Classification analysis of Spotify genres

The Spotify platform has a large database of secondary data that analysts can use to identify patterns and relationships between different characteristsics in songs. In this project we have looked into the possibility of predicting a music genre by taking this secondary data such as loudness or amount of speech in each song. From these characteristics of each song it is possible to implement a machine learning model to predict the genre of music of a given song by it's own characteristsics.

First the data was cleaned and preprocessed for an easier analysis and understanding. This was followed by a broad exploratory analysis of the training data set to find usefull details or features that will aid in refining the model needed. Using this information, we started to refine the important features and remove data that might be counter productive when creating a machine learning model. This process involved a lot of trial and error but eventually resulted in a final data set that could be plugged into machine learning models for results. This is our final process and how we finalised the model with a random classifier tree machine learning solution.

Required libraries

This notebook makes use of several Python packages that come with the **Anaconda Python distribution**. Here we import any of the potential packages that could be used for easyness in a machine learning project such as this one. Important ones to note include:

- pandas: A data frame structure that stores data while being quick and efficient.
- numpy: A library in python adding support for large-multi dimensional arrays.
- matplotlib: The standard package for plotting in Python.
- scikit-learn: A machine learning library for python, essential for a classification task. This also includes all the individual imported functions and models needed.

Note: printing the current version of scikit-learn to make sure it is the same as the scikit library functions being used.

In [9]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.linear_model import SGDClassifier
from sklearn.multiclass import OneVsOneClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import BaggingClassifier
print(sklearn.__version__)
```

```
1.0.2
```

1. Read data

Reading the data from local CSV files downloaded from the Spotify dataset available at **Kaggle**. This dataset includes the secondary data for 453 songs on the Spotify platform, as well as a test set without genre for prediction.

In [10]:

```
train = pd.read_csv('CS98XClassificationTrain.csv')
test = pd.read_csv('CS98XClassificationTest.csv')
```

2. Exploratory Analysis

The first step involved some simple and quick data cleaning to remove unwanted string columns as well as removing any rows with NaN values. While the artist column could potentially be used, we thought the huge variety of artists as well as the harder processing string format would prove not very usefull.

Next was finding out vital information to explore the dataset. This included finding the amount of total genres, amount of songs in each genre category as well as useful mean and standard devations of each genre. This gave us a quick picture into what this data could tell us, such as the very large amount of genres with minimal songs in them. Furthermore the scale in varience of the columns in each genre was a useful tool to see where they where generally placed. Proceding this, individual plots were created to provide a better visual picture.

In [11]:

```
train.head()
```

Out[11]:

	Id	title	artist	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	pop	ge
0	1	My Happiness	Connie Francis	1996	107	31	45	-8	13	28	150	75	3	44	a standa
1	2	Unchained Melody	The Teddy Bears	2011	114	44	53	-8	13	47	139	49	3	37	N
2	3	How Deep Is Your Love	Bee Gees	1979	105	36	63	-9	13	67	245	11	3	77	a standa
3	4	Woman in Love	Barbra Streisand	1980	170	28	47	-16	13	33	232	25	3	67	a standa
4	5	Goodbye Yellow Brick Road - Remastered 2014	Elton John	1973	121	47	56	-8	15	40	193	45	3	63	g r

In [12]:
```
train.drop(columns = ['Id', 'title', 'artist'], inplace = True)
test.drop(columns = ['Id', 'title', 'artist'], inplace = True)
```

In [13]:
```
train = train.dropna()
testa = test.dropna()
testa
```

Out[13]:

	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	pop
0	2005	154	93	65	-3	75	74	213	1	18	72
1	1994	161	39	30	-15	11	14	292	26	3	59
2	1977	64	46	27	-7	12	18	179	38	3	76
3	2010	127	92	71	-9	37	53	216	6	4	50
4	2018	115	46	56	-12	21	34	153	18	3	44
...	...	...	...	...	...	...	...	...	...	...	...
108	2005	125	57	61	-8	38	76	209	3	47	78
109	2010	130	89	67	-6	10	80	215	4	3	44
110	1994	84	58	78	-7	14	76	253	43	27	74
111	1978	127	97	72	-5	12	73	287	6	14	71
112	1986	123	89	53	-4	29	80	249	8	3	83

113 rows × 11 columns

In [14]:
```
train["top genre"].value_counts()
```

Out[14]:
```
adult standards       68
album rock            66
dance pop             61
brill building pop    16
glam rock             16
                      ..
bow pop                1
australian rock        1
boogaloo               1
british comedy         1
alternative rock       1
Name: top genre, Length: 86, dtype: int64
```

In [15]:
```
train.describe()
```

Out[15]:

	year	bpm	nrgy	dnce	dB	live	val	
count	438.000000	438.000000	438.000000	438.000000	438.000000	438.000000	438.000000	438.000
mean	1990.881279	118.326484	60.504566	59.780822	-8.787671	17.605023	59.625571	228.267
std	16.697047	25.175735	22.089660	15.404757	3.591005	13.807492	24.480160	63.426
min	1948.000000	62.000000	7.000000	18.000000	-24.000000	2.000000	6.000000	98.000
25%	1976.000000	100.000000	44.000000	50.000000	-11.000000	9.000000	42.250000	184.500
50%	1993.000000	120.000000	64.000000	62.000000	-8.000000	13.000000	61.000000	224.000

	year	bpm	nrgy	dnce	dB	live	val	
75%	2006.000000	133.000000	78.000000	70.750000	-6.000000	23.000000	80.000000	264.000
max	2019.000000	199.000000	100.000000	96.000000	-1.000000	93.000000	99.000000	511.000

In [16]:

```
train.describe(include=['object'])
```

Out[16]:

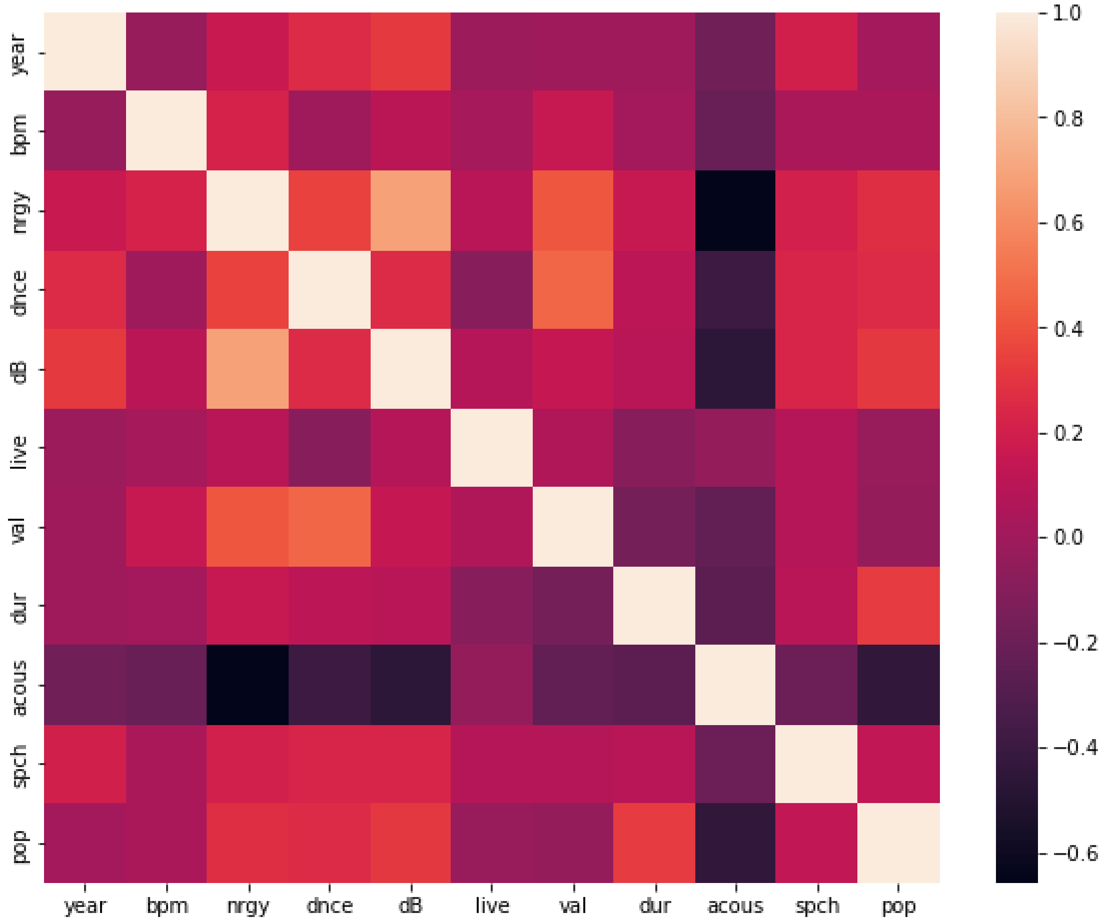
	top genre
count	438
unique	86
top	adult standards
freq	68

In [17]:

```
plt.figure(figsize = (10,8))
sns.heatmap(train.corr())
train.corr()
```

Out[17]:

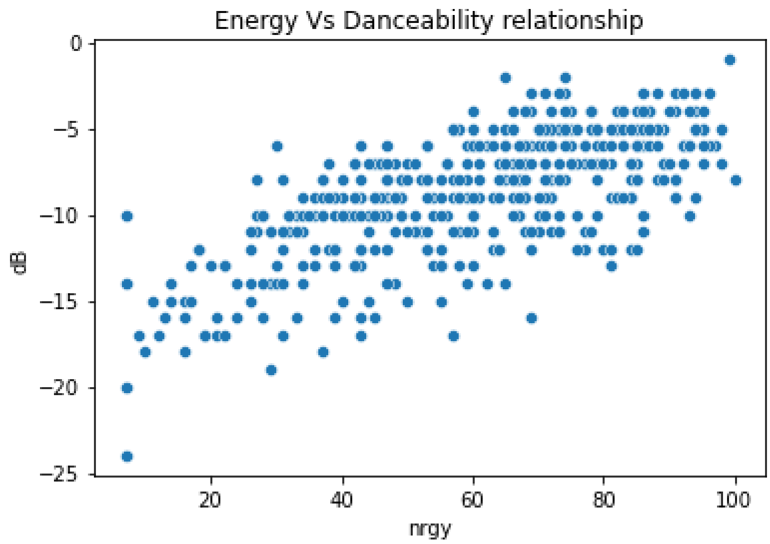
	year	bpm	nrgy	dnce	dB	live	val	dur	aco
year	1.000000	-0.032804	0.162237	0.255916	0.317036	-0.012204	-0.009710	-0.006768	-0.1744
bpm	-0.032804	1.000000	0.219684	-0.009999	0.103035	0.022333	0.151454	0.012545	-0.2177
nrgy	0.162237	0.219684	1.000000	0.345836	0.687504	0.099285	0.411215	0.156341	-0.6582
dnce	0.255916	-0.009999	0.345836	1.000000	0.255577	-0.091241	0.467307	0.108891	-0.3896
dB	0.317036	0.103035	0.687504	0.255577	1.000000	0.080707	0.147616	0.094271	-0.4608
live	-0.012204	0.022333	0.099285	-0.091241	0.080707	1.000000	0.062184	-0.092681	-0.0409
val	-0.009710	0.151454	0.411215	0.467307	0.147616	0.062184	1.000000	-0.159837	-0.2414
dur	-0.006768	0.012545	0.156341	0.108891	0.094271	-0.092681	-0.159837	1.000000	-0.2609
acous	-0.174468	-0.217720	-0.658299	-0.389641	-0.460821	-0.040973	-0.241475	-0.260960	1.0000
spch	0.200995	0.038356	0.204097	0.231468	0.232366	0.084009	0.075536	0.095829	-0.2046
pop	0.018926	0.042695	0.274006	0.256099	0.312952	-0.025493	-0.040035	0.321028	-0.4437



A heatmap was produced to notice any correlations between the secondary song data. There are a few notible correlations here which make a lot of sense such as the acousitcs amount with energy, dance, pop and decibells. As acoustic songs only usually involve acoustic instruments by themselves with no other sounds sources, it makes sense they have a negative correlation with the four categories that feature loud and exciting features.

In [18]:
```
sns.scatterplot(x= 'nrgy',y = 'dB',data = train)
plt.title("Energy Vs Danceability relationship")
```

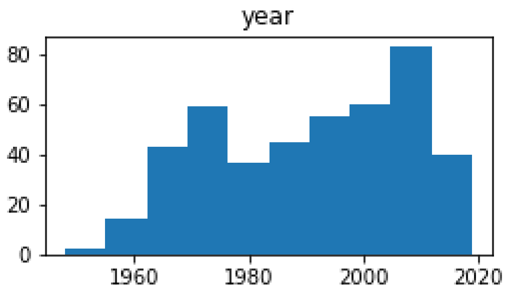
Out[18]: `Text(0.5, 1.0, 'Energy Vs Danceability relationship')`

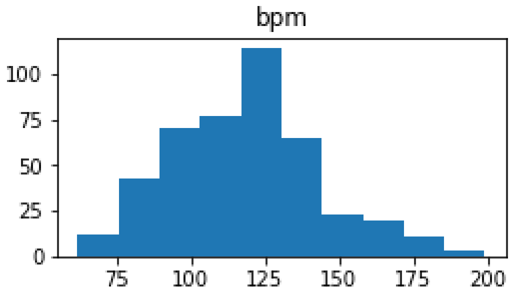


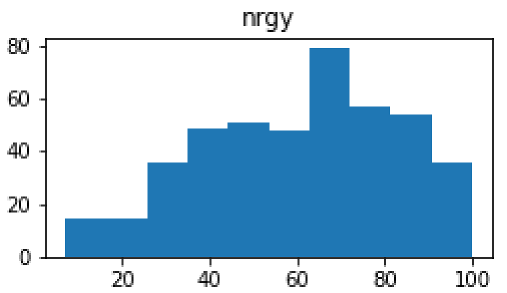
The relationship between energy and danceability can be seen here quite easily with an obvious slope showing the correlation between the two columns. There is a posibility of removing a column to simplify the model if needed as they are both very similar.

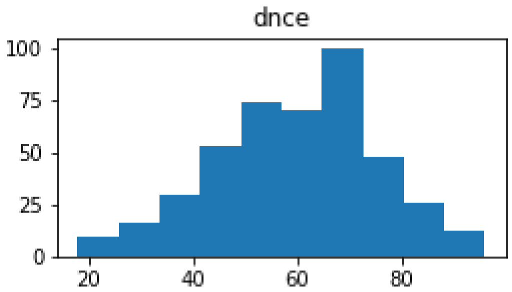
In [27]:

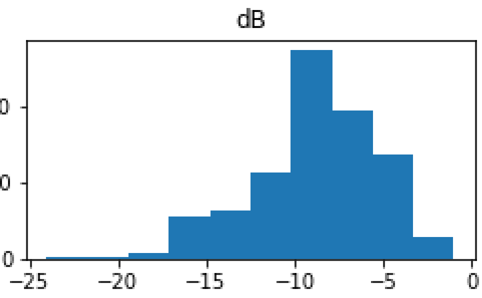
```
for i in train.columns:
    plt.figure(figsize=(4,2))
    plt.hist(train[i])
    plt.title(i)
    plt.show()
```

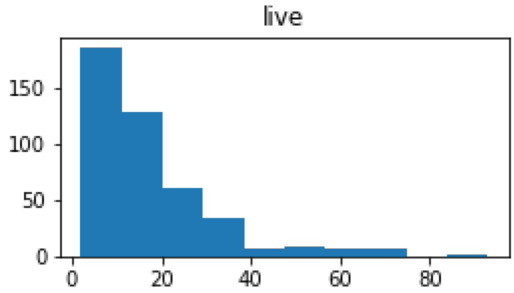


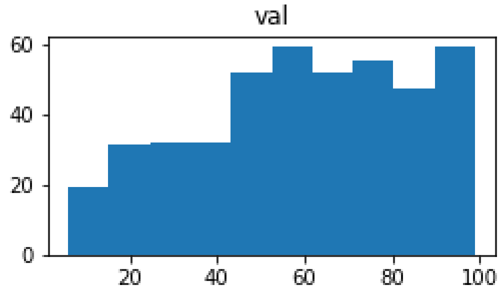


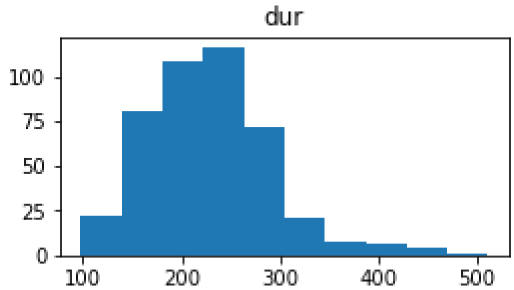


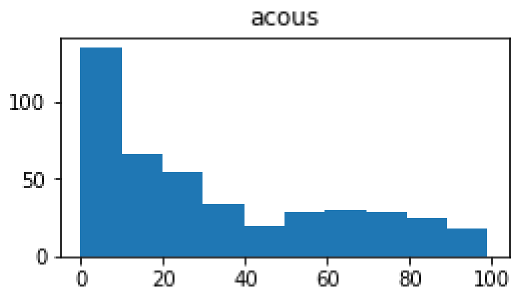


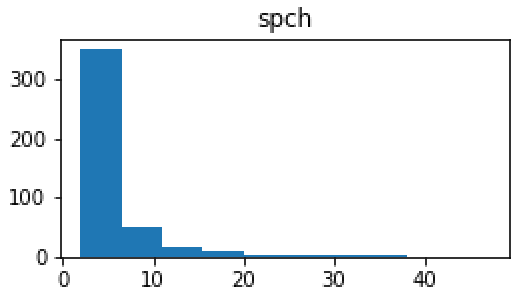


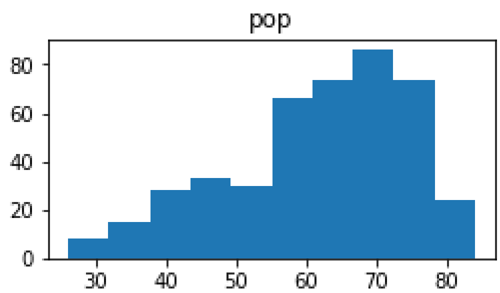

live
150
100
50
0
0
20
40
60
80

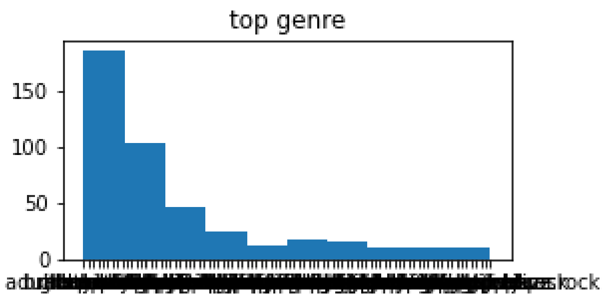

val
60
40
20
0
20
40
60
80
100


dur
100
75
50
25
0
100
200
300
400
500


acous
100
50
0
0
20
40
60
80
100


spch
300
200
100
0
0
10
20
30
40


pop
80
60
40
20
0
30
40
50
60
70
80



These histograms provide valuable visual insight into the range of vlues from each column. This was key to distinguish which columns had obvious differences in the data and which were very similar. This is all in the process of finding potential columns to drop that might negatively effect the data if it does not provide any difference in values. From these graphs it seems that possible outliers could invlude valence, energy and acoustics which all have a relatively flat shape meaning there isn't much difference between genres.

In [16]:

```
train['genre'] = train['top genre'].str.lower()
replacements = {
   'top genre': {
      r'(dance rock|rockl|country rock|alternative rock|classic rock|british invasio
      r'(deep adult standards|adult standards)': 'Adult standards',
      r'(r&b|classic soul|chicago soul|british soul)': 'RnBorSoul',
      r'(yodeling|louisiana blues|drone folk|doo-wop|canadian folk|british folk|brit
      r'(britpop|afropop|europop)': 'Pop',
      r'(pop|new wave pop|italian pop|hip pop|dance pop|classic uk pop|classic danis
      r'(uk garage|neo mellow|mellow gold|classic girl group|chanson|bubble trance|b
      r'(hip hop|hi-nrg|g funk|east coast hip hop|disco house|disco|detroit hip hop|
      r'(german dance|eurodance|bubblegum dance|british dance band|big room|belgian

            }

                }

train.replace(replacements, regex=True, inplace=True)
train.genre
```

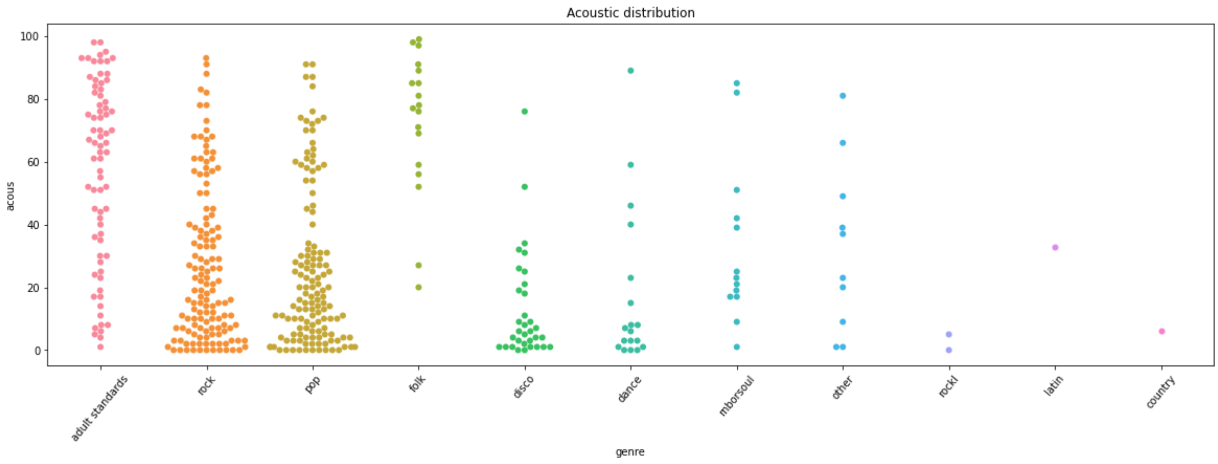
Out[16]:

```
0      adult standards
2      adult standards
3      adult standards
4                 rock
5                  pop
            ...
448    adult standards
449                pop
450                pop
451                pop
452               rock
Name: genre, Length: 438, dtype: object
```

Condensing genres

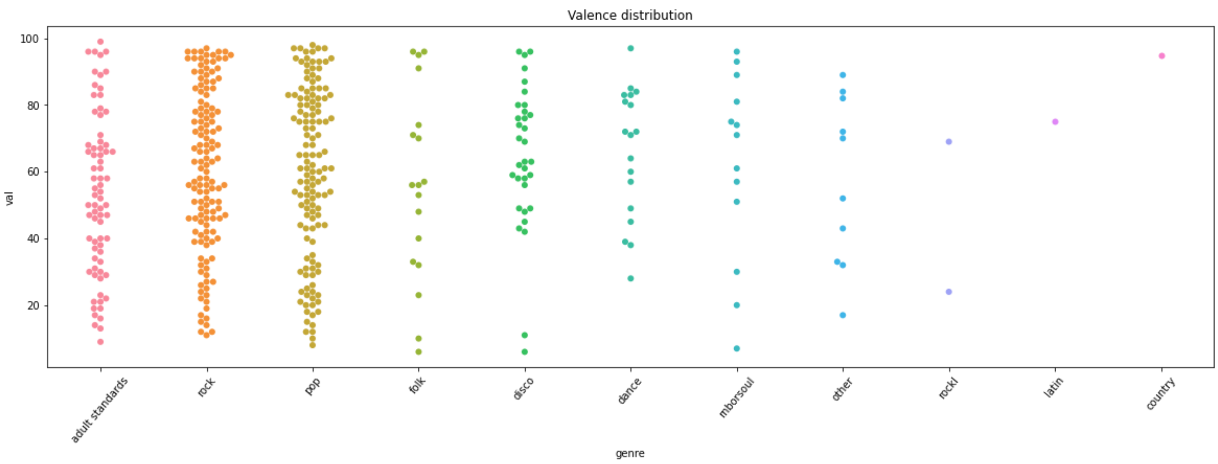
For easier exploratory analysis into the genres, they were condensed into 10 broader genres that fit best with each individual one. This gave us a great visualization of where each genre had a bigger desnity of points for each column. Originally this was used to predict a final model with scores of up to 0.6, however as the test set scoring was based upon the exact genres this was later removed. This would be the approach we would go for in an individual project as it would most likely provide better results and enough known genres for good analysis.

In [24]:
```
plt.figure(figsize = (20,6))
plot = sns.swarmplot(x='genre', y='acous', data=train, size=6)
plt.setp(plot.get_xticklabels(), rotation=50)
plt.title('Acoustic distribution')
plt.show()
```



We further explored the histogram plots for more information. The above plot shows the distribution of acoustics over the new 10 condensed genres. There is a large density of points in the low range of rock and pop, showing that a large majority of both genres don't use acoustics. This can also be seen for other categories such as disco and dance which make a lot of sense. While it could be a problem having a large amount of genres having similar distributions, we believe it is worth it to keep this column as categories such as Adult Standards and RnBorSoul have a large difference.

In [25]:
```
plt.figure(figsize = (20,6))
plot = sns.swarmplot(x='genre', y='val', data=train, size=6)
plt.setp(plot.get_xticklabels(), rotation=50)
plt.title('Valence distribution')
plt.show()
```



In this plot of the valence we can see that most genres have a very spread out distribution of valence. Due to all the genres having mostly equal densities thoughout valence values, there is potential to drop this column as it could be not very usefull at classifying genres.

In [602…
```
genre = (train["top genre"].str.strip()).str.lower()
genre
```

Out[602…
```
0          adult standards
2          adult standards
```

```
3          adult standards
4                glam rock
5                      pop
               ...
448        adult standards
449     brill building pop
450              dance pop
451               boy band
452             album rock
Name: top genre, Length: 438, dtype: object
```

3. Data Processing

Here we decided how to process the data to fit the models we wanted to try out. Due to the large amount of time used fitting models with the condensed 10 categories, we opted for simpler processing here due to time constraints. This code simply removes categories that have under 2 songs in each as these are very unlikely to be predicted with such a short sample size. While better models and accuracy could be achieved with less genres, we believed that removing too many would not be optimal in real life situation where you could not compare with actual results.

After many model attempts using this new method of simply removing the unwanted genres, we found that keeping all the numerical data columns to be optimal. While there was varying results in the models with different amounts of data used, our optimal model of random forest classfier gave us the best score using all the column data. From exploratory analysis we found there to be possible data points to remove, these possible negative impacts were not strong enough to be considered to be removed.

In [34]:
```
counts = train['top genre'].value_counts()

res = train[~train['top genre'].isin(counts[counts < 3].index)]
res
```

Out[34]:

	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	pop	top genre
0	1996	107	31	45	-8	13	28	150	75	3	44	adult standards
2	1979	105	36	63	-9	13	67	245	11	3	77	adult standards
3	1980	170	28	47	-16	13	33	232	25	3	67	adult standards
4	1973	121	47	56	-8	15	40	193	45	3	63	glam rock
5	2010	110	56	71	-7	12	23	223	15	6	74	pop
...	...	...	...	...	...	...	...	...	...	...	...	...
448	1959	80	22	18	-17	10	16	214	92	4	45	adult standards
449	2010	148	81	53	-13	23	96	147	50	3	50	brill building pop
450	2002	168	55	73	-8	20	61	289	23	14	77	dance pop
451	2000	165	87	64	-5	6	88	191	5	8	62	boy band
452	2002	105	73	68	-8	14	94	281	11	2	59	album rock

358 rows × 12 columns

In [35]:
```
res["top genre"].value_counts()
```

Out[35]:
```
adult standards         68
album rock              66
dance pop               61
glam rock               16
brill building pop      16
europop                 14
dance rock              13
boy band                10
british invasion         8
disco                    7
bubblegum dance          7
art rock                 7
barbadian pop            6
deep adult standards     6
atl hip hop              6
eurodance                6
soft rock                5
british soul             5
classic soul             5
pop                      5
doo-wop                  4
east coast hip hop       4
classic uk pop           4
disco house              3
new wave pop             3
g funk                   3
Name: top genre, dtype: int64
```

4. Predictor creation

Here we setup the data to begin building a prediction model by creating a train and test set from our original training data on Kaggle. We eventually decided on a 80/20 split in train and test data due to the positive influence a larger train set had on the random forest classifier. The train sets were created by taking the values of all columns except genre for X_train, and then the Y_train set only involved the genres. This was the same for the test sets. To categorize the genres in the sets, they were given the unique values of each genre.

Using label_binarise was used in a one-vs-all fashion to extend these algorithms to a multi-class classification case. This function makes it possible to compute this transformation for a set of class labels already known. From here the genre column is read as numerical so as to the model can create numeric predictions. This will later be tranformed back into string format for actual predictions.

In [47]:
```
train_set, test_set = train_test_split(res, test_size = 0.2, random_state = 0)

X_train = train_set.values[:,:-1]
Y_train = train_set.values[:,-1]


X_test = test_set.values[:,:-1]
Y_test = test_set.values[:,-1]
```

In [48]:
```
unique = np.unique(Y_train)
unique
```

Out[48]:
```
array(['adult standards', 'album rock', 'art rock', 'atl hip hop',
       'barbadian pop', 'boy band', 'brill building pop',
       'british invasion', 'british soul', 'bubblegum dance',
```

```
       'classic soul', 'classic uk pop', 'dance pop', 'dance rock',
       'deep adult standards', 'disco', 'disco house', 'doo-wop',
       'east coast hip hop', 'eurodance', 'europop', 'g funk',
       'glam rock', 'new wave pop', 'pop', 'soft rock'], dtype=object)
```

In [49]:

```
from sklearn.preprocessing import label_binarize
from sklearn.preprocessing import LabelEncoder


y_train_1hot = label_binarize(Y_train, classes = unique)
y_test_1hot = label_binarize(Y_test, classes = unique)


y_test_label = LabelEncoder()
```

5. Classification Model selection

In this section we have included all the potential models that were tried and tweaked. Due to the issue regarding the condensed genre code where we did not test on Kaggle until too late we restarded the project with simpler inputs and more general models to find the best suited one in a decent time. From our initial discovery into the multipl models attempted, the random forest classfier stood out with continually having the highest accuracy scores. This also made the model very simple as not many changes had to be made apart from the number of genres. From our chosen test data sample which includes all numeric columns as well as a cap of at least 3 songs to be in the genre we went ahead used random forest classifier as our strongest predictor.

Below are our individual models and scores using our final train data sample.

In [51]:

```
from sklearn import tree
dtc = tree.DecisionTreeClassifier(max_depth=50)
dtc.fit(X_train, y_train_1hot)
dtc.score(X_test, y_test_1hot)

y_pred = dtc.predict(X_test)
dtc.score(X_test, y_test_1hot)
```

Out[51]:

```
0.20833333333333334
```

The decision tree classifier starts at the root node and works down a path seperating following if-else pattern. After this process the prediction is the value of genre at the end of the if-else questions asked to the train set. This classfier was not able to keep up with the random forest classifier which makes sense as they are an ensemble of decision trees. Decision trees have a inclination to be overfitted which is most likely a factor in this model.

In [62]:

```
from sklearn.linear_model import LogisticRegression
lrc = LogisticRegression(max_iter=10000)
lrc.fit(X_train, Y_train)
lrc.score(X_test, Y_test)
```

Out[62]:

```
0.3472222222222222
```

Logistic regression finds the best fitting model to describe the relationship between the dependent and independent variable. Here it was founf that the logistic regresion model increased heavily when the data was more simple. However it did not perform as well as the random forest classfier,

In [427…

```
svc_classifier = SVC()
svc_classifier.fit(X_train, Y_train)
y_pred_svc = svc_classifier.predict(X_test)
accuracy_score(Y_test, y_pred_svc)
```

Out[427…

```
0.16666666666666666
```

The Support Vector Classifier makes a hyperplane in multidimensional space to separate different classes. This tried to find the best margin to separate classes, which best works on a alarge data set. As the genres are all very closely related in multiple categories this most likely provided diffciculties.

In [56]:

```
from sklearn.ensemble import GradientBoostingClassifier
gb_clf = GradientBoostingClassifier(n_estimators=3, learning_rate=0.5, max_depth=1,
gb_clf.fit(X_train, Y_train)
y_preds = gb_clf.predict(X_test)
print(accuracy_score(Y_test, y_preds))
```

```
0.25
```

Finally, the model we used for the Kaggle predictions was the random forest classifier. This fits a number of decision tree classifiers on various sub samples of the dataset and uses averaging to improve the prediction accuracy and reduce overfiting of decision trees. The output prediction being the class selected by the most trees. This classfier was perfect for our use as we did not have too much time to try many combinations of inout data and processing as it has a inclination to not overfit with more features. Making it a simple choice to pick when wantign a quick predictor. We did still attempt many variations to improve the model as much as possible resulting in using all numerical columns as well as any genre with over 3 entries. Hyper parameters help with ensuring more accuracy, here we included a mininum sample split and max depth, which were tuned to create better accuracy scores in our 80/20 test set.

In [57]:

```
from sklearn import ensemble
rfc = ensemble.RandomForestClassifier(n_estimators=100, min_samples_split = 6)
rfc.fit(X_train, Y_train)
rfc.score(X_test, Y_test)
y_preds = rfc.predict(X_test)
print(rfc.score(X_test, Y_test))
```

```
0.4305555555555556
```

6. Predictions

The random forest classifier was used to train our input data, we then used the this model to test the data supplied on Kaggle. The code below shows the creation of the model and using it to create an array of predictions to what the model thinks the genres of songs are from the test data. Our 80/20 test score came in at 0.44 which was the highest we could produce with the model. To then test this model against the real values on Kaggle we created a new dataframe with just Id and top genre by adding it to the empty column in the test data and removing all other unwanted columns. This file was then saved and uploaded for results.

In [647…
```
model_method1 = ensemble.RandomForestClassifier(n_estimators=100, min_samples_split
print(model_method1.score(X_test, Y_test))

predictions_method1 = model_method1.predict(testa.values)
predictions_method1
```

```
0.4444444444444444
```

Out[647…
```
array(['dance pop', 'dance pop', 'adult standards', 'dance pop',
       'adult standards', 'album rock', 'adult standards',
       'brill building pop', 'dance pop', 'album rock', 'adult standards',
       'dance pop', 'adult standards', 'album rock', 'dance pop',
       'dance pop', 'album rock', 'dance pop', 'dance pop', 'dance pop',
       'adult standards', 'adult standards', 'adult standards',
       'adult standards', 'adult standards', 'album rock', 'dance pop',
       'adult standards', 'brill building pop', 'dance pop', 'dance pop',
       'adult standards', 'album rock', 'dance pop', 'album rock',
       'dance pop', 'dance pop', 'adult standards', 'dance pop',
       'dance pop', 'adult standards', 'album rock', 'album rock',
       'adult standards', 'album rock', 'adult standards', 'album rock',
       'dance pop', 'album rock', 'album rock', 'adult standards',
       'album rock', 'album rock', 'adult standards', 'album rock',
       'adult standards', 'album rock', 'adult standards', 'dance pop',
       'adult standards', 'album rock', 'album rock', 'adult standards',
       'album rock', 'dance pop', 'dance pop', 'adult standards',
       'album rock', 'adult standards', 'dance pop', 'adult standards',
       'album rock', 'dance pop', 'adult standards', 'dance pop',
       'dance pop', 'dance pop', 'album rock', 'adult standards',
       'dance pop', 'album rock', 'adult standards', 'album rock',
       'album rock', 'dance pop', 'dance pop', 'dance pop',
       'brill building pop', 'adult standards', 'adult standards',
       'dance pop', 'album rock', 'dance pop', 'album rock', 'dance pop',
       'dance pop', 'album rock', 'album rock', 'adult standards',
       'album rock', 'dance pop', 'adult standards', 'adult standards',
       'adult standards', 'album rock', 'album rock', 'album rock',
       'adult standards', 'dance pop', 'dance pop', 'dance pop',
       'album rock', 'album rock'], dtype=object)
```

In [648…
```
testa1 = pd.read_csv('CS98XClassificationTest.csv')
testa1.drop(columns = ['title', 'bpm', 'artist', 'year','nrgy','dnce','dB','live','v
testa1['top genre'] = predictions_method1
testa1
```

Out[648…

	Id	top genre
0	454	dance pop
1	455	dance pop
2	456	adult standards
3	457	dance pop
4	458	adult standards
...	...	...
108	563	dance pop
109	564	dance pop
110	565	dance pop
111	566	album rock
112	567	album rock

113 rows × 2 columns

In [649…

```
testa1.to_csv('test17.csv', index=False)
```

The configuration of input data into our classifier resulted in a score of 0.32142 in the Kaggle testing. On the leaderboard this resulted in a tied position with a few other groups at 44th. This was actually quite expected as with the limited time it had to be quite a simple model. This is also something which we could see other people using as well explaining the many groups with the exact same score. This score is quite a bit lower than our own test score at 0.44. A big factor that could contribute to this is the fact that our training data could have somewhat genre similarities compared to a completely different dataset in the kaggle test data. When we split our data into train and test there is a chance that the genres are split quite evenly.

While this is not the worst model, we believe our original idea that grouped up the huge range of categories into 10 large ones would be a great way to analyse the data. Grouping 8 types of rock music into one category makes a lot of sense. However we failed to think about how the kaggle predictor would work while focussing on our own test accuracy scores. I believe if we had access to the test data genres and used a similar grouping system, it could be a very efficient way to create a predictor.