DAND Term 1 Project 3

Jesse Fredrickson

8/15/18

TMDb movie data

[Link (https://www.google.com/url?q=https://d17h27t6h515a5.cloudfront.net/topher/2017/October/59dd1c4c_tmdb-movies/tmdb-movies.csv&sa=D&ust=1534386214169000)]

In this project, I will conduct my own data analysis and document my findings in this jupyter notebook. Please note that all findings are tentative and based simply on observations, not on inferential statistics or machine learning. Correlation does not imply causation

The dataset I have chosen to work with is a list of movies along with information on their release and performance, as well as their cast and director. I am interested in learning about how movies have changed over the timeline of this dataset, which has results dating back almost 60 years. Are production companies spending more on movies now than they did in previous decades? Has this trend been consistant? Are companies releasing more movies now than previous decades? Do audiences respond better to movies with a higher production budget? Have audiences always responded this way?

Brainstorming and cleaning

I will first import the data, print a brief header to see what the format looks like, then clean the data to remove NaNs and duplicates if necessary

In [12]:
```
# imports, loads, and magics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from mpl_toolkits import mplot3d

%matplotlib inline

df = pd.read_csv('tmdb-movies.csv')
df.head(2)
```

Out[12]:

	id	imdb_id	popularity	budget	revenue	original_title	cast	homepage	directo
0	135397	tt0369610	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	http://www.jurassicworld.com/	Colin Trevorro
1	76341	tt1392190	28.419936	150000000	378436354	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic...	http://www.madmaxmovie.com/	George Miller

2 rows × 21 columns

In [13]:
```
df.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
```

I can see from this already that there are some cells that will require wrangling to get useful info from. Namely, `genres` and `cast` are both strings which would be more useful to me as a list, and would require some restructuring of the dataframe in order to make it so each title had a separate row for each cast member or genre. Fortunately, I am investigating budget, popularity, and release year, which all appear to be the correct data format. I will proceed to remove the rows that I know I will not need, but I am going to conserve most columns - just in case.

In [15]:
```
# Remove columns which are irrelevant
dropcols = ['homepage', 'tagline', 'overview']
df.drop(dropcols, axis=1, inplace=True)
```

In [16]:
```
# check for duplicates in all columns
df.drop_duplicates(inplace = True)
for col in df.columns:
    print('Duplicates in ' + col + ': ' + str(df.duplicated(col).sum()))
```

```
Duplicates in id: 0
Duplicates in imdb_id: 9
Duplicates in popularity: 51
Duplicates in budget: 10308
Duplicates in revenue: 6163
Duplicates in original_title: 294
Duplicates in cast: 145
Duplicates in director: 5797
Duplicates in keywords: 2060
Duplicates in runtime: 10618
Duplicates in genres: 8825
Duplicates in production_companies: 3419
Duplicates in release_date: 4956
Duplicates in vote_count: 9576
Duplicates in vote_average: 10793
Duplicates in release_year: 10809
Duplicates in budget_adj: 8251
Duplicates in revenue_adj: 6025
```

There are lots of duplicates in this set. I can see that there are no duplicates in the `id` field which is good, but both the `imdb_id` and `original_title` columns have duplicates, which warrants investigation. I do not want to be double counting any films, and I am not sure what the two ID fields are based on if not just the movie title. I will next look at `imdb_id` and `original_title`duplicates to see what I am dealing with here.

In [17]:
```
# examine duplicates
df[df['imdb_id'].duplicated()].head(5)
```

Out[17]:

	id	imdb_id	popularity	budget	revenue	original_title	cast	director	ke
997	287663	NaN	0.330431	0	0	Star Wars Rebels: Spark of Rebellion	Freddie Prinze Jr.|Vanessa Marshall|Steve Blum...	Steward Lee|Steven G. Lee	NaN
1528	15257	NaN	0.607851	0	0	Hulk vs. Wolverine	Fred Tatasciore|Bryce Johnson|Steve Blum|Nolan...	Frank Paur	marvel comic|superhero|wolverine|hu my...
1750	101907	NaN	0.256975	0	0	Hulk vs. Thor	Graham McTavish|Fred Tatasciore|Matthew Wolf|J...	Sam Liu	marvel comic|superhero|hulk| mythology|su...
2401	45644	NaN	0.067753	0	0	Opeth: In Live Concert At The Royal Albert Hall	Mikael Ã… kerfeldt|Martin "Axe" Axenrot|Martin ...	NaN	NaN
4797	369145	NaN	0.167501	0	0	Doctor Who: The Snowmen	Matt Smith|Jenna Coleman|Richard E. Grant|Ian ...	NaN	NaN

In [18]:
```
df[df['original_title'].duplicated()].head(5)
```

Out[18]:

	id	imdb_id	popularity	budget	revenue	original_title	cast	director	k
1133	281778	tt3297792	0.188264	0	0	Survivor	Danielle Chuchran|Kevin Sorbo|Rocky Myers|Ruby...	John Lyde	survivor
1194	296626	tt3534842	0.132764	0	0	Finders Keepers	Jaime Pressly|Kylie Rogers|Tobin Bell|Patrick ...	Alexander Yellen	profession|evil doll|possession|murder|p
1349	42222	tt0076245	0.398651	0	0	Julia	Jane Fonda|Vanessa Redgrave|Jason Robards|Maxi...	Fred Zinnemann	friends|playwright
1440	7445	tt0765010	1.223786	26000000	43318349	Brothers	Tobey Maguire|Jake Gyllenhaal|Natalie Portman|...	Jim Sheridan	brother brother relationship|brother-in-la
1513	62320	tt1014762	0.688361	0	0	Home	Glenn Close|Yann Arthus-Bertrand|Jacques Gambl...	Yann Arthus-Bertrand	climate change|earth|glo warming|water poll...

The duplicate `imdb_id` rows appear to be Nulls, and since I am not going to be aggregating over that column I don't need to worry about those. The `original title` rows are NOT Nulls, meaning some will need to be dropped. I could just drop all the duplicate titles, but I noticed that in both cases, the rows displayed above have a number of missing fields in some columns. If any of the duplicate rows are in fact complete, I want to preserve the information they contained. In fact, even if none of the duplicate title rows are complete, I want to preserve as much info as they can provide as a whole. One way to do this is to replace all missing values (zeros) with Null (np.nan), group the dataframe by title, and then ffill and bfill before finally dropping duplicate title rows. This way, each row left will be unique and contain all of the information that the removed rows would have left behind.

In [5]:
```
# want to drop all original title duplicates while preserving all relevant data.
# Saving only the first occurrance might throw away valuable data
# on way to do this would be to groupby original_title, replace all 0s with np.nan, ffill and bfill
# then drop duplicates.
def fillmeupdaddy(df, col):
    df.groupby([col]).ffill().bfill(inplace=True)
    df.drop_duplicates(['original_title'], inplace=True)
    return df

df.replace(0,np.nan)
clean_df = fillmeupdaddy(df, 'original_title')
```

In [19]:
```
for col in clean_df.columns:
    print('Duplicates in ' + col + ': ' + str(clean_df.duplicated(col).sum()))
```

```
Duplicates in id: 0
Duplicates in imdb_id: 9
Duplicates in popularity: 48
Duplicates in budget: 10020
Duplicates in revenue: 5981
Duplicates in original_title: 0
Duplicates in cast: 141
Duplicates in director: 5571
Duplicates in keywords: 2024
Duplicates in runtime: 10328
Duplicates in genres: 8571
Duplicates in production_companies: 3315
Duplicates in release_date: 4814
Duplicates in vote_count: 9294
Duplicates in vote_average: 10499
Duplicates in release_year: 10515
Duplicates in budget_adj: 8016
Duplicates in revenue_adj: 5853
```

There, that got rid of all of the duplicate rows in the `duplicate_title` column. There are still lots of duplicates, but most of them can be attributed to common themes/actors and rounded budget and vote figures. As a final check I will look at what NaNs are left.

In [7]:
```
# Now count Nans
df.isnull().sum()
```

Out[7]:
```
id                         0
imdb_id                   10
popularity                 0
budget                     0
revenue                    0
original_title             0
cast                      74
director                  43
keywords                1467
runtime                    0
genres                    23
production_companies    1019
release_date               0
vote_count                 0
vote_average               0
release_year               0
budget_adj                 0
revenue_adj                0
dtype: int64
```

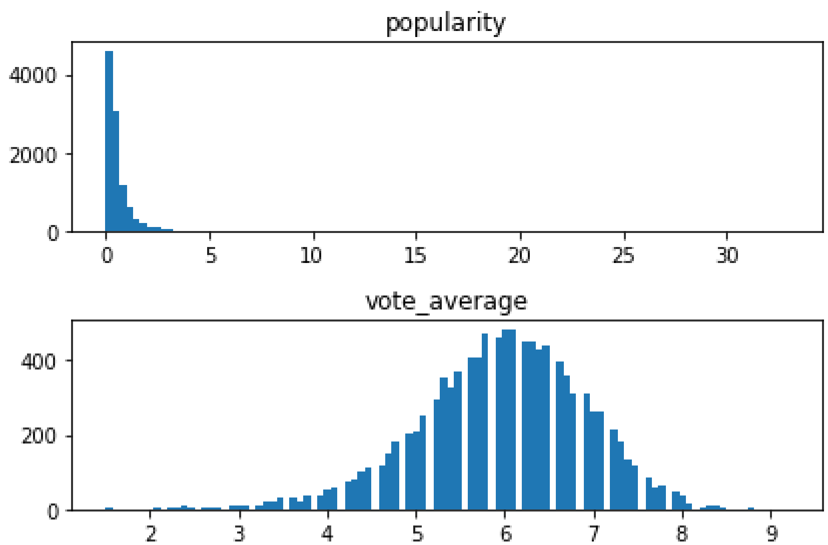
It is worth noting that although the `clean_df` dataframe is clean of relevant duplicates, it has a number of NaNs in some columns. It may be necessary to have to remove NaNs for some analyses, but since I do not plan to aggregate using the columns with NaNs, I will leave them.

Exploratory Data Analysis

Q1: Are we releasing more movies every year? Do audiences tend to rate modern movies higher or lower than older movies?

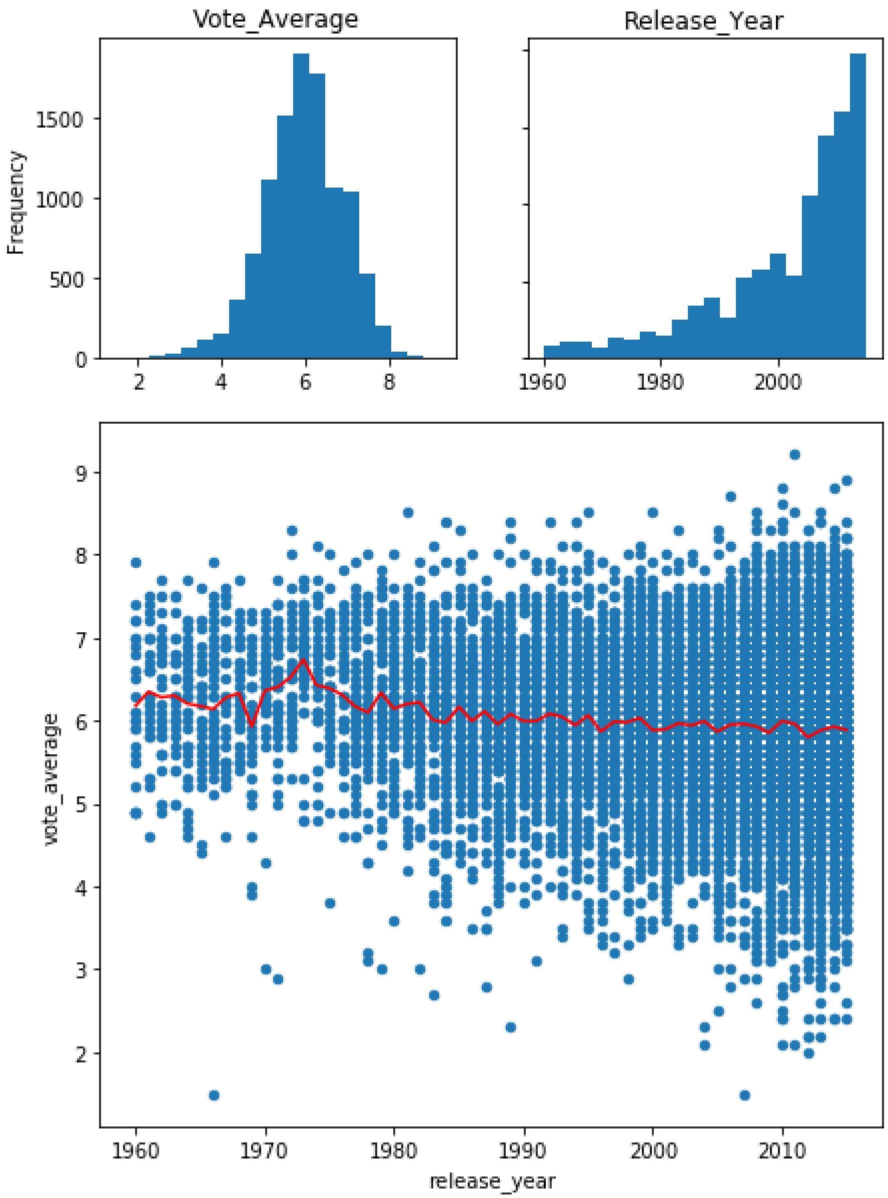
First I need to decide how I am going to evaulate popularity, since there are two columns that seem relevant: `popularity` and `vote_average`

In [30]:
```
fig_0, ax_0 = plt.subplots(2)
ax_0[0].hist(clean_df['popularity'], 100);
ax_0[1].hist(clean_df['vote_average'], 100);
ax_0[0].set_title('popularity');
ax_0[1].set_title('vote_average');
plt.tight_layout()
```



Evidently the vast majority of `popularity` values are between 0 and 1, while `vote_average` values fit a nice bell curve as one would expect. I will be using `vote_average` for my popularity metric. From now on, whenever I reference popularity, I am referring to it's average vote score.

In [8]:
```
# examine votes by year
gs = gridspec.GridSpec(3, 2)
ax1 = plt.subplot(gs[0, 0])
ax2 = plt.subplot(gs[0, 1])
ax3 = plt.subplot(gs[-2:, :])
clean_df['vote_average'].plot(kind='hist', bins=20, title='Vote_Average', ax = ax1, figsize=(7,10));
clean_df['release_year'].plot(kind='hist', bins=20, title='Release_Year', ax = ax2, sharey = ax1);
clean_df.plot.scatter(x='release_year', y='vote_average', ax = ax3);

line_x = clean_df.groupby(['release_year'])['vote_average'].mean().index
line_y = clean_df.groupby(['release_year'])['vote_average'].mean().values

ax3.plot(line_x,line_y,color='r');
```


Vote_Average
Release_Year
Frequency
vote_average
release_year

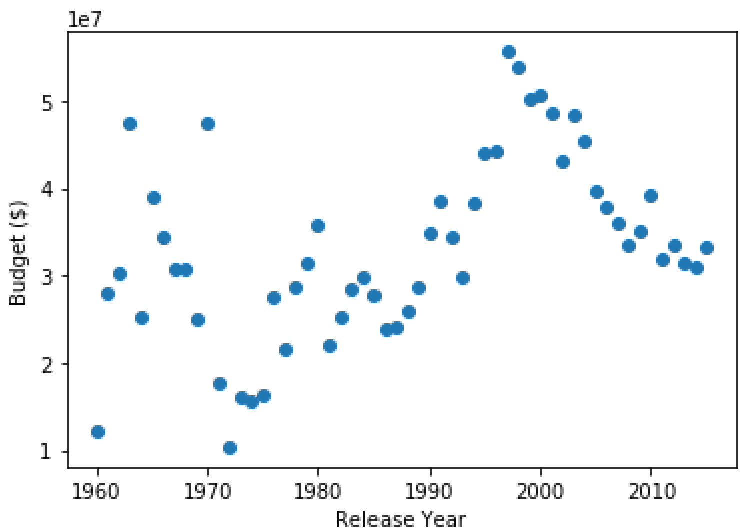
Interestingly, although the number of movies being produced per year is increasing, the mean rating for each is gradually decreasing. It is uncertain if this is related to the quality of the movie, or other factors (viewing experience, viewer factors).

To investigate this further, we can look at the adjusted budget of movies per year, and see if this has a correlation with the vote score.

Q2: Are production companies spending more on average per movie now than in previous decades?

The dataset is very kind in that it gives a column `budget_adj` which adjusts the movie budget into modern dollars, meaning inflation is removed from the equation. Recall that our dataset had a number of zeros - I am going to query the budget_adj column to exclude rows where the budget was 0.

In [9]:
```
# first look at the mean budget per movie per year
# important that we exlude listings where the budget is 0
clean_budget = clean_df.query('budget_adj > 0')
budyr = clean_budget.groupby(['release_year'])['budget_adj'].mean()

plt.scatter(x = budyr.index, y = budyr.values);
plt.xlabel('Release Year');
plt.ylabel('Budget ($)');
```



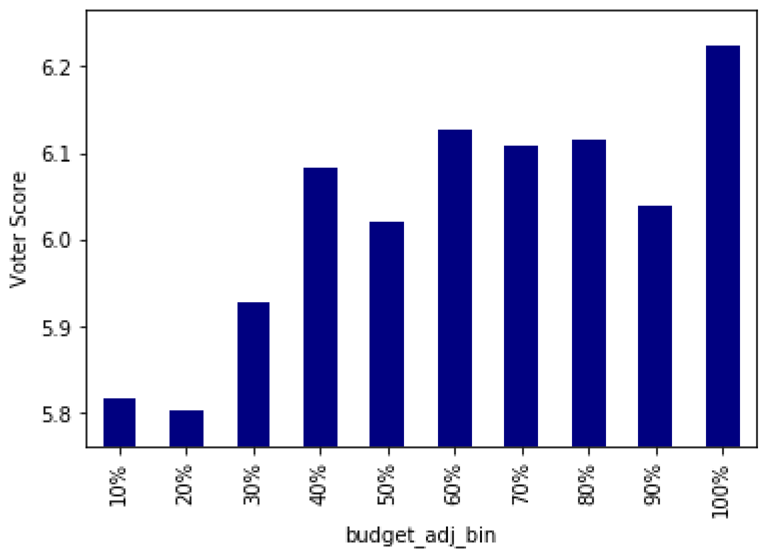
Interesting that the average budget for movies peaked around 2000, and has been decreasing since. That is not what I would have expected. Budgets began decreasing before the 2008 recession and have actually somewhat leveled out since.

Q3: Are higher budget movies more popular than lower grossing ones?

Next, I will compare movie budget with vote score. To simplify things, I will group movie budgets into quantiles, binning by decile. Note that the error from the next cell is a false positive and may safely be ignored.

In [32]:
```
# bin budget_adjusted into 10 percentiles and find the mean voter rating per bin
b_labels = ['10%', '20%', '30%', '40%', '50%', '60%', '70%', '80%', '90%', '100%']
clean_budget['budget_adj_bin'] = pd.qcut(clean_budget['budget_adj'], q=10, labels = b_labels)
# scatter(x = clean_budget['budget_adj_bin'], y = clean_budget.groupby(['budget_adj_bin'])['vote_average'].me
an())
bud_bin = clean_budget.groupby(['budget_adj_bin'])['vote_average'].mean()
bud_bin.plot(kind='bar', color = 'navy');
low = bud_bin.min()
high = bud_bin.max()
plt.ylim(low-.1*(high-low), high+.1*(high-low))
plt.ylabel('Voter Score');
```

```
C:\Users\Jesse\Anaconda3\lib\site-packages\ipykernel\__main__.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view
-versus-copy
  app.launch_new_instance()
```



Again, interesting that once you are in the top 60% of movie budgets, there is actually a negative correlation between budget and voter score until a movie is in the top 10% of movie budgets. It is possible that some 'big budget' movies relied too heavily costly effects to earn ratings, and couldn't amass enough of a budget for this to actually pay off. Again, this is speculation.

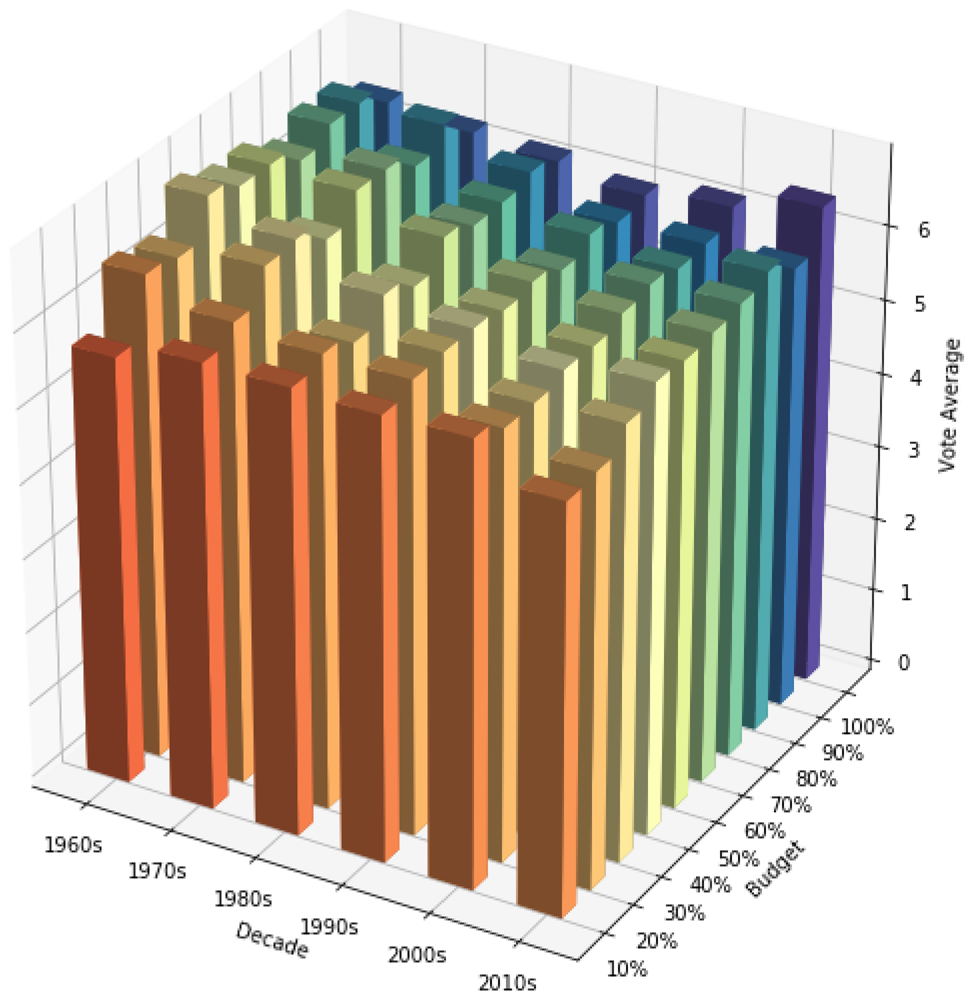
Q4: How have audiences responded to movie budget over the years? Has this changed?

Bringing the previous questions together, I will look at how movie budget has affected movie popularity since 1960. I will seek to learn if modern audiences appreciate high budget movies moreso than audiences throughout the previous decades. I am comparing 3 variables here (`release_year`, `budget_adj`, and `vote_average`) and will use a 3d graph to view them all simultaneously. Note that the error from the next cell is a false positive and may safely be ignored.

In [175]:
```
# x = years
# y = buget bucket
# z = voter rating
decades = ['1960s', '1970s', '1980s', '1990s', '2000s', '2010s']
clean_budget['decade'] = pd.cut(clean_budget['release_year'], bins = [1960, 1970, 1980, 1990, 2000, 2010, 202
0], labels = decades)

# create multi index series for plotting
cb3d = clean_budget.set_index(['decade', 'budget_adj_bin']).groupby(level=[0,1])['vote_average'].mean()
cb3d = pd.DataFrame(cb3d)

# Setup
L = []
for i, group in cb3d.groupby(level=1)['vote_average']:
    L.append(group.values)
z = np.hstack(L).ravel()

xlabels = cb3d.index.get_level_values('decade').unique()
ylabels = cb3d.index.get_level_values('budget_adj_bin').unique()
x = np.arange(xlabels.shape[0])
y = np.arange(ylabels.shape[0])

x_M, y_M = np.meshgrid(x, y, copy=False)

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')

# Making the intervals in the axes match with their respective entries
ax.w_xaxis.set_ticks(x + 0.5/2.)
ax.w_yaxis.set_ticks(y + 0.5/2.)

# Renaming the ticks as they were before
ax.w_xaxis.set_ticklabels(xlabels)
ax.w_yaxis.set_ticklabels(ylabels)

# Labeling the 3 dimensions
ax.set_xlabel('Decade')
ax.set_ylabel('Budget')
ax.set_zlabel('Vote Average')

# Choosing the range of values to be extended in the set colormap
values = np.linspace(0.2, 1., x_M.ravel().shape[0])
```

```
# Selecting an appropriate colormap
colors = plt.cm.Spectral(values)
#colors = plt.cm.jet(values.flatten()/float(values.max()))
ax.bar3d(x_M.ravel(), y_M.ravel(), z*0, dx=0.5, dy=0.5, dz=z, color=colors)
plt.show()
```

```
C:\Users\Jesse\Anaconda3\lib\site-packages\ipykernel\__main__.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view
-versus-copy
```



Here is what I suspected and was looking for. You can see that in earlier decades, low and mid budget movies got higher average vote scores, but this trend has changed post 2000. I am guessing that in the past, some movies became 'cult-classics' without a high budget, whereas now many movies rely on high budget effects to wow audiences. Again, just guessing here - there may be another reason behind this phenomenon.

Conclusion

In this investigation I evaluated the budget and popularity of movies over the years, with some surprising results. As was expected, more movies have been released every year - in fact, the trend appears exponential. However, the voter score for movies has not been increasing, and if anything it appears to be decreasing. This could have to do with nostalgia for older movies, or it could mean that since fewer movies were being released in the past, the audience was historically more likely to recognize and respond to the cast in the movie they were watching. The budget for movies has also increased over the years, but surprisingly it dropped severly post 2000. One hypothesis for this phenomenon would be that the cost of computer generated effects has diminished over the years, meaning a production company could achieve the same effects for a smaller budget. I also showed that budget does not correspond completely positively with popularity, as movies between the 60th and 90th percentiles actually show a dip in popularity compared to their counterparts on the edges of that window. Perhaps most interesting is this relationship as viewed per year - in the 1960s through 1980s, movies in the bottom half of the budget bracket were actually more popular than their more expensive counterparts. Again, I posit that this could be tied to a nostalgia/cult-classic effect, but there are too many external factors that I have not examined at play to make a significant conclusion.