Load Packages and Data

```
# Load packages

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns

from sklearn.utils import all_estimators
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.model_selection import train_test_split, GridSearchCV, RepeatedStrat
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.feature_selection import SelectKBest, mutual_info_regression, RFE, S
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

from sklearn.linear_model import LinearRegression, Lasso, LassoCV, LogisticRegres
```

```
spotify_data = pd.read_csv('spotify_data.csv')
```

```
spotify_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169909 entries, 0 to 169908
Data columns (total 19 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   acousticness      169909 non-null  float64
 1   artists           169909 non-null  object 
 2   danceability      169909 non-null  float64
 3   duration_ms       169909 non-null  int64  
 4   energy            169909 non-null  float64
 5   explicit          169909 non-null  int64  
 6   id                169909 non-null  object 
 7   instrumentalness  169909 non-null  float64
 8   key               169909 non-null  int64  
 9   liveness          169909 non-null  float64
 10  loudness          169909 non-null  float64
 11  mode              169909 non-null  int64  
 12  name              169909 non-null  object 
 13  popularity        169909 non-null  int64  
 14  release_date      169909 non-null  object 
 15  speechiness       169909 non-null  float64
 16  tempo             169909 non-null  float64
 17  valence           169909 non-null  float64
 18  year              169909 non-null  int64  
dtypes: float64(9), int64(6), object(4)
memory usage: 24.6+ MB
```

```
spotify_data.head()
```

	acousticness	artists	danceability	duration_ms	energy	explicit	
0	0.995	['Carl Woitschach']	0.708	158648	0.1950	0	6Kb(
1	0.994	['Robert Schumann', 'Vladimir Horowitz']	0.379	282133	0.0135	0	6K
2	0.604	['Seweryn Goszczyński']	0.749	104300	0.2200	0	6L63
3	0.995	['Francisco Canaro']	0.781	180760	0.1300	0	6M94I
4	0.990	['Frédéric Chopin', 'Vladimir Horowitz']	0.210	687733	0.2040	0	

Data Cleaning

```
numerical_cols = list(spotify_data.select_dtypes(int).columns)
dummy_cols = list(spotify_data.select_dtypes(object).columns)
```

```
print(dummy_cols)
```

```
['artists', 'id', 'name', 'release_date']
```

All of these columns will not be used in dataset

```
no_variance_cols = []
for col in spotify_data.columns:
  if len(spotify_data[col].value_counts()) == 1:
    print(f"Dropping column: {col}")
    numerical_cols.remove(col)
    no_variance_cols.append(col)

spotify_data = spotify_data.drop(no_variance_cols, axis = 1)
```

```
spotify_data['year']
```

```
0         1928
1         1928
2         1928
3         1928
4         1928
          ...
169904    2020
169905    2020
169906    2020
169907    2020
169908    2020
Name: year, Length: 169909, dtype: int64
```

```
def create_decades(row):
  curr_year = row['year']
  dec_year = curr_year - int(str(curr_year)[-1])

  return f"{dec_year}s"
```

```
spotify_data['decade'] = spotify_data.apply(lambda row: create_decades(row), axis
```

```
spotify_data['decade'].value_counts()
```

```
1960s    20000
1970s    20000
1980s    20000
1990s    20000
2000s    20000
1950s    19950
2010s    19900
1940s    14968
1930s     8889
1920s     4446
2020s     1756
Name: decade, dtype: int64
```

```
spotify_data_with_dummies = pd.get_dummies(spotify_data, columns = ['decade'])
```

```
spotify_data_with_dummies.columns
```

```
Index(['acousticness', 'artists', 'danceability', 'duration_ms', 'energy',
       'explicit', 'id', 'instrumentalness', 'key', 'liveness', 'loudness',
       'mode', 'name', 'popularity', 'release_date', 'speechiness', 'tempo',
       'valence', 'year', 'decade_1920s', 'decade_1930s', 'decade_1940s',
       'decade_1950s', 'decade_1960s', 'decade_1970s', 'decade_1980s',
       'decade_1990s', 'decade_2000s', 'decade_2010s', 'decade_2020s'],
      dtype='object')
```

```
metric_cols = list(spotify_data_with_dummies.select_dtypes(float).columns)
```

```
metric_cols
```

```
['acousticness',
 'danceability',
 'energy',
 'instrumentalness',
 'liveness',
 'loudness',
 'speechiness',
 'tempo',
 'valence']
```

```
decade_cols = [col for col in spotify_data_with_dummies.columns if 'decade' in col
```

Feature Engineering

```
X_unscaled_all = spotify_data_with_dummies.drop(dummy_cols, axis=1)
```

```
X_unscaled_all_interactions = X_unscaled_all.copy()
```

The code below creates interaction terms for every song metric + decade

```
for col in metric_cols:
  for decade in decade_cols:
    new_col = f"{col} {decade}"
    X_unscaled_all_interactions[new_col] = X_unscaled_all[col] * X_unscaled_all[de
```

```
    <ipython-input-19-d07f48d8c16e>:4: PerformanceWarning: DataFrame is highly fr
      X_unscaled_all_interactions[new_col] = X_unscaled_all[col] * X_unscaled_all
```

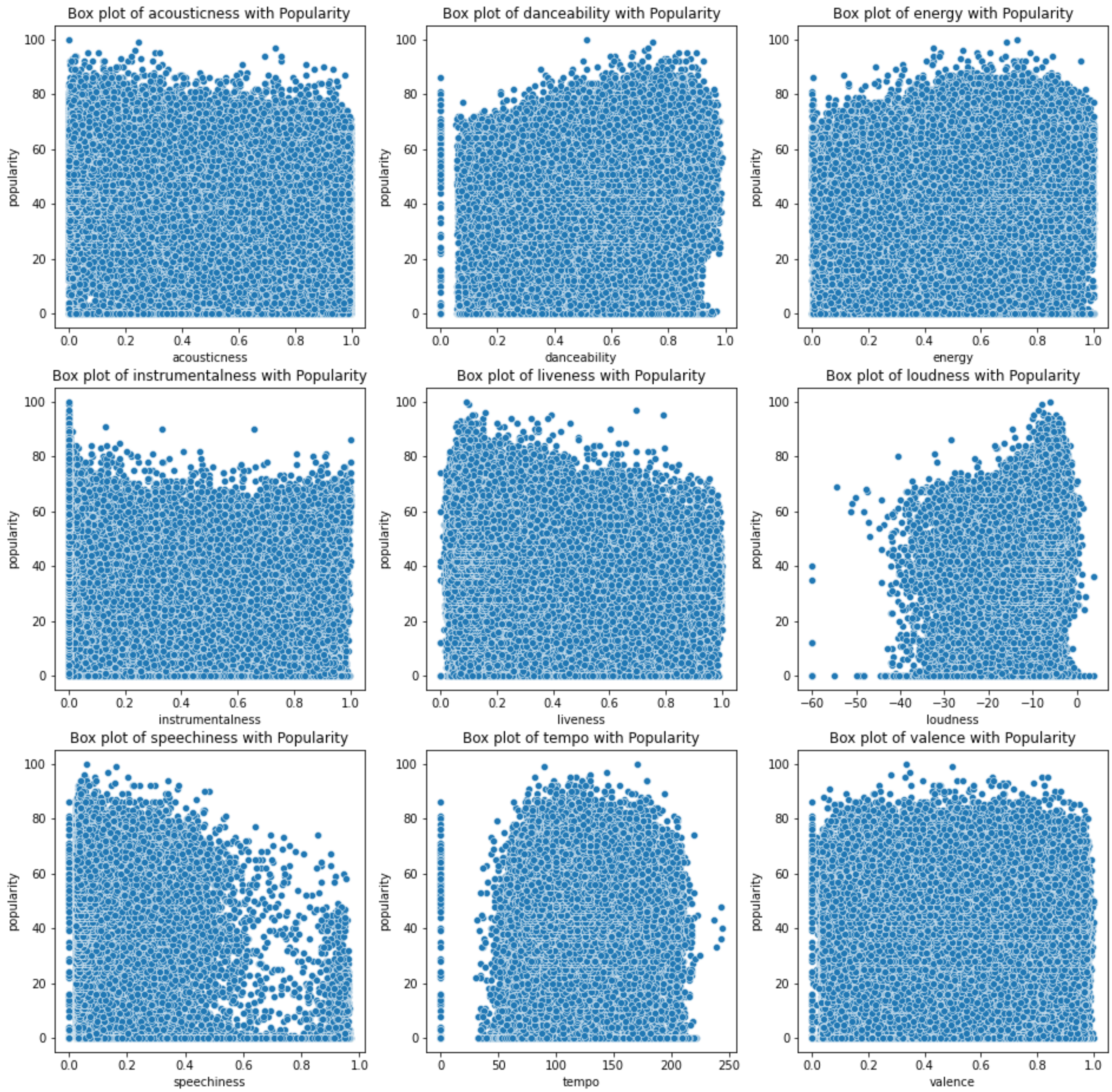
```
X_unscaled_all = X_unscaled_all.drop(decade_cols + ['year'], axis = 1)
X_unscaled_all_interactions = X_unscaled_all_interactions.drop(decade_cols + ['yea
```

EDA:

Popularity relationships

```
def scatterplots_subplot():
  plt.figure(figsize=(16,16))

  plotted = 0
  for col in metric_cols:
    plt.subplot(3,3,plotted+1)

    sns.scatterplot(y = 'popularity', x = col, data = spotify_data_with_dummies)
    plt.title(f'Scatter plot of {col} with Popularity')
    plotted +=1
```

```
scatterplots_subplot()
```


Box plot of acousticness with Popularity
popularity
acousticness
Box plot of danceability with Popularity
popularity
danceability
Box plot of energy with Popularity
popularity
energy
Box plot of instrumentalness with Popularity
popularity
instrumentalness
Box plot of liveness with Popularity
popularity
liveness
Box plot of loudness with Popularity
popularity
loudness
Box plot of speechiness with Popularity
popularity
speechiness
Box plot of tempo with Popularity
popularity
tempo
Box plot of valence with Popularity
popularity
valence

Clear outliers with tempo 0. and danceability 0- may be interesting to explore some outliers

```
spotify_data_with_dummies[ (spotify_data_with_dummies['tempo'] == 0) & (spotify_da
```

	artists	name	popularity
87456	['White Noise Meditation', 'Lullaby Land', 'Wh...	Brown Noise - Loopable with No Fade	74
87651	['Granular']	White Noise - 500 hz	81
87652	['Granular']	White Noise - 145 hz	80
87653	['Erik Eriksson', 'White Noise Baby Sleep', 'W...	Clean White Noise - Loopable with no fade	86
97541	['High Altitude Samples']	Soft Brown Noise	78
107149	['Microdynamic Recordings']	Calm Pour	76
116201	['White Noise Baby Sleep', 'White Noise for Ba...	The Early Morning Rain	71
116385	['Erik Eriksson', 'White Noise for Babies', 'W...	Pure Brown Noise - Loopable with no fade	74

```
spotify_data_with_dummies[ (spotify_data_with_dummies['danceability'] == 0) & (spo
```

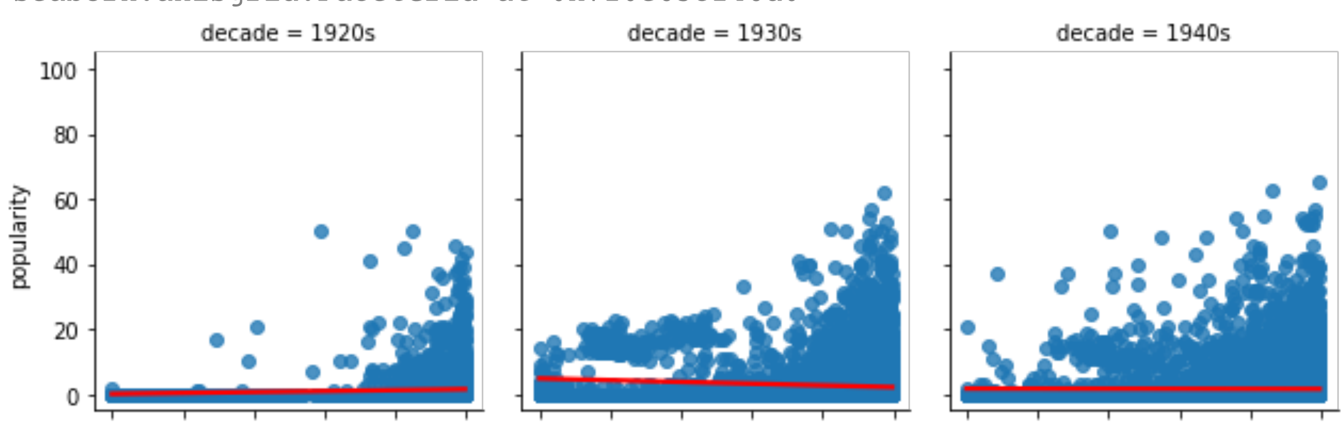
	artists	name	popularity
87456	['White Noise Meditation', 'Lullaby Land', 'Wh...	Brown Noise - Loopable with No Fade	74
87651	['Granular']	White Noise - 500 hz	81
87652	['Granular']	White Noise - 145 hz	80
87653	['Erik Eriksson', 'White Noise Baby Sleep', 'W...	Clean White Noise - Loopable with no fade	86
97541	['High Altitude Samples']	Soft Brown Noise	78
107149	['Microdynamic Recordings']	Calm Pour	76
116201	['White Noise Baby Sleep', 'White Noise for Ba...	The Early Morning Rain	71
116385	['Erik Eriksson', 'White Noise for Babies', 'W...	Pure Brown Noise - Loopable with no fade	74

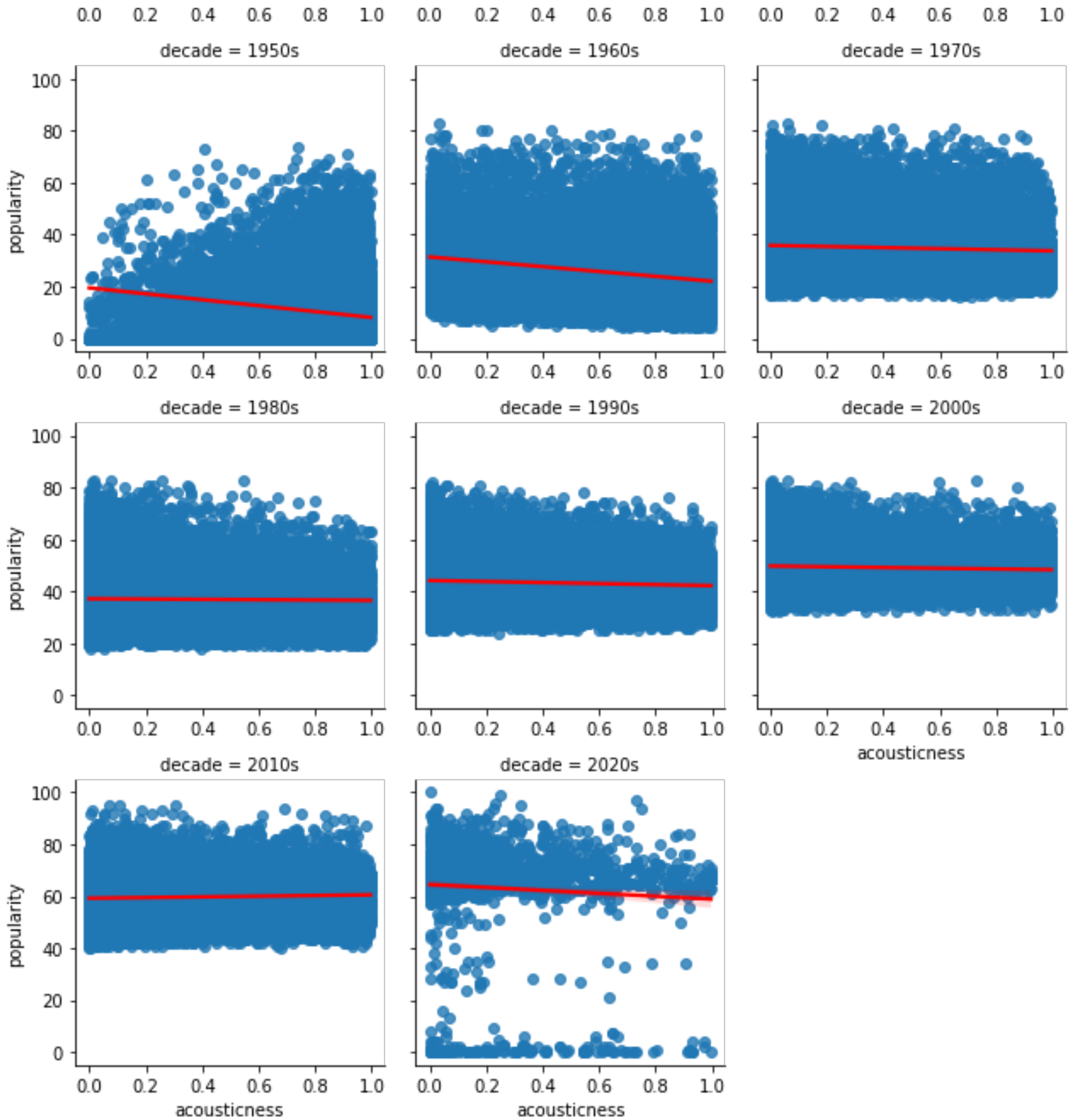
Based on this analysis, it appears that the dataset used for this project is not perfectly clean (a noisy dataset one might say), in that there we are trying to predict song popularity, but there are non-songs such as white noise and rain noises that are in the dataset, creating outlier points.

Decade-wise scatter plots of popularity relationships

```
g = sns.FacetGrid(spotify_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, 'acousticness', "popularity", line_kws = {'color':'red'})
```

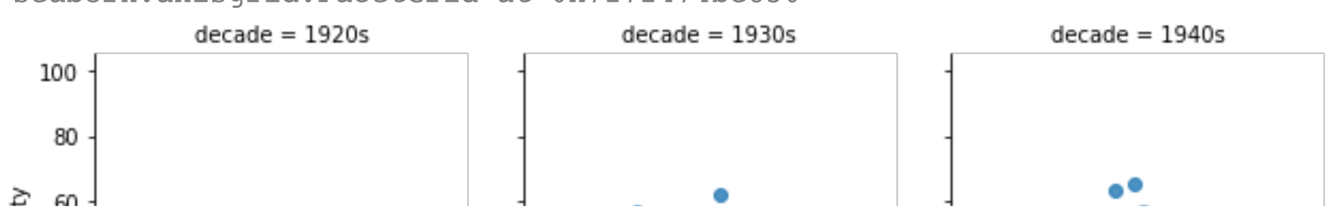
```
<seaborn.axisgrid.FacetGrid at 0x7f0e65cf46d0>
```

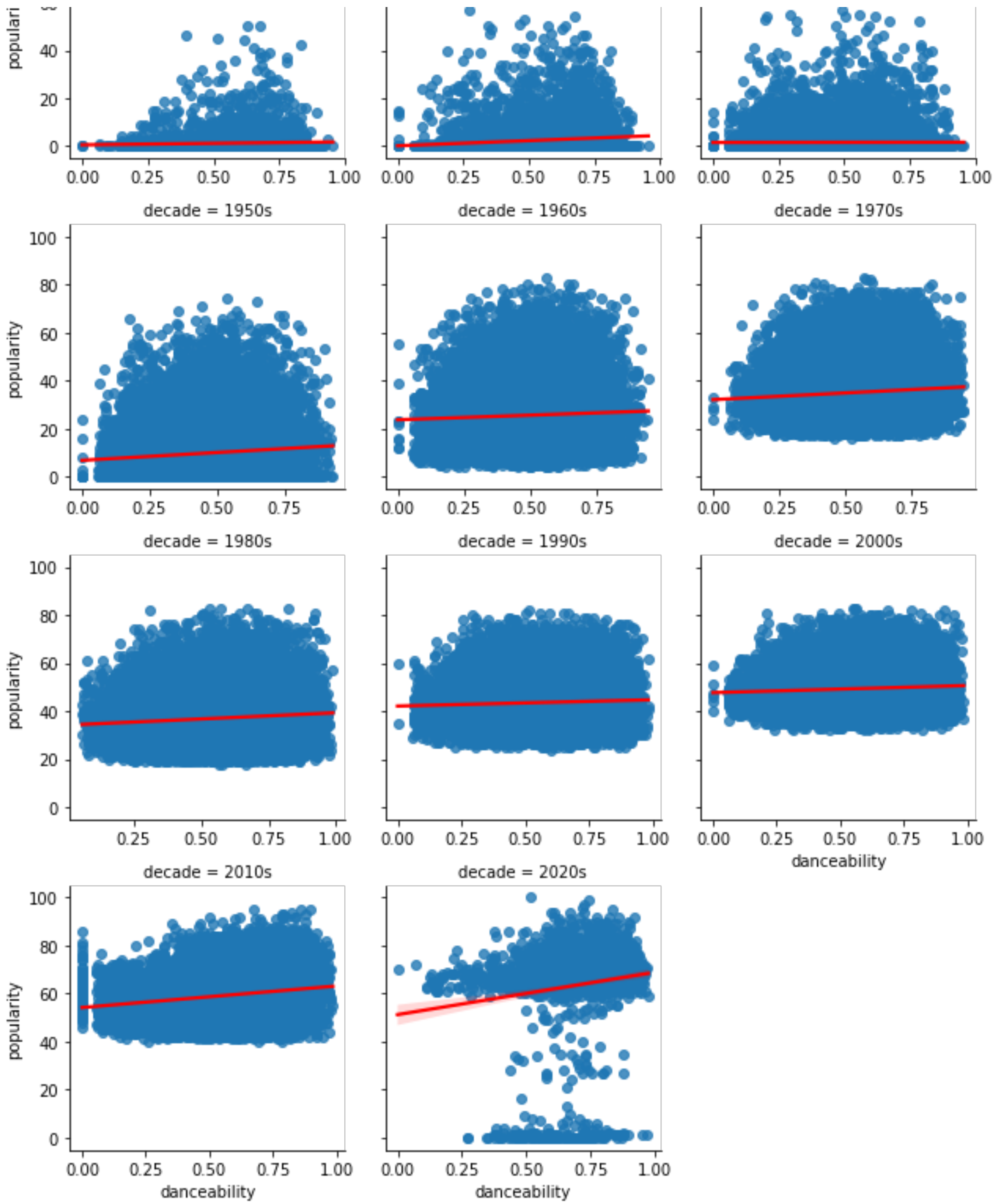





```
g = sns.FacetGrid(spotify_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, 'danceability', "popularity", line_kws = {'color':'red'})
```

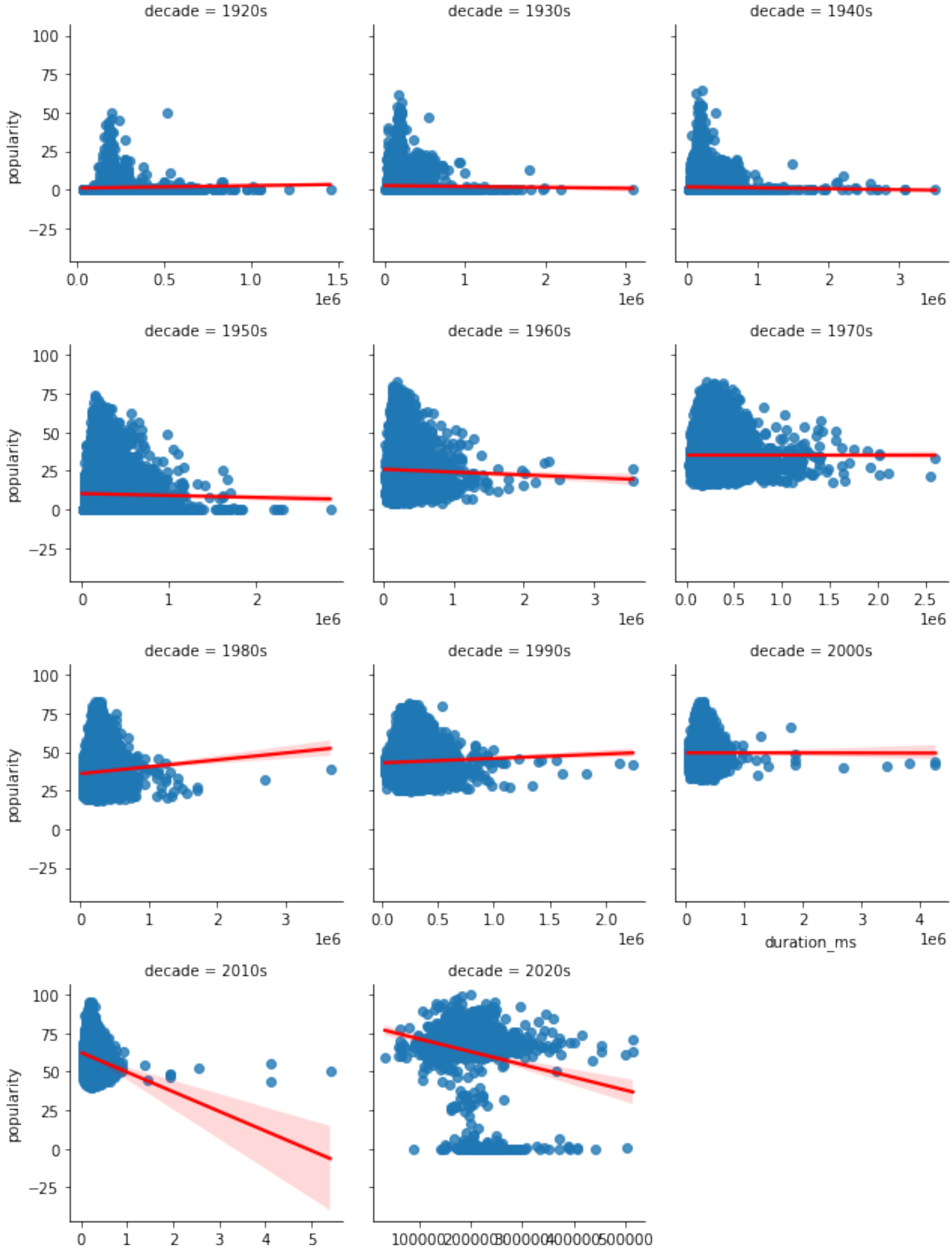
```
<seaborn.axisgrid.FacetGrid at 0x7f72474be850>
```






```
g = sns.FacetGrid(spotify_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, 'duration_ms', "popularity", line_kws = {'color':'red'})
```

```
<seaborn.axisgrid.FacetGrid at 0x7f724445f820>
```



There are some key leverage points here that are influencing the line plots, including songs that are extraordinarily long

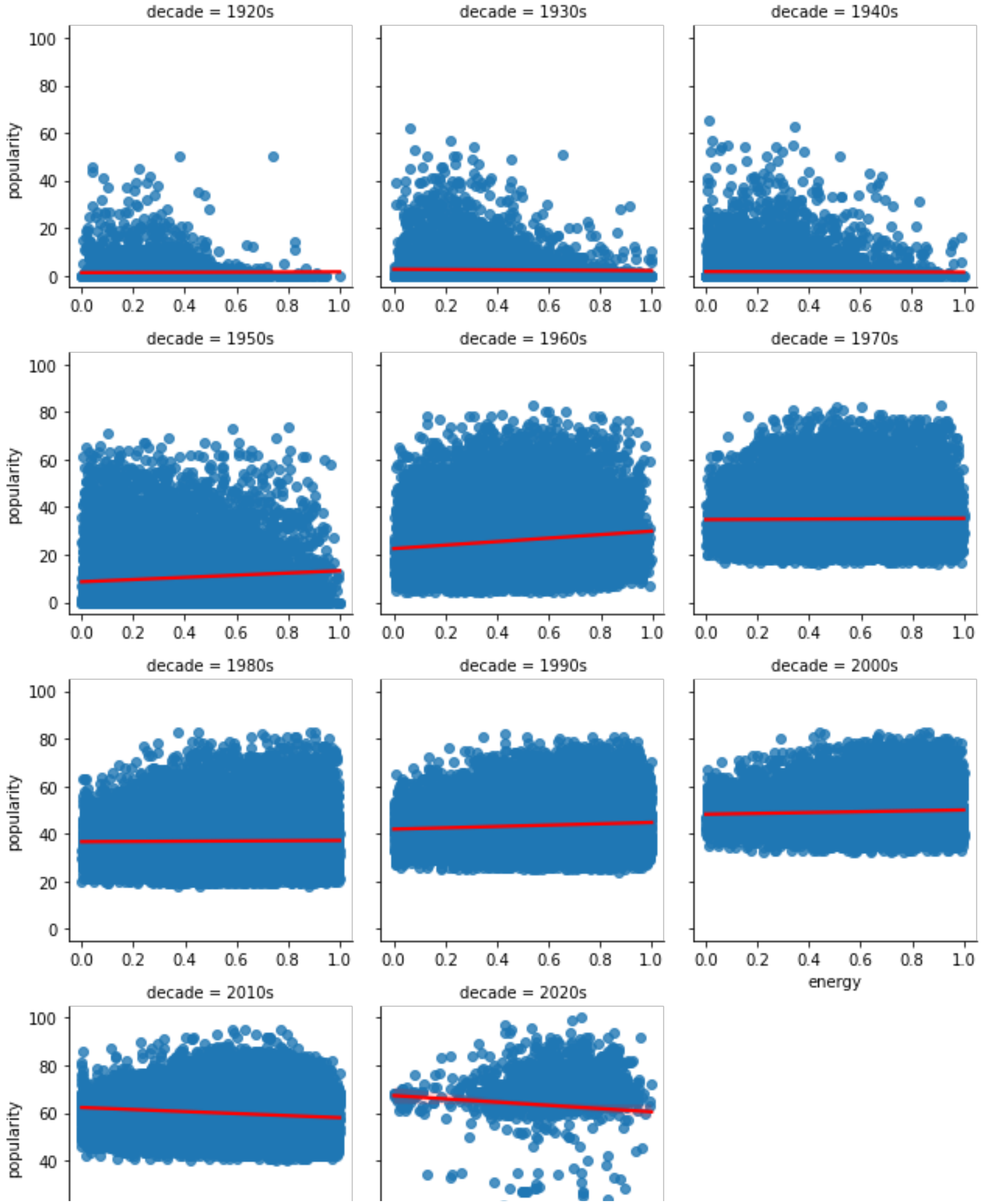
The songs listed below are 3000000 milliseconds, or rather 50 Minutes. Many of these are sounds used for helping people fall asleep, as well as a few real songs, and interestingly, someone's documentation of the Yalta Conference.

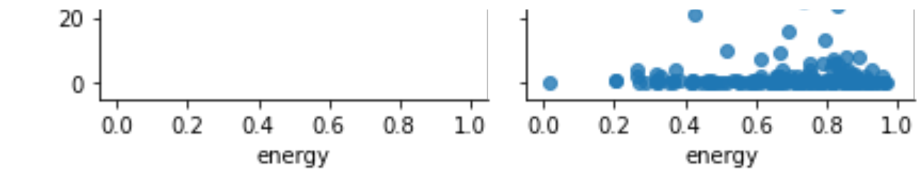
```
spotify_data[spotify_data['duration_ms'] > 3000000]
```

	acousticness	artists	danceability	duration_ms	energy	explici
7411	0.111000	['Sound Dreamer']	0.0000	5403500	0.000099	
9104	0.983000	['Umm Kulthum']	0.3960	3089255	0.347000	
10573	0.900000	['Umm Kulthum']	0.4100	3551152	0.451000	
38491	0.003670	['Sleep']	0.1600	3816373	0.572000	
46427	0.776000	['Chuck Riley']	0.5320	3432107	0.307000	
46972	0.975000	['Sounds for Life']	0.1530	4270034	0.079200	
54511	0.000385	['Lightning, Thunder and Rain Storm']	0.1160	4269407	0.338000	
54675	0.932000	['Ocean Sounds']	0.0797	4120258	0.995000	
72889	0.601000	['Environments']	0.1590	3557955	0.562000	
117370	0.993000	['Portugallien']	0.0644	3093226	0.132000	
118566	0.937000	['Franklin Delano Roosevelt']	0.6560	3091373	0.356000	
118777	0.891000	['Umm Kulthum']	0.3450	3523619	0.723000	
125152	0.932000	['Ocean Waves For Sleep']	0.0797	4120258	0.995000	
140784	0.976000	['Brian Eno']	0.0918	3650800	0.056900	

```
g = sns.FacetGrid(spotify_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, 'energy', "popularity", line_kws = {'color':'red'})
```

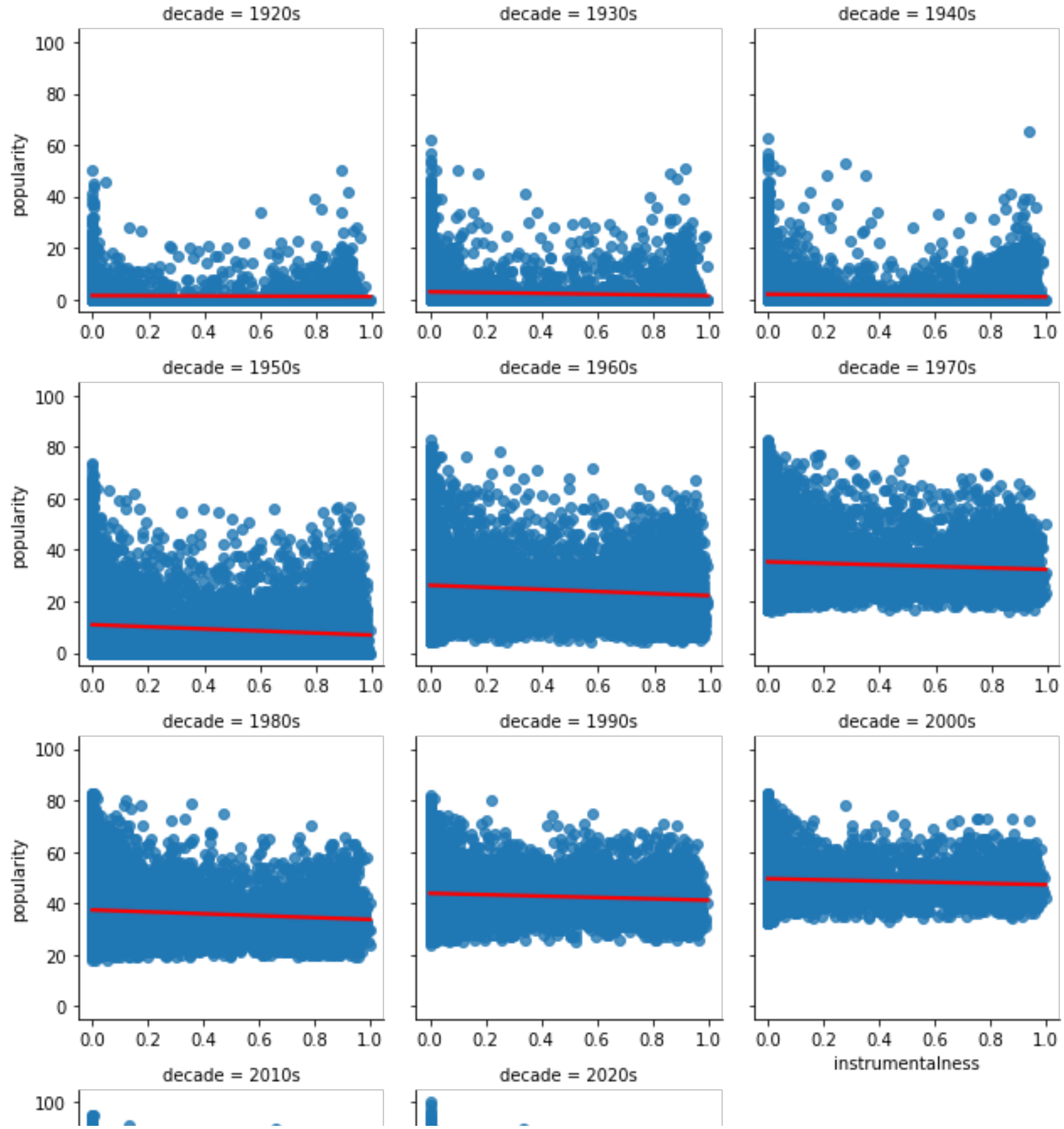
<seaborn.axisgrid.FacetGrid at 0x7f0e661f5f40>

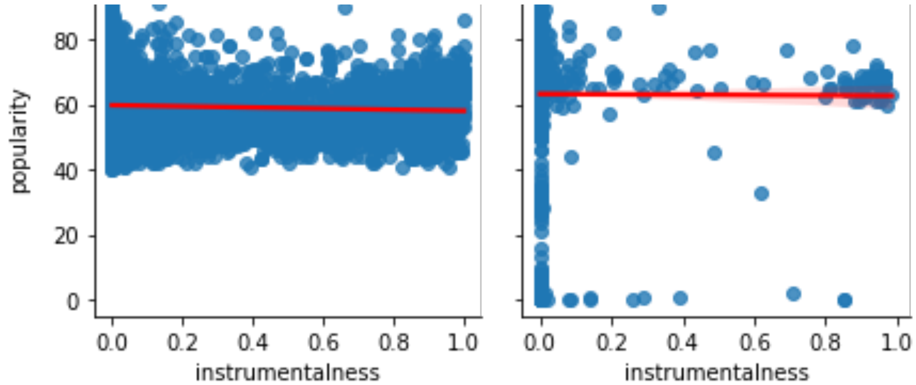





```
g = sns.FacetGrid(spotify_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, 'instrumentalness', "popularity", line_kws = {'color':'red'})
```

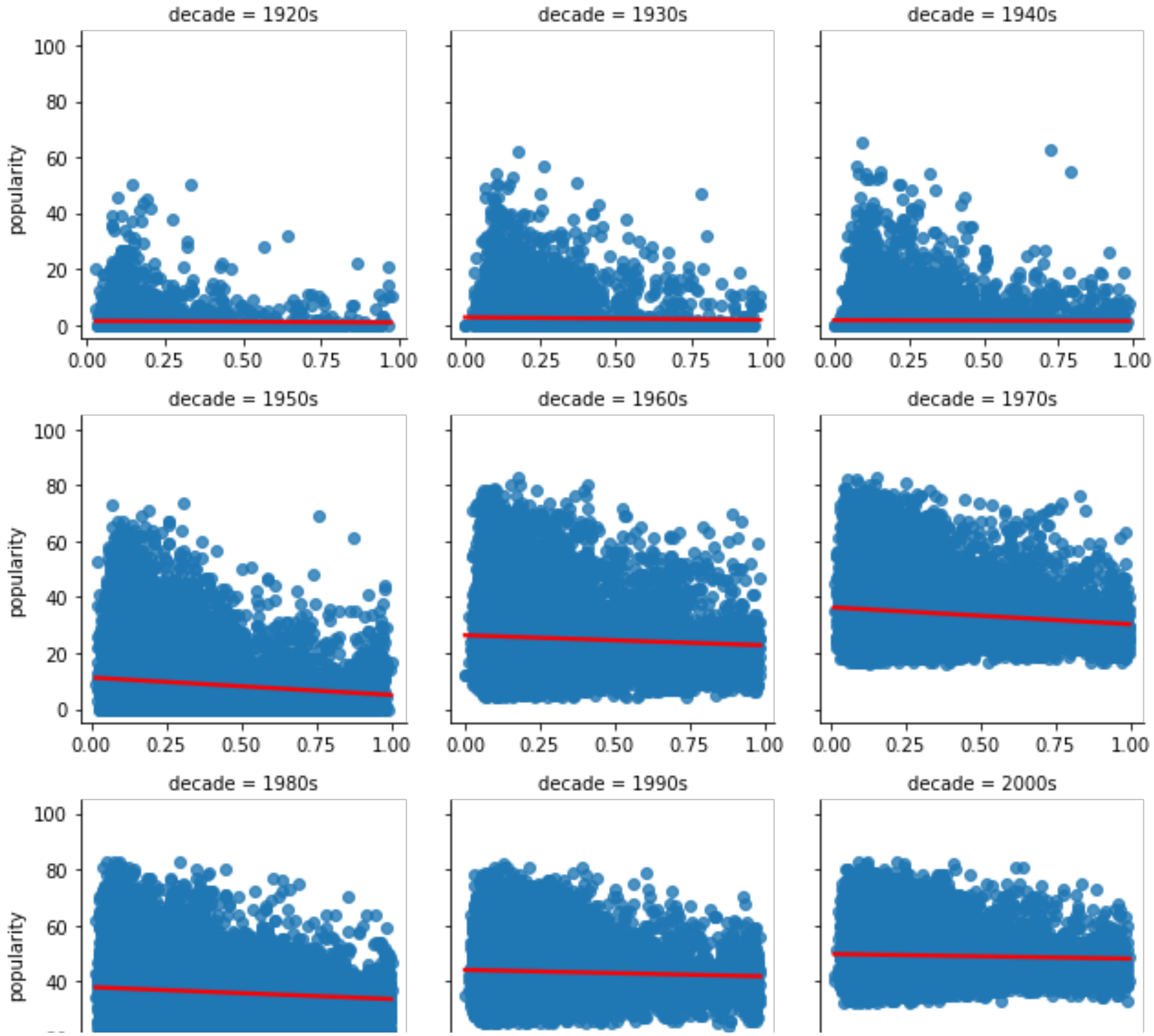
<seaborn.axisgrid.FacetGrid at 0x7f7248c93ac0>

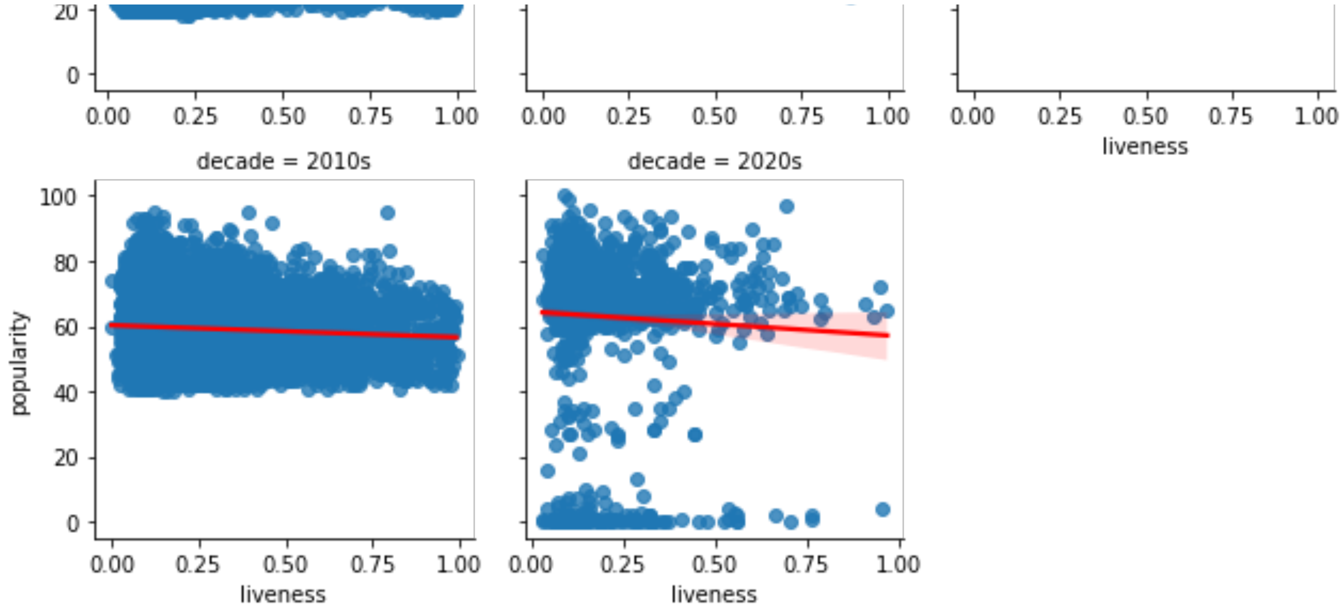





```
g = sns.FacetGrid(spotify_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, 'liveness', "popularity", line_kws = {'color':'red'})
```

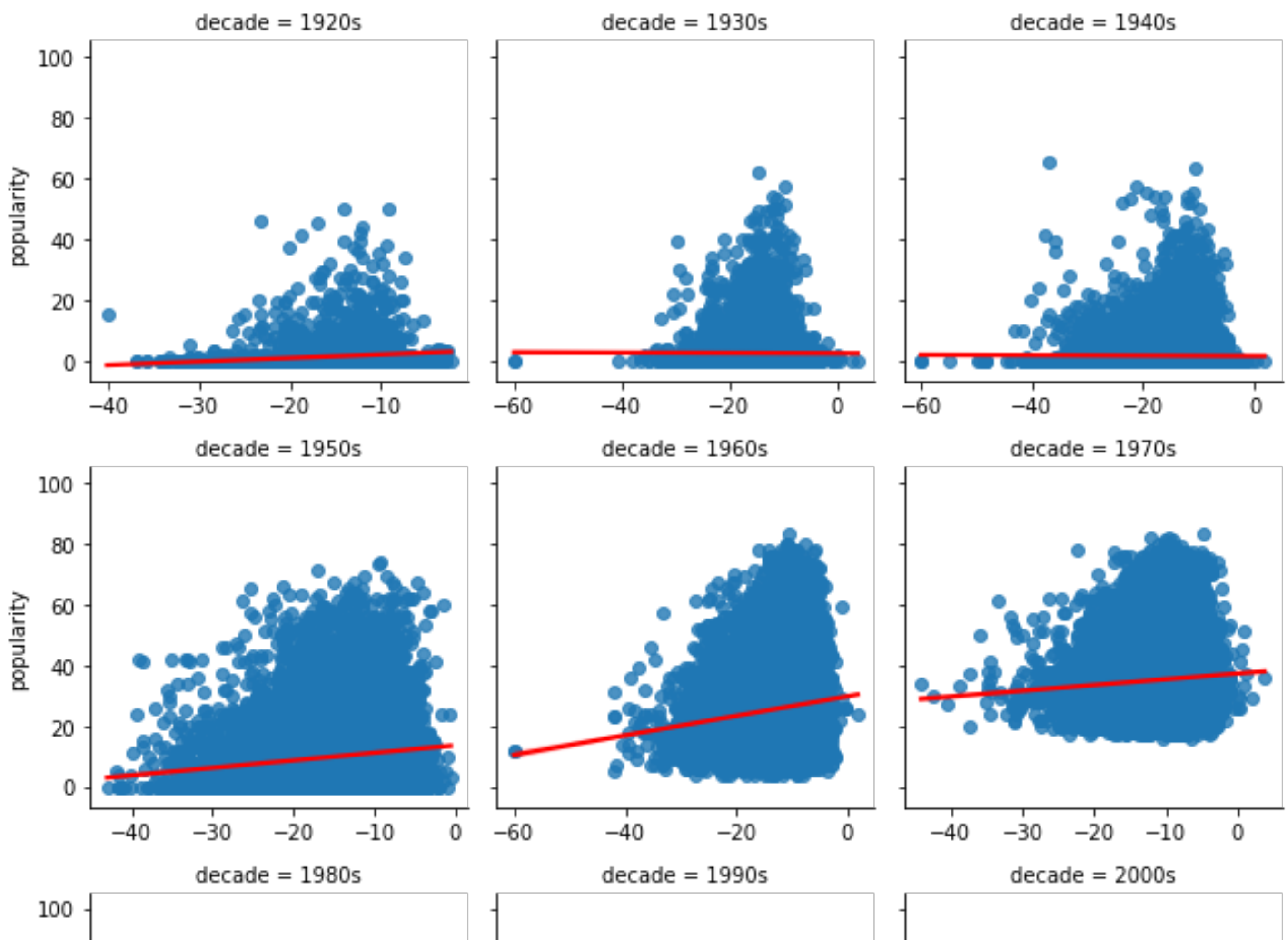
```
<seaborn.axisgrid.FacetGrid at 0x7f7247b64130>
```

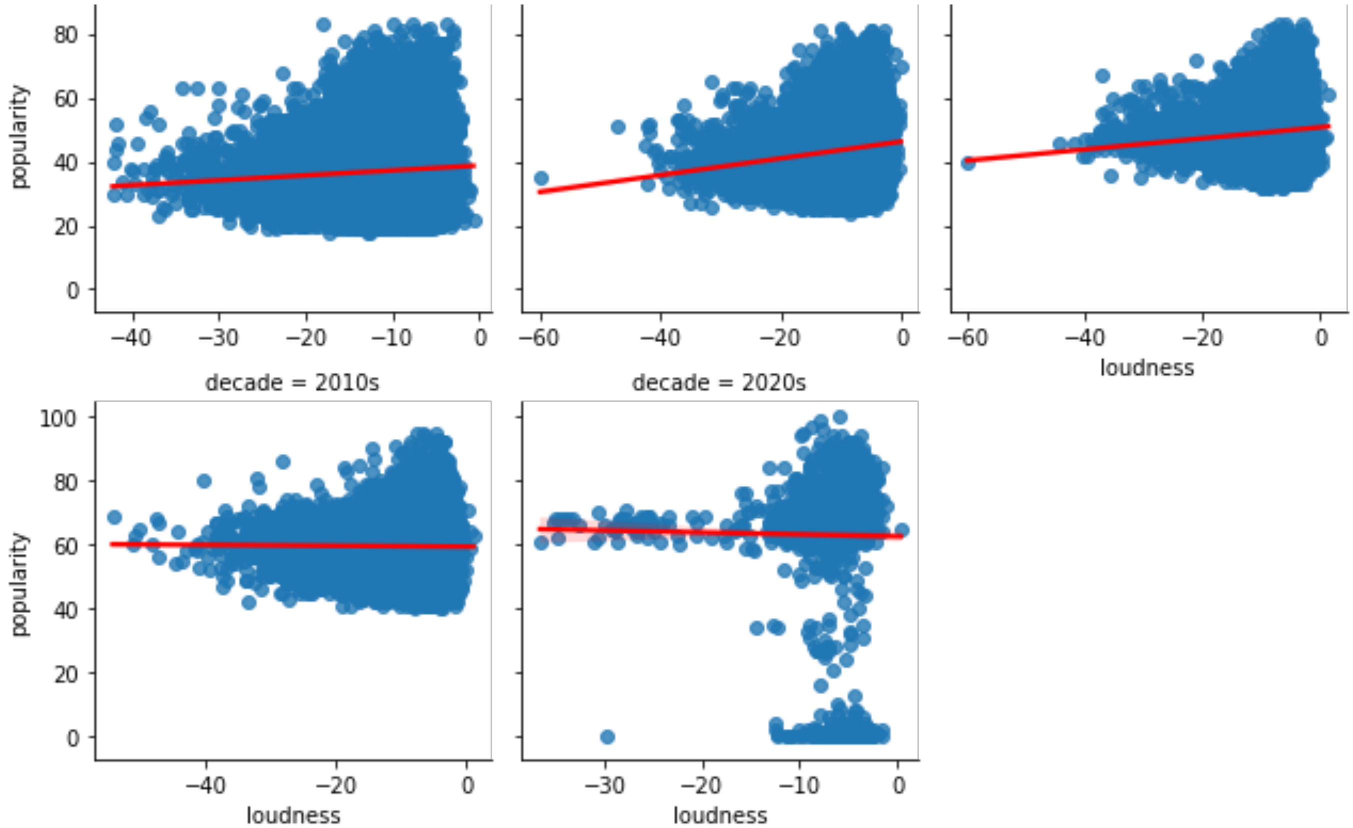





```
g = sns.FacetGrid(spotify_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, 'loudness', "popularity", line_kws = {'color':'red'})
```

```
<seaborn.axisgrid.FacetGrid at 0x7f0e65c4cd60>
```

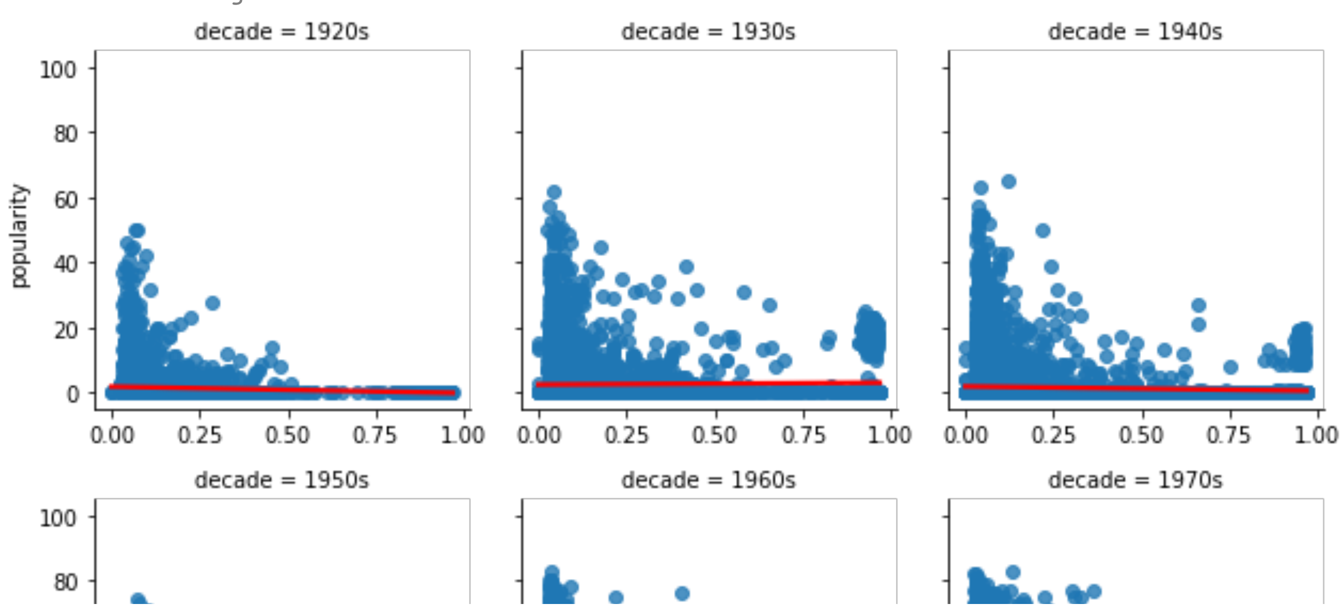


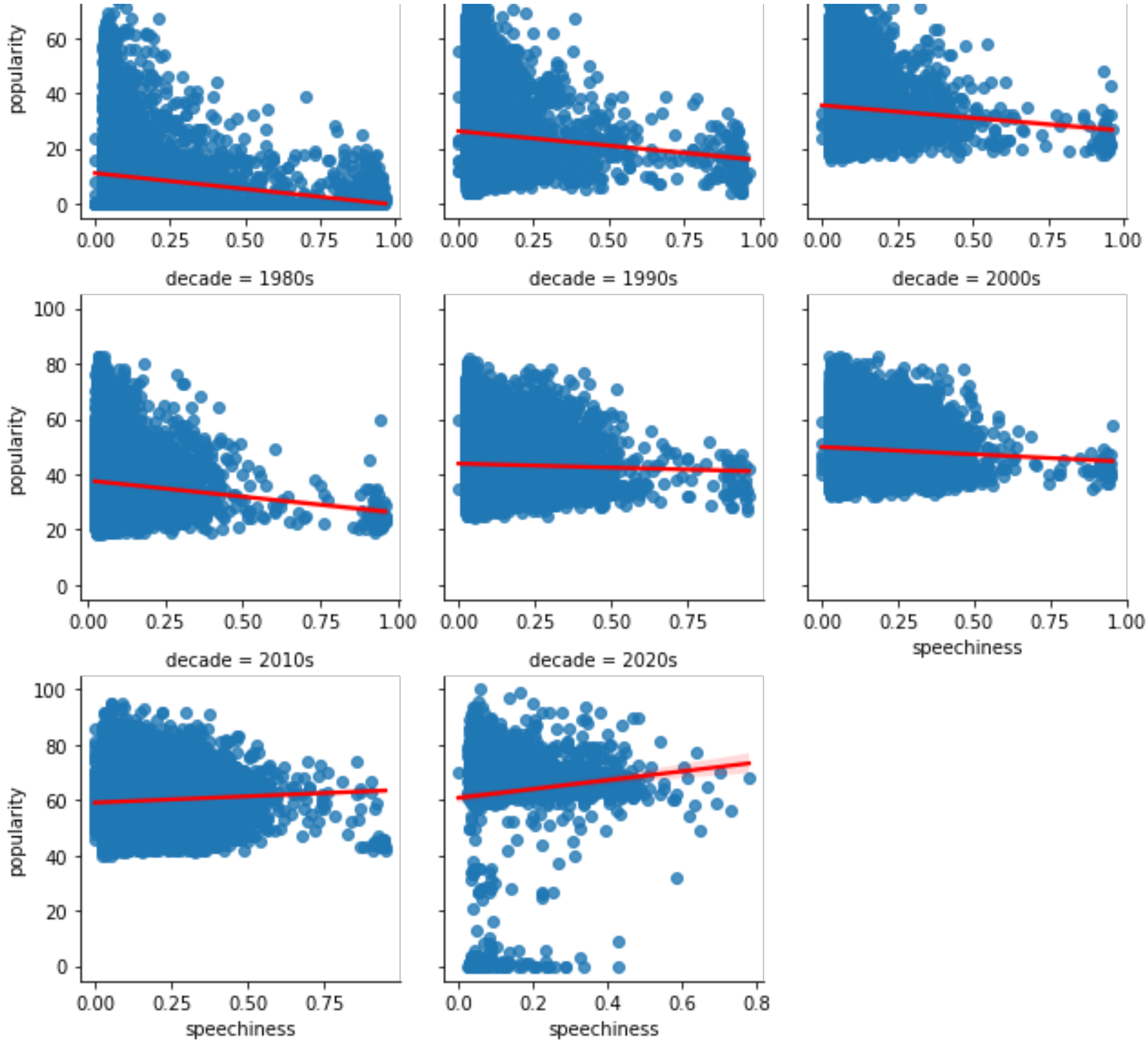


Positive relationship between loudness and popularity in many decades is observed (even when ignoring the leverage points)

```
g = sns.FacetGrid(spotify_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, 'speechiness', "popularity", line_kws = {'color':'red'})
```

```
<seaborn.axisgrid.FacetGrid at 0x7f7244b9c970>
```

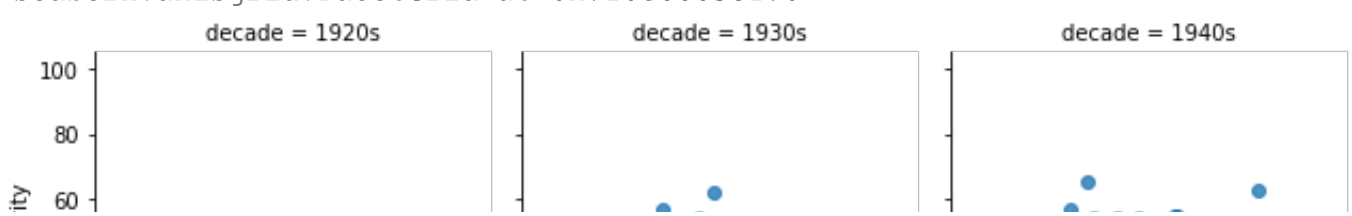


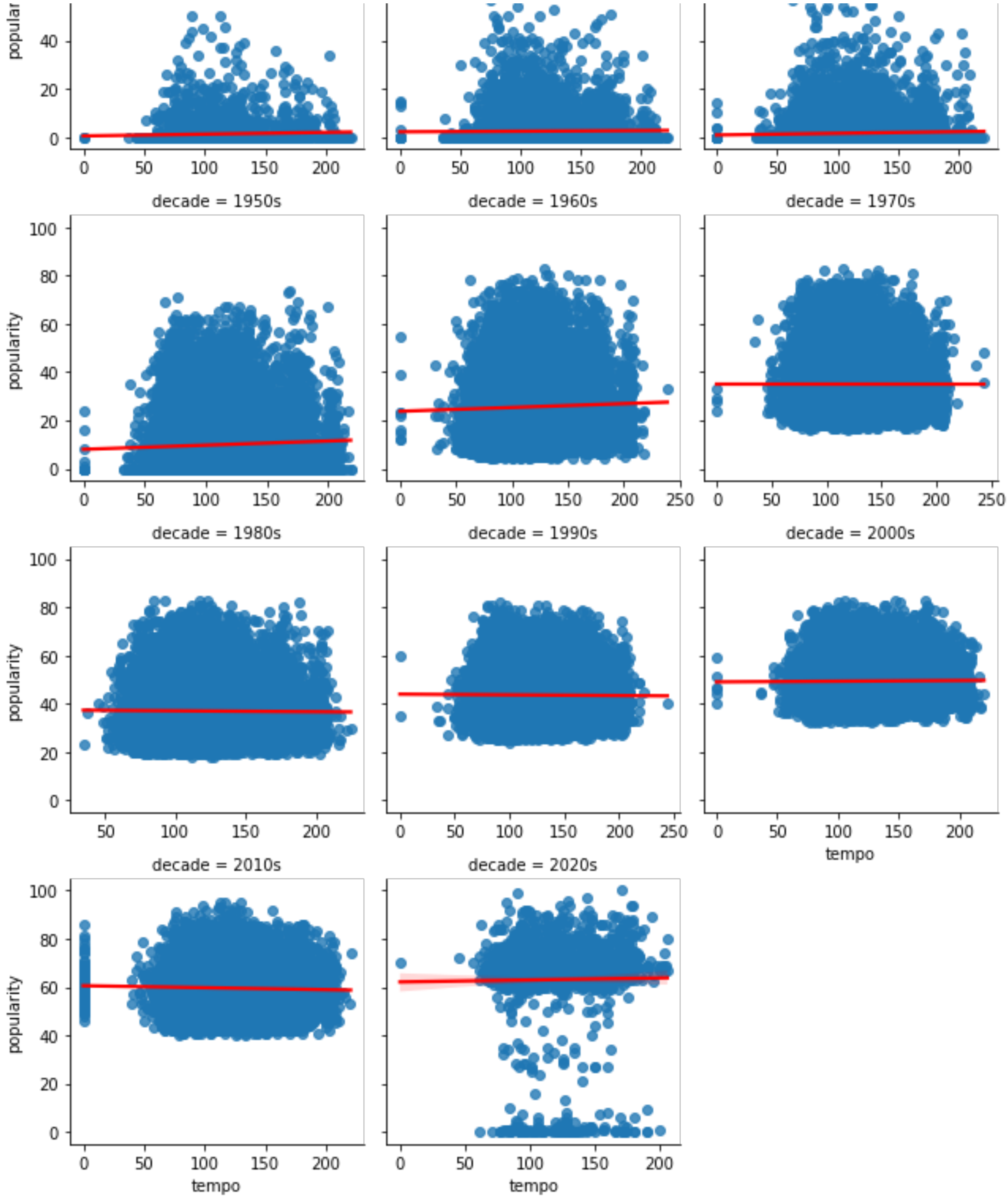


Negative relationship between speechiness and popularity observed in some of the mid to older decades (50s-80s)

```
g = sns.FacetGrid(spotify_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, 'tempo', "popularity", line_kws = {'color':'red'})
```

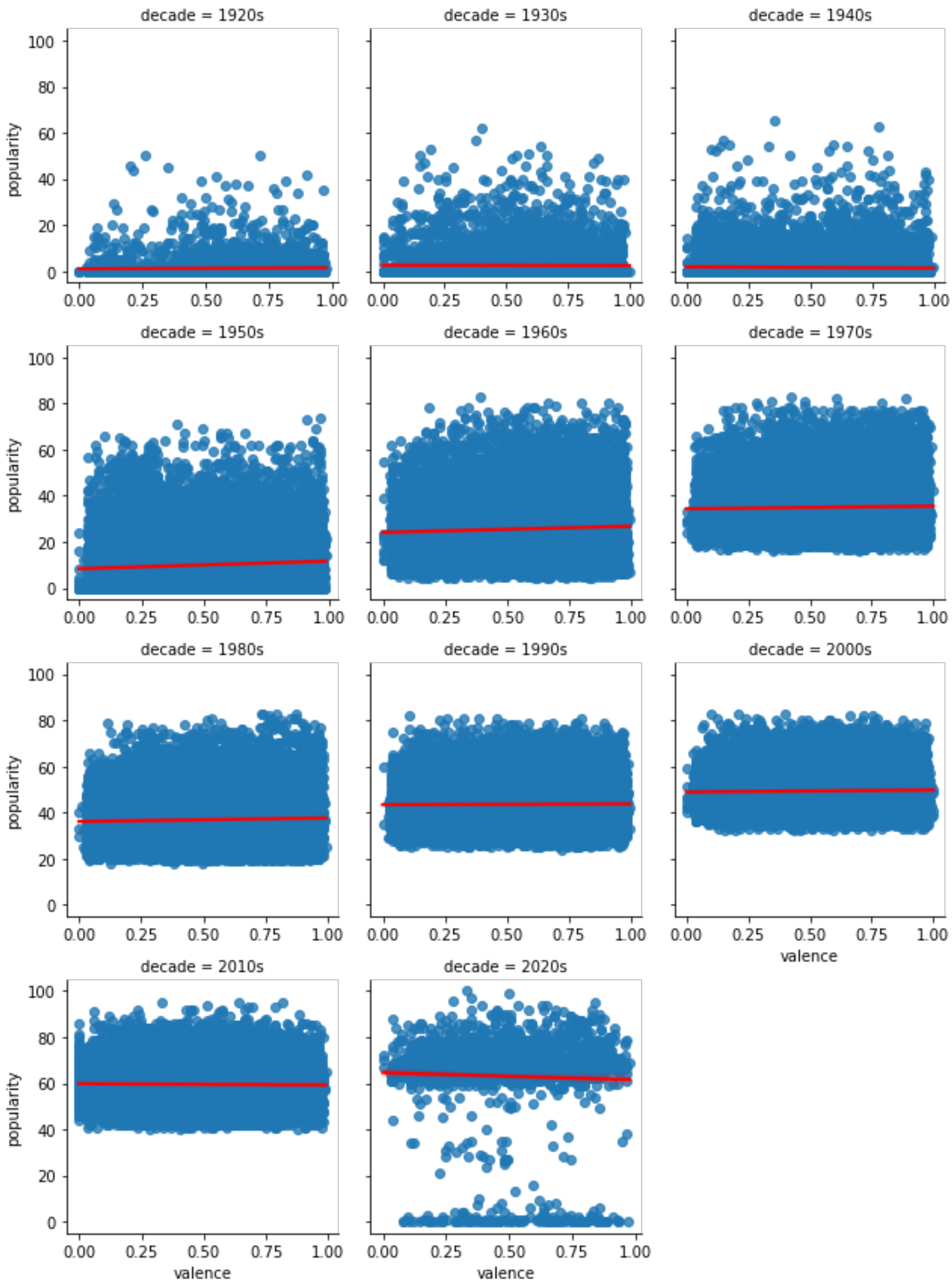
```
<seaborn.axisgrid.FacetGrid at 0x7f0e66658f70>
```






```
g = sns.FacetGrid(spotify_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, 'valence', "popularity", line_kws = {'color':'red'})
```

```
<seaborn.axisgrid.FacetGrid at 0x7f0e65f6e100>
```



Decade-wise line plots of popularity relationships

```
relplot_data = spotify_data.copy()
```

```
round_dict = {col:2 for col in metric_cols}
round_dict['tempo'] = 0
round_dict['loudness'] = 1
```

Creating "bins" for relplot data so it is not averaging the y over too precise of an x

```
relplot_data = relplot_data.round(round_dict)
```

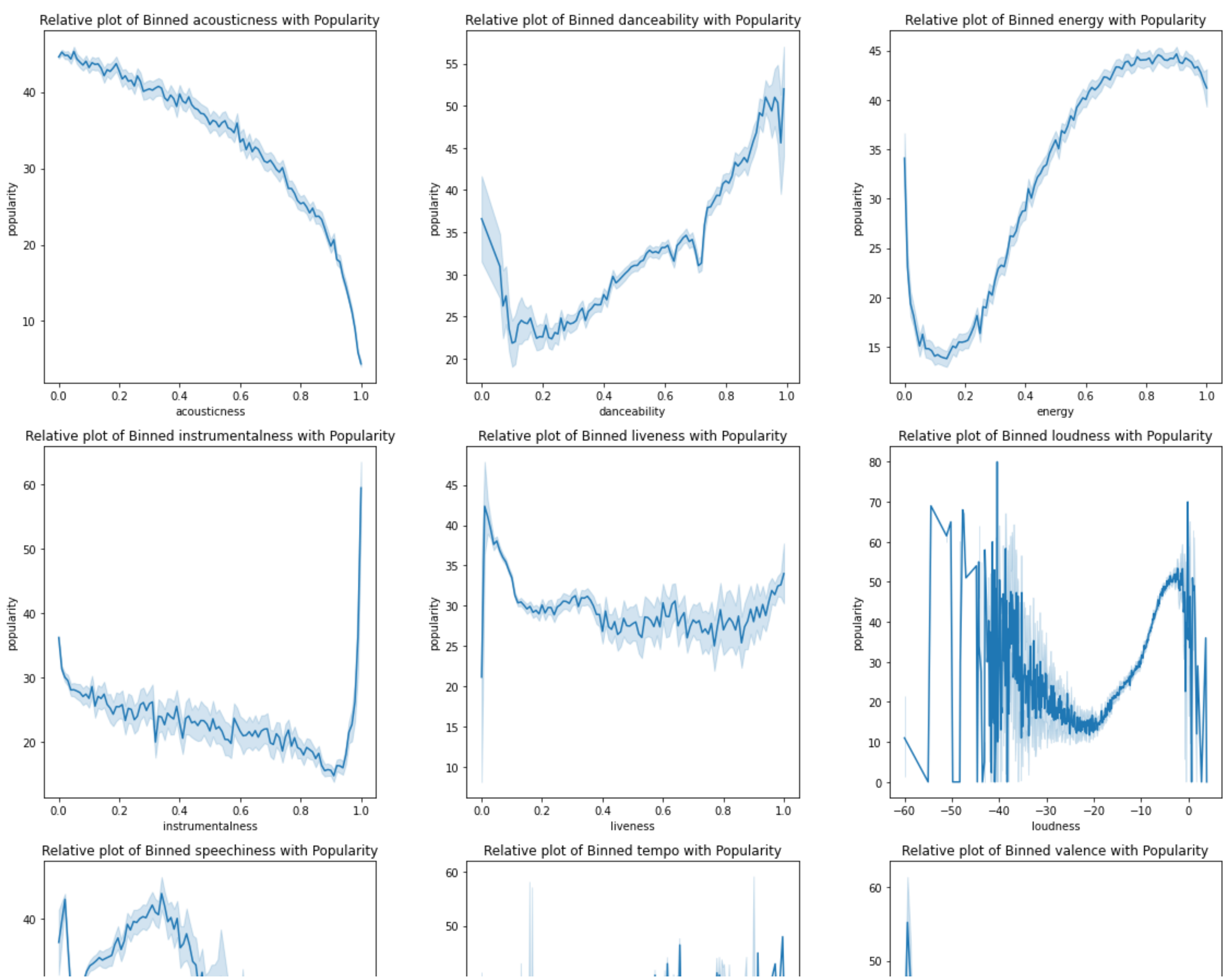
Data not great at being predictive, makes EDA not very interpretable

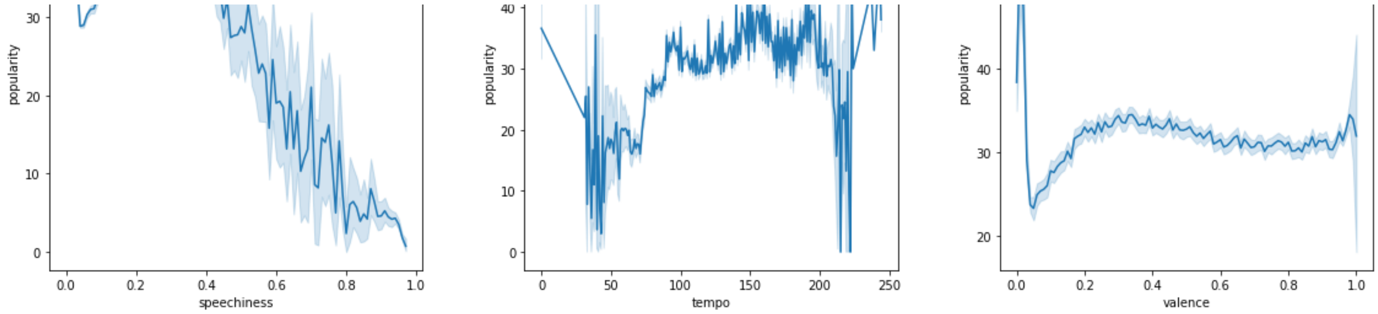
```
#g = sns.FacetGrid(spotify_data, col_wrap = 3, sharex = False, col = "decade")
#g.map(sns.relplot, 'acousticness', "popularity", kind="line")
```

```
metric_cols
```

```
['acousticness',
 'danceability',
 'energy',
 'instrumentalness',
 'liveness',
 'loudness',
 'speechiness',
 'tempo',
 'valence']
```

```
def relplots_subplot():
  figure = plt.figure(figsize=(16,16))
  #fig, axs = plt.subplots(3,3, figsize=(16, 16))

  plotted = 1
  for col in metric_cols:
    #sns.relplot(data=relplot_data, x = col, y = "popularity", kind="line", ax = a
    plt.subplot(3,3,plotted)
    sns.lineplot(data=relplot_data, x = col, y = "popularity")
    plt.title(f'Line plot of Binned {col} with Popularity')
    plotted += 1

  figure.tight_layout()
  plt.show()


relplots_subplot()
```

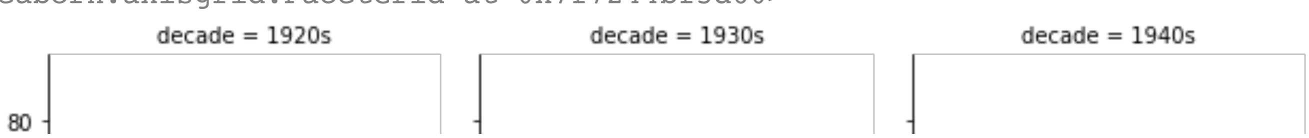


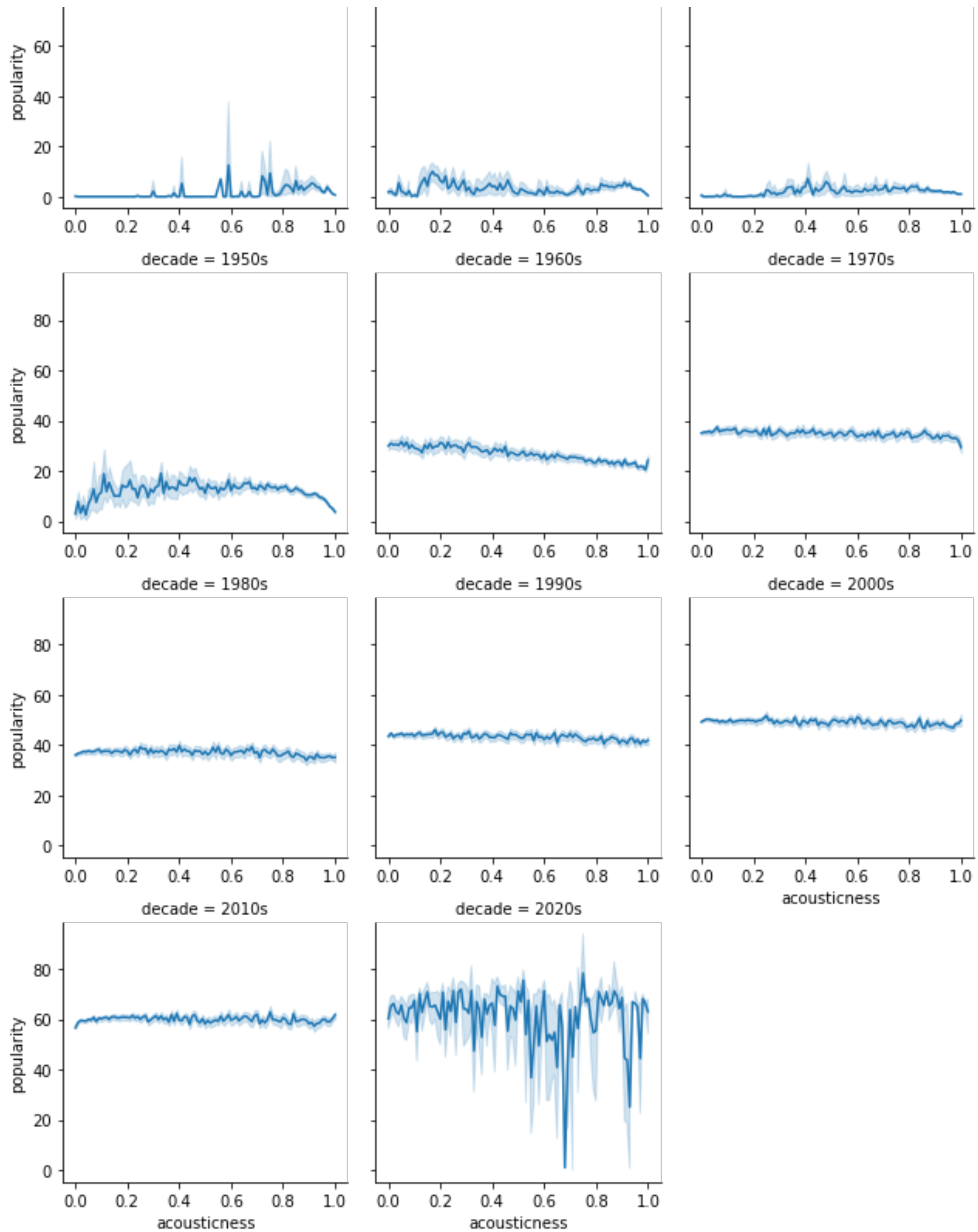


Decade-wise plots of binned column relationships with popularity

```
g = sns.FacetGrid(relplot_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.lineplot, 'acousticness', "popularity")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f7244bf5d60>
```

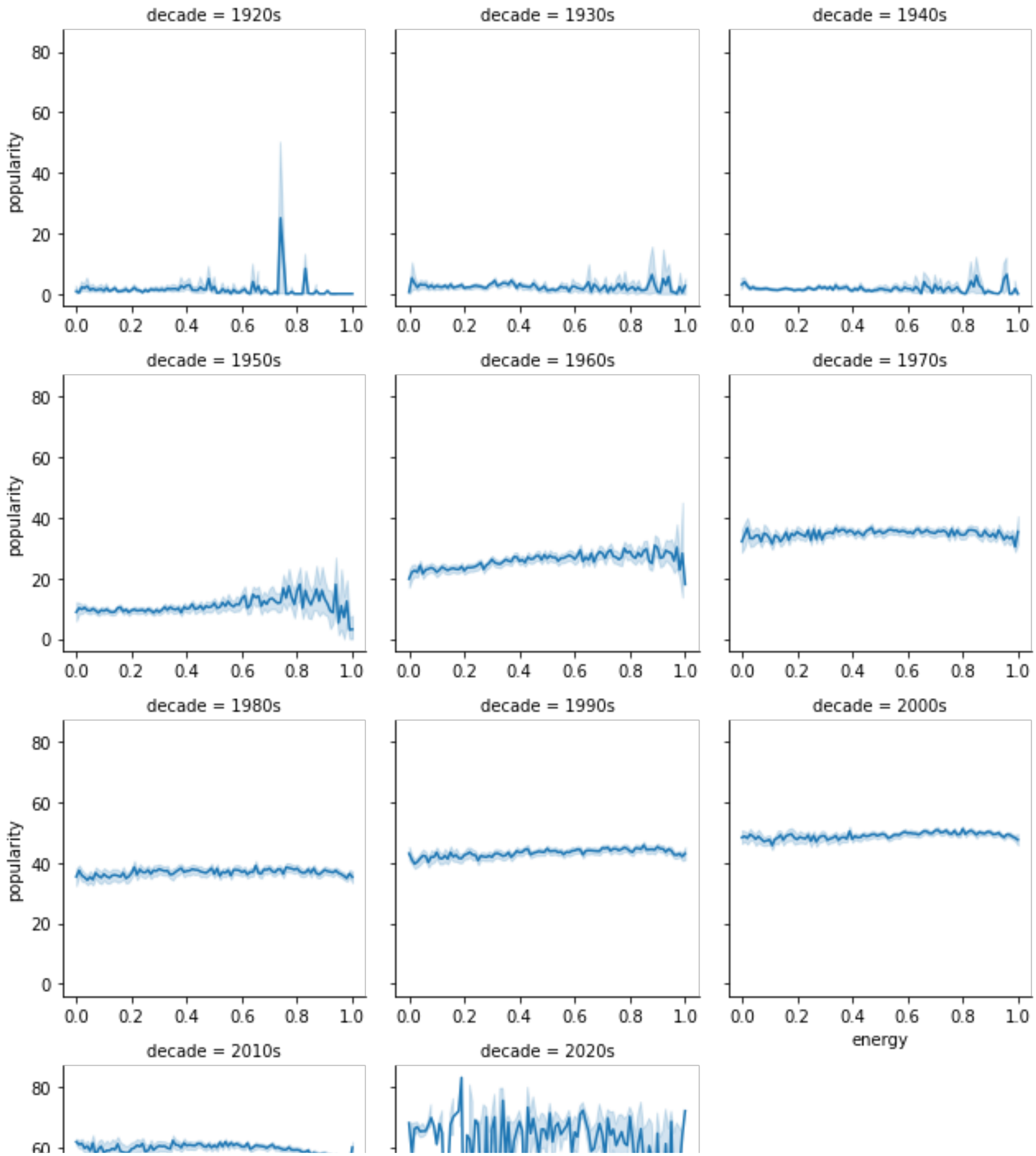


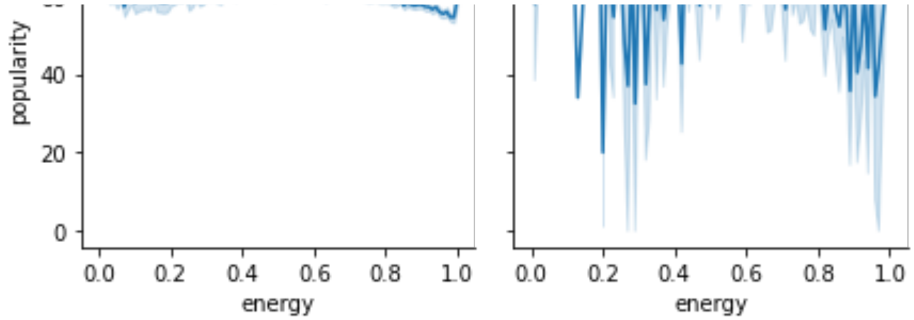

popularity
acousticness
decade = 1950s
decade = 1960s
decade = 1970s
decade = 1980s
decade = 1990s
decade = 2000s
decade = 2010s
decade = 2020s
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80

```
g = sns.FacetGrid(relplot_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.lineplot, 'danceability', "popularity")
```

```
g = sns.FacetGrid(relplot_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.lineplot, 'energy', "popularity")
```

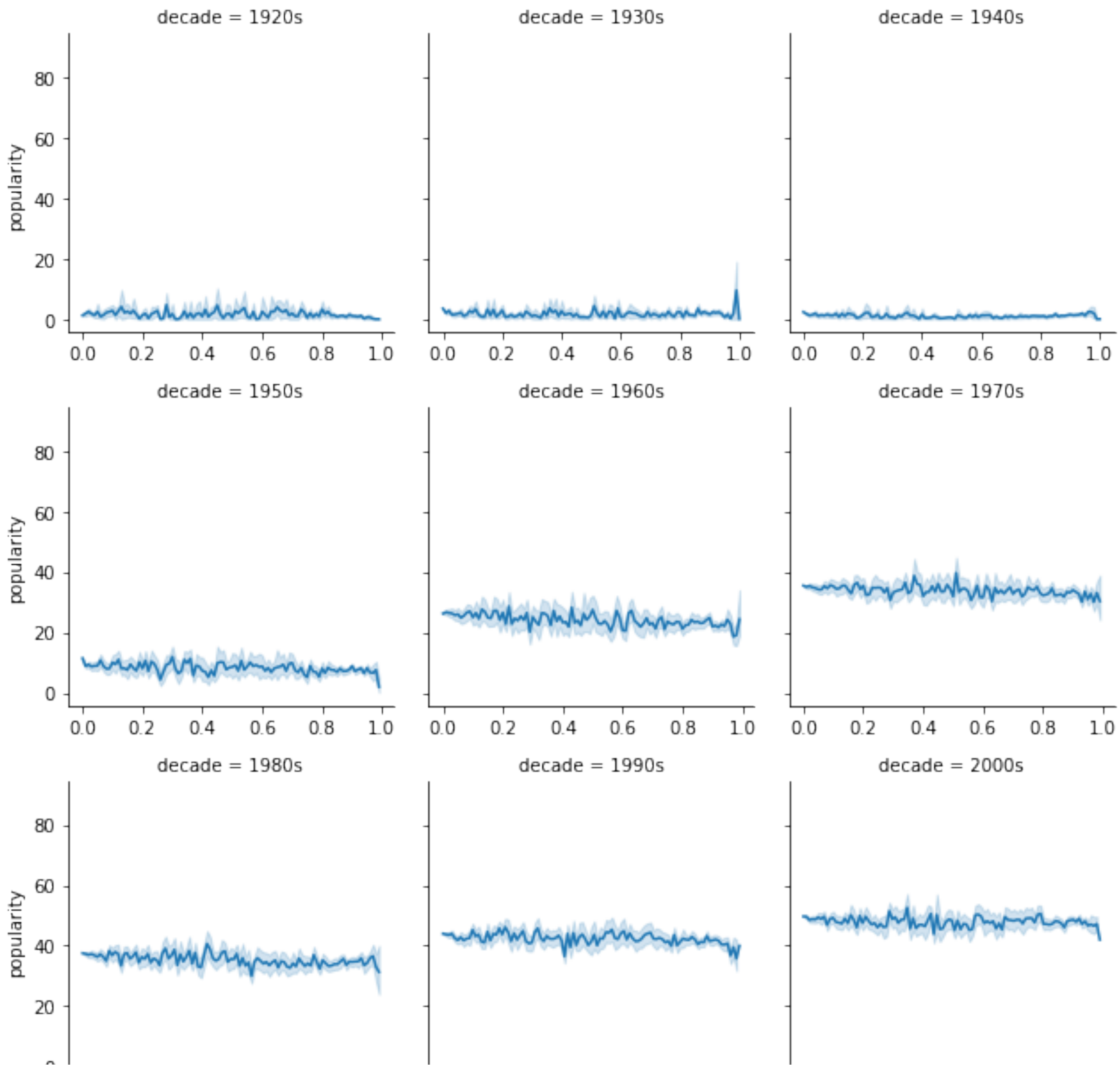
```
<seaborn.axisgrid.FacetGrid at 0x7f7243f05460>
```

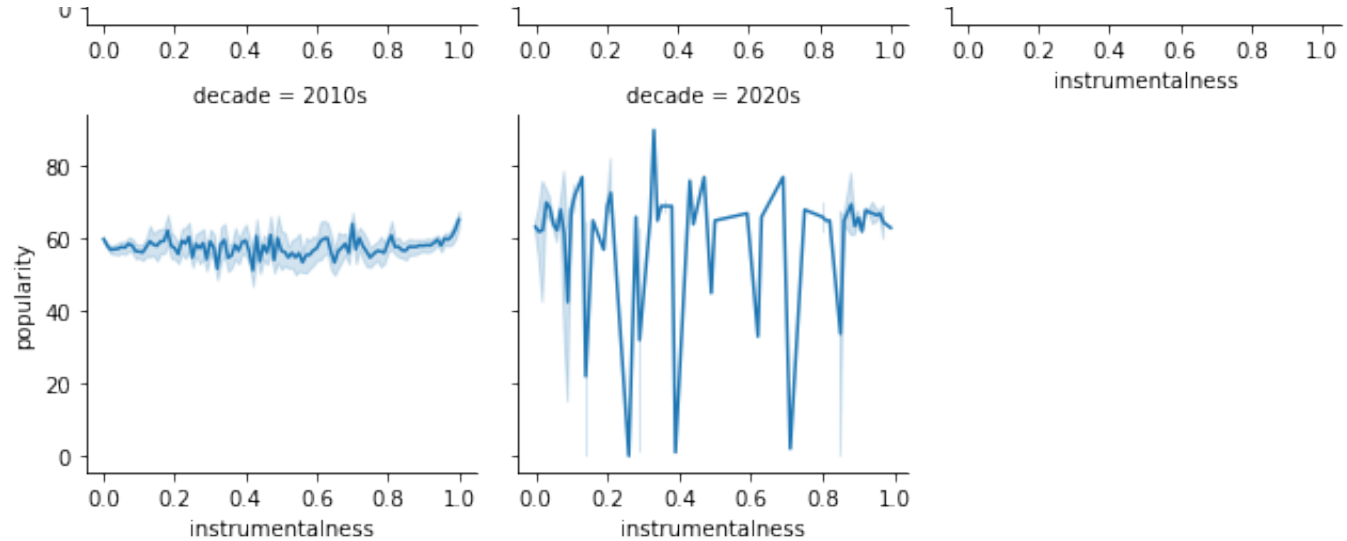




```
g = sns.FacetGrid(relplot_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.lineplot, 'instrumentalness', "popularity")
```

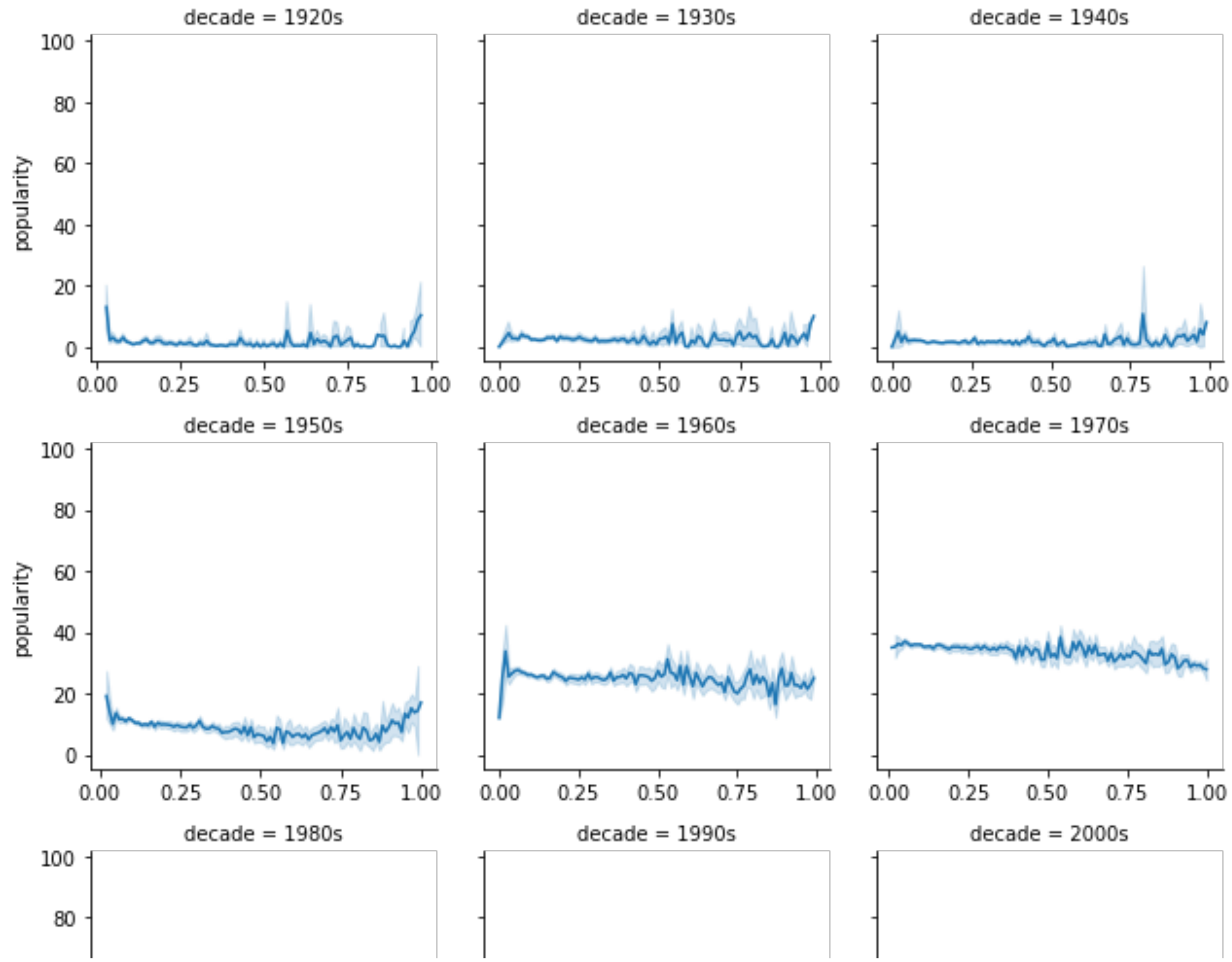
```
<seaborn.axisgrid.FacetGrid at 0x7f7243c6c910>
```

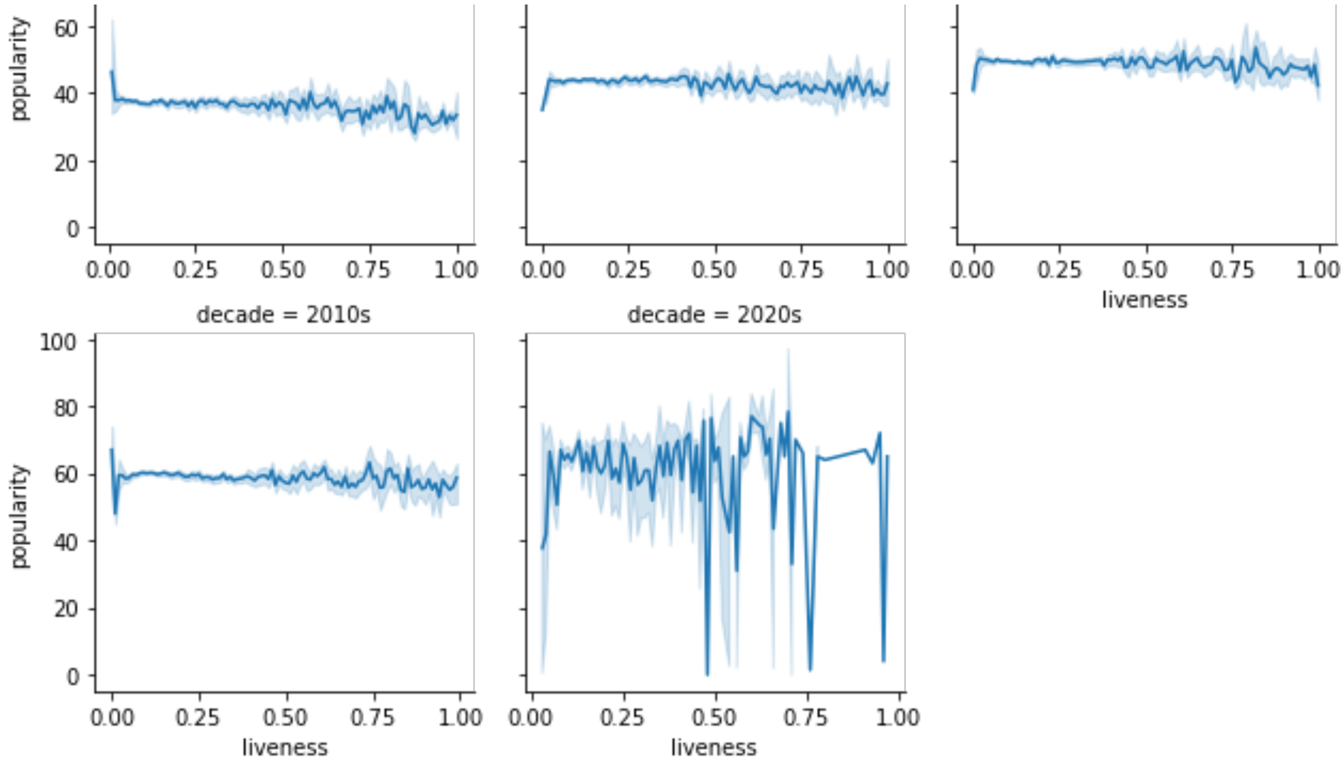





```
g = sns.FacetGrid(relplot_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.lineplot, 'liveness', "popularity")
```

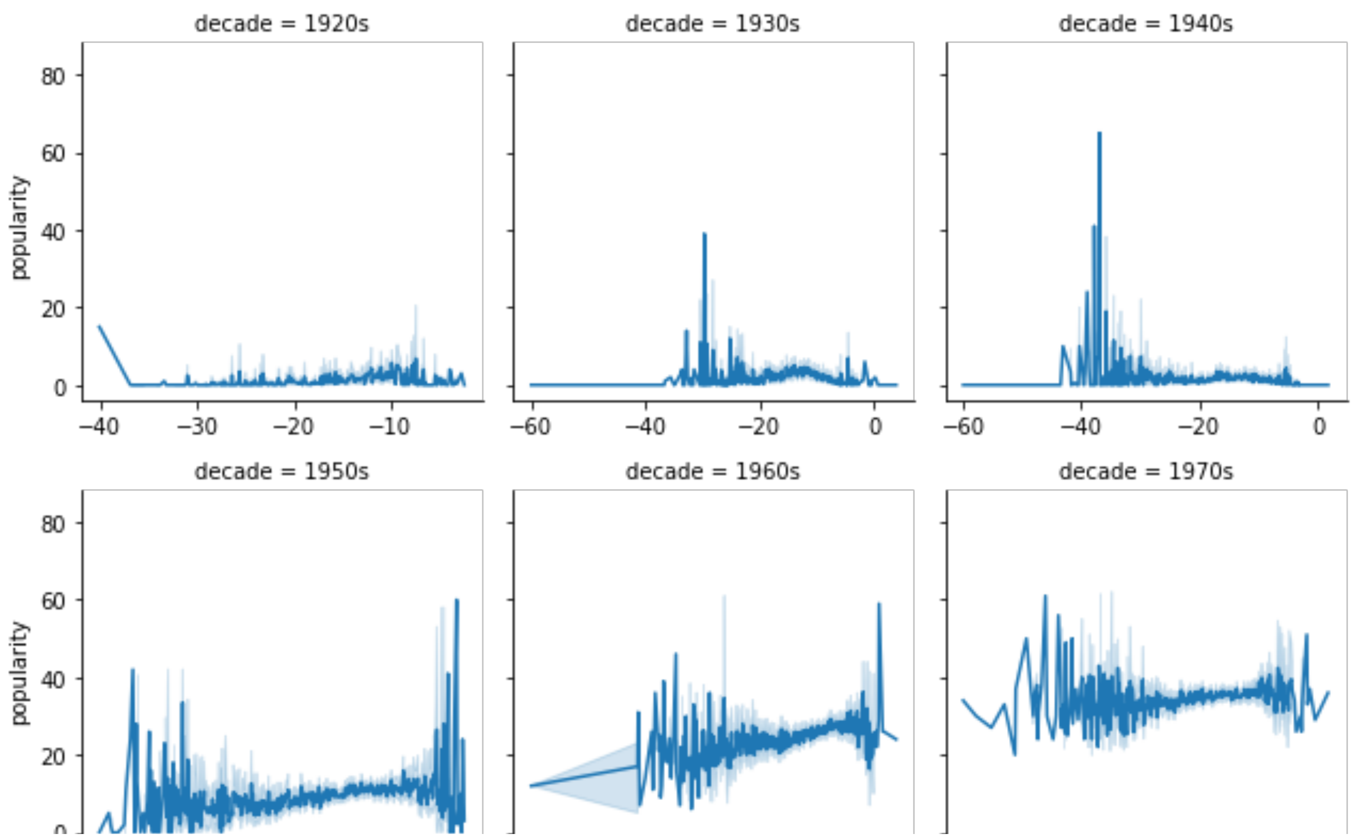
```
<seaborn.axisgrid.FacetGrid at 0x7f72438da730>
```

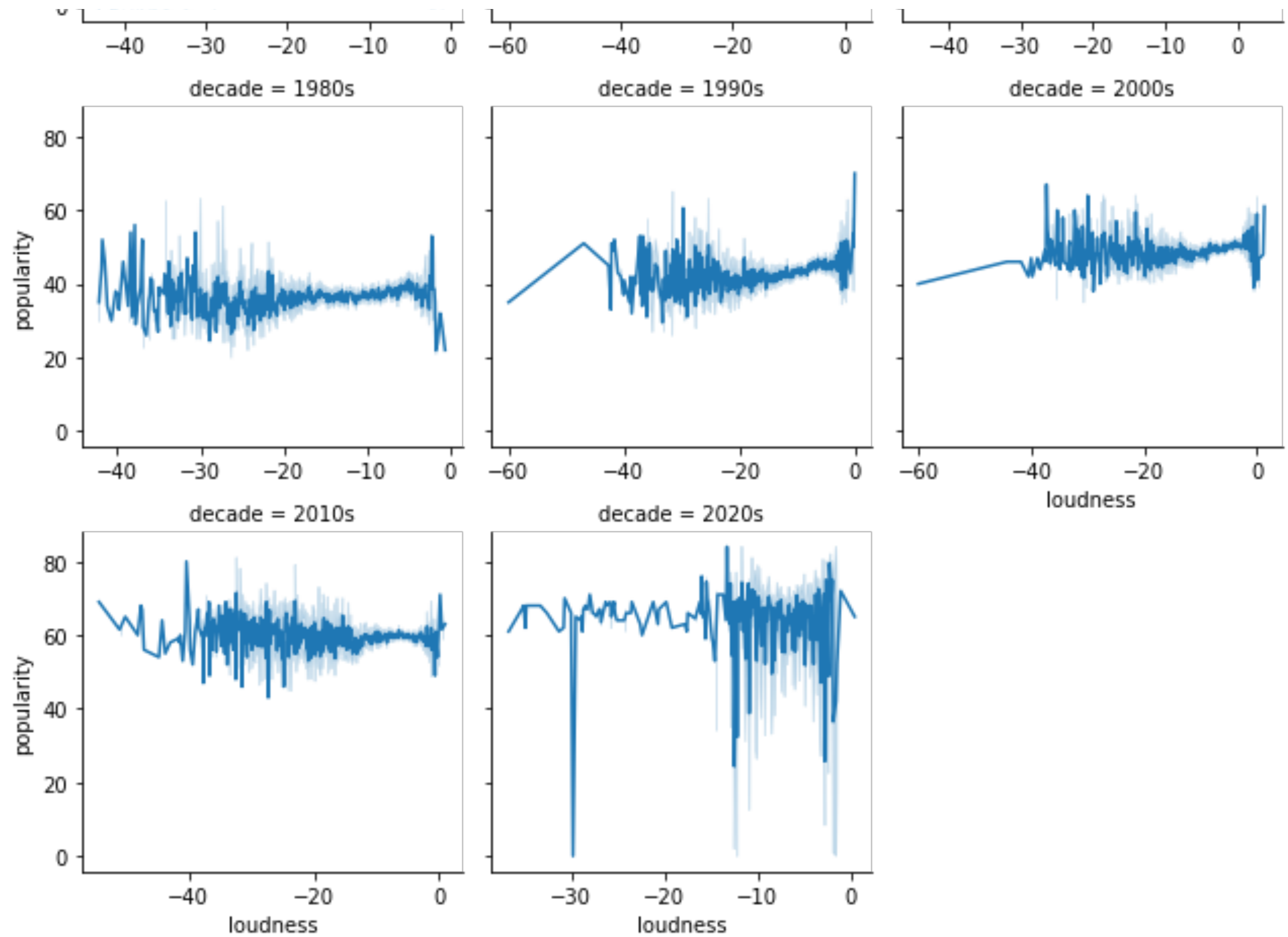





```
g = sns.FacetGrid(relplot_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.lineplot, 'loudness', "popularity")
```

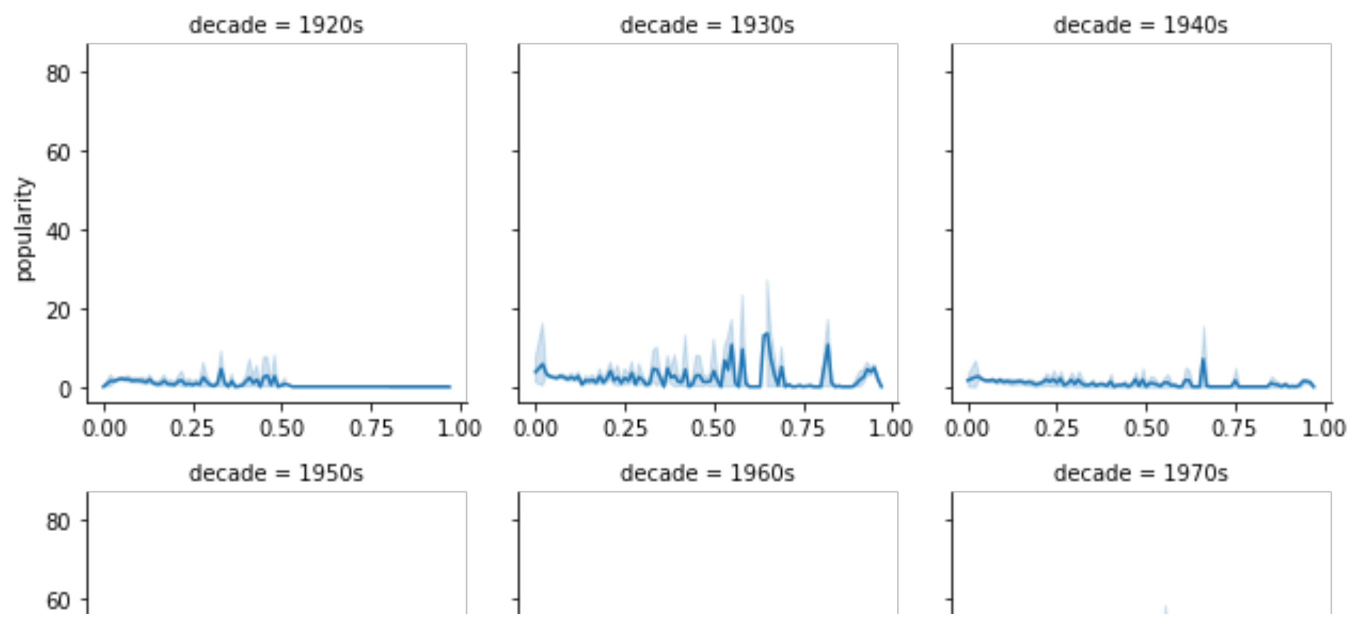
```
<seaborn.axisgrid.FacetGrid at 0x7f724343faf0>
```

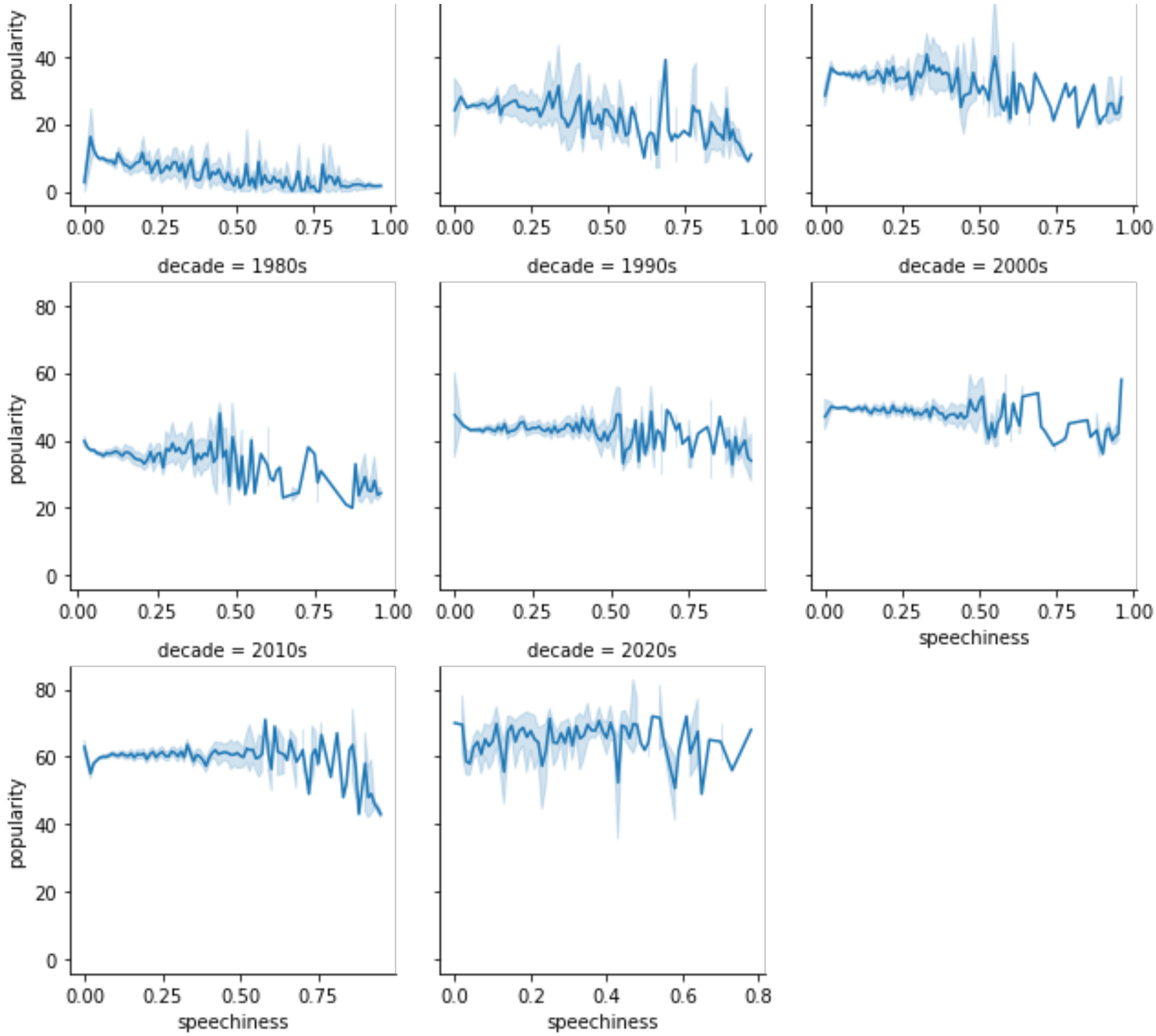





```
g = sns.FacetGrid(relplot_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.lineplot, 'speechiness', "popularity")
```

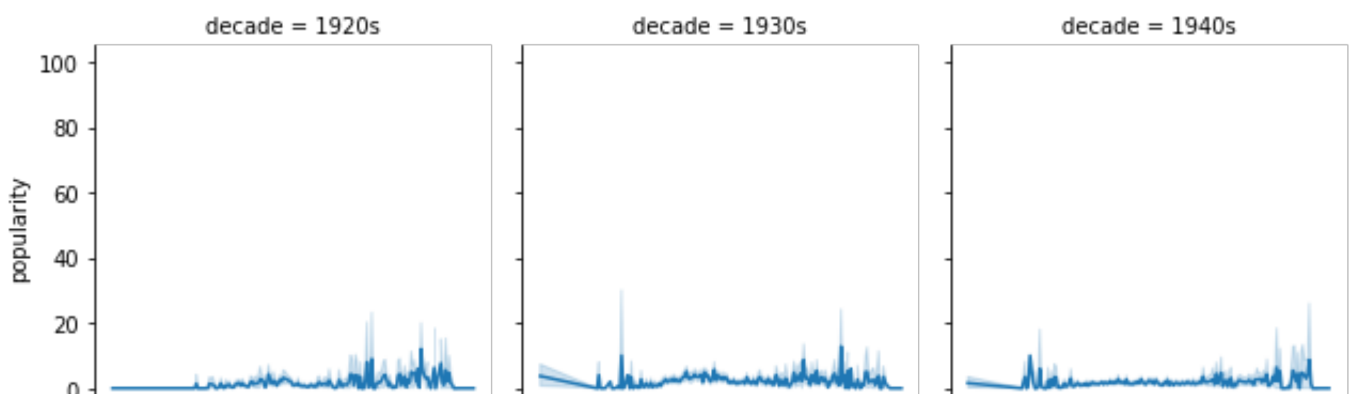
```
<seaborn.axisgrid.FacetGrid at 0x7f7243ddb520>
```

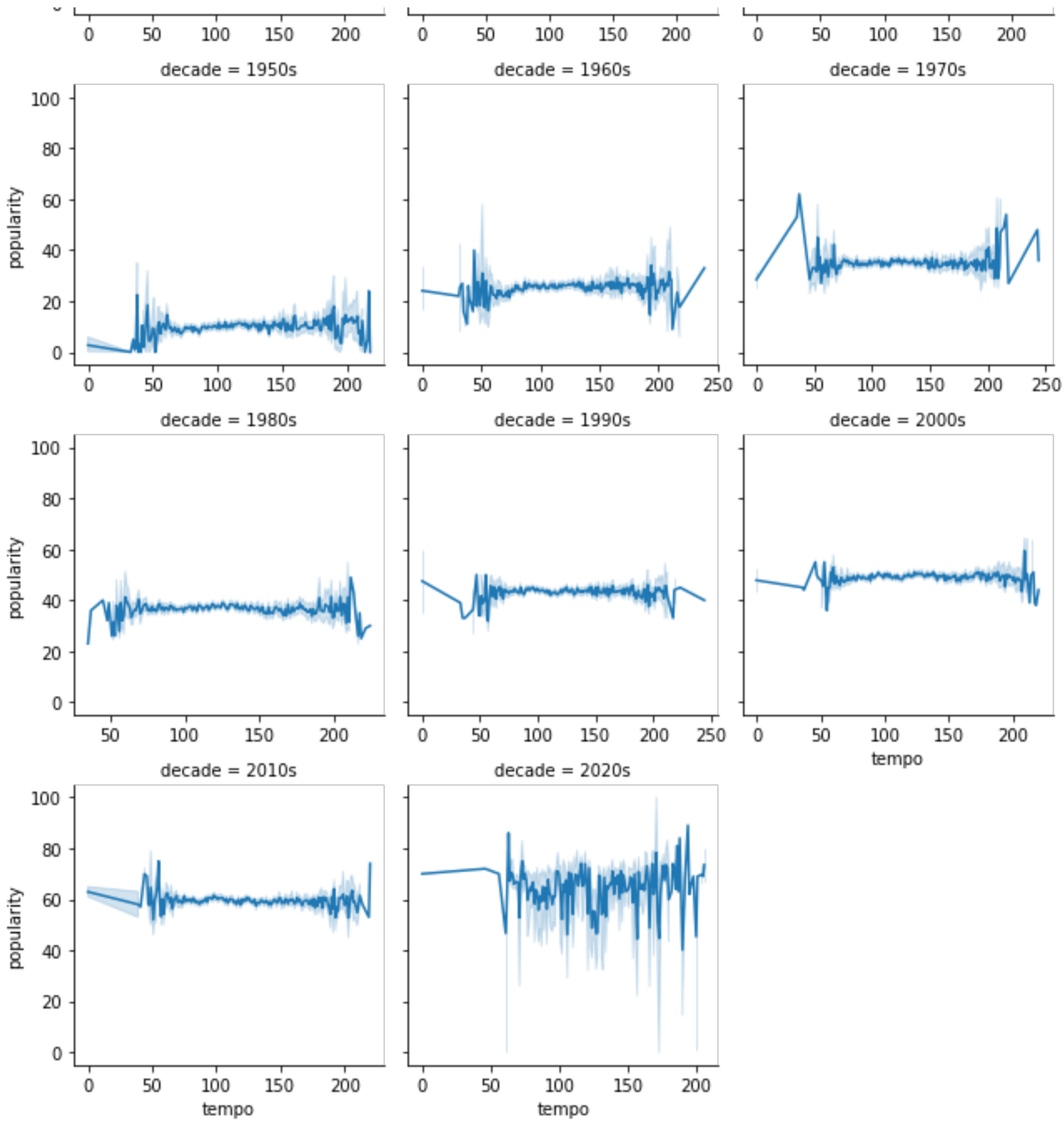





```
g = sns.FacetGrid(relplot_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.lineplot, 'tempo', "popularity")
```

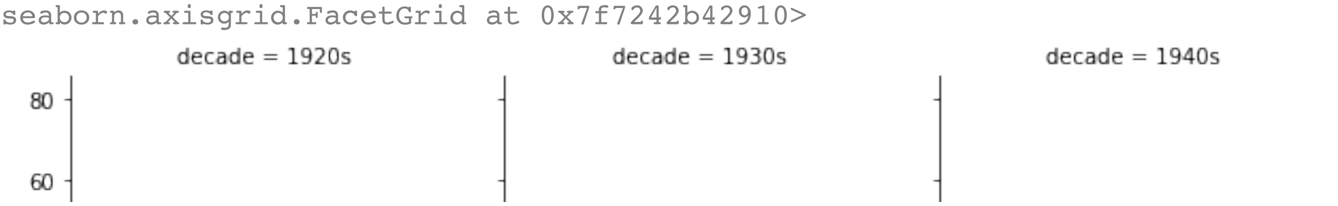
```
<seaborn.axisgrid.FacetGrid at 0x7f7242bfd9a0>
```

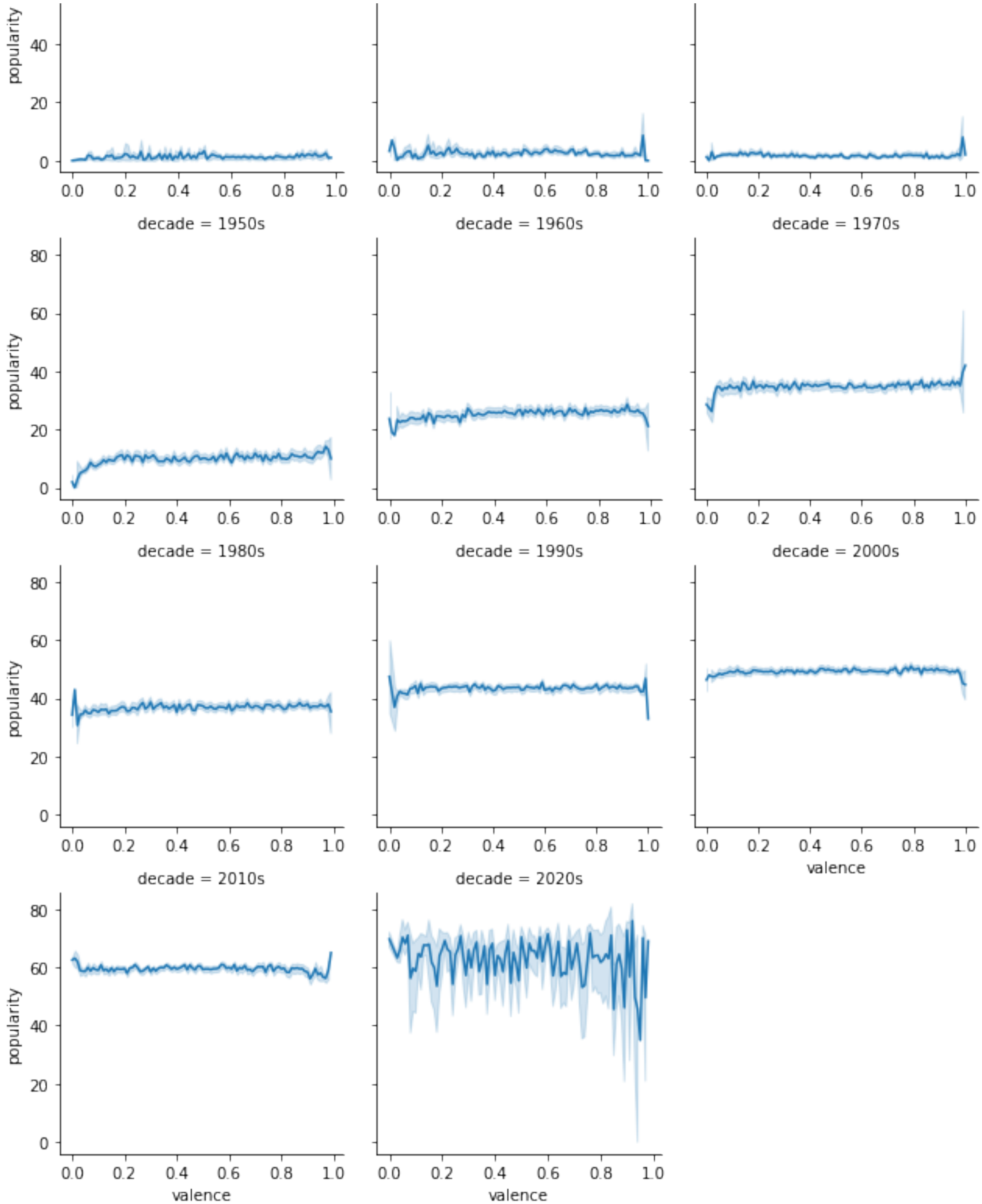





```
g = sns.FacetGrid(relplot_data, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.lineplot, 'valence', "popularity")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f7242b42910>
```




popularity
decade = 1950s
decade = 1960s
decade = 1970s
decade = 1980s
decade = 1990s
decade = 2000s
decade = 2010s
decade = 2020s
valence

Feature Selection

```
corr_table = X_unscaled_all_interactions.corr()['popularity'].sort_values(key=abs,
pd.DataFrame(corr_table)[:15]
```

	popularity
acousticness	-0.593345
energy	0.497488
loudness	0.466546
danceability decade_2010s	0.459437
tempo decade_2010s	0.455195
energy decade_2010s	0.438103
acousticness decade_1940s	-0.415330
valence decade_2010s	0.411938
tempo decade_1940s	-0.410924
danceability decade_1940s	-0.401696
loudness decade_1940s	0.400164
loudness decade_2010s	-0.391430
valence decade_1940s	-0.371414
acousticness decade_1950s	-0.363151
energy decade_1940s	-0.352126

It is important to isolate the feature from the interaction in viewing this plot. In many cases, the relationship between the feature and popularity is greater than the association between the interaction. This suggests that passing in the feature may be more ideal, and that passing in the extra information of the decade does not improve the model

```
X_unscaled_no_popularity = X_unscaled_all.drop(['popularity'], axis = 1)
X_unscaled_inter_no_popularity = X_unscaled_all_interactions.drop(['popularity'],


sc0 = StandardScaler()
X_scaled_all = pd.DataFrame(sc0.fit_transform(X_unscaled_no_popularity), index=X_u
y = spotify_data_with_dummies['popularity']


sc1 = StandardScaler()
X_scaled_all_interactions = pd.DataFrame(sc1.fit_transform(X_unscaled_inter_no_pop


pca1 = PCA(n_components=len(X_scaled_all.columns))
latent_vars = pca1.fit_transform(X_scaled_all)


print ("Variance explained by each latent variable in PCA: ", pca1.explained_varia
print("\n")
```

```
Variance explained by each latent variable in PCA:  [0.24396335 0.12075394 0.
 0.06859278 0.06442256 0.05751638 0.03956368 0.02971379 0.02569044
 0.00939707]
```

```
#Better way of attributing how much a latent dimension attributes for variance

latent_data = [X_scaled_all.columns]

for j in range(0,3):
    latent_data.append(np.round(pca1.components_[j],4))

latent_df = pd.DataFrame(np.array(latent_data).T, columns = ['Column', 'Latent Dim
```

```
latent_df.sort_values(by='Latent Dim1', key=abs, ascending=False)
```

	Column	Latent Dim1	Latent Dim2	Latent Dim3
3	energy	-0.4747	0.2646	0.125
8	loudness	-0.4609	0.2325	0.0258
0	acousticness	0.4381	-0.1464	-0.2297
5	instrumentalness	0.3146	0.0755	0.0245
1	danceability	-0.314	-0.3817	-0.3208
12	valence	-0.3073	-0.1382	-0.4927
4	explicit	-0.1849	-0.4513	0.3814
11	tempo	-0.1812	0.1931	-0.0357
10	speechiness	-0.0758	-0.6029	0.2753
2	duration_ms	0.0459	0.2581	0.4336
7	liveness	-0.0436	-0.0172	0.3458
9	mode	0.0282	0.1051	-0.2276
6	key	-0.0281	-0.0397	0.0528

In looking at the latent dimensions of a principal components analysis, we are looking at the dimensions that account for the most variance in the dataset (in an unsupervised setting). The first dimension consists strongly of energy, loudness, and acousticness, suggesting there may be some association between these variables. It is important to note sources of relationships between variables or collinearity before going further in the modeling process (even if it turns out that these variables are used together).

```
rfe_selector = RFE(estimator=LinearRegression(),n_features_to_select = 10, step =
rfe_selector.fit(X_scaled_all, y)
rfe_cols = list(X_scaled_all.columns[rfe_selector.get_support()])

rfe_cols
```

```
['acousticness',
 'danceability',
 'energy',
 'explicit',
 'instrumentalness',
 'liveness',
 'loudness',
 'speechiness',
 'tempo',
 'valence']
```

```
rfe_inter_selector = RFE(estimator=LinearRegression(),n_features_to_select = 30, s
rfe_inter_selector.fit(X_scaled_all_interactions, y)
rfe_inter_cols = list(X_scaled_all_interactions.columns[rfe_inter_selector.get_sup

rfe_inter_cols
```

```
['loudness',
 'liveness decade_1930s',
 'liveness decade_1940s',
 'liveness decade_1950s',
 'loudness decade_1920s',
 'loudness decade_1930s',
 'loudness decade_1940s',
 'loudness decade_1950s',
 'loudness decade_1960s',
 'loudness decade_1970s',
 'loudness decade_1980s',
 'loudness decade_1990s',
 'loudness decade_2000s',
 'loudness decade_2010s',
 'loudness decade_2020s',
 'tempo decade_1920s',
 'tempo decade_1930s',
 'tempo decade_1940s',
 'tempo decade_1950s',
 'tempo decade_1960s',
 'tempo decade_1990s',
 'tempo decade_2000s',
 'tempo decade_2010s',
 'tempo decade_2020s',
 'valence decade_1920s',
 'valence decade_1930s',
 'valence decade_1940s',
 'valence decade_1950s',
 'valence decade_2000s',
 'valence decade_2010s']
```

Recursive feature elimination is used to determine the best features for a predictive model. While this is not perfect by any means, it is an effective way of reducing variables down to a small, effective size. It appears that liveness, loudness, tempo, and valence have the largest associations, which is consistent with our EDA.

```
model_cols = rfe_inter_cols.copy()
```

```
X_unscaled_all_interactions[model_cols].corr()
```

```
X_unscaled_all_interactions[model_cols].corr()
```

	loudness	liveness decade_1930s	liveness decade_1940s	liveness decade_1950s	loudness decade_1920s
loudness	1.000000	-0.077358	-0.151798	-0.152686	0.181594
liveness decade_1930s	-0.077358	1.000000	-0.046119	-0.052826	0.029571
liveness decade_1940s	-0.151798	-0.046119	1.000000	-0.067337	0.037695
liveness decade_1950s	-0.152686	-0.052826	-0.067337	1.000000	0.043176
loudness decade_1920s	0.181594	0.029571	0.037695	0.043176	1.000000
loudness decade_1930s	0.168433	-0.744064	0.053657	0.061459	-0.034404
loudness decade_1940s	0.321221	0.054948	-0.697634	0.080227	-0.044911
loudness decade_1950s	0.324068	0.064645	0.082403	-0.692825	-0.052836
loudness decade_1960s	0.197020	0.064151	0.081773	0.093664	-0.052432
loudness decade_1970s	0.092918	0.064725	0.082504	0.094502	-0.052901
loudness decade_1980s	0.113057	0.062849	0.080114	0.091763	-0.051368
loudness decade_1990s	0.040557	0.061998	0.079029	0.090521	-0.050673
loudness decade_2000s	-0.079662	0.058888	0.075064	0.085979	-0.048130
loudness decade_2010s	-0.051852	0.057296	0.073035	0.083655	-0.046830
loudness decade_2020s	-0.021650	0.016510	0.021045	0.024106	-0.013494
tempo decade_1920s	-0.125316	-0.030005	-0.038248	-0.043809	-0.892357
tempo decade_1930s	-0.096092	0.775340	-0.054811	-0.062781	0.035144

decade_1930s					
tempo decade_1940s	-0.203465	-0.056517	0.748868	-0.082518	0.046193
tempo decade_1950s	-0.195844	-0.066435	-0.084685	0.731980	0.054299
tempo decade_1960s	-0.072252	-0.067012	-0.085420	-0.097842	0.054771
tempo decade_1990s	0.104025	-0.066974	-0.085372	-0.097786	0.054740
tempo decade_2000s	0.261016	-0.067024	-0.085435	-0.097858	0.054780
tempo decade_2010s	0.264776	-0.066877	-0.085248	-0.097644	0.054661
tempo decade_2020s	0.076089	-0.018852	-0.024030	-0.027525	0.015408
valence decade_1920s	-0.103573	-0.028444	-0.036257	-0.041530	-0.814333
valence decade_1930s	-0.071358	0.745828	-0.052184	-0.059772	0.033460
valence decade_1940s	-0.120366	-0.050827	0.697415	-0.074211	0.041543
valence decade_1950s	-0.113900	-0.059171	-0.075424	0.662258	0.048362
valence decade_2000s	0.266173	-0.061913	-0.078920	-0.090396	0.050603
valence decade_2010s	0.277269	-0.060546	-0.077178	-0.088400	0.049486

30 rows × 30 columns

In looking at this table, there is definitely collinearity, however, many of these variables are "switched" off when another is active, due to the fact that they are interaction variables from different decades, meaning only one interaction variable for a given song metric will be used for a given decade. One can think of this as a linear model being set up for each decade, by use of these different interaction variables.

```
interesting_cols = ['year', 'popularity']


for interesting_col in interesting_cols:
  X_unscaled_all_interactions[interesting_col] = spotify_data_with_dummies[interes


interesting_cols.append('decade')


X_unscaled_all_interactions['decade'] = spotify_data['decade']


X_final_cols_unscaled = X_unscaled_all_interactions.copy()
```

Train/Test Split

```
X_train_pd, X_val_pd, y_train, y_val = train_test_split(X_final_cols_unscaled, y,


# Scale whole matrix of features to prevent information leakage
# Scale for training set and validation set
sc2 = StandardScaler()
X_train = sc2.fit_transform(X_train_pd[model_cols])
X_val = sc2.transform(X_val_pd[model_cols])


# Scale for training set and validation set
sc4 = StandardScaler()
X_train_simple = sc4.fit_transform(X_train_pd[metric_cols])
X_val_simple = sc4.transform(X_val_pd[metric_cols])
```

Simple Model (no interactions)

```
# Fit a classifier with parameters found above
lin_regressor_simple = LinearRegression()
lin_regressor_simple.fit(X_train_simple, y_train)
```

```
LinearRegression()
```

```
for coef, col in zip(lin_regressor_simple.coef_, metric_cols):
  print(f"{col}: {coef}")
```

```
acousticness: -8.70890032844128
danceability: 4.725511699386118
energy: 3.0230875281940155
instrumentalness: -2.1648771721870097
liveness: -1.3400239544709236
loudness: 1.8206445604067238
speechiness: -3.870777766773358
tempo: 0.7732670288440187
valence: -5.992685131340313
```

```
# Predict both class and probability for the training set
y_train_pred_simple = lin_regressor_simple.predict(X_train_simple)

# Predict both class and probability for the test set
y_val_pred_simple = lin_regressor_simple.predict(X_val_simple)
```

```
X_train_pd['simple_pred'] = y_train_pred_simple
X_val_pd['simple_pred'] = y_val_pred_simple
```

Analysis of Fit on Train

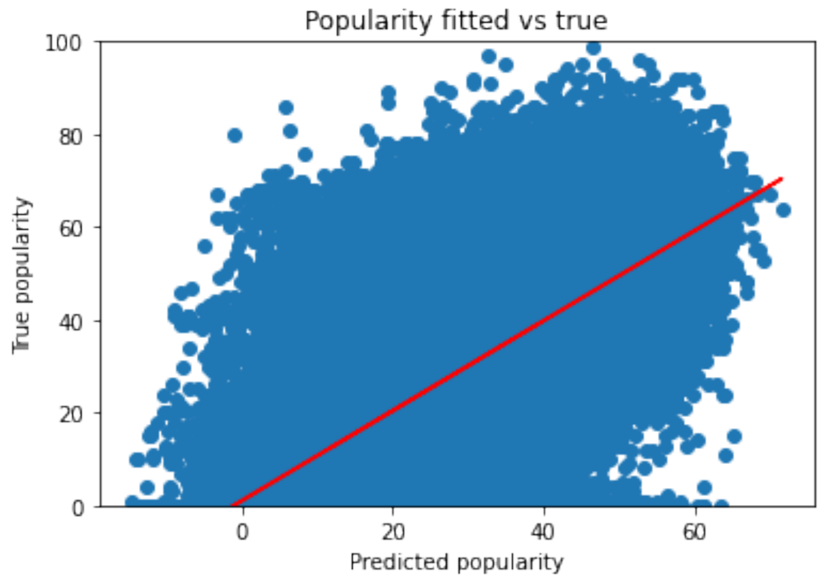
```
print(f"RMSE score train: {mean_squared_error(y_train, y_train_pred_simple)}")
print(f"R^2 score train: {r2_score(y_train, y_train_pred_simple)}")
```

```
RMSE score train: 251.5287560382064
R^2 score train: 0.458596346799527
```

```
a, b = np.polyfit(X_train_pd['simple_pred'], X_train_pd['popularity'], 1)
```

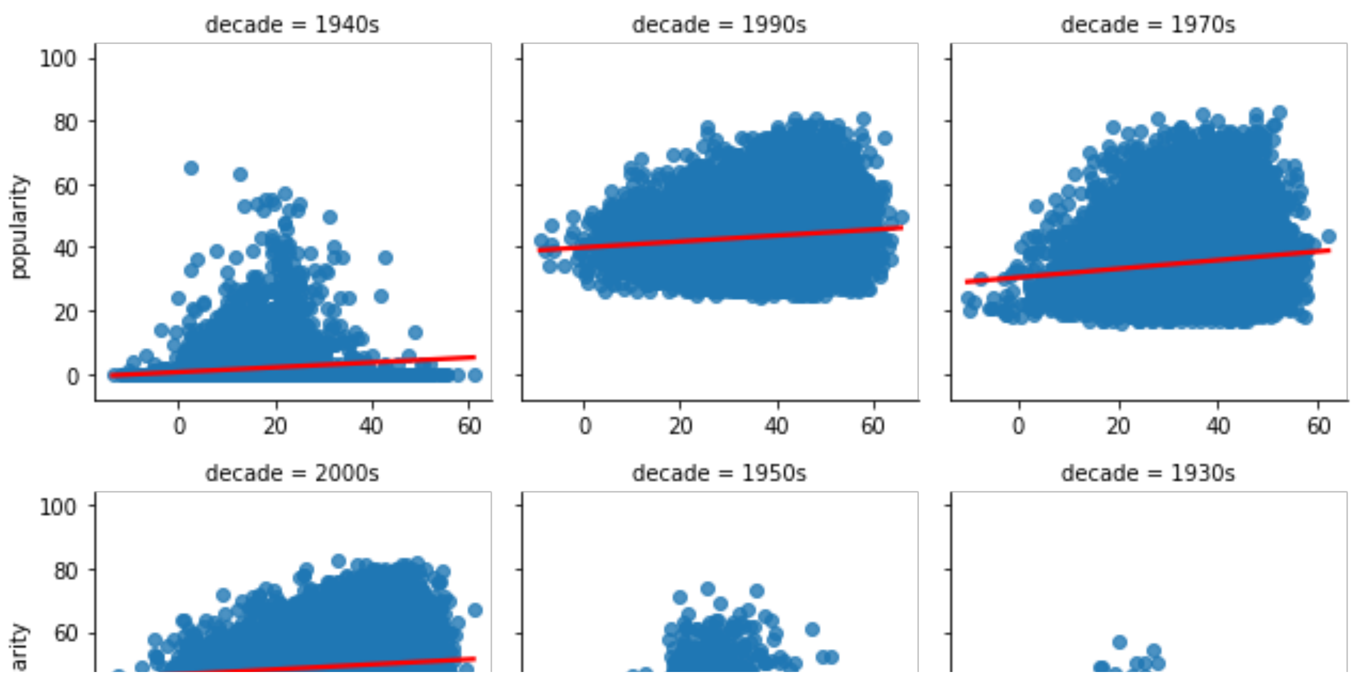
```
plt.scatter(X_train_pd['simple_pred'], X_train_pd['popularity'])
plt.plot(X_train_pd['simple_pred'], a * X_train_pd['simple_pred'] + b, color = 're
plt.xlabel('Predicted popularity')
plt.ylabel('True popularity')
plt.title('Popularity fitted vs true')
plt.ylim(0, 100)
```

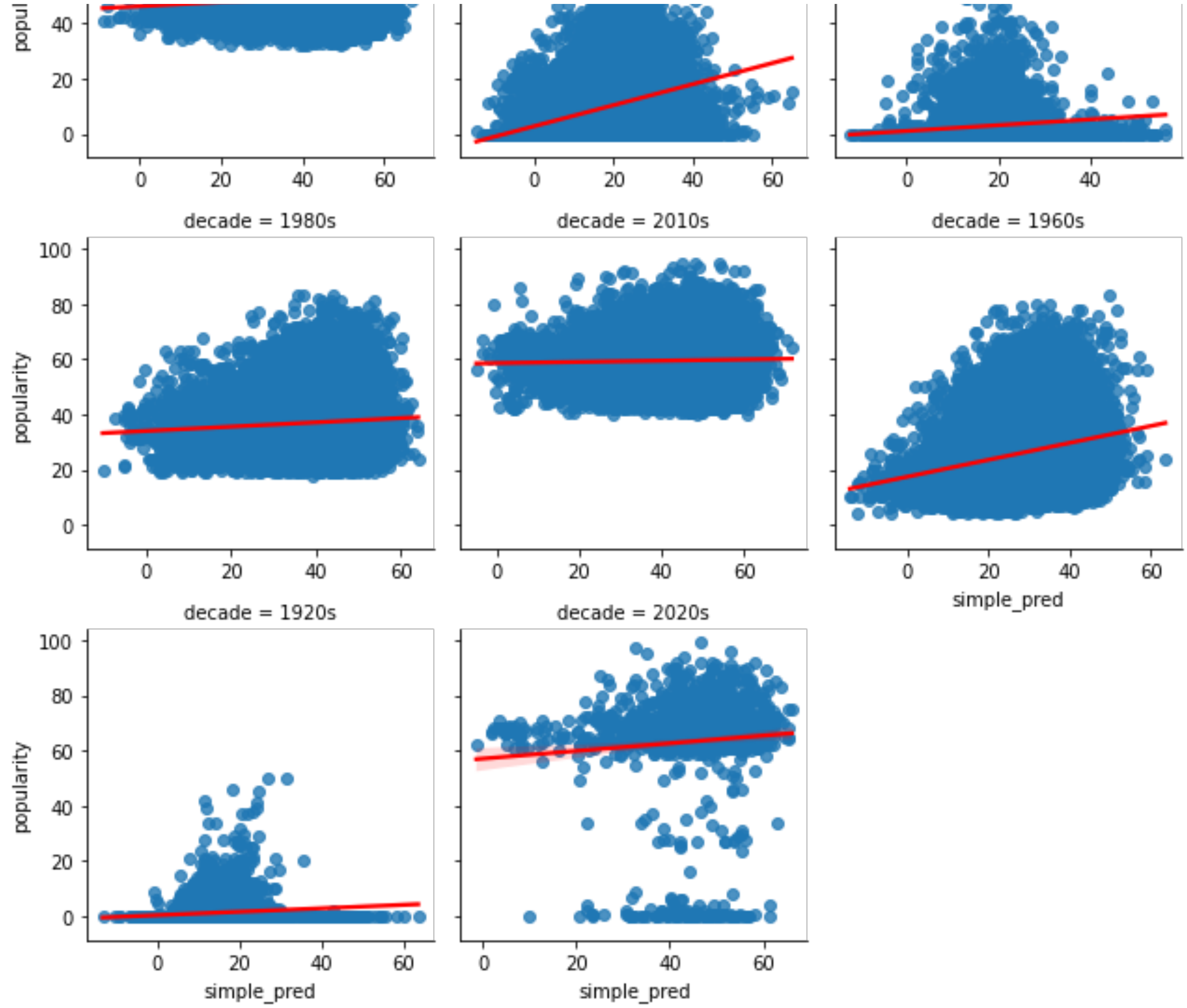
```
(0.0, 100.0)
```




```
g = sns.FacetGrid(X_train_pd, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, "simple_pred", "popularity", line_kws = {'color':'red'})
```

```
<seaborn.axisgrid.FacetGrid at 0x7f84cf0210a0>
```





It seems as though different relationships between the target and a metric over different decades led to an "averaging" effect over coefficients not allowing any of the relationships to be learned. Thus, this could be benefited by including interaction effects by decade for more nuanced relatipnships between metrics and popularity scores.

Analysis of Fit on Val

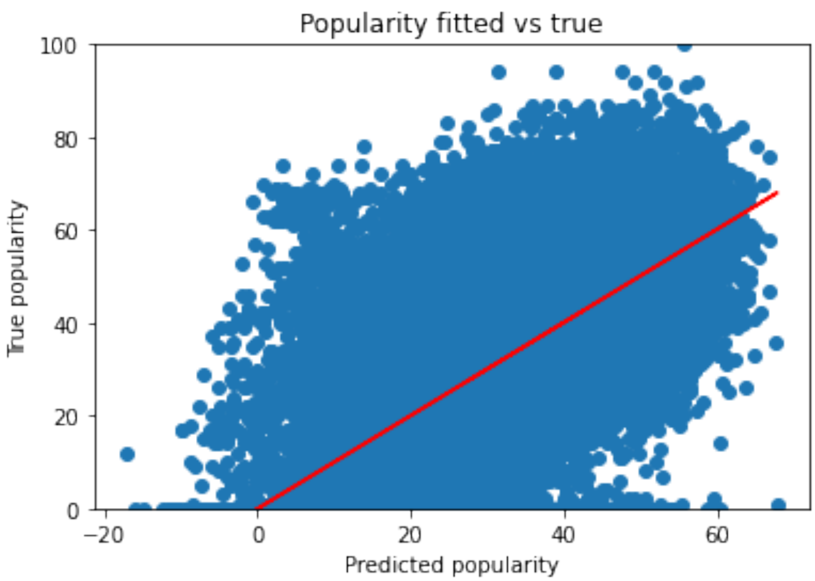
```
print(f"RMSE score val: {mean_squared_error(y_val, y_val_pred_simple)}")
print(f"R^2 score val: {r2_score(y_val, y_val_pred_simple)}")
```

```
RMSE score val: 253.76717755170372
R^2 score val: 0.4594513660578803
```

```
a_val, b_val = np.polyfit(X_val_pd['simple_pred'], X_val_pd['popularity'], 1)
```

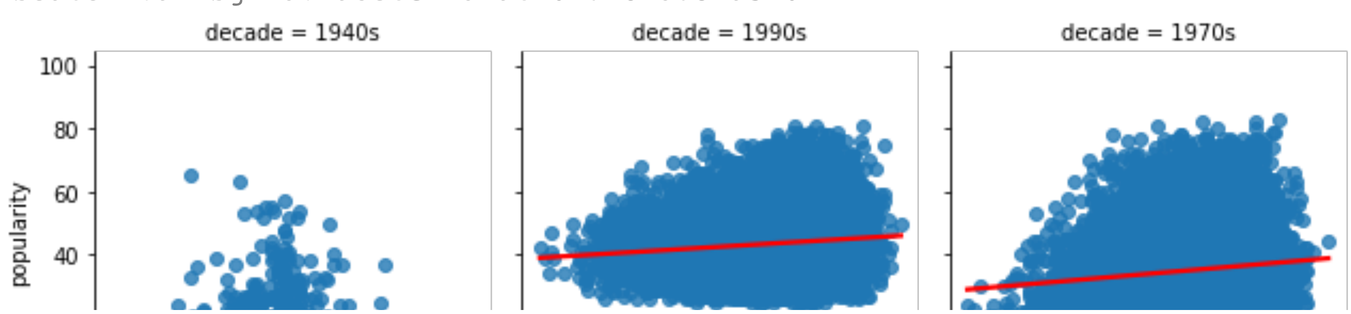
```
plt.scatter(X_val_pd['simple_pred'], X_val_pd['popularity'])
plt.plot(X_val_pd['simple_pred'], a_val * X_val_pd['simple_pred'] + b_val, color =
plt.xlabel('Predicted popularity')
plt.ylabel('True popularity')
plt.title('Popularity fitted vs true')
plt.ylim(0, 100)
```

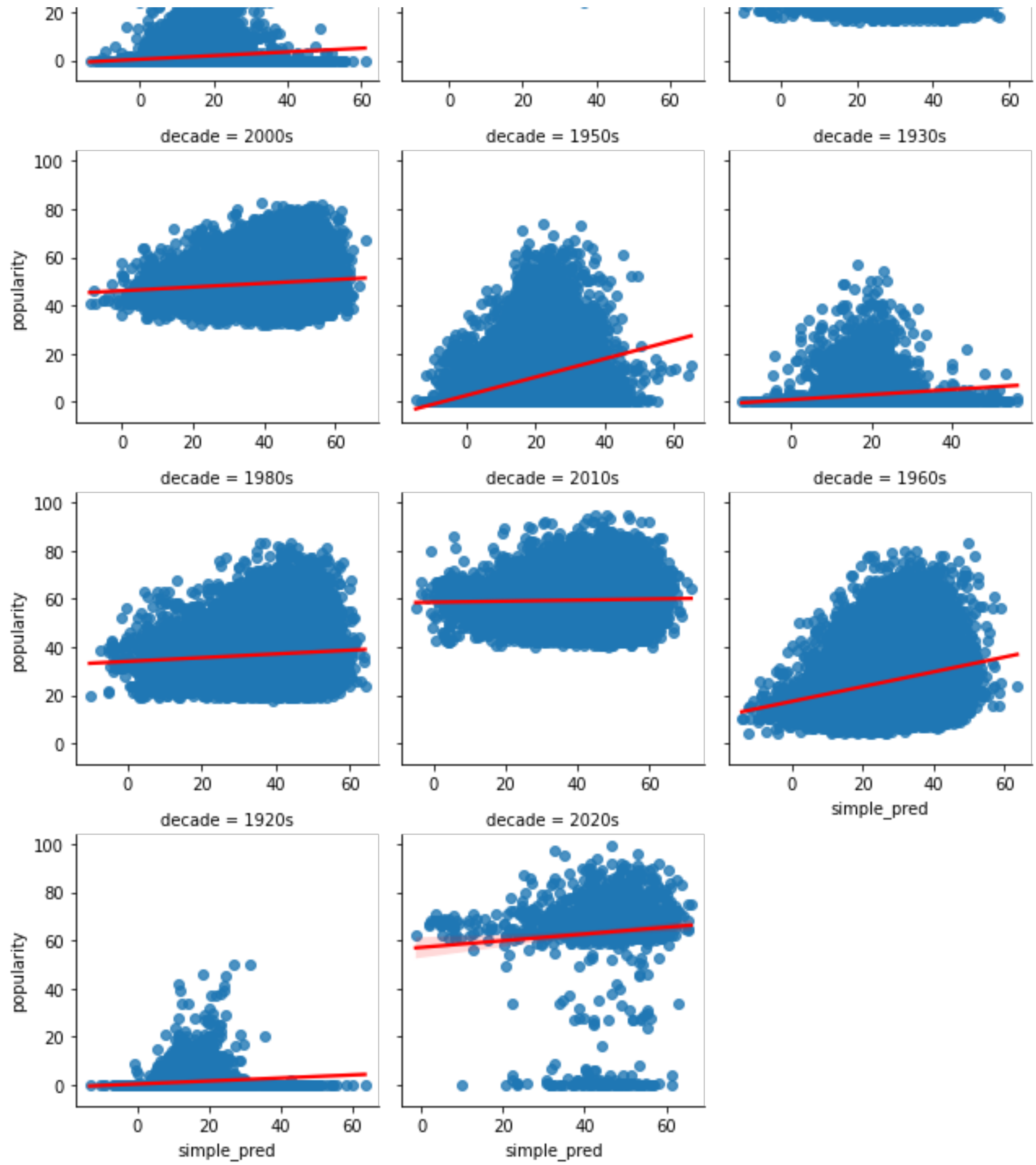
```
(0.0, 100.0)
```




```
g = sns.FacetGrid(X_train_pd, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, "simple_pred", "popularity", line_kws = {'color':'red'})
```

```
<seaborn.axisgrid.FacetGrid at 0x7f84d731a520>
```





I hypothesize that the model is underfit

Modeling with a Temporal Component

```
# Fit a classifier with parameters found above
lin_regressor = LinearRegression()
lin_regressor.fit(X_train, y_train)
```

```
LinearRegression()
```

```
for coef, col in zip(lin_regressor.coef_, model_cols):
  print(f"{col}: {coef}")
```

```
loudness: 1.955019764713686
liveness decade_1930s: -0.7365795509716904
liveness decade_1940s: -0.8330378819254411
liveness decade_1950s: -1.1160949945848566
loudness decade_1920s: 1.653104295372457
loudness decade_1930s: 1.7785506336976447
loudness decade_1940s: 2.468405932240749
loudness decade_1950s: 2.4906150362532293
loudness decade_1960s: 0.7324174939096625
loudness decade_1970s: -0.9017162603079788
loudness decade_1980s: -1.5031282035185953
loudness decade_1990s: -1.2358471318714634
loudness decade_2000s: -1.518849246918148
loudness decade_2010s: -2.76271164629178
loudness decade_2020s: -0.8417724430242697
tempo decade_1920s: -1.9963257698227217
tempo decade_1930s: -2.6059709982508203
tempo decade_1940s: -3.3781652112909732
tempo decade_1950s: -2.683114942238923
tempo decade_1960s: -1.3043675883530166
tempo decade_1990s: 2.2889791360769687
tempo decade_2000s: 2.7200342426489135
tempo decade_2010s: 4.527613081101399
tempo decade_2020s: 2.2093543680604357
valence decade_1920s: -1.1716105895566549
valence decade_1930s: -1.539505935712829
valence decade_1940s: -2.083877729848254
valence decade_1950s: -1.0206622976672919
valence decade_2000s: 1.0991800478138372
valence decade_2010s: 1.5955956244671379
```

The model was able to learn from these features and learned different associations by decade, as shown by the change in sign for valence between earlier songs and more recent songs, as well as similar trends being seen for loudness and liveness. Different decades had different signs in their relation with the target variable. However, that being said, there is likely some collinearity between song metrics themselves, such that this could confound some of the sign switches in the coefficients.

```
# Predict both class and probability for the training set
y_train_pred = lin_regressor.predict(X_train)

# Predict both class and probability for the test set
y_val_pred = lin_regressor.predict(X_val)
```

```
X_train_pd['pred'] = y_train_pred
X_val_pd['pred'] = y_val_pred
```

```
X_train_pd['pred']
```

```
724       -0.033742
94924     42.124325
66065     33.504859
14854     46.486826
46826     49.371208
            ...
97639     60.901769
95939     48.002215
152315    59.446199
117952     1.692481
43567     34.982841
Name: pred, Length: 127431, dtype: float64
```

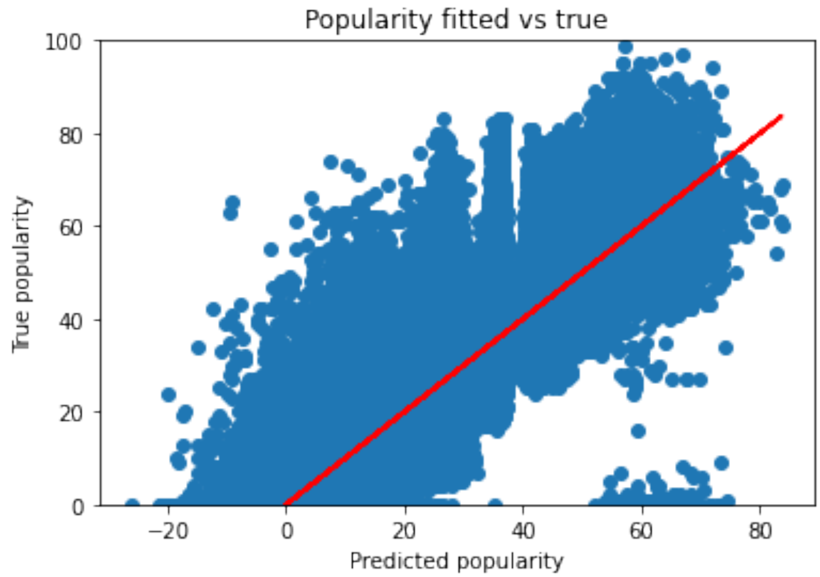
Analysis of Fit on Train

```
print(f"RMSE score train: {mean_squared_error(y_train, y_train_pred)}")
print(f"R^2 score train: {r2_score(y_train, y_train_pred)}")
```

```
RMSE score train: 109.02166043352437
R^2 score train: 0.7653360746247005
```

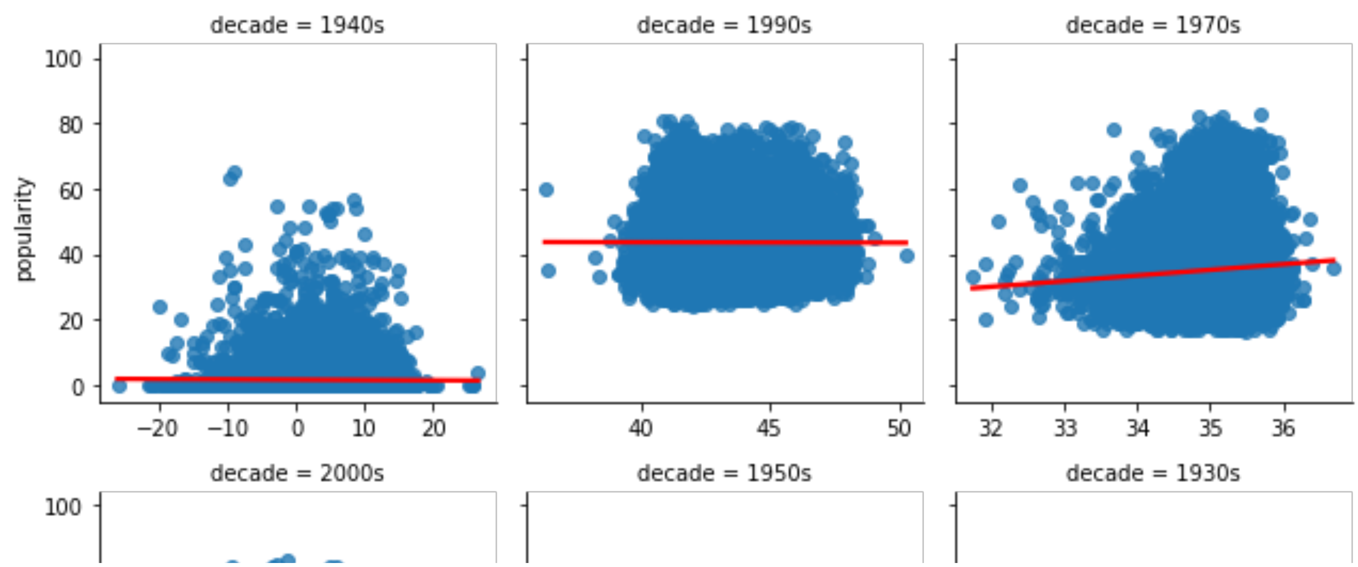
```
a, b = np.polyfit(X_train_pd['pred'], X_train_pd['popularity'], 1)

plt.scatter(X_train_pd['pred'], X_train_pd['popularity'])
plt.plot(X_train_pd['pred'], a * X_train_pd['pred'] + b, color = 'red')
plt.xlabel('Predicted popularity')
plt.ylabel('True popularity')
plt.title('Popularity fitted vs true')
plt.ylim(0, 100)
```

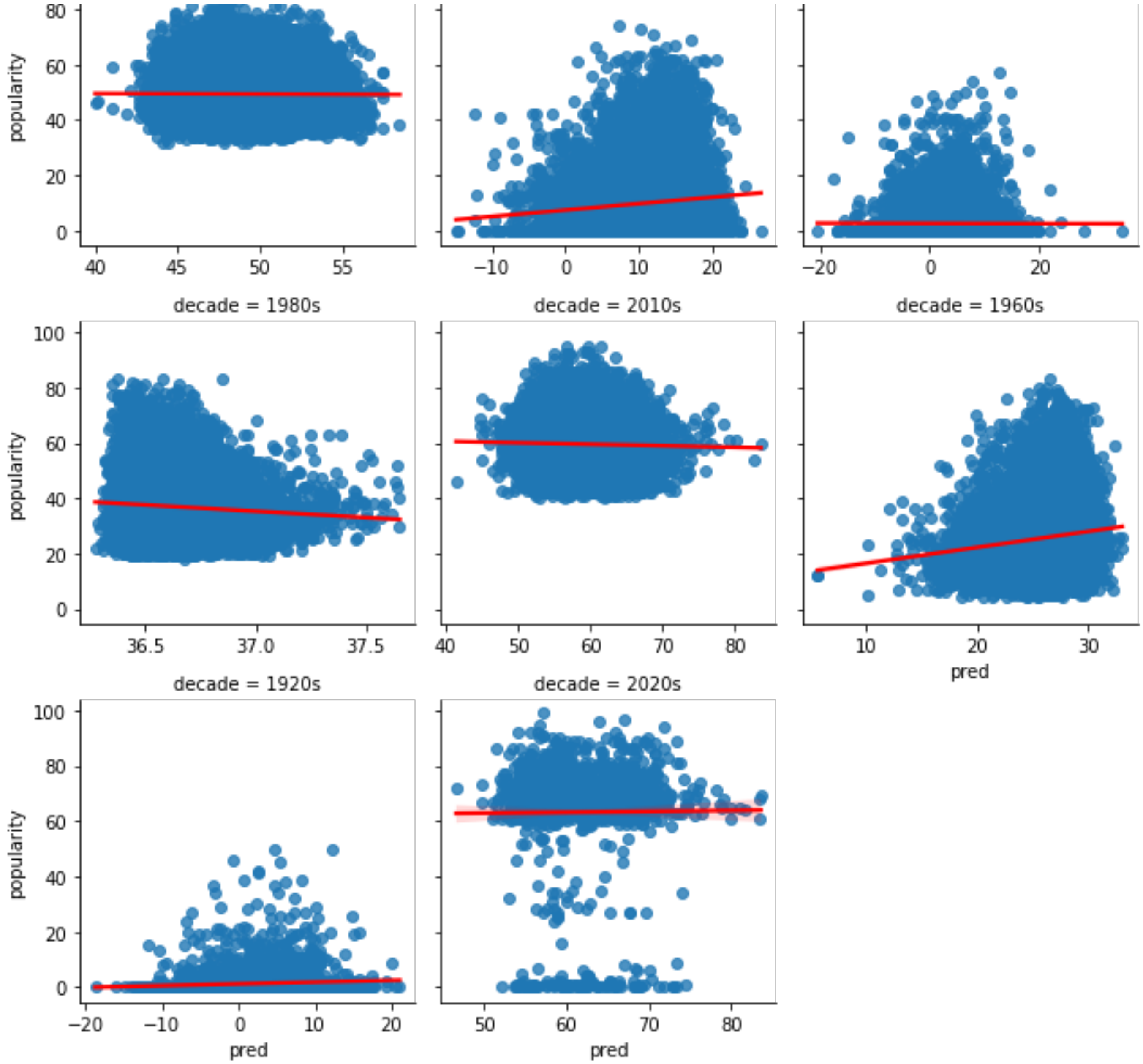
```
(0.0, 100.0)
```




```
g = sns.FacetGrid(X_train_pd, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, "pred", "popularity", line_kws = {'color':'red'})
```

```
<seaborn.axisgrid.FacetGrid at 0x7f85335ea550>
```





It seems as though different relationships between the target and a metric over different decades led to an "averaging" effect over coefficients not allowing any of the relationships to be learned. Thus, this could be benefited by including interaction effects by decade for more nuanced relatipnships between metrics and popularity scores.

Analysis of Fit on Validation

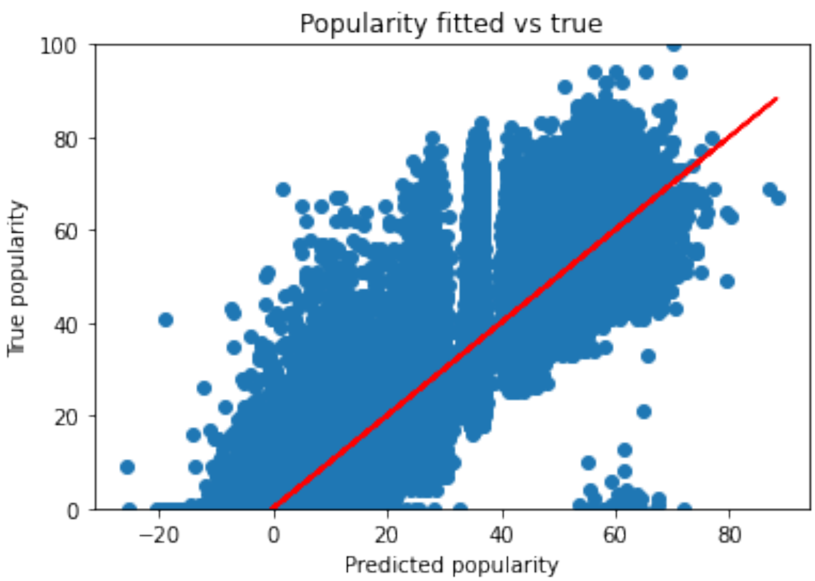
```
print(f"RMSE score val: {mean_squared_error(y_val, y_val_pred)}")
print(f"R^2 score val: {r2_score(y_val, y_val_pred)}")
```

```
RMSE score val: 110.30700501688433
R^2 score val: 0.7650354098138841
```

```
a_val, b_val = np.polyfit(X_val_pd['pred'], X_val_pd['popularity'], 1)
```

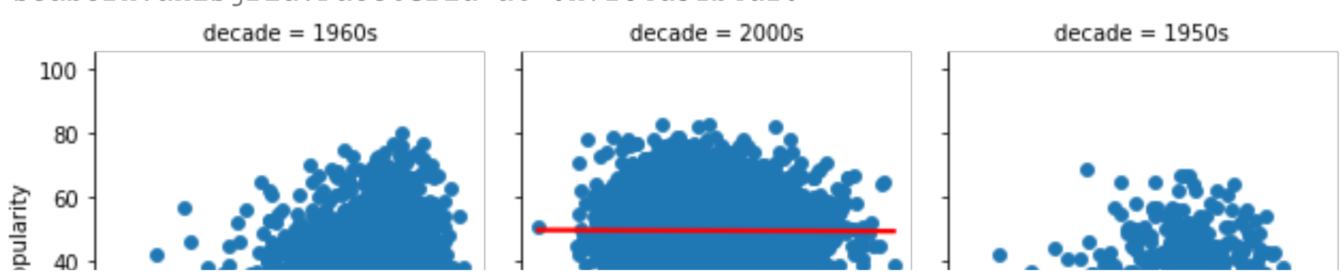
```
plt.scatter(X_val_pd['pred'], X_val_pd['popularity'])
plt.plot(X_val_pd['pred'], a_val * X_val_pd['pred'] + b_val, color = 'red')
plt.xlabel('Predicted popularity')
plt.ylabel('True popularity')
plt.title('Popularity fitted vs true')
plt.ylim(0, 100)
```

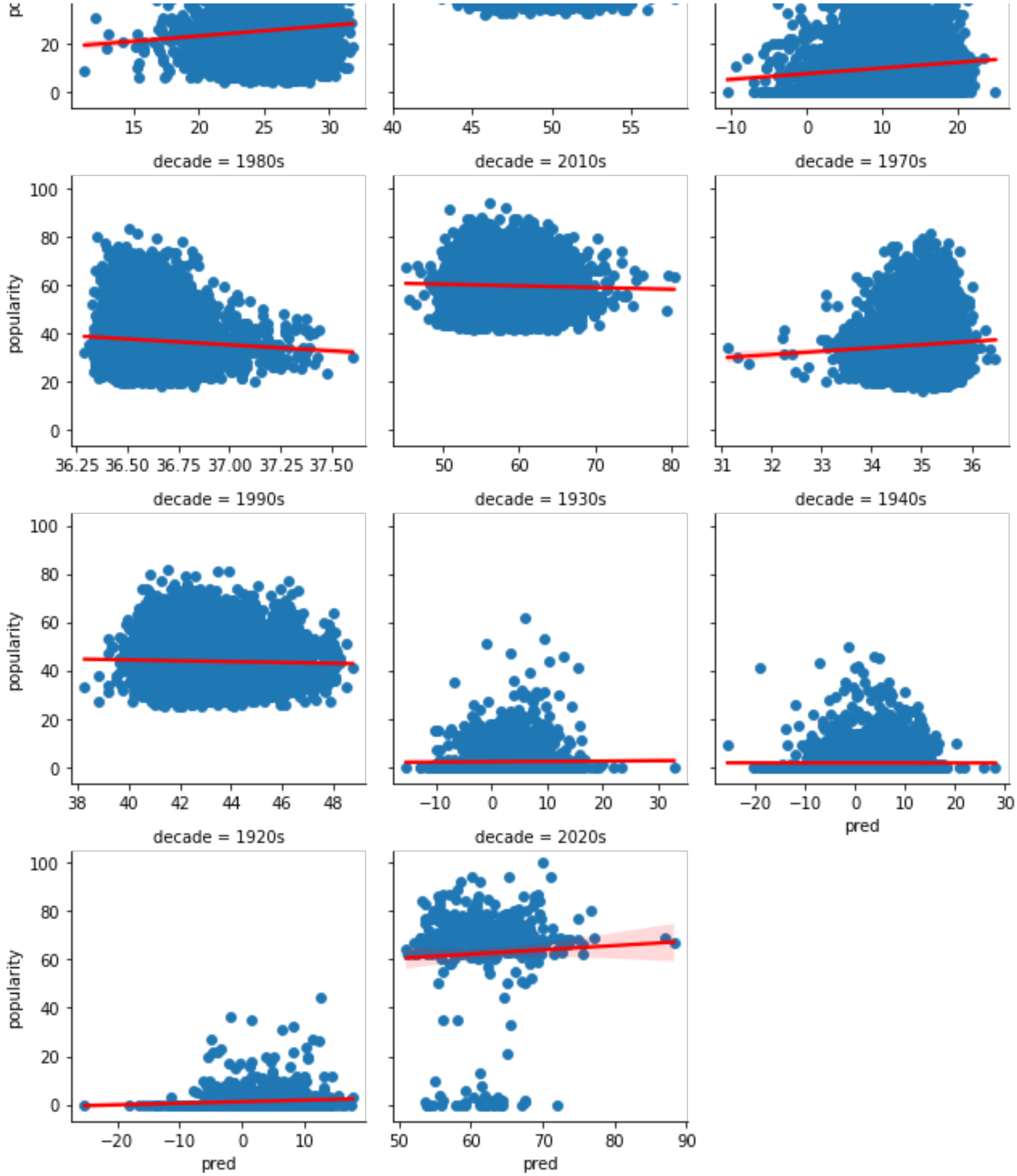
```
(0.0, 100.0)
```




```
g = sns.FacetGrid(X_val_pd, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.scatterplot, "pred", "popularity")
g.map(sns.regplot, "pred", "popularity", line_kws = {'color':'red'})
```

```
<seaborn.axisgrid.FacetGrid at 0x7f84d51b4df0>
```





Exploration of Change Point Detection

Ultimately, due to a time constraint change point detection was not able to be tested for the model. This may have proved beneficial, as the decade splits created a sparser data set with many features to learn. Finding the change points for each variable in their relations with popularity would have created a model that was more precise in terms of its engineered features, and would have allowed it to better learn the 'eras' of music.

Final Thoughts on the Data

It is clear from the scatter plots that the data here has high variance- songs with very similar song metrics can and often will have widely different popularities. This is ultimately because the data itself is not nuanced enough to produce a truly great predictor of a song's popularity.

In thinking about what goes into a popular song nowadays, while there are absolutely trends in how songs are made, this will not be able to solely tell you the popularity. There is so much music nowadays that most music with any combination of song metrics will not have a high popularity, since it is challenging to break the glass ceiling.

Furthermore, more data is definitely required to truly predict a song's popularity score. For example, two artists (e.g. Kendrick Lamar and Billy Joel)releasing songs with the same song metrics will lead to wildly different results. There are many factors that go into songs making the charts nowadays, including how famous an artist already is, its use in social media/pop culture leading to it trending, as well as other factors that make people like a given song. Additionally, data about what niche the artist/its fans fall into could be helpful to know how a song and its given metrics falls into a niche. While this would be helpful, it is still really challenging to create a model that will predict any kind of break through from a small artist, given data on fame. However, one can make a model that ranks songs well in terms of their ability to become popular, given a bunch of small artists, which seems like a great use case of this kind of song data.

Ultimately, music is quite hard to quantify, and more data is required to truly determine what will make people like and listen to songs.

Extra: Attempt to Model the Data Better with Neural Networks

Here, we explore Neural Networks to create an uninterpretable, but perhaps better predictive model for the data. There is a lot of data, such that neural networks may train better in this setting. Furthermore, neural networks can learn complex relationships, interactions, and dependencies between variables, helping reduce the need for manual feature engineering and feature selection.

While LSTMs are often used do the sequential nature of Recurrent Neural Networks and the LSTMs ability to use memory and forget gates to keep and drop important information, the data we have is time series but not sequential, as the observations are generally independent of each other, but rather the observations and their co-variates/popularity scorres are just not independent from time/era.

Thus, we use an Artificial Neural Network with a ReLU final activation function. In doing so, we bring a longer training and inference time, and the potential of overfitting. Additionally, we do not have the time to truly tune the neural network in terms of its architecture, dropout rate, regularization, optimizer, etc. Meanwhile, a Linear Regression can easily be tuned (for parameters like L1/L2 penalties and such) with a Grid Search in minutes. Finally, with the neural network, we lose interpretability in favor of predictive accuracy.

```
#NN- Sequence of layers
from keras.models import Sequential
#Dense- output layer
from keras.layers import Dense
#LSTM layers
from keras.layers import LSTM
#Dropout for regularization (prevent overfitting)
from keras.layers import Dropout
import tensorflow as tf
```

```
nn_cols = list(spotify_data.columns)
bad_cols = ['id', 'release_date', 'name', 'artists', 'decade', 'popularity']

for col in bad_cols:
  nn_cols.remove(col)
```

Here, I split the training set further, into a true validation set, so the other holdout set of calidation data can be used as a test data. This is because I want to tune my model, but also want an objective test set to evaluate my performance on (same set as linear regression). In tuning the model, I use early stopping, which uses validation performance to determine which epoch of the model is best. Using the other validation set would be cheating and a form of data leakage, leading to high performance.

```
X_true_train_pd, X_val_tune_pd, y_true_train, y_val_tune = train_test_split(X_trai
```

```
# Scale whole matrix of features to prevent information leakage
# Scale for training set and validation set
sc3 = StandardScaler()
X_train_nn = sc3.fit_transform(X_true_train_pd[nn_cols])
X_val_nn = sc3.transform(X_val_tune_pd[nn_cols])
X_test_nn = sc3.transform(X_val_pd[nn_cols])
```

```
# #Predicting continuous output- regression
# regressor = Sequential()
# #First LSTM layers
# #units - LSTM cells/mem units (increase dimensionality)
# #return_sequences - true as stacked LSTM (several layers,false (default) only on
# #input_shape - shape of input containing x train (timesteps, indicators)- first
# regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.s
# #Classic number to use- 20% of neurons ignored in training (10 neurons)
# regressor.add(Dropout(0.2))
# #Input shape does not need to be specified as recognized due to units
# regressor.add(LSTM(units = 50, return_sequences = True))
# regressor.add(Dropout(0.2))
# regressor.add(LSTM(units = 50, return_sequences = True))
# regressor.add(Dropout(0.2))
# #Last layer- return sequences is false
# regressor.add(LSTM(units = 50))
# regressor.add(Dropout(0.2))
# #1 output- stock price
# regressor.add(Dense(units = 1))
```

```
def create_model(input_shape, metrics, optimizer, loss_function, output_bias=None)
  if output_bias is not None:
    output_bias = tf.keras.initializers.Constant(output_bias)
  ann = tf.keras.models.Sequential()
  ann.add(tf.keras.layers.Dense(input_shape = (input_shape, ), units=200, activati
  ann.add(tf.keras.layers.BatchNormalization())
  ann.add(tf.keras.layers.Dropout(0.4))
  ann.add(tf.keras.layers.Dense(units=100, activation='relu'))
  ann.add(tf.keras.layers.BatchNormalization())
  ann.add(tf.keras.layers.Dropout(0.4))
  ann.add(tf.keras.layers.Dense(units=1, activation='relu', bias_initializer = out
  ann.compile(optimizer = optimizer, loss = loss_function, metrics=metrics)
  return ann


regressor = create_model(input_shape = X_train_nn.shape[1], metrics = ['mse'], opt


EPOCHS = 100
BATCH_SIZE = 32
val_data = (X_val_nn, y_val_tune)

earlystopping = tf.keras.callbacks.EarlyStopping(monitor ="val_mse",
                                        mode ="min", patience = 5,
                                        restore_best_weights = True)
```

```
#Number of times to forward and backpropagate
#100- teacher observed convergence
regressor.fit(X_train_nn, y_true_train, epochs = EPOCHS, batch_size = BATCH_SIZE,
```

```
Epoch 1/100
3584/3584 [==============================] - 16s 5ms/step - loss: 95.9063 - m
Epoch 2/100
3584/3584 [==============================] - 14s 4ms/step - loss: 95.5716 - m
Epoch 3/100
3584/3584 [==============================] - 15s 4ms/step - loss: 95.7997 - m
Epoch 4/100
3584/3584 [==============================] - 14s 4ms/step - loss: 95.5758 - m
Epoch 5/100
3584/3584 [==============================] - 14s 4ms/step - loss: 95.6841 - m
Epoch 6/100
3584/3584 [==============================] - 14s 4ms/step - loss: 95.5875 - m
Epoch 7/100
3584/3584 [==============================] - 14s 4ms/step - loss: 95.1620 - m
Epoch 8/100
3584/3584 [==============================] - 14s 4ms/step - loss: 95.0520 - m
Epoch 9/100
3584/3584 [==============================] - 15s 4ms/step - loss: 94.7759 - m
Epoch 10/100
3584/3584 [==============================] - 14s 4ms/step - loss: 95.1040 - m
Epoch 11/100
3584/3584 [==============================] - 14s 4ms/step - loss: 94.6648 - m
<keras.callbacks.History at 0x7f84cf15dca0>
```

```
# Predict both class and probability for the training set
y_train_nn_pred = regressor.predict(X_train_nn)
y_val_tune_nn_pred = regressor.predict(X_val_nn)

# Predict both class and probability for the test set
y_val_nn_pred = regressor.predict(X_test_nn)
```

```
3584/3584 [==============================] - 10s 3ms/step
399/399 [==============================] - 1s 2ms/step
1328/1328 [==============================] - 2s 2ms/step
```

```
X_true_train_pd['nn_pred'] = y_train_nn_pred
X_val_tune_pd['nn_pred'] = y_val_tune_nn_pred
X_val_pd['nn_pred'] = y_val_nn_pred
```

Analysis of Fit on Train (training + tuning data)

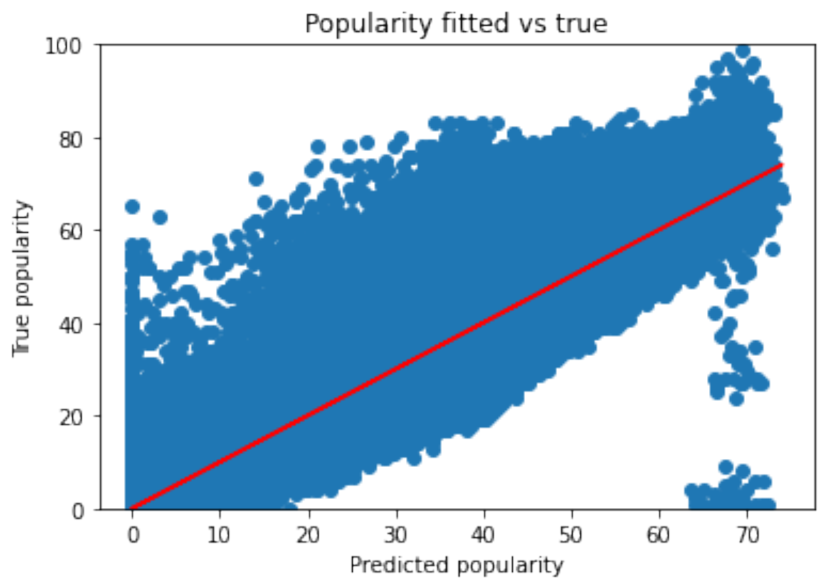
```
print(f"RMSE score train: {mean_squared_error(y_true_train, y_train_nn_pred)}")
print(f"R^2 score train: {r2_score(y_true_train, y_train_nn_pred)}")
```

```
RMSE score train: 83.05018252324457
R^2 score train: 0.8214223559974899
```

```
a, b = np.polyfit(X_true_train_pd['nn_pred'], X_true_train_pd['popularity'], 1)
```

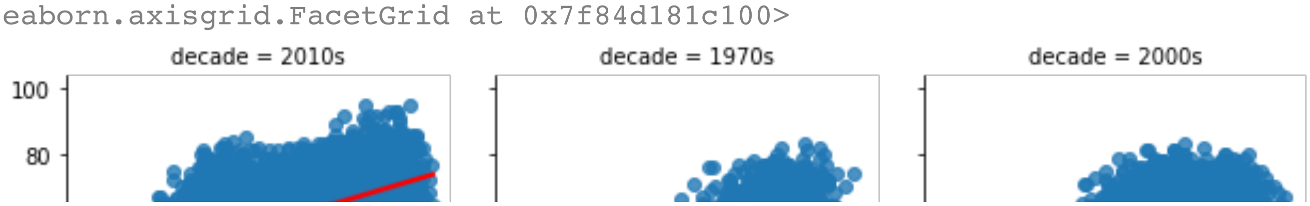
```
plt.scatter(X_true_train_pd['nn_pred'], X_true_train_pd['popularity'])
plt.plot(X_true_train_pd['nn_pred'], a * X_true_train_pd['nn_pred'] + b, color = '
plt.xlabel('Predicted popularity')
plt.ylabel('True popularity')
plt.title('Popularity fitted vs true')
plt.ylim(0, 100)
```

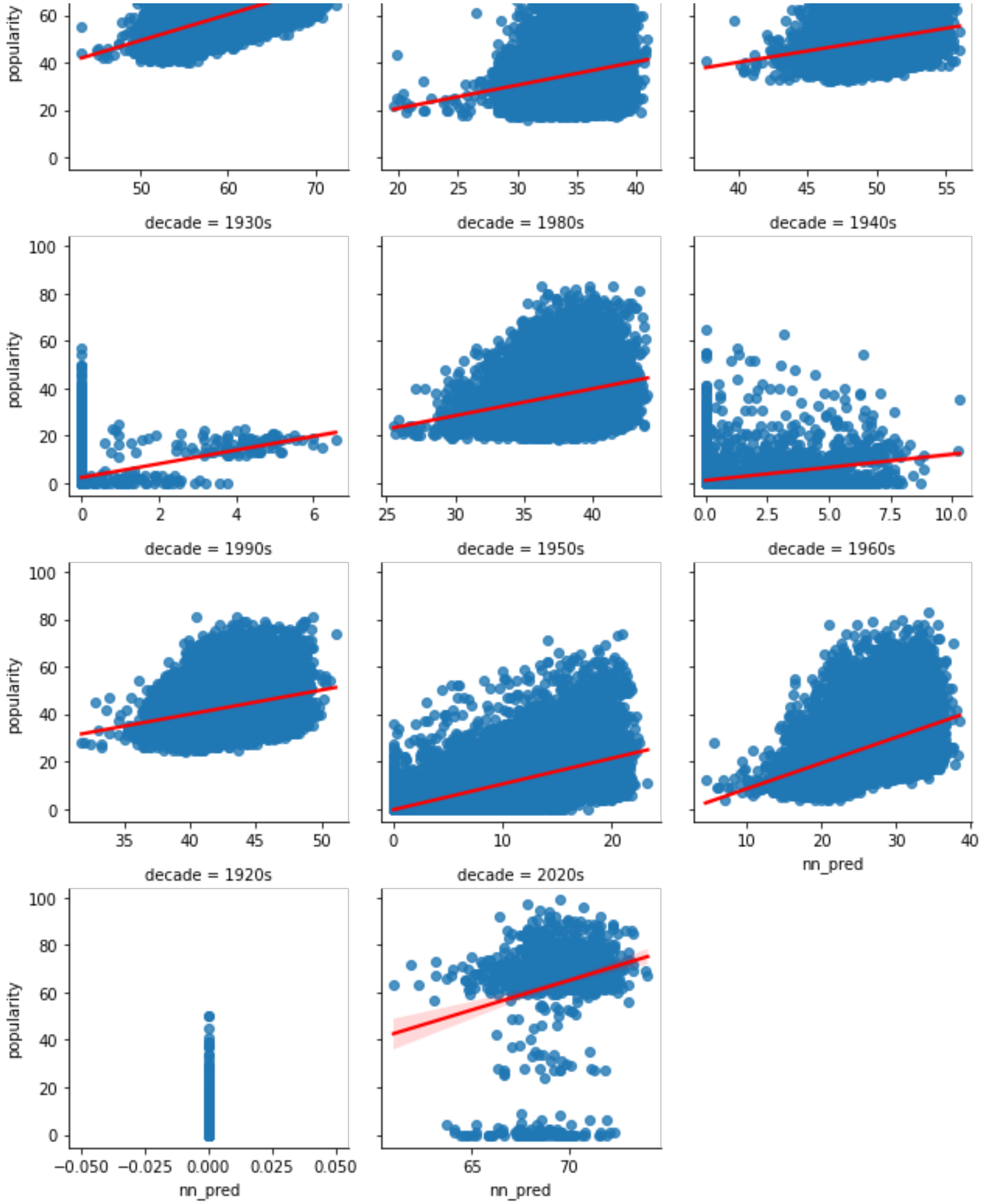
```
(0.0, 100.0)
```




```
g = sns.FacetGrid(X_true_train_pd, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, "nn_pred", "popularity", line_kws = {'color':'red'})
```

```
<seaborn.axisgrid.FacetGrid at 0x7f84d181c100>
```





The models struggles with 1920s and 1930s, even in the training set

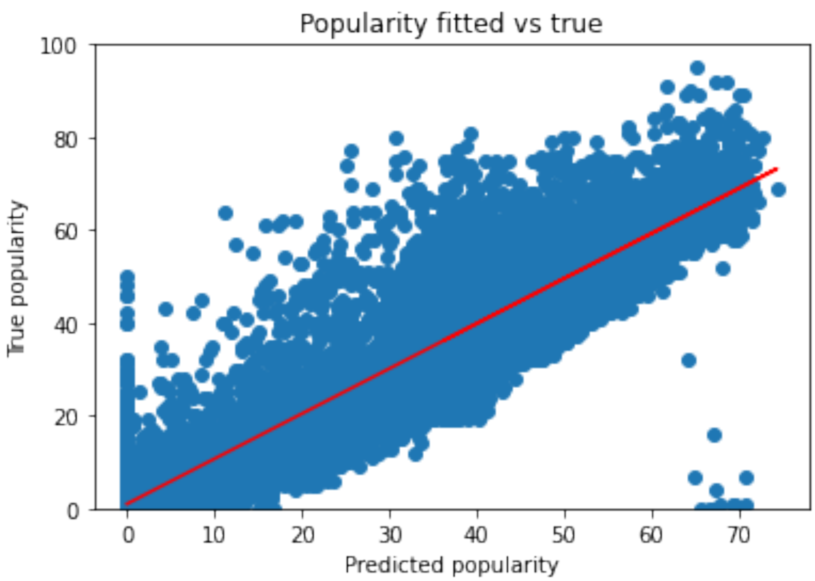
```
print(f"RMSE score train: {mean_squared_error(y_val_tune, y_val_tune_nn_pred)}")
print(f"R^2 score train: {r2_score(y_val_tune, y_val_tune_nn_pred)}")
```

```
RMSE score train: 82.33490789768832
R^2 score train: 0.8211174388568983
```

```
a, b = np.polyfit(X_val_tune_pd['nn_pred'], X_val_tune_pd['popularity'], 1)
```

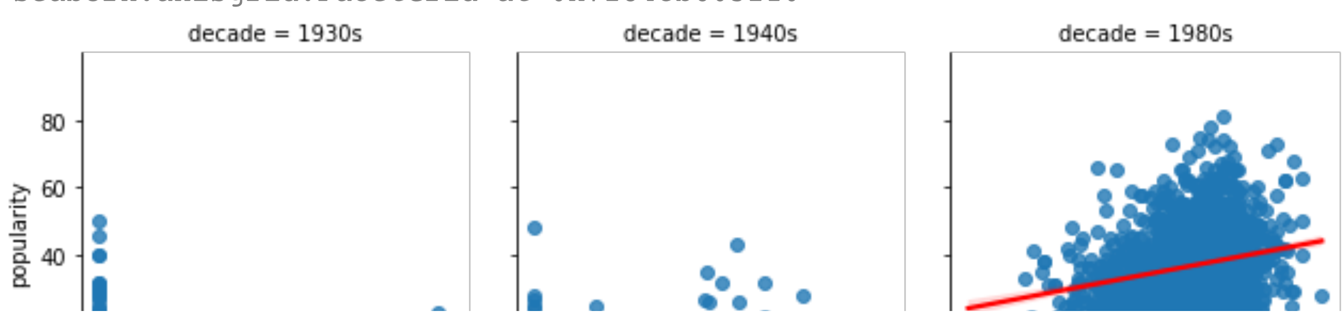
```
plt.scatter(X_val_tune_pd['nn_pred'], X_val_tune_pd['popularity'])
plt.plot(X_val_tune_pd['nn_pred'], a * X_val_tune_pd['nn_pred'] + b, color = 'red'
plt.xlabel('Predicted popularity')
plt.ylabel('True popularity')
plt.title('Popularity fitted vs true')
plt.ylim(0, 100)
```

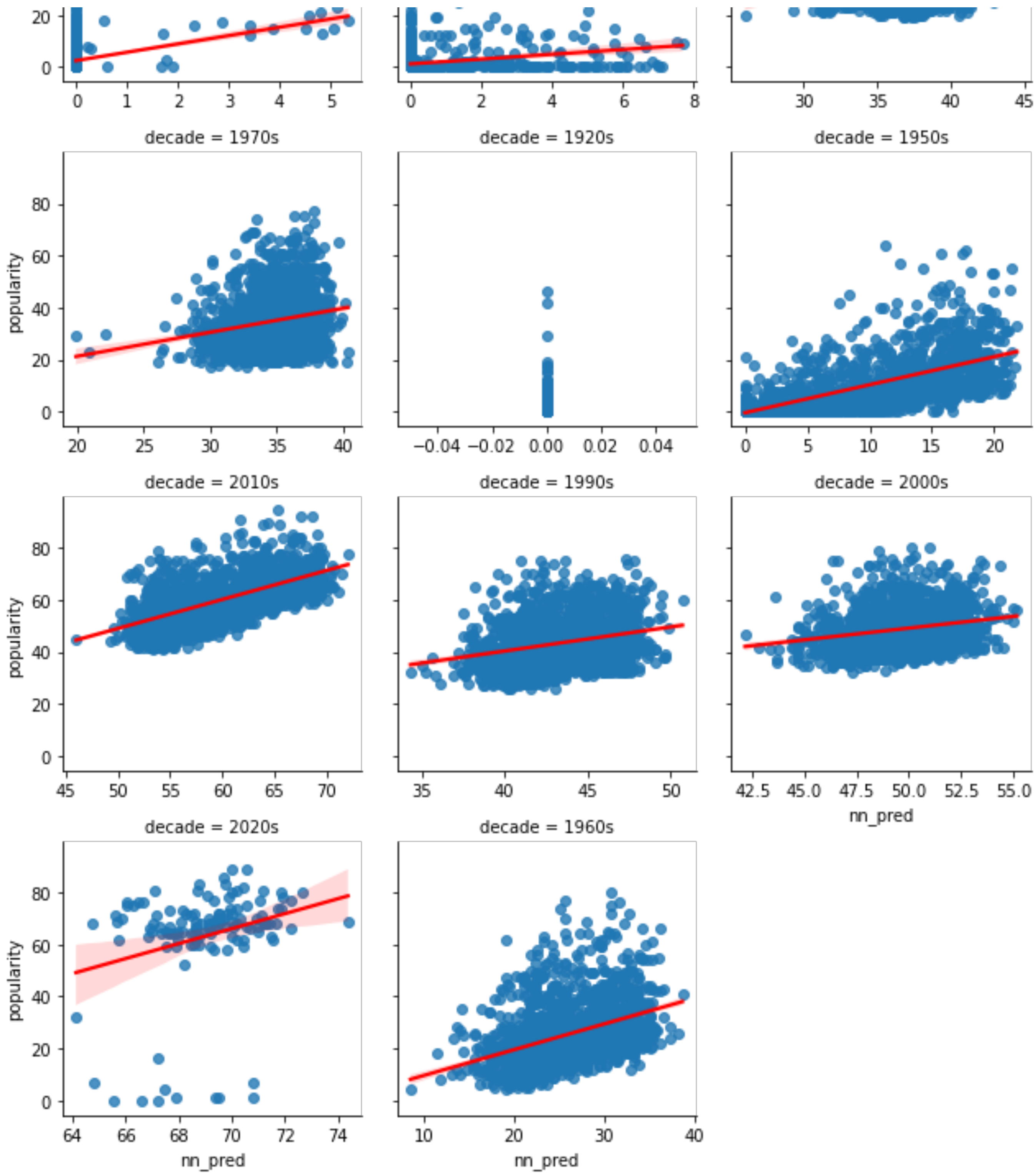
```
(0.0, 100.0)
```




```
g = sns.FacetGrid(X_val_tune_pd, col_wrap = 3, sharex = False, col = "decade")
g.map(sns.regplot, "nn_pred", "popularity", line_kws = {'color':'red'})
```

```
<seaborn.axisgrid.FacetGrid at 0x7f84cb605f10>
```





Analysis of Fit on Validation (Test)

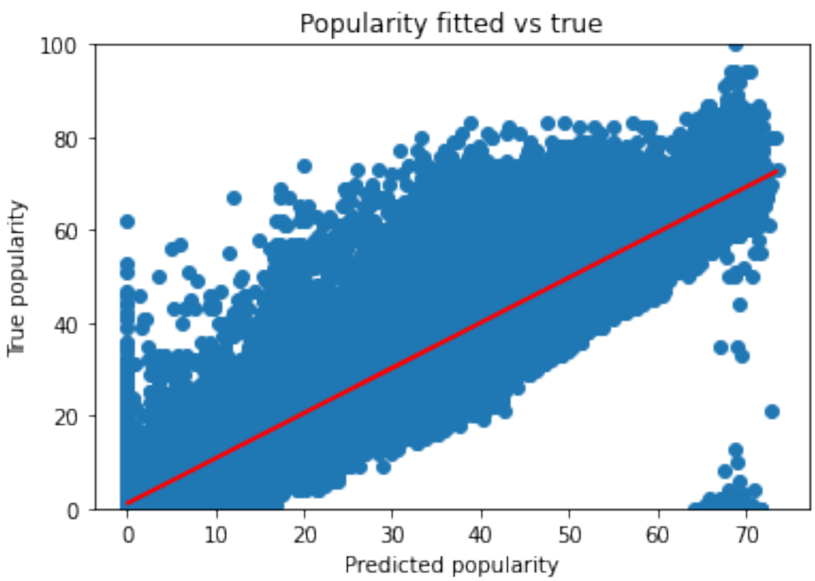
```
print(f"RMSE score val: {mean_squared_error(y_val, y_val_nn_pred)}")
print(f"R^2 score val: {r2_score(y_val, y_val_nn_pred)}")
```

```
RMSE score val: 84.26029562585006
R^2 score val: 0.8205174201977617
```

```
a_val, b_val = np.polyfit(X_val_pd['nn_pred'], X_val_pd['popularity'], 1)
```

```
plt.scatter(X_val_pd['nn_pred'], X_val_pd['popularity'])
plt.plot(X_val_pd['nn_pred'], a_val * X_val_pd['nn_pred'] + b_val, color = 'red')
plt.xlabel('Predicted popularity')
plt.ylabel('True popularity')
plt.title('Popularity fitted vs true')
plt.ylim(0, 100)
```

```
(0.0, 100.0)
```



The Neural Network performs better than the Linear Regression, with its R squared of 0.85 and RMSE of 84.2. The RMSE appears to be high in both the linear regression and neural network due to some especially high residuals taht come from the recent 2020s data. In the future, I would filter out extremely recent data as its popularity score is only low due to the recency of it coming out. This data can then be run through the model as a "forecasting set", in which we predict what these songs future popularity scores will be.

Interestingly, the model can not learn the training data that well, which I believe a further reflection of the fact that the metrics + temporal information is not a great proxy for predicting popularity.

Some further data cleaning to remove non-songs could also be helpful. For example, there are non-song outliers, songs too recent for a popularity score, and a strong sampling bias in the data in terms of recent songs generally being of a higher popularity than other decades (proportions of popularity scores- some of this is trend in music, but also some less known tracks are just not being included).

Colab paid products - Cancel contracts here