ITMAL Øvelser - Uge 5

Øvelse 1

For this subexercise the group is to perform data analysis of the given dataset, namely "california housing prices". The subexercise consists of 5 parts which will be described herafter in chronological order. Before starting, the group needs to get the data read in the python workspace, to which is has been decided to store it as a pandas dataframe. This is done as pandas is build on numpy, and as such, has the same features as numpy with more.

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm
import pandas as pd
import math

# Load data - vægt data (kvinder/mænd)
data = pd.read_csv('.../Uge5_files/housing.csv', sep=',',header=0)
```

a)

For the first part we are to plot the distribution of the median_income data for the districts. This is done using matplotlibs *hist()* function. Futhermore, mean, variance, std. deviatien and median can all be found using numpy's build in methods.

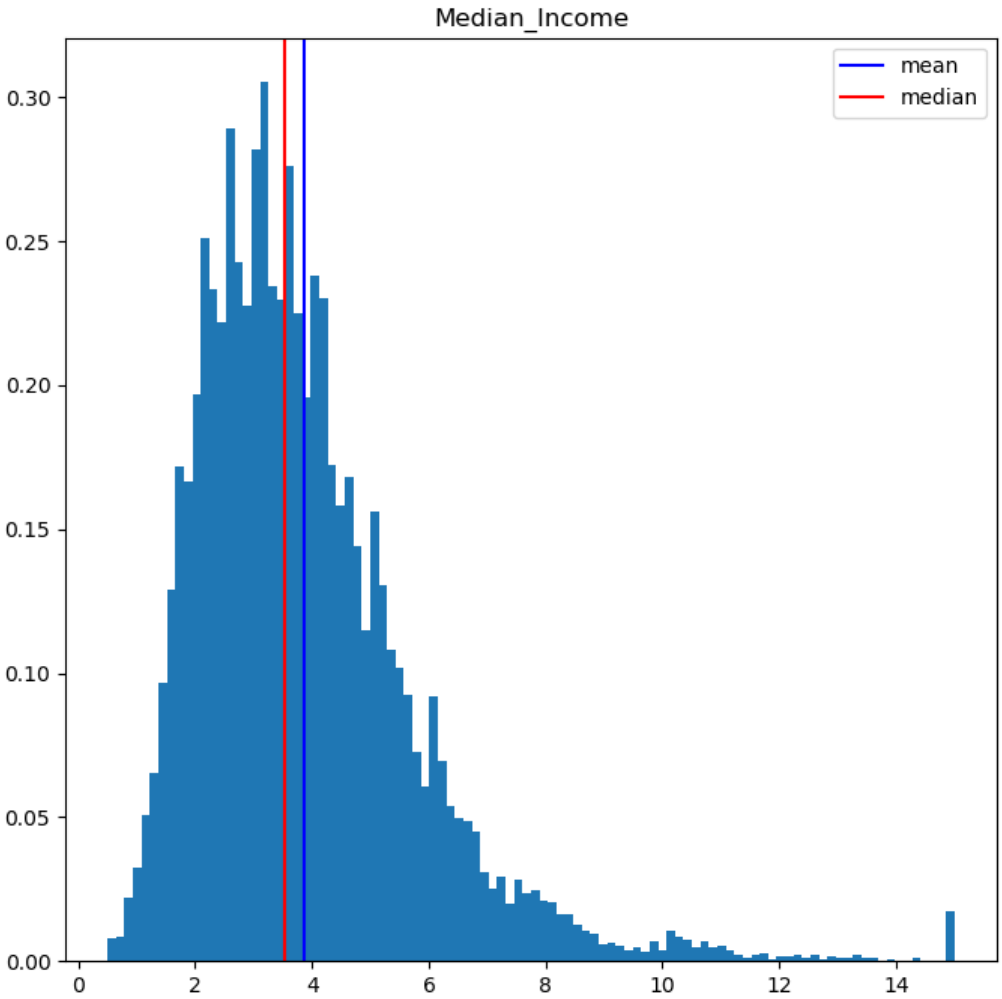
```
median_income = data['median_income'] # Extract column of df

mu = np.mean(median_income)         # Mean
sigma = np.std(median_income)       # Var
sigma2 = np.var(median_income)      # std deviatien
median = np.median(median_income)   # Median
print(f"Mean: {mu}, Varians {sigma2}, Std Deviatien {sigma}, Median {median}")
```

Yields:

```
Mean: 3.8706710029070246, Varians 3.60914768969746, Std Deviatien
1.899775694574878, Median 3.5347999999999997
```

With the plot using matplotlib:

```
fig, ax = plt.subplots(1, 1, figsize=[8,8])   # Create the plot object
ax.hist(median_income,bins=100, density=True) # Normalises the histogram
```



Median income distribution from california housing prices

From the above it is seen that the histogram somewhat resembles a normal distribution, although it must be noted that is rather tail heavy towards the higher median income values. Futhermore, it is noted that there is a rather large amount of districts towards the right with a extremely high median income - this and the heavy tail could explain the skewedness in the mean value compared to the median.

b)

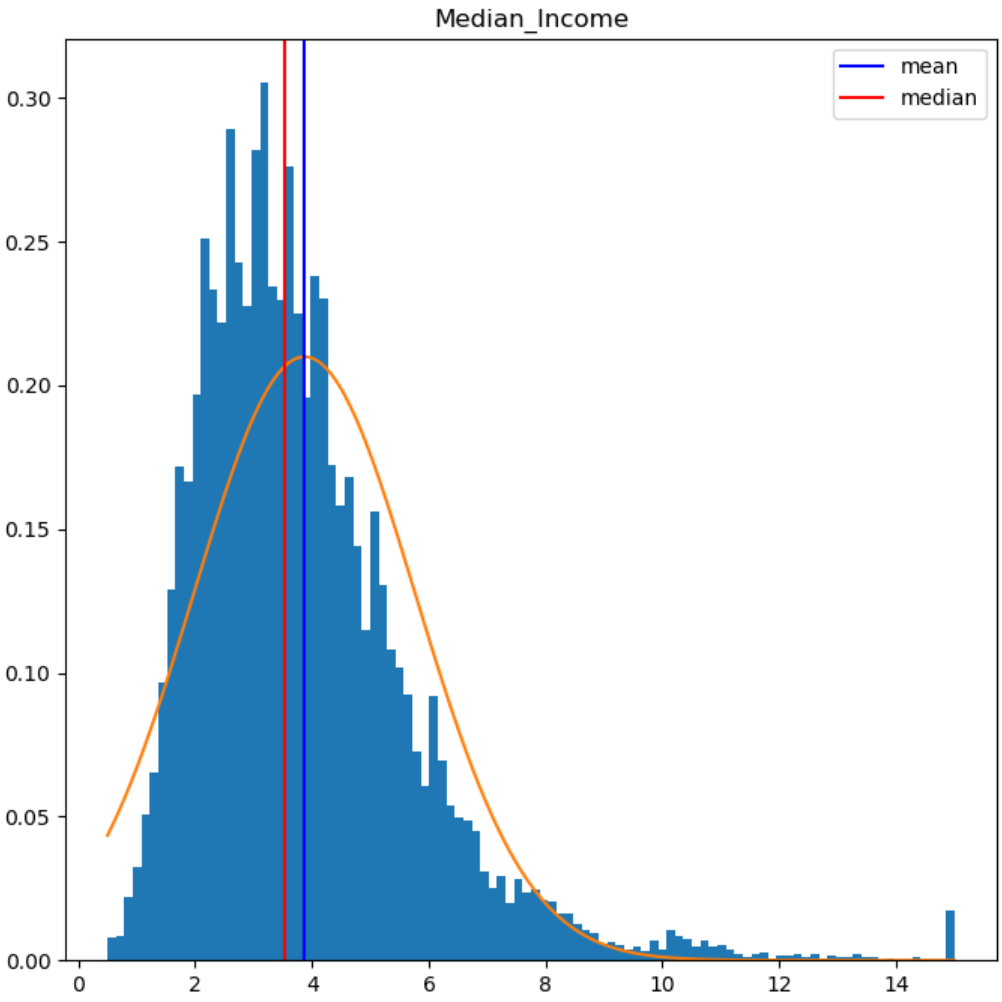
Now, looking at the mean and medianb, it is seen these are not identical. This is easily explained if "median_income" is seen as a column vector, where the mean is exactly the sum of all it's values, divided by it's length. The median however, is the value of the center index in the vector. So, which is the most telling? In the case of median income, the mean can be quite biased, as this is more easily influenced by outliers - for example the mean income of the population of a country might be heavily influenced by the top 1%, and therefore not very telling of the average citizen. The same goes for our example.

c)

For this part, a true normal distribution is fitted on top of the histogram - since we already normalised the histogram, this should be directly comparable:

```
myMax = np.max(median_income)
myMin = np.min(median_income)
xarr = np.linspace(myMax, myMin, 500)
ax.plot(xarr, norm.pdf(xarr, mu, sigma)) # Plot
```

Which yields the plot:



Median income distribution from california housin prices, now with a true normal distribution on top

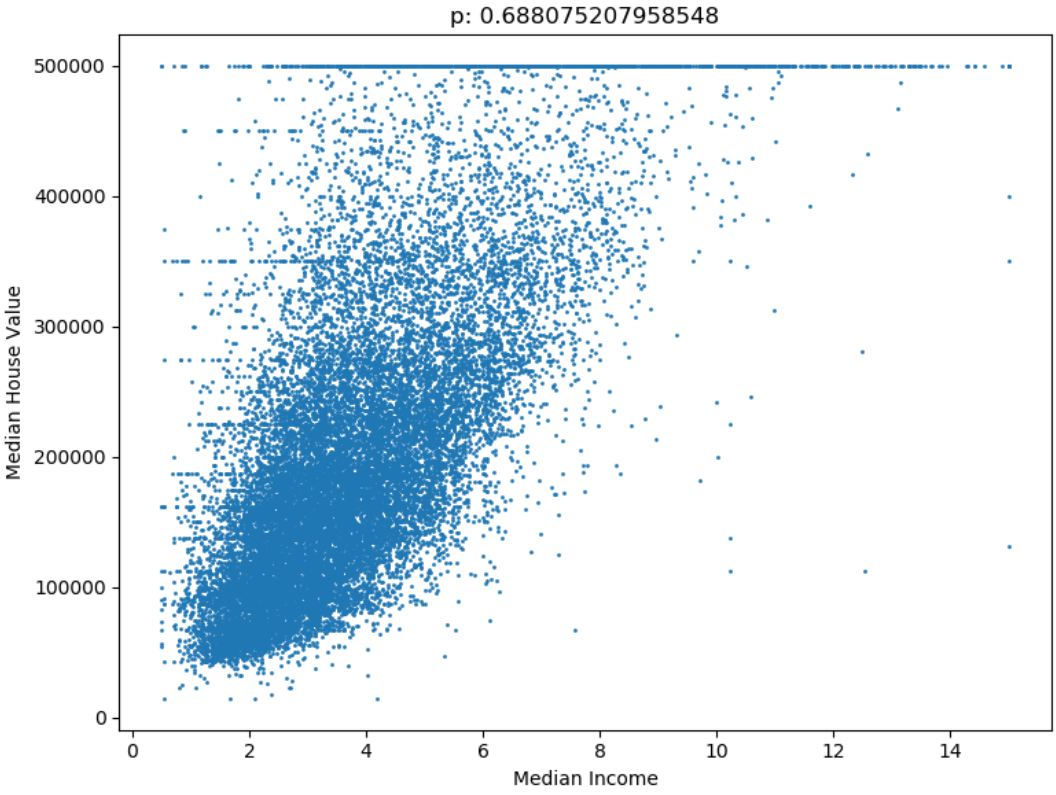
From which it is seen that our distribution does not follow that of a normal distribution, as it is too tail heavy, and as such, has shifted it's mean too far to the right. Furthermore, it would seem that our variance is also affected by the tail heavyness, as the true gaussian model does not encapsulate(reach) our top points around the median or mean.

d)

In order to determine if there is a correlation between median house value, and median income, this is computed and plotted:

```
# Check if there is a corelation between median income and median house value
medIncAndHVal = data[['median_income','median_house_value']]
corrcoef = np.corrcoef(medIncAndHVal.T) # obs: rækker=variable, kolonner=samples
(modsat normalt..)

plt.scatter(medIncAndHVal['median_income'], medIncAndHVal['median_house_value'],
s=1)

plt.title(f"p: {corrcoef[1,0]}")
plt.xlabel('Median Income')
plt.ylabel('Median House Value')
```

Which yields:



Correlation plot of median house value and income

From the above, a trend can be seen along the axis' as they seem to be increasing together(a linear upward trend, or positive correlation). The correlation coefficient(title of plot) verifies this. At the same time, some vague horisontal lines can be seen. These seem to limitations in the house value, which must stem from an unknown bias(or prerendition) of the data we are processing. The upmost horizontol line of the data plots seem to stem from an upper limit on the measurement collecting the data.

e)

For this part, the 5% and 95% percentile is to be found. This is done using numpy's build in functions like before:

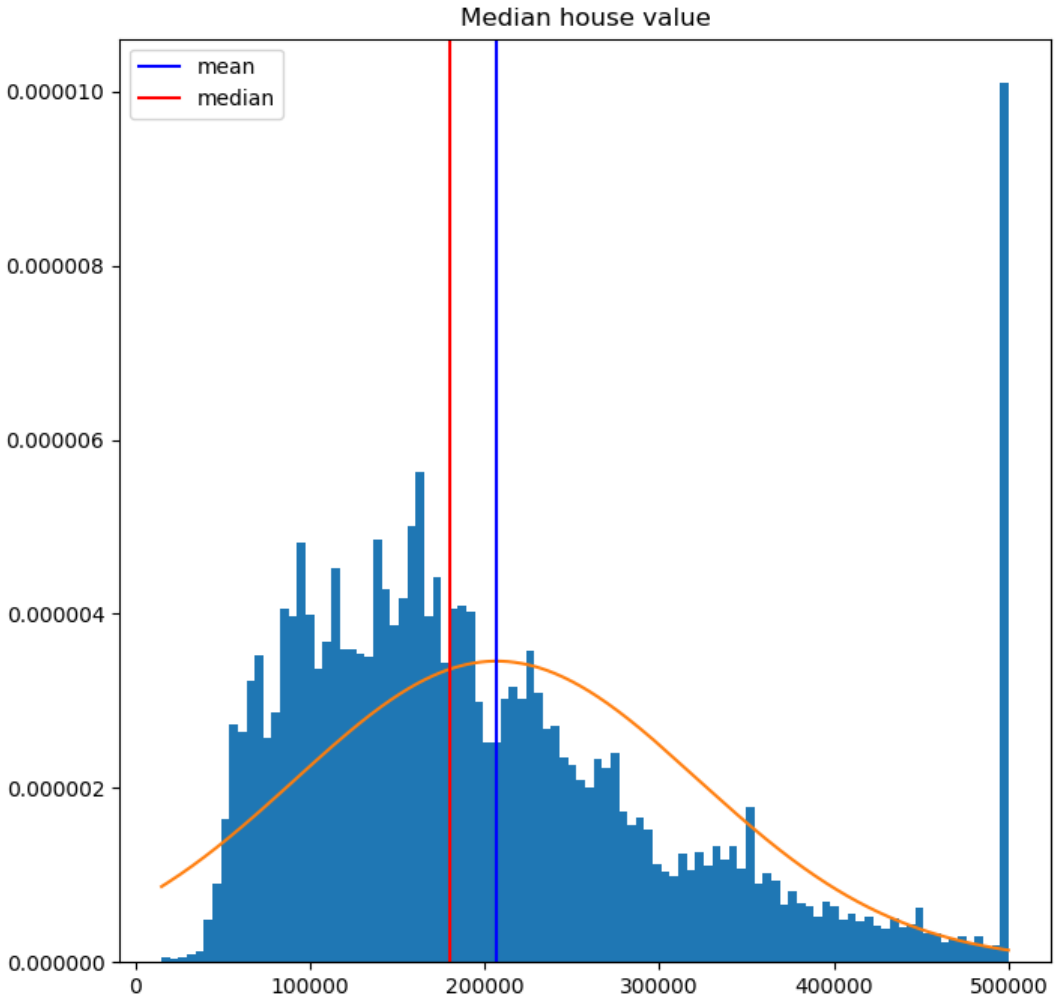
```
print("Fifth Percentile of med H val: ", np.percentile(data['median_house_value'],
5))

print("Ninety-fith Percentile of H val: ",
np.percentile(data['median_house_value'], 95))

print("With max: ", np.max(data['median_house_value']), " and Min: ",
np.min(data['median_house_value']))
```

Which yields:

```
Fifth Percentile of med H val:  66200.0
Ninety-fith Percentile of H val:  489809.9999999998
With max:  500001.0  and Min:  14999.0
```

Telling us that 5% of the districts has a median income of 66200 or less, and 95% has a median income of 489809 or less.

Using the same code as before, but with another feature from the dataset, median house value, a distribution can be found:



Median house value histogram

From the above, an obvious fault can be seen(the reason for the horizontol line before) at a median house value of 500000. Like said, this must stem from the data not being able to measure past this value, and as such, this value cannot be seen as representative, as these values can range from 500000 to ∞. A solution to this could be to remove all districts with this value, or simply assign them the median instead. Looking at the representation, e.g the numbers of samples with this value, the first seems the best, as the other would assign an artificial bias to the set. Also, disregarding the top value, the distribution does not seem to follow that of a gausian one, as this is heavily tail heavy towards the upper values - it seems more to resemble that of a Reyleigh distribution if one were to guess.

Øvelse 2

The group has chosen a dataset based on data collected from spotifys API, containing information about songs on the platform. Each song is described by 16 features:

- Genre
- Artist name
- Track name
- Track ID

- Popularity

The remaining 11 features are 'audio features' which describe different aspects of the tracks ***sound*** and ***feel***. These are

- Acousticness
- Danceability
- Duration in ms
- Energy
- Instrumentalness
- Key
- Liveness
- Loudness
- Mode
- Speechiness
- Tempo
- Time signature
- Valence

The group wishes to investigate the relationship, correlations, etc, between these audiofeatures, genre and popularity.

Importing the data

The data is loaded from the csv file into a pandas dataframe

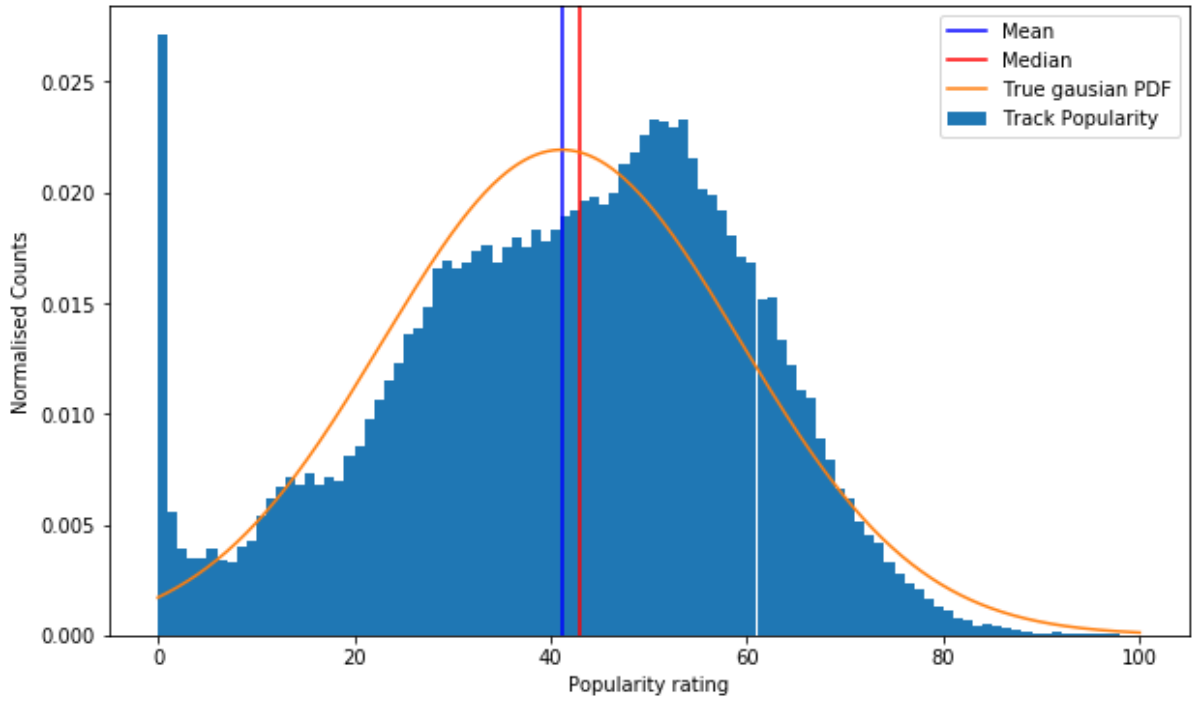
```
data_csv_path = "path/to/SpotifyFeatures.csv"
spotifyDBData = pd.read_csv(data_csv_path, sep=',', header=0)
```

Now we can investigate the distibution of the different features in the dataset

```
# Lets look at popularity and see if it follows a normal distribution
trackPopularity = spotifyDBData['popularity']

# Find the statistical properties of the popularity
mu = np.mean(trackPopularity)
sigma = np.std(trackPopularity)
sigma2 = np.var(trackPopularity)
median = np.median(trackPopularity)

# Create an object to plot into
fig, ax = plt.subplots(1, 1, figsize=[10,6])

# Plot the popularity data to get an idea of it's distribution
ax.hist(trackPopularity,bins=100, density=True, label='Track Popularity') #
Histogram is normalized
plt.xlabel('Popularity rating')
plt.ylabel('Normalised Counts')
```

```
ax.axvline(mu, color='b', label = "Mean")
ax.axvline(median, color='r', label="Median")

# Try and fit a gausian distribution
xarr = np.linspace(np.max(trackPopularity), np.min(trackPopularity), 500)
ax.plot(xarr, norm.pdf(xarr, mu, sigma), label='True gausian PDF')
ax.legend()
```

Running this results in the following output:



Popularity histogram

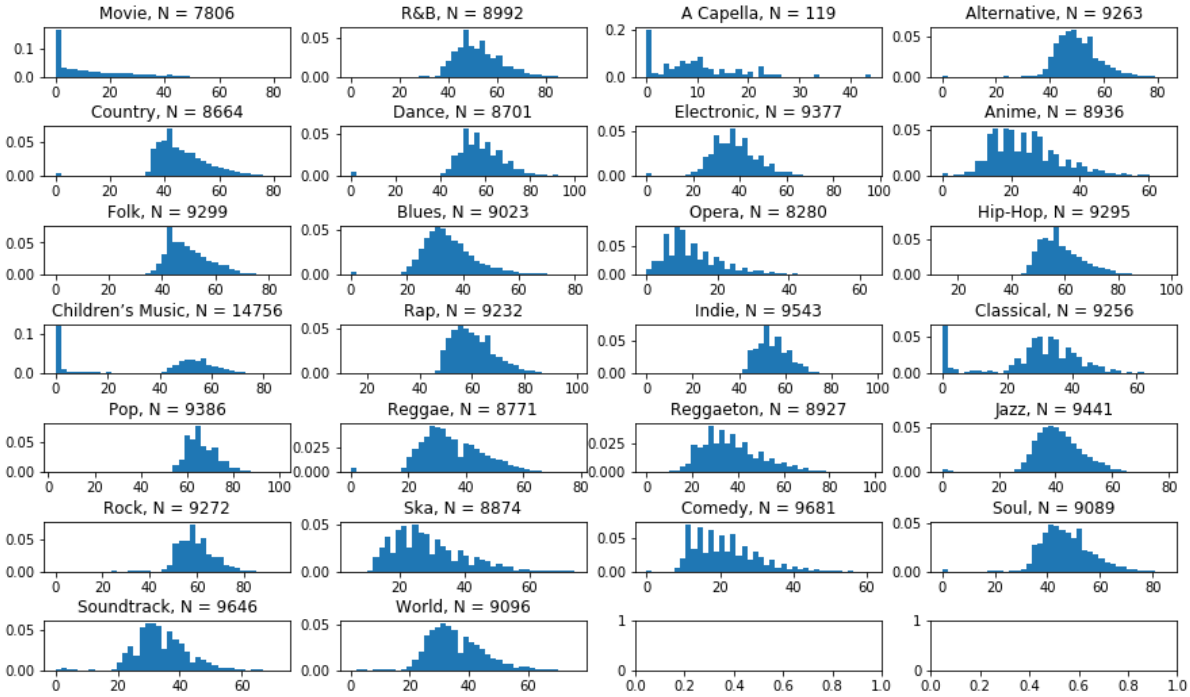
It can be seen that the distribution resembles a normal distribution, but deviates significantly in some places. The most notable difference being the big spike in the first bin, which could have several explanations. It might be that there are simply just a lot of unpopular songs on the plaform, as the data would seem to indicate. Another explanation could be that the popularity score is not calculated by spotify before some criteria is met, e.g. a track has been on the planform for period of time. It depends on how and when the score is computed, when the tracks were put on the platform, all information that the group does not currently have.

Popularity distribution by Genre

```
Genres = spotifyDBData['genre'] # Series containing genres of each track
UniqueGenres = Genres.unique()  # Contains the name of each genre included in DB

cols = 4   # How many subplots pr row
width = 15 # Width of figure
prop = 1/3 # Subplot proportions, width/height ratio of subfigures

rows = int(len(UniqueGenres)/cols)+1
height = (rows/cols)*width*prop
```

```
fig, ax = plt.subplots(rows, cols, figsize=(width,height))
plt.subplots_adjust(wspace=0.2, hspace=1)
for index, genre in enumerate(UniqueGenres):
    row, col = int(index/cols), index % cols
    genre_tracks = spotifyDBData.loc[spotifyDBData['genre'] == genre]
    popularity = genre_tracks['popularity']
    title = genre + ", N = " + str(len(popularity))
    ax[row,col].hist(popularity, bins=40, density=True, label='Track Popularity')
    ax[row,col].set_title(title)
```

This produces the following output:



Popularity histograms for each genre

Here it can be seen that the spike in the first bin is only present in a few of the genres, "A Capella", "Movie", "Children's Music" and "Classical".

Genre Pre-processing

The genre feature is a text string in the original dataset, but in order to be more useful to a machine learning algorithm, numerical values must be assigned through some form of preprocessing. As the unprocessed genre feature contains one of 18 possible textstrings with the name of the gerne, one-hot encoding seems suitable in this case. This is done with pandas function `get_dummies`:

```
# Apply One Hot Encoding to the genre feature
onehotenc = pd.get_dummies(spotifyDBData, columns=["genre"])

# Rename/format genre column (lowercase, no special symbols)
replacements = {' ':'-', '&':'n', '’':''}
onehotenc.columns = map(lambda s:
s.lower().translate(str.maketrans(replacements)), onehotenc.columns)
```

```
print(list(onehotenc.columns)) # Check column names
```

Which outputs the following

```
['artist_name', 'track_name', 'track_id', 'popularity', 'acousticness',
'danceability', 'duration_ms', 'energy', 'instrumentalness', 'key', 'liveness',
'loudness', 'mode', 'speechiness', 'tempo', 'time_signature', 'valence', 'genre_a-
capella', 'genre_alternative', 'genre_anime', 'genre_blues', 'genre_childrens-
music', 'genre_classical', 'genre_comedy', 'genre_country', 'genre_dance',
'genre_electronic', 'genre_folk', 'genre_hip-hop', 'genre_indie', 'genre_jazz',
'genre_movie', 'genre_opera', 'genre_pop', 'genre_rnb', 'genre_rap',
'genre_reggae', 'genre_reggaeton', 'genre_rock', 'genre_ska', 'genre_soul',
'genre_soundtrack', 'genre_world']
```

It is seen that the genre column has been replaced with distinct culumns, one pr possible genre. If a track belongs to a genre, it has a value of 1 in its column, and 0 in the other genres.

This structure is also useful for predicting the genre, as each value between 0 and 1 can represent an algorithm's confidence that a track belongs to a certain genre.

Scaling

Numerical features can be scaled, so that they are always between 0 and 1. The code below scales a selection of numerical features in such a way by using scikit learns `MinMaxScaler`

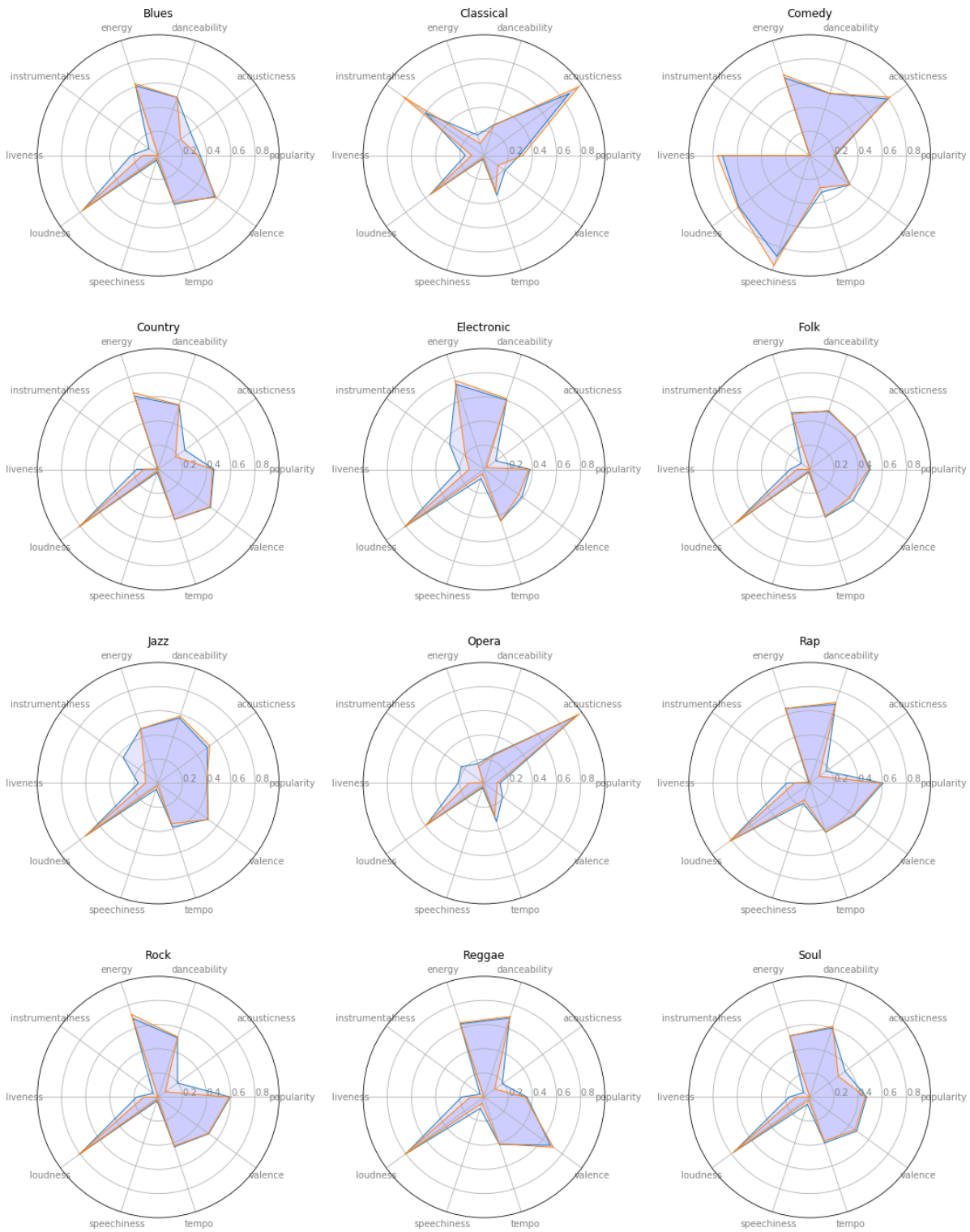
```
genres = list(filter(lambda name: "genre_" in name, onehotenc.columns))
audiofeature_cols = ['popularity', 'acousticness', 'danceability', 'energy',
                'instrumentalness', 'liveness', 'loudness', 'speechiness',
                'tempo', 'valence']

minmaxscaler = MinMaxScaler()

df_scaled = pd.DataFrame(onehotenc)
df_scaled[audiofeature_cols] = pd.DataFrame(
        minmaxscaler.fit_transform(df_scaled[audiofeature_cols]),
        index=df_scaled[audiofeature_cols].index,
        columns=df_scaled[audiofeature_cols].columns)
```

Audio Features by genre

To get an overview of how the different audiofeatures are distributed internally in each genre, a "radar plot" can be created for each genre, with the mean/average value of audiofeatures.

```
#Extract audiofeatures from a tracks of a specific genre
tracks = {genre: df_scaled.loc[df_scaled[genre] == 1]
                    .reset_index(drop=True)
                    .filter(audiofeature_cols)
            for genre in genres}

# Mean and median
afeatures_mean = {genre: tracks[genre].mean().values.flatten().tolist()
                  for genre in genres}
afeatures_median = {genre: tracks[genre].median().values.flatten().tolist()
                    for genre in genres}

def radar_subplot(categories, data, title=None, subplotpos=(1,1,1)):
    # Dimension angles
    N = len(categories)
    angles = [n / float(N) * 2 * math.pi for n in range(N)]

    # Create polar subplot
    ax = plt.subplot(subplotpos[0],subplotpos[1],subplotpos[2], polar=True)
    ax.set_rlabel_position(0)
    ax.set_title(title)
    plt.xticks(angles, categories, color='grey', size=10)
    plt.yticks([0.2,0.4,0.6,0.8], ["0.2","0.4","0.6","0.8"], color="grey",
size=10)
    plt.ylim(0,1)

    # Plot one line/shape pr item in data
    plotdata = data if isinstance(data[0],list) else [data]
    angles += angles[:1]
    for values in plotdata:
        plotvalues = values + values[:1]  # Line finishes same place as it starts
        ax.plot(angles, plotvalues, linewidth=1, linestyle='solid')
        ax.fill(angles, plotvalues, 'b', alpha=0.1)

def plot_audiofeatures(genres, cols=3, width=18, hspace=0.3):
    rows = int(len(genres)/cols)+1
    height = width*(rows/cols)
    plt.figure(figsize=(width,height))
    plt.subplots_adjust(hspace=hspace)
    for idx, genre in enumerate(genres):
        genre = genre if genre.startswith("genre_") else "genre_" + genre
        categories = list(tracks[genre])
        subplot = (rows, cols, idx+1)
        layers = [afeatures_mean[genre], afeatures_median[genre]]
        title = genre.replace("genre_","").capitalize()
        radar_subplot(categories, layers, title, subplot)

plot_audiofeatures(["blues", "classical", "comedy", "country", "electronic",
                    "folk", "jazz", "opera", "rap", "rock", "reggae", "soul"])
```

Executing the above code, results in the following output:

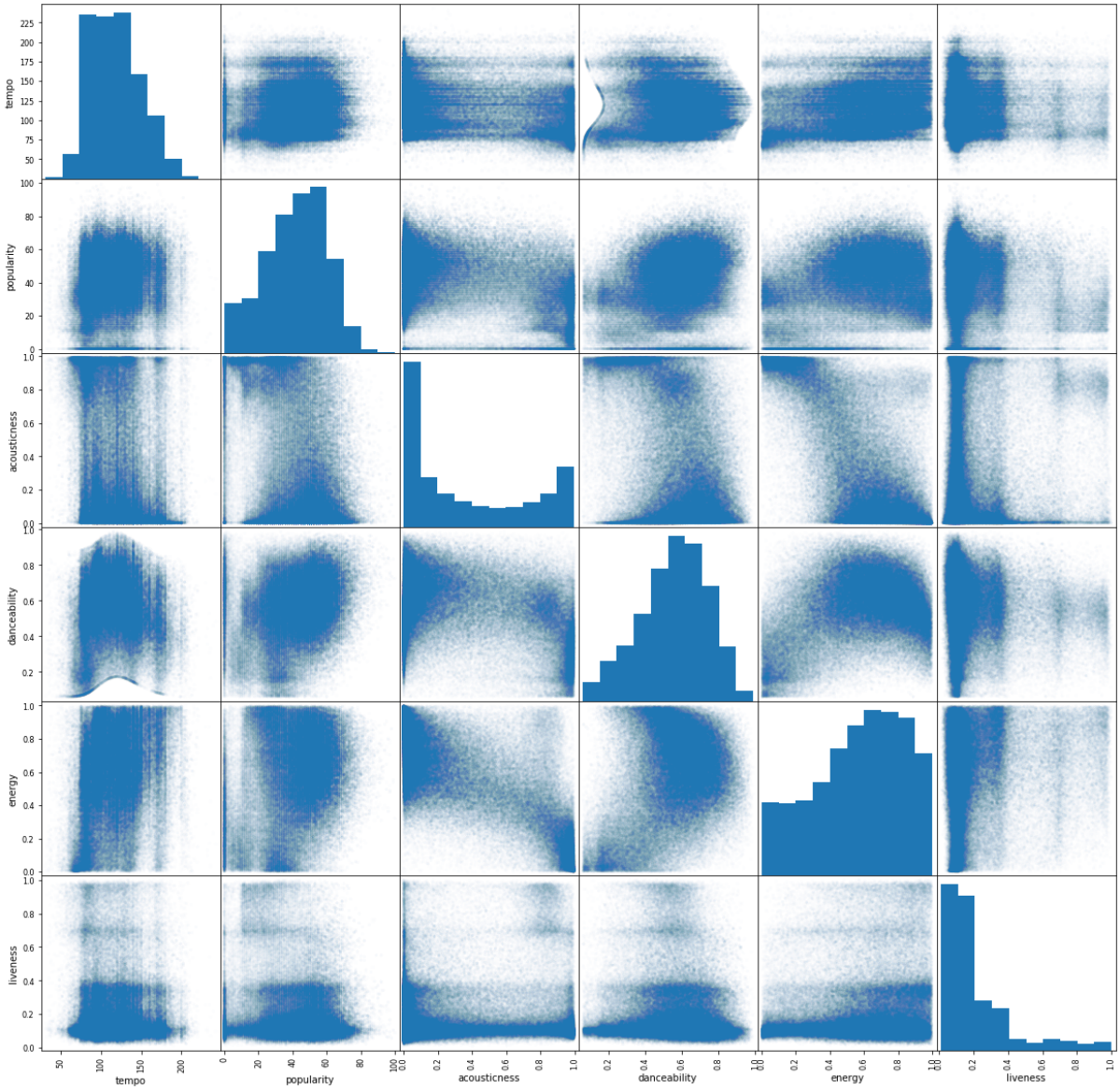


Mean and average values audiofeatures for 12 genres

Correlation between audio-features

To investigate any relation/correlation between the audiofeature, a scatter matrix can be made

```
attributes = ["tempo", "popularity", "acousticness", "danceability", "energy"]
axs = scatter_matrix(onehotenc[attributes], figsize=(20,20), alpha=0.01)
```



Scatter matrix for 6 audiofeatures

No single feature seems to be good for directly predicing another with any accuracy. Some slight correlation can be observed, for instance what looks like a negative correlation between "acousticness" and "energy".

But especially for the audiofeatures, it is perhaps not surprising that the audiofeatures aren't strongly correlated, as one could imagine that they were engineered in the first place to describe different aspects of a song.