THE RUSSIAN GOVERNMENT
FEDERAL STATE AUTONOMUS EDUCATIONAL INSTITUTION
FOR HIGHER PROFESSIONAL EDUCATION
NATIONAL RESEARCH UNIVERSITY
"HIGHER SCHOOL OF ECONOMICS"

# The Analysis of On-screen Movie Kills

Homework Project 2018/2019

**The team:**
Boris Tseitlin
Konstantin Romashchenko
Yulia Gurova

**MSc Program Data Science**
$1^{st}$ year
Faculty of Computer Science

Moscow 2018

# Table Of Contents

# 1 THE CHOICE OF THE DATASET

The dataset that is used for the project contains information about on-screen deaths in movies. There are 545 movies (more then 100 objects) and 8 characteristics including names, so the dataset meets the requirements.

The sourse for the data is the thematic web-site moviebodycounts.com. This dataset was processed and published on figshare.com. It was gathered in accordance with the rules, which are published on the web-site. We took several characteristics for the consideration: the release year of the film, MPAA rating (Motion Picture Association of America film rating system), genre or genres, the name of the director, the lenth of the film in minutes, IMDB rating based on user ratings. The main feature that we consider is the number of on-screen deaths in the movie.

The analysis of the data may reveal how the ratings depend on the number of deaths, how this number relates to the release year and so on. Moreover, genre and length of the film combined with on-screen violence may provide information on age ratings. This is a good set for the classification and clustering problems, as the films are grouped by genres and MPAA ratings. Movies in these groups are similar inside the groups, but dissimilar between them.

This analysis may be the first step to the automation of age rating systems. Moreover it may be helpful in the development of recommendation systems. And, as watching movies is the common interest of our team, the work with the dataset will inspire us to further conquest in Data Analysis.

# 2 K-means clustering

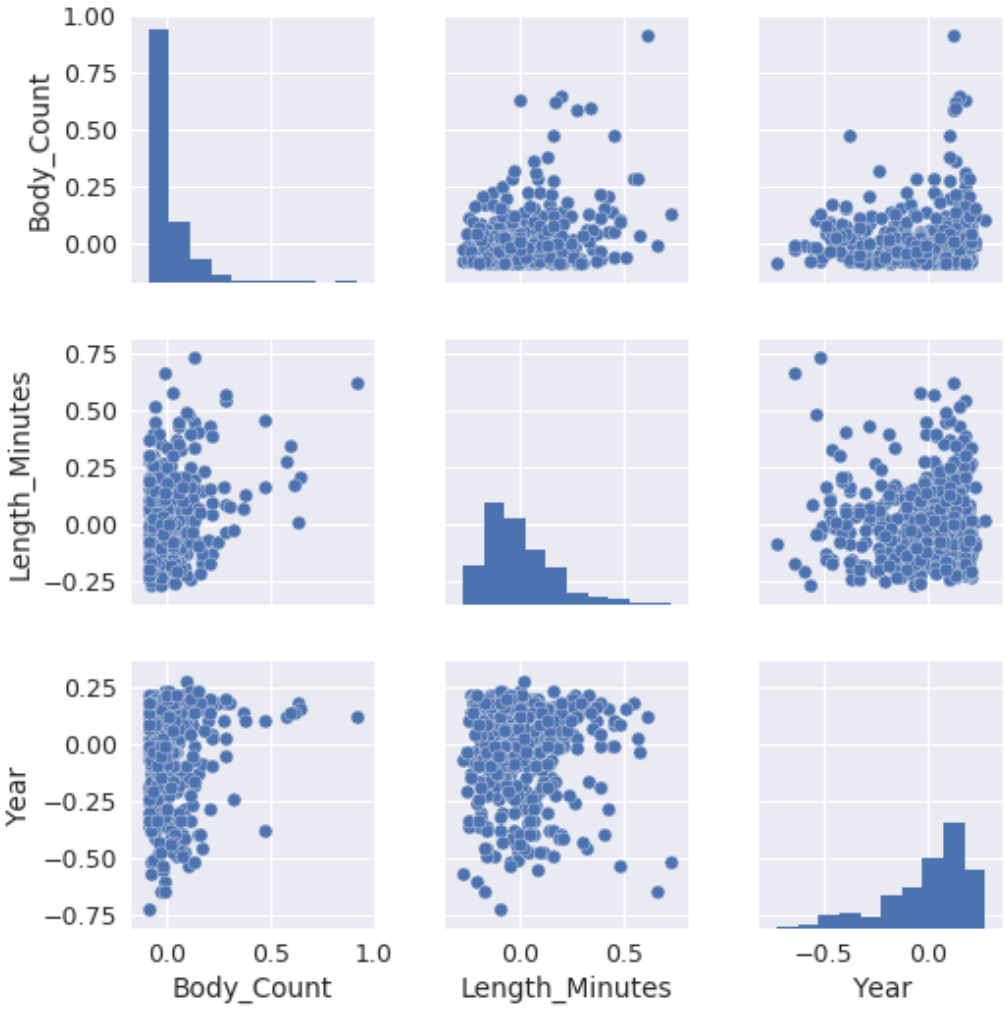
For this part of the task we choose three quantitative features: *Body_ Count*, the number of on-screen deaths in the movie; *Length_ Minutes*, the length in minutes; *Year*, the year of the release. These are quantitive features that, by our hypotheses, may be useful in cluster analysis of the dataset.

## 2.1 Preprocessing

First we standardize the dataset. We center by mean and normalize by range:

```
In [11]:  def normalize(vec):
    return (vec - vec.mean())/(vec.max() - vec.min())
```

Before applying clustering methods, we visualize data on all possible pairplots. There are no obvious clusters in the two-dimensional visualization.



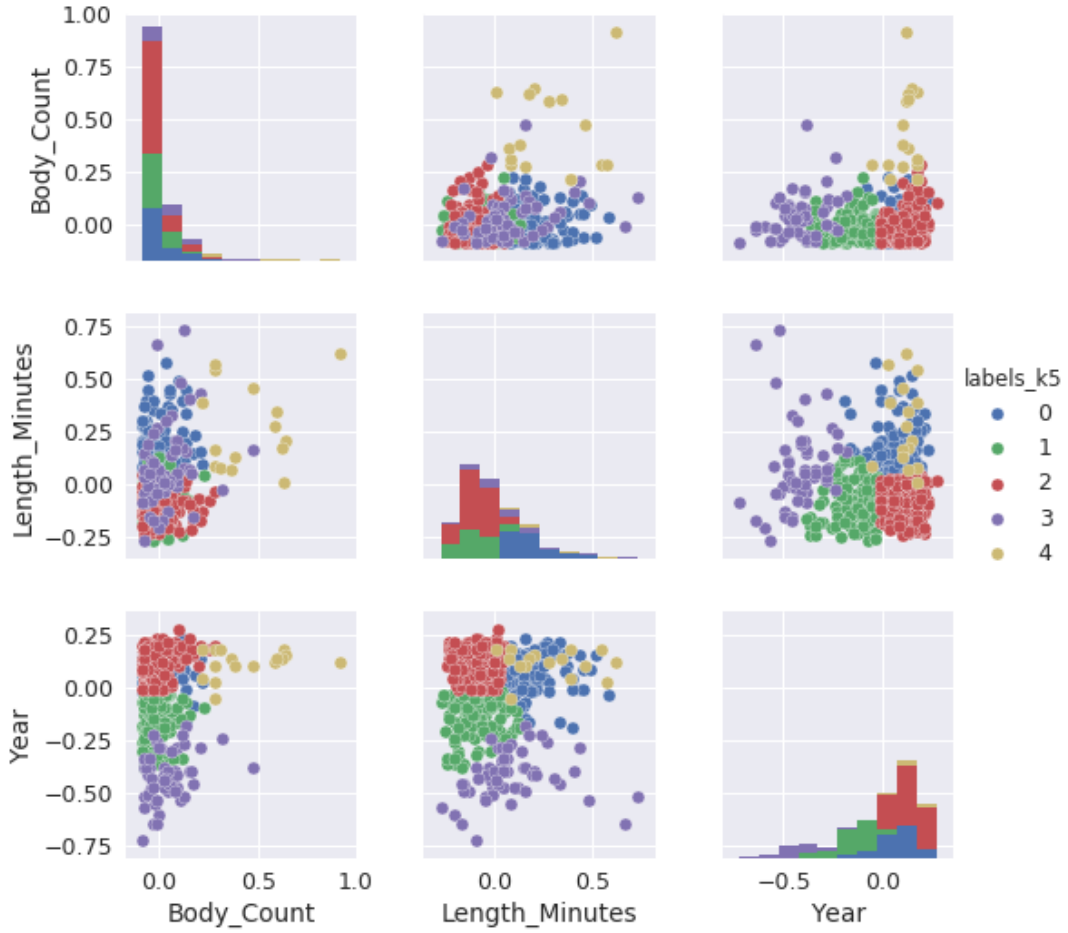
Clusters might follow a categorical feature already present in data. A categorical attribute with the least amount of variants we have is the MPAA rating. We color the plots with accordance to the ratings to see how they divide our data.


## 2.2 K-means application

The first method we apply to find the clusters is k-means at k=5. We take random initializations of 5 cluster centers and choose the best by k-means criteria from 10 initializations. The sum of squared distances from points to cluster centers, the K-means criterion: 13.613.

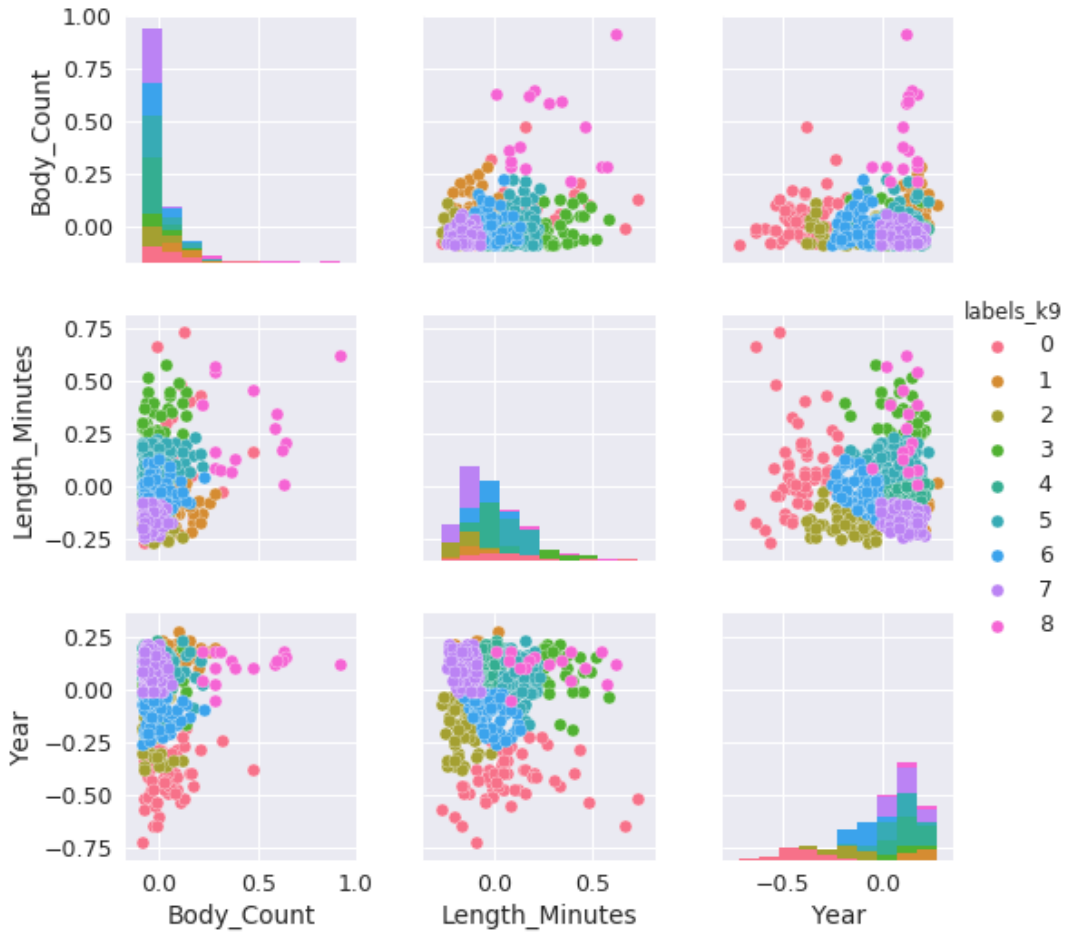
```
In [16]:
kmeans_k5 = KMeans(n_clusters=5, init='random', n_init=10, random_state=RANDOM_SEED)
kmeans_k5.fit(task_df)
task_df['labels_k5'] = pd.Series(kmeans_k5.predict(task_df))
print('Sum of squared distances from points to cluster centers, k=5:', kmeans_k5.inertia_)
sns.pairplot(task_df, hue='labels_k5', vars=quant_features)
```

We visualize the obtained clusters using pairplots. They don't match to MPAA_Rating directly, but divide the data into reasonable descriptive categories.




We would specially distinguish cluster 3 (purple colored), the old films, mostly not long and with small amount of deaths. Cluster 4 (yellow colored) is not large and contains movies with the highest number of deaths, they differ in length, and are relatively recent ones.

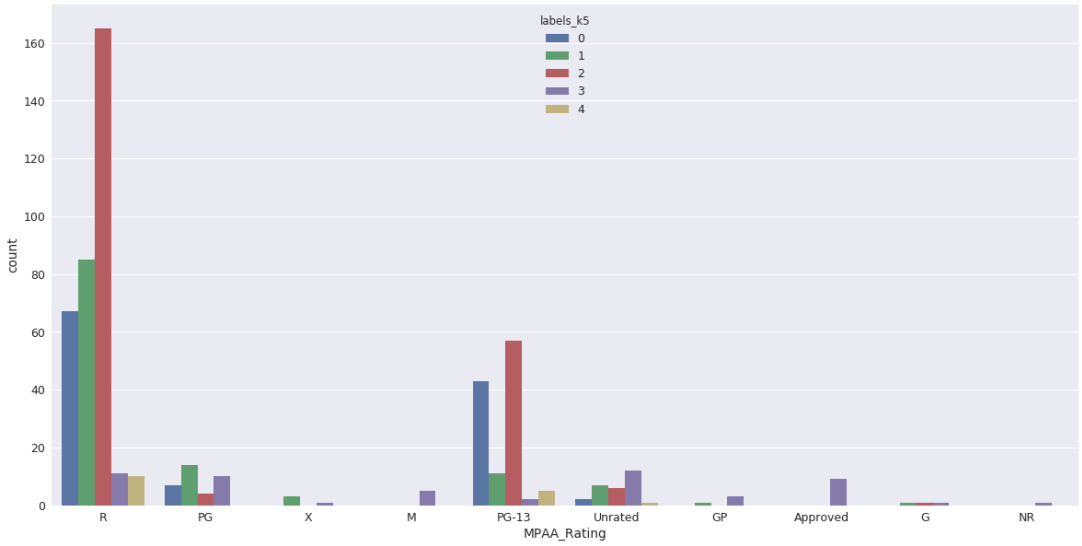
Now we apply K-means at k=9 and get a lot of smaller clusters. Sum of squared distances from points to cluster centers, k=9: 10.079.




Some clusters are obviously inherited from the pevious ones: new cluster 8 follows cluster 3 with the higest body count, the new cluster 0 follows the discussed cluster 3 (with old films) from the previous plot. Besides these two there are many small clusters at the main body of the films, overlapping on most pairplots, without potent interpretation. We conclude that 9 clusters are too many for this dataset, and 5 clusters describe the data better: the cluster boundaries are clearer and clusters are reasonably interpretable.

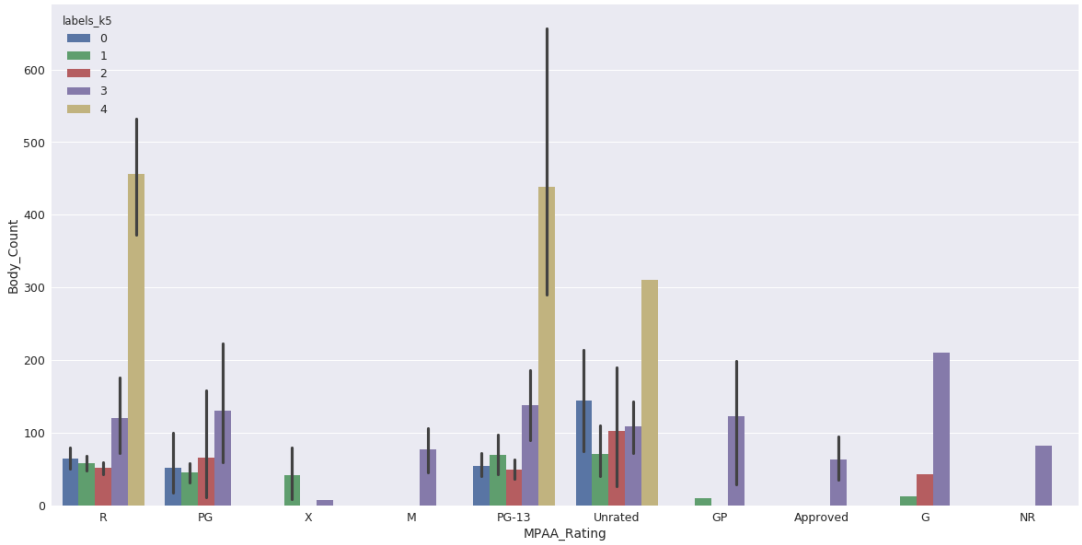
## 2.3 Interpretation

For the rest of this task we consider the partition with k=5. First we explore how movies are destributed among clusters, minding their MPAA ratings (the correspondence of the ratings is in application 1).



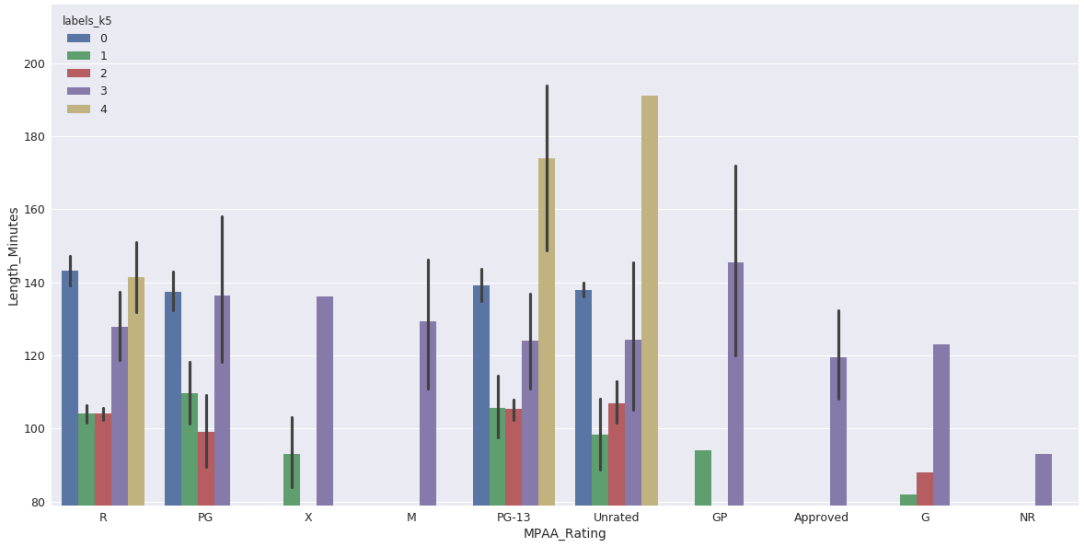

The plot shows the amount of movies of each rating in each cluster. For example we see that 165 R movies are in cluster 2. Clusters 0, 1, 2 aggregate the most part of the movies, mostly with R and PG-13 ratings. That may show that we have mostly adult films in the dataset. That is not a surprise due to considered subject. The number of films at the clusters are: 2 - 233, 1 - 122, 0 - 119, 3 - 55, 4 - 16.

Next we compare the mean values of features between the clusters. The following barplot shows the mean body count per clusters, separated by ratings. The height of each bar shows the average body count of movies of a certain rating in a cluster. The vertical black bar represents the spread - the highest point shows the maximum, the lowest point shows the minimum.



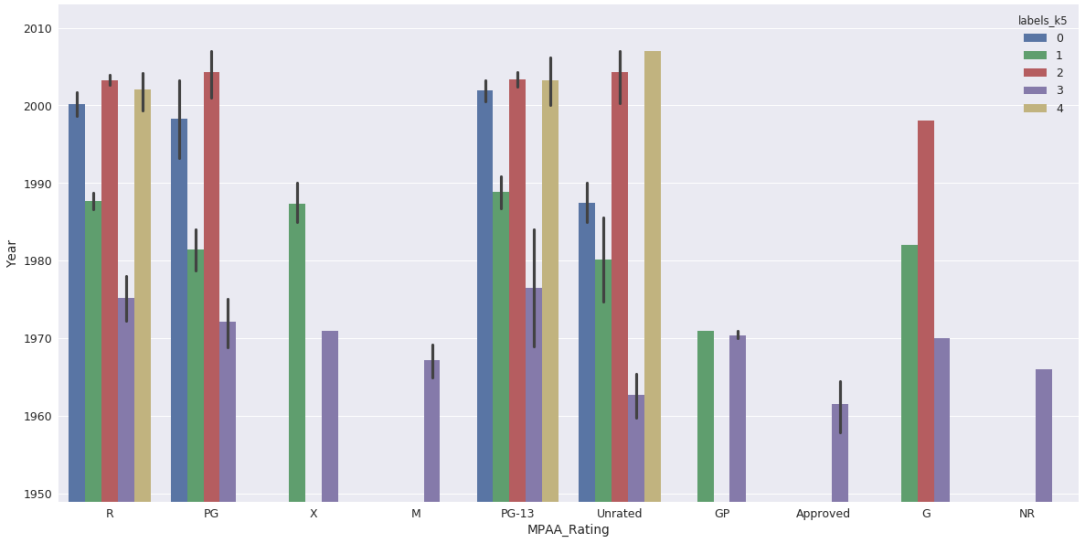
It's clear that cluster 4 contains the movies with the highest body count, as was supposed in the cluster interpretation. It contains the PG-13 movie with the highest body count in the dataset (836): *"Lord of the Rings: Return of the King"*. The highest point of the black bars shows this movie.

Next we consider the average length of the movies by clusters:




This plot is not very informative. We can see that clusters 1 and 2 are on avearage the shortest ones. And the movies from the cluster 4, with highest body count, are long on average.

Next we consider the release years of movies by cluster and ratings.




It can be seen that the cluster 3 contains mostly old movies, as was discovered before. Now we can see that these are mostly pre-1980 movies.

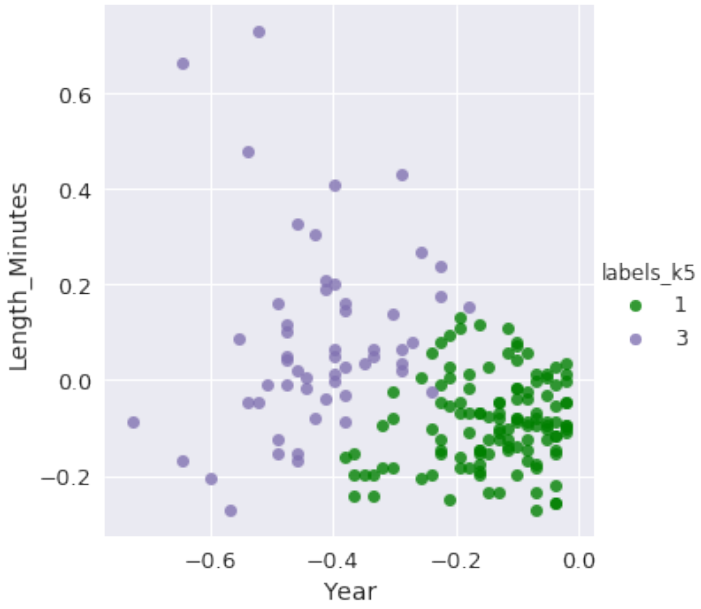
Cluster 1 contains movies from about the 90-s. And it is interesting that the highest body count movies, captured by cluster 4, are mostly recent - post-2010.

Moreover, from this diagram it can be seen that for some ratings there are films only from one-two clusters, mostly from the third one, capturing old films. These ratings were used in previous years, but then were renamed or changed. As there are only several such movies in our dataset, we do not think it influence any results.

## 2.4 Bootstrap

In this part of the paper we inspect two clusters: 1 and 3. As was mentioned, cluster 3 contains mostly old, pre-1980 movies and cluster 1 has movies from the 90-s.

First, we consider length and release year, and then we focus on our main feature, the body count. From the scatter plot the difference between the clusters can be clearly seen.

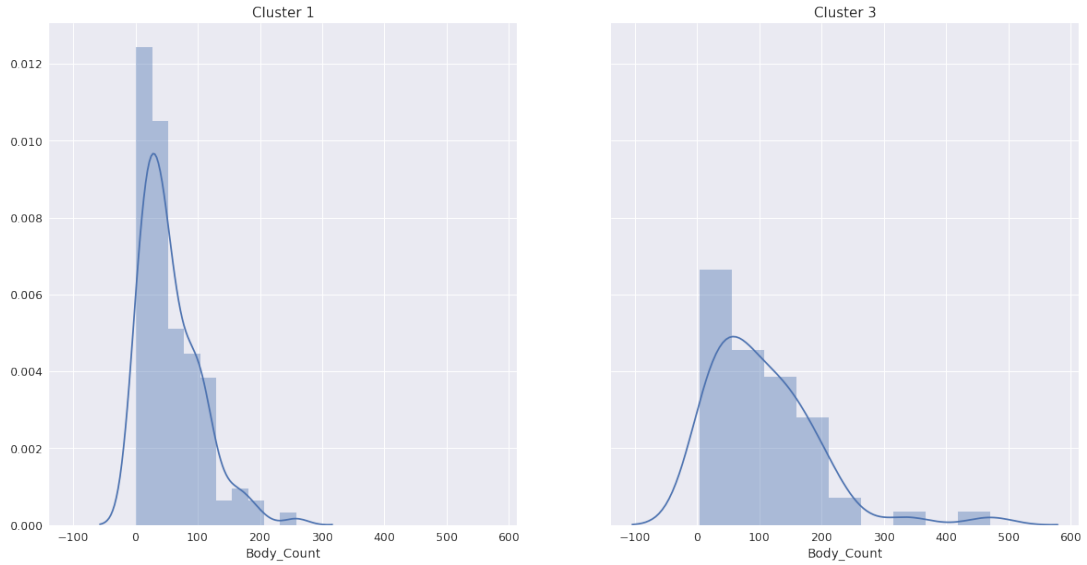



We inspect the distribution of our primary feature, body count, in the two clusters.

In [48]:

```
print(cluster_0[target_feature].describe())
print(cluster_1[target_feature].describe())
```

```
fig = plt.figure(figsize=(20,10))
axes = fig.subplots(1, 2, sharex=True, sharey=True)

sns.distplot(cluster_0.Body_Count, ax=axes[0])
sns.distplot(cluster_1.Body_Count, ax=axes[1])
```

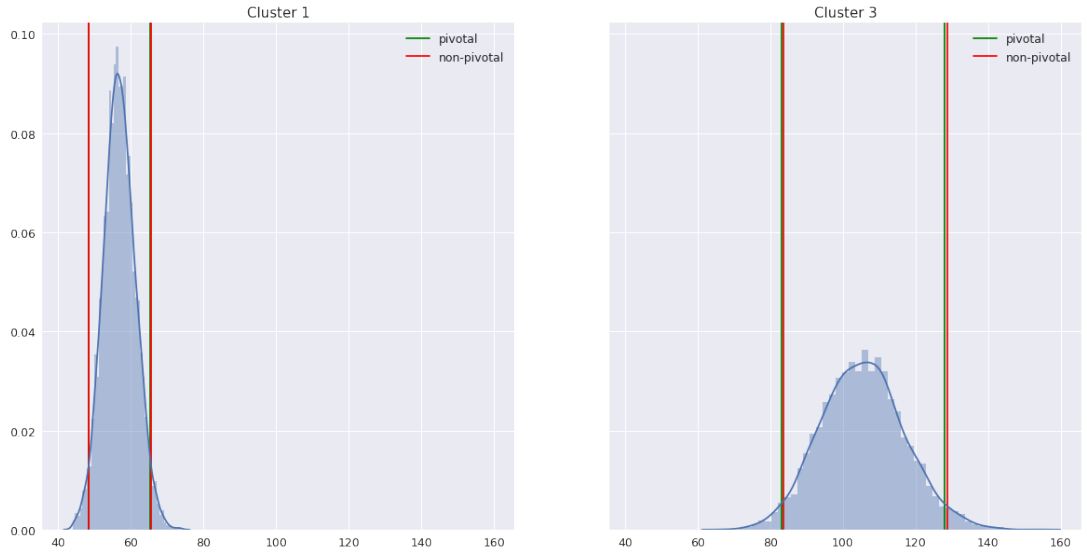



The parameters of the distributions are listed in the table. The distributions seem to be different but we cannot be sure as the samples are rather small.

| **Parameter** | **Cluster 1** | **Cluster 3** |
|---|---|---|
| count | 122.000000 | 55.000000 |
| mean | 56.795082 | 105.654545 |
| std | 47.931414 | 87.318323 |
| min | 1.000000 | 4.000000 |
| 25% | 20.000000 | 44.500000 |
| 50% | 42.500000 | 91.000000 |
| 75% | 87.250000 | 147.000000 |
| max | 258.000000 | 471.000000 |

To compare the distribution between the clusters we apply bootstrap in both pivotal and non-pivotal versions with 5000 samples. We obtain that the distributions of Body_Count really differ between these clusters. They not only have different means, as evident by confidence intervals, but also have different bell shapes.

We provide the following histogram plots for bootstrap sample means. Green lines show pivotal 95% confidence intervals, red lines show 95% non-pivotal confidence intervals. It it interesting that pivotal and non-pivotal confidence intervals are nearly identical. From that we conclude that the distribution of the feature is approximately Gaussian.




| Parameter | Cluster 1 | Cluster 3 |
|---|---|---|
| mean | 56.844 | 105.385 |
| pivotal | (48.317, 65.371) | (82.949, 127.823) |
| non-pivotal | (48.474, 65.590) | (83.672, 128.711) |

The code for bootstrap implementation and the confidence intervals is below. We use pivotal and non-pivotal bootstrap for the confidence intervals.

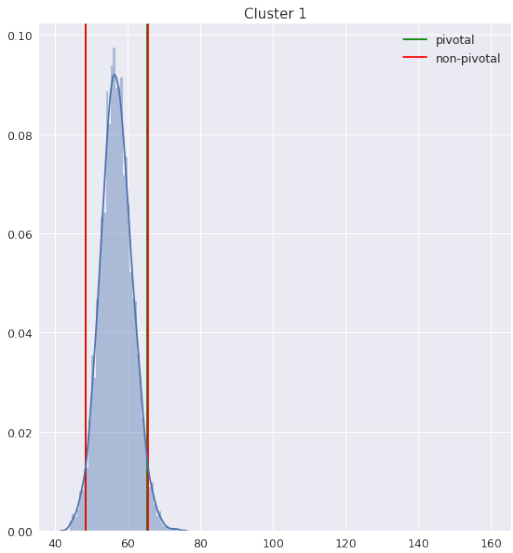
```
In [91]: def bootstrap_sample(vec, size):
return np.random.choice(vec, size=(vec.shape[0], size), replace=True)

def bootstrap_means(srs,     sample_amount=5000):
    samples_ix = bootstrap_sample(srs.index, size=sample_amount).T
    means = np.array([srs.loc[sample].mean() for sample in samples_ix])
    return means

b_means_cluster_0 = bootstrap_means(cluster_0.Body_Count)
b_means_cluster_1 = bootstrap_means(cluster_1.Body_Count)

def confidence_interval_pivotal(vec):
    mean = vec.mean()
    std = vec.std()
```

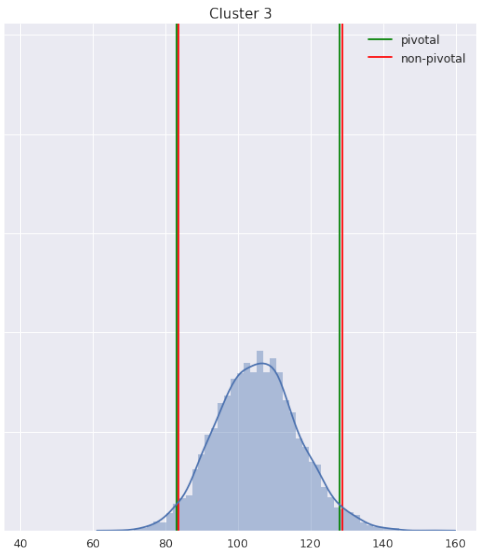
```
    return [mean-1.96*std, mean+1.96*std]

def confidence_interval_non_pivotal(vec, alpha):
    left = np.percentile(vec, (100-alpha)/2)
    right = np.percentile(vec, alpha+(100-alpha)/2)
    return [left, right]

def distplot_with_conf_intervals(vec, ax=None):
    if not ax:
        ax = plt.gca()
    for x in confidence_interval_pivotal(vec):
        line = ax.axvline(x=x, color='g')
    line.set_label('pivotal')
    for x in confidence_interval_non_pivotal(vec, 95):
        line = ax.axvline(x=x, color='r')
    line.set_label('non-pivotal')
    ax.legend(loc='best')
    return sns.distplot(vec, ax=ax, norm_hist=False)
```

Next we build the 95% confidence intervals for the grand mean of the feature by using bootstrap. Also we use bootstrap to compare cluster 0 to the grand mean. We obtain:

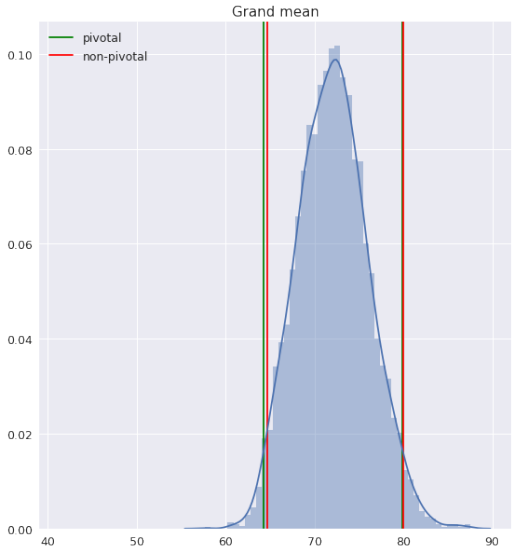
| **Parameter** | **Cluster 1** | **Grand mean** |
|---|---|---|
| mean | 56.844 | 72.157 |
| pivotal | (48.317, 65.371) | (64.374, 79.340) |
| non-pivotal | (48.474, 65.590) | (64.65, 80.238) |

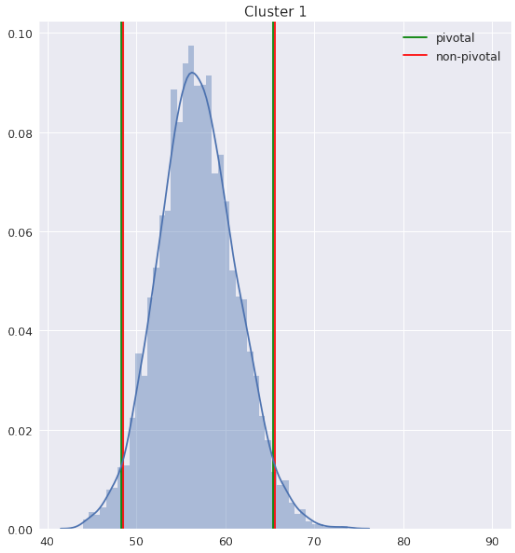
Comparison between the grand mean and cluster 0:



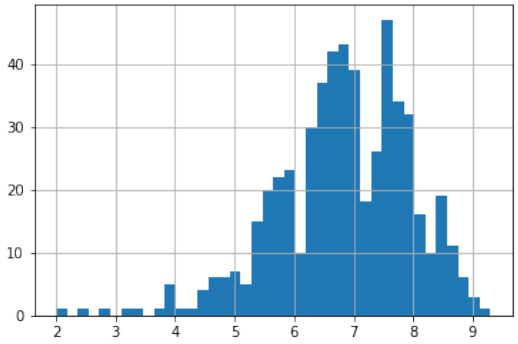

We pay attention here to the fact that the bell shape of cluster 1 body count resembles the grand mean bell shape closely.



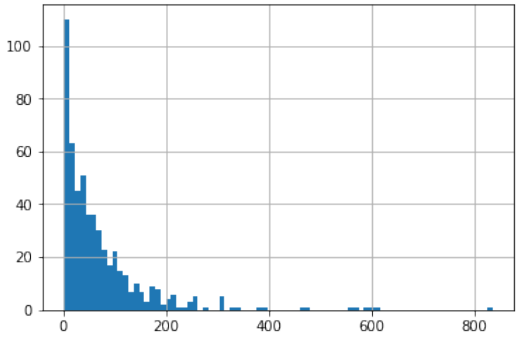




# 3 Contingency Table Analysis

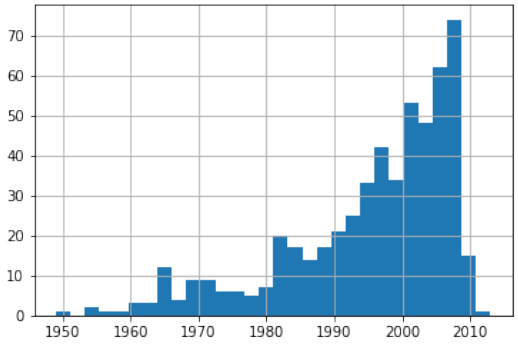
In this part of the paper we consider the contingency tables. We take three nominal features from the dataset, producing them by splitting from the quantitative features. Consider the histograms for IMDB_Rating, Year and Body_count:




IMDB_Rating




Body_count




Year

To make the features we split all three attributes with reasonable visible thresholds, shown on the graphs. We take 4 intervals for each attribute. For the IMDB rating we choose 6, 7 and 8 as thresholds, as they are caught by the histogram. It can be really well interpreted as very bad, bad, good and very good films. For the Year we choose 1980, 1990, 2000, 2005 as thresholds. In the previous part we had the cluster of old films, which appeared before 1980, and we can use it here to set the thresholds. All the others were chosen to make the groups approximately equal. For the on-screen deaths we have chosen 20, 50, 120.

```
def categorize_func(split_points):
    def categorize(param):
        split_points.sort()
        if (param < split_points[0]):
            return '<' + str(split_points[0])
        for index in range(len(split_points) - 1):
            if param < split_points[index + 1]:
                return '' + str(split_points[index])
        return '' + str(split_points[-1])
    return categorize
```

```
bodies = df['Body_Count'].apply(categorize_func([10, 100, 200]))
imdb_rating = df['IMDB_Rating'].apply(categorize_func([5.0, 7.0, 8.0]))
year = df['Year'].apply(categorize_func([1990, 2000, 2005]))
```

Next we build the contingency tables for the determined features:

```
contingency_table_imdb_bodies = pd.crosstab(bodies, imdb_rating, margins=True, \
                                margins_name='Total')
contingency_table_year_bodies = pd.crosstab(bodies, year, margins=True, margins_name='Total')
```

| IMDB_Rating<br>Body_Count | <6.0 | ⩾6.0 | ⩾7.0 | ⩾8.0 | Total |
|---|---|---|---|---|---|
| <20 | 28 | 65 | 47 | 22 | 162 |
| ⩾120 | 14 | 20 | 44 | 14 | 92 |
| ⩾20 | 28 | 40 | 39 | 23 | 130 |
| ⩾50 | 39 | 49 | 52 | 21 | 161 |
| Total | 109 | 174 | 182 | 80 | 545 |

| Year | <1980 | ⩾1980 | ⩾1990 | ⩾2000 | ⩾2005 | Total |
|---|---|---|---|---|---|---|
| Body_Count | | | | | | |
| <20 | 12 | 14 | 39 | 44 | 53 | 162 |
| ⩾120 | 20 | 10 | 13 | 24 | 25 | 92 |
| ⩾20 | 10 | 23 | 48 | 19 | 30 | 130 |
| ⩾50 | 23 | 25 | 39 | 30 | 44 | 161 |
| Total | 65 | 72 | 139 | 117 | 152 | 545 |

The first table shows the joint distribution of the IMDB rating and the number of deaths. The second shows the joint distribution between the Year and the number of deaths.

## 3.1 Relative probabilities tables

To see the relative probabilities we build the relative frequency tables over the same features:

```
def make_relative(probabilites_df):
    res = probabilites_df.copy()
    res.drop(index='Total', columns='Total', inplace=True)
    for col in res:
        for row in res[col].index:
            res[col][row] /= probabilites_df['Total'][row]
    return res

def normalize(contingency_df):
    res = contingency_df.copy()
```

```
    res = res.astype('float')
    for col in res:
        for row in res[col].index:
            res[col][row] /= contingency_df['Total']['Total']
    return res
make_relative(normalize(contingency_table_imdb_bodies)) * 100
```

The table for the rating and the number of deaths:

| IMDB_Rating | <6.0 | ⩾6.0 | ⩾7.0 | ⩾8.0 |
|---|---|---|---|---|
| Body_Count | | | | |
| <20 | 17.283951 | 40.123457 | 29.012346 | 13.580247 |
| ⩾20 | 21.538462 | 30.769231 | 30.000000 | 17.692308 |
| ⩾50 | 24.223602 | 30.434783 | 32.298137 | 13.043478 |
| ⩾120 | 15.217391 | 21.739130 | 47.826087 | 15.217391 |

We can conclude that the film with more than 200 bodies probably will be well appreciated: it has probability nearly 48% to have 7 and 22% to have 6, while the film with <20 deaths will most probably have 6 (40%) and less probably 7 (29%).

The table for the year and the number of deaths:

| Year | <1980 | ⩾1980 | ⩾1990 | ⩾2000 | ⩾2005 |
|---|---|---|---|---|---|
| Body_Count | | | | | |
| <20 | 7.407407 | 8.641975 | 24.074074 | 27.160494 | 32.716049 |
| ⩾20 | 7.692308 | 17.692308 | 36.923077 | 14.615385 | 23.076923 |
| ⩾50 | 14.285714 | 15.527950 | 24.223602 | 18.633540 | 27.329193 |
| ⩾120 | 21.739130 | 10.869565 | 14.130435 | 26.086957 | 27.173913 |

We see that if movie has high amount of deaths it is most probably the new one of very old one. And, as we have much more new films in the dataset, the probability for each cathegory of deaths grows with the year (except one cell).

## 3.2 Quetelet index tables

Now we build Quetelet index tables:

```
def make_quetelet(probabilites_df):
    res = probabilites_df.copy()
    res.drop(index='Total', columns='Total', inplace=True)
    for col in res:
        for row in res[col].index:
            res[col][row]=(probabilites_df[col][row]/probabilites_df['Total'][row] - \
```

```
                    probabilites_df[col]['Total']) / probabilites_df[col]['Total']
return res
make_quetelet(normalize(contingency_table_imdb_bodies)) * 100
make_quetelet(normalize(contingency_table_year_bodies)) * 100
```

| IMDB_Rating | <6.0 | ⩾6.0 | ⩾7.0 | ⩾8.0 |
|---|---|---|---|---|
| Body_Count | | | | |
| <20 | -13.580247 | 25.674046 | -13.122371 | -7.484568 |
| ⩾120 | -23.913043 | -31.909045 | 43.215480 | 3.668478 |
| ⩾20 | 7.692308 | -3.625111 | -10.164835 | 20.528846 |
| ⩾50 | 21.118012 | -4.672664 | -3.283052 | -11.141304 |

The table shows the relative probability compared to the average. We can conclude that Films that have 120 or more bodies are more appreciated: they have odds to have rating more than 8.0 ≈4% more than average and chances that film will have rating less than 6.0 is ≈23% less than average

| Year | <1980 | ⩾1980 | ⩾1990 | ⩾2000 | ⩾2005 |
|---|---|---|---|---|---|
| Body_Count | | | | | |
| <20 | -37.891738 | -34.585048 | -5.608846 | 26.516830 | 17.304256 |
| ⩾120 | 82.274247 | -17.723430 | -44.596497 | 21.516165 | -2.567220 |
| ⩾20 | -35.502959 | 33.920940 | 44.770338 | -31.919790 | -17.257085 |
| ⩾50 | 19.780220 | 17.537957 | -5.022566 | -13.202739 | -2.010461 |

This is an quetelet table for Year and the number of deaths. We can conclude that the films with the highest amount of deaths have probability ≈82% more then average to be released before 1980. At the same time films with less then 20 deaths are most probable released after 2000.

## 3.3 Chi-square & Quetelet index

We apply Pearson's chi-squared test to measure the correlation between two variables. The value is the same as Summary Quetelet index. It shows the average relative increase in the prediction of the number ofthe number of deaths in the movie, when the year or the IMDB rating becomes known.

First, we find expected number of films in each category in assumption of independence of features. Also we find number of degrees of freedom. Let's consider the Year and the Body_Count:

```
obs_values_year_bodies = trunc_total(contingency_table_year_bodies)
exp_values_year_bodies = expected_values(contingency_table_year_bodies)
degrees_of_freedom_year_bodies = reduce(mul, map(lambda x: x - 1, \
                                    obs_values_year_bodies.shape), 1)
ddof_year_bodies = obs_values_year_bodies.size - degrees_of_freedom_year_bodies - 1
```

The tables for observed frequencies and expected under the independence condition:

| Year | <1980 | ⩾1980 | ⩾1990 | ⩾2000 | ⩾2005 |
|---|---|---|---|---|---|
| Body_Count | | | | | |
| <20 | 12 | 14 | 39 | 44 | 53 |
| ⩾120 | 20 | 10 | 13 | 24 | 25 |
| ⩾20 | 10 | 23 | 48 | 19 | 30 |
| ⩾50 | 23 | 25 | 39 | 30 | 44 |

| Year | <1980 | ⩾1980 | ⩾1990 | ⩾2000 | ⩾2005 |
|---|---|---|---|---|---|
| Body_Count | | | | | |
| <20 | 19.321101 | 21.401835 | 41.317431 | 34.777982 | 45.181651 |
| ⩾120 | 10.972477 | 12.154128 | 23.464220 | 19.750459 | 25.658716 |
| ⩾20 | 15.504587 | 17.174312 | 33.155963 | 27.908257 | 36.256881 |
| ⩾50 | 19.201835 | 21.269725 | 41.062385 | 34.563303 | 44.902752 |

The obtained degree of freedom for $\chi^2$ test is 12.

The $\chi^2$ test for the hypothesis of independence between the year and number of bodies:

```
scipy.stats.chisquare(trunc_total(contingency_table_year_bodies), \
        expected_values(contingency_table_year_bodies), ddof=ddof_year_bodies, axis=None)
```

We can see that critical pvalue is less than $10^{-4}$. So we can reject the hypothesis of independence between the year and the number of deaths in the movie with level of confidence 99.99

Now consider the same tables for the other feature, IMDB rating, with the number of bodies. Observed frequencies and expected under the independence condition:

| IMDB_Rating | <6.0 | ⩾6.0 | ⩾7.0 | ⩾8.0 |
|---|---|---|---|---|
| Body_Count | | | | |
| <20 | 28 | 65 | 47 | 22 |
| ⩾120 | 14 | 20 | 44 | 14 |
| ⩾20 | 28 | 40 | 39 | 23 |
| ⩾50 | 39 | 49 | 52 | 21 |

| MDB_Rating | <6.0 | ⩾6.0 | ⩾7.0 | ⩾8.0 |
|---|---|---|---|---|
| Body_Count | | | | |
| <20 | 32.4 | 51.721101 | 54.099083 | 23.779817 |
| ⩾120 | 18.4 | 29.372477 | 30.722936 | 13.504587 |
| ⩾20 | 26.0 | 41.504587 | 43.412844 | 19.082569 |
| ⩾50 | 32.2 | 51.401835 | 53.765138 | 23.633028 |

The number of degrees of freedom is 9, and it is expectedly less than in previous case as we have less categories for rating.

The $\chi^2$ test results for the hypothesis of independence between the IMDB rating and number of bodies:

```
Power_divergenceResult(statistic=18.231013449960972, pvalue=0.03258602359739147)
```

The critical pvalue here is about 3%. So we can reject the hypothesis

of independence with level of confidence 95%, but we cannot reject the hypothesis with level of confidence 97% or higher.

Now we compute chi-square summary quetelet index tables for both features:

| IMDB_Rating | <6.0 | Total | ⩾6.0 | ⩾7.0 |
|---|---|---|---|---|
| Body_Count | | | | |
| <20 | -3.802469 | 5.071541 | 16.688130 | -6.167515 |
| Total | 3.239576 | 18.231013 | 6.566671 | 7.175824 |
| ⩾120 | -3.347826 | 9.798763 | -6.381809 | 19.014811 |
| ⩾20 | 2.153846 | 1.461151 | -1.450044 | -3.964286 |
| ⩾50 | 8.236025 | 1.899559 | -2.289605 | -1.707187 |

| Year | <1980 | Total | ⩾1980 | ⩾1990 | ⩾2000 | ⩾2005 |
|---|---|---|---|---|---|---|
| Body_Count | | | | | | |
| <20 | -4.547009 | 9.262296 | -4.841907 | -2.187450 | 11.667405 | 9.171256 |
| Total | 12.906996 | 39.298444 | 5.572056 | 11.545967 | 6.805703 | 2.467722 |
| ⩾120 | 16.454849 | 13.407037 | -1.772343 | -5.797545 | 5.163880 | -0.641805 |
| ⩾20 | -3.550296 | 14.499397 | 7.801816 | 21.489762 | -6.064760 | -5.177126 |
| ⩾50 | 4.549451 | 2.129715 | 4.384489 | -1.958801 | -3.960822 | -0.884603 |

The quetelet index is the sum of the values in the table. We compute it to make sure that it is equal to $\chi^2$ statistics.

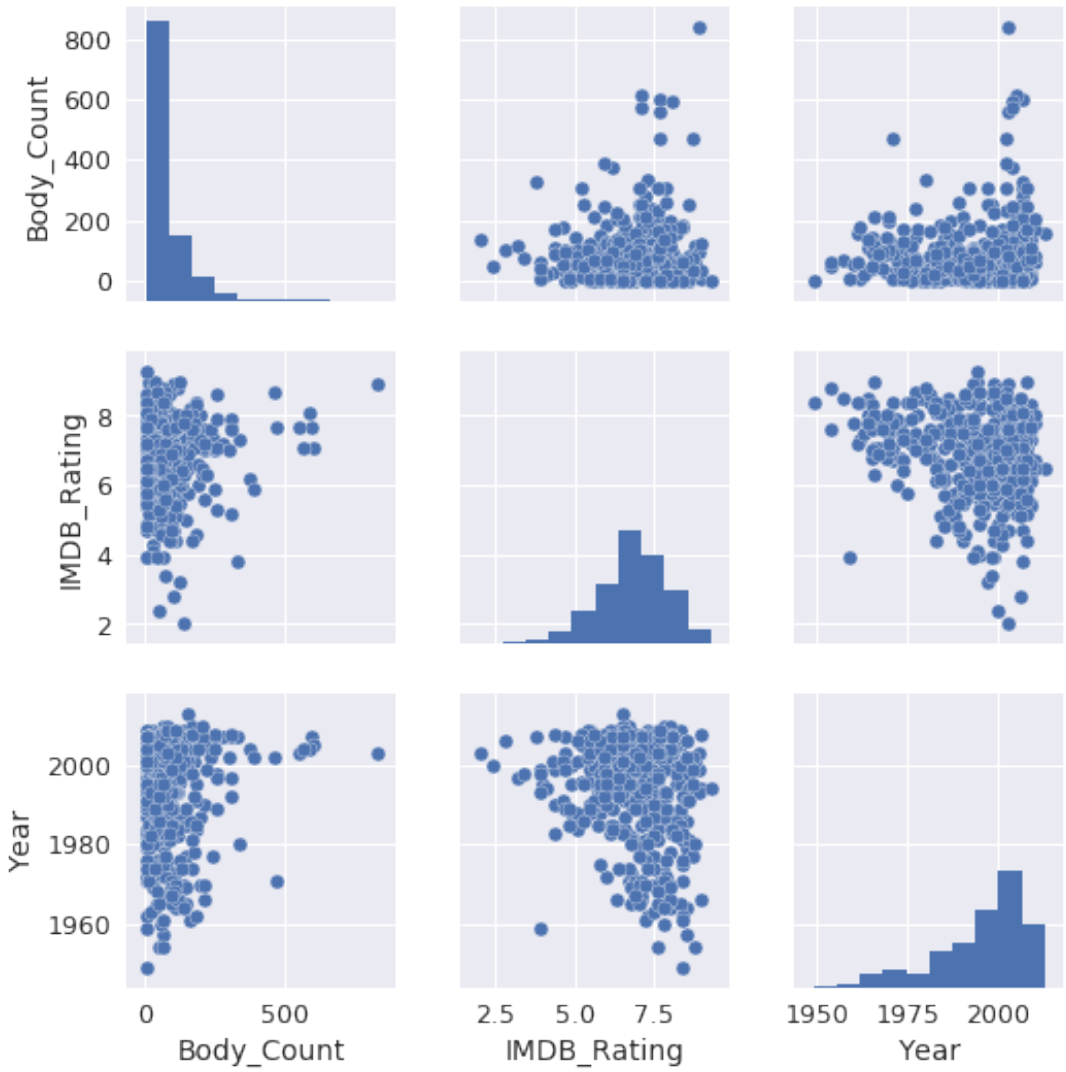
# 4 PCA: Hidden Factor & Data visualization

In this part of the paper we consider the principal component analysis. we choose three features to take into consederation in t part. First one is the Body_Count, as the main feature of interest. The second one is Year, as we know from the previous analysis that they are well connected. The third one is IMDB rating.

First we normalize the features. There are two ways of normalization, by standard deviation and by range:

```
def normalize_range(vec):
    return (vec - vec.mean())/(vec.max() - vec.min())
def normalize_std(vec):
    return (vec - vec.mean())/(vec.std())
```

Recall the plots of joint and self distributions of the features:




First we apply PCA and inspect the data scatter explained by each PC. We apply PCA to data centered and normalized by range, as proper PCA application requires data to be on the same scale. The supplementary functions may be found in the Applications.

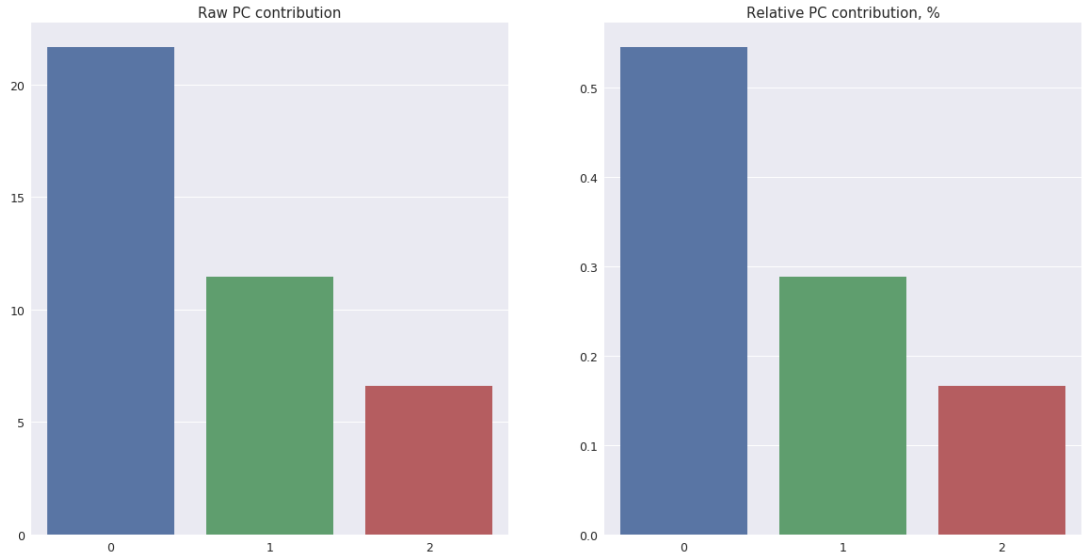
```
pc_range = PCA()
pc_range.fit(task_df_normalized_range)
scatter = data_scatter(task_df_normalized_range)
contibs = pd.Series([pc_contrib(s) for s in pc_range.singular_values_])
contibs_rel = pd.Series([pc_contrib_rel(s, scatter) for s in pc_range.singular_values_])
print('Data scatter', scatter)
print('Contributions', contibs, contibs_rel)

fig = plt.figure(figsize=(20,10))
axes = fig.subplots(1, 2, sharex=True)
axes[0].set_title('Raw PC contribution')
sns.barplot(contibs.index, contibs.values, ax=axes[0])
```

```
axes[1].set_title('Relative PC contribution, %')
sns.barplot(contibs_rel.index, contibs_rel.values, ax=axes[1])
```

The results of the application of principal components are the following. The contributions of the components to the explanation of the variance in distribution are:

| # of Component | Raw contribution | Relative contribution |
|---|---|---|
| 0 | 21.672266 | 0.545363 |
| 1 | 11.444299 | 0.287985 |
| 2 | 6.622589 | 0.166651 |

So, the first PC explains 55% of data scatter, the second one explains 29%, the third one 17%. The diagram illustrates the numbers from the table.

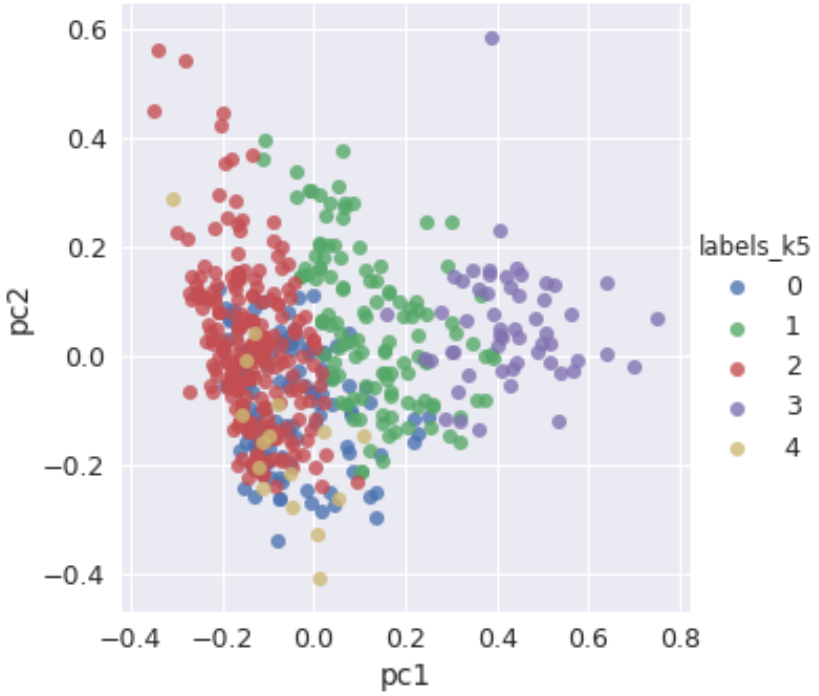



## 4.1 Visualization

We will use cluster labels from homework 1 to see how PCA helps vizualize them. So, we add the clusters (for k=5) into the set.

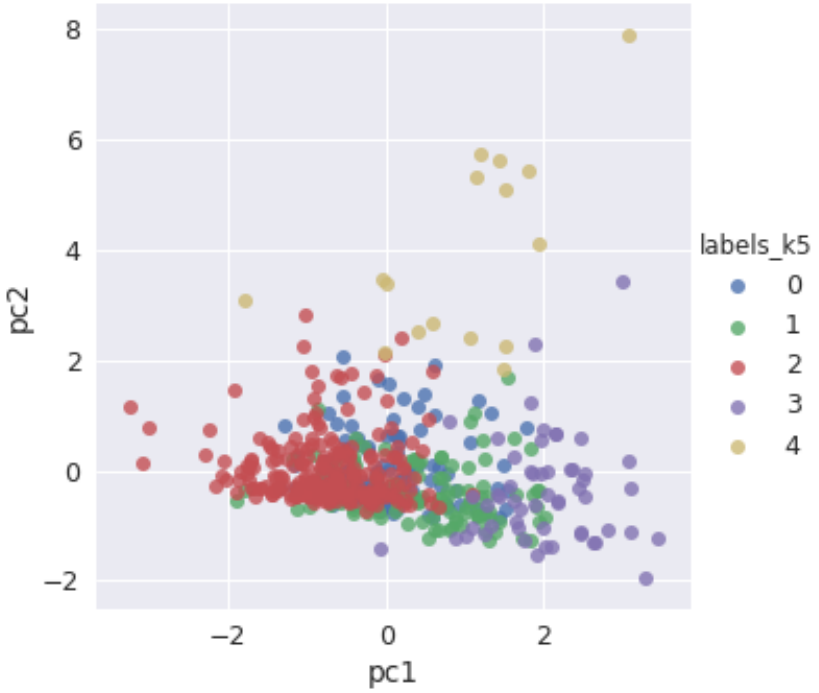
We apply PCA to data with clusters centered by mean and normalized by range to show how the first two components explain the scatter.

```
pc_range = PCA(n_components = 2)
pca_result = pc_range.fit_transform(task_df_normalized_range)
pca_result = pd.DataFrame(pca_result, columns=['pc1', 'pc2'])
pca_result['labels_k5'] = labels_k5
fig = plt.figure(figsize=(20,10))
sns.lmplot(x='pc1', y='pc2', fit_reg=False, hue='labels_k5', data=pca_result)
```




Evidently the clusters are easily separatable, except for 0 and 4, which tend to overlap with other clusters.

Next we Applying PCA to data normalized by standard deviation. The code is similar, the resulting plot is as follows:

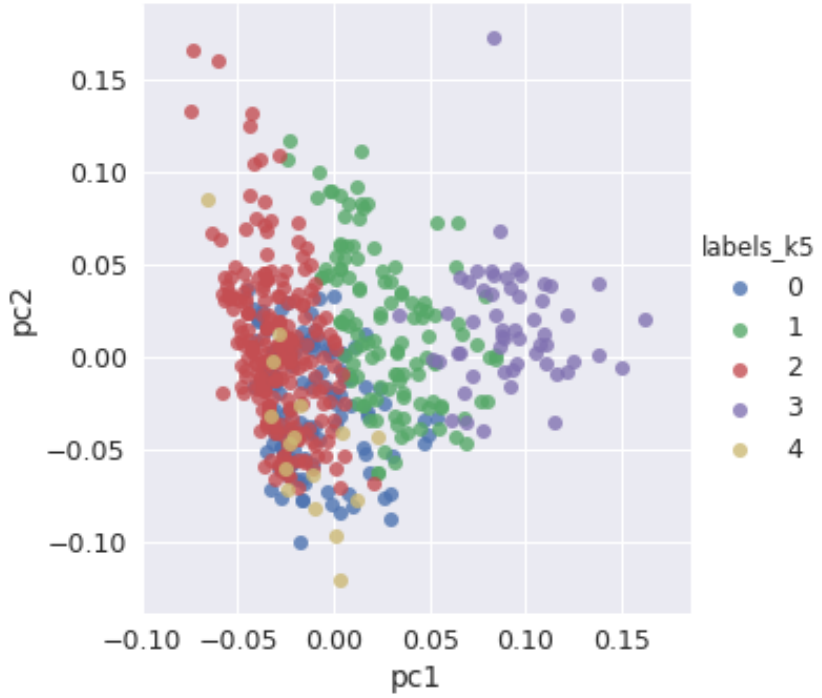



This one is much less descriptive, the clusters overlap alot.

Next, we apply conventional PCA to the same visualization.

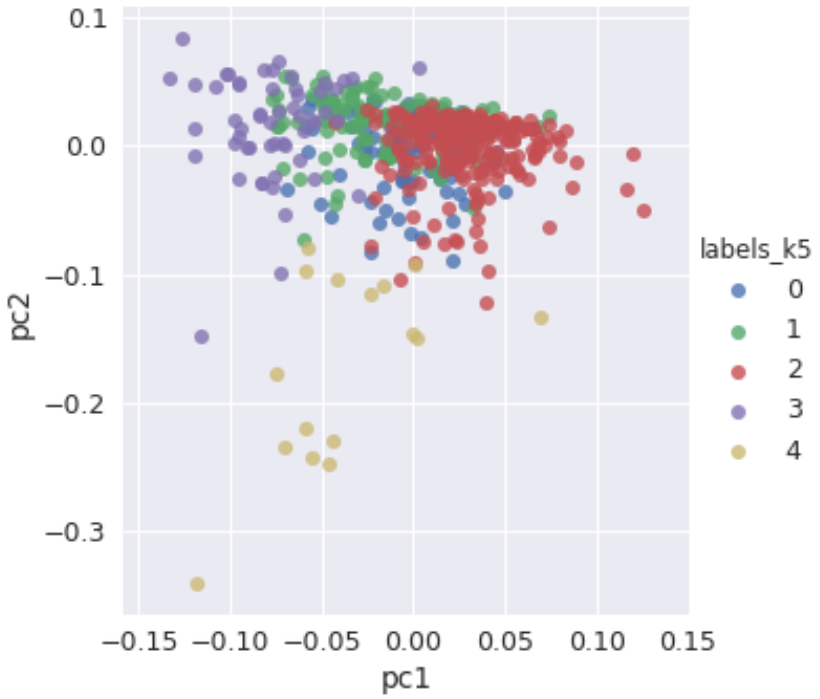
```
def conv_pca(X):
    X = X - X.mean(axis=0)

    covariance_matrix = (X.T.dot(X))/X.shape[0]

    eigvals, eigvectors = np.linalg.eig(covariance_matrix)
    order = np.argsort(eigvals)
    ind1, ind2 = order[-2:][::-1] # top 2 max eigvalues indices
    eigvectors = -eigvectors.T
    pc1 = X.dot(eigvectors[ind1])  / np.sqrt(X.shape[0]*abs(eigvals[ind1]))
    pc2 = X.dot(eigvectors[ind2])  / np.sqrt(X.shape[0]*abs(eigvals[ind2]))
    return pc1, pc2
```

Firstly we apply the conventional PCA to data normalized by range:




Applying conventional PCA to data normalized by range yields nearly identical results to using model-based PCA. That is expected, because it is the same method. When using conventional PCA it's important to center our data (while model-based PCA can be applied to non-centered data). Here the data is centered, so we get same results.

Secondly we apply the conventional PCA to data normalized by standard deviation. It also gives identical results, but rotated 180 degrees:




## 4.2 Hidden factor analysis

In this part we try to uncover the hidden factor behind the selected features and determine which of them contribute the most. In this function we implement applying PCA, extracting the first singular triplet and rescaling it:

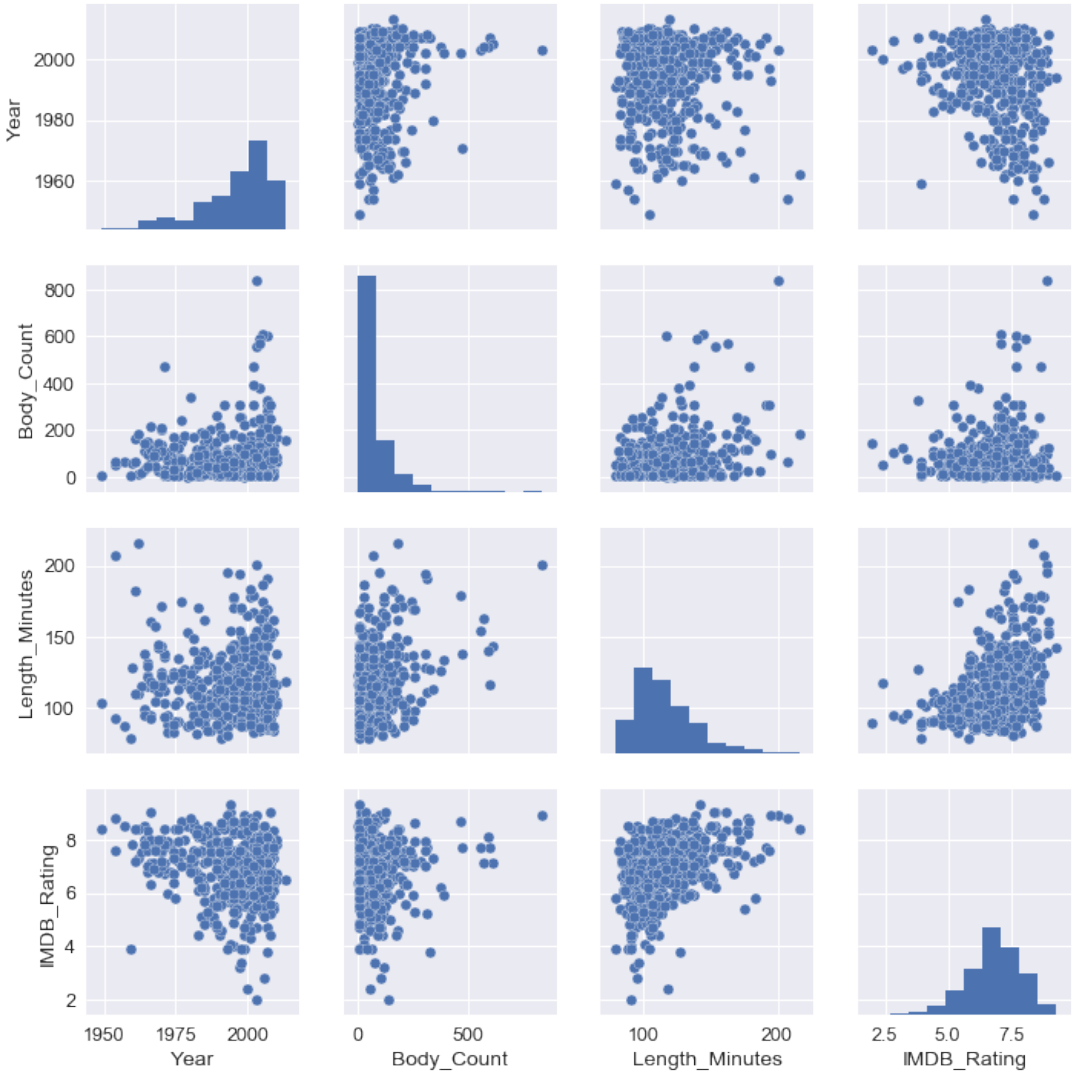
```
def get_feature_weights(X):
    u, s, vh = np.linalg.svd(X)
    z = abs(u[:, 0])
    sigma = max(s)
    c = abs(vh[:, 0])
    alpha = 1/c.sum()
    feature_weights = c*alpha
    contrib = sigma**2 / data_scatter(X)
    return feature_weights, contrib

weights, contrib = get_feature_weights(task_df)
print(list(task_df.columns))
print(contrib*100, weights.round(3)*100)
['Body_Count', 'IMDB_Rating', 'Year']
99.78585508347525 [ 3.5 96.4  0.1]
```

So, on the 0-100 rank we obtain that the first principal component explains 99.8% of data scatter. The most contributing feature is

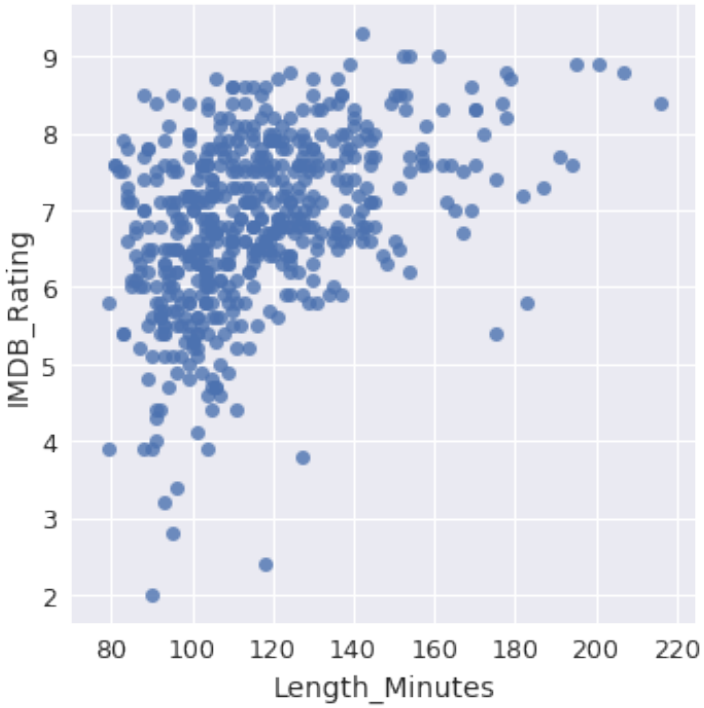
IMDB_Rating, which determines 96% of the PC.

# 5 2D regression

In this part of the paper we examine the 2D regressions. First, we choose two variables to take into consideration. All the scatter-plots are presented on the picture.



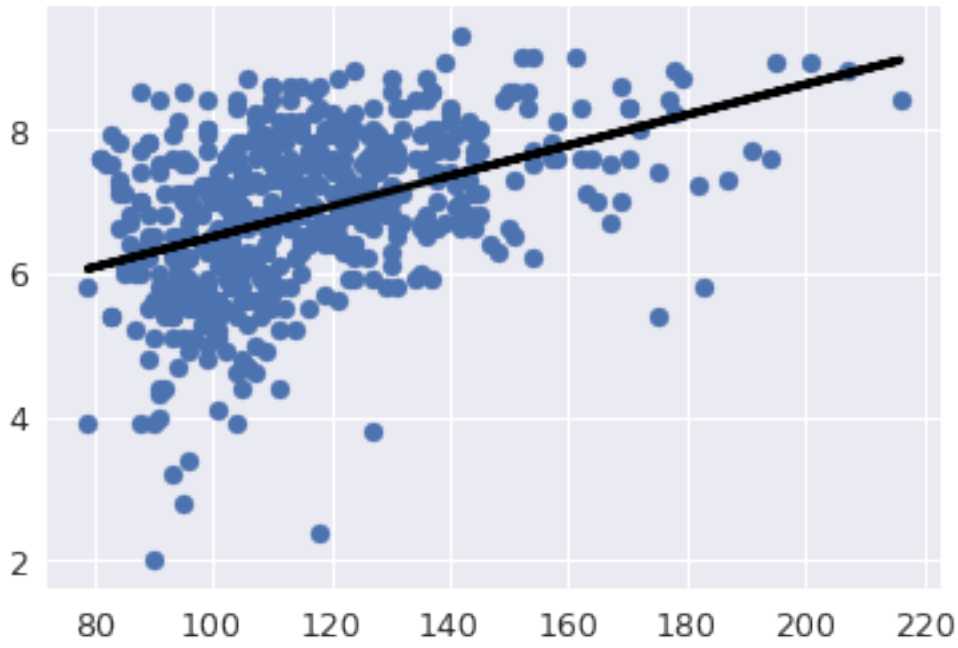

None of these has really "linear-like"plot, but we choose Length_Minutes and IMDB_Rating as the best variant. First, explore closer their scatter-plot.




Based on the visualization, we expect positive correlation here. We build a regression of Length_Minutes to IMDB_Rating.

```
training_x = df['Length_Minutes'].values.reshape(-1, 1)
training_y = df['IMDB_Rating'].values
classificator = LinearRegression().fit(training_x, training_y)
predicted_y = classificator.predict(training_x)
plt.scatter(training_x, training_y)
plt.plot(training_x, predicted_y, color='black', linewidth=3)
classificator.coef_
```

The visualization of the regression line:




The slope of the regression is 0.0211439. That means that every minute of the film gives additional 0.02114397 to the rating. It doesn't seem much, but these 0.02 may raise the film on tens of places in the search or in the top-movies rating, as there are >5 mln movies on IMDB rated at ten-point scale.

This correlation may follow the suggested hypothesis: the production of the long movie usually requires a huge budget, which will be approved if the film is supposed to be good. So, long films are among the good ones.

The determinacy coefficient and mean absolute error:

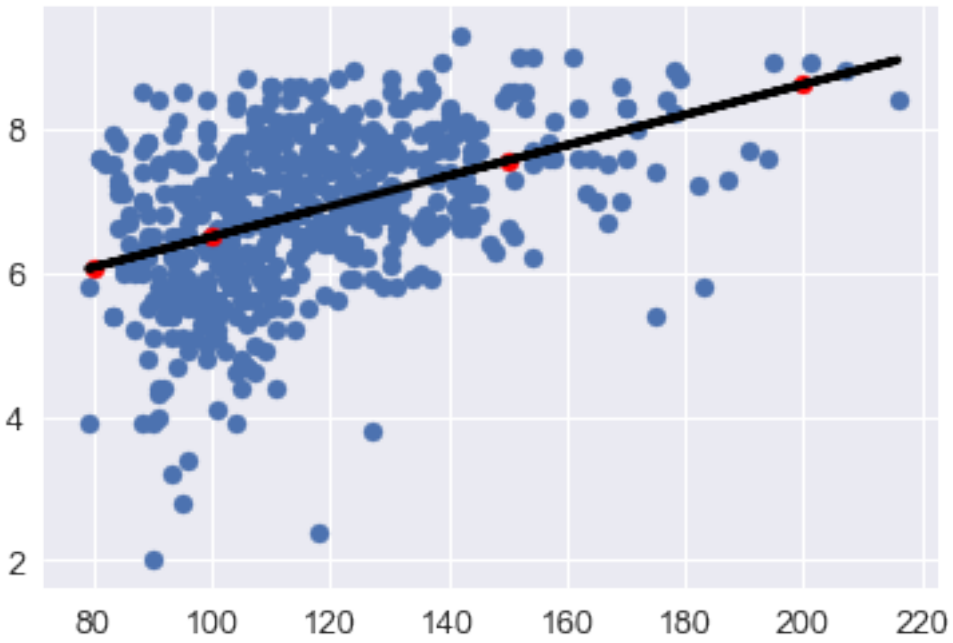
```
classificator.score(training_x, training_y)
Out[14]: 0.18600394479373705
mean_absolute_error(training_y, predicted_y)
Out[15]: 0.7823580398705626
```

The determinacy coefficient is 0.18600394479373705, which is very low value. At the same time mean absolute error is very high. It shows that the classificator works really bad on these features.

Nevertheless, we try to predict values of the rating by the length of the movie. We take four values for the length of the movie: [80, 100, 150, 200] minutes. The predicted ratings are: [6.0716766 , 6.49455604, 7.55175466, 8.60895327].

```
lengths_to_predict = np.array([60, 90, 150])
prdicted_ratings = classificator.predict(lengths_to_predict.reshape(-1, 1))
```

The visualization among the data is the following:




The predictions stay among the other points while we use the value for minutes from the same range that the sample values.

# 6 Conclusion

In this project we explore the data on movies. The the first section contains the descriprion of the data and some hypothesis. In the next section we apply the clustering methods to analyze the data. The third section contains the contingency table analysis, $\chi^2$ and quetelet index. In the next we study the hidden factors by principal component analysis. Then, we use 2D regression to find some more patterns.

The main feature of the consideration was the number of on-screen deaths in the movies. The most number in our data is 836 *("Lord of the Rings: Return of the King")*. There are several things of great interest among the patterns and correlations that we found. The old films, which were filmed before 1980 stay aside the others as they are mostly longer and highly-rated.

Modern films, of course, have higher number of deaths. And the films with a lot of violence are rated with stronger restrictions by MPAA.

PC analysis revealed the high ability of IMDB rating to describe the other numerical features, as it is the main contributor to the first component. That shows that the ratings really reflect some important properties and summarize them.

# 7 Applications

1. **MPAA Ratings interpretation.** Sourse: MPAA

   - **G General audiences**. All ages admitted. Nothing that would offend parents for viewing by children.
   - **PG Parental Guidance Suggested**. Some material may not be suitable for children. Parents urged to give "parental guidance". May contain some material parents might not like for their young children.
   - **M:** Suggested for Mature Audiences parental discretion advised (before 1984). The same as modern PG.
   - **GP** All Ages Admitted Parental Guidance Suggested (before 1972). Renamed to PG.
   - **PG-13 Parents Strongly Cautioned**. Some material may be inappropriate for children under 13. Parents are urged to be cautious. Some material may be inappropriate for pre-teenagers.
   - **R Restricted**. Under 17 requires accompanying parent or adult guardian. Contains some adult material. Parents are urged to learn more about the film before taking their young children with them.
   - **NC-17 Adults Only.** No One 17 and Under Admitted. Clearly adult. Children are not admitted.
   - **X**. No one under 17 admitted (before 1984). Was renamed to NC-17