Solution - Box Office Predictions

# Solution: Box Office Predictions

## Import Libraries and Dataset

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

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
```

In [2]:
```
df = pd.read_csv('data/box_office_predictions.csv')
```

## Exploratory Analysis and Data Cleaning

We'll start by "getting to know" the dataset. The goal is to understand the dataset at a qualitative level, note anything that should be cleaned up, and spot opportunities for feature engineering.

**Tip:** In our Machine Learning Masterclass, we separate exploratory analysis and data cleaning into separate steps. However, because take-home challenges are meant to be shorter/condensed analyses, feel free to combine these steps. Just remember to keep your code clean and organized.

In [3]:
```
# Example observations
df.head()
```

Out[3]:

| | budget | country | director | genre | gross | name | rating | runtime | score | star |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 237000000.0 | UK | James Cameron | Action | 760507625.0 | Avatar (2009) | PG-13 | 162 | 7.8 | Sam Worthington |
| **1** | 200000000.0 | USA | James Cameron | Drama | 658672302.0 | Titanic (1997) | PG-13 | 194 | 7.8 | Leonardo DiCaprio |
| **2** | 150000000.0 | USA | Colin Trevorrow | Action | 652270625.0 | Jurassic World (2015) | PG-13 | 124 | 7.0 | Chris Pratt |
| **3** | 220000000.0 | USA | Joss Whedon | Action | 623357910.0 | The Avengers (2012) | PG-13 | 143 | 8.1 | Robert Downey Jr. |

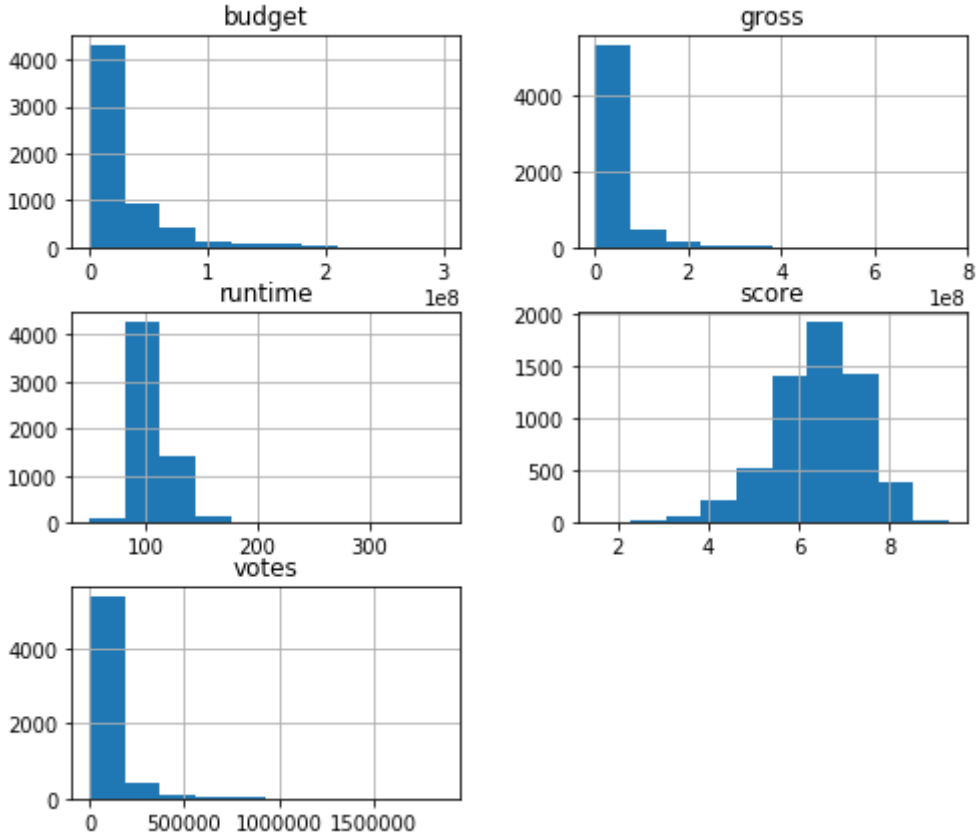
| | budget | country | director | genre | gross | name | rating | runtime | score | star |
|---|---|---|---|---|---|---|---|---|---|---|
| **4** | 185000000.0 | USA | Christopher Nolan | Action | 534858444.0 | The Dark Knight (2008) | PG-13 | 152 | 9.0 | Christian Bale |

First, we look at some examples observations to get a "feel" for the dataset. We learn that:

- The dataset has a mix of numeric and categorical features.
- We have `budget` and `gross` revenue, but no variable for `profit` or `roi`. We'll need to create these later.
- The "name" feature also includes the year the film was released. We can extract this information to create an `age` of film feature.

Next, we'll dive deeper into the distributions and statistics of our numeric features.

In [4]:
```
# Numeric feature distributions
df.hist(figsize=(8,7))
plt.show()
```




In [5]:
```
# Numeric feature summary statistics
df.describe()
```

Out[5]:

| | budget | gross | runtime | score | votes |
|---|---|---|---|---|---|

| | budget | gross | runtime | score | votes |
|---|---|---|---|---|---|
| **count** | 6.000000e+03 | 6.000000e+03 | 6000.000000 | 6000.000000 | 6.000000e+03 |
| **mean** | 2.469918e+07 | 3.341635e+07 | 106.587000 | 6.386383 | 7.188537e+04 |
| **std** | 3.721710e+07 | 5.735205e+07 | 18.026885 | 0.994921 | 1.308033e+05 |
| **min** | 0.000000e+00 | 4.410000e+02 | 50.000000 | 1.500000 | 2.700000e+01 |
| **25%** | 0.000000e+00 | 1.527796e+06 | 95.000000 | 5.800000 | 7.791750e+03 |
| **50%** | 1.100000e+07 | 1.229897e+07 | 102.000000 | 6.500000 | 2.660150e+04 |
| **75%** | 3.262500e+07 | 4.007256e+07 | 115.000000 | 7.100000 | 7.677475e+04 |
| **max** | 3.000000e+08 | 7.605076e+08 | 366.000000 | 9.300000 | 1.868308e+06 |

Based on the summary statistics, we see that some films have a budget of `0` in the dataset. Here are a few examples:

In [6]:
```
# Examples of films with 0 budget
df[df.budget == 0].head()
```

Out[6]:

| | budget | country | director | genre | gross | name | rating | runtime | score | star | studio |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **56** | 0.0 | UK | David Yates | Adventure | 295983305.0 | Harry Potter and the Deathly Hallows: Part 1 (... | PG-13 | 146 | 7.7 | Daniel Radcliffe | Warner Bros. |
| **207** | 0.0 | USA | Walt Becker | Action | 168273550.0 | Wild Hogs (2007) | PG-13 | 100 | 5.9 | Tim Allen | Touchs Picture |
| **431** | 0.0 | USA | John G. Avildsen | Action | 115103979.0 | The Karate Kid Part II (1986) | PG | 113 | 5.9 | Pat Morita | Columl Picture Corpor |
| **553** | 0.0 | USA | Nora Ephron | Comedy | 95318203.0 | Michael (1996) | PG | 105 | 5.7 | John Travolta | Turner Picture |
| **592** | 0.0 | USA | Tyler Perry | Comedy | 90485233.0 | Madea Goes to Jail (2009) | PG-13 | 103 | 4.3 | Tyler Perry | Tyler P Compa The |

A Harry Potter film with `0` budget? No way. These are most likely missing values / data collection errors. But since our exercise is to investigate ***profitability*** of films, we cannot study films with missing budget values.

To improve the analysis in the future, we'd want to troubleshoot our data source to find out if we

In [7]:
```
# Remove films with "0" budget
df = df.loc[df.budget > 0,:]
```

Next, we'll look at the distributions of our categorical features.

In [8]:
```
# Categorical feature summary statistics
df.describe(include=['object'])
```

Out[8]:

| | country | director | genre | name | rating | star | studio |
|---|---|---|---|---|---|---|---|
| **count** | 4089 | 4089 | 4089 | 4089 | 4089 | 4089 | 4089 |
| **unique** | 42 | 1757 | 16 | 4089 | 8 | 1501 | 1232 |
| **top** | USA | Woody Allen | Comedy | Miss Peregrine's Home for Peculiar Children (2... | R | Nicolas Cage | Universal Pictures |
| **freq** | 3275 | 26 | 1136 | 1 | 2001 | 36 | 235 |

The immediate insight that jumps out is that some categorical features have a large number of unique classes relative to the number of total observations. For example, there are 1232 unique studios in the dataset of 4089 observations (after filtering for non-zero budget). This will most likely lead to **sparse classes**, so we'll want to address this during feature engineering.

For example, many of the studios only have 1 film in the dataset:

In [9]:
```
# Reverse sort studios by total number of films
df.studio.value_counts().tail()
```

Out[9]:
```
Film Workshop                1
Human Stew Factory           1
Aramid Entertainment Fund    1
Code Entertainment           1
Enchantment Films Inc.       1
Name: studio, dtype: int64
```

Compare that to the top studios in the dataset, and we can see that it will be useful to combine low-frequency studios into tiered classes.

In [10]:
```
# Top studios by total number of films
df.studio.value_counts().head()
```

Out[10]:
```
Universal Pictures          235
Warner Bros.                231
Paramount Pictures          197
```

```
Twentieth Century Fox Film Corporation    148
New Line Cinema                           123
Name: studio, dtype: int64
```

The second insight that jumps out is that there are as many unique `name` values as the number of total observations. For all intents and purposes, the `name` feature acts as an index column. However, as mentioned above, we can still extract useful information based on the year the film was released.

## Feature Engineering

First, we'll create our target variable:

In [11]:
```
df['profit'] = df.gross - df.budget
```

Next, let's combine the sparse classes in the dataset.

We'll start with `studio`. Note: There are other valid ways of combining classes. For example, you could try combining studios based on their average production budget (as a proxy for studio size). We will combine them based on their total number of films in the dataset.

In [12]:
```
# Number of films from each studio
studio_counts = df.studio.value_counts()

# Tiers for sparser studios
one_timers = studio_counts[studio_counts <= 1].index
three_timers = studio_counts[(studio_counts > 1) & (studio_counts <= 3)].
five_timers = studio_counts[(studio_counts > 3) & (studio_counts <= 5)].i
ten_timers = studio_counts[(studio_counts > 5) & (studio_counts <= 10)].i

# Combine sparse studios
df['studio'].replace(one_timers, 'One Timer', inplace=True)
df['studio'].replace(three_timers, 'Three Timer', inplace=True)
df['studio'].replace(five_timers, 'Five Timer', inplace=True)
df['studio'].replace(ten_timers, 'Ten Timer', inplace=True)
```

Same with `star` ...

In [13]:
```
# Number of films from each star
star_counts = df.star.value_counts()

# Tiers for sparser stars
one_timers = star_counts[star_counts <= 1].index
```

```
three_timers = star_counts[(star_counts > 1) & (star_counts <= 3)].index
five_timers = star_counts[(star_counts > 3) & (star_counts <= 5)].index
ten_timers = star_counts[(star_counts > 5) & (star_counts <= 10)].index

# Combine sparse stars
df['star'].replace(one_timers, 'One Timer', inplace=True)
df['star'].replace(three_timers, 'Three Timer', inplace=True)
df['star'].replace(five_timers, 'Five Timer', inplace=True)
df['star'].replace(ten_timers, 'Ten Timer', inplace=True)
```

And `director` ...

In [14]:
```
# Number of films from each director
director_counts = df.director.value_counts()

# Tiers for sparser directors
one_timers = director_counts[director_counts <= 1].index
three_timers = director_counts[(director_counts > 1) & (director_counts <
five_timers = director_counts[(director_counts > 3) & (director_counts <=
ten_timers = director_counts[(director_counts > 5) & (director_counts <=

# Combine sparse directors
df['director'].replace(one_timers, 'One Timer', inplace=True)
df['director'].replace(three_timers, 'Three Timer', inplace=True)
df['director'].replace(five_timers, 'Five Timer', inplace=True)
df['director'].replace(ten_timers, 'Ten Timer', inplace=True)
```

For `country`, we'll group all countries with fewer than 50 films into a single `'Other'` class.

In [15]:
```
# Number of films from each country
country_counts = df.country.value_counts()

# Combine countries with fewer than 50 films
other_countries = country_counts[country_counts < 50].index
df['country'].replace(other_countries, 'Other', inplace=True)

# New class frequencies
df.country.value_counts()
```

Out[15]:
```
USA        3275
UK          323
Other       243
France       98
Germany      81
Canada       69
Name: country, dtype: int64
```

Same with `genre`.

In [16]:
```
# Number of films in each genre
genre_counts = df.genre.value_counts()

# Combine genres with fewer than 50 films
other_genres = genre_counts[genre_counts < 50].index
df['genre'].replace(other_genres, 'Other', inplace=True)

# New genre frequencies
df.genre.value_counts()
```

Out[16]:
```
Comedy       1136
Action        972
Drama         719
Crime         315
Adventure     253
Animation     205
Biography     203
Horror        201
Other          85
Name: genre, dtype: int64
```

The `rating` feature has a different issue, where **"unrated"** films have a few different class labels that mean the same thing.

In [17]:
```
df.rating.value_counts()
```

Out[17]:
```
R                2001
PG-13            1372
PG                562
G                  90
NOT RATED          36
UNRATED            15
NC-17               9
Not specified       4
Name: rating, dtype: int64
```

Let's fix that.

In [18]:
```
# Fix "unrated" labels
df['rating'].replace(['NOT RATED', 'UNRATED', 'Not specified'], 'NR', inp
```

Finally, we'll create an `age` feature for the age of the film. We'll use this as a proxy for factors like

inflation and general industry growth.

**Note:** We will set "today" to 2014 to imitate an analysis performed in 2014 to predict films in 2015 and 2016.

In [19]:
```
def extract_age(s, today=2014):
    return today - int( s[-5:-1] )

# Example
extract_age('Titanic (1997)')
```

Out[19]: 17

In [20]:
```
# Create "age" feature
df['age'] = df.name.apply(extract_age)
```

Here's what the dataframe looks like now.

In [21]:
```
df.head()
```

Out[21]:

| | budget | country | director | genre | gross | name | rating | runtime | score | star | studi |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 237000000.0 | UK | Five Timer | Action | 760507625.0 | Avatar (2009) | PG-13 | 162 | 7.8 | Three Timer | Twent Centu Fox F Corpo |
| **1** | 200000000.0 | USA | Five Timer | Drama | 658672302.0 | Titanic (1997) | PG-13 | 194 | 7.8 | Leonardo DiCaprio | Twent Centu Fox F Corpo |
| **2** | 150000000.0 | USA | Three Timer | Action | 652270625.0 | Jurassic World (2015) | PG-13 | 124 | 7.0 | Three Timer | Unive Pictur |
| **3** | 220000000.0 | USA | Three Timer | Action | 623357910.0 | The Avengers (2012) | PG-13 | 143 | 8.1 | Robert Downey Jr. | Ten Ti |
| **4** | 185000000.0 | USA | Ten Timer | Action | 534858444.0 | The Dark Knight (2008) | PG-13 | 152 | 9.0 | Christian Bale | Warn Bros. |

## Correlations

To answer the first challenge question, we'll calculate correlations using our cleaned dataset:

In [22]:
```
# Correlations
df[['score', 'votes', 'profit']].corr()
```

Out[22]:

| | score | votes | profit |
|---|---|---|---|
| **score** | 1.000000 | 0.469748 | 0.249742 |
| **votes** | 0.469748 | 1.000000 | 0.500129 |
| **profit** | 0.249742 | 0.500129 | 1.000000 |

We see that `votes` has a higher correlation with `profit` than `score` does. This makes sense intuitively because `votes` should be a proxy for the overall popularity of the film.

However, one thing we should check is whether `votes` is also correlated with some other factor, such as `budget`.

In [23]:
```
# Correlations with budget included
df[['score', 'votes', 'profit', 'budget']].corr()
```

Out[23]:

| | score | votes | profit | budget |
|---|---|---|---|---|
| **score** | 1.000000 | 0.469748 | 0.249742 | 0.067490 |
| **votes** | 0.469748 | 1.000000 | 0.500129 | 0.452685 |
| **profit** | 0.249742 | 0.500129 | 1.000000 | 0.094817 |
| **budget** | 0.067490 | 0.452685 | 0.094817 | 1.000000 |

Aha. We see that `budget` is correlated with `votes`, which simply indicates that studios with bigger budgets are more popular or can spend more on marketing a film. However, what's very interesting is that `budget` is not necessarily correlated with `profit` while `votes` is.

**Note:** This might seem contradictory at first, but try drawing a venn diagram to shed some light on how this situation can occur. (See the Appendix of this solution for an example.)

## Machine Learning

Next, we'll create our analytical base table for machine learning. We will drop `name` because it's basically an index column and the `gross`, `votes`, and `score` features because we do not know them at the time.

In [24]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
```

In [25]:
```
# Create analytical base table (ABT)
abt = pd.get_dummies ( df.drop(['name', 'gross', 'votes', 'score'], axis=
```

We'll split the data based on date of training data (2014 and earlier) and test data (2015, 2016)

In [26]:
```
# Train / Test split based on date of training data (2014 and earlier) a
train = abt[abt.age >= 0]
test = abt[abt.age <= 0]

y_train = train.profit
X_train = train.drop(['profit'], axis=1)

y_test = test.profit
X_test = test.drop(['profit'], axis=1)
```

We'll fit and train a basic random forest.

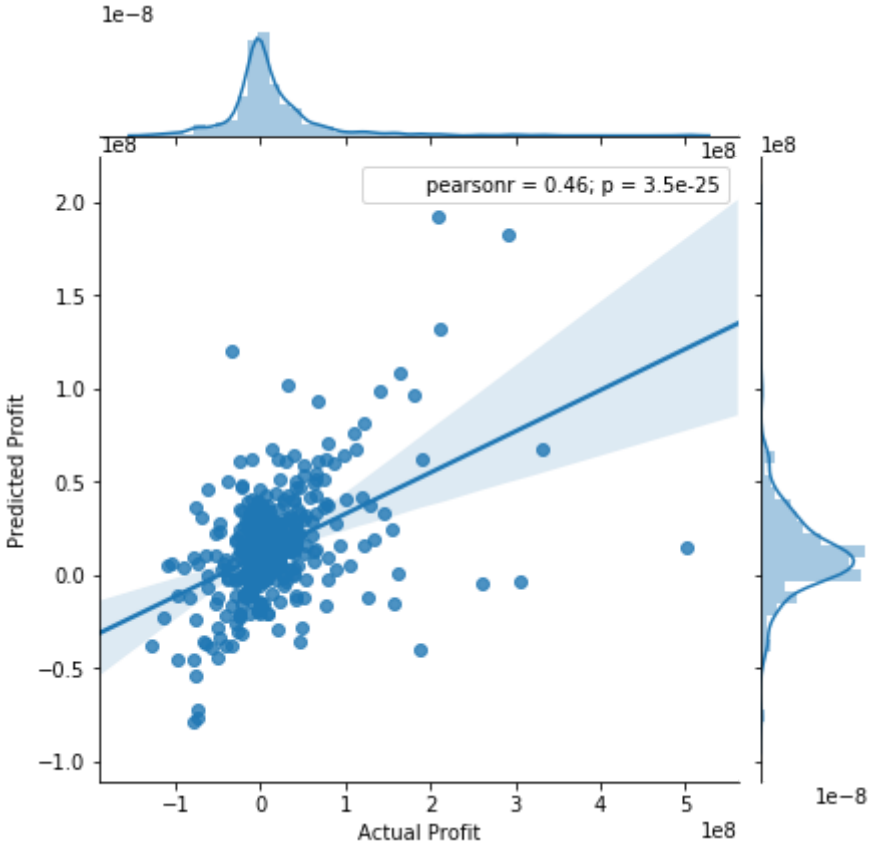
In [27]:
```
# Train a basic random forest model
rf = RandomForestRegressor(random_state=1234)
rf.fit(X_train, y_train)

# Make prediction on test set
pred = rf.predict(X_test)
```

**Note:** In take-home challenges, you typically do not need spend a large amount of time optimizing your model. Just pick a reasonable model (we prefer random forests), explain why (because they work well out-of-the-box, can model non-linear relationships, and are quite robust to outliers), and make a note of what you would try if given more time (tune hyperparameters, try other algorithms, etc).

Next, we'll plot the model performance.

In [28]:
```
sns.jointplot(y_test, pred, kind='reg')
plt.xlabel('Actual Profit')
plt.ylabel('Predicted Profit')
plt.show()
```




In [29]:
```
# R^2
r2_score(y_test, pred)
```

Out[29]: 0.21246010082831357

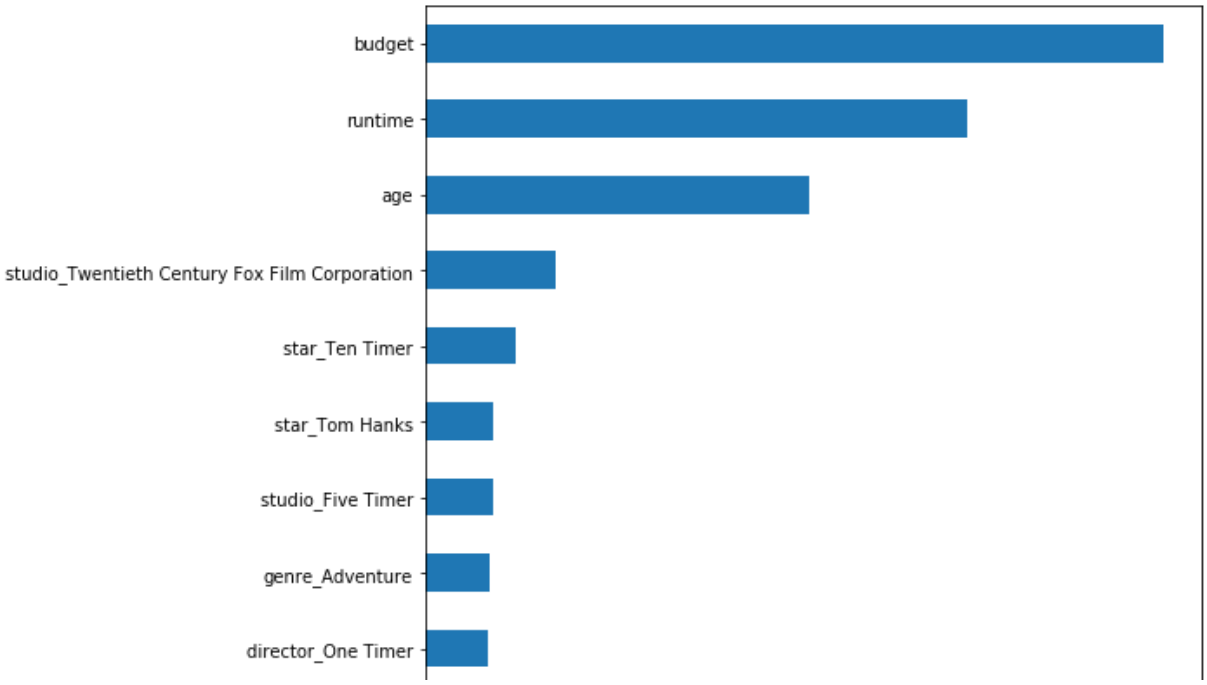
As you can see in the plot, our basic model actually does a decent job predicting the profitability of films based on the limited data we have (and the limited time in optimizing the model).

Whether this model performance is "good enough" will depend on the use-case. For example, in a betting market, this model would already give a formidable edge.

Finally, we'll plot the feature importances.

In [30]:
```
# Helper function for plotting feature importances
def plot_feature_importances(columns, feature_importances, show_top_n=10)
    feats = dict( zip(columns, feature_importances) )
    imp = pd.DataFrame.from_dict(feats, orient='index').rename(columns={(
    imp.sort_values(by='Gini-importance').tail(show_top_n).plot(kind='bar
    plt.show()
```

In [31]:
```
plot_feature_importances(X_train.columns, rf.feature_importances_)
```




As a whole, the `budget` feature was most important in the model. But wait, didn't we find earlier that `budget` and `profit` were not correlated?

This seems contradictory, but the answer is simple, and it has to do with the difference between first-order correlations and a full model. Earlier, we were looking at the correlation between `budget` and `profit` at an aggregate level.

But now that we've built a model, we can look at the affect of `budget` *while* controlling for all the other input features as well.

**Note:** This is a key distinction, and something worth noting in your write-ups.

## Machine Learning w/ Pre-Screen

Next, we'll create a new analytical base table for the scenario where we're able to collect an accurate `score` input based on film pre-screenings.

In [32]:
```
# Create new analytical base table (ABT)
abt_ps = pd.get_dummies ( df.drop(['name', 'gross', 'votes'], axis=1) )
```

Same train/test split.

In [33]:
```
train = abt_ps[abt_ps.age >= 0]
test = abt_ps[abt_ps.age <= 0]

y_train = train.profit
```

```
X_train = train.drop(['profit'], axis=1)

y_test = test.profit
X_test = test.drop(['profit'], axis=1)
```

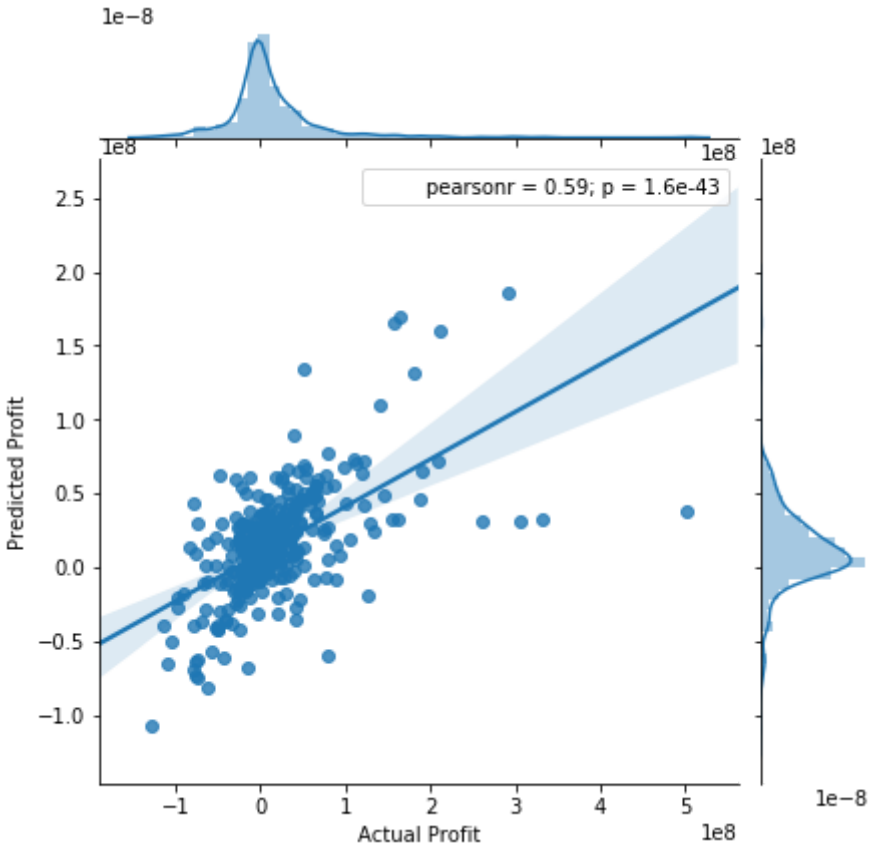
And same model training process.

In [34]:
```
# Train a basic random forest model
rf = RandomForestRegressor(random_state=1234)
rf.fit(X_train, y_train)

# Make prediction on test set
pred = rf.predict(X_test)
```

Here's the new model's performance:

In [35]:
```
sns.jointplot(y_test, pred, kind='reg')
plt.xlabel('Actual Profit')
plt.ylabel('Predicted Profit')
plt.show()
```

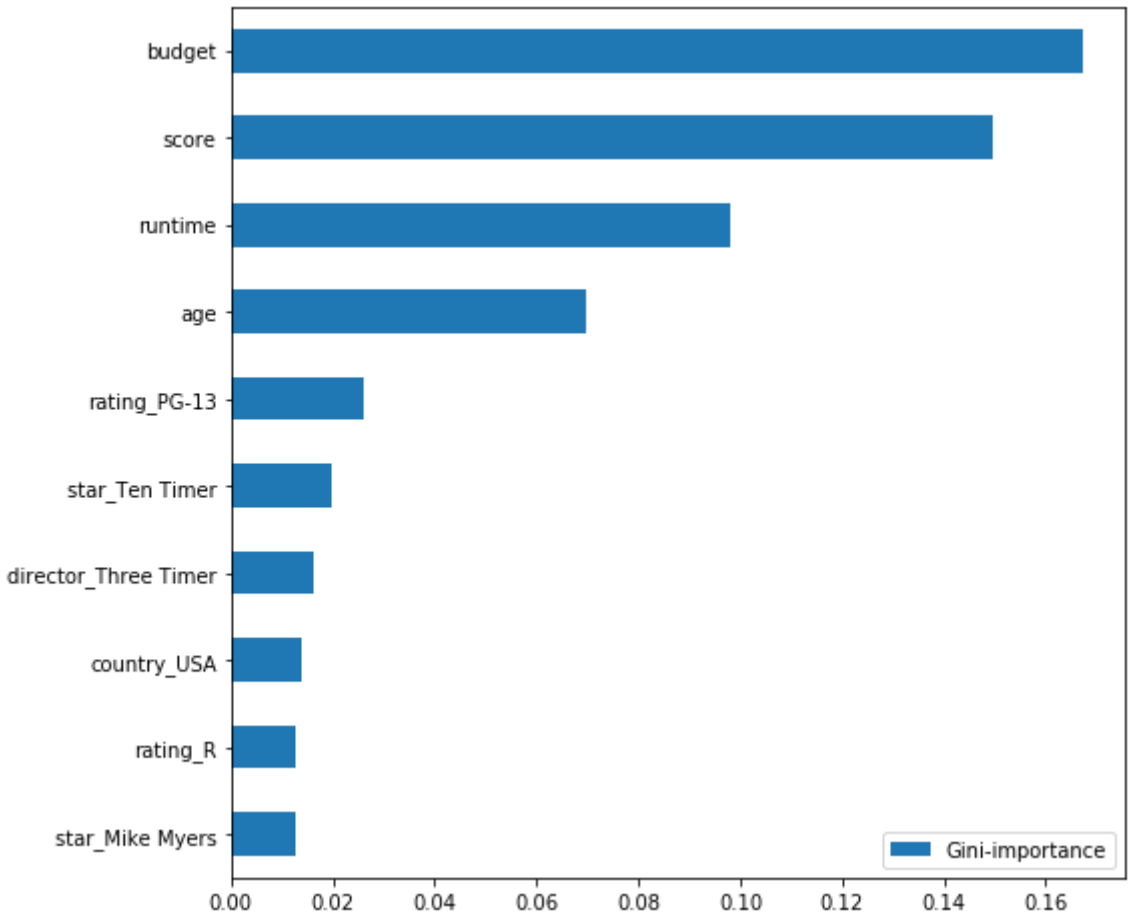


In [36]:
```
r2_score(y_test, pred)
```

Out[36]:
```
0.34504043080338764
```

Including the **`score`** feature improves our model's performance substantially. We should make effort to collect this data for any film we'd like to predict.

In [37]:
```
plot_feature_importances(X_train.columns, rf.feature_importances_)
```



The `score` feature is the second most importance feature, right behind `budget`.