Government of Russian Federation
Federal state autonomous educational institution
of higher professional education

National Research University
Higher School of Economics

Faculty of Computer Science
Master educational program
Data science

HOME ASSIGNMENT

for the course

Modern Methods for Data Analysis

1st year student
Pokoev Aleksei Petrovich
Instructor:
DSc, Mirkin B.G.

Moscow 2018

1 Introduction

In this work I analyzed the movie database, which I took from the http://www.kinobusiness.com/best/usa/world-fees/ and https://www.imdb.com.

2 Motivation

There are several reasons why I chose this database: 1. I have a huge interest in this database, partly because I love movies. 2. This dataset fits perfectly under the requirements 3. It provides the ability to analyze different correlations (e.g. how budget depends on the gross, etc).

3 The structure of dataset

Number of instance: 100
Number of Features: 8
Missing Attribute Values: None.

4 Features information:

1. **Year** – time when movie was released (1977, 1982, 1993, 1994, 1996, 1997, 1999, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016);
2. **Studio** – a company that produced a movie ('DISNEY', 'DREAMWORKS', 'FOX', 'LIONSGATE', 'NEW LINE', 'PARAMOUNT', 'SONY', 'SUMMIT', 'UNIVERSAL', 'WARNER BROS');
3. **Budget** – budget of a movie (in millions of dollars);
4. **Worldwide gross** – worldwide gross of a movie (in millions of dollars);
5. **Gross in USA** – gross of a movie in USA (in millions of dollars);
6. **Runtime** – duration of a movie (in minutes);
7. **Certification** – age limit (0, 6, 12, 16, 18).
8. **IMDB score** – rating of the movie in the popular internet database imdb.com (from 0 to 10).

5 List of top-100 Highest-grossing films

First of all, let's load our data

```
In [1]: import pandas as pd
        import numpy as np
```

```
In [132]: data = pd.read_csv('film_gross_1_2.csv', sep='\t', index_col=0)
```

```
In [133]: data
```

```
Out[133]:                                          Title  Year        Studio  \
          0                                       AVATAR  2009           FOX
          1                                      TITANIC  1997     PARAMOUNT
          2         STAR WARS: EPISODE VII - THE FORCE AWAKENS  2015        DISNEY
```

3	JURASSIC WORLD	2015	UNIVERSAL
4	THE AVENGERS	2012	DISNEY
5	FURIOUS 7	2015	UNIVERSAL
6	AVENGERS: AGE OF ULTRON	2015	DISNEY
7	HARRY POTTER AND THE DEATHLY HALLOWS: PART 2	2011	WARNER BROS.
8	FROZEN	2013	DISNEY
9	IRON MAN 3	2013	DISNEY
10	MINIONS	2015	UNIVERSAL
11	CAPTAIN AMERICA: CIVIL WAR	2016	DISNEY
12	TRANSFORMERS: DARK OF THE MOON	2011	PARAMOUNT
13	THE LORD OF THE RINGS: THE RETURN OF THE KING	2003	NEW LINE
14	SKYFALL	2012	SONY
15	TRANSFORMERS: AGE OF EXTINCTION	2014	PARAMOUNT
16	THE DARK KNIGHT RISES	2012	WARNER BROS.
17	TOY STORY 3	2010	DISNEY
18	PIRATES OF THE CARIBBEAN: DEAD MAN'S CHEST	2006	DISNEY
19	PIRATES OF THE CARIBBEAN: ON STRANGER TIDES	2011	DISNEY
20	JURASSIC PARK	1993	UNIVERSAL
21	STAR WARS: EPISODE I - THE PHANTOM MENACE	1999	FOX
22	ALICE IN WONDERLAND	2010	DISNEY
23	ZOOTOPIA	2016	DISNEY
24	THE HOBBIT: AN UNEXPECTED JOURNEY	2012	WARNER BROS.
25	THE DARK KNIGHT	2008	WARNER BROS.
26	HARRY POTTER AND THE SORCERER'S STONE	2001	WARNER BROS.
27	DESPICABLE ME 2	2013	UNIVERSAL
28	THE LION KING	1994	DISNEY
29	THE JUNGLE BOOK	2016	DISNEY
..	...	...	...
70	THE HUNGER GAMES: MOCKINGJAY - PART 1	2014	LIONSGATE
71	SHREK FOREVER AFTER	2010	PARAMOUNT
72	X-MEN: DAYS OF FUTURE PAST	2014	FOX
73	MADAGASCAR 3: EUROPE'S MOST WANTED	2012	PARAMOUNT
74	THE CHRONICLES OF NARNIA: THE LION, THE WITCH ...	2005	DISNEY
75	MONSTERS UNIVERSITY	2013	DISNEY
76	THE MATRIX RELOADED	2003	WARNER BROS.
77	UP	2009	DISNEY
78	GRAVITY	2013	WARNER BROS.
79	CAPTAIN AMERICA: THE WINTER SOLDIER	2014	DISNEY
80	THE TWILIGHT SAGA: BREAKING DAWN - PART 1	2011	SUMMIT
81	DAWN OF THE PLANET OF THE APES	2014	FOX
82	THE TWILIGHT SAGA: NEW MOON	2009	SUMMIT
83	TRANSFORMERS	2007	PARAMOUNT
84	THE AMAZING SPIDER-MAN 2	2014	SONY
85	SUICIDE SQUAD	2016	WARNER BROS.
86	THE TWILIGHT SAGA: ECLIPSE	2010	SUMMIT
87	MISSION: IMPOSSIBLE - GHOST PROTOCOL	2011	PARAMOUNT
88	THE HUNGER GAMES	2012	LIONSGATE
89	MISSION: IMPOSSIBLE - ROGUE NATION	2015	PARAMOUNT

90	FORREST GUMP	1994	PARAMOUNT
91	INTERSTELLAR	2014	PARAMOUNT
92	THE SIXTH SENSE	1999	DISNEY
93	MAN OF STEEL	2013	WARNER BROS.
94	KUNG FU PANDA 2	2011	PARAMOUNT
95	ICE AGE: THE MELTDOWN	2006	FOX
96	BIG HERO 6	2014	DISNEY
97	PIRATES OF THE CARIBBEAN: THE CURSE OF THE BLA...	2003	DISNEY
98	THE HUNGER GAMES: MOCKINGJAY PART 2	2015	LIONSGATE
99	STAR WARS: EPISODE II - ATTACK OF THE CLONES	2002	FOX

	Budget	Worldwide gross	Gross in USA	Runtime	Certification	\
0	237.0	2787.965087	760.507625	162	12	
1	200.0	2186.772302	658.672302	194	12	
2	245.0	2068.178225	936.662225	138	12	
3	150.0	1670.400637	652.270625	124	12	
4	220.0	1519.557910	623.357910	143	12	
5	190.0	1516.045911	353.007020	137	16	
6	250.0	1405.413868	459.005868	141	12	
7	125.0	1341.511219	381.011219	130	12	
8	150.0	1276.480355	400.738009	102	0	
9	200.0	1215.439994	409.013994	130	12	
10	74.0	1159.398397	336.045770	91	6	
11	250.0	1152.745930	408.059143	147	12	
12	195.0	1123.794079	352.390543	154	12	
13	94.0	1119.928711	377.845095	201	12	
14	200.0	1108.561013	304.360277	143	16	
15	210.0	1104.039076	245.439076	165	12	
16	250.0	1084.939099	448.139099	164	12	
17	200.0	1066.969703	415.004880	103	6	
18	225.0	1066.179725	423.315812	151	12	
19	250.0	1045.713802	241.071802	136	12	
20	63.0	1029.153862	402.453882	127	6	
21	115.0	1027.044677	474.544677	136	0	
22	200.0	1025.467110	334.191110	108	12	
23	150.0	1023.446389	341.268248	108	6	
24	180.0	1021.103568	303.003568	169	12	
25	185.0	1004.558444	534.858444	152	12	
26	125.0	974.755371	317.575550	152	12	
27	76.0	970.761885	368.061265	98	0	
28	45.0	968.483777	422.783777	89	0	
29	175.0	964.062422	363.928757	106	6	
..	...	...	...	...	...	
70	125.0	755.356711	337.135885	123	12	
71	165.0	752.600867	238.736787	93	6	
72	200.0	747.862775	233.921534	132	12	
73	145.0	746.921274	216.391482	93	0	
74	180.0	745.013115	291.710957	143	6	

75	200.0	744.229437	268.492764	104	6
76	150.0	742.128461	281.576461	138	16
77	175.0	735.099082	293.004164	96	0
78	100.0	723.192705	274.092705	91	12
79	170.0	714.421503	259.766572	136	12
80	110.0	712.171856	281.287133	117	12
81	170.0	710.644566	208.545589	130	12
82	50.0	709.827462	296.623634	130	16
83	150.0	709.709780	319.246193	144	12
84	200.0	708.982323	202.853933	142	12
85	175.0	699.407853	307.407853	123	16
86	68.0	698.491347	300.531751	124	12
87	145.0	694.713380	209.397903	133	12
88	78.0	694.394724	408.010692	142	12
89	150.0	682.342377	195.042377	131	16
90	55.0	677.945399	330.252182	142	18
91	165.0	675.120017	188.020017	169	12
92	40.0	672.806292	293.506292	107	12
93	225.0	668.045518	291.045518	143	12
94	150.0	665.692281	165.249063	90	0
95	80.0	660.940780	195.330621	91	6
96	165.0	657.828828	222.527828	102	6
97	140.0	654.264015	305.413918	143	12
98	160.0	653.428261	281.723902	137	16
99	115.0	649.398328	310.676740	142	0

	Aspect ratio	IMDB score
0	1.78	7.9
1	2.35	7.7
2	2.35	7.1
3	2.00	7.0
4	1.85	8.1
5	2.35	7.2
6	2.35	7.5
7	2.39	8.1
8	2.24	7.6
9	2.35	7.2
10	1.85	6.4
11	2.35	8.2
12	2.35	6.3
13	2.35	8.9
14	2.35	7.8
15	2.35	5.7
16	2.35	8.5
17	1.85	8.3
18	2.35	7.3
19	2.35	6.7
20	1.85	8.1

```
21          2.35         6.5
22          1.85         6.5
23          2.39         8.0
24          2.35         7.9
25          2.35         9.0
26          2.35         7.5
27          1.85         7.5
28          1.66         8.5
29          1.85         7.8
..           ...         ...
70          2.35         6.7
71          2.35         6.4
72          2.35         8.0
73          1.85         6.9
74          2.35         6.9
75          1.85         7.3
76          2.35         7.2
77          1.85         8.3
78          2.35         7.8
79          2.35         7.8
80          2.35         4.9
81          1.85         7.6
82          2.35         4.6
83          2.35         7.1
84          2.35         6.7
85          2.35         6.9
86          2.35         4.9
87          2.35         7.4
88          2.35         7.3
89          2.35         7.4
90          2.35         8.8
91          2.35         8.6
92          1.85         8.1
93          2.35         7.2
94          2.35         7.3
95          1.85         6.9
96          2.39         7.9
97          2.35         8.1
98          2.35         6.6
99          1.78         6.7

[100 rows x 10 columns]
```

Homework 2

1. Choose 3-6 features, Explain the choice, Apply K-means at K=5, 9. In both cases: 10 or more random initializations, chose the best over the K-means criterion

I decided to choose features '**Worldwide gross**', '**Year**' and '**Certification**', because I think that age limit, gross and movie release time can separate a bunch of movies in different clusters.

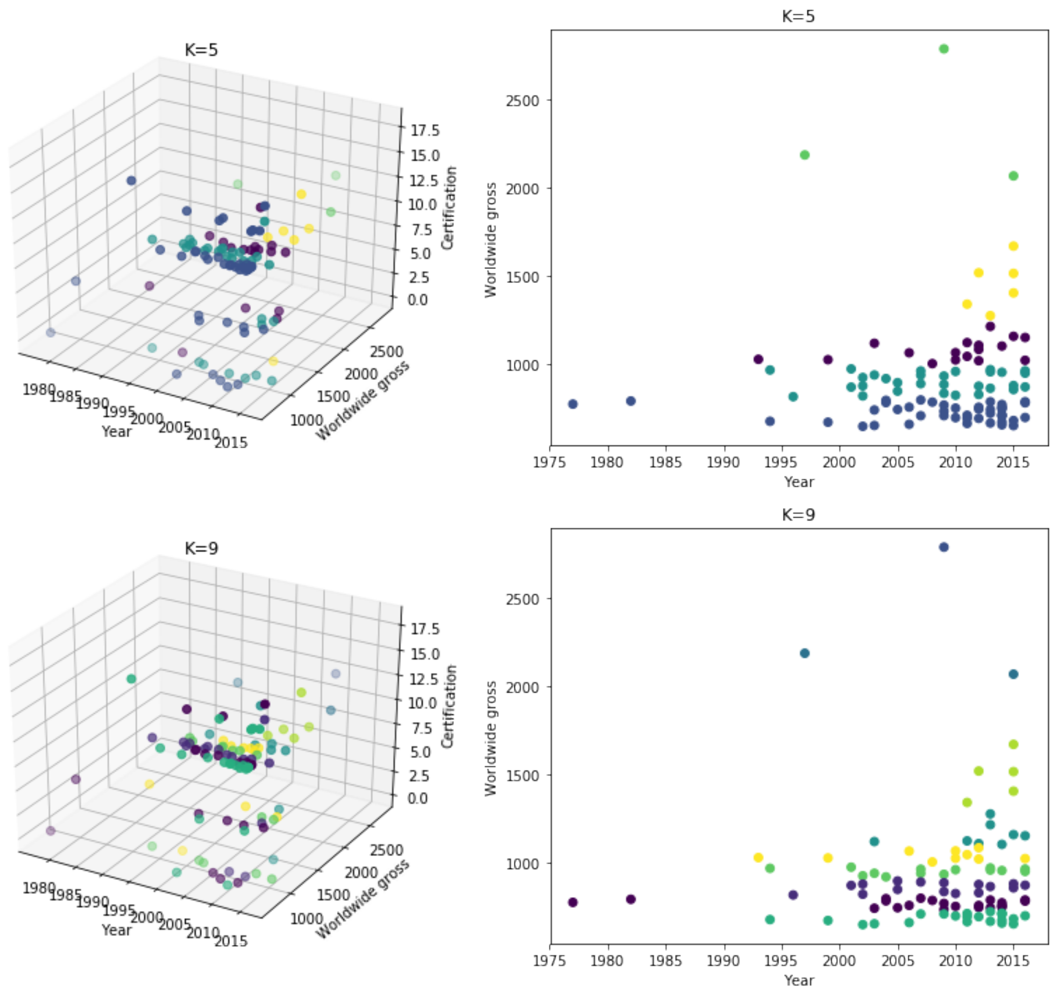
In [4]:
```
from sklearn.cluster import KMeans
```

In [5]:
```
X = data[['Year', 'Worldwide gross', 'Certification']]
y_pred = np.arange(200).reshape(2, 100)
for i, k in enumerate([5, 9]):
    kmeans = KMeans(n_clusters=k, random_state=42, init='random', n_init=10)
    y_pred[i] = kmeans.fit_predict(X)
```

In [6]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

In [7]:
```
fig = plt.figure(figsize=(14,12))

ax = fig.add_subplot(221, projection='3d')
ax.scatter(X.iloc[:, 0].values, X.iloc[:, 1].values,
           X.iloc[:, 2].values, c=y_pred[0], s=35)

ax.set_title('K=5')
ax.set_xlabel('Year')
ax.set_ylabel('Worldwide gross')
ax.set_zlabel('Certification')

ax = fig.add_subplot(222)
ax.scatter(X.iloc[:, 0].values, X.iloc[:, 1].values, c=y_pred[0], s=35)

ax.set_title('K=5')
ax.set_xlabel('Year')
ax.set_ylabel('Worldwide gross')

ax = fig.add_subplot(223, projection='3d')
ax.scatter(X.iloc[:, 0].values, X.iloc[:, 1].values,
           X.iloc[:, 2].values, c=y_pred[1], s=35)

ax.set_title('K=9')
ax.set_xlabel('Year')
ax.set_ylabel('Worldwide gross')
ax.set_zlabel('Certification')

ax = fig.add_subplot(224)
ax.scatter(X.iloc[:, 0].values, X.iloc[:, 1].values, c=y_pred[1], s=35)
```

```
ax.set_title('K=9')
ax.set_xlabel('Year')
ax.set_ylabel('Worldwide gross')

plt.show()
```



2. Interpret each found partition by using features from the data table. Explain why you consider one of them better than the other in this perspective.

Calculate relative difference of the feature '**Worldwide gross**' for each cluster (separately when k=5 and k=9):

```
In [8]: rel_dif = pd.DataFrame(np.zeros(5), columns=['Relative difference'],
                              index = ['Cluster ' + str(i) for i in range(1, 6)])
        worldwidegross_mean = data['Worldwide gross'].mean()
        for i in range(5):
```

```
            rel_dif.iloc[i] = 100 * (data[y_pred[0] == i]['Worldwide gross'].mean()
                                    / worldwidegross_mean - 1)
        rel_dif
```

```
Out[8]:            Relative difference
        Cluster 1            15.982789
        Cluster 2           -22.035713
        Cluster 3            -3.141526
        Cluster 4           151.862800
        Cluster 5            56.086892
```

```
In [9]: rel_dif = pd.DataFrame(np.zeros(9), columns=['Relative difference'],
                               index = ['Cluster ' + str(i) for i in range(1, 10)])
        for i in range(9):
            rel_dif.iloc[i] = 100 * (data[y_pred[1] == i]['Worldwide gross'].mean()
                                    / worldwidegross_mean - 1)
        rel_dif
```

```
Out[9]:            Relative difference
        Cluster 1           -17.676514
        Cluster 2            -7.752542
        Cluster 3           199.102558
        Cluster 4           128.242920
        Cluster 5            24.185809
        Cluster 6           -26.394913
        Cluster 7             2.128207
        Cluster 8            59.915223
        Cluster 9            11.516618
```

Comment: we can see that each cluster can be described by relative difference of the feature 'Worldwide gross' (for example, by using gradation from much smaller than the average to much then the average). Moreover, all cluster all linearly separable by this feature, but in case of K=5, values of relative difference is more unlike each other than in case of K=9.

3. Take one of the partitions

3.1. Compare one of the features between two clusters with using bootstrap I took **Worldwide gross** feature as a comparable one.

3.2. Take a feature, find the 95% confidence interval for its grand mean by using bootstrap

```
In [10]: mean_data = data['Worldwide gross'].mean()
         print('Mean of the feature (Worldwide gross):', mean_data)
```

```
Mean of the feature (Worldwide gross): 932.1100783399997
```

Generate samples of size 5000 using bootstrap:

```
In [11]: def generate_bootstrap_sample(data):
             len_data = len(data)
             r = np.random.randint(0, len_data, size=(len_data, 5000))

             gross = data['Worldwide gross'].tolist()

             xr = np.zeros(r.shape)
             for i in range(r.shape[0]):
                 for j in range(r.shape[1]):
                     xr[i, j] = gross[r[i, j]]
             return xr
```

```
In [12]: xr = generate_bootstrap_sample(data)
         xr_mean = xr.mean(axis=0)
         xr_std = xr.std(axis=0)
```

There are 2 methods to compute 95% confidence intervals for means:
1. Pivotal : Gaussian.

$$\mu \pm 1.96\sigma \tag{1}$$

```
In [13]: m1 = xr_mean.mean()
         std1 = xr_mean.std()
         CI1 = [m1 - 1.96 * std1, m1 + 1.96 * std1]
         print('95% confidence interval for mean of the feature:', CI1)
```

```
95% confidence interval for mean of the feature: [868.0032432359413, 995.5629261412829]
```

2. Non-pivotal:

Take 2.5% and 97.5% percentiles as the boundaries. To do this it is necessary to sort mean values and take 5000 $\times$ 0.025+1=126th and 5000 $\times$ 0.975=4875th values.

```
In [14]: smr = sorted(xr_mean)
         lbn = smr[125]
         rbn = smr[4874]
         print('95% confidence interval for mean of the feature:'\
               '[{}, {}]'.format(lbn, rbn))
```

```
95% confidence interval for mean of the feature:[872.1881515600003, 999.61421965]
```

3.3. Take a cluster, and compare the grand mean with the within-cluster mean for the feature by using bootstrap At first compute within-cluster mean for the feature '**Worlwide gross**' using bootstrap:

```
In [15]: yr = generate_bootstrap_sample(data[y_pred[0]==0])
         yr_mean = yr.mean(axis=0)
         yr_std = yr.std(axis=0)
```

Then compute 95% confidence interval for the difference:

```
In [16]: m2 = yr_mean.mean()
         std2 = yr_mean.std()
         m = m2 - m1
         std = np.sqrt(std1**2 + std2**2)
         CI = [m - 1.96 * std, m + 1.96 * std]
         print('95% confidence interval for the difference:', CI)
```

```
95% confidence interval for the difference: [80.05648487122136, 218.72172597703678]
```

As we can see, 0 is not in interval. So we can say with 95% confidence that the grand mean and the within-cluster mean are not equal.

Homework 3: Contingency Table

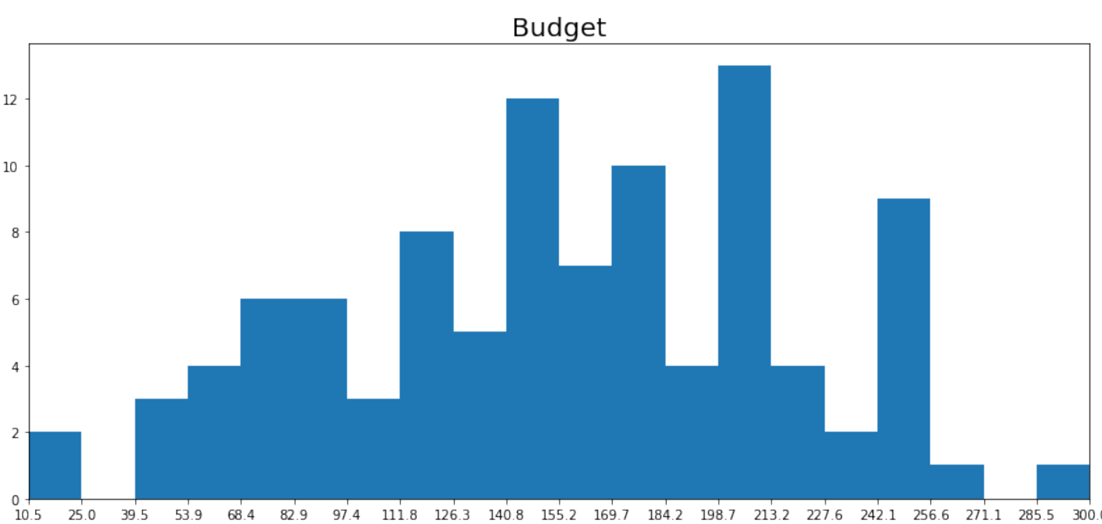
1 Consider three nominal features (one of them, not more, may be taken from nominal features in your data)

Take one nominal feature from our data (**Certification**) and develop others from features **Budget** and **Runtime**.

First of all, compute histograms and define points break in them to generate new nominal features.

```
In [17]: fig = plt.figure(figsize=(14, 6))

         axes = fig.add_axes([0.1, 0.1, 0.8, 0.8])
         axes.set_xlim([10.5, 300])
         axes.set_xticks(np.arange(10.500000, 300.1, 14.475))
         axes.hist(data['Budget'], bins = 20)
         axes.set_title('Budget', size=20)
```

```
Out[17]: Text(0.5,1,'Budget')
```

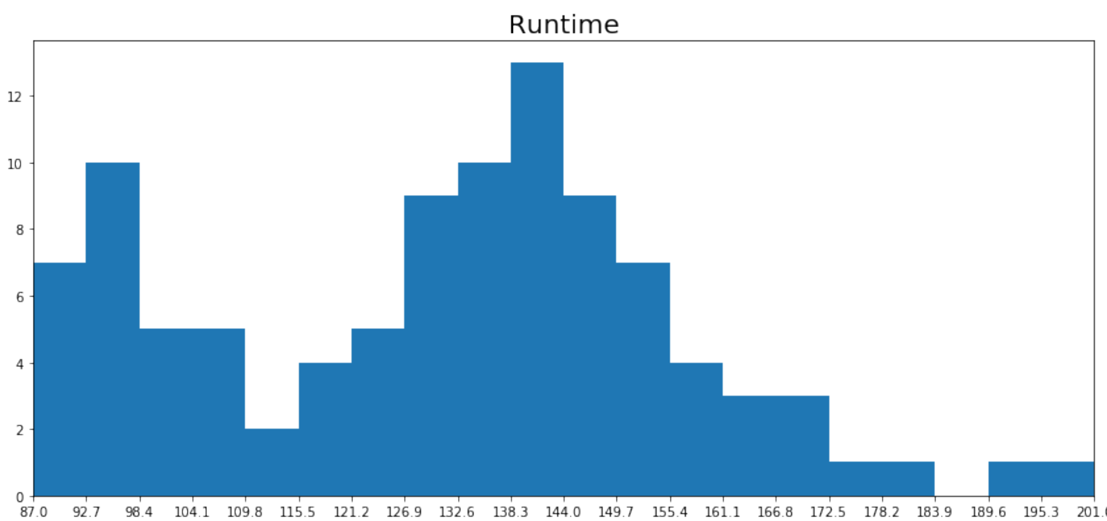


Let's split a feature 'Budget' into 3 groups and develop a new nominal one ('Budget_category'): 1. [10.5, 97.4) 2. [97.4, 184.2) 3. [184.2, 300.0]

```
In [18]: split = np.array([10.5, 97.4, 184.2, 300.1])
         g1 = []
         for i in range(split.shape[0] - 1):
             g1.append(data[(data['Budget'] >= split[i]) & (data['Budget'] < split[i+1])])
```

```
In [19]: fig = plt.figure(figsize=(14, 6))

         axes = fig.add_axes([0.1, 0.1, 0.8, 0.8])
         axes.set_xlim([87,201])
         axes.set_xticks(np.arange(87, 202, 5.7))
         axes.hist(data['Runtime'], bins = 20)
         axes.set_title('Runtime', size=20)
```

```
Out[19]: Text(0.5,1,'Runtime')
```



Let's split a feature 'Runtime' into 3 groups and develop a new nominal one ('Runtime_category'): 1. [87, 109.8) 2. [109.8, 161.1) 3. [161.1, 201]

```
In [20]: split = np.array([87, 109.8, 161.1, 202])
         g2 = []
         for i in range(split.shape[0] - 1):
             g2.append(data[(data['Runtime'] >= split[i]) & (data['Runtime'] < split[i+1])])
```

2 Build two contingency tables over them: present a conditional frequency table and Quetelet relative index tables. Make comments on relations between categories of the common (to both tables) feature and two others.

```
In [21]: row_list = []
         for age in np.unique(data['Certification']):
```

```
    d = {}
    for num, group in enumerate(g1):
        d['BC' + str(num+1)] = group[group['Certification'] == age].shape[0]
    d['total'] = d['BC1'] + d['BC2'] + d['BC3']
    row_list.append(d)

Budget_category = pd.DataFrame(row_list)
Budget_category_cf = pd.DataFrame(row_list)

total = Budget_category.sum(axis=0)
Budget_category.loc[5] = total

for i in range(1, 4, 1):
    Budget_category_cf['BC' + str(i)] = \
        Budget_category['BC' + str(i)].apply(lambda x: x/total[i-1])

Budget_category_cf.loc[5] = total

certfication = [0, 6, 12, 16, 18, 'total']
Budget_category.index = Budget_category_cf.index = certfication
Budget_category.index.name = Budget_category_cf.index.name = 'Certifiaction'
```

Contingency (Co-Occurrence) table for Certification and Budget_Category

In [22]:
```
Budget_category
```

Out[22]:

	BC1	BC2	BC3	total
Certifiaction				
0	6	8	1	15
6	5	6	3	14
12	7	27	26	60
16	1	4	4	9
18	2	0	0	2
total	21	45	34	100

Conditional frequency table for Certification and Budget_Category

In [23]:
```
Budget_category_cf
```

Out[23]:

	BC1	BC2	BC3	total
Certifiaction				
0	0.285714	0.177778	0.029412	15
6	0.238095	0.133333	0.088235	14
12	0.333333	0.600000	0.764706	60
16	0.047619	0.088889	0.117647	9
18	0.095238	0.000000	0.000000	2
total	21.000000	45.000000	34.000000	100

Quetelet relative index table

```
In [24]: def compute_quetlet_index(df):
             df = df.copy()
             df.iloc[:-1, :] /= df.iloc[-1]
             df.iloc[:-1, :-1] = (df.iloc[:-1, :-1].values - df.iloc[:-1, -1]\
                                                         .values.reshape(-1, 1)) /\
                                 df.iloc[:-1, -1].values.reshape(-1, 1)
             df = df.rename({'total': 'Prob.'}, axis='columns')
             return df
```

```
In [25]: Budget_category_q = compute_quetlet_index(Budget_category)
         Budget_category_q
```

```
Out[25]:                     BC1        BC2        BC3   Prob.
         Certifiaction
         0              0.904762   0.185185  -0.803922    0.15
         6              0.700680  -0.047619  -0.369748    0.14
         12            -0.444444   0.000000   0.274510    0.60
         16            -0.470899  -0.012346   0.307190    0.09
         18             3.761905  -1.000000  -1.000000    0.02
         total         21.000000  45.000000  34.000000  100.00
```

Comment: 18-years age limit movies, given BC1 (low-budget movies), is 376% more frequent than on average (while in the conditional frequency table value in this cell is the second lowest in the column).
0-years age limit movies, given BC2 (medium-budget movies), is 19% more frequent than on average (while in the conditional frequency table value in this cell is the second highest in the column).
16-years age limit movies, given BC3 (high-budget movies), is 31% more frequent than on average (while in the conditional frequency table value in this cell is the second highest in the column).
Compute **Summary Quetelet index**:

```
In [26]: np.multiply((Budget_category/100).iloc[:-1, :-1],
                         Budget_category_q.iloc[:-1, :-1]).sum().sum()
```

```
Out[26]: 0.2047300401642138
```

Comment: on average knowledge of Budget_Category ”adds” 20.5% to frequency of Certification.

```
In [27]: row_list = []
         for time in np.unique(data['Certification']):
             d = {}
             for num, group in enumerate(g2):
                 d['RC' + str(num+1)] = group[group['Certification'] == time].shape[0]
             d['total'] = d['RC1'] + d['RC2'] + d['RC3']
             row_list.append(d)

         Runtime_category = pd.DataFrame(row_list)
```

```
Runtime_category_cf = pd.DataFrame(row_list)

total = Runtime_category.sum(axis=0)
Runtime_category.loc[5] = total

for i in range(1, 4, 1):
    Runtime_category_cf['RC' + str(i)] = \
        Runtime_category['RC' + str(i)].apply(lambda x: x/total[i-1])

Runtime_category_cf.loc[5] = total

Runtime_category.index = Runtime_category_cf.index = certfication
Runtime_category.index.name = Runtime_category_cf.index.name = 'Certifiaction'
```

Contingency (Co-Occurrence) table for Certification and Runtime_Category

```
In [28]: Runtime_category
```

Out[28]:

Certifiaction	RC1	RC2	RC3	total
0	12	3	0	15
6	10	4	0	14
12	4	46	10	60
16	0	9	0	9
18	1	1	0	2
total	27	63	10	100

Conditional frequency table for Certification and Runtime_Category

```
In [29]: Runtime_category_cf
```

Out[29]:

Certifiaction	RC1	RC2	RC3	total
0	0.444444	0.047619	0.0	15
6	0.370370	0.063492	0.0	14
12	0.148148	0.730159	1.0	60
16	0.000000	0.142857	0.0	9
18	0.037037	0.015873	0.0	2
total	27.000000	63.000000	10.0	100

Quetelet relative index table

```
In [30]: Runtime_category_q = compute_quetlet_index(Runtime_category)
         Runtime_category_q
```

Out[30]:

Certifiaction	RC1	RC2	RC3	Prob.
0	1.962963	-0.682540	-1.000000	0.15
6	1.645503	-0.546485	-1.000000	0.14

```
12              -0.753086   0.216931   0.666667    0.60
16              -1.000000   0.587302  -1.000000    0.09
18               0.851852  -0.206349  -1.000000    0.02
total           27.000000  63.000000  10.000000  100.00
```

Comment: 0-years age limit movies, given RC1 (short-length movies), is 196% more frequent than on average (while in the conditional frequency table value in this cell is the highest in the column).
16-years age limit movies, given RC2 (medium-length movies), is 59% more frequent than on average (while in the conditional frequency table value in this cell is the second highest in the column).
12-years age limit movies, given RC3 (long-length movies), is 67% more frequent than on average (while in the conditional frequency table value in this cell is the highest in the column).

Compute **Summary Quetelet index**:

```
In [31]: np.multiply((Runtime_category/100).iloc[:-1, :-1],
                     Runtime_category_q.iloc[:-1, :-1]).sum().sum()
```

```
Out[31]: 0.5534139581758629
```

Comment: on average knowledge of Runtime_Category ”adds” 55.3% to frequency of Certification.

3 Compute and visualize the chi-square-summary_Quetelet_index over both tables. Comment on the meaning of the values in the data analysis context.

```
In [32]: def compute_indepent_frequency(df):
             df = df.copy() / df.iloc[-1, -1]
             df.iloc[:-1, :-1] = df['total'][:-1].values.reshape(-1, 1) @ \
             df.iloc[-1, :-1].values.reshape(1, -1)
             return df
```

```
In [33]: Budget_category_of = Budget_category.copy() / 100
         Budget_category_i = compute_indepent_frequency(Budget_category)
```

Observed frequencies $P(C_l \cap BC_l)$

```
In [34]: Budget_category_of
```

```
Out[34]:                 BC1   BC2   BC3  total
         Certifiaction
         0              0.06  0.08  0.01   0.15
         6              0.05  0.06  0.03   0.14
         12             0.07  0.27  0.26   0.60
         16             0.01  0.04  0.04   0.09
         18             0.02  0.00  0.00   0.02
         total          0.21  0.45  0.34   1.00
```

Frequencies expected under independence $P(C_l)P(BC_l)$

```
In [35]: Budget_category_i
```

```
Out[35]:                  BC1     BC2     BC3  total
         Certifiaction
         0             0.0315  0.0675  0.0510   0.15
         6             0.0294  0.0630  0.0476   0.14
         12            0.1260  0.2700  0.2040   0.60
         16            0.0189  0.0405  0.0306   0.09
         18            0.0042  0.0090  0.0068   0.02
         total         0.2100  0.4500  0.3400   1.00
```

Pearson's chi-squared

```
In [36]: ((Budget_category_of - Budget_category_i)**2 / \
          Budget_category_i).values[:-1,:-1].sum()
```

```
Out[36]: 0.20473004016421384
```

Comment: this value is equal to summary Quetlet index, as it said in lecture.

```
In [37]: def compute_r_table(df):
             a = df / df.iloc[-1, -1]
             b = compute_indepent_frequency(df)
             sl = (a.iloc[:-1, :-1] - b.iloc[:-1, :-1]) / np.sqrt(b.iloc[:-1, :-1])
             r_table = df.copy()
             r_table.iloc[:-1, :-1] = sl
             return r_table
```

```
In [38]: def compute_pq_table(df):
             a = df / df.iloc[-1, -1]
             b = a/compute_indepent_frequency(df) - 1
             sl = np.multiply(a.iloc[:-1, :-1], b.iloc[:-1, :-1])
             pq_table = df.copy()
             pq_table.iloc[:-1, :-1] = sl
             return pq_table
```

Visualize **r(k,l)** and **pq(k.l)** tables:

```
In [39]: compute_r_table(Budget_category)
```

```
Out[39]:                     BC1        BC2        BC3  total
         Certifiaction
         0              0.160579   0.048113  -0.181551     15
         6              0.120142  -0.011952  -0.080669     14
         12            -0.157762   0.000000   0.123986     60
         16            -0.064738  -0.002485   0.053736      9
         18             0.243799  -0.094868  -0.082462      2
         total         21.000000  45.000000  34.000000    100
```

```
In [40]: compute_pq_table(Budget_category)
```

```
Out[40]:                    BC1        BC2        BC3  total
         Certifiaction
         0             0.054286   0.014815  -0.008039     15
         6             0.035034  -0.002857  -0.011092     14
         12           -0.031111   0.000000   0.071373     60
         16           -0.004709  -0.000494   0.012288      9
         18            0.075238  -0.000000  -0.000000      2
         total        21.000000  45.000000  34.000000    100
```

Comment: Overall, the structures of signs are similar; the largest positions are more or less similar; the difference in the scales of values is due to the fact that r(k,l) are the square roots. These structures do not add any relation, because all values are too small.

```
In [41]: Runtime_category_of = Runtime_category.copy() / 100
         Runtime_category_i = compute_indepent_frequency(Runtime_category)
```

Observed frequencies $P(C_l \cap R_l)$

```
In [42]: Runtime_category_of
```

```
Out[42]:               RC1   RC2  RC3  total
         Certifiaction
         0             0.12  0.03  0.0   0.15
         6             0.10  0.04  0.0   0.14
         12            0.04  0.46  0.1   0.60
         16            0.00  0.09  0.0   0.09
         18            0.01  0.01  0.0   0.02
         total         0.27  0.63  0.1   1.00
```

Frequencies expected under independence $P(C_l)P(R_l)$

```
In [43]: Runtime_category_i
```

```
Out[43]:                 RC1     RC2    RC3  total
         Certifiaction
         0             0.0405  0.0945  0.015   0.15
         6             0.0378  0.0882  0.014   0.14
         12            0.1620  0.3780  0.060   0.60
         16            0.0243  0.0567  0.009   0.09
         18            0.0054  0.0126  0.002   0.02
         total         0.2700  0.6300  0.100   1.00
```

Pearson's chi-squared

```
In [45]: ((Runtime_category_of - Runtime_category_i)**2 / \
          Runtime_category_i).values[:-1,:-1].sum()
```

```
Out[45]: 0.5534139581758629
```

Comment: this value is equal to summary Quetlet index, as it said in lecture.
Visualize **r(k,l)** and **pq(k.l)** tables:

In [46]: compute_r_table(Runtime_category)

Out[46]:

	RC1	RC2	RC3	total
Certifiaction				
0	0.395039	-0.209819	-0.122474	15
6	0.319922	-0.162298	-0.118322	14
12	-0.303111	0.133373	0.163299	60
16	-0.155885	0.139847	-0.094868	9
18	0.062598	-0.023163	-0.044721	2
total	27.000000	63.000000	10.000000	100

In [47]: compute_pq_table(Runtime_category)

Out[47]:

	RC1	RC2	RC3	total
Certifiaction				
0	0.235556	-0.020476	-0.000000	15
6	0.164550	-0.021859	-0.000000	14
12	-0.030123	0.099788	0.066667	60
16	-0.000000	0.052857	-0.000000	9
18	0.008519	-0.002063	-0.000000	2
total	27.000000	63.000000	10.000000	100

Comment: Overall, the structures of signs are similar; the largest positions are more or less similar; the difference in the scales of values is due to the fact that r(k,l) are the square roots. These structures add relation RC1 ⇒ 0.

4 Tell what numbers of observations would suffice to see the features as associated at 95% confidence level; 99% confidence level.

Pearson: Under the hypothesis that the features are independent in the population, and entity sampling has been done randomly and independently, the density function of random variable NX^2 tends to distribution χ^2 with $f = (K-1)(L-1)$ degrees of freedom.
We have $K = 5, L = 3$, therefore $f = 8$. At $f = 8$, there is a 5% chance that the NX^2 value will be greater than 15.51 if the hypothesis of independence is true. So, we have to find such N that NX^2 will be greater than 15.51.
The value of X^2 of the first table is roughly equal to 0.2.

$$0.2N > 15.51$$

$$N > \frac{15.51}{0.2} = 77.55$$

Then, $N = 78$ observation will suffice to see the features of the first table are associated at 95% confidence level.
At $f = 8$, there is a 1% chance that the NX^2 value will be greater than 20.09 if the hypothesis of independence is true.

$$0.2N > 20.09$$

$$N > \frac{20.09}{0.2} = 100.45$$

Then, $N = 101$ observation will suffice to see the features of the first table are associated at 95% confidence level.
The value of X^2 of the second table is roughly equal to 0.55.

$$0.55N > 15.51$$

$$N > \frac{15.51}{0.55} = 28.2$$

Then, $N = 79$ observation will suffice to see the features of the second table are associated at 95% confidence level.

$$0.55N > 20.09$$

$$N > \frac{20.09}{0.55} = 36.52$$

Then, $N = 37$ observation will suffice to see the features of the second table are associated at 95% confidence level.

Homework 4: PCA/SVD

1. In your data set, select a subset of 3-6 features related to the same aspect and explain your choice

I selected **Budget**, **Worldwide gross**, **Gross in USA** features, because they are all related to quantity of money. Moreover, movie's gross often depends on the budget.

```
In [48]: X = data[['Budget', 'Worldwide gross', 'Gross in USA']]
         X.head()
```

Out[48]:

	Budget	Worldwide gross	Gross in USA
0	237.0	2787.965087	760.507625
1	200.0	2186.772302	658.672302
2	245.0	2068.178225	936.662225
3	150.0	1670.400637	652.270625
4	220.0	1519.557910	623.357910

2. Standardize the selected subset; compute its data scatter and determine contributions of all the principal components to the data scatter, naturally and per cent

```
In [49]: from sklearn.decomposition import PCA
         from sklearn.preprocessing import LabelEncoder
```

```
In [50]: X_centered = X - X.mean(axis=0)
         data_scatter = np.square(X_centered).sum().sum()
         print('Data scatter:', data_scatter)
         _, mu, _ = np.linalg.svd(X_centered)
         contribution = mu**2/data_scatter
         print('Contribution of the first PC: {:.5f} ({:.2%})'.format(contribution[0],
                                                                       contribution[0]))
         print('Contribution of the second PC: {:.5f} ({:.2%})'.format(contribution[1],
                                                                        contribution[1]))
         print('Contribution of the third PC: {:.5f} ({:.2%})'.format(contribution[2],
                                                                       contribution[2]))
```

```
Data scatter: 12375933.627273217
Contribution of the first PC: 0.92864 (92.86%)
Contribution of the second PC: 0.04755 (4.76%)
Contribution of the third PC: 0.02381 (2.38%)
```

Comment: the first principal component takes into account mainly movie's worldwide gross and partly gross in USA. So, arguably it can be movie's gross. The second principal component, takes into account only gross in USA while other features are negative. So, it can be a movie revenue in USA.

3. Visualize the data with these features using standardization with two versions of normalization: (a) over ranges and (b) over standard deviations. At these visualizations, use a distinct shape/colour for points representing a pre-specified by you group of objects. Also, apply the conventional PCA for the visualization and see if there is any difference. Comment on which of the normalizations is better and why.

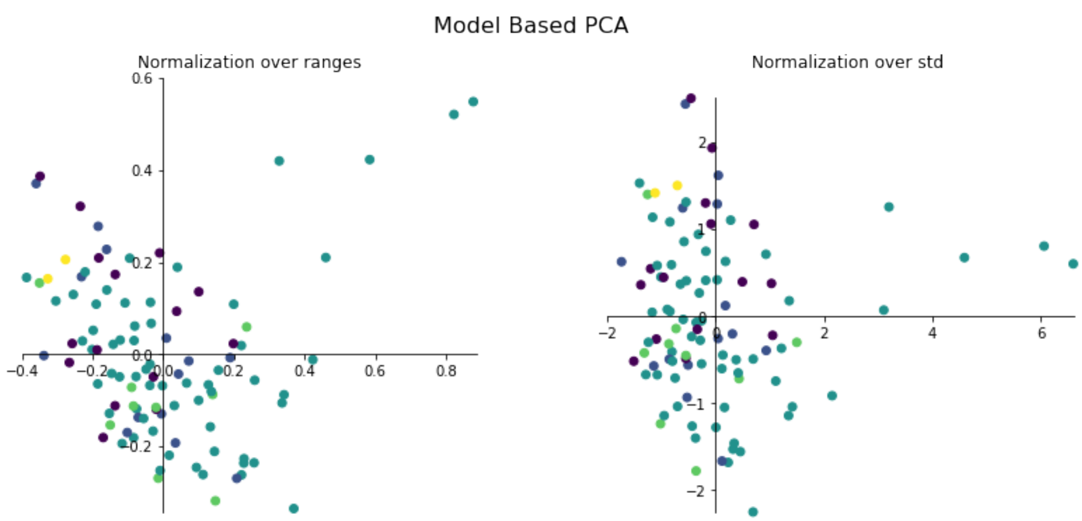
I decided to divide all objects into groups by age limit (or in other words use a feature 'Certification' as a target)

```
In [51]: y = LabelEncoder().fit_transform(data['Certification'])
```

Now normalize features in two ways and vizualize the data with model-based PCA:

```
In [52]: X_range = X_centered / (X_centered.max() - X_centered.min())
         pca = PCA(n_components=2)
         comps = pca.fit_transform(X_range)
         z1, z2 = list(zip(*comps))

         X_std = X_centered / X_centered.std()
         pca = PCA(n_components=2)
         comps = pca.fit_transform(X_std)
         z3, z4 = list(zip(*comps))
```

```
In [53]: fig = plt.figure(figsize=(14,6))
         fig.suptitle('Model Based PCA', size=16)

         ax = fig.add_subplot(121)
         ax.scatter(z1, z2, c= y, s=35)

         ax.set_title('Normalization over ranges')
         ax.spines['left'].set_position('zero')
         ax.spines['right'].set_color('none')
         ax.spines['bottom'].set_position('zero')
         ax.spines['top'].set_color('none')
         ax.spines['left'].set_smart_bounds(True)
         ax.spines['bottom'].set_smart_bounds(True)
         ax.xaxis.set_ticks_position('bottom')
         ax.yaxis.set_ticks_position('left')

         ax = fig.add_subplot(122)
         ax.scatter(z3, z4, c=y, s=35)

         ax.set_title('Normalization over std')
         ax.spines['left'].set_position('zero')
         ax.spines['right'].set_color('none')
         ax.spines['bottom'].set_position('zero')
         ax.spines['top'].set_color('none')
         ax.spines['left'].set_smart_bounds(True)
         ax.spines['bottom'].set_smart_bounds(True)
         ax.xaxis.set_ticks_position('bottom')
         ax.yaxis.set_ticks_position('left')

         plt.show()
```

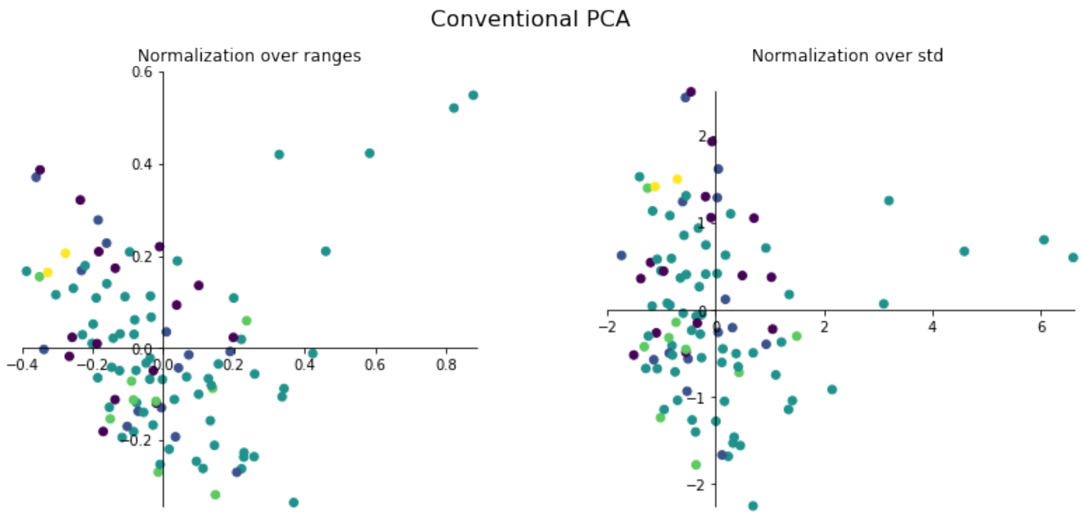


Vizualize the data with conventional PCA:

```
In [54]: def conventional_pca(X_centered):
             Z, LA, _ = np.linalg.svd(X_centered)
             z1 = -(Z[:, 0] * LA[0]).reshape(X_centered.shape[0], -1)
             z2 = (Z[:, 1] * LA[1]).reshape(X_centered.shape[0], -1)
             return np.hstack((z1, z2))
```

```
In [55]: comps = conventional_pca(X_range)
         z1, z2 = list(zip(*comps))

         comps = conventional_pca(X_std)
         z3, z4 = list(zip(*comps))
```

```
In [56]: fig = plt.figure(figsize=(14,6))
         fig.suptitle('Conventional PCA', size=16)

         ax = fig.add_subplot(121)
         ax.scatter(z1, z2, c=y, s=35)

         ax.set_title('Normalization over ranges')
         ax.spines['left'].set_position('zero')
         ax.spines['right'].set_color('none')
         ax.spines['bottom'].set_position('zero')
         ax.spines['top'].set_color('none')
         ax.spines['left'].set_smart_bounds(True)
         ax.spines['bottom'].set_smart_bounds(True)
         ax.xaxis.set_ticks_position('bottom')
         ax.yaxis.set_ticks_position('left')

         ax = fig.add_subplot(122)
         ax.scatter(z3, z4, c=y, s=35)

         ax.set_title('Normalization over std')
         ax.spines['left'].set_position('zero')
         ax.spines['right'].set_color('none')
         ax.spines['bottom'].set_position('zero')
         ax.spines['top'].set_color('none')
         ax.spines['left'].set_smart_bounds(True)
         ax.spines['bottom'].set_smart_bounds(True)
         ax.xaxis.set_ticks_position('bottom')
         ax.yaxis.set_ticks_position('left')

         plt.show()
```



Comment: there is no difference in PCA approaches. I think, that in this case there is no the best method of normalization, because both scatter looks pretty the same, but one is rotated; groups are mixed and no method divide them better than another.

4. Compute and interpret a hidden factor behind the selected features. The factor should be expressed in a 0-100 rank scale (as well as the features).

```
In [57]: X_rescaled = X / X.max(axis=0) * 100
         Z, Mu, C = np.linalg.svd(X_rescaled)
         z = -Z[:,0].reshape(-1, 1)
         c = -C[0,:]
         alpha = 100 / (100*c).sum()
         hidden_factor_vector = c * alpha
         print('Hidden factor: {:.3f}*Budget + {:.3f}*Worldwide ' \
               'gross + {:.3f}*Gross in USA'.format(*hidden_factor_vector))
         print('Contribution to the data scatter: ' \
               '{:.2%}'.format(Mu[0]*Mu[0] / np.square(X_rescaled).sum().sum()))
```

```
Hidden factor: 0.434*Budget + 0.274*Worldwide gross + 0.292*Gross in USA
Contribution to the data scatter: 94.87%
```

Comment: weight of budget is the highest. Arguably, this is because mostly the higher the budget, the higher the gross. Furthermore, the contribution of the hidden factor score is pretty big.

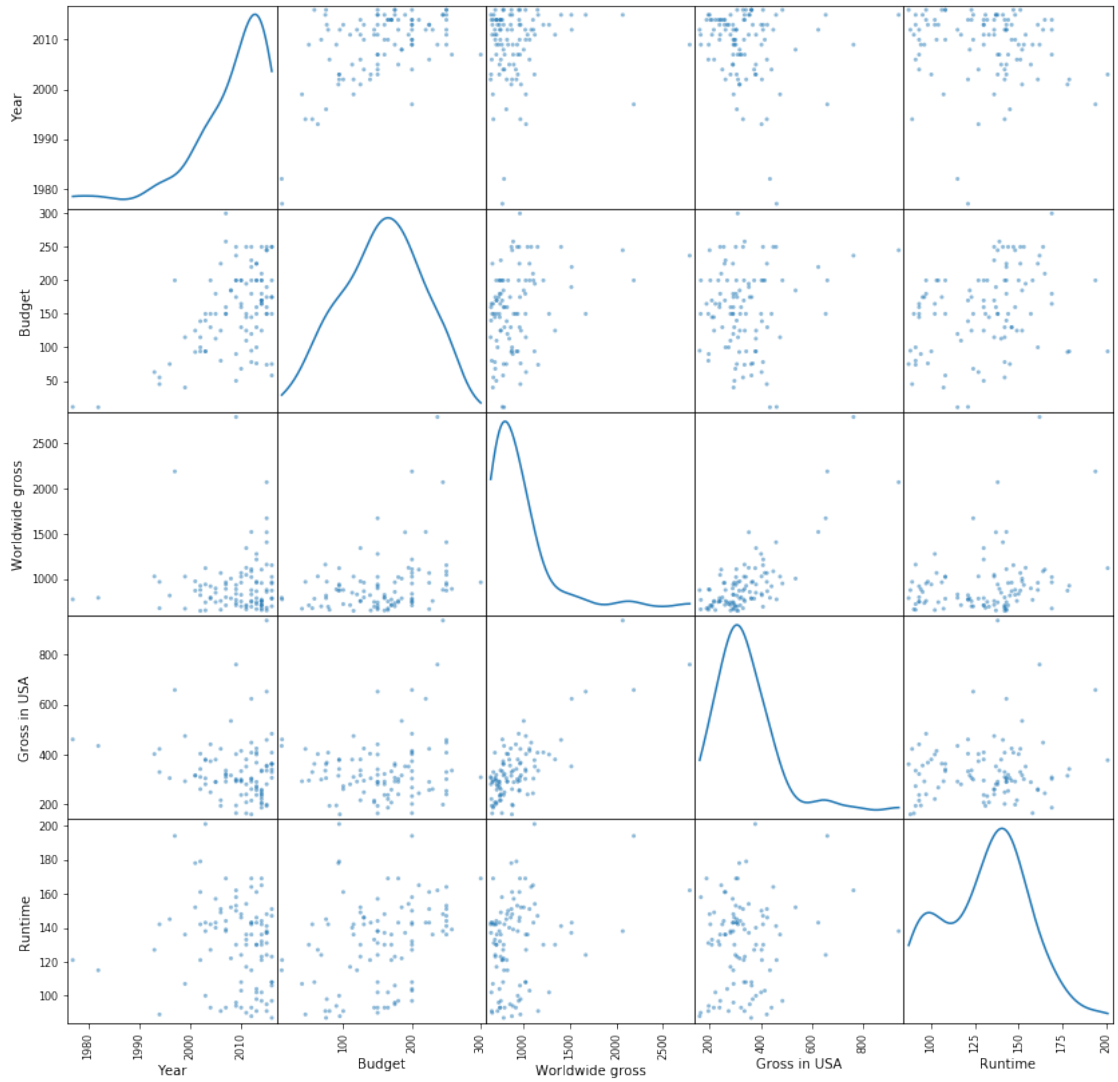
Homework 5

1 Find two features in your dataset with more or less "linear-like" scatterplot.

```
In [58]: from pandas.plotting import scatter_matrix
         scatter_matrix(data[['Year', 'Budget', 'Worldwide gross',
```

```
                 'Gross in USA', 'Runtime']],
                 alpha=0.5, figsize=(14, 14), diagonal='kde')
```

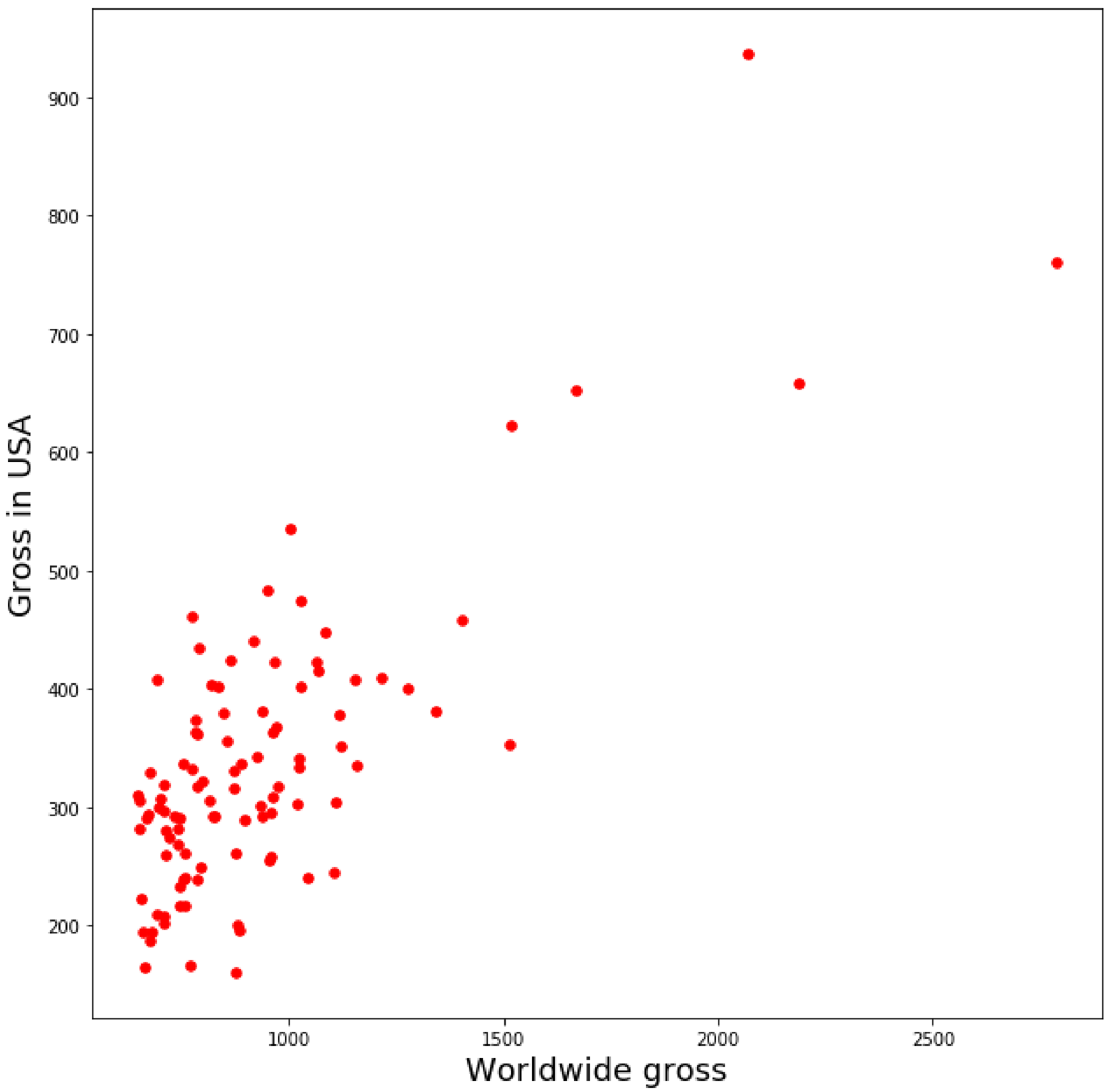
```
Out[58]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f9e03627978>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e019a7128>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e019cb7b8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e01973e48>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e01922518>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x7f9e01922550>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e018fb240>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e018a48d0>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e018ccf60>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e0187d5f8>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x7f9e01824c88>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e017d6358>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e017fc9e8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e017ae0b8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e01756748>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x7f9e0177ddd8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e017304a8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e016d7b38>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e0170a208>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e016b2898>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x7f9e0165af28>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e0168c5f8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e0064dc88>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e00602358>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f9e005ab9e8>]],
               dtype=object)
```



Comment: As we can see, features 'Worldwide gross' and 'Gross in USA' are more "linear-like' than each others.

2 Display the scatter-plot.

```
In [59]: fig = plt.figure(figsize=(10, 10))

         axes = fig.add_axes([0.1, 0.1, 0.8, 0.8])

         axes.scatter(data['Worldwide gross'], data['Gross in USA'], color='red', s=30)

         axes.set_ylabel('Gross in USA', fontsize = 18)
         axes.set_xlabel('Worldwide gross', fontsize = 18)

Out[59]: Text(0.5,0,'Worldwide gross')
```



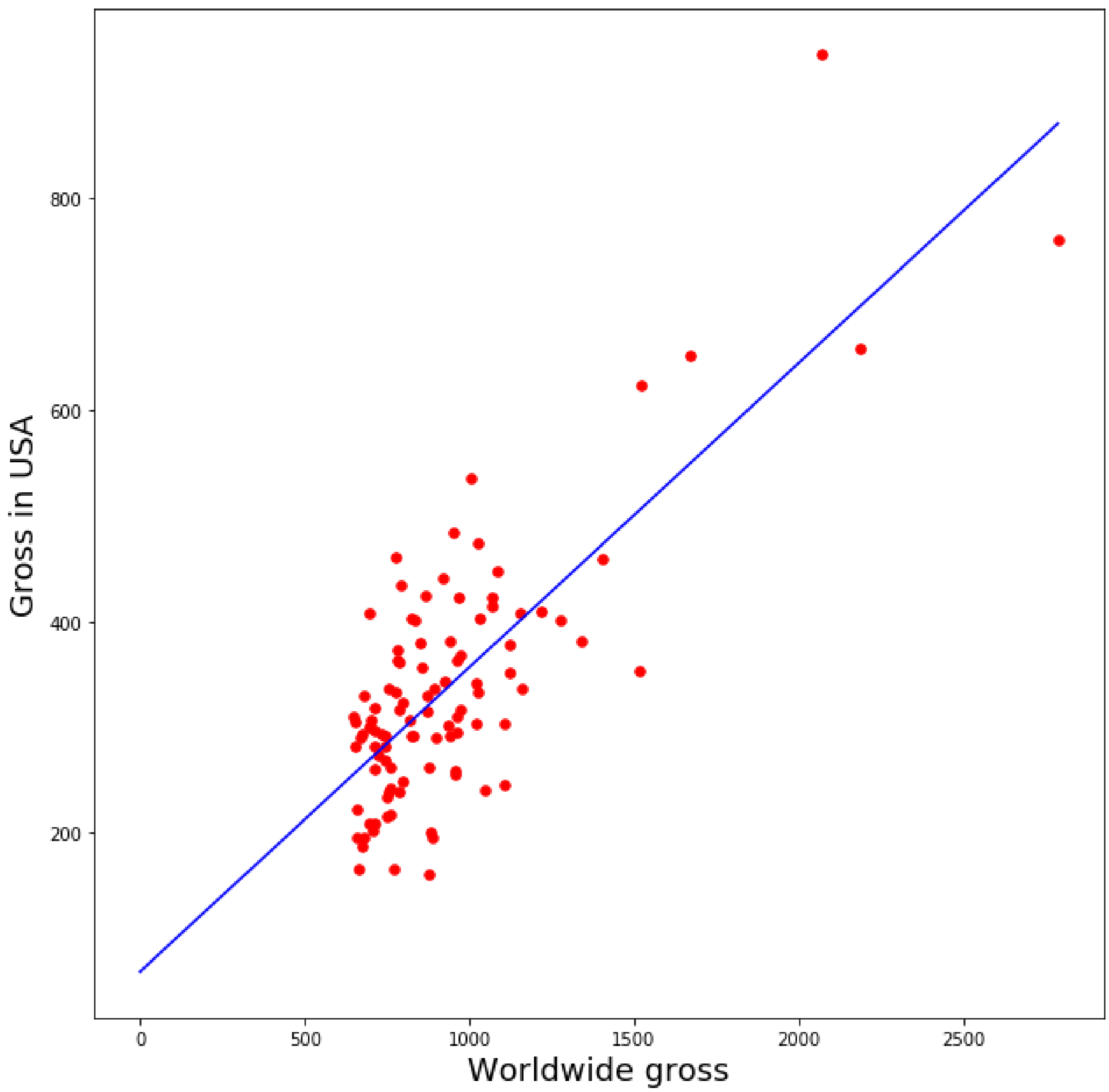
3 Build a linear regression of one of the features over the other. Make a comment on the meaning of the slope.

```
In [60]: y = data['Gross in USA']
         x = data['Worldwide gross']
         a = ((y.mean() * x.mean()) - (x*y).mean())/((x.mean() ** 2) - (x**2).mean())
         b = ((x ** 2).mean() * y.mean() - x.mean() *
              (x*y).mean())/(-(x.mean() ** 2) + (x**2).mean())
         print('a = ', a, '\nb = ', b)
```

```
a =  0.28780482278649333
b =  68.84193724585181
```

In [61]:
```
fig = plt.figure(figsize=(10, 10))

axes = fig.add_axes([0.1, 0.1, 0.8, 0.8])

axes.scatter(data['Worldwide gross'], data['Gross in USA'], color='red', s=30)
x = np.arange(0, data['Worldwide gross'].max())
axes.plot(x, a*x+b, 'b')

axes.set_ylabel('Gross in USA', fontsize = 18)
axes.set_xlabel('Worldwide gross', fontsize = 18)
```

Out[61]:
```
Text(0.5,0,'Worldwide gross')
```



Comment: slope of a linear regression means change in the Gross in USA at the Worldwide gross changed by 1. Its value is less than 1, so change of the value of Gross in USA is less than the value of Worldwide gross changed by 1.

4 Find the correlation and determinacy coefficients, and comment on the meaning of the latter.

In [62]:
```
x = data['Worldwide gross']
y = data['Gross in USA']
corr_coef = np.corrcoef(x, y)[0, 1]

print("Correlation coefficient", corr_coef)
print("Coefficient of determination:", corr_coef**2)
```

```
Correlation coefficient 0.7716312493787117
Coefficient of determination: 0.5954147850177517
```

Comment: in regression, the coefficient of determination is a statistical measure of how well the regression line approximates the real data points. An coefficient of 1 indicates that the regression line perfectly fits the data. In our case, the coefficient of determination is approximately equal to 0.595. This means that our model describes well the dependence, but not perfect. This can be explained by the large spread of values relative to the line.

• Make a prediction of the target values for given two or three predictor′ values; make a comment

Take 3 random values and make predictions for them:

In [63]:
```
ix1, ix2, ix3 = np.random.choice(100, 3, replace=False)
```

In [64]:
```
x1 = data['Worldwide gross'][ix1]
y1_true = data['Gross in USA'][ix1]
x2 = data['Worldwide gross'][ix2]
y2_true = data['Gross in USA'][ix2]
x3 = data['Worldwide gross'][ix3]
y3_true = data['Gross in USA'][ix3]
y1_pred = a*x1+b
y2_pred = a*x2+b
y3_pred = a*x3+b
print('Target value: {:0.3f}; Predicted value: {:0.3f}'.format(y1_true,
                                                                y1_pred))
print('Target value: {:0.3f}; Predicted value: {:0.3f}'.format(y2_true,
                                                                y2_pred))
print('Target value: {:0.3f}; Predicted value: {:0.3f}'.format(y3_true,
                                                                y3_pred))
```

```
Target value: 296.624; Predicted value: 273.134
Target value: 303.004; Predicted value: 362.720
Target value: 295.984; Predicted value: 345.216
```

Comment: predicted and target values are pretty similar and the higher the the target value the more closer to it the predicted value.

• Compare the mean relative absolute error of the regression on all points of your set and the determinacy coefficient and make comments.

```
In [65]: def relative_absolute_error(y_true, y_pred):
             return np.mean(np.abs(y_true-y_pred) / np.abs(y_true))
```

```
In [66]: x = data['Worldwide gross']
         y = data['Gross in USA']
         print('Mean relative absolute error:', relative_absolute_error(y, a*x+b))
         print("Coefficient of determination:", corr_coef**2)
```

```
Mean relative absolute error: 0.19624049818504846
Coefficient of determination: 0.5954147850177517
```

Comment: as we can see, the determinacy coefficient is bigger than the mean relative absolute error. The value of the latter is small, so the difference between predited and target values are not high too. The value of determinacy coefficient is also shows that the regression line well fits the data.