# Box Office Prediction-Copy1

October 25, 2018

## 0.1 Initial imports and loading data with Pandas

```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn import tree
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.linear_model import LinearRegression
        from sklearn.metrics import mean_squared_error, r2_score
        from sklearn.model_selection import train_test_split
        %matplotlib inline

        pd.set_option('mode.chained_assignment', None)
        pd.set_option('display.float_format', '{:,.2f}'.format)
```

```
In [3]: data = pd.read_csv('movie_metadata.csv')
```

## 0.2 Taking a look at the data

You need to "run" the two cells below, to do that select the cell and press: *Shift-Enter*

```
In [4]: # Run this cell (to do so press Shift-Enter)
        data.head(5)
```

```
Out[4]:    color      director_name  num_critic_for_reviews  duration  \
        0  Color      James Cameron                  723.00    178.00
        1  Color     Gore Verbinski                  302.00    169.00
        2  Color         Sam Mendes                  602.00    148.00
        3  Color  Christopher Nolan                  813.00    164.00
        4    NaN        Doug Walker                     nan       nan

           director_facebook_likes  actor_3_facebook_likes      actor_2_name  \
        0                     0.00                  855.00  Joel David Moore
        1                   563.00                1,000.00     Orlando Bloom
        2                     0.00                  161.00      Rory Kinnear
        3                22,000.00               23,000.00    Christian Bale
        4                   131.00                     nan        Rob Walker
```

```
   actor_1_facebook_likes          gross                         genres  \
0                1,000.00 760,505,847.00  Action|Adventure|Fantasy|Sci-Fi
1               40,000.00 309,404,152.00         Action|Adventure|Fantasy
2               11,000.00 200,074,175.00        Action|Adventure|Thriller
3               27,000.00 448,130,642.00                  Action|Thriller
4                  131.00            nan                      Documentary

          ...          num_user_for_reviews language  country  content_rating  \
0         ...                      3,054.00  English      USA           PG-13
1         ...                      1,238.00  English      USA           PG-13
2         ...                        994.00  English       UK           PG-13
3         ...                      2,701.00  English      USA           PG-13
4         ...                           nan      NaN      NaN             NaN

          budget  title_year actor_2_facebook_likes imdb_score  aspect_ratio  \
0 237,000,000.00    2,009.00                 936.00       7.90          1.78
1 300,000,000.00    2,007.00               5,000.00       7.10          2.35
2 245,000,000.00    2,015.00                 393.00       6.80          2.35
3 250,000,000.00    2,012.00              23,000.00       8.50          2.35
4            nan         nan                  12.00       7.10           nan

  movie_facebook_likes
0                33000
1                    0
2                85000
3               164000
4                    0

[5 rows x 28 columns]
```

```
In [5]: data.shape
```

```
Out[5]: (5043, 28)
```

```
In [6]: data.describe()
```

```
Out[6]:        num_critic_for_reviews  duration  director_facebook_likes  \
count                4,993.00  5,028.00                 4,939.00
mean                   140.19    107.20                   686.51
std                    121.60     25.20                 2,813.33
min                      1.00      7.00                     0.00
25%                     50.00     93.00                     7.00
50%                    110.00    103.00                    49.00
75%                    195.00    118.00                   194.50
max                    813.00    511.00                23,000.00

       actor_3_facebook_likes  actor_1_facebook_likes          gross  \
count                5,020.00                5,036.00       4,159.00
```

| | | | |
|---|---|---|---|
| mean | 645.01 | 6,560.05 | 48,507,385.63 |
| std | 1,665.04 | 15,020.76 | 68,471,915.43 |
| min | 0.00 | 0.00 | 162.00 |
| 25% | 133.00 | 614.00 | 5,351,178.00 |
| 50% | 371.50 | 988.00 | 25,528,495.00 |
| 75% | 636.00 | 11,000.00 | 62,319,957.00 |
| max | 23,000.00 | 640,000.00 | 760,505,847.00 |

| | num_voted_users | cast_total_facebook_likes | facenumber_in_poster \ |
|---|---|---|---|
| count | 5,043.00 | 5,043.00 | 5,030.00 |
| mean | 83,668.16 | 9,699.06 | 1.37 |
| std | 138,485.26 | 18,163.80 | 2.01 |
| min | 5.00 | 0.00 | 0.00 |
| 25% | 8,593.50 | 1,411.00 | 0.00 |
| 50% | 34,359.00 | 3,090.00 | 1.00 |
| 75% | 96,309.00 | 13,756.50 | 2.00 |
| max | 1,689,764.00 | 656,730.00 | 43.00 |

| | num_user_for_reviews | budget | title_year \ |
|---|---|---|---|
| count | 5,022.00 | 4,551.00 | 4,935.00 |
| mean | 272.77 | 39,752,620.44 | 2,002.47 |
| std | 377.98 | 206,114,898.45 | 12.47 |
| min | 1.00 | 218.00 | 1,916.00 |
| 25% | 65.00 | 6,000,000.00 | 1,999.00 |
| 50% | 156.00 | 20,000,000.00 | 2,005.00 |
| 75% | 326.00 | 45,000,000.00 | 2,011.00 |
| max | 5,060.00 | 12,215,500,000.00 | 2,016.00 |

| | actor_2_facebook_likes | imdb_score | aspect_ratio | movie_facebook_likes |
|---|---|---|---|---|
| count | 5,030.00 | 5,043.00 | 4,714.00 | 5,043.00 |
| mean | 1,651.75 | 6.44 | 2.22 | 7,525.96 |
| std | 4,042.44 | 1.13 | 1.39 | 19,320.45 |
| min | 0.00 | 1.60 | 1.18 | 0.00 |
| 25% | 281.00 | 5.80 | 1.85 | 0.00 |
| 50% | 595.00 | 6.60 | 2.35 | 166.00 |
| 75% | 918.00 | 7.20 | 2.35 | 3,000.00 |
| max | 137,000.00 | 9.50 | 16.00 | 349,000.00 |

Some key points from this table: - Avg movie duration is 107.2 minutes - Avg imdb is 6.44 - Avg number of users reviews is 272

## 0.3 Cleaning the data

### 0.3.1 Dealing with duplicates

```
In [10]: print ('Number of duplicates in data: {}'.format(
             sum(data.duplicated(subset=['movie_title', 'title_year'], keep=False))))
```

```
Number of duplicates in data: 0
```

```
In [9]: data = data.drop_duplicates(subset=['movie_title', 'title_year'], keep='first').copy()
```

### 0.3.2 Fixing Null and some zero values

```
In [11]: # check if data has any null/nan values
         data.isnull().values.any()
```

```
Out[11]: True
```

```
In [15]: # Check how many values are null in each column
         def show_missing_data(data):
             missing_data = data.isnull().sum().reset_index()
             missing_data.columns = ['column_name', 'missing_count']
             missing_data['filling_factor'] = (data.shape[0] - missing_data['missing_count']) /
             return missing_data.sort_values('filling_factor').reset_index(drop=True)

         show_missing_data(data)[:5]
```

Out[15]:

| | column_name | missing_count | filling_factor |
|---|---|---|---|
| 0 | budget | 266 | 93.44 |
| 1 | aspect_ratio | 104 | 97.44 |
| 2 | content_rating | 64 | 98.42 |
| 3 | plot_keywords | 40 | 99.01 |
| 4 | actor_3_name | 13 | 99.68 |

As we are working with the Gross Box Office, rows without it are of no use. So we will exclude those films that are missing the Gross Box Office.

```
In [16]: data.dropna(subset=['gross'], how='all', inplace=True)
         show_missing_data(data)[:5]
```

Out[16]:

| | column_name | missing_count | filling_factor |
|---|---|---|---|
| 0 | budget | 266 | 93.44 |
| 1 | aspect_ratio | 104 | 97.44 |
| 2 | content_rating | 64 | 98.42 |
| 3 | plot_keywords | 40 | 99.01 |
| 4 | actor_3_name | 13 | 99.68 |

Fill out missing budget datapoints with the median budget for the year it was released.

```
In [386]: median_budget_per_year = data.groupby('title_year')['budget'].transform('median')
          data['budget'].fillna(median_budget_per_year, inplace=True)

          show_missing_data(data)[:5]
```

```
Out[386]:                  column_name  missing_count  filling_factor
          0             aspect_ratio            104           97.44
          1           content_rating             64           98.42
          2            plot_keywords             40           99.01
          3             actor_3_name             13           99.68
          4   actor_3_facebook_likes             13           99.68
```

Fill out the rest of the missing data

```
In [387]: data.fillna(0, inplace=True)
```

Delete all rows where `title_year` is zero

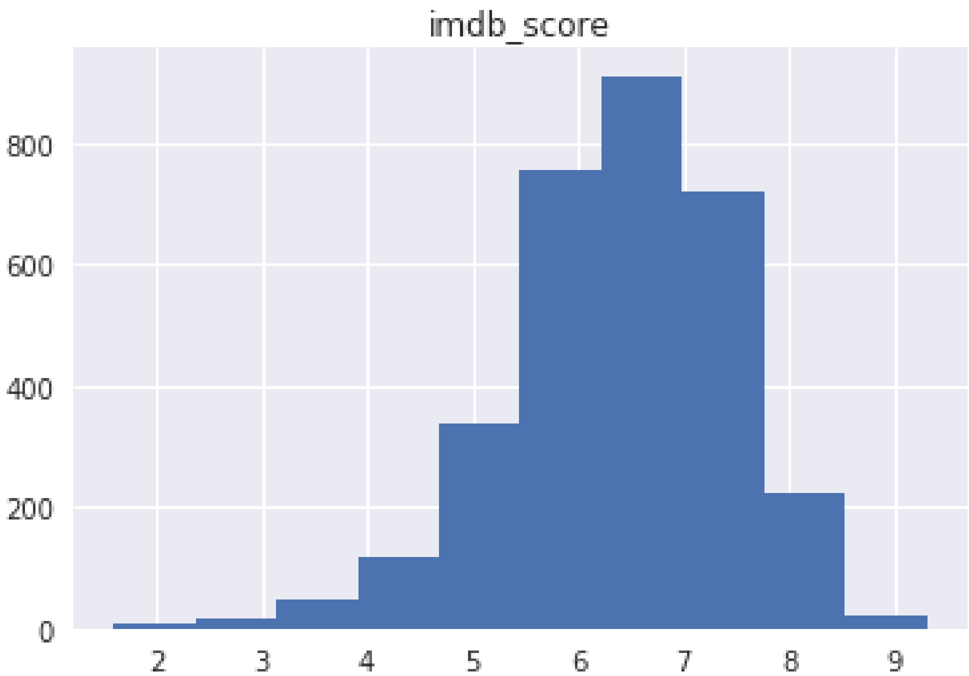
```
In [388]: data = data[data['title_year'] != 0]
```

Budgets are in each country's currency so we are going to use only US movies

```
In [389]: data = data[data['country'] == 'USA']
```

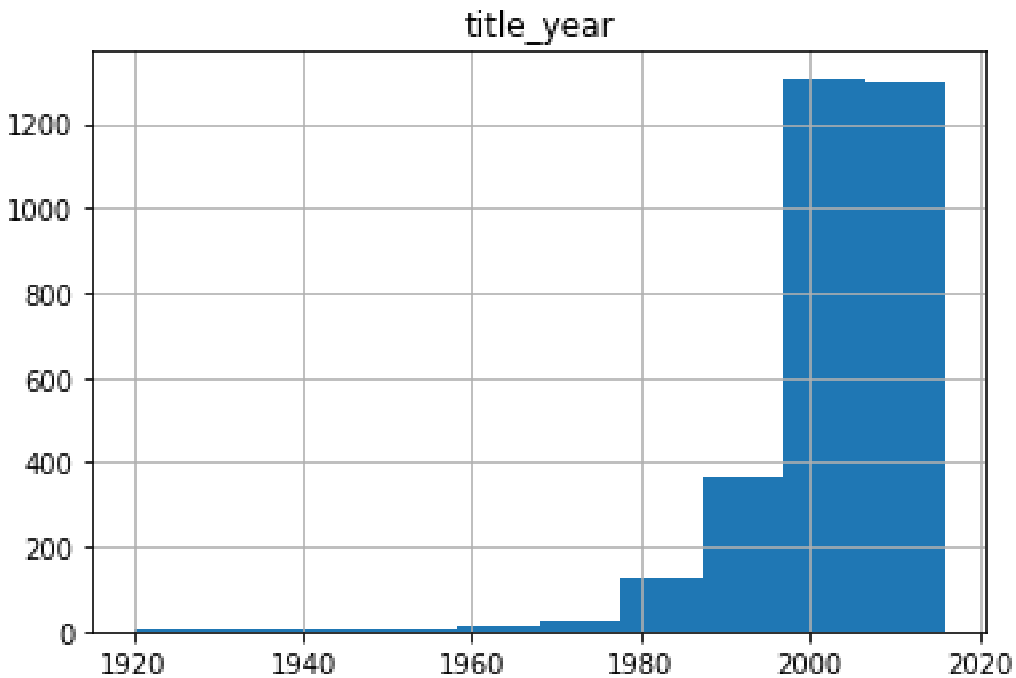
## 0.4 Understanding the data

```
In [25]: # IMDb rating distribution
         data.hist(column='imdb_score')

Out[25]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fa49f313c88>]], dtype=objec

/opt/conda/lib/python3.5/site-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: F
  (prop.get_family(), self.defaultFamily[fontext]))
```



In [391]:
```
# Movies per year
data.hist(column='title_year')
```

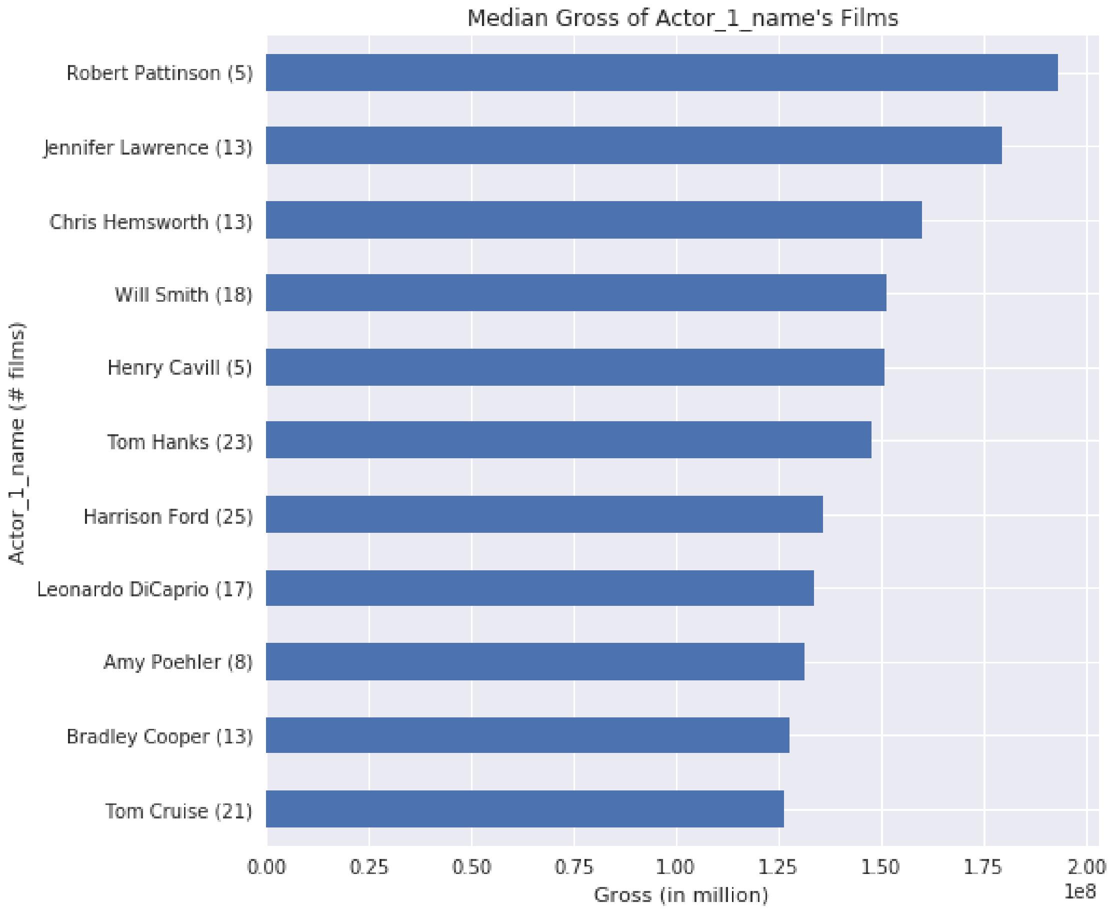
Out[391]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x129be56d8>]], dtype=object)
```



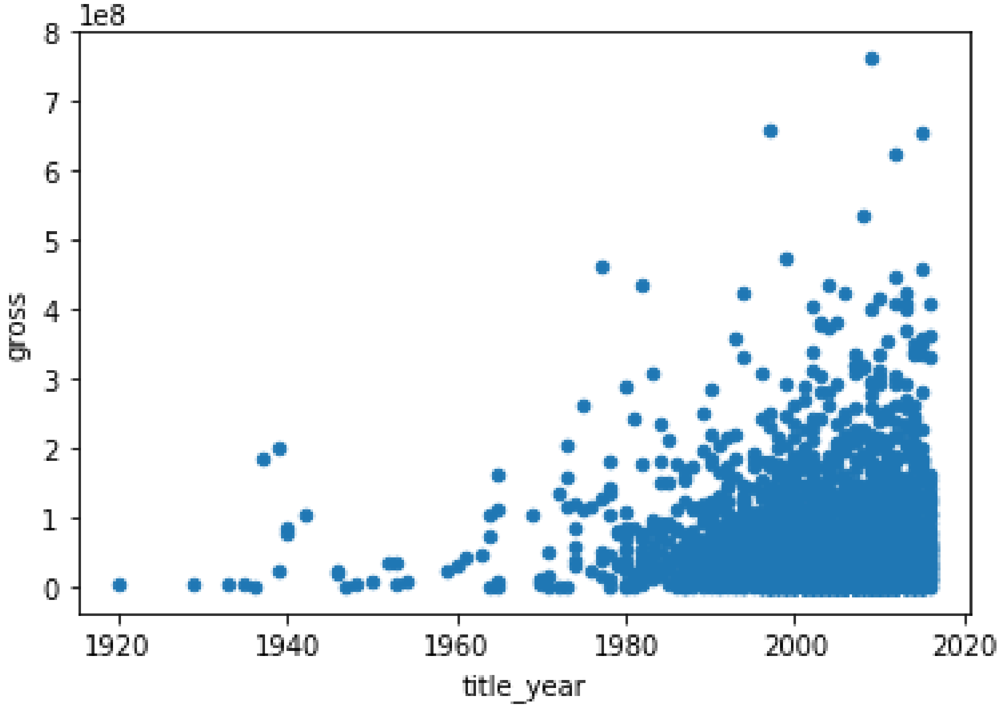

In [31]:
```
# Median gross box office per actor
fig = plt.figure(figsize=(8,8))
comparison_df = data.groupby('actor_1_name', as_index=False).mean().sort_values('gross'

name_count_key = data['actor_1_name'].value_counts().to_dict()
comparison_df['films'] = comparison_df['actor_1_name'].map(name_count_key)

comparison_df['actor_1_name'] = comparison_df['actor_1_name'].map(str) + " (" + compari

comparison_df[comparison_df['films'] >= 5][['actor_1_name', 'gross']][10::-1].set_index
plt.legend().set_visible(False)
plt.title("Median Gross of Actor_1_name's Films")
plt.ylabel("Actor_1_name (# films)")
plt.xlabel("Gross (in million)")
```

Out[31]:
```
<matplotlib.text.Text at 0x7fa49ef14eb8>
```

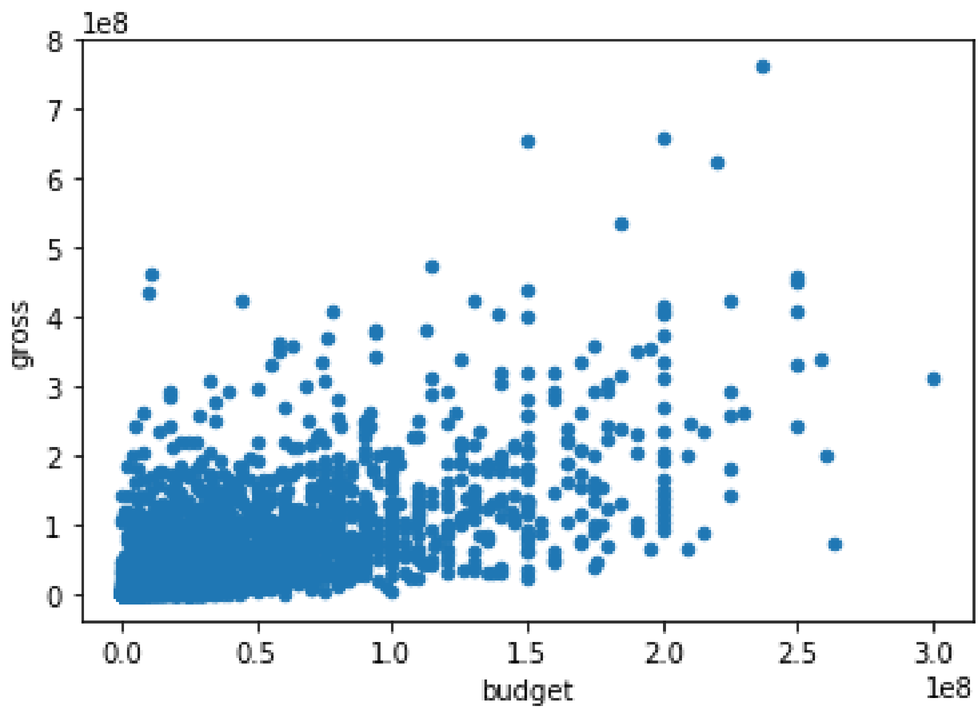
```
/opt/conda/lib/python3.5/site-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: F
  (prop.get_family(), self.defaultFamily[fontext]))
```




```
In [393]: # title year vs gross
          data.plot.scatter(x='title_year', y='gross')

Out[393]: <matplotlib.axes._subplots.AxesSubplot at 0x12aa7a9b0>
```




```
In [394]: # budget vs gross
          data.plot.scatter(x='budget', y='gross')

Out[394]: <matplotlib.axes._subplots.AxesSubplot at 0x12b46a9e8>
```



In [32]:
```
data.corr()
```

Out[32]:
```
                           num_critic_for_reviews  duration  \
num_critic_for_reviews                       1.00      0.28
duration                                     0.28      1.00
director_facebook_likes                      0.19      0.21
actor_3_facebook_likes                       0.28      0.14
actor_1_facebook_likes                       0.18      0.10
gross                                        0.49      0.29
num_voted_users                              0.61      0.37
cast_total_facebook_likes                    0.25      0.14
facenumber_in_poster                        -0.03      0.01
num_user_for_reviews                         0.58      0.37
budget                                       0.49      0.30
title_year                                   0.39     -0.11
actor_2_facebook_likes                       0.28      0.15
imdb_score                                   0.36      0.38
aspect_ratio                                 0.24      0.18
movie_facebook_likes                         0.70      0.25

                           director_facebook_likes  actor_3_facebook_likes  \
num_critic_for_reviews                        0.19                    0.28
duration                                      0.21                    0.14
director_facebook_likes                       1.00                    0.13
actor_3_facebook_likes                        0.13                    1.00
actor_1_facebook_likes                        0.09                    0.25
gross                                         0.14                    0.28
num_voted_users                               0.32                    0.27
cast_total_facebook_likes                     0.12                    0.47
facenumber_in_poster                         -0.05                    0.10
num_user_for_reviews                          0.25                    0.22
budget                                        0.10                    0.27
title_year                                   -0.06                    0.12
actor_2_facebook_likes                        0.12                    0.54
imdb_score                                    0.22                    0.09
aspect_ratio                                  0.06                    0.07
movie_facebook_likes                          0.18                    0.30

                           actor_1_facebook_likes  gross  num_voted_users  \
num_critic_for_reviews                       0.18   0.49             0.61
duration                                     0.10   0.29             0.37
director_facebook_likes                      0.09   0.14             0.32
actor_3_facebook_likes                       0.25   0.28             0.27
actor_1_facebook_likes                       1.00   0.13             0.18
gross                                        0.13   1.00             0.64
```

| | | | |
|---|---|---|---|
| num_voted_users | 0.18 | 0.64 | 1.00 |
| cast_total_facebook_likes | 0.95 | 0.22 | 0.25 |
| facenumber_in_poster | 0.06 | -0.03 | -0.04 |
| num_user_for_reviews | 0.13 | 0.56 | 0.79 |
| budget | 0.15 | 0.65 | 0.42 |
| title_year | 0.09 | 0.03 | 0.02 |
| actor_2_facebook_likes | 0.38 | 0.24 | 0.25 |
| imdb_score | 0.12 | 0.26 | 0.50 |
| aspect_ratio | 0.07 | 0.13 | 0.14 |
| movie_facebook_likes | 0.13 | 0.38 | 0.53 |

| | cast_total_facebook_likes | facenumber_in_poster \ |
|---|---|---|
| num_critic_for_reviews | 0.25 | -0.03 |
| duration | 0.14 | 0.01 |
| director_facebook_likes | 0.12 | -0.05 |
| actor_3_facebook_likes | 0.47 | 0.10 |
| actor_1_facebook_likes | 0.95 | 0.06 |
| gross | 0.22 | -0.03 |
| num_voted_users | 0.25 | -0.04 |
| cast_total_facebook_likes | 1.00 | 0.08 |
| facenumber_in_poster | 0.08 | 1.00 |
| num_user_for_reviews | 0.19 | -0.08 |
| budget | 0.23 | -0.03 |
| title_year | 0.12 | 0.08 |
| actor_2_facebook_likes | 0.62 | 0.07 |
| imdb_score | 0.13 | -0.08 |
| aspect_ratio | 0.09 | 0.01 |
| movie_facebook_likes | 0.21 | 0.01 |

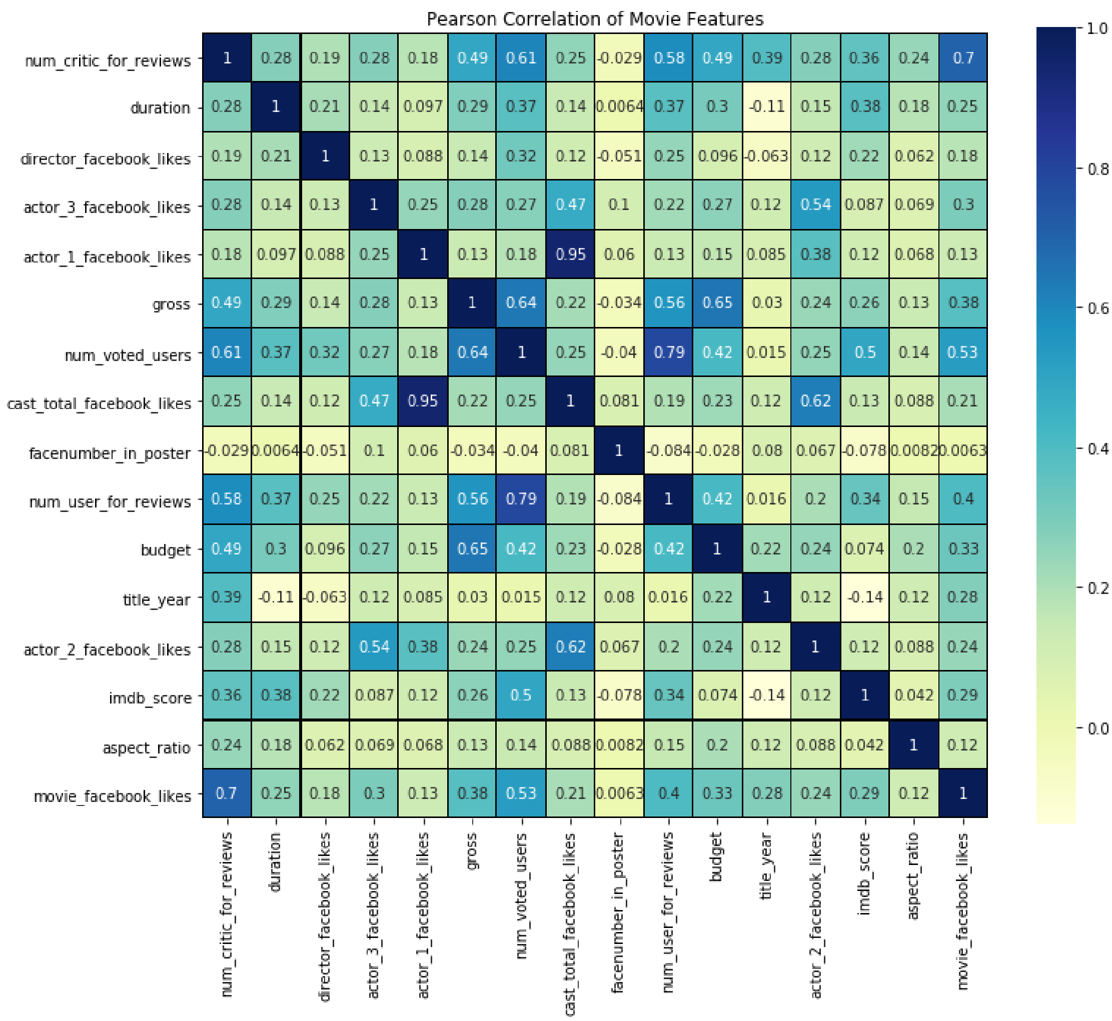
| | num_user_for_reviews | budget | title_year \ |
|---|---|---|---|
| num_critic_for_reviews | 0.58 | 0.49 | 0.39 |
| duration | 0.37 | 0.30 | -0.11 |
| director_facebook_likes | 0.25 | 0.10 | -0.06 |
| actor_3_facebook_likes | 0.22 | 0.27 | 0.12 |
| actor_1_facebook_likes | 0.13 | 0.15 | 0.09 |
| gross | 0.56 | 0.65 | 0.03 |
| num_voted_users | 0.79 | 0.42 | 0.02 |
| cast_total_facebook_likes | 0.19 | 0.23 | 0.12 |
| facenumber_in_poster | -0.08 | -0.03 | 0.08 |
| num_user_for_reviews | 1.00 | 0.42 | 0.02 |
| budget | 0.42 | 1.00 | 0.22 |
| title_year | 0.02 | 0.22 | 1.00 |
| actor_2_facebook_likes | 0.20 | 0.24 | 0.12 |
| imdb_score | 0.34 | 0.07 | -0.14 |
| aspect_ratio | 0.15 | 0.20 | 0.12 |
| movie_facebook_likes | 0.40 | 0.33 | 0.28 |

| | actor_2_facebook_likes | imdb_score | aspect_ratio \ |
|---|---|---|---|

```
num_critic_for_reviews                         0.28        0.36          0.24
duration                                       0.15        0.38          0.18
director_facebook_likes                        0.12        0.22          0.06
actor_3_facebook_likes                         0.54        0.09          0.07
actor_1_facebook_likes                         0.38        0.12          0.07
gross                                          0.24        0.26          0.13
num_voted_users                                0.25        0.50          0.14
cast_total_facebook_likes                      0.62        0.13          0.09
facenumber_in_poster                           0.07       -0.08          0.01
num_user_for_reviews                           0.20        0.34          0.15
budget                                         0.24        0.07          0.20
title_year                                     0.12       -0.14          0.12
actor_2_facebook_likes                         1.00        0.12          0.09
imdb_score                                     0.12        1.00          0.04
aspect_ratio                                   0.09        0.04          1.00
movie_facebook_likes                           0.24        0.29          0.12

                           movie_facebook_likes
num_critic_for_reviews                     0.70
duration                                   0.25
director_facebook_likes                    0.18
actor_3_facebook_likes                     0.30
actor_1_facebook_likes                     0.13
gross                                      0.38
num_voted_users                            0.53
cast_total_facebook_likes                  0.21
facenumber_in_poster                       0.01
num_user_for_reviews                       0.40
budget                                     0.33
title_year                                 0.28
actor_2_facebook_likes                     0.24
imdb_score                                 0.29
aspect_ratio                               0.12
movie_facebook_likes                       1.00
```

In [396]:
```python
# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(12, 10))
plt.title('Pearson Correlation of Movie Features')

# Draw the heatmap using seaborn
sns.heatmap(data.corr(),linewidths=0.25,vmax=1.0, square=True, cmap="YlGnBu", linecolo
```

Out[396]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x130f627f0>
```




As we can see from the heatmap, there are regions (features) where we can see quite positive linear correlations amongst each other, given the darker shade of the colours - top left-hand corner and bottom right quarter. This is a good sign as it means we may be able to find linearly correlated features for which we can perform PCA projections on.

```
In [33]: data.corr()['gross'].sort_values(ascending=False)
```

```
Out[33]: gross                      1.00
         budget                     0.65
         num_voted_users            0.64
         num_user_for_reviews       0.56
         num_critic_for_reviews     0.49
         movie_facebook_likes       0.38
         duration                   0.29
         actor_3_facebook_likes     0.28
         imdb_score                 0.26
         actor_2_facebook_likes     0.24
```

```
cast_total_facebook_likes    0.22
director_facebook_likes      0.14
actor_1_facebook_likes       0.13
aspect_ratio                 0.13
title_year                   0.03
facenumber_in_poster        -0.03
Name: gross, dtype: float64
```

The gross box office correlates strongly with num_voted_users, num_users_for_reviews and movie_facebook_likes. But some of those features are also highly correlated among each other (as you can see in the heatmap above).

## 0.5 Gross Box Office Prediction

### 0.5.1 Getting numerical data

In [34]:
```python
numerical_columns = data.dtypes[data.dtypes != 'object'].index
numerical_data = data[numerical_columns]

# we drop aspect_ratio, as it doesn't provide any useful info
numerical_data.drop('aspect_ratio', axis=1, inplace=True)
numerical_data.head(3)
```

Out[34]:
```
   num_critic_for_reviews  duration  director_facebook_likes  \
0                  723.00    178.00                     0.00
1                  302.00    169.00                   563.00
3                  813.00    164.00                22,000.00

   actor_3_facebook_likes  actor_1_facebook_likes          gross  \
0                  855.00                1,000.00 760,505,847.00
1                1,000.00               40,000.00 309,404,152.00
3               23,000.00               27,000.00 448,130,642.00

   num_voted_users  cast_total_facebook_likes  facenumber_in_poster  \
0           886204                       4834                  0.00
1           471220                      48350                  0.00
3          1144337                     106759                  0.00

   num_user_for_reviews         budget  title_year  actor_2_facebook_likes  \
0              3,054.00 237,000,000.00    2,009.00                  936.00
1              1,238.00 300,000,000.00    2,007.00                5,000.00
3              2,701.00 250,000,000.00    2,012.00               23,000.00

   imdb_score  movie_facebook_likes
0        7.90                 33000
1        7.10                     0
3        8.50                164000
```

### 0.5.2 Preparing train and test datasets

In [35]:
```
train, test = train_test_split(numerical_data, test_size=0.2)
target_train = train.pop('gross')
target_test = test.pop('gross')
```

In [36]:
```
print('Train data: {} / {} = {}'.format(len(train), len(numerical_data), float(len(trai
print('Test data: {} / {} = {}'.format(len(test), len(numerical_data), float(len(test)/
```

```
Train data: 2523 / 3154 = 0.7999365884590995
Test data: 631 / 3154 = 0.20006341154090043
```

### 0.5.3 Linear Regression

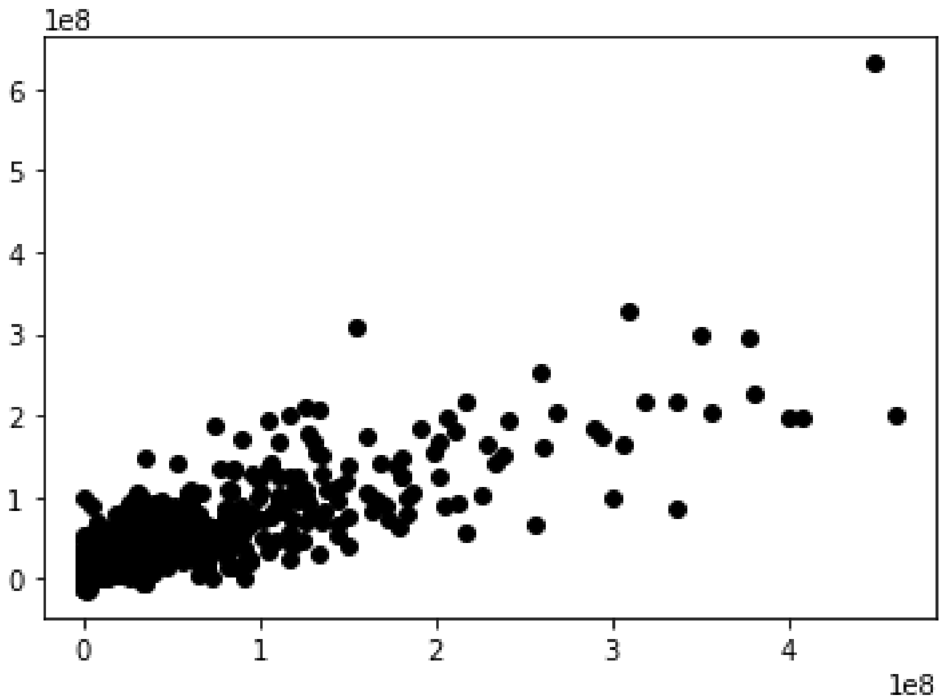
In [37]:
```
model = LinearRegression()
model.fit(train, target_train)
```

Out[37]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [38]:
```
prediction = model.predict(test)
```

In [411]:
```
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(target_test, prediction))

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(target_test, prediction))

# Plot outputs
plt.scatter(target_test, prediction,  color='black')
# plt.plot(test, prediction, color='blue', linewidth=3)


plt.show()
```

```
Mean squared error: 1922173864183296.50
Variance score: 0.62
```




### 0.5.4 Random Forest

```
In [47]: forest = RandomForestRegressor(
             max_depth=25,
             min_samples_split=15,
             n_estimators=1000,
             random_state=1)

         forest.fit(train, target_train)
```

```
Out[47]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=25,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_split=1e-07, min_samples_leaf=1,
                    min_samples_split=15, min_weight_fraction_leaf=0.0,
                    n_estimators=1000, n_jobs=1, oob_score=False, random_state=1,
                    verbose=0, warm_start=False)
```

```
In [48]: forest.feature_importances_
```

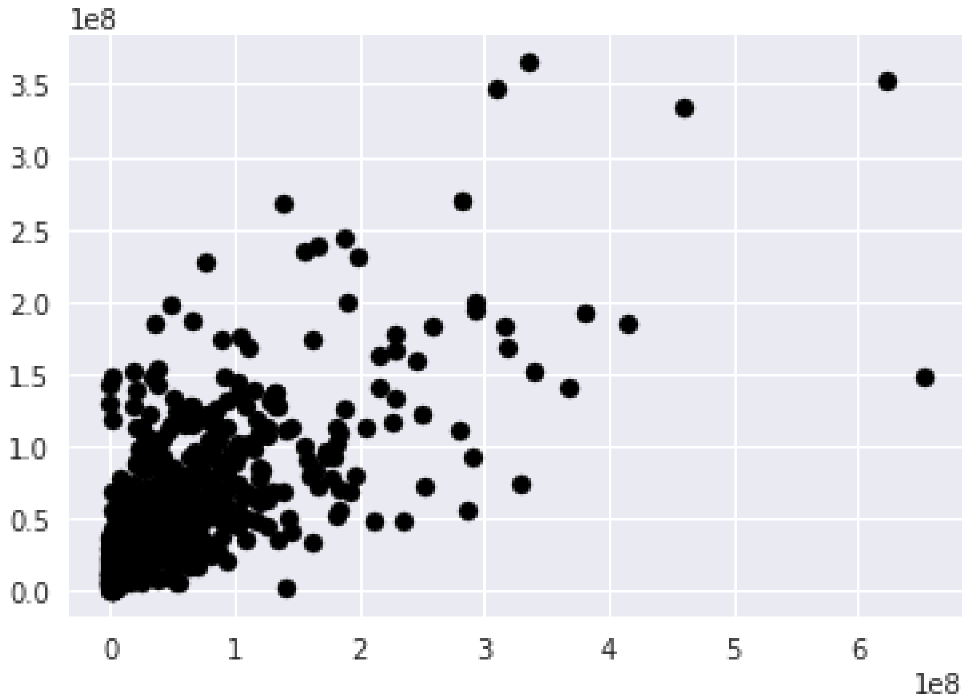
```
Out[48]: array([ 0.09460968,  0.0574659 ,  0.05838327,  0.03151676,  0.04906885,
                 0.00854848,  0.58500998,  0.06361421,  0.05178287])
```

```
In [49]: forest_prediction = forest.predict(test)
```

```
In [65]: # The mean squared error
         print("Mean squared error: %.2f" % mean_squared_error(target_test, forest_prediction))

         # Explained variance score: 1 is perfect prediction
         print('Variance score: %.2f' % r2_score(target_test, forest_prediction))

         # Plot outputs
         plt.scatter(target_test, forest_prediction,  color='black')
         # plt.plot(test, prediction, color='blue', linewidth=3)


         plt.show()
```

```
Mean squared error: 3009956585804540.50
Variance score: 0.48
```

```
/opt/conda/lib/python3.5/site-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: F
  (prop.get_family(), self.defaultFamily[fontext]))
```




## 0.6 Dropping post-fact data

There are post-fact variables in our data set making the prediction more accurate. Things like `num_voted_users` and `num_user_for_reviews` are after the fact metrics, so probably not as useful for prediction.

In [57]: train.head(2)

Out[57]:

```
      duration  director_facebook_likes  actor_3_facebook_likes  \
4462     95.00                12,000.00                  636.00
3467     95.00                   213.00                   92.00

      actor_1_facebook_likes  cast_total_facebook_likes  facenumber_in_poster  \
4462               12,000.00                      14420                  1.00
3467                  225.00                        791                  0.00

           budget  title_year  actor_2_facebook_likes
4462 1,300,000.00    1,996.00                  680.00
3467 4,000,000.00    1,983.00                  174.00
```

In [66]:

```
train.drop(['num_critic_for_reviews', 'num_voted_users', 'num_user_for_reviews', 'imdb_
test.drop(['num_critic_for_reviews', 'num_voted_users', 'num_user_for_reviews', 'imdb_s
train.head(2)
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-66-4f4aae4c6783> in <module>()
----> 1 train.drop(['num_critic_for_reviews', 'num_voted_users', 'num_user_for_reviews', 'im
      2 test.drop(['num_critic_for_reviews', 'num_voted_users', 'num_user_for_reviews', 'imd
      3 train.head(2)


/opt/conda/lib/python3.5/site-packages/pandas/core/generic.py in drop(self, labels, axis
   1905                 new_axis = axis.drop(labels, level=level, errors=errors)
   1906             else:
-> 1907                 new_axis = axis.drop(labels, errors=errors)
   1908             dropped = self.reindex(**{axis_name: new_axis})
   1909             try:


/opt/conda/lib/python3.5/site-packages/pandas/indexes/base.py in drop(self, labels, erro
   3260             if errors != 'ignore':
   3261                 raise ValueError('labels %s not contained in axis' %
-> 3262                                  labels[mask])
   3263             indexer = indexer[~mask]
   3264         return self.delete(indexer)


ValueError: labels ['num_critic_for_reviews' 'num_voted_users' 'num_user_for_reviews'
 'imdb_score' 'movie_facebook_likes'] not contained in axis
```

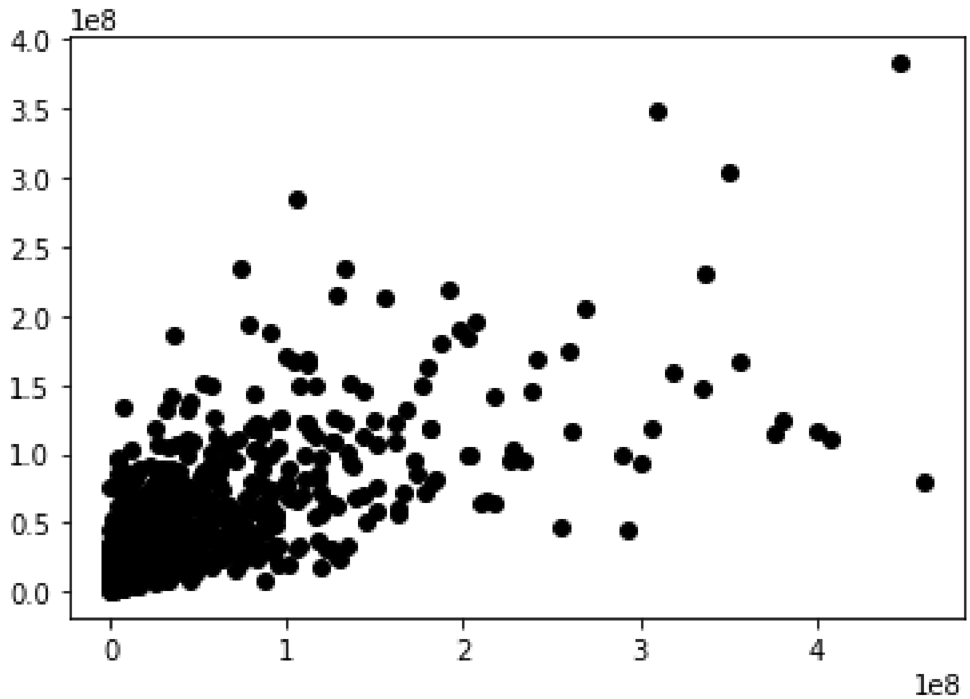
```
In [41]: pre_data_forest = RandomForestRegressor(
             max_depth=25,
             min_samples_split=15,
             n_estimators=1000,
             random_state=1)

         pre_data_forest.fit(train, target_train)
```

```
Out[41]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=25,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_split=1e-07, min_samples_leaf=1,
                    min_samples_split=15, min_weight_fraction_leaf=0.0,
                    n_estimators=1000, n_jobs=1, oob_score=False, random_state=1,
                    verbose=0, warm_start=False)
```

```
In [415]: second_prediction = pre_data_forest.predict(test)
```

```
In [416]: # The mean squared error
          print("Mean squared error: %.2f" % mean_squared_error(target_test, second_prediction))

          # Explained variance score: 1 is perfect prediction
          print('Variance score: %.2f' % r2_score(target_test, second_prediction))

          # Plot outputs
          plt.scatter(target_test, second_prediction,  color='black')
          # plt.plot(test, prediction, color='blue', linewidth=3)


          plt.show()
```

```
Mean squared error: 3013829195927549.50
Variance score: 0.41
```

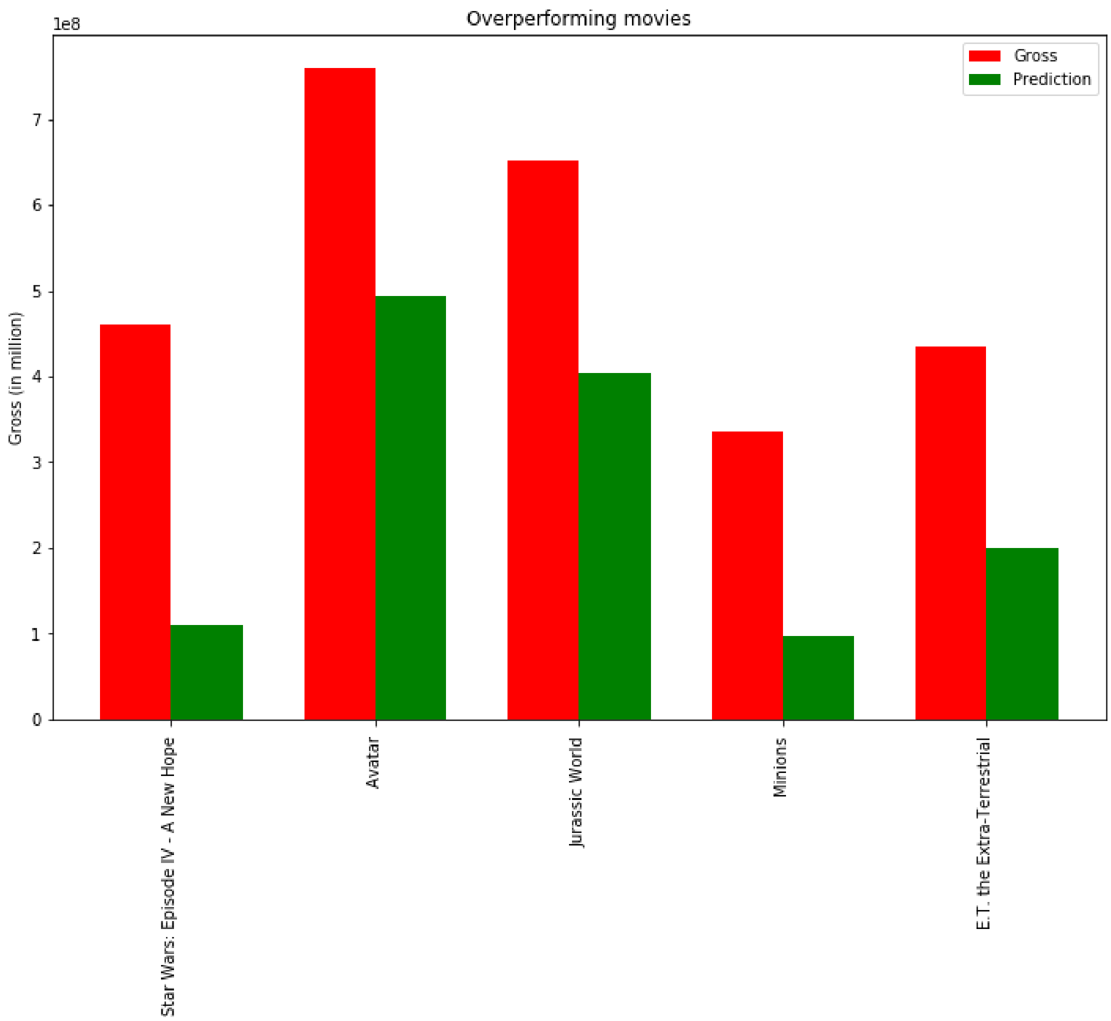



### 0.7 Over/Under performing movies

```
In [417]: numerical_data_target = numerical_data.pop('gross')
          all_data_prediction = forest.predict(numerical_data)

In [484]: performance_df = data.copy()

          performance_df["prediction"] = all_data_prediction
          performance_df["performance_diff"] = numerical_data_target - all_data_prediction

          performance_df.sort_values(['performance_diff'], ascending=False, inplace=True)

In [485]: ind = np.arange(5)
          width = 0.35

          fig, ax = plt.subplots(figsize=(12, 8))

          gross = ax.bar(ind, performance_df.gross[:5], width, color='r')
          predicted_gross = ax.bar(ind + width, performance_df.prediction[:5], width, color='g')

          plt.title("Overperforming movies")
          plt.ylabel("Gross (in million)")

          ax.set_xticks(ind + width / 2)
```

```
ax.set_xticklabels(performance_df.movie_title[:5], rotation='vertical')
ax.legend((gross[0], predicted_gross[0]), ('Gross', 'Prediction'))
```

Out[485]: <matplotlib.legend.Legend at 0x130e9be10>




In [59]:
```
performance_repr[:-6:-1]

ind = np.arange(5)
width = 0.35

fig, ax = plt.subplots(figsize=(12, 8))

gross = ax.bar(ind, performance_df.gross[:-6:-1], width, color='r')
predicted_gross = ax.bar(ind + width, performance_df.prediction[:-6:-1], width, color='

plt.title("Overperforming movies")
```

```
plt.ylabel("Gross (in million)")

ax.set_xticks(ind + width / 2)
ax.set_xticklabels(performance_df.movie_title[:-6:-1], rotation='vertical')
ax.legend((gross[0], predicted_gross[0]), ('Gross', 'Prediction'))
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-59-42fe547378f6> in <module>()
----> 1 performance_repr[:-6:-1]
      2
      3 ind = np.arange(5)
      4 width = 0.35
      5


NameError: name 'performance_repr' is not defined
```

```
In [ ]:
```