IMDB 5000 metadata

Jakobe Iversen
s143262

Mattias Andersen
s154057

October 3, 2017

1 IMDB 5000 movie metadata

We will start by loading the dataset into a pandas dataframe, and inspect the attributes of the first entry

```
In [1]: import pandas as pd
        import seaborn as sns
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.decomposition import PCA # Principal component analysis
        from sklearn.preprocessing import StandardScaler # to standardize the data

        df = {}
        df['raw'] = pd.read_csv("./data/movie_metadata.csv")
```

2 1. A description of the data set. (Jakob)

2.0.1 Problem of interest

The data consists of 28 attributes, regarded as meta data of movies.

2.0.2 Where the dataset was obtained

The dataset was provided by kaggle.com After obtaining the dataset, it has been removed due to a DMCA complaint, and replaced with an alternative dataset. This project is not compatible with the new dataset.

2.0.3 What have previously been done to the data

The data has been scraped from the IMDB web site, using a python script. The data has not been pre-proccesed, which means we will expect some NaN values.

2.0.4 Aim and relevant attributes

Our aim is to do a classification, to predict what score a movie will get on IMDB. We also want to do a linear regresion of the gross of a movie. To consider which attributes are relevant, we will print out the first movie, and look at what attributes will be relevant.

```
In [2]: print(df['raw'].iloc[0])

color                                                                     Color
director_name                                                     James Cameron
num_critic_for_reviews                                                      723
duration                                                                    178
director_facebook_likes                                                       0
actor_3_facebook_likes                                                      855
actor_2_name                                                   Joel David Moore
actor_1_facebook_likes                                                     1000
gross                                                               7.60506e+08
genres                                          Action|Adventure|Fantasy|Sci-Fi
actor_1_name                                                        CCH Pounder
movie_title                                                             Avatar 
num_voted_users                                                          886204
cast_total_facebook_likes                                                  4834
actor_3_name                                                          Wes Studi
facenumber_in_poster                                                          0
plot_keywords                               avatar|future|marine|native|paraplegic
movie_imdb_link               http://www.imdb.com/title/tt0499549/?ref_=fn_t...
num_user_for_reviews                                                       3054
language                                                                English
country                                                                     USA
content_rating                                                            PG-13
budget                                                                 2.37e+08
title_year                                                                 2009
actor_2_facebook_likes                                                      936
imdb_score                                                                  7.9
aspect_ratio                                                               1.78
movie_facebook_likes                                                      33000
Name: 0, dtype: object
```

From this data, we will chose to focus on numerical data. We do this, partly because we feel some of the categorical data is better explained by the numerical data, I.E. wether or not the instructor is James cameron or not, might not be as relevant as measuring the popularity of a director through facebook likes. This might not be true for categories like genre, but we hope to build a more simple model of prediction, by limiting the prediction to numerical data.

3 2. Detailed explanation of the attributes(Mattias)

This dataset consists of 28 different attributes and they together hold information about a movie.

3.1 Attribute description

A detailed explination of the attributes is shown in the table below. Where each attribute can be discrete or continous, and each attributes' objects are of different types.

Attribute description

Attribute	**Description**	**Discrete/Continous**	**Type of attribute**
movie_title	Holds title of the movie.	Discrete	Nominal
director_name	Name of director of the movie.	Discrete	Nominal
color	Shown in color or black and white.	Discrete	Nominal
duration	Duration of the movie in minutes.	Discrete	Rati
actor_1_name	Name of lead actor.	Discrete	Nominal
actor_2_name	Name of second actor.	Discrete	Nominal
actor_3_name	Name of third actor.	Discrete	Nominal
title_year	Year of release.	Discrete	Interval
genres	Genres the movie belongs to.	Discrete	Nominal
aspect_ratio	Aspect ratio	Discrete	Nominal
facenumber_in_poster	Number of faces shown in movie poster.	Discrete	Ratio
language	Language spoken in the movie.	Discrete	Nominal
country	Country where the movie is filmed.	Discrete	Nominal
budget	Cost of the movie.	Continous	Ratio
gross	Income of the movie.	Continous	Ratio
movie_facebook_likes	Count of facebook likes for the movie.	Discrete	Ratio
director_facebook_likes	Count of facebook likes the director has.	Discrete	Ratio
actor_1_facebook_likes	Facebook likes actor 1 has.	Discrete	Ratio
actor_2_facebook_likes	Facebook likes actor 2 has.	Discrete	Ratio
actor_3_facebook_likes	Facebook likes actor 3 has.	Discrete	Ratio
cast_total_facebook_likes	Total facebook likes for the whole cast of the movie.	Discrete	Ratio
plot_keywords	Keywords describing the movie.	Discrete	Nominal
content_rating	Rating of the movie.	Discrete	Nominal
num_user_for_reviews	Number of users who wrote reviews.	Discrete	Ratio
num_critic_for_reviews	Number of critics who wrote reviews.	Discrete	Ratio
num_voted_users	Count of users who have voted the movie.	Discrete	Ratio
movie_imdb_link	Holds a link to the movie on the site imdb.	Discrete	Nominal
imdb_scoreMovie	score on IMDB	Continous	Ordinal

3.2 Summary statistics

A summary over the different numerical attributes of the dataset.

```
In [21]: df['raw'].describe()
```

```
Out[21]:        num_critic_for_reviews     duration  director_facebook_likes  \
         count             4993.000000  5028.000000              4939.000000
         mean               140.194272   107.201074               686.509212
         std                121.601675    25.197441              2813.328607
         min                  1.000000     7.000000                 0.000000
         25%                 50.000000    93.000000                 7.000000
         50%                110.000000   103.000000                49.000000
         75%                195.000000   118.000000               194.500000
         max                813.000000   511.000000             23000.000000

                actor_3_facebook_likes  actor_1_facebook_likes         gross  \
         count             5020.000000             5036.000000  4.159000e+03
         mean               645.009761             6560.047061  4.846841e+07
         std               1665.041728            15020.759120  6.845299e+07
         min                  0.000000                0.000000  1.620000e+02
         25%                133.000000              614.000000  5.340988e+06
         50%                371.500000              988.000000  2.551750e+07
         75%                636.000000            11000.000000  6.230944e+07
         max              23000.000000           640000.000000  7.605058e+08

                num_voted_users  cast_total_facebook_likes  facenumber_in_poster  \
         count     5.043000e+03                5043.000000           5030.000000
         mean      8.366816e+04                9699.063851              1.371173
         std       1.384853e+05               18163.799124              2.013576
         min       5.000000e+00                   0.000000              0.000000
         25%       8.593500e+03                1411.000000              0.000000
         50%       3.435900e+04                3090.000000              1.000000
         75%       9.630900e+04               13756.500000              2.000000
         max       1.689764e+06              656730.000000             43.000000

                num_user_for_reviews        budget   title_year  \
         count           5022.000000  4.551000e+03  4935.000000
         mean             272.770808  3.975262e+07  2002.470517
         std              377.982886  2.061149e+08    12.474599
         min                1.000000  2.180000e+02  1916.000000
         25%               65.000000  6.000000e+06  1999.000000
         50%              156.000000  2.000000e+07  2005.000000
         75%              326.000000  4.500000e+07  2011.000000
         max             5060.000000  1.221550e+10  2016.000000

                actor_2_facebook_likes    imdb_score  aspect_ratio  movie_facebook_likes
         count             5030.000000   5043.000000   4714.000000           5043.000000
         mean              1651.754473      6.442138      2.220403           7525.964505
```

std	4042.438863	1.125116	1.385113	19320.445110
min	0.000000	1.600000	1.180000	0.000000
25%	281.000000	5.800000	1.850000	0.000000
50%	595.000000	6.600000	2.350000	166.000000
75%	918.000000	7.200000	2.350000	3000.000000
max	137000.000000	9.500000	16.000000	349000.000000

4 3. data visualization(Jakob) and PCA(Mattias)

Before we start, we will take alle the numeric data of the dataset, and drop the lines with NA. We drop the lines, because it is assumed the web scrapper made an error while scraping for the movie.

```
In [23]: df['numeric'] = df['raw']._get_numeric_data()
         df['numeric'] = df['numeric'].dropna()
         df['numeric_std'] = (df['numeric'] - df['numeric'].mean())/df['numeric'].std()
         print(100-(df['raw'].shape[0]-df['numeric'].shape[0])/df['raw'].shape[0],"% of the data
         list(df['numeric'])
```

```
99.75371802498513 % of the dataset remain, after dropping NA's.
A list of the remaining attributes are shown below
```

```
Out[23]: ['num_critic_for_reviews',
          'duration',
          'director_facebook_likes',
          'actor_3_facebook_likes',
          'actor_1_facebook_likes',
          'gross',
          'num_voted_users',
          'cast_total_facebook_likes',
          'facenumber_in_poster',
          'num_user_for_reviews',
          'budget',
          'title_year',
          'actor_2_facebook_likes',
          'imdb_score',
          'aspect_ratio',
          'movie_facebook_likes']
```

The list printout of the dataset only with the numerical data, shows that the analysation of the data will only include these 16 attributes.

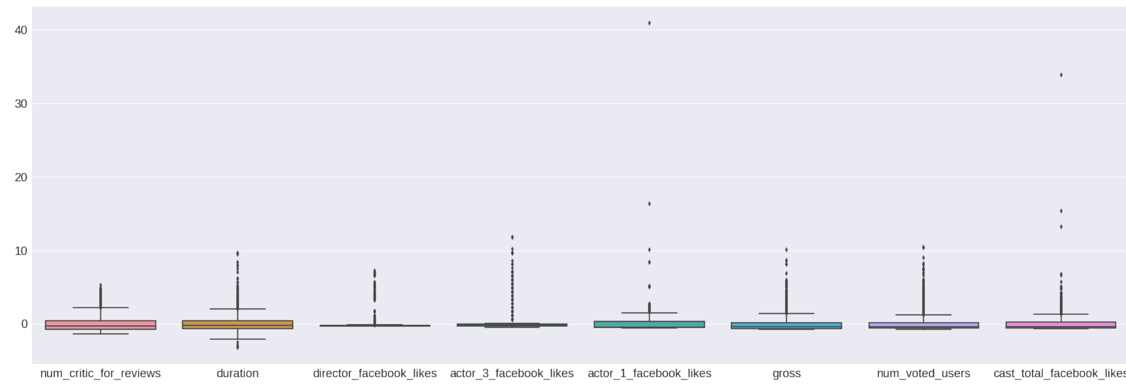
4.1 Boxplot

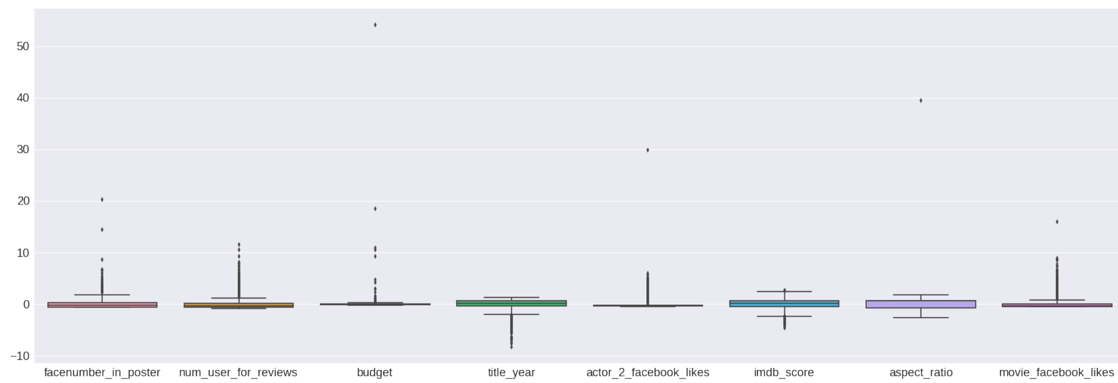
We will use boxplots, to investigete wether or not the dataset contains outliers

```
In [5]: %matplotlib inline
        plt.figure(figsize=(30,10))
```

```
sns.boxplot(data = df['numeric_std'].iloc[:,:8]);
plt.tick_params(labelsize=18)

plt.figure(figsize=(30,10))
sns.boxplot(data = df['numeric_std'].iloc[:,8:]);
plt.tick_params(labelsize=18)
```

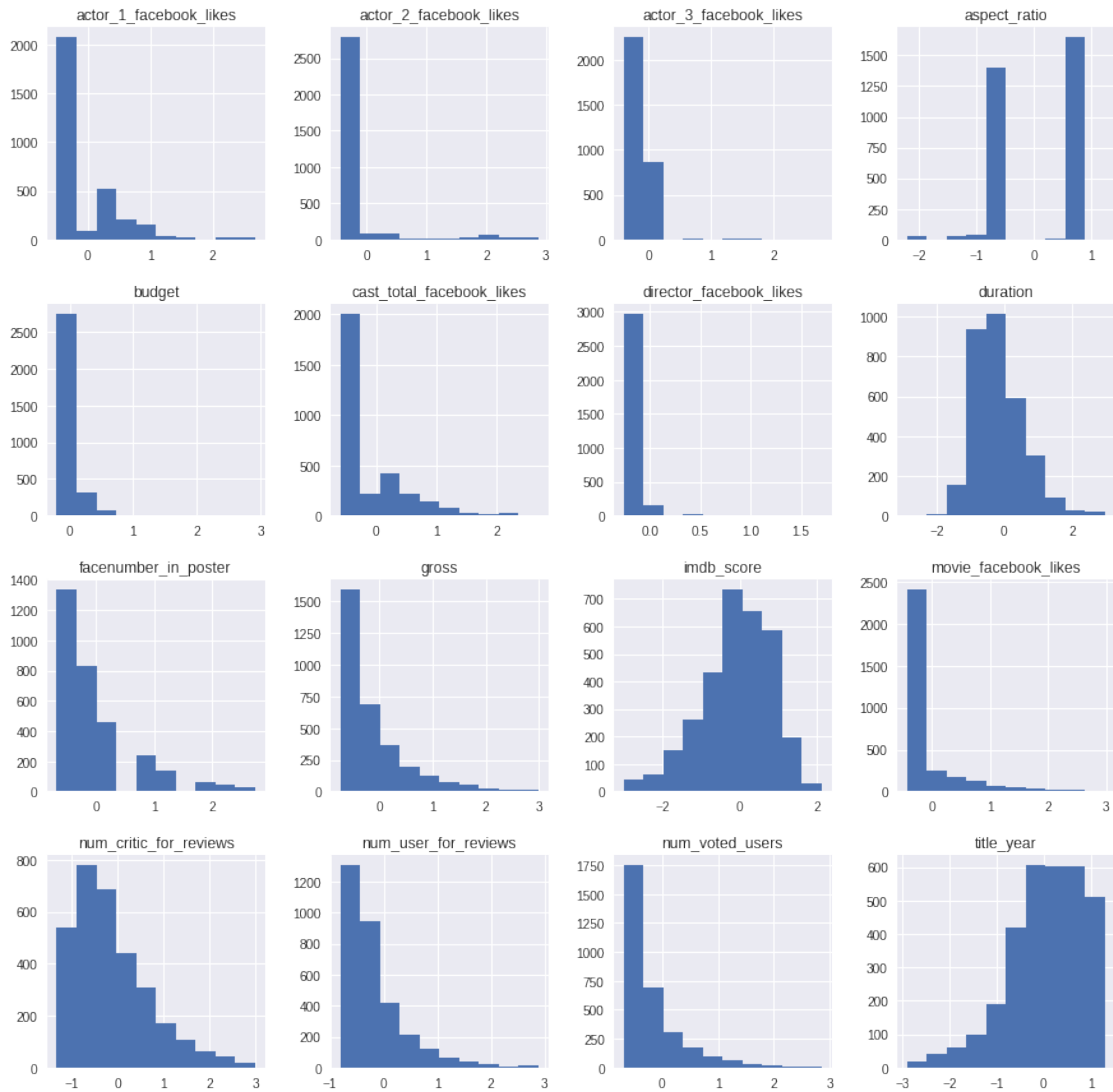




4.2 Histogram

The boxplot tells us that all the attributes contain outliers. These will have to be removed from the dataset, before plotting the histograms, to give a meaningfull insight as to wether or not the attributes are normally distributed.

```
In [6]: from scipy import stats
        df['no_outliers'] = df['numeric_std'][(np.abs(stats.zscore(df['numeric_std'])) < 3).all(
        df['no_outliers'].hist(figsize=(15,15));
```

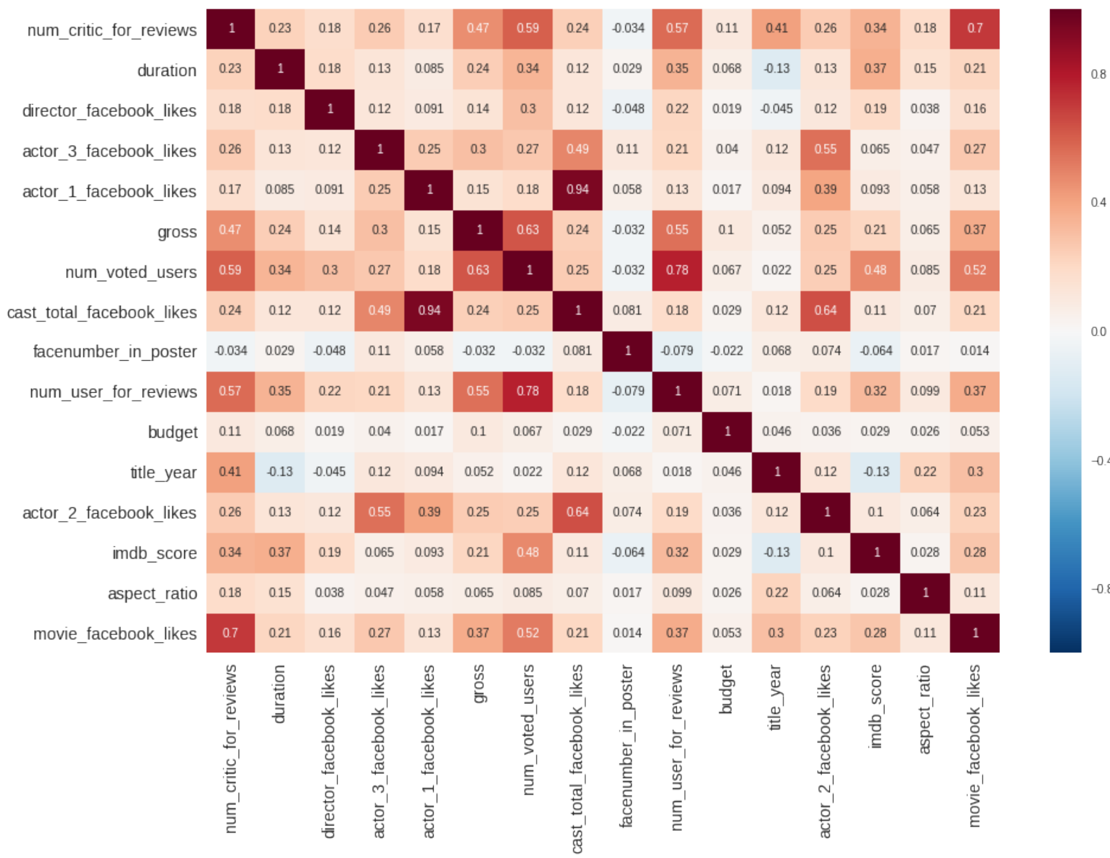


From the histograms, it is clear that only IMDB_score and duration is somewhere normally distributed. IMDB_score seemes to be a little left-skewed, which tells us that most movies are good movies, while not so many bad movies are included. Duration on the other hand, seems to be slightly right hand-skewed, which tells us that only a few movies are considered long.

4.3 Heatmap

We will use a heatmap, to investigate what attributes correlate with each other.

```
In [7]: plt.figure(figsize=(15,10))
        sns.heatmap(df['numeric'].corr(),annot=True);
        plt.tick_params(labelsize=14)
```



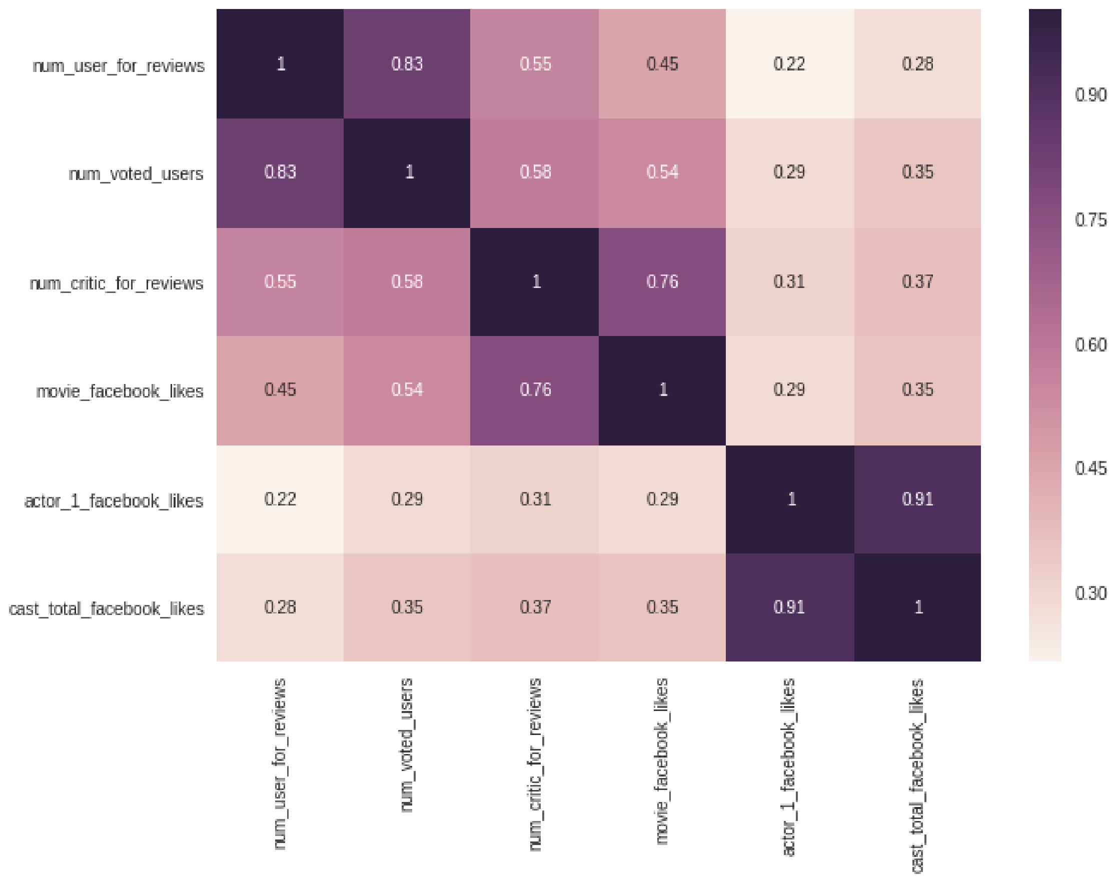
From the heatmap, we can identify correlation on the dataset to be: num_user_for_reviews & num_voted_users (medium) num_critics_for_reviews & movie_facebook_likes (low) actor_1_facebook_likes & cast_total_facebook_likes (high) We will now choose a minimum value for the attributes, and see if that changes the correlation. Facebook likes (generel): 100 reviews (generel): 10 votes: 100

```
In [8]: df['heatmap'] = df['numeric'][['num_user_for_reviews','num_voted_users',
                                       'num_critic_for_reviews','movie_facebook_likes',
                                       'actor_1_facebook_likes', 'cast_total_facebook_likes']]
        df['heatmap'] = df['heatmap'][(df['heatmap']['num_user_for_reviews'] > 10) &
                                      (df['heatmap']['num_critic_for_reviews'] > 10) &
                                      (df['heatmap']['num_voted_users'] > 100) &
                                      (df['heatmap']['movie_facebook_likes'] > 100) &
                                      (df['heatmap']['actor_1_facebook_likes'] > 100) &
                                      (df['heatmap']['cast_total_facebook_likes'] > 100)]
        print("This removes",((df['numeric'].shape[0]-df['heatmap'].shape[0])*100)/df['numeric']
```

```
This removes 49.72375690607735 % of the data, leaving 1911 rows.
```

By setting these conditions, we removed nearly half of the dataset. This is far from perfect, and could be done more efficently. We choose to try and map the correlation, to get an undertanding of wether or not this approach changes anything.

```
In [9]: plt.figure(figsize=(10,7))
        sns.heatmap(df['heatmap'].corr(),annot=True);
        plt.tick_params(labelsize=10)
```



From the new heatmap, it is clear that the correlation is nearly the same. As expected the correlation has generally increased, except for actor 1 vs cast facebook likes, which has decreased. The increased correlation dosent seem to be of a significant magnitude.

4.4 Eigendecomposition

The data is preprocessed, where the eigenvectors and eigenvalues are found with an eigendecomposition of the covariance matrix.

4.4.1 Covariance between features

To be able to perform an eigendecomposition to find the eigenvalues and eigenvectors, we first need to find the covariance between the features.

```
In [16]: mean_vector = np.mean(df['numeric_std'], axis=0)
         cov_matrix = np.cov(df['numeric_std'].T)
         print('Covariance matrix: \n', pd.DataFrame(cov_matrix))
```

```
Covariance matrix:
          0         1         2         3         4         5         6   \
0   1.000000  0.227705  0.176916  0.255086  0.170198  0.468535  0.594990
1   0.227705  1.000000  0.179734  0.125771  0.084720  0.244743  0.338038
2   0.176916  0.179734  1.000000  0.118240  0.090733  0.139938  0.300619
3   0.255086  0.125771  0.118240  1.000000  0.253720  0.301584  0.269455
4   0.170198  0.084720  0.090733  0.253720  1.000000  0.147045  0.182265
5   0.468535  0.244743  0.139938  0.301584  0.147045  1.000000  0.626948
6   0.594990  0.338038  0.300619  0.269455  0.182265  0.626948  1.000000
7   0.241005  0.121171  0.119741  0.490686  0.944925  0.238687  0.251940
8  -0.034009  0.029100 -0.047619  0.105018  0.057580 -0.032254 -0.032026
9   0.566795  0.350391  0.218311  0.207321  0.125221  0.547107  0.779925
10  0.105681  0.068161  0.018559  0.040478  0.017086  0.100389  0.066824
11  0.410380 -0.129422 -0.044606  0.115535  0.093742  0.052368  0.021938
12  0.255837  0.129452  0.116900  0.554182  0.392676  0.254659  0.246660
13  0.343881  0.366124  0.190838  0.064974  0.093131  0.212124  0.477917
14  0.180641  0.153114  0.037871  0.047123  0.057604  0.065260  0.085485
15  0.703969  0.214936  0.162737  0.272513  0.131778  0.368494  0.518691

          7         8         9         10        11        12        13  \
0   0.241005 -0.034009  0.566795  0.105681  0.410380  0.255837  0.343881
1   0.121171  0.029100  0.350391  0.068161 -0.129422  0.129452  0.366124
2   0.119741 -0.047619  0.218311  0.018559 -0.044606  0.116900  0.190838
3   0.490686  0.105018  0.207321  0.040478  0.115535  0.554182  0.064974
4   0.944925  0.057580  0.125221  0.017086  0.093742  0.392676  0.093131
5   0.238687 -0.032254  0.547107  0.100389  0.052368  0.254659  0.212124
6   0.251940 -0.032026  0.779925  0.066824  0.021938  0.246660  0.477917
7   1.000000  0.080985  0.182288  0.029423  0.124015  0.644016  0.106259
8   0.080985  1.000000 -0.079404 -0.021757  0.067952  0.074138 -0.064292
9   0.182288 -0.079404  1.000000  0.071254  0.017594  0.189582  0.322522
10  0.029423 -0.021757  0.071254  1.000000  0.046293  0.036211  0.029041
11  0.124015  0.067952  0.017594  0.046293  1.000000  0.119739 -0.129265
12  0.644016  0.074138  0.189582  0.036211  0.119739  1.000000  0.102060
13  0.106259 -0.064292  0.322522  0.029041 -0.129265  0.102060  1.000000
14  0.069675  0.016620  0.098557  0.025796  0.219779  0.064215  0.028454
15  0.207061  0.014332  0.371970  0.053035  0.302835  0.233632  0.279478

          14        15
0   0.180641  0.703969
1   0.153114  0.214936
2   0.037871  0.162737
3   0.047123  0.272513
4   0.057604  0.131778
5   0.065260  0.368494
6   0.085485  0.518691
7   0.069675  0.207061
8   0.016620  0.014332
9   0.098557  0.371970
```

```
10  0.025796  0.053035
11  0.219779  0.302835
12  0.064215  0.233632
13  0.028454  0.279478
14  1.000000  0.110318
15  0.110318  1.000000
```

4.4.2 Eigenvalues and eigenvectors

Now the eigendecomposition of the covariance matrix can be performed.

```
In [17]: eigen_val, eigen_vec = np.linalg.eig(cov_matrix)
         print('Eigenvalues: \n', pd.DataFrame(eigen_val))
         print('Eigenvectors: \n', pd.DataFrame(eigen_vec))
```

```
Eigenvalues:
            0
0   4.445805
1   2.136388
2   1.496763
3   0.001771
4   0.149833
5   0.245490
6   0.415714
7   0.443200
8   0.483090
9   0.591156
10  0.781094
11  1.051047
12  1.013853
13  0.995797
14  0.865921
15  0.883081
Eigenvectors:
           0         1         2         3         4         5         6   \
0  -0.362354 -0.155232 -0.327054 -0.001620 -0.331574 -0.750503 -0.041216
1  -0.206547 -0.177457  0.272901  0.001781 -0.082209  0.022426  0.001478
2  -0.158009 -0.094838  0.208406 -0.000006  0.057453 -0.049707 -0.031895
3  -0.252237  0.277675  0.006544  0.111629 -0.007320 -0.052765  0.640803
4  -0.222852  0.463064  0.164287  0.616236  0.013754 -0.008826  0.140744
5  -0.317778 -0.135717  0.017333  0.012664  0.150519 -0.075840 -0.205560
6  -0.385210 -0.240217  0.093722 -0.001502 -0.659253  0.483335  0.060286
7  -0.284547  0.505361  0.148158 -0.757892  0.013455 -0.004442  0.039677
8  -0.004622  0.161083 -0.088785  0.000465  0.027568 -0.051492 -0.042603
9  -0.340355 -0.257060  0.074307 -0.000977  0.542281 -0.049412  0.136651
10 -0.057873 -0.037668 -0.067696 -0.000343  0.006653  0.035775  0.009101
11 -0.102572  0.114371 -0.667981  0.000974  0.065662  0.271071  0.102190
```

```
12 -0.263037  0.365829  0.062501  0.182208  0.001756  0.042682 -0.686442
13 -0.222104 -0.238787  0.272306 -0.004567  0.197252 -0.001867  0.072548
14 -0.092043 -0.007484 -0.290456 -0.000429  0.010861  0.009207 -0.009985
15 -0.317372 -0.122225 -0.300752  0.001010  0.289985  0.331305 -0.108632

           7         8         9        10        11        12        13  \
0  -0.116389  0.036545 -0.029760  0.071827 -0.061316  0.037905  0.081510
1   0.068313 -0.101619 -0.703490  0.249215  0.480791 -0.061195 -0.157182
2   0.093815 -0.012812 -0.034403 -0.273099  0.126949  0.120085  0.160521
3   0.045459  0.056373  0.076949  0.343233 -0.138729 -0.284859 -0.135369
4   0.040965  0.103225 -0.075917 -0.261221  0.078790  0.287897  0.113896
5   0.715876  0.092710 -0.024031 -0.296164 -0.248506 -0.109607 -0.126009
6  -0.143677 -0.060834  0.163757 -0.202532 -0.102664 -0.064023  0.037687
7   0.004679  0.040474 -0.030542 -0.093371  0.023544  0.164534  0.054089
8  -0.024626 -0.010567  0.133912 -0.320901  0.345420 -0.759305 -0.207488
9  -0.511961 -0.285736 -0.027260 -0.303659 -0.110103 -0.017139 -0.005135
10 -0.050465  0.029301  0.084763 -0.009318 -0.066195  0.321987 -0.900841
11  0.222318 -0.561258 -0.183168  0.011778  0.024169  0.089325  0.090690
12 -0.196303 -0.228572  0.107281  0.309857 -0.067118 -0.102478 -0.065726
13  0.268416 -0.242306  0.517461  0.416090  0.199927  0.049891  0.102415
14 -0.022946  0.190690  0.330770 -0.110311  0.682436  0.255166 -0.006485
15 -0.101522  0.644562 -0.122173  0.240690 -0.063741 -0.081187  0.117026

          14        15
0  -0.164751 -0.053200
1   0.065932 -0.090268
2  -0.252499  0.843224
3   0.313751  0.295252
4  -0.229883 -0.260482
5   0.315229 -0.105331
6   0.030401 -0.064023
7  -0.086317 -0.127295
8  -0.310962 -0.060707
9   0.185893 -0.116929
10 -0.230444  0.076745
11 -0.154129  0.057410
12  0.198561  0.172609
13 -0.342140 -0.183407
14  0.464810  0.037680
15 -0.255101  0.013326
```

4.5 Selecting principal components

The next step towards the goal of the principal component analysis is to find the number of principal components to include in the model.

First the eigenvalues and eigenvectors are gathered in tuples.

```
In [12]: eigen_pairs = [(np.abs(eigen_val[i]), eigen_vec[:,i]) for i in range(len(eigen_val))]

         eigen_pairs.sort()
         eigen_pairs.reverse()

         print('Eigenvalues from highest to lowest:')
         for eig_val in eigen_pairs:
             print(eig_val[0])
```

```
Eigenvalues from highest to lowest:
4.44580469646
2.1363882565
1.49676251129
1.05104701557
1.01385279214
0.995797014572
0.883080919082
0.86592055526
0.781093607224
0.591155511436
0.483090304644
0.443199530174
0.415713571524
0.245489846152
0.149832591928
0.00177127604066
```

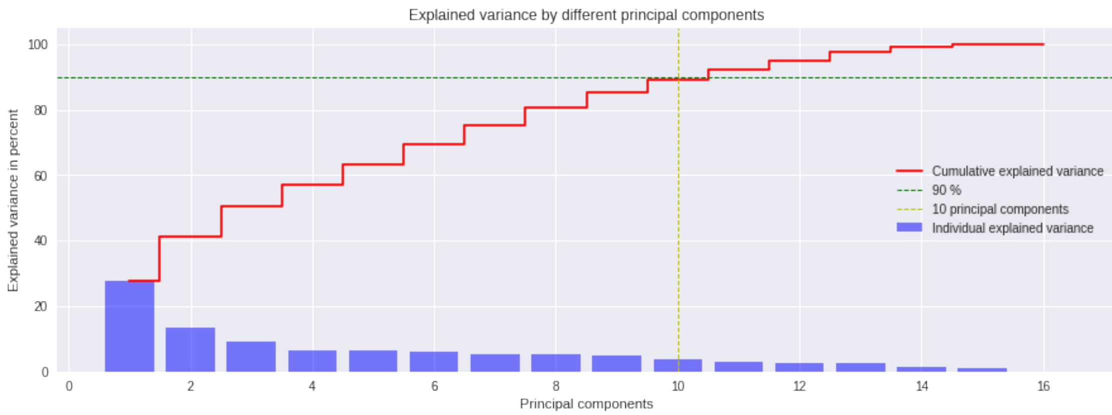
4.5.1 Explained variance

A plot is generated showing the explained variance.

```
In [13]: var_explained = [(i / sum(eigen_val))*100 for i in sorted(eigen_val, reverse=True)]
         cum_var_explained = np.cumsum(var_explained)

         plt.figure(figsize=(15,5))
         plt.bar(range(1,17), var_explained, alpha=0.5, align='center', label='Individual explai
         plt.step(range(1,17), cum_var_explained, where='mid', label='Cumulative explained varia
         plt.axhline(y=90, linewidth=1, color='g', linestyle='dashed', label='90 %')
         plt.axvline(x=10, linewidth=1, color='y', linestyle='dashed', label='10 principal compo
         plt.title('Explained variance by different principal components')
         plt.ylabel('Explained variance in percent')
         plt.xlabel('Principal components')
         plt.legend(loc='center right')
         plt.show()

         print('Explained variance with 10 principal components: {} %'.format(sum(var_explained[
```




```
Explained variance with 10 principal components: 89.13064299709966 %
```

It is shown in the plot above, that 10 principal components explain approximately 90% of the total variance.

4.6 Data projection

The 16 principal components represent a 16 dimensional feature space. We can acheive 90 % of the explained variance by projecting 10 principal components onto a new feature space of 10 dimensions.

A projection matrix is constructed, which will represent a 10-dimensional feature space including the 10 first principal components as columns.

```
In [20]: projection_mat = np.hstack((eigen_pairs[0][1].reshape(16,1),
                                     eigen_pairs[1][1].reshape(16,1),
                                     eigen_pairs[2][1].reshape(16,1),
                                     eigen_pairs[3][1].reshape(16,1),
                                     eigen_pairs[4][1].reshape(16,1),
                                     eigen_pairs[5][1].reshape(16,1),
                                     eigen_pairs[6][1].reshape(16,1),
                                     eigen_pairs[7][1].reshape(16,1),
                                     eigen_pairs[8][1].reshape(16,1),
                                     eigen_pairs[9][1].reshape(16,1)))
         print('Principal component space(10-dimensional): \n',pd.DataFrame(projection_mat))
```

```
Principal component space(10-dimensional):
          0         1         2         3         4         5         6  \
0  -0.362354 -0.155232 -0.327054 -0.061316  0.037905  0.081510 -0.053200
1  -0.206547 -0.177457  0.272901  0.480791 -0.061195 -0.157182 -0.090268
2  -0.158009 -0.094838  0.208406  0.126949  0.120085  0.160521  0.843224
3  -0.252237  0.277675  0.006544 -0.138729 -0.284859 -0.135369  0.295252
4  -0.222852  0.463064  0.164287  0.078790  0.287897  0.113896 -0.260482
```

5	-0.317778	-0.135717	0.017333	-0.248506	-0.109607	-0.126009	-0.105331
6	-0.385210	-0.240217	0.093722	-0.102664	-0.064023	0.037687	-0.064023
7	-0.284547	0.505361	0.148158	0.023544	0.164534	0.054089	-0.127295
8	-0.004622	0.161083	-0.088785	0.345420	-0.759305	-0.207488	-0.060707
9	-0.340355	-0.257060	0.074307	-0.110103	-0.017139	-0.005135	-0.116929
10	-0.057873	-0.037668	-0.067696	-0.066195	0.321987	-0.900841	0.076745
11	-0.102572	0.114371	-0.667981	0.024169	0.089325	0.090690	0.057410
12	-0.263037	0.365829	0.062501	-0.067118	-0.102478	-0.065726	0.172609
13	-0.222104	-0.238787	0.272306	0.199927	0.049891	0.102415	-0.183407
14	-0.092043	-0.007484	-0.290456	0.682436	0.255166	-0.006485	0.037680
15	-0.317372	-0.122225	-0.300752	-0.063741	-0.081187	0.117026	0.013326

	7	8	9
0	-0.164751	0.071827	-0.029760
1	0.065932	0.249215	-0.703490
2	-0.252499	-0.273099	-0.034403
3	0.313751	0.343233	0.076949
4	-0.229883	-0.261221	-0.075917
5	0.315229	-0.296164	-0.024031
6	0.030401	-0.202532	0.163757
7	-0.086317	-0.093371	-0.030542
8	-0.310962	-0.320901	0.133912
9	0.185893	-0.303659	-0.027260
10	-0.230444	-0.009318	0.084763
11	-0.154129	0.011778	-0.183168
12	0.198561	0.309857	0.107281
13	-0.342140	0.416090	0.517461
14	0.464810	-0.110311	0.330770
15	-0.255101	0.240690	-0.122173