -0.011243    duration_ms
1     -0.012954         energy
2     -0.024244       liveness
3      0.060157       loudness
4     -0.018017          tempo
```

```
(ggplot(coef, aes(x = "predictor_name", y = "coefficients", fill =
"predictor_name")) +
 geom_bar( stat = "identity")) + labs(title = "Coefficient Plot", x =
"predictors",y = "coefficients") + theme(axis_text_x =
element_text(angle = 45))
```




```
<ggplot: (8777361230132)>
```

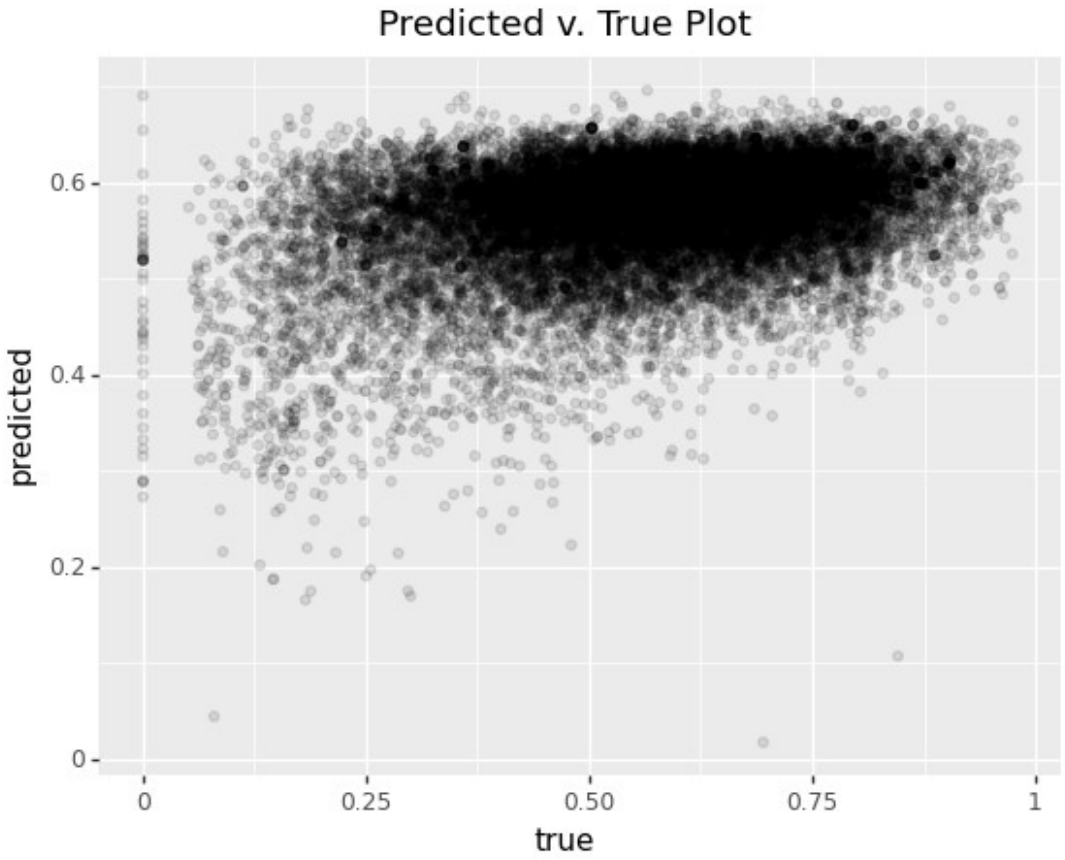
```
true_vs_pred = pd.DataFrame({"predicted": predictions,
                             "true": y_test})
```

```
true_vs_pred
```

```
        predicted   true
81428    0.620736  0.687
61844    0.591481  0.574
```

```
55044    0.582729  0.560
68004    0.620433  0.705
96847    0.575586  0.747
...           ...    ...
38644    0.540505  0.237
37093    0.625612  0.545
95755    0.597521  0.619
1536     0.503865  0.389
109668   0.492482  0.720

[22800 rows x 2 columns]
```

```
(ggplot(true_vs_pred, aes(x = "true", y = "predicted")) +
geom_point(alpha = .1) + labs(title = "Predicted v. True Plot"))
```




```
<ggplot: (8777361229300)>
```

Analysis:

In our Linear Regression Model, we are predicting the continuous value of danceability in terms of the following predictor variables: duration, energy, loudness, liveness, and tempo. When we ran the model, we were able to see that our only loudness has the only positive (as one variable goes up in value so does the variable we are trying to predict) relationship with danceability. We can see that the magnitude (how drastic the coefficient is) is .06. This

tells us that there is a really weak, positive correlation between loudness and danceability. We can see that all the other variables had negative relationships and that the variables of tempo and liveness had the strongest magnitude in terms of negatively correlating with a song's danceability. When looking at the validity of our model, we have an mean absolute error score and an r^2 score as a form of metrics. The mean absolute error tells us differences between the values we trained to fit our model versus the values we projected onto new data with our model. Mean absolute errors takes the residual (difference between the actual versus the predicted value) and then averages those errors. The r^2 score tells us the variance (how spread apart the data is starting from the mean or average value in a dataset) between the training and test sets in our model. They both scale from a value to 0 to 1. However, for r^2 a score closer to 1 is better; whereas, a score of 0 is better for MAE. For our R^2 scores, we got values of .10 for our test set and .11 for our training set. This tells us that our model fits itself a little bit better to how we trained it, so when exposed to new data, it isn't as accurate. However, with this little of a difference our model still fits pretty well. We can see that for our MAE, we have .13 as our scores. This is pretty good as a threshold of 0.5 tells us the main difference if a model is going to do well when looking at these scores. From these metrics, we can see that the model does a good job of predicting danceability while showing us that there is a low spread of variation in our model. Looking over at our true/predicted plot, we can re-verify our model does a good job because looking at this plot, there is not a big splatter of points. It's relatively dense and is going along the top of the graph. In conclusion, we can say that if we want a score to have a high score of danceability, that we should increase the loudness of a song and make sure to decrease the values of variables of all the other variables but especially the tempo and liveness in these songs as they had the strongest negative correlation.

Changes to original analysis:

For this model, the only thing we had to edit in terms of our original analysis plan was adding in our r^2 score as a metric. If we only looked at the MAE, we only would have seen if our model works. The r^2 value allows us to see the variation in our model based on the baseline model(mean) to our regression model.

Q2)When we look at predictors such as danceability, popularity and energy, what clusters are we able to get out of them? (Clustering Model)

Clustering Model

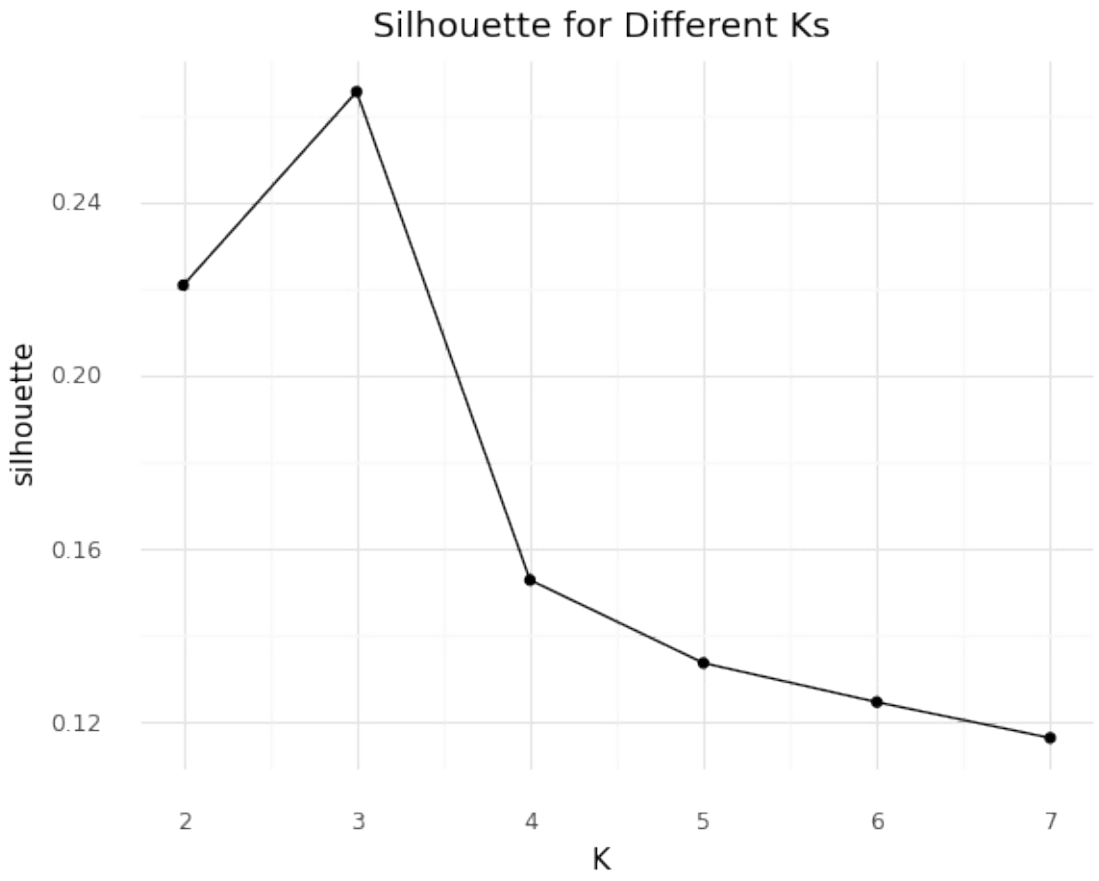
Dissects the features for our model

```
features = ["danceability","energy","popularity"]

clustering_features = data[features]

z = StandardScaler()
```

```
clustering_features[features] =
z.fit_transform(clustering_features[features])
```

tests for best k-value for clustering

```
n_components = [2,3,4,5,6,7]

sils = []

for n in n_components:
  gmm = GaussianMixture(n_components=n)
  gmm.fit(clustering_features[features])

  colName = "assignments" + str(n)
  clusters = gmm.predict(clustering_features[features])

  clustering_features[colName] = clusters

  sils.append(silhouette_score(clustering_features[features],
clusters))

sil_df = pd.DataFrame({"K": n_components,
                       "silhouette": sils})

(ggplot(sil_df, aes(x = "K", y = "silhouette")) + geom_point() +
geom_line() +
theme_minimal() +
labs(title = "Silhouette for Different Ks"))
```




```
<ggplot: (8776741421879)>
```

Final Model

```
gmm = GaussianMixture(n_components = 3)
gmm.fit(clustering_features[features])
clusters = gmm.predict(clustering_features[features])
clustering_features["cluster"] = clusters
```

Silhoutte Scores/ Metrics:

```
print(silhouette_score(clustering_features[features], clusters))
```

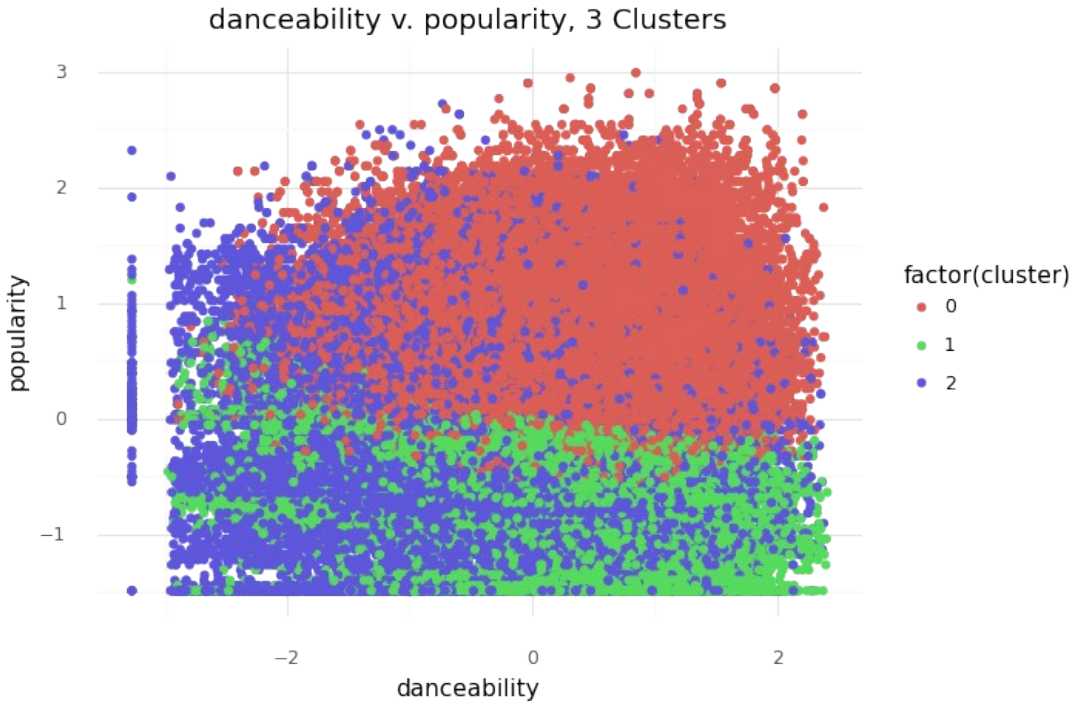
```
0.2655539257831771
```

For these values, I chose to have a Gaussian Mixture model to cluster our variables of danceability, popularity, and energy. It's a clustering model that uses probability distributions in order to make clusters. These models are really good for high dimensional data (data with lots of variables and points) and looking at overlapping clusters, which looking at this from a face value perspective is relatively something you would think would occur with variables such as if a song is danceable, popular, and has a lot of energy. To begin, we have to pick a set amount of clusters for the model to work. We use silhouette scores, which tells us the difference between separation(how far apart each cluster is)and

cohesion(how close the points in a cluster are to each other). A perfect silhouette score is 1, which tells us that our clusters are super cohesive and separated. We run our model to see which amount of clusters we choose to separate to give us the best silhouette score. We can see from our plot of different variables ranging from 2-7(these values chosen because we can usually around this many clusters if it'll be a good solution). We can see that 3 clusters or Ks is the best because it has the highest silhouette score. We also can see right after a k-value of 3 that there is a steep plunge of the silhouette score decreasing. Comparing our features to each other, in terms of scoring we see that popularity has a range of -2 to 3, danceability has a range of -4 to 2, and energy has a range of -3 to 2. This stems from standardizing them with z-scores. In our plots, we can see on the side it says "factor(cluster)" that mostly tells us how we separated our categorical variables(the categories being the 3 clusters we chose) into those three colors. Generally looking at the data, we can see that our silhouette score is .27 which is not good because the normal threshold of a good silhouette score is above .5. This means that our model has a bit of trouble predicting models, so looking at these cluster plots, we would want to look at them with a grain of salt.

(Danceability v. Popularity)

```
(ggplot(clustering_features, aes(x = "danceability", y = "popularity",
color = "factor(cluster)")) + geom_point() +
theme_minimal() + labs(title = "danceability v. popularity, 3
Clusters"))
```

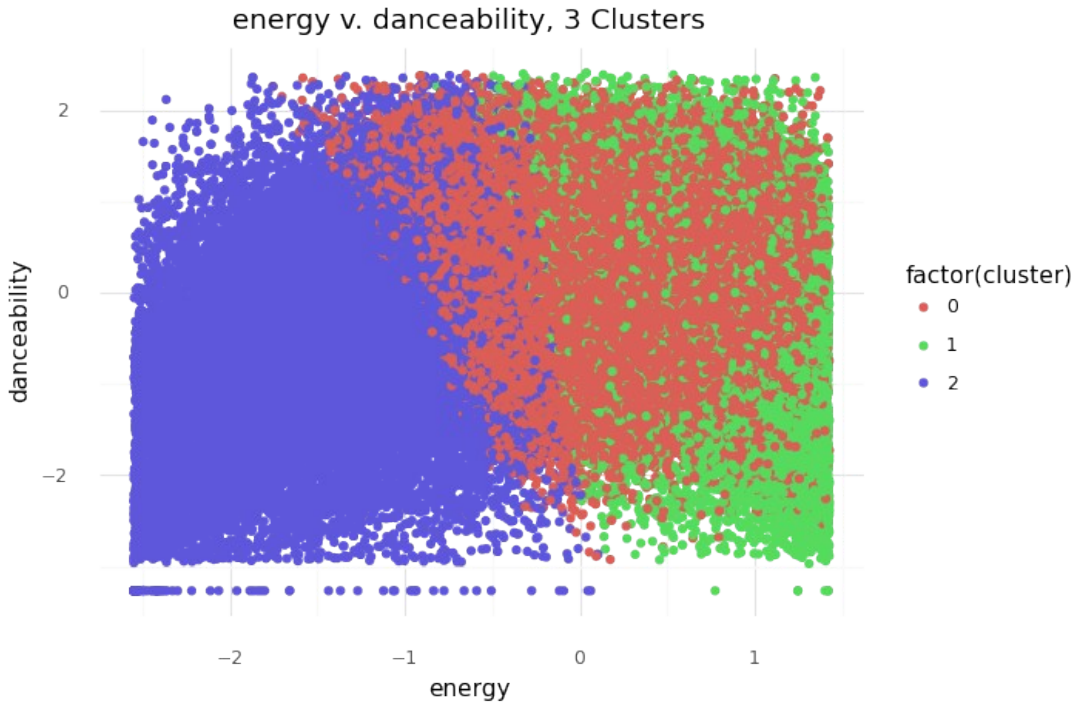



```
<ggplot: (8760850595048)>
```

For this plot, we can see that in our looking at our red cluster it's a very wide spread of data that tells us that there is a big spread of music that has higher scores of popularity with higher scores of danceability. For the blue values we can see the values bleed everywhere onto our plot, However, we can see a bigger distinction of values but more on the edge of lower popularity and lower danceability. Then, the green clusters show us a cluster where even though they are danceable songs, they have a negative score when it comes to popularity. Looking at this I would say the red cluster represents more pop songs, the blue cluster represents indie/alternative songs, and green songs would represent kids bop probably. These assumptions stem from a general Bayesian perspective of music of looking at general genres and then looking at the clusters based off of these scores.

(Energy v. Danceability)

```
(ggplot(clustering_features, aes(x = "energy", y = "danceability",
color = "factor(cluster)")) + geom_point() +
theme_minimal() + labs(title = "energy v. danceability, 3 Clusters"))
```



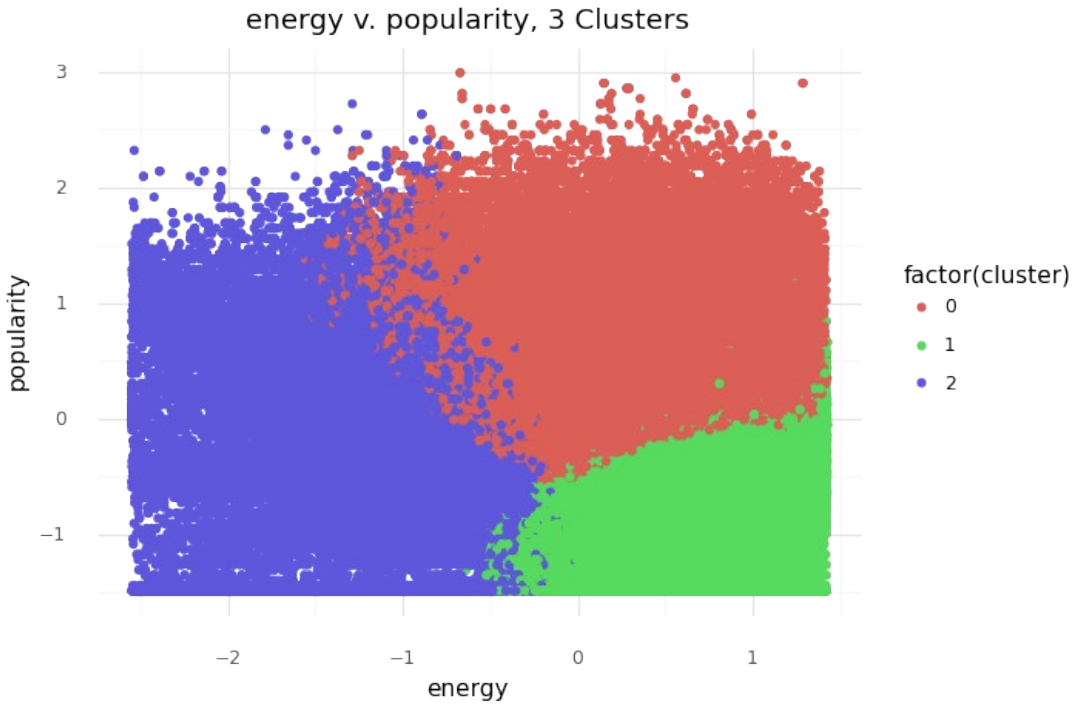

```
<ggplot: (8760849510723)>
```

This plot is super interesting because it has more of a distinct look to its clusters than our danceability v. popularity cluster. We can see from these variables that the blue cluster mostly ranges from low scores of energy and ranges in all the scores of danceability. The red cluster spreads itself across mostly from about -1 to 2. in terms of energy and ranges from all the values of danceability. Then we can see our green cluster ranges from 0 to 2 in terms of energy and keeps the same amount of danceability. When looking at these we can see that these types of songs are telling us that the extension of these songs differ mostly in terms of their energy. None of the clusters really have a set value of range in terms of

danceability but in terms of energy there are definite visible ranges in the model. The green and blue clusters do a pretty good job of looking clumped together and not as spread out as the red cluster is in their data points. If we wanted to compare these to song genres for instance we could say that slow songs you would listen to at a dance would be the blue cluster, the red cluster would be music that isn't too low or high in energy, and the green cluster are songs that are pretty amped up.

(Energy v. Popularity)

```
(ggplot(clustering_features, aes(x = "energy", y = "popularity", color
= "factor(cluster)")) + geom_point() +
theme_minimal() + labs(title = "energy v. popularity, 3 Clusters"))
```




```
<ggplot: (8760853376491)>
```

From the beginning of this plot we can see really clear distinctions between the clusters aside from the borders of the blue and red clusters. This plot tells us that the blue cluster ranges from the energy values of -3 to about -0.5 and popularity ranges from -2 to about 1.5 with some outliers in the cluster. Then, we see our red cluster roughly ranging from 0 to 1 in terms of energy, with a light diagonal veering to the right from -1 to about 0. Then, popularity ranges from about -0.5 to 2 with some outliers. Our green cluster then has energy ranging from -0.5 to 2 and popularity from -2 to roughly 0. Looking at the blue cluster we can assume these are songs that could be classics and considered soothing because those types of songs don't have a lot of energy such as classics. The red cluster would represent songs that have more popularity to them but still veers on the growing side of energy, this could be songs that are r and b where it can have a range of low to higher energy but still remains popular, probably someone like Beyonce. Lastly, the green

cluster would be songs that have higher energy but lower popularity; these could be techno/edm because we mostly see niche groups listen to that type of music compared to the general population.

Comparison between the different variables

Overall, when looking at the data we can see that there is a very wide range in variety of looking at how the clusters form. All of the clusters between these three variables gave us somewhat different clusters. We can see in terms of danceability it can range a lot regardless if there is a lot of energy or popularity. In terms of popularity, energy can have a slight distinction of itself from which we can see the red and green clusters and the same goes for danceability in terms of mostly the red and green clusters of that plot too. What we can pull from the plots is that we can do more investigating on popularity with variables like danceability and energy. Despite our clusters not having that well of silhouette scores, This would be super cool to look at because we do see a lot of popular artists with slow songs like Adele but obviously we know that if they produce music with more “danceability” and “energy” there’s a distinction of it being popular to the public.

Changes to Original Analysis

For this model, we reduced down the amount of variables we wanted to cluster because it would be really extensive and long to do this. It would be super impractical to have all these types of solutions here in terms of sharing findings of data.

Harshita FInal

December 14, 2022

[23]:
```
import warnings
warnings.filterwarnings('ignore')
from plotnine import *

from sklearn.decomposition import PCA
import pandas as pd

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

from sklearn import metrics
from sklearn.preprocessing import StandardScaler



from sklearn.model_selection import GridSearchCV
from sklearn import decomposition, datasets
from sklearn.pipeline import Pipeline


import numpy as np
```

[24]:
```
data = pd.read_csv("https://raw.githubusercontent.com/anoushkasarma/
↪DataScienceFinal/main/dataset.csv")
```

[25]:
```
data = data.dropna()
```

[26]:
```
data.head()
```

[26]:
```
   Unnamed: 0                track_id                 artists  \
0           0  5SuOikwiRyPMVoIQDJUgSV             Gen Hoshino
1           1  4qPNDBW1i3p13qLCt0Ki3A            Ben Woodward
2           2  1iJBSr7s7jYXzM8EGcbK5b  Ingrid Michaelson;ZAYN
3           3  6lfxq3CG4xtTiEg7opyCyx            Kina Grannis
4           4  5vjLSffimiIP26QG5WcN2K        Chord Overstreet

                                          album_name  \
0                                             Comedy
1                                   Ghost (Acoustic)
2                                     To Begin Again
3  Crazy Rich Asians (Original Motion Picture Sou…
4                                           Hold On

                  track_name  popularity  duration_ms  explicit  \
0                     Comedy          73       230666     False
1          Ghost - Acoustic           55       149610     False
2            To Begin Again           57       210826     False
3  Can't Help Falling In Love          71       201933     False
4                   Hold On            82       198853     False

   danceability  energy  …  loudness  mode  speechiness  acousticness  \
0         0.676  0.4610  …    -6.746     0       0.1430        0.0322
1         0.420  0.1660  …   -17.235     1       0.0763        0.9240
2         0.438  0.3590  …    -9.734     1       0.0557        0.2100
3         0.266  0.0596  …   -18.515     1       0.0363        0.9050
4         0.618  0.4430  …    -9.681     1       0.0526        0.4690

   instrumentalness  liveness  valence    tempo  time_signature  track_genre
0          0.000001    0.3580    0.715   87.917               4     acoustic
1          0.000006    0.1010    0.267   77.489               4     acoustic
2          0.000000    0.1170    0.120   76.332               4     acoustic
3          0.000071    0.1320    0.143  181.740               3     acoustic
4          0.000000    0.0829    0.167  119.949               4     acoustic

[5 rows x 21 columns]
```

1 Q3:Which variable is the highest predictor of popularity (supervised model)?

[27]:
```
explicit_dummy= pd.get_dummies(data,columns= ["explicit"])

predictors= ["valence", "acousticness", "instrumentalness", "explicit_False",␣
 ↪"explicit_True", "danceability", "energy", "key", "loudness", "mode",
             "speechiness", "liveness", "tempo"]
```

```
Contin = ["valence", "acousticness", "instrumentalness", "danceability",␣
 ↪"energy", "loudness",
             "speechiness", "liveness", "tempo"]

X= explicit_dummy[predictors]
y= explicit_dummy["popularity"]

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

[28]:
```
z= StandardScaler()
z.fit(X_train[Contin])


X_train[Contin]=z.transform(X_train[Contin])
X_test[Contin]= z.transform(X_test[Contin])

X_train
```

[28]:

	valence	acousticness	instrumentalness	explicit_False	\
96253	-0.196163	-0.494867	-0.503032	1	
70417	-0.161413	1.441588	-0.505483	1	
66688	0.711186	1.092785	-0.505483	1	
51391	0.695742	-0.739028	-0.505483	1	
95123	0.857906	0.587623	-0.505483	1	
…	…	…	…	…	
76821	-1.250232	1.817453	2.257574	1	
110269	-1.312009	-0.917339	2.341303	1	
103695	1.568341	0.840204	-0.505483	1	
860	-1.373786	1.901646	-0.505483	1	
15795	-1.510081	1.850529	2.431473	1	

	explicit_True	danceability	energy	key	loudness	mode	\
96253	0	0.427460	0.950305	11	0.370570	0	
70417	0	0.583241	-1.112037	5	-0.285686	0	
66688	0	1.264062	-1.652458	9	-1.635483	1	
51391	0	0.819797	0.747647	0	0.486391	1	
95123	0	1.264062	0.242990	0	0.302941	1	
…	…	…	…	…	…	…	
76821	0	-0.374524	-1.791537	10	-0.736874	1	
110269	0	-0.045653	1.125147	7	0.645051	1	
103695	0	0.358224	-0.321274	0	-0.631762	0	
860	0	-0.841867	-2.121352	6	-1.372703	1	
15795	0	-0.011035	-1.592852	10	-2.064260	1	

	speechiness	liveness	tempo

```
96253     -0.230378  2.086197 -0.971364
70417     -0.467827  0.171212 -1.607835
66688      4.618883 -0.064882 -0.402106
51391      1.515962  1.823870  0.528133
95123     -0.370388 -0.489851 -0.603445
…             …        …        …
76821     -0.458367 -0.679775  0.321024
110269    -0.164157  3.413570  0.763428
103695    -0.307005 -0.164566 -0.107604
860       -0.484855 -0.657215 -0.597174
15795     -0.386470  0.869001 -0.239862

[91199 rows x 13 columns]
```

[29]:
```
LR= LinearRegression()
LR.fit(X_train,y_train)
```

[29]:
```
LinearRegression()
```

[30]:
```
y_pred=LR.predict(X_test)
y_pred
```

[30]:
```
array([38.05710995, 27.77593967, 32.31355393, …, 31.55296625,
       31.77209428, 33.15442605])
```

[31]:
```
print("MSE for Train",mean_squared_error(y_train, LR.predict(X_train)))
print("MSE for Test",mean_squared_error(y_test, LR.predict(X_test)))
print("R2 for Train",r2_score(y_train, LR.predict(X_train)))
print("R2 for Test",r2_score(y_test, LR.predict(X_test)))
```

```
MSE for Train 485.2732753351761
MSE for Test 485.54646451202643
R2 for Train 0.025248937264740534
R2 for Test 0.021252232799070958
```

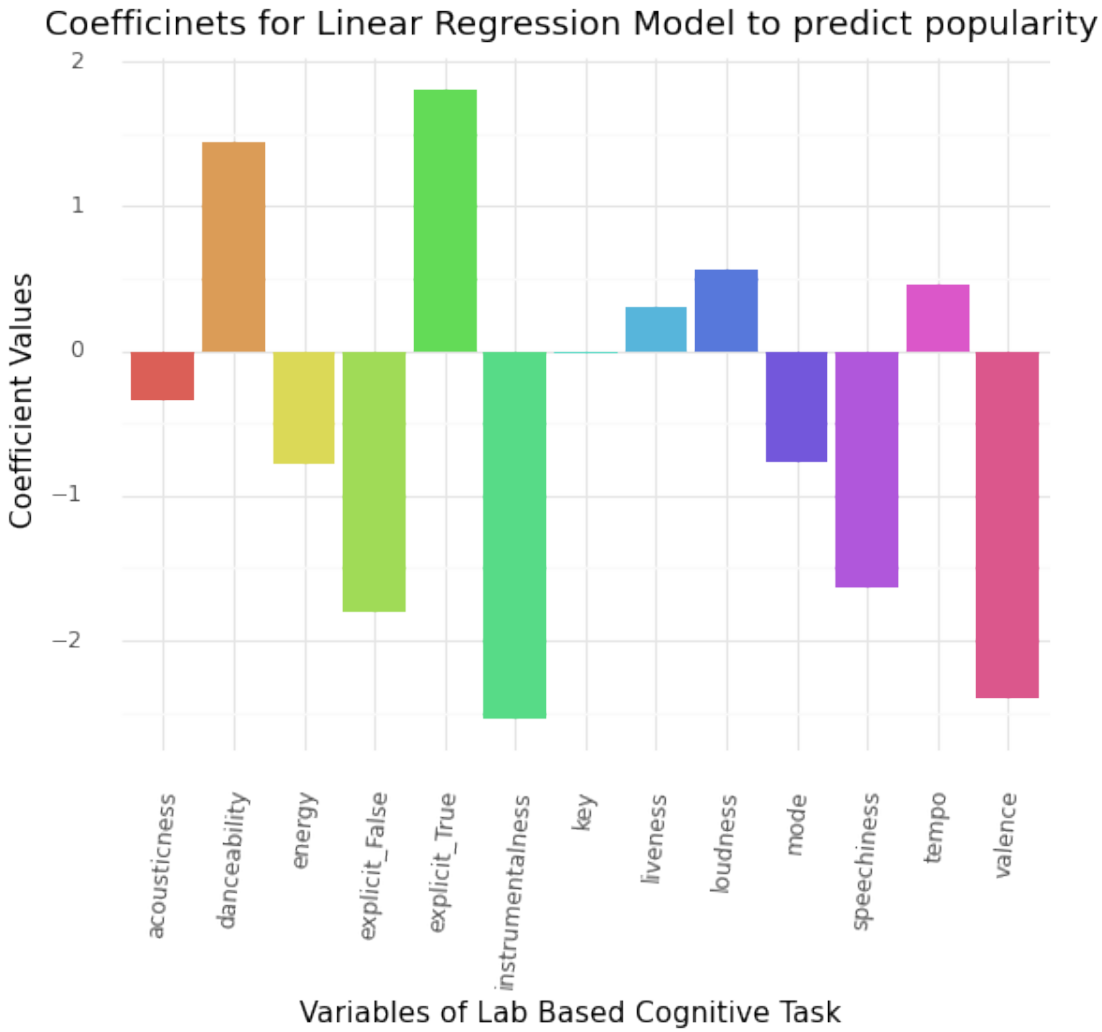
[32]:
```
coefficients= pd.DataFrame({"Coef":LR.coef_,
                            "Names":predictors})
coefficients
```

[32]:

	Coef	Names
0	-2.403115	valence
1	-0.341277	acousticness
2	-2.544561	instrumentalness
3	-1.806626	explicit_False
4	1.806626	explicit_True
5	1.443408	danceability
6	-0.780304	energy

```
7  -0.015986              key
8   0.558146         loudness
9  -0.775797             mode
10 -1.634527       speechiness
11  0.302051          liveness
12  0.463941             tempo
```

[33]:
```
(ggplot(coefficients,aes(x="Names",y="Coef",fill="Names"))+geom_bar(stat =␣
↪"identity")+
theme_minimal()+ labs(title= "Coefficinets for Linear Regression Model to␣
↪predict popularity",
                      x="Variables of Lab Based Cognitive Task",
                      y="Coefficient Values")+
theme(axis_text_x=element_text(angle =85),legend_position ="none"))
```



[33]: <ggplot: (8749112755896)>

2 Analysis

> The question we wanted to answer was what variable out of the predictors that we choose had the strongest relationship to a song's popularity. We chose 13 varibles to use as predictors where most where continous varibles and there was a few ordinals and one variable that was binary. We choose to answer this question using a supervised model specifically a Linear Regression Model because it would be able to predict a continous variable and also would be easier to show the variable relationship's with the songs popularity.

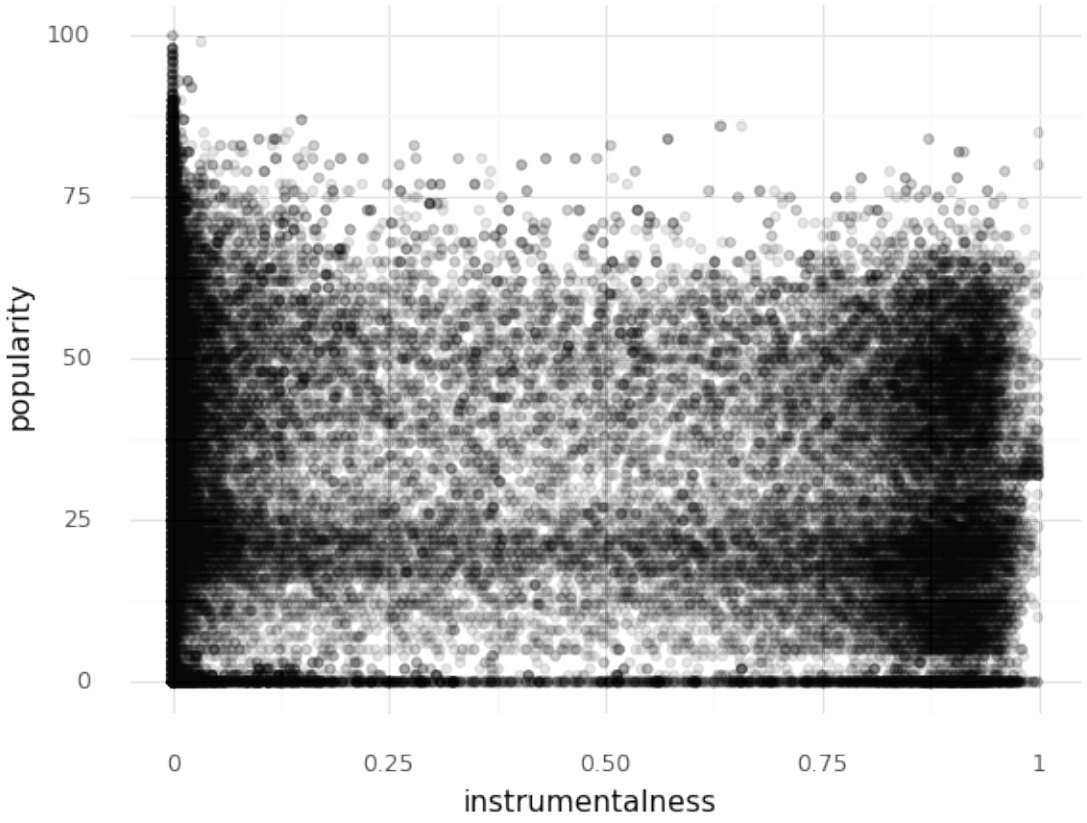
Process > First, since the variable"explicit" was a binary variable we had to create a dummy variable so that the variable would be seen as a 0 or a 1. 0 being not explicit and 1 being explicit. Then we created our linear regression model and made sure to z score our continuous varibles so that they were all in the same scale. Once the model was created we made used the MSE and R2 scores to measure the performance of our model on how it would be able to predict unseen data. The results we saw were shocking.

Results > The MSE also known as the mean squared error uses the loss function to see how well the model is performing it is essentially the sum of squared errors divided by the data point. The R2 score is measring the total variance and how different the data points are from the mean. The MSE score for the train set was 485.2 and for the test was 485.5. The R2 train score was 0.025 and the test was 0.021. The closer the MSE score is to 0 the better and the closer the R2 score is to 1 the better. Clearly looking at our results the MSE scores and the R2 scores were very bad. However the overall model was not overfit or underfit becasue of how similar the train and test scores were, meaning that the model performed well on unseen data. Looking at the specific coefficients for each of the variables there were some interesting results. To answer the question the highest predictor for popularity was if a song was explicit, "explicit_True". It had a coefficient of 1.8, being the strongest relationship to a song's popularity. Dancebility had the next strongest relationship to a song's popularity with a coefficient of 1.4. Instrumentalness on the other hand had the most negative relationship to a song's popularity with a coefficient of -2.5. These results show that a song that is explxcit has a higher chance of being popular, and that dancebility has the next strongest influence on the song's popularity. A song being explicit or not being the strongest predictor does make sense to us when we thought about it but it wasn't something that we knew from the beginning.

Change in from the Analysis Plan

> Originally this was question five and question three used only three continuous variables to see which had the stronger relationship to a song's popularity. However we decided to take that question out and then use all the more variables to see which one had the strongest relationship. Also, we later wanted to compare this non PCA Linear Regression model with the PCA Linear Regression model (something that we also changed from the analysis plan) for question five.

[34]:
```
(ggplot(explicit_dummy,aes(x="instrumentalness", y= "popularity"))+␣
↪geom_point(alpha=0.1)+ theme_minimal())
```

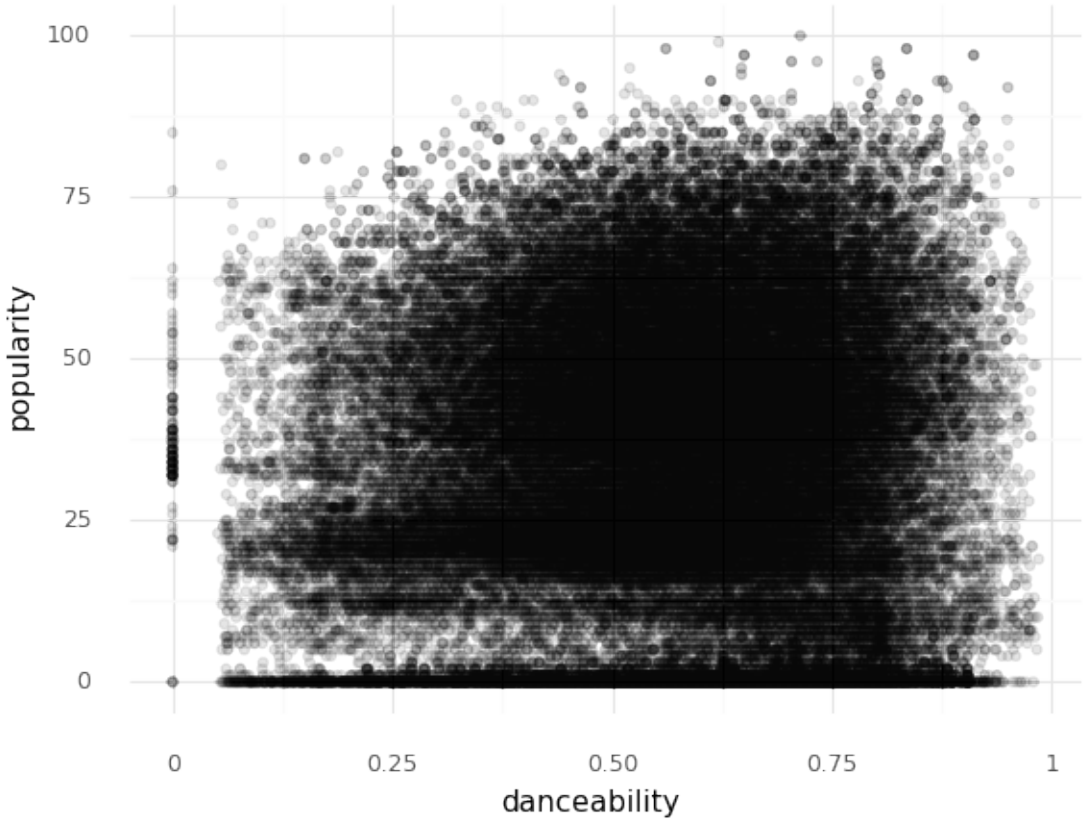


[34]: <ggplot: (8749114008974)>

3 Popularity VS Instrumentalness

The graph above shows the relationship with a song's instrumentalness and its popularity. Like mentioned previously, instrumentalness had the most negative relationship to the song's populairty. As we can see in the scatter plot above we see a slight downward trend in the popularity as the x axis instrumentalness increases. Due to how large our dataset is, there are a lot of data points that make it hard to see, but the downward trend is still visible.

[35]:
```
(ggplot(explicit_dummy,aes(x="danceability", y= "popularity"))+␣
↪geom_point(alpha=0.1)+ theme_minimal())
```



[35]: <ggplot: (8749108177634)>

4 Popularity VS Danceability

The graph above shows the relationship with a song's danceability and its popularity. Like mentioned previously, danceability had the second most positive relationship to the song's populairty. As we can see in the scatter plot above we see a slight upward trend in the popularity as the x axis danceability increases. Due to how large our dataset is, there are a lot of data points that make it hard to see, but the upward trend is still visible. Additionally compared to the Popularity VS Instrumentalness graph there is a clear difference between the downward negative relationship to a song's popularity for instrumentalness and the upward positive relationship to a song's popularity for danceability.

5 4:How can we reduce the dimensionality of the dataset when predicting the level of popularity with the continuous variables (PCA)

[36]:
```
PCA_LR= LinearRegression()

PCA_LR_X_train, PCA_LR_X_test, PCA_LR_y_train, PCA_LR_y_test =␣
↪train_test_split(X,y, test_size = 0.1, random_state=42)

PCA_LR_X_train[Contin]= z.fit_transform(PCA_LR_X_train[Contin])
PCA_LR_X_test[Contin] = z.transform(PCA_LR_X_test[Contin])

PCA = PCA()
PCA.fit(PCA_LR_X_train)
```

[36]:
```
PCA()
```

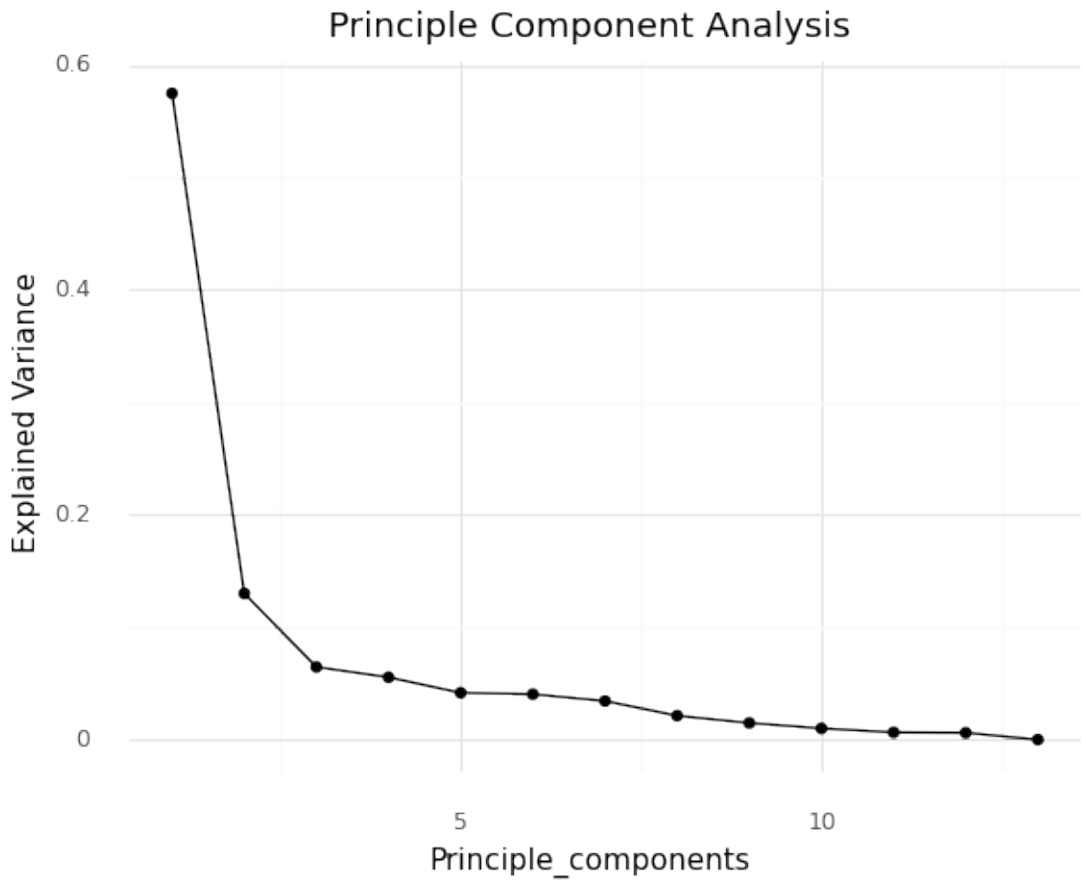
[37]:
```
PCA_LR_X_train = PCA.transform(PCA_LR_X_train)
PCA_LR_X_test = PCA.transform(PCA_LR_X_test)

PCA_LR.fit(PCA_LR_X_train, PCA_LR_y_train)

PCA_LR_y_pred = PCA_LR.predict(PCA_LR_X_test)

PCA_LR_mse = mean_squared_error(PCA_LR_y_test,PCA_LR_y_pred)
PCA_LR_r2 = r2_score(PCA_LR_y_test, PCA_LR_y_pred)
```

[38]:
```
PCA_df = pd.DataFrame({
    "Explained_Var": PCA.explained_variance_ratio_,
    "Principle_Com": range(1,14),
    "Cumulative_Var": PCA.explained_variance_ratio_.cumsum()
})

PCA_df.head()
```

[38]:

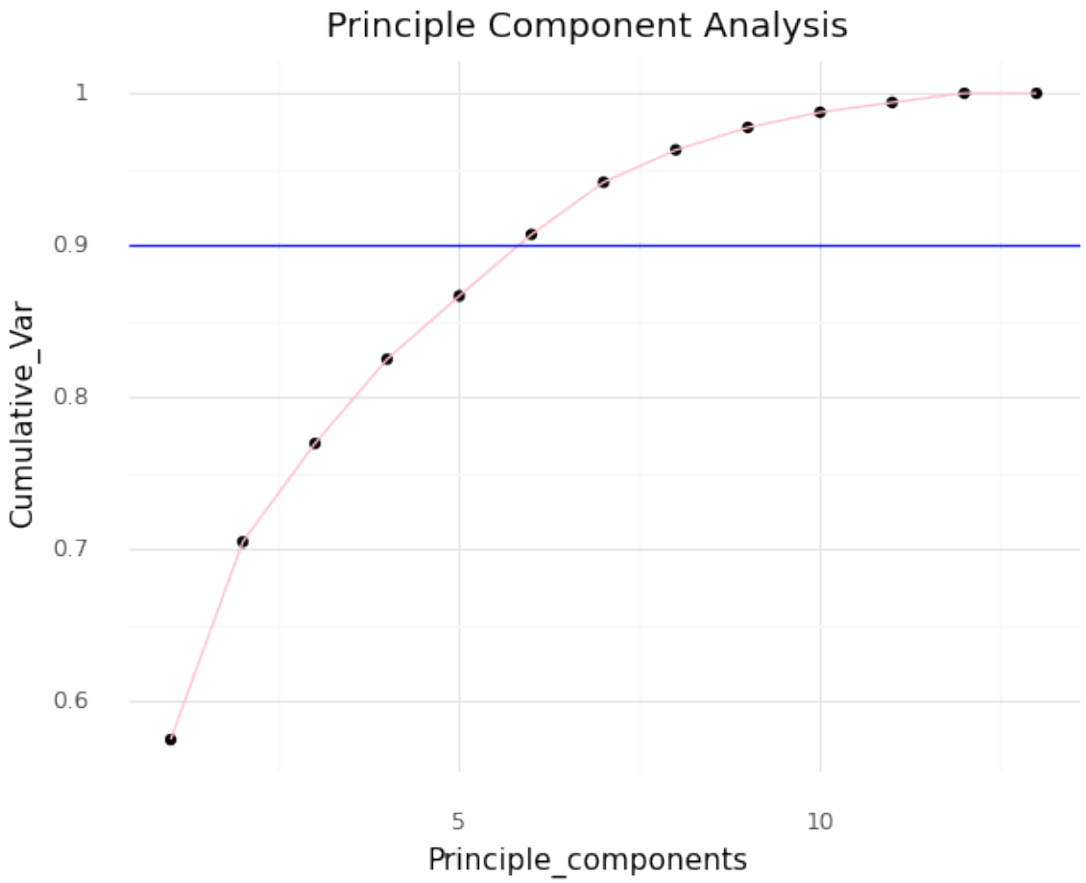
	Explained_Var	Principle_Com	Cumulative_Var
0	0.575081	1	0.575081
1	0.130065	2	0.705146
2	0.064615	3	0.769760
3	0.055438	4	0.825198
4	0.041601	5	0.866799

[39]:
```
(ggplot(PCA_df, aes(x= "Principle_Com", y= "Explained_Var"))+ geom_point()+␣
↪geom_line()+ theme_minimal()+ ggtitle("Principle Component Analysis")+␣
↪labs(x= "Principle_components", y = "Explained Variance"))
```



[39]: <ggplot: (8749107455404)>

[40]:
```
(ggplot(PCA_df, aes(x = "Principle_Com", y = "Cumulative_Var")) + geom_point()␣
↪+ geom_line(color = "pink") + geom_hline(yintercept = 0.90, color = "blue")␣
↪+ theme_minimal() + ggtitle("Principle Component Analysis") + labs(x =␣
↪"Principle_components", y = "Cumulative_Var"))
```



[40]: <ggplot: (8749113454532)>

6 Analysis

The next question is asking about how we can reduce the dimentionality of the dataset while still predicting the level of popularity with the continous variables. We essentially used the same predictors as we did in question three with a non PCA Linear Regression model and made a PCA model. We decided to use a PCA model for this question becasue it is a type of unsupervised machine learning and is a method for dimentionality reduction. Dimentionality reduction is essentially taking a large set of predictors and condensing them into a smaller set. This is useful when you have a model that you want to run efficiently and quickly. This is through having a model that works with fewer predictors which speeds up the model to make those predictions. PCA essentially take a set of variables and then creates a new more efficient set of variables for us. PCA then returns a new set of variables in order from most variation explained to the least.

Process > We started off by creating the PCA model and using the z scored data from question three. We then made a dataframe showing the explained variance and the cummulative variance,

we then created a skree plot. The skree plot is a scatter plot that shows us how much variation can be explained by adding a principle component. The x axis has the number of each component and the y axis has proportion of variance explained. Primary components are the features/ variables that we are using to predict the popularity. The closer the explained variance is to 100 percent the better because we would want the principal components to explain the model variation as opposed to the outside noise. For example if we look at the first pricipal component it has around 57% of the explained variation. This means that if we used the first principal component to build the PCA model then the PCA model would be 57 percent explained by the one predictor. The next pricipal component has a 12 percent explained variation which means that the second component would account for 12 percent of the variance in the model. Cumulative variation is what we see when we add the explained variances together in this case it was around 70 percent. Another things to observe is that skree graphs are always going to be descending becasue they go from the most variance explained to the least variance explained as seen in the graph on the left. For this specific model we wanted to make sure that the model will retain 90 percent of the original variance. I found out that having 6 principal components would allow the model to retain 90 percent of the original variance.

Results >When we ran the train and test split we realized that it was very similar to the non PCA linear regression model we did in question three. The MSE score for the train set was 485 and the test set was 480. For the R2 scores the train set was 0.024 and the testing set was 0.022. Again the train and test scores for both the MSE and R2 were very bad becasue we would want a score closer to 0 for the MSE and a score closer to 1 for the R2 however the overall model didn't perform too badly, becasue of how similar the scores were to eachother this model is not considered overfit or underfit.

Changes from the Analysis Plan > We originally had another PCA model for this question but we realized that we would not be able to run a PCA model for the question we proposed initially. We then decided to compare the model we made in question three to a PCA model to see what the results of the PCA model would be after being able to reduce the dimentionality of the model. We wanted to see if the model would perform better or worse and see which model we should use in the end if we were to use this dataset again.

Further Comparision > Once we ran the test for both the non PCA Linear Regression Model in question three and the PCA Linear Regression model in questoin four we were pleasently surprised. The results we got with the PCA model were very similar to the ones we got in the non PCA model. We were able to see that through the similarity in the MSE scores with both of them being near 485 and the R2 scores being near 0.02. Given that the PCA model's MSE scores were slightly different from eachother with the training set at 485 and the testing set at 480 this differnce is not significant enough to use the entire dataset and all the variables to come to this answer. For this I would use the PCA model beccaue it increases the performance of the model and it is computationally less extensive and would give more efficient variables, giving only six principal components to use as opppsed to the thirteen. The advantages of using the PCA model would be that it would be less computationally extensive, it reduces overfitting because of limiting the variables to just the principal components, and 90 percent of the dataset's explained variabilty is stil retained.

[41]:
```
def calc_min_pc(data_frame, col_name, threshold):
  pc_index = 0
  for pc in data_frame[ col_name]:
    pc_index += 1
```

```
    if pc>+ threshold:
      return pc_index
```

[42]:
```
min_pc = calc_min_pc(PCA_df, 'Cumulative_Var', 0.90)

min_pc
```

[42]:
```
6
```

[43]:
```
mod_train_y_pred = PCA_LR.predict(PCA_LR_X_train)

train_mod_mse = mean_squared_error(PCA_LR_y_train, mod_train_y_pred)
test_mod_mse = mean_squared_error(PCA_LR_y_test, PCA_LR_y_pred)

train_mod_r2 = r2_score(PCA_LR_y_train, mod_train_y_pred)
test_mod_r2 = r2_score(PCA_LR_y_test, PCA_LR_y_pred)
```

[44]:
```
print("PCA Model MSE (Train): " + str(train_mod_mse))
print("PCA Model MSE (Test): " + str(test_mod_mse))

print("PCA Model r2 (Train): " + str(train_mod_r2))
print("PCA Model r2 (Test): " + str(test_mod_r2))
```

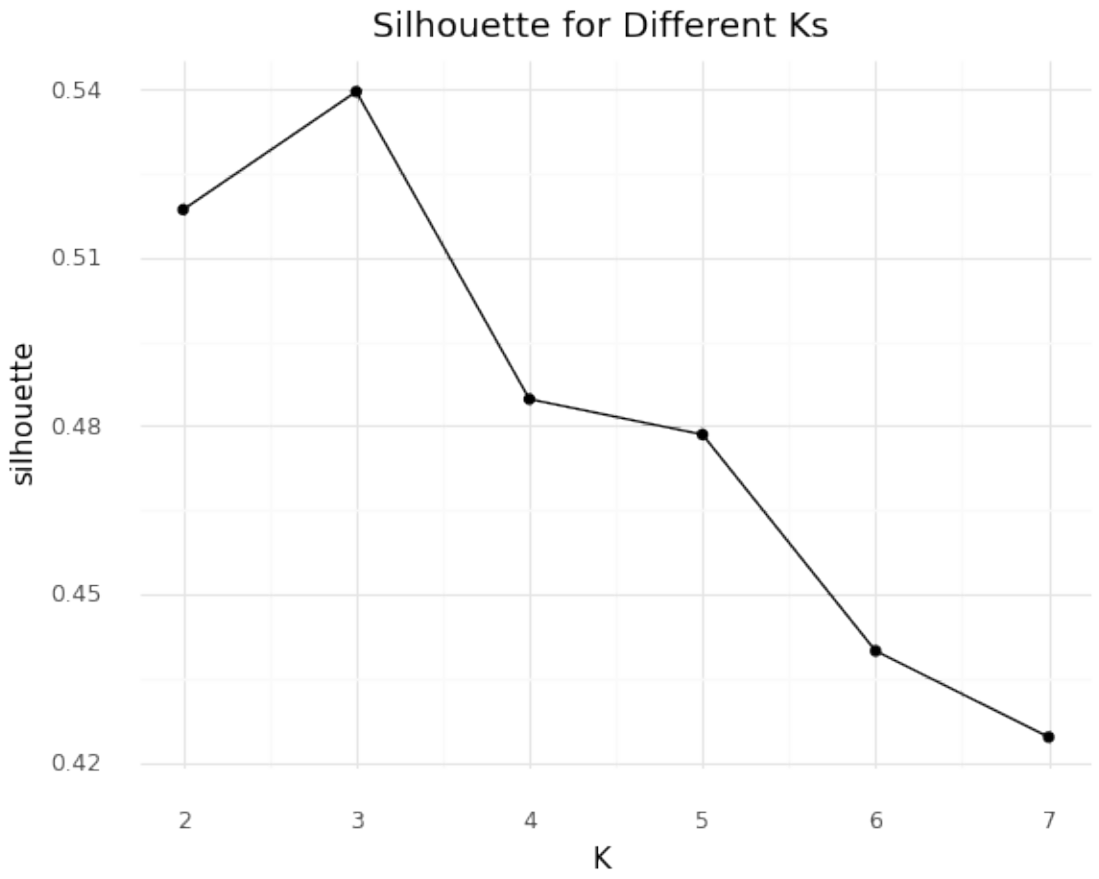
```
PCA Model MSE (Train): 485.03669560107807
PCA Model MSE (Test): 487.8829013621005
PCA Model r2 (Train): 0.024726219525965987
PCA Model r2 (Test): 0.022352209873351958
```

[ ]:
```
# doesn't show this cells output when downloading PDF
!pip install gwpy &> /dev/null

# installing necessary files
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!sudo apt-get update
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
 ↪texlive-plain-generic

# installing pypandoc
!pip install pypandoc

# connecting your google drive
from google.colab import drive
drive.mount('/content/drive')

# copying your file over. Change "Class6-Completed.ipynb" to whatever your file␣
 ↪is called (see top of notebook)
```

```
import warnings
warnings.filterwarnings('ignore')
from plotnine import *

from sklearn.decomposition import PCA
import pandas as pd

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Ridge, Lasso,
ElasticNet
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

from sklearn import metrics
from sklearn.preprocessing import StandardScaler


from sklearn.model_selection import GridSearchCV
from sklearn import decomposition, datasets
from sklearn.pipeline import Pipeline

from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture

from sklearn.metrics import silhouette_score



import numpy as np
from sklearn.linear_model import LogisticRegression # Logistic
Regression Model
from sklearn.preprocessing import StandardScaler #Z-score variables
from sklearn.metrics import accuracy_score, confusion_matrix,
plot_confusion_matrix,\
 f1_score, recall_score, plot_roc_curve, precision_score,
roc_auc_score

data =
pd.read_csv("https://raw.githubusercontent.com/anoushkasarma/DataScien
ceFinal/main/dataset.csv")

data = data.dropna()
```

#Q5) What clusters emerge when looking at the following variables: loudness, danceability, explicit, energy, and tempo (cluster)? What conclusion can we draw about the commonalities for each cluster?

```
predictors = ["loudness", "danceability", "energy", "tempo"]

X = data[predictors]

z = StandardScaler()
X[predictors] = z.fit_transform(X[predictors])

ks = [2,3,4,5,6,7]

sse =  []
sil = []

for k in ks:
  km = KMeans(n_clusters = k)
  km.fit(X[predictors])

  sse.append(km.inertia_)
  sil.append(silhouette_score(X[predictors],
km.predict(X[predictors])))

sse_df = pd.DataFrame({"K": ks,"silhouette": sil})

# sil
(ggplot(sse_df, aes(x = "K", y = "silhouette")) +
 geom_line() + geom_point() +
 theme_minimal() +
 labs(title = "Silhouette for Different Ks"))
```




```
<ggplot: (8731697269498)>
```

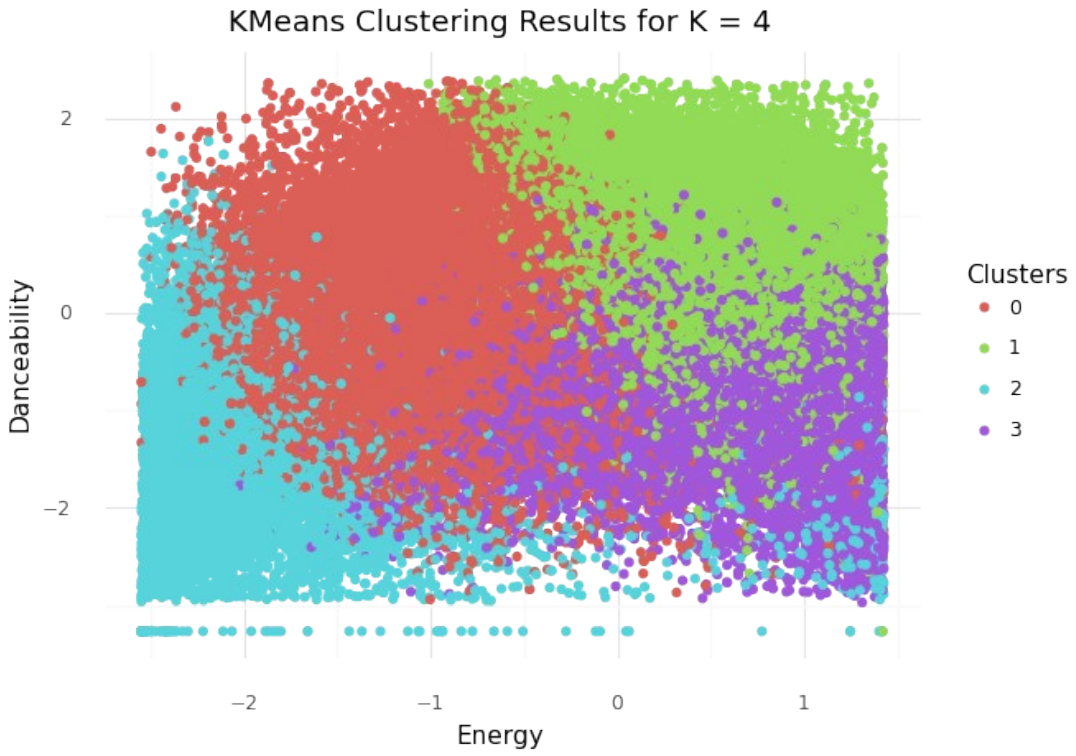
```
model = KMeans(n_clusters = 4)
model.fit(X)
```

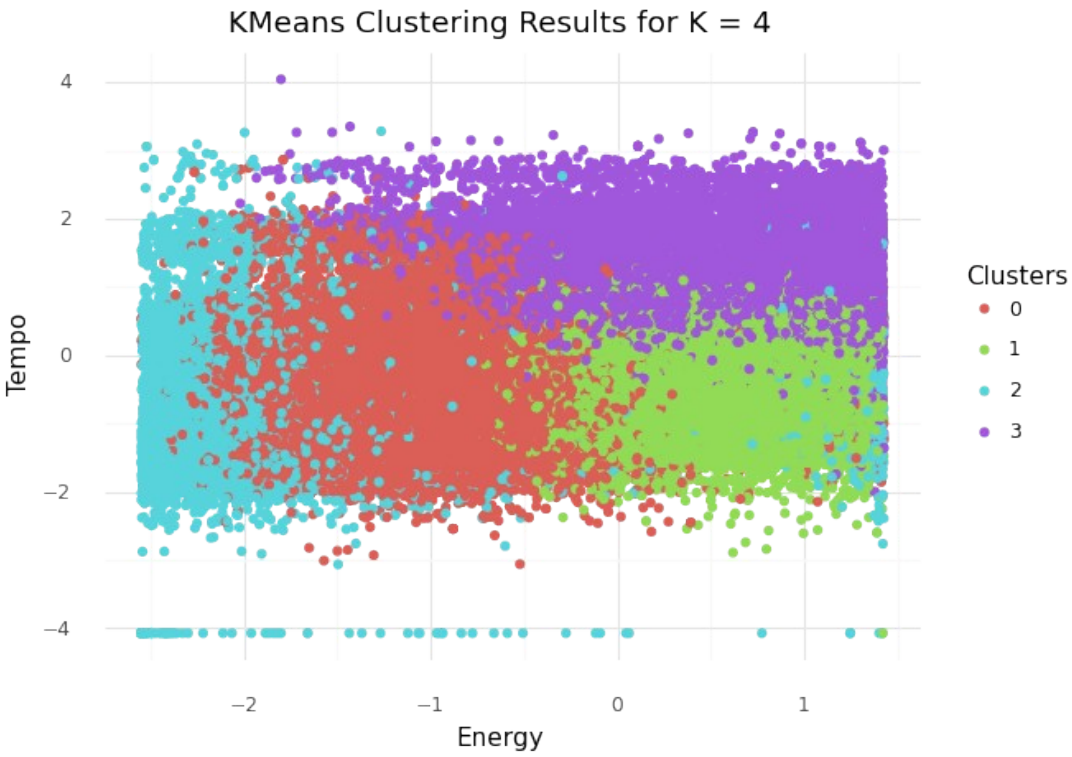
```
KMeans(n_clusters=4)
```

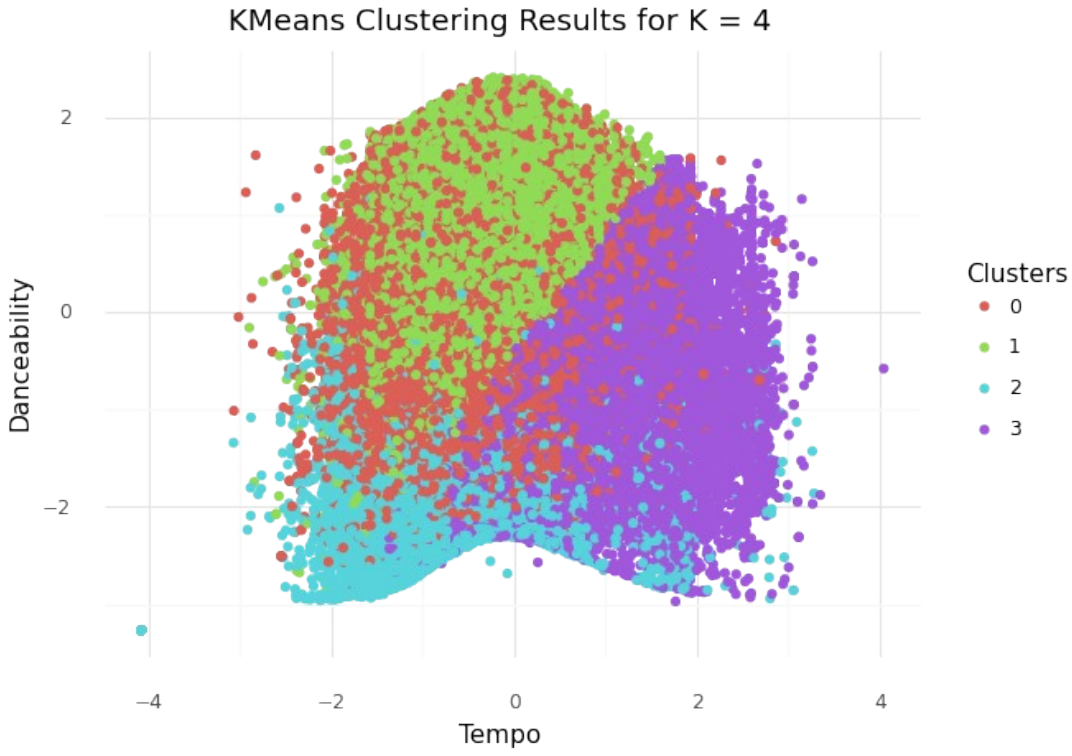
```
model = KMeans(n_clusters = 4)
model.fit(X)
X["clusters"] = model.predict(X)
```

```
print(ggplot(X, aes(x = "energy", y = "danceability", color =
"factor(clusters)" )) +
      geom_point() + theme_minimal() +
     labs(x = "Energy", y = "Danceability", title = "KMeans Clustering
Results for K = 4",
          color = "Clusters"))
print(ggplot(X, aes(x = "energy", y = "tempo", color =
"factor(clusters)" )) +
      geom_point() + theme_minimal() +
     labs(x = "Energy", y = "Tempo", title = "KMeans Clustering
Results for K = 4",
          color = "Clusters"))
print(ggplot(X, aes(x = "tempo", y = "danceability", color =
```

```
"factor(clusters)" )) +
      geom_point() + theme_minimal() +
     labs(x = "Tempo", y = "Danceability", title = "KMeans Clustering
Results for K = 4",
         color = "Clusters"))
```




KMeans Clustering Results for K = 4
Tempo
Energy
Clusters
0
1
2
3
4
2
0
−2
−4
−2
−1
0
1


KMeans Clustering Results for K = 4
Danceability
Tempo
Clusters
0
1
2
3
2
0
−2
−4
−2
0
2
4

Determining the distinct clusters will allow a category of music to emerge. Commonalities among the clusters can be drawn. For example, if a cluster has high danceability, loudness, energy, and explicit language, we can predict the cluster may be rap or hip hop. Looking at the cluster graphs can help us make these distinctions.

Energy and Danceability Graphs In general the data points should have equal danceability to energy due to the variables similarities. However, we do see some differences and can make inferences about what genre these differences could be. Purple cluster: Songs with high energy and low to medium danceability are likely faster pace songs like techno or heavy metal. Green cluster: Songs that have high danceability and high energy could be songs like pop or rap that are easy to dance to and have a lot of intensity in terms of the vocals of the artist or instruments used. Red cluster: Songs that have low energy but a medium to high danceability may be songs where the vocalist may not be as powerful of a singer but the beat is fast and the song is easy to dance to or has a catchy beat. Blue cluster: This group has lower than average danceability and lower than average energy. These may be sad songs that are hard to dance to and have low intensity. This could also be a song that is instrumental and has no words like classical music.

Energy and Tempo Graphs Purple cluster: Songs with high tempo and medium to high energy are likely faster pace songs like rap or hip hop. Green cluster: Songs that have medium tempo and higher energy could be songs meaning that the song's beats may be slower but the energy may be higher. For example, these could be songs that are slower but where the singer has high and intensely strong vocals. Red cluster: Songs that have medium to low energy but a medium tempo may be a song that is not as intense but has a medium beat that maybe we do not hear as much but is in the background. Blue cluster: This group is very similar to the red cluster but this cluster does show fairly low energy and medium to very low tempo. This is likely a slower song but maybe has strong beat but the singer is not as powerful.

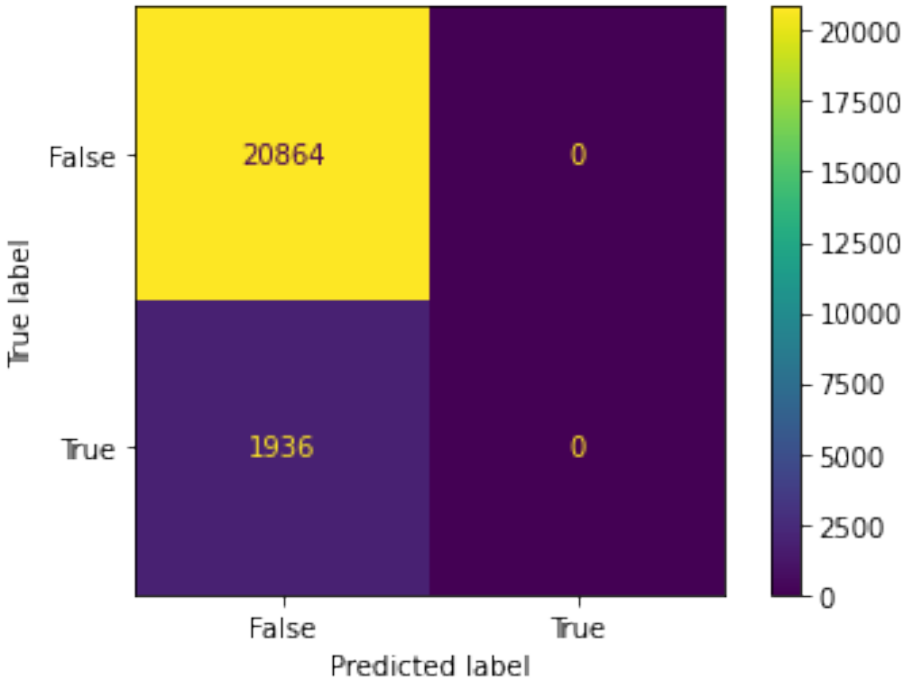
Danceability and Tempo Graphs Purple cluster: Songs with high tempo and medium danceability are likely faster pace songs like Techno but range in the ability to dance. Some techno songs are very danceable but some are not as danceable. Green cluster: Songs that have high tempo and high danceability could be songs where the song's beats may be fast and additionally the danceability is fast. A genre for this cluster may potentially be rap, pop, or hip hop. The cluster is more songs you would hear at a party or club. Red cluster: This cluster is very similar to the Green cluster, however this cluster has data points/songs that are less danceable in general. Possibly this cluster could represent songs that are faster but people would not dance to these songs like heavy metal or rock. Blue cluster: This cluster has low to high tempo and low to medium danceability. Therefore, a genre that could be present here is likely techno or goth.

#Q6) Out of the predictors used in Q5, what percentage of genres were actually predicted correctly?

```
X = data[predictors]
y = data["explicit"]

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size =
```

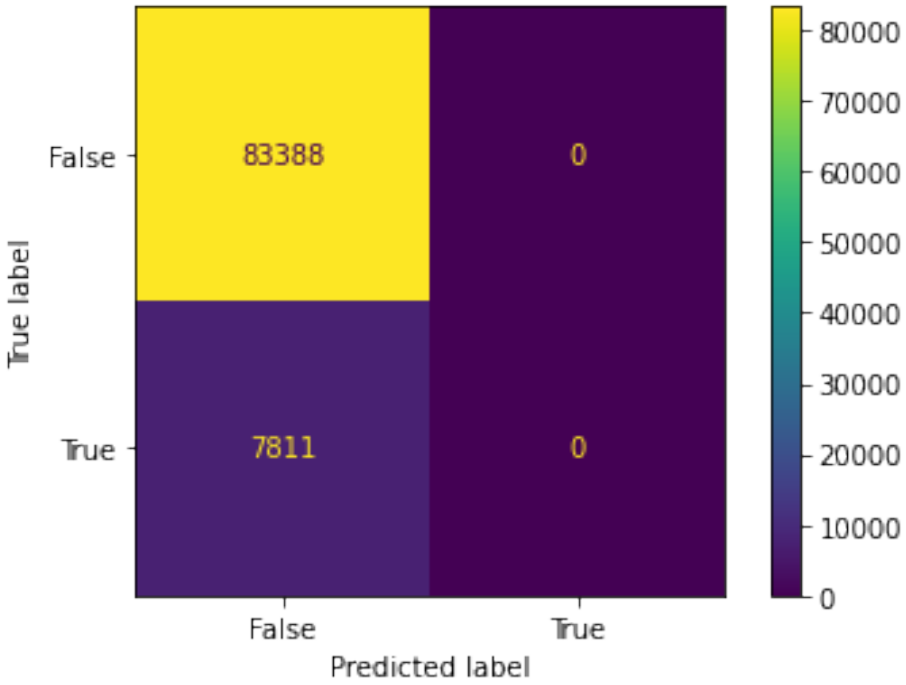
```
0.2) # put in the predictors, put in the outcome

X_train[predictors] = z.fit_transform(X_train)
X_test[predictors] = z.transform(X_test)
#create model
lr = LogisticRegression()
#fit
lr.fit(X_train, y_train)
#predict
predictedVals = lr.predict(X_test)
predictedProbs = lr.predict_proba(X_test) #get the probability of that
data

print(plot_confusion_matrix(lr, X_test, y_test))
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object
at 0x7f4f5fc53c10>
```




```
print(plot_confusion_matrix(lr, X_train, y_train))
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object
at 0x7f4f416115e0>
```

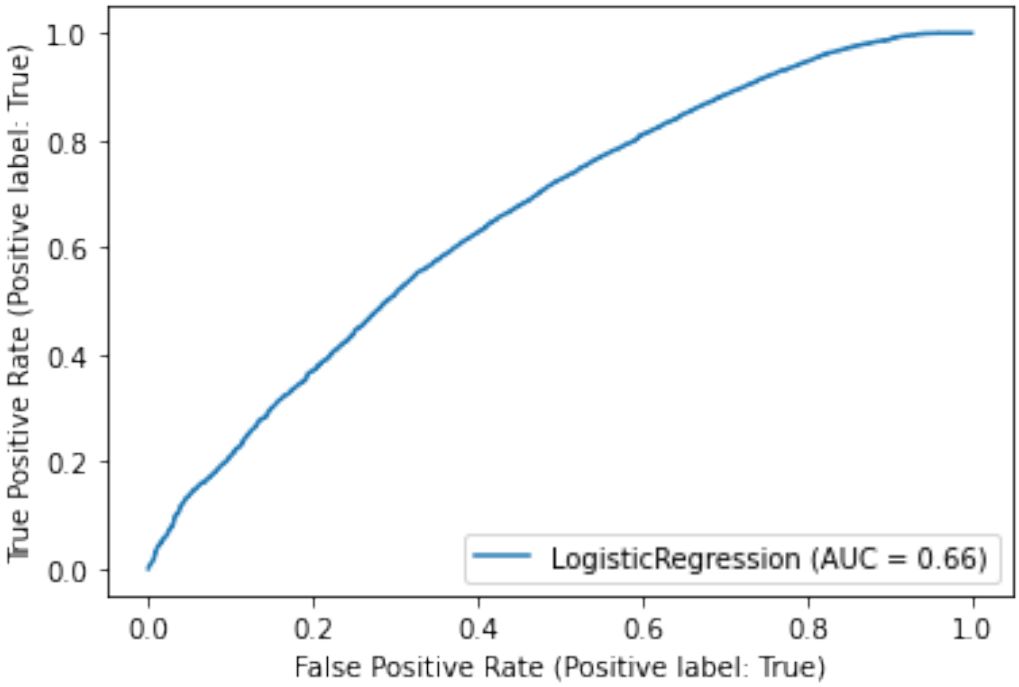


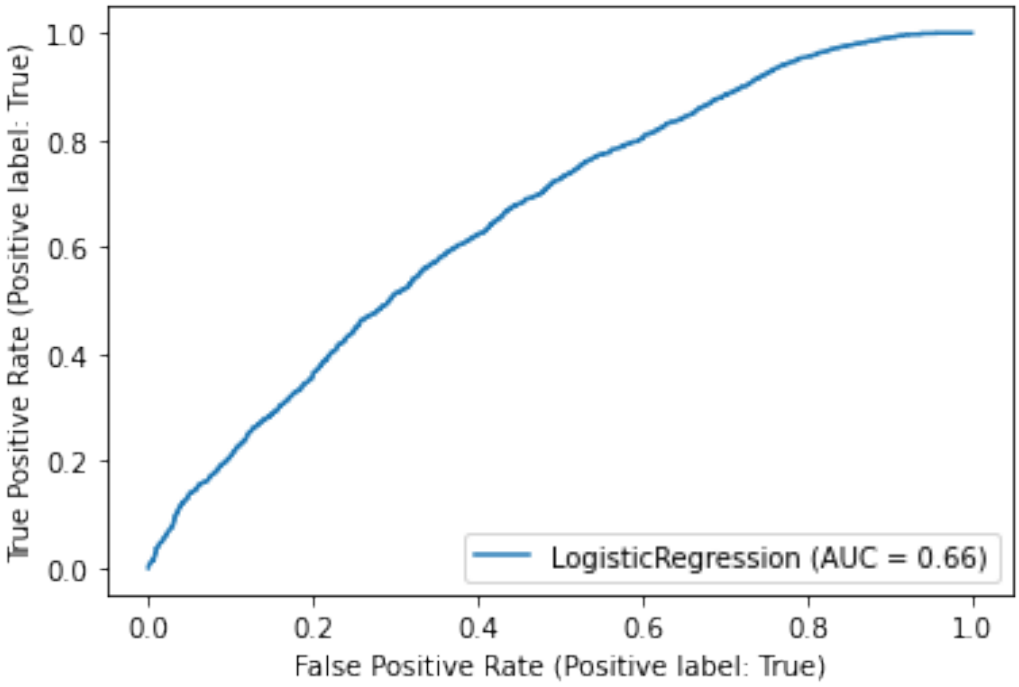

```
# metrics test
print("Testing Metrics")
print("Accuracy: ", accuracy_score(y_test, predictedVals)) #takes in
actual values and predicted values
#print("F1 Score: ", f1_score(y_test, predictedVals)) #precision +
recall
#print("Recall: ", recall_score(y_test, predictedVals))
print("Precision: ", precision_score(y_test, predictedVals)) #how
often we predict correct positives

# metrics train
print("\nTraining Metrics")
print("Accuracy: ", accuracy_score(y_train, lr.predict(X_train)))
#print("F1 Score: ", f1_score(y_train, lr.predict(X_train)))
#print("Recall: ", recall_score(y_train, lr.predict(X_train)))
print("Precision: ", precision_score(y_train, lr.predict(X_train)))
```

```
Testing Metrics
Accuracy:  0.9150877192982456
Precision:  0.0

Training Metrics
Accuracy:  0.9143521310540685
Precision:  0.0
```

```
plot_roc_curve(lr, X_train, y_train)
plot_roc_curve(lr, X_test, y_test)
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f1018b59e50>
```





High accuracy means the model performed well. A low precision score (<0.5) means your classifier has a high number of False positives. An ROC/AUC curve with the line trending more towards the upper left would mean there was better model performance. Our model was closer to flat/linear so the model performed poorly. The confusion matrix had many false positives, confirming our precision score. The model was not overfit as both the training and testing set did almost exactly as poor based on our metrics and graphs.

From the precision score, ROC curve, and Confusion Matrix, this model did very poorly and did poorly at predicting whether a song would be explicit or not.

This could be due to a multitude of genres likely to have explicit words. Based on the variables picked–loudness, tempo, danceability, and energy–a song with low loudness, low tempo, low danceability, and low energy is likely to be explicit depending on the emotions of the writer in addition to a song like rap where many songs have explicit language. Songs that would not have rap may be kids bop or disney songs in which the model still did a poor job predicting. Some disney songs may have high tempos, danceability, energy, and loudness so it would be hard to figure out the language in the song because this model does not have the ability to tell what words are explicit or not.

For example, slow and sad songs do not have high danceability, however there could be explicit words. Therefore, it makes sense why explicit would be difficult to predict.