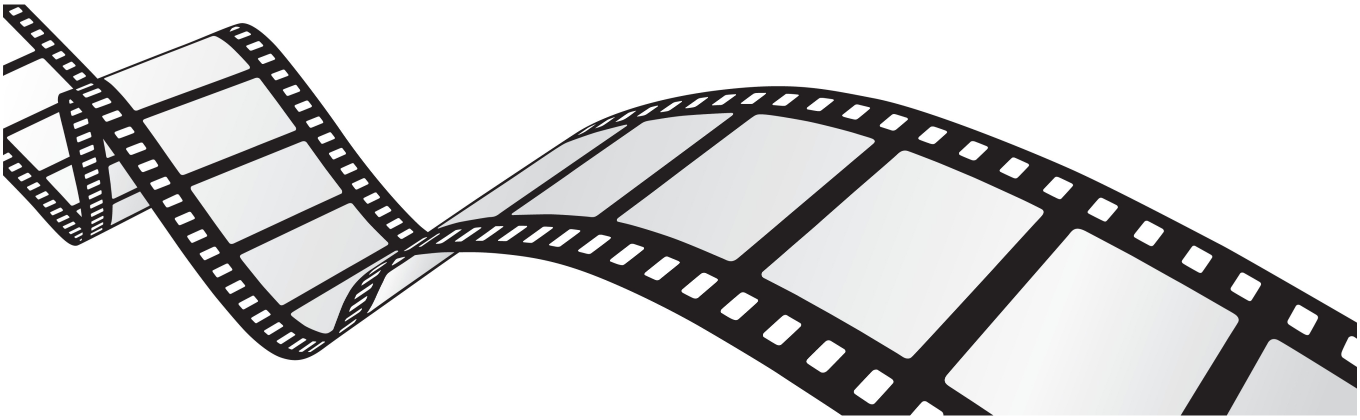


UDACITY Data Analysis Nanodegree

Project:- Movie Data Analysis

Grant Patience, 3rd July 2019

Table of Contents

1. Determine Objectives and Assess the Situation

For this project we will use the **CRISP-DM process (https://www.sv-europe.com/crisp-dm-methodology/)**. The first stage of the CRISP-DM process is to understand what you want to accomplish. The goal of this stage of the process is to uncover important factors that could influence the outcome of the project.

1.1 Outline of Steps

- In this section we discuss what it is we wish to achieve, decide which Questions we want to ask of the data and state what resources are available to us
- We will extract the data we need Data Extraction
- Import the data into Python for analysis to perform some Data Wrangling and Understanding to help us understand the data, and Verify Data Quality and resolve any data issues.
- Perform Exploratory Data Analysis where we will research the answers to our questions
- Create visualisations to aid exploration and research
- Draw our Conclusion based on the data and communicate our findings

1.2 What are the desired outputs of the project?

- Accurate project submission:

> **What to include in your submission** A PDF or HTML file containing your analysis. This file should include: A note specifying which dataset you analyzed A statement of the question(s) you posed A description of what you did to investigate those questions Documentation of any data wrangling you did Summary statistics and plots communicating your final results Code you used to perform your analysis. If you used a Jupyter notebook, you can submit your .ipynb. Otherwise, you should submit the code separately in .py file(s). A list of Web sites, books, forums, blog posts, github repositories, etc. that you referred to or used in creating your submission (add N/A if you did not use any such resources).

- Sucesfully answer all queries in "What Questions are We Trying to Answer?" Section
- Meet the Criteria of the Udacity Rubric:

Code Functionality

- Does the code work? All code is functional and produces no errors when run. The code given is sufficient to reproduce the results described.
- Does the project use NumPy and Pandas appropriately? The project uses NumPy arrays and Pandas Series and DataFrames where appropriate rather than Python lists and dictionaries. Where possible, vectorized operations and built-in functions are used instead of loops.
- Does the project use good coding practices? The code makes use of functions to avoid repetitive code. The code contains good comments and variable names, making it easy to read.

Quality of Analysis

- Is a question clearly posed? The project clearly states one or more questions, then addresses those questions in the rest of the analysis.

Data Wrangling Phase

- Is the data cleaning well documented? The project documents any changes that were made to clean the data, such as merging multiple files, handling missing values, etc.

Exploration Phase

- Is the data explored in many ways? The project investigates the stated question(s) from multiple angles. At least three variables are investigated using both single-variable (1d) and multiple-variable (2d) explorations.
- Are there a variety of relevant visualizations and statistical summaries? The project's visualizations are varied and show multiple comparisons and trends. Relevant statistics are computed throughout the analysis when an inference is made about the data.
- At least two kinds of plots should be created as part of the explorations.

Conclusions Phase

- Has the student correctly communicated tentativeness of findings? The results of the analysis are presented such that any limitations are clear. The analysis does not state or imply that one change causes another based solely on a correlation.

Communication

- Is the flow of the analysis easy to follow? Reasoning is provided for each analysis decision, plot, and statistical summary.
- Is the data visualized using appropriate plots and parameter choices? Visualizations made in the project depict the data in an appropriate manner that allows plots to be readily interpreted.

1.3 What Resources are Available?

- Dataset supplied (Details in Section [Data Extraction](#))
- Jupyter Python Notebook

1.4 What Questions Are We Trying To Answer?

Q1. How have the success of genres changed over time (Revenue/Rating)?

Q1.1 How many movies of a particular genre have been released?

Q1.2 Howhave the fortunes of the genres compared over time?

Q2. How succesful are different genres (Revenue/Rating)?

Q2.1 Which genres have the largest revenue and largest budgets?

Q2.2 Which genres are most profitable after working out Return on Investment?

Q2.3 Which genres are the most popular?

Q3. Which Directors are the most successful (Revenue/Rating)?

Q4. Which Attributes indicate a movie's chances of success (Revenue/Rating)?

---

2. Data Wrangling and Understanding

The second stage of the process is where we acquire the data listed in the project resources. Describe the methods used to acquire them and any problems encountered. We record problems you encountered and any resolutions achieved. Tis includes any data quality issues, and any resolution steps taken. This initial collection includes extraction details and source details, and subsequently loaded into Python and analysed in Jupyter notebook.

2.1 Data Description and Extraction

The data is a cleaned version of [this Kaggle (https://www.kaggle.com/tmdb/tmdb-movie-metadata/data)](https://www.kaggle.com/tmdb/tmdb-movie-metadata/data) data-set, containing around 10,000 movies collected from The Movie Database (TMDb)

Some of the features of the data are:

- This data set contains information about 10,000 movies collected from The Movie Database (TMDb), including user ratings and revenue.
- Certain columns, like ‘cast’ and ‘genres’, contain multiple values separated by pipe (|) characters.
- The final two columns ending with “_adj” show the budget and revenue of the associated movie in terms of 2010 dollars, accounting for inflation over time.

To extract the data, we can download from [Here (https://www.google.com/url?q=https://d17h27t6h515a5.cloudfront.net/topher/2017/October/59dd1c4c_tmdb-movies/tmdb-movies.csv&sa=D&ust=1532469042115000)](https://www.google.com/url?q=https://d17h27t6h515a5.cloudfront.net/topher/2017/October/59dd1c4c_tmdb-movies/tmdb-movies.csv&sa=D&ust=1532469042115000)

2.2 Describe Data's General Properties

In this section we describe the data that has been acquired including its format, its quantity (for example, the number of records and fields in each table), the identities of the fields and any other surface features which have been discovered. Evaluate whether the data acquired satisfies requirements.

In [1]:

```
# Import necessary libraries for initial data understanding, visualisations and explora
tory data analysis
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set_style('darkgrid')
```

In [2]:

```
# reads the data from the file - denotes as CSV, it has no header row, sets column head
ers
df_movie = pd.read_csv('Data/tmdb-movies.csv')
```

Now let's take our first look at the data.

In [3]:

```
df_movie.head(3)
```

Out[3]:

	id	imdb_id	popularity	budget	revenue	original_title	cast	
0	135397	tt0369610	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	
1	76341	tt1392190	28.419936	150000000	378436354	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic...	
2	262500	tt2908446	13.112507	110000000	295238201	Insurgent	Shailene Woodley|Theo James|Kate Winslet|Ansel...	http://v

3 rows × 21 columns

In [4]:

```
df_movie.tail(3)
```

Out[4]:

	id	imdb_id	popularity	budget	revenue	original_title	cast	home
10863	39768	tt0060161	0.065141	0	0	Beregis Avtomobilya	Innokentiy Smoktunovskiy|Oleg Efremov|Georgi Z...	
10864	21449	tt0061177	0.064317	0	0	What's Up, Tiger Lily?	Tatsuya Mihashi|Akiko Wakabayashi|Mie Hama|Joh...	
10865	22293	tt0060666	0.035919	19000	0	Manos: The Hands of Fate	Harold P. Warren|Tom Neyman|John Reynolds|Dian...	

3 rows × 21 columns

Looks sensible - we can see director, budgets, revnue, rating, titles, and other interesting features. Cast, production company and genre both look like they contain multiple entries seperated with the '|' character so we might want to deal with that. Popularity looks to be an arbitrary floating point number rating system, we'll need to research what what means.

Now lets take a look at some of the general properties of the data:

In [5]:

```
df_movie.shape
```

Out[5]:

```
(10866, 21)
```

In [6]:

```
df_movie.columns
```

Out[6]:

```
Index(['id', 'imdb_id', 'popularity', 'budget', 'revenue', 'original_titl
e',
       'cast', 'homepage', 'director', 'tagline', 'keywords', 'overview',
       'runtime', 'genres', 'production_companies', 'release_date',
       'vote_count', 'vote_average', 'release_year', 'budget_adj',
       'revenue_adj'],
      dtype='object')
```

In [7]:

```
df_movie.nunique()
```

Out[7]:

```
id                      10865
imdb_id                 10855
popularity              10814
budget                    557
revenue                  4702
original_title          10571
cast                    10719
homepage                 2896
director                 5067
tagline                  7997
keywords                 8804
overview                10847
runtime                   247
genres                   2039
production_companies     7445
release_date             5909
vote_count               1289
vote_average               72
release_year               56
budget_adj               2614
revenue_adj              4840
dtype: int64
```

In [8]:

```
df_movie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10866 entries, 0 to 10865
Data columns (total 21 columns):
id                      10866 non-null int64
imdb_id                 10856 non-null object
popularity              10866 non-null float64
budget                  10866 non-null int64
revenue                 10866 non-null int64
original_title          10866 non-null object
cast                    10790 non-null object
homepage                2936 non-null object
director                10822 non-null object
tagline                 8042 non-null object
keywords                9373 non-null object
overview                10862 non-null object
runtime                 10866 non-null int64
genres                  10843 non-null object
production_companies    9836 non-null object
release_date            10866 non-null object
vote_count              10866 non-null int64
vote_average            10866 non-null float64
release_year            10866 non-null int64
budget_adj              10866 non-null float64
revenue_adj             10866 non-null float64
dtypes: float64(4), int64(6), object(11)
memory usage: 1.7+ MB
```

- We can now see that there are 10865 entries 21 columns.
- Some attributes look like they will be useful for our research - we know the **movie, release year, director cast, budget, revenue, rating** which are all key to our research.
- Some of them contain a few missing values such as **cast and director**. While there are other attributes containing many more missing values such as **homepage, tagline, keywords and production_companies**.
- We will look at these missing values more in the Data Quality section
- Columns **imdb_id, homepage, tagline, overview** look to be interesting but not of much value in the research required here, so may be worth dropping

We can now start to take a look at some of the general statistics of the data:

In [9]:

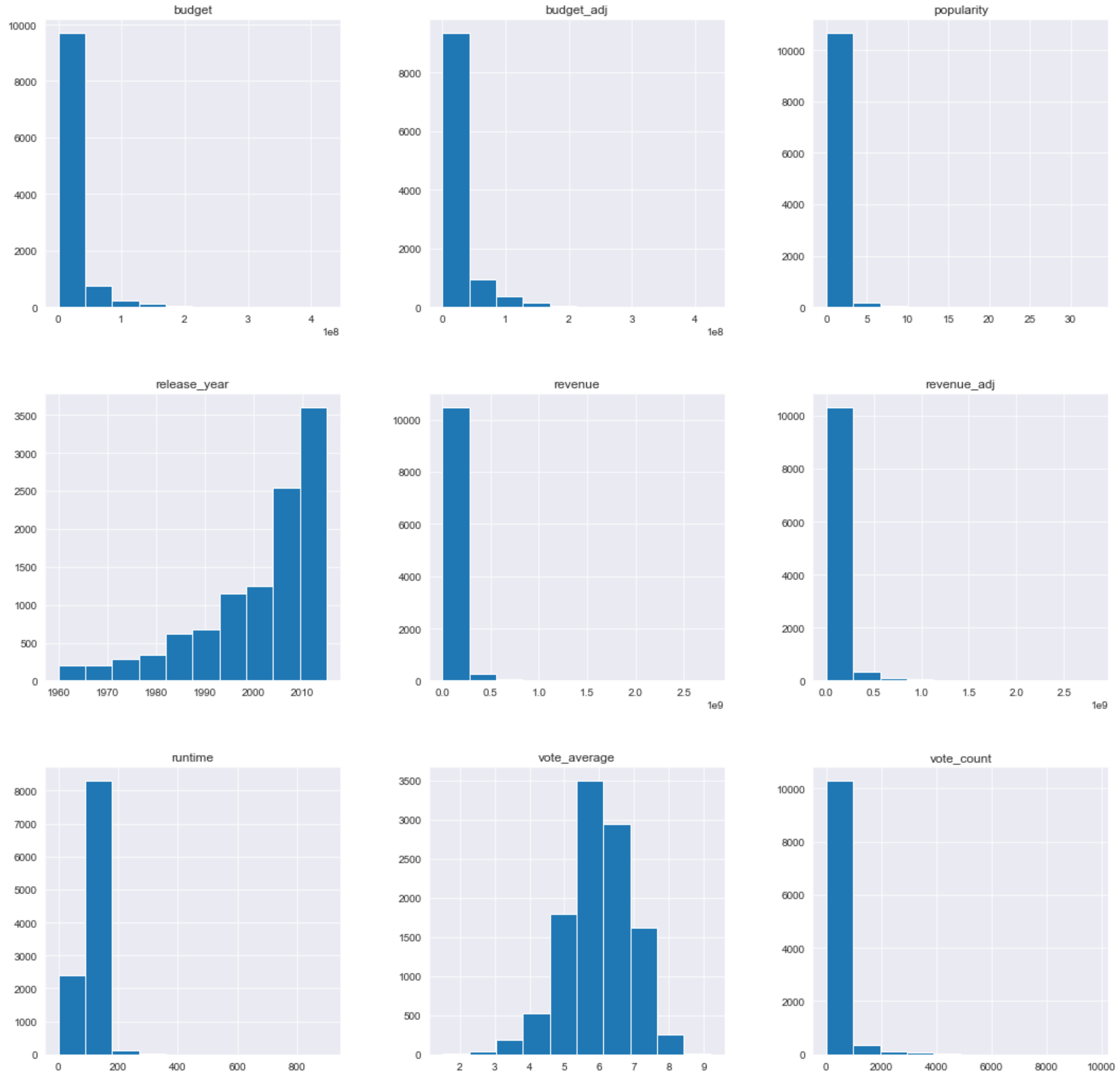
```
df_movie.describe()
```

Out[9]:

	id	popularity	budget	revenue	runtime	vote_count
count	10866.000000	10866.000000	1.086600e+04	1.086600e+04	10866.000000	10866.000000
mean	66064.177434	0.646441	1.462570e+07	3.982332e+07	102.070863	217.389748
std	92130.136561	1.000185	3.091321e+07	1.170035e+08	31.381405	575.619058
min	5.000000	0.000065	0.000000e+00	0.000000e+00	0.000000	10.000000
25%	10596.250000	0.207583	0.000000e+00	0.000000e+00	90.000000	17.000000
50%	20669.000000	0.383856	0.000000e+00	0.000000e+00	99.000000	38.000000
75%	75610.000000	0.713817	1.500000e+07	2.400000e+07	111.000000	145.750000
max	417859.000000	32.985763	4.250000e+08	2.781506e+09	900.000000	9767.000000

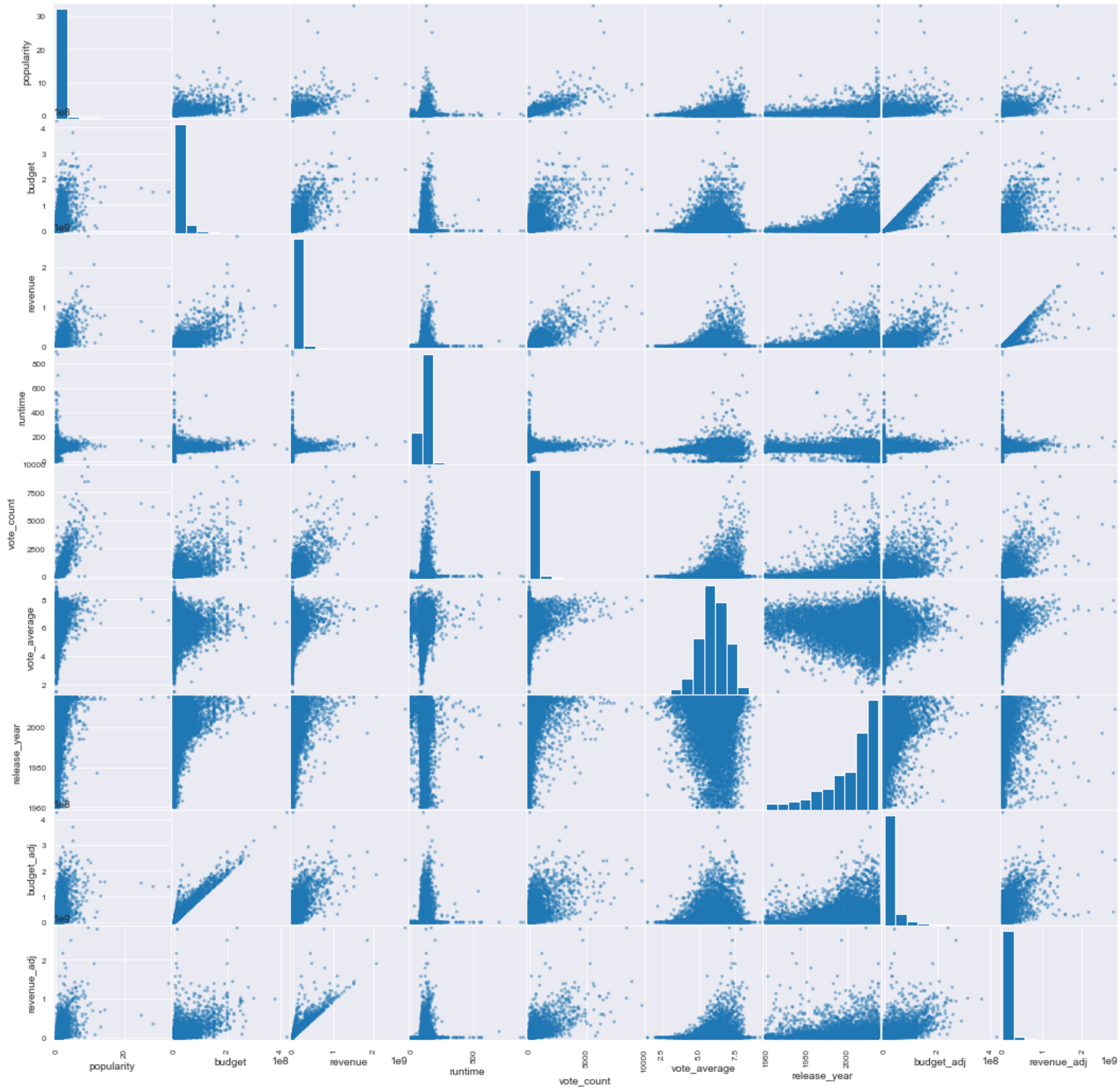
In [10]:

```
# Lets get a quick Histogram plot up and running.
# We want to ignore the id column since it's not a relevant for plotting

df_movie.drop(['id'], axis=1).hist(figsize=( 18,18));
```



In [11]:

```
pd.plotting.scatter_matrix(df_movie.drop(['id'], axis=1), figsize=(18,18));
```

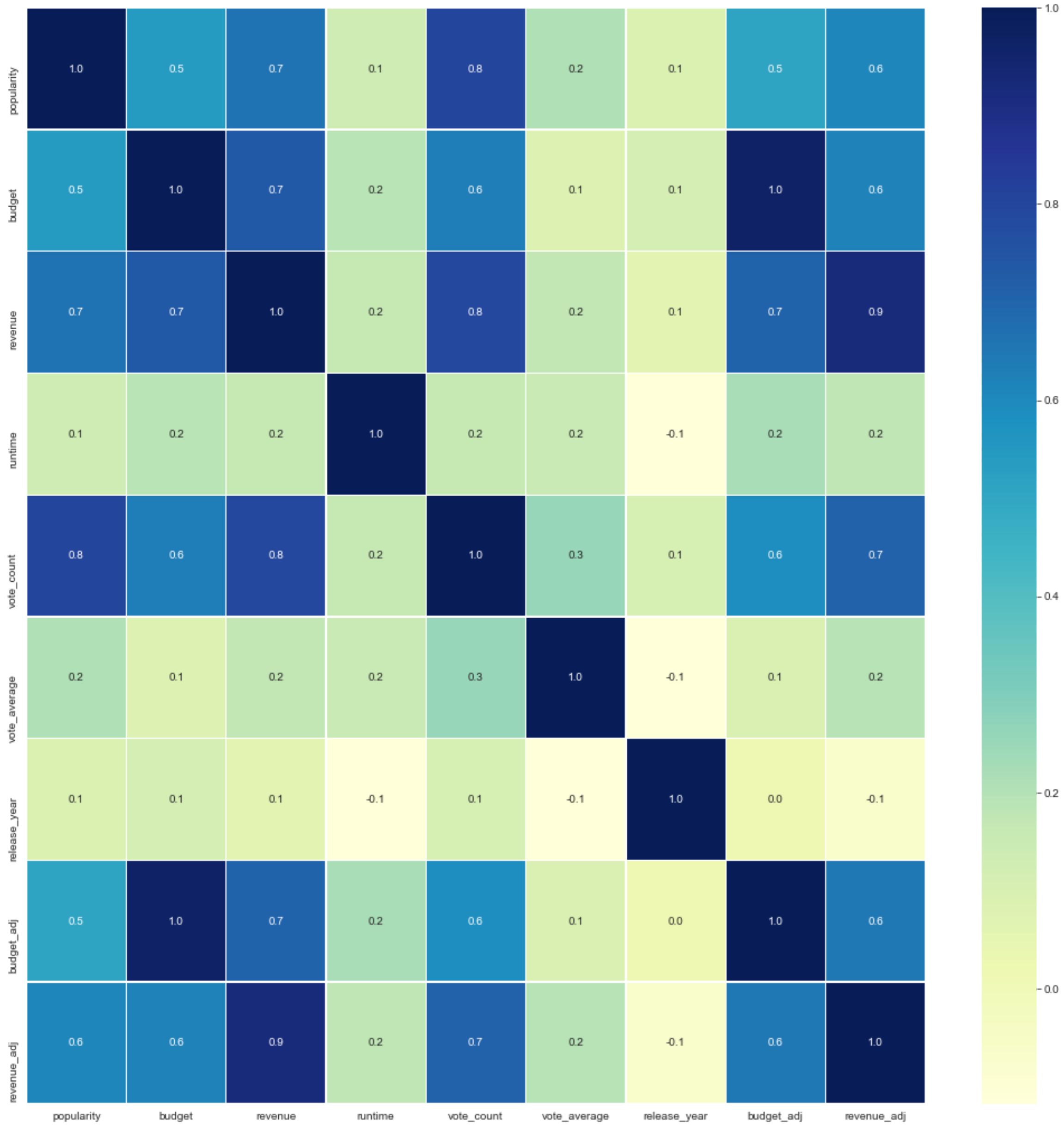


In [12]:

```
df_corr = df_movie.drop(['id'], axis=1).corr()
```

In [13]:

```
f, ax= plt.subplots(figsize=(18,18))
sns.heatmap(df_corr, annot=True, linewidths=.3, cmap="YlGnBu", fmt='.1f', ax=ax)
plt.show()
```



> Looks like the distributions are showing a lot of 0 values for revenue and budget and their respective adjusted

2.3 Verify Data Quality

Examine the quality of the data, addressing questions such as:

- Is the data complete (does it cover all the cases required)?
- Is it correct, or does it contain errors and, if there are errors, how common are they?
- Are there missing values in the data? If so, how are they represented, where do they occur, and how common are they?

2.3.1. Missing Data

In addition to incorrect datatypes, another common problem when dealing with real-world data is missing values. These can arise for many reasons and have to be either filled in or removed before we train a machine learning model. First, let's get a sense of how many missing values are in each column

While we always want to be careful about removing information, if a column has a high percentage of missing values, then it probably will not be useful to our model. The threshold for removing columns should depend on the problem

In [14]:

```
# Function that will take an input table with aggregated values to columns, and then cr
eate an output table with
# two columns - the values and the percentage of total values in that column
def values_table(data):
        val = data
        val_percent = 100 * val / len(data)
        val_table = pd.concat([val, val_percent], axis=1)
        val_table_ren_columns = val_table.rename(
        columns = {0 : 'Values', 1 : '% of Total Values'})
        val_table_ren_columns = val_table_ren_columns[
            val_table_ren_columns.iloc[:,0] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)
        #print ("Your selected dataframe has " + str(data.shape[1]) + " columns.\n"
         #   "There are " + str(val_table_ren_columns.shape[0]) +
          #    " columns that have the values you filtered for.")
        return val_table_ren_columns
```

In [15]:

```
values_table(df_movie.isnull().sum() )
```

Out[15]:

	Values	% of Total Values
homepage	7930	37761.9
tagline	2824	13447.6
keywords	1493	7109.5
production_companies	1030	4904.8
cast	76	361.9
director	44	209.5
genres	23	109.5
imdb_id	10	47.6
overview	4	19.0

Options

- We may want to remove null rows entirely from the dataset. To do so we would run something like the following

```
df.dropna()
```

- We may want to drop the columns if they appear to be predominantly NA. To do so we would run something like the following

```
# Get the columns with > 50% missing
missing_df = missing_values_table(df);
missing_columns = list(missing_df[missing_df['% of Total Values'] > 50].inde
x)
print('We will remove %d columns.' % len(missing_columns))
df = df.drop(list(missing_columns))
```

- We may want to fill the missing values with the mean values from the dataset. To do so we would run something like the following

```
mean = df['x'].mean()
df['x'].fillna(mean, inplace=True)
```

Decision

- We will drop columns imdb_id, homepage, tagline since we identified these as low importance for our research purposes anyway.
- We will keep keywords and production_companies as they may be interesting for research
- We need casts, director and genres for our analysis though - we can't infer these values from the data. We'll decide to drop rows with no values for these so that it doesn't affect analysis

2.3.2. Outliers

At this point, we may also want to remove outliers. These can be due to typos in data entry, mistakes in units, or they could be legitimate but extreme values. For this project, we will remove anomalies based on the definition of extreme outliers:

We noticed a lot of 0s in the histograms earlier, let's get some exact figures

In [16]:

```
values_table((df_movie == 0).sum() )
```

Out[16]:

	Values	% of Total Values
revenue	6016	28647.6
revenue_adj	6016	28647.6
budget	5696	27123.8
budget_adj	5696	27123.8
runtime	31	147.6

Options

- We may want to remove these rows entirely from the dataset. To do so we would run something like the following

```
df[df.budget = 0]
```

- We may want to drop the columns if they appear to be unreliable. To do so we would run something like the following

```
# Get the columns with > 50% missing
missing_df = missing_values_table(df);
missing_columns = list(missing_df[missing_df['% of Total Values'] > 50].inde
x)
print('We will remove %d columns.' % len(missing_columns))
df = df.drop(list(missing_columns))
```

- We may want to change the values with the mean values from the dataset, or another value of our choosing. This maintains the data for some analysis, but will not impact other analysis by having the erroneous value presnt. To do so we would run something like the following

```
mean = df['x'].mean()
df['x'].fillna(mean, inplace=True)
```

Decision

- In this case, we do not want missing values for budget, budget_adj, revenu and revenue_adj affecting the research for our analysis, so we will make these values NULL instead
- That leaves runtime, which might be of value as one of our attributes in our research. Since it might be an important attribute and the number of affected rows is low, we will drop these rows

2.3.3. Duplicates

There may be duplicates in the data. However, these may be legitimate new rows depending on the structure of the data. We need to discover them, then decide what to do with them

In [17]:

```
sum(df_movie.duplicated())
```

Out[17]:

1

Options We may want to remove duplicate rows entirely from the dataset. To do so we would run the following

```
df.drop_duplicates(inplace=True)
```

Decision

- There is only one duplicated row, so we will go ahead and drop this row

Data Summary Report

Category	Description	Decision
Row count	10866	N/A
Column count	21	N/A
Un-Meaningful Columns	imdb_id, homepage, tagline, overview are of little use to our analysis	Drop columns from analysis
Missing Values	imdb_id, homepage, tagline have missing values in a substantial number of cases	Drop columns from analysis
Missing Values	casts, director and genres	Drop affected rows from analysis
Outliers	budget, budget_adj, revenu and revenue_adj have high amount of 0s	We will change the value to NULL for these columns
Outliers	runtime has a number of 0s	We will drop these rows
Duplicates	1 Duplicates found	Drop the duplicated record

Data Cleansing

Drop Un-Meaningful Columns

In [18]:

```
# Drop extraneous columns
columns = ['imdb_id', 'homepage', 'tagline', 'overview']
df_movie.drop(columns, axis=1, inplace=True)
```

In [19]:

```
# Confirm action
df_movie.head(3)
```

Out[19]:

	id	popularity	budget	revenue	original_title	cast	director	
0	135397	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	Colin Trevorrow	
1	76341	28.419936	150000000	378436354	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic...	George Miller	
2	262500	13.112507	110000000	295238201	Insurgent	Shailene Woodley|Theo James|Kate Winslet|Ansel...	Robert Schwentke	novel

Drop Null values

In [20]:

```
#drop the null values in cast, director, genres columns
columns = ['cast', 'director', 'genres']
df_movie.dropna(subset = columns, how='any', inplace=True)
```

In [21]:

```
# Confirm rows dropped
df_movie.isnull().sum()
```

Out[21]:

```
id                         0
popularity                 0
budget                     0
revenue                    0
original_title             0
cast                       0
director                   0
keywords                1425
runtime                    0
genres                     0
production_companies     959
release_date               0
vote_count                 0
vote_average               0
release_year               0
budget_adj                 0
revenue_adj                0
dtype: int64
```

Change Outlier values

In [22]:

```
df_movie['budget'] = df_movie['budget'].replace(0, np.NaN)
df_movie['revenue'] = df_movie['revenue'].replace(0, np.NaN)
df_movie['budget_adj'] = df_movie['budget_adj'].replace(0, np.NaN)
df_movie['revenue_adj'] = df_movie['revenue_adj'].replace(0, np.NaN)
```

In [23]:

```
# Confirm values changed
df_movie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10732 entries, 0 to 10865
Data columns (total 17 columns):
id                      10732 non-null int64
popularity              10732 non-null float64
budget                  5154 non-null float64
revenue                 4844 non-null float64
original_title          10732 non-null object
cast                    10732 non-null object
director                10732 non-null object
keywords                9307 non-null object
runtime                 10732 non-null int64
genres                  10732 non-null object
production_companies    9773 non-null object
release_date            10732 non-null object
vote_count              10732 non-null int64
vote_average            10732 non-null float64
release_year            10732 non-null int64
budget_adj              5154 non-null float64
revenue_adj             4844 non-null float64
dtypes: float64(6), int64(4), object(7)
memory usage: 1.5+ MB
```

Remove the rows with 0 runtime

In [24]:

```
df_movie = df_movie.query('runtime != 0')
```

In [25]:

```
# directly filter the runtime data with nonzero value
df_movie.query('runtime == 0')
```

Out[25]:

	id	popularity	budget	revenue	original_title	cast	director	keywords	runtime	genres

Drop duplicate records

In [26]:

```
df_movie.drop_duplicates(inplace=True)
```

Results

In [27]:

```
df_movie.shape
```

Out[27]:

```
(10703, 17)
```

In [28]:

```
df_movie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10703 entries, 0 to 10865
Data columns (total 17 columns):
id                      10703 non-null int64
popularity              10703 non-null float64
budget                  5150 non-null float64
revenue                 4843 non-null float64
original_title          10703 non-null object
cast                    10703 non-null object
director                10703 non-null object
keywords                9293 non-null object
runtime                 10703 non-null int64
genres                  10703 non-null object
production_companies    9759 non-null object
release_date            10703 non-null object
vote_count              10703 non-null int64
vote_average            10703 non-null float64
release_year            10703 non-null int64
budget_adj              5150 non-null float64
revenue_adj             4843 non-null float64
dtypes: float64(6), int64(4), object(7)
memory usage: 1.5+ MB
```

In [ ]:

```

```

In [29]:

```
df_movie.describe()
```

Out[29]:

	id	popularity	budget	revenue	runtime	vote_count
count	10703.000000	10703.000000	5.150000e+03	4.843000e+03	10703.000000	10703.000000
mean	64904.988321	0.653818	3.084401e+07	8.933981e+07	102.736896	220.333178
std	91161.996308	1.005687	3.893782e+07	1.621546e+08	30.079331	579.481969
min	5.000000	0.000188	1.000000e+00	2.000000e+00	3.000000	10.000000
25%	10538.500000	0.211533	6.000000e+06	7.779664e+06	90.000000	17.000000
50%	20235.000000	0.388036	1.750000e+07	3.191160e+07	99.000000	39.000000
75%	73637.000000	0.722438	4.000000e+07	1.000000e+08	112.000000	149.000000
max	417859.000000	32.985763	4.250000e+08	2.781506e+09	900.000000	9767.000000

3. Exploratory Data Analysis

Q1. How have the success of genres changed over time (Revenue/Rating)?

First of all, lets take a look at the popularty of genres throughout the years in the dataset. Here, we will judge popularity as the commonality of the releases.

In [30]:

```
df_movie.genres.unique()
```

Out[30]:

```
array(['Action|Adventure|Science Fiction|Thriller',
       'Adventure|Science Fiction|Thriller',
       'Action|Adventure|Science Fiction|Fantasy', ...,
       'Adventure|Drama|Action|Family|Foreign',
       'Comedy|Family|Mystery|Romance',
       'Mystery|Science Fiction|Thriller|Drama'], dtype=object)
```

Ah! We forgot that some of the columns contain multiple entries delimitted by '|'. Lets use the split function to split out the rows

In [31]:

```
# Check original row count
df_movie.shape
```

Out[31]:

```
(10703, 17)
```

In [32]:

```
df_movie_genre = df_movie
# columns to split by "|"
df_movie_genre['genres'] = df_movie['genres'].apply(lambda x: x.split("|")[0])
```

In [34]:

```
# Check new row count
df_movie_genre.shape
```

Out[34]:

```
(10703, 17)
```

In [35]:

```
# Confirm the action worked and split Genres out
df_movie_genre.genres.unique()
```

Out[35]:

```
array(['Action', 'Adventure', 'Western', 'Science Fiction', 'Drama',
       'Family', 'Comedy', 'Crime', 'Romance', 'War', 'Mystery',
       'Thriller', 'Fantasy', 'History', 'Animation', 'Horror', 'Music',
       'Documentary', 'TV Movie', 'Foreign'], dtype=object)
```

In [36]:

```
df_movie_genre.head(5)
```

Out[36]:

	id	popularity	budget	revenue	original_title	cast	director
0	135397	32.985763	150000000.0	1.513529e+09	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	Colin Trevorrow
1	76341	28.419936	150000000.0	3.784364e+08	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic...	George Miller
2	262500	13.112507	110000000.0	2.952382e+08	Insurgent	Shailene Woodley|Theo James|Kate Winslet|Ansel...	Robert Schwentke
3	140607	11.173104	200000000.0	2.068178e+09	Star Wars: The Force Awakens	Harrison Ford|Mark Hamill|Carrie Fisher|Adam D...	J.J. Abrams
4	168259	9.335014	190000000.0	1.506249e+09	Furious 7	Vin Diesel|Paul Walker|Jason Statham|Michelle ...	James Wan

That looks more like it!!

Now if we want to research genre trends over time, lets count the number of movies aggregated by genre over the years using the groupby function, assign it to a new dataframe and view the results

Q1.1 How many movies of a particular genre have been released?

In [37]:

```
df_genres_year = df_movie_genre.groupby(['release_year', 'genres']).count()['id'].unsta
ck()
```

In [38]:

```
df_genres_year.head(5)
```

Out[38]:

genres	Action	Adventure	Animation	Comedy	Crime	Documentary	Drama	Family
release_year								
1960	8.0	2.0	NaN	7.0	1.0	NaN	5.0	NaN
1961	3.0	2.0	NaN	8.0	NaN	NaN	7.0	NaN
1962	5.0	4.0	NaN	2.0	3.0	NaN	11.0	NaN
1963	3.0	5.0	1.0	9.0	NaN	NaN	7.0	NaN
1964	2.0	5.0	2.0	10.0	5.0	NaN	10.0	NaN

In [39]:

```
df_movie_genre.groupby(['genres']).count()['id'].sort_values(ascending=False)
```

Out[39]:

```
genres
Drama              2439
Comedy             2307
Action             1586
Horror              909
Adventure           585
Thriller            489
Documentary         385
Crime               380
Animation           375
Fantasy             270
Science Fiction     211
Romance             182
Family              141
Mystery             125
Music                95
TV Movie             72
War                  58
History              44
Western              42
Foreign               8
Name: id, dtype: int64
```

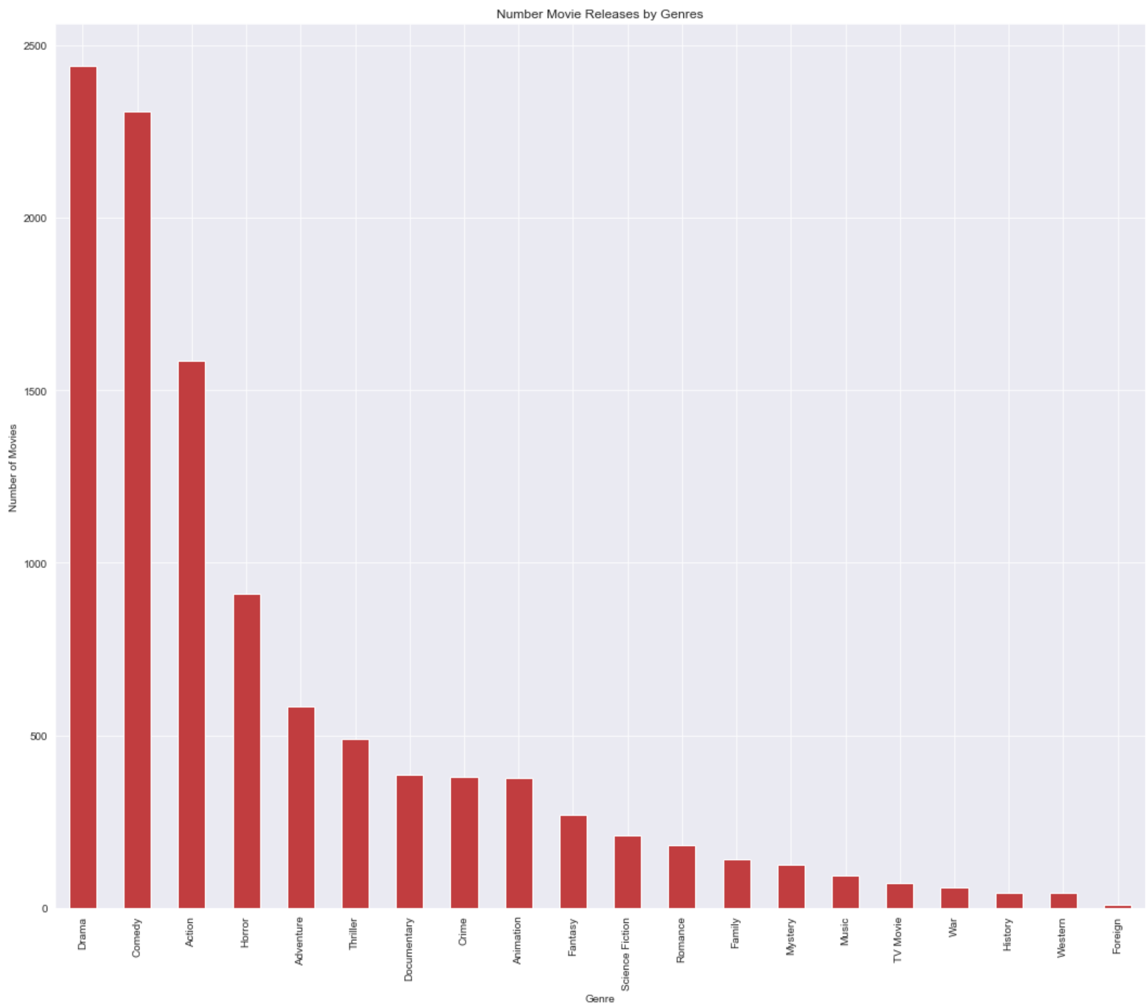
This looks good! Now lets plot this to find the total number of releases for that genre.

In [40]:

```
sns.set_style('darkgrid')

# plot data
fig, ax = plt.subplots(figsize=(18,15))

sns.set_palette("Set1", 20, .65)

# use unstack()
df_movie_genre.groupby(['genres']).count()['id'].sort_values(ascending=False).plot(kind
="bar",  ax=ax);
ax.set(xlabel='Genre', ylabel='Number of Movies', title = 'Number Movie Releases by Gen
res')
```

Out[40]:

```
[Text(0, 0.5, 'Number of Movies'),
 Text(0.5, 0, 'Genre'),
 Text(0.5, 1.0, 'Number Movie Releases by Genres')]
```



We find that Drama is the most frequent genre of film, followed by comedy and action.

Q1.2 How have the fortunes of the genres compared over time?

We now have our genre types, we have our total releases - but how do those releases compare over time?

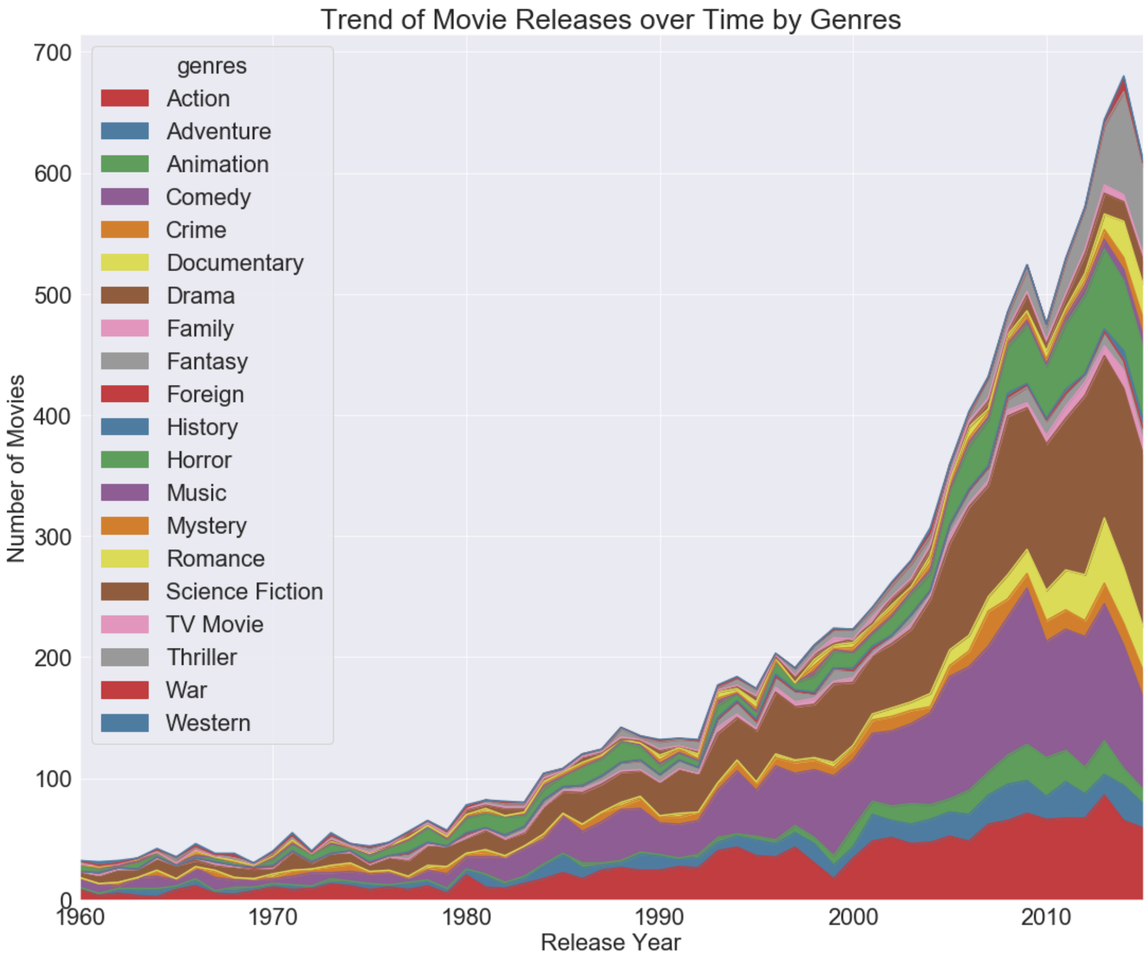
Lets group by release year and genre, and compare using an area chart

In [41]:

```
#Set global font size
plt.rcParams.update({'font.size': 22})
sns.set_style('darkgrid')

# plot data
fig, ax = plt.subplots(figsize=(18,15))

sns.set_palette("Set1", 20, .65)

# use unstack()
df_movie_genre.groupby(['release_year', 'genres']).count()['id'].unstack().plot.area(ax
=ax);
ax.set(xlabel='Release Year', ylabel='Number of Movies', title = 'Trend of Movie Releas
es over Time by Genres')
```

Out[41]:

```
[Text(0, 0.5, 'Number of Movies'),
 Text(0.5, 0, 'Release Year'),
 Text(0.5, 1.0, 'Trend of Movie Releases over Time by Genres')]
```



The **popularit**y of movie releases has generally grown over time, from 1960 til around 2010 where the numbers contract slightlty. Picking out specific genres, Drama, Thriller, Comedy and Action movies seem to be the predominant genres.

This gives us the commonality of movies over time, inferring there is a demand by the public for these movies, but how **Succesfull** are these genres?

If we use revenue and average rating as our attributes to guage success, let's try and incorporate those into our analysis

In [42]:

```
genre_year = df_movie_genre.groupby(['genres', 'release_year']).mean().sort_index()
```

In [43]:

```
genre_year.head(5)
```

Out[43]:

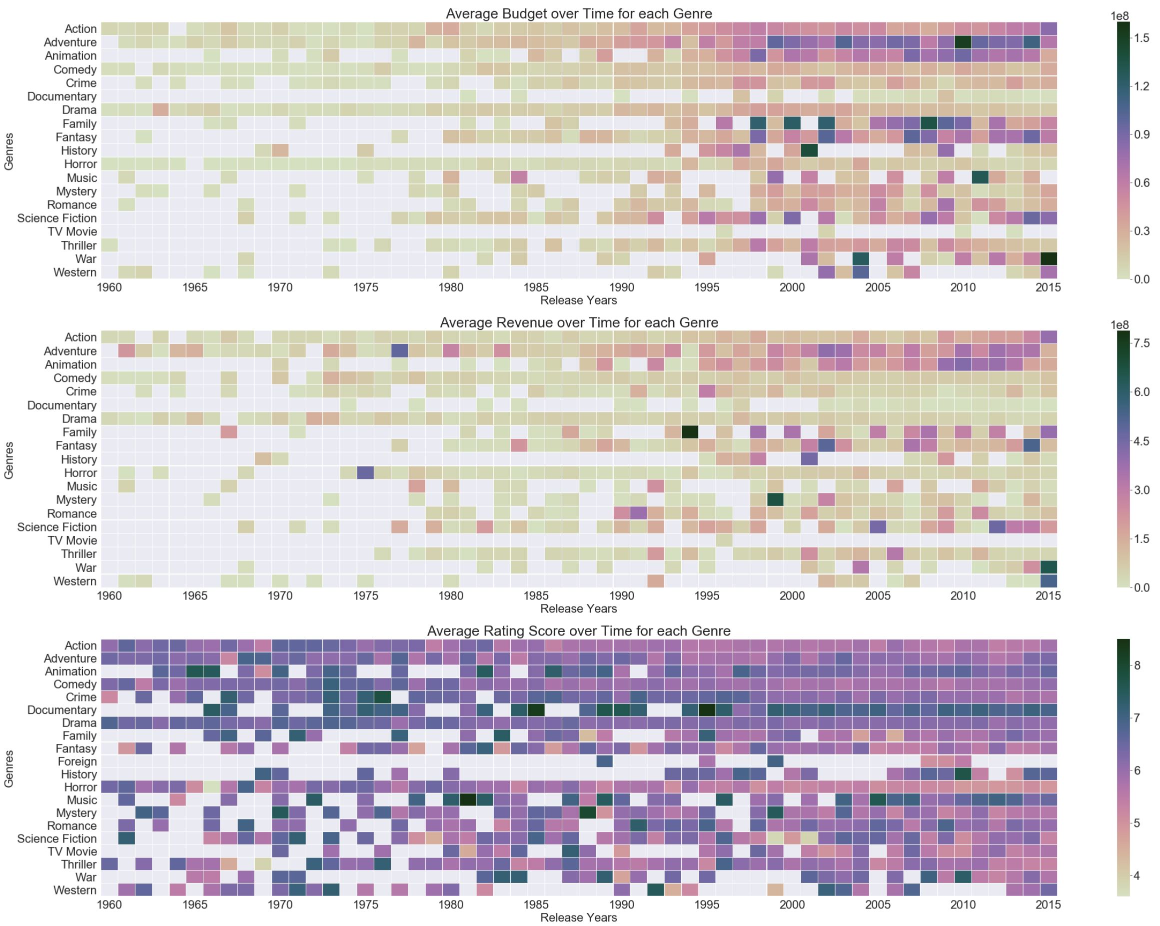
		id	popularity	budget	revenue	runtime	vote_count
genres	**release_year**						
Action	**1960**	7602.250000	0.590724	7000000.0	32452500.0	137.0	65.875000
	1961	15514.000000	0.540904	6000000.0	28900000.0	147.0	43.333333
	1962	24739.000000	0.299207	10000000.0	NaN	123.4	38.000000
	1963	17721.333333	1.008599	9750000.0	44449382.5	127.0	166.666667
	1964	18975.500000	0.254216	NaN	NaN	128.0	40.000000

This gives us a table of each genre over every release year, and gives us the mean of each attribute - we are interested in vote_average and revenue in particular.

Let's plot these into a time series heat map to guage the change over time per genre...

In [44]:

```
df_gyBudget = genre_year.pivot_table(index=['genres'], columns=['release_year'], values
='budget', aggfunc=np.mean)
df_gyGross = genre_year.pivot_table(index=['genres'], columns=['release_year'], values=
'revenue', aggfunc=np.mean)
df_gyVote = genre_year.pivot_table(index=['genres'], columns=['release_year'], values=
'vote_average', aggfunc=np.mean)
f, [axA, axB, axC] = plt.subplots(figsize=(40, 30), nrows=3)
cmap = sns.cubehelix_palette(start=1.3, rot=1.3, as_cmap=True)
sns.heatmap(df_gyBudget, xticklabels=5, cmap=cmap, linewidths=0.05, ax=axA)
sns.heatmap(df_gyGross, xticklabels=5, cmap=cmap, linewidths=0.05, ax=axB)
sns.heatmap(df_gyVote, xticklabels=5, cmap=cmap, linewidths=0.05, ax=axC)
axA.set_title('Average Budget over Time for each Genre')
axB.set_title('Average Revenue over Time for each Genre')
axC.set_title('Average Rating Score over Time for each Genre')
axA.set_xlabel('Release Years')
axA.set_ylabel('Genres')
axB.set_xlabel('Release Years')
axB.set_ylabel('Genres')
axC.set_xlabel('Release Years')
axC.set_ylabel('Genres')
plt.show()
```



Based on the heatmaps, looks like those trends are confirmed. Some interesting snippets we can see are that revenues and budgets have generally grown over time. This would make sense since we found that more films are being relesed. Budget & Revenue for the Fantasy genre has shown a marked increase since around the year 2000, this would coincide with the Lord of the Rings trilogy release presumably the capabilities of production companies to bring such fantastical elements to life through CGI.

That doesn't necessarily mean average ratings are increasing though, in particular you can see the average ratings of Horror movies has declined over time, while the quality of Dramas has increased.

Q2. How succesful are different genres (Revenue/Rating)?

First, how would we define success? There could be two ways to interpret this - by return on investment or user rating.

Let's take a look at both

In [45]:

```
df_movie[['genres', 'revenue', 'budget', 'popularity', 'vote_average']].groupby(['genre
s']).mean()
```

Out[45]:

	revenue	budget	popularity	vote_average
genres				
Action	1.170983e+08	4.231958e+07	0.838266	5.751009
Adventure	2.071020e+08	6.477152e+07	1.219834	6.049744
Animation	2.399754e+08	6.229486e+07	0.853208	6.401867
Comedy	6.471663e+07	2.320338e+07	0.539358	5.880971
Crime	6.167864e+07	2.271513e+07	0.694063	6.217632
Documentary	9.839752e+06	3.690254e+06	0.184708	6.916623
Drama	5.334542e+07	2.149751e+07	0.554855	6.198524
Family	1.807031e+08	5.275776e+07	0.744438	5.941844
Fantasy	1.345879e+08	4.588527e+07	0.864781	5.789630
Foreign	NaN	NaN	0.178917	5.687500
History	9.294606e+07	3.186905e+07	0.764636	6.381818
Horror	4.763156e+07	1.137138e+07	0.470718	5.320902
Music	6.064779e+07	2.843784e+07	0.465062	6.568421
Mystery	5.807465e+07	2.339429e+07	0.596896	5.900800
Romance	8.389153e+07	2.494572e+07	0.717200	6.151099
Science Fiction	1.654990e+08	4.478218e+07	1.087261	5.938389
TV Movie	4.200000e+07	3.900000e+06	0.248304	5.722222
Thriller	6.534306e+07	2.645800e+07	0.675204	5.641718
War	1.231160e+08	4.989667e+07	0.777887	6.187931
Western	6.218189e+07	3.725972e+07	0.690646	6.080952

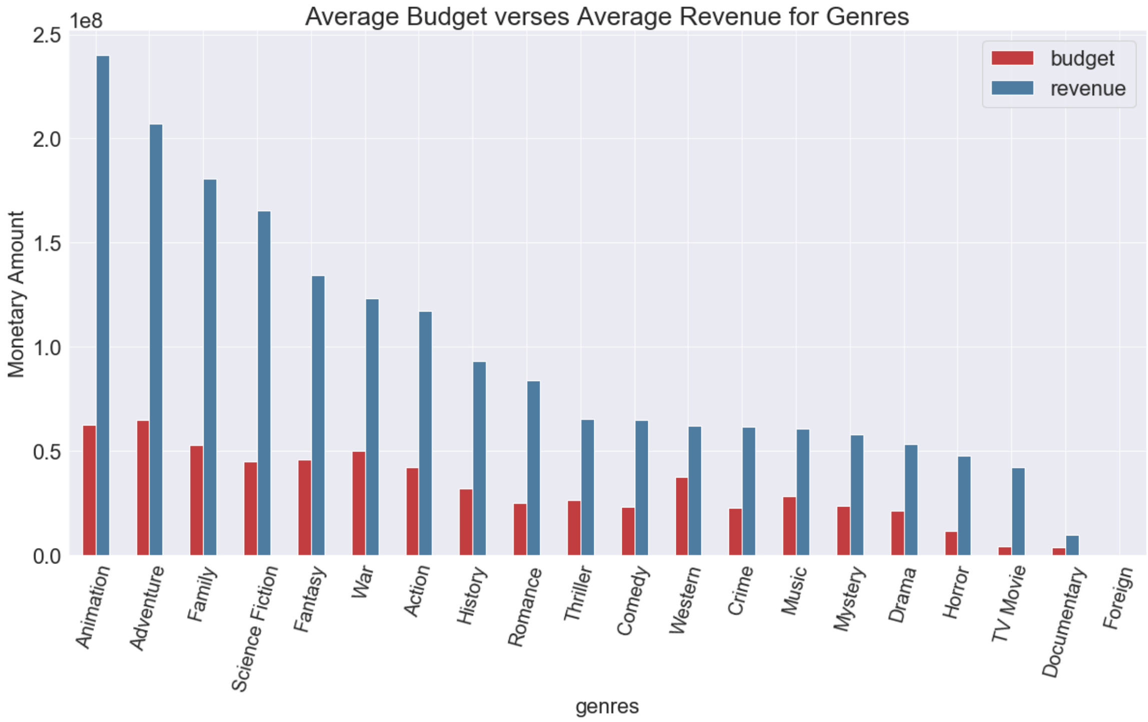
This gives us a table of the average budget, revenue and ratingfor each genre

Q2.1 Which genres have the largest revenue and largest budgets?

Now we have a table of the mean revenues, budgets and ratings for each genre, lets plot them in a bar chart to make them easier to analyse

In [46]:

```
# average budget and revenue of each genres' movies

f,ax=plt.subplots(figsize=(20, 10))
df_movie[['genres', 'budget', 'revenue']].groupby(['genres']).mean().sort_values(["reve
nue","budget"], ascending=False).plot(kind="bar",  ax=ax);
plt.xticks(rotation=75,fontsize=20)

ax.set(ylabel = 'Monetary Amount', title = 'Average Budget verses Average Revenue for G
enres')

plt.show()
```



Great! Looks like we have a sorted list of genres by their average budget and revenue! Looks like animation is a huge earner, and then there is Adventure - but it's budget is higher, surely that means that in Return on Investment terms, it's less successful? Looks like foreign movies, documentaries and TV Movies are at the bottom in terms of revenue.

Let's work it out...

Q2.2 Which genres are most profitable after working out Return on Investment?

In [47]:

```
# Create our new dataframe to work on RoI
df_movie_roi = df_movie[['genres', 'revenue', 'budget', 'popularity', 'vote_average']].
groupby(['genres']).mean()
```

In [48]:

```
df_movie_roi.head(2)
```

Out[48]:

	revenue	budget	popularity	vote_average
genres				
Action	1.170983e+08	4.231958e+07	0.838266	5.751009
Adventure	2.071020e+08	6.477152e+07	1.219834	6.049744

In [49]:

```
# Add a new attribute with the calculated RoI
df_movie_roi['RoI'] = df_movie_roi['revenue']/df_movie_roi['budget']
```

In [50]:

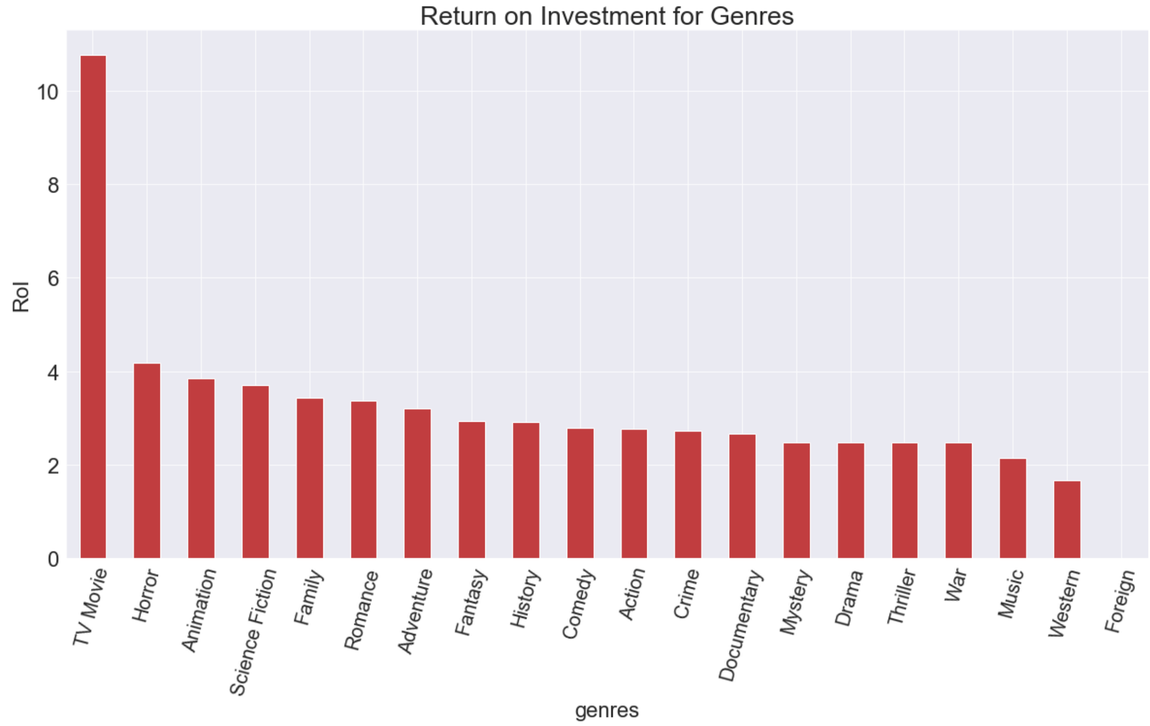
```
df_movie_roi.head(2)
```

Out[50]:

	revenue	budget	popularity	vote_average	RoI
genres					
Action	1.170983e+08	4.231958e+07	0.838266	5.751009	2.767000
Adventure	2.071020e+08	6.477152e+07	1.219834	6.049744	3.197424

In [51]:

```
# RoI of each genres' movies

f,ax=plt.subplots(figsize=(20, 10))
df_movie_roi['RoI'] .sort_values(ascending=False).plot(kind="bar",  ax=ax);
plt.xticks(rotation=75,fontsize=20)

ax.set(ylabel = 'RoI', title = 'Return on Investment for Genres')

plt.show()
```



Wow! Looks like TV Movies shot right up to the number 1 spot!! This makes sense - it has comparable revenue to other genres, but it's budget is tiny in comparison.

In [52]:

```
df_movie_roi.head()
```

Out[52]:

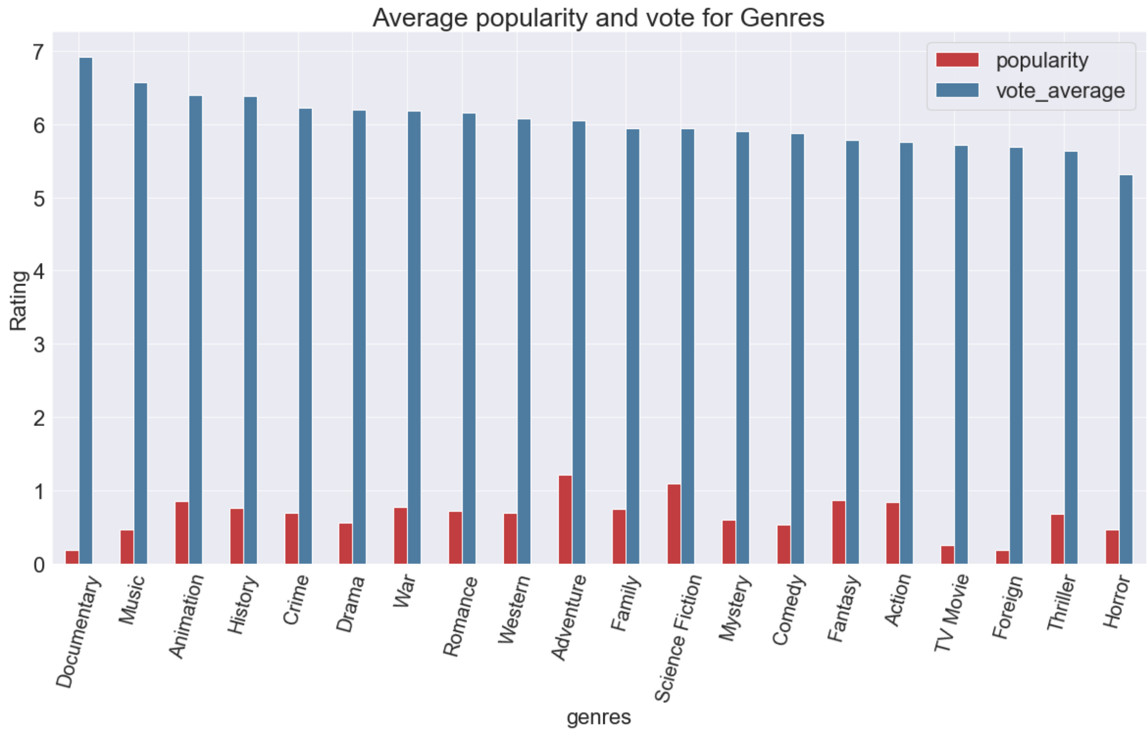
	revenue	budget	popularity	vote_average	RoI
genres					
Action	1.170983e+08	4.231958e+07	0.838266	5.751009	2.767000
Adventure	2.071020e+08	6.477152e+07	1.219834	6.049744	3.197424
Animation	2.399754e+08	6.229486e+07	0.853208	6.401867	3.852250
Comedy	6.471663e+07	2.320338e+07	0.539358	5.880971	2.789104
Crime	6.167864e+07	2.271513e+07	0.694063	6.217632	2.715311

Now let's see if we can plot popularity and vote average to see how they compare for each genre

Q2.3 Which genres are the most popular?

In [53]:

```
f,ax=plt.subplots(figsize=(20, 10))
df_movie[['genres', 'popularity', 'vote_average']].groupby(['genres']).mean().sort_valu
es(["vote_average"], ascending=False).plot(kind="bar",  ax=ax);
plt.xticks(rotation=75,fontsize=20)

ax.set(ylabel = 'Rating', title = 'Average popularity and vote for Genres')

plt.show()
```



We can see that Documentaries are consistantly highly rated, but have a very low popularity score indicating that while they are not frequently watched, they are enjoyed by those who do. At the opposite end of the scale we can see Horror has a reasonable popularity score indicating that it is popular with the viewing public, but it has the worst average rating indicating that quality is consistently poor.

Q3. Which Directors are the most successful (Revenue/Rating)?

We've carried out some interesting analysis on genres, but who is making these brilliant movies? Let's do some work to analyse the director attribute and find out who the top directors are

In [54]:

```
#Create new dataframe
df_director_movies = df_movie
#split out the director field
df_director_movies['director'] = df_director_movies['director'].apply(lambda x: x.split
("|")[0])
```

In [55]:

```
df_director_movies.shape
```

Out[55]:

```
(10703, 17)
```

In [56]:

```
df_director_movies.head(5)
```

Out[56]:

	id	popularity	budget	revenue	original_title	cast	director
0	135397	32.985763	150000000.0	1.513529e+09	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	Colin Trevorrow
1	76341	28.419936	150000000.0	3.784364e+08	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic...	George Miller
2	262500	13.112507	110000000.0	2.952382e+08	Insurgent	Shailene Woodley|Theo James|Kate Winslet|Ansel...	Robert Schwentke
3	140607	11.173104	200000000.0	2.068178e+09	Star Wars: The Force Awakens	Harrison Ford|Mark Hamill|Carrie Fisher|Adam D...	J.J. Abrams
4	168259	9.335014	190000000.0	1.506249e+09	Furious 7	Vin Diesel|Paul Walker|Jason Statham|Michelle ...	James Wan

Let's use revenue as our guage of success here, and aggregate the sum of revenue for our Directors

In [57]:

```
#Create our new dataframe for the Sumation of revenue for directors over time
df_director_revenue = df_director_movies.groupby(['director', 'release_year']).sum()['r
evenue']#.nlargest(10)
df_director_revenue = pd.DataFrame(df_director_revenue)
```

In [58]:

```
df_director_revenue.head(10)
```

Out[58]:

		revenue
director	**release_year**	
FrÃ©dÃ©ric Jardin	**2011**	3358.0
A.R. Murugadoss	**2008**	76000000.0
Aaron Aites	**2008**	0.0
Aaron Blaise	**2003**	250.0
Aaron Hann	**2015**	0.0
Aaron Harvey	**2011**	0.0
Aaron Katz	**2014**	0.0
Aaron Keeling	**2015**	0.0
Aaron Moorhead	**2015**	49970.0
Aaron Norris	**1988**	6193901.0

In [59]:

```
#Create new data frame for the total sumation of revenue for directors
df_director_revenue_total = df_director_revenue.groupby(['director']).sum()
df_director_revenue_total = pd.DataFrame(df_director_revenue_total)
df_director_revenue_total = df_director_revenue_total.sort_values(by = ['revenue'], asc
ending = False)
```

In [60]:

```
df_director_revenue_total.head(10)
```

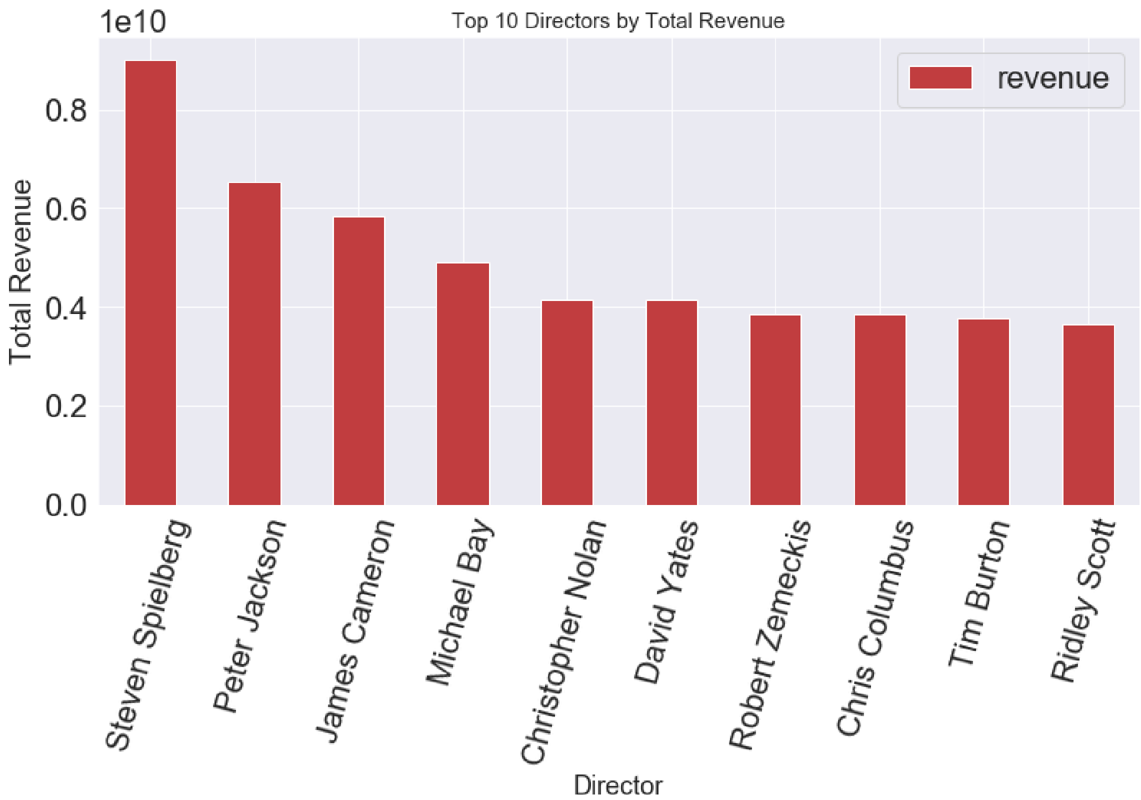
Out[60]:

	revenue
director	
Steven Spielberg	9.018564e+09
Peter Jackson	6.523245e+09
James Cameron	5.841895e+09
Michael Bay	4.917208e+09
Christopher Nolan	4.167549e+09
David Yates	4.154296e+09
Robert Zemeckis	3.869691e+09
Chris Columbus	3.851492e+09
Tim Burton	3.782610e+09
Ridley Scott	3.649996e+09

Cool - Looks like Stevn Spielberg is our most succesfull director, followed by Peter Jackson. Lets plot this to see how it looks

In [61]:

```
#plot a barh graph
df_director_revenue_total[:10].plot(kind = 'bar', figsize=(13,6))

#setup the title and the labels
plt.title("Top 10 Directors by Total Revenue",fontsize=15)
plt.xticks(rotation=75)
plt.xlabel("Director",fontsize= 18)
plt.ylabel("Total Revenue",fontsize= 20)
sns.set_style("whitegrid")
```



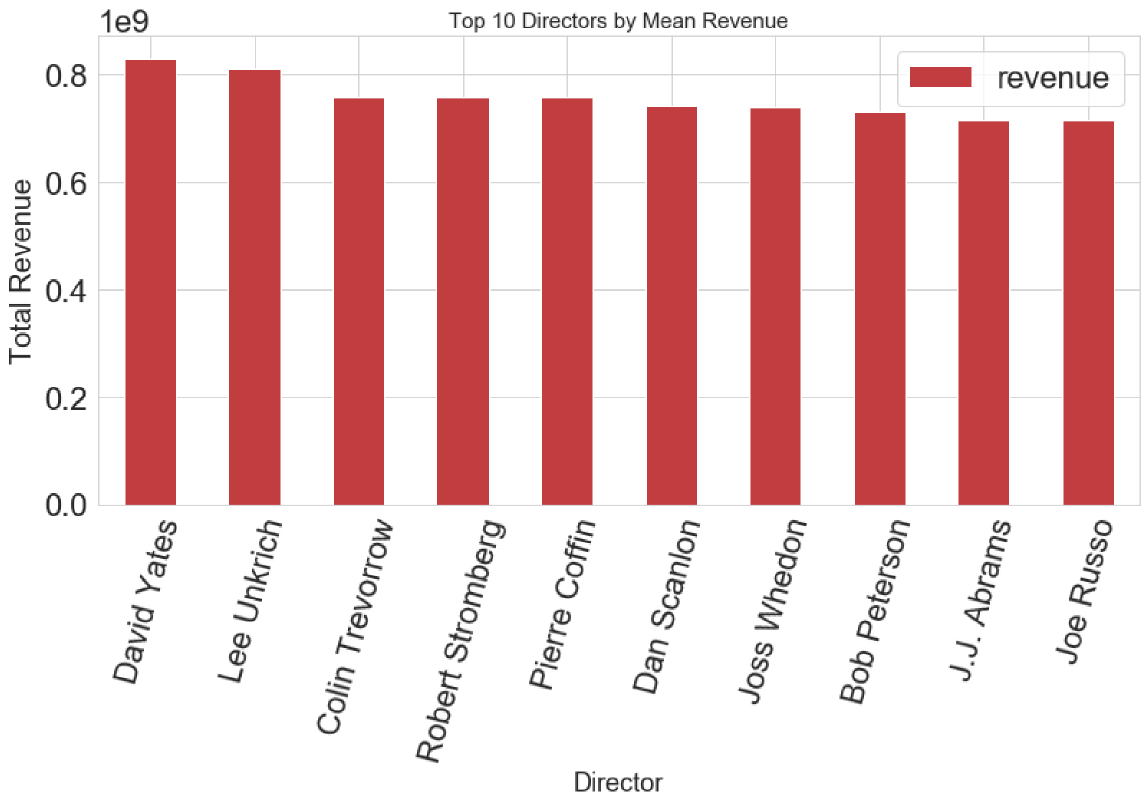
However, that skews our data toward those directors who have released more movies over their career - this would make sense. Instead, we could use the mean revenue of each film

In [62]:

```
#Create new data frame for the mean of revenue for directors
df_director_revenue_mean = df_director_revenue.groupby(['director']).mean()
df_director_revenue_mean = pd.DataFrame(df_director_revenue_mean)
df_director_revenue_mean = df_director_revenue_mean.sort_values(by = ['revenue'], ascen
ding = False)
```

In [63]:

```
#plot a barh graph
df_director_revenue_mean[:10].plot(kind = 'bar', figsize=(13,6))

#setup the title and the labels
plt.title("Top 10 Directors by Mean Revenue",fontsize=15)
plt.xticks(rotation=75)
plt.xlabel("Director",fontsize= 18)
plt.ylabel("Total Revenue",fontsize= 20)
sns.set_style("whitegrid")
```



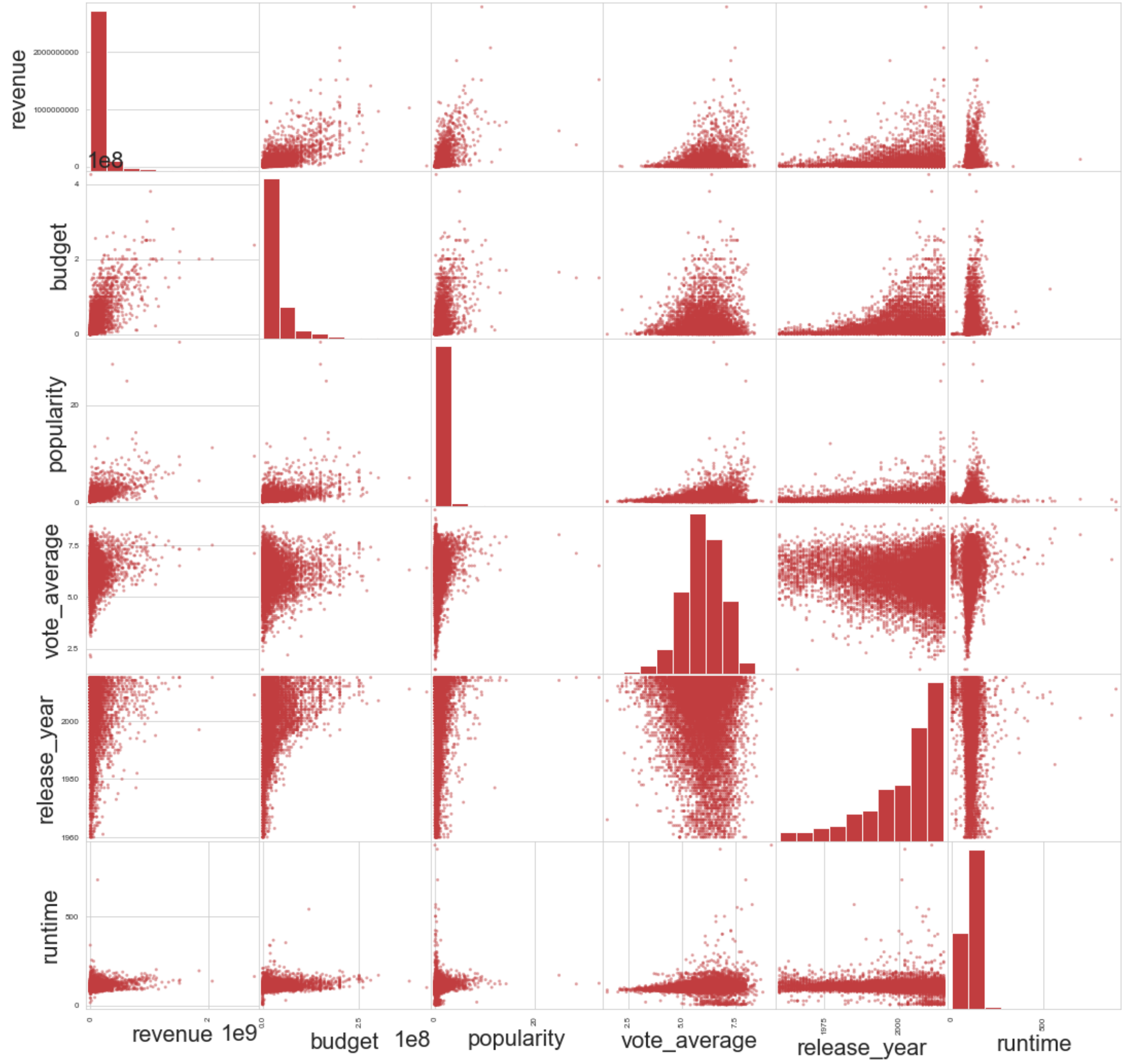
Looks like David Yates is the most succesfull in terms of average revenue, indicating they are the more consistent revenue generator

Q4. Which Attributes indicate a movie's chances of success (Revenue/Rating)?

To try and finid correlations, let's look at a scatter matrix with some of our features as a subset

In [64]:

```
aux_df = df_movie[['revenue', 'budget', 'popularity', 'vote_average', 'release_year',
'runtime']]

pd.plotting.scatter_matrix(aux_df, figsize=(18,18));
```



Looks like there is a positive correlation between budget and revenue, and a very slight positive correlation with release year and budget. With average rating slightly positive influenced by budget. These are only slight though, so the analysis here is limited. This does not indicate a causation in improvement in revenue/rating and a much deeper analysis would be required to find any correlation

4. Observations and Conclusion

Q1. How have the success of genres changed over time (Revenue/Rating)?

We found that Drama, comedy and action were the 3 top most frequent type of movies. The popularity of movie releases has generally grown over time, from 1960 til around 2010 where the numbers contract slightlty. From out area chart we could confirm that Drama, Thriller, Comedy and Action movies seem to be the predominant genres.

By plotting into heatmaps, looks like those trends are confirmed. This confirmed that revenues and budgets have generally grown over time.

Q2. How succesful are different genres (Revenue/Rating)?

When looking at the success of genres, we analysed a list of genres by their average budget and revenue where it looked like like animation is a huge earner, followed by Adventure while foreign movies, documentaries and TV Movies are at the bottom in terms of revenue. If we calculated a return on investment by dividing revenue by budget, TV Movies moved to the top of the RoI analysis results, meaning it earned the most per the amount of budget spent.

Q3. Which Directors are the most successful (Revenue/Rating)?

We found that by looking at the total revenue generated by directors, Stevn Spielberg is our most succesfull director, followed by Peter Jackson. However, with that result it favoured directors who had a long career, creating many movies. To mitigate that, we looked at a smoother feature using the average revenue per movie which resulted in David Yates being our more succesfull director.

Again, this could be skewed to those directors who may have submitted very few but highly lucrative movies. In future, we could incorporate avarage rating, Return on Investment to see how profitable they were, limiting the directors to those who submitted a minimum number of movies

Q4. Which Attributes indicate a movie's chances of success (Revenue/Rating)?

We skimmed the surface of what attributes helped define a movies chance of success. We tentatively would say there Looks like there is a positive correlation between budget and revenue, and a very slight positive correlation with release year and budget. This would suggest that with a better budget the movie has a better chance of success in revenue. Increase in budget and revenue correlated to release year suggests that as time passes, revenues and budgets grow. We could account for inflation to confirm this.

Limitations and Assumptions

- Assumimng 0 revenue and 0 budget are actually missing values and not actually 0 revenue and 0 budget.
- Only used the original budget and revenue figures, ignoring figures adjusted for inflation.
- Vote average can be skewed by the number of votes - vote nubmers were never taken into account.
- Correlations to the success of a movie are limited to only numerical values, not nearly indepth enough to use as an indication of a movies success. When looking at revenue and budget over release year, we should account for inflation here to smooth out the figures, as more recent films will have larger revenunes and budgets due to the impact of inflation.

- Director analysis based on a sum of revenue, which would skew toward directors who had created more movies. We coud have generated a "per movie" figure, used the mean to even out the results. We could also have looked at our definition of success and incorporated average rating for each director
- Again, in Director analysis our mean revenue per director could be skewed to those directors who may have submitted very few but highly lucrative movies. In future, we could incorporate avarage rating, Return on Investment to see how profitable they were, limiting the directors to those who submitted a minimum number of movies

---

References

- [TMDB movie data (cleaned from Kaggle, by Udacity) (https://www.google.com/url?q=https://d17h27t6h515a5.cloudfront.net/topher/2017/October/59dd1c4c_tmdb-movies/tmdb-movies.csv&sa=D&ust=1532469042115000)](https://www.google.com/url?q=https://d17h27t6h515a5.cloudfront.net/topher/2017/October/59dd1c4c_tmdb-movies/tmdb-movies.csv&sa=D&ust=1532469042115000)
- [UDACITY Data Analyst Nanodegree (https://eu.udacity.com/course/data-analyst-nanodegree--nd002?v=a)](https://eu.udacity.com/course/data-analyst-nanodegree--nd002?v=a)
- [Investigate TMDb Movie Dataset (Python Data Analysis Project) by Lorna Yen (https://medium.com/@onpillow/01-investigate-tmdb-movie-dataset-python-data-analysis-project-part-1-data-wrangling-3d2b55ea7714)](https://medium.com/@onpillow/01-investigate-tmdb-movie-dataset-python-data-analysis-project-part-1-data-wrangling-3d2b55ea7714)
- [Investigate a dataset (https://praxitelisk.github.io/DAND-P1-Investigate-a-Dataset/Investigate_a_Dataset.html)](https://praxitelisk.github.io/DAND-P1-Investigate-a-Dataset/Investigate_a_Dataset.html)
- [Title Image (http://pluspng.com/)](http://pluspng.com/)

In [ ]:

```

```