Table of Contents

1 Introduction:

1.1 Citation

Much of this project and its code come directly from https://github.com/ElyLin/MovieAnalysisProject (https://github.com/ElyLin/MovieAnalysisProject) where I was a contributor. After being a contributor to the group project, I wanted to go back and re-explore my process and analysis on my own.

In my presentation, I recommend making films based on DOOM, Wolfenstein, The Elder Scrolls: Skyrim, Fallout and Dishonored, all Bethesda games that Microsoft bought the rights to in 2020. https://news.microsoft.com/2020/09/21/microsoft-to-acquire-zenimax-media-

and-its-game-publisher-bethesda-softworks/ (https://news.microsoft.com/2020/09/21/microsoft-to-acquire-zenimax-media-and-its-game-publisher-bethesda-softworks/)

1.2 Overview

Microsoft sees all the big companies creating original video content and they want to get in on the fun. They have decided to create a new movie studio, but they don't know anything about creating movies. You are charged with exploring what types of films are currently doing the best at the box office. You must then translate those findings into actionable insights that the head of Microsoft's new movie studio can use to help decide what type of films to create.

1.3 Business Understanding:

In this project, I hope to make a recommendation to Microsoft based on the Net Profit of the films. I'm using Net Profit as opposed to using Return on Investment (ROI) as my main financial data metric since Microsoft is already so large and successful that they can afford to put some investment into their films. They don't need to focus on getting a high profit from a low-budget film, they don't need to be so conservative with their investments. They can instead safely invest by following my recommendations.

I will be making recommendations based on the answers to the following questions:

1) What is the best genre to make a film in?

2) What is the best time to release the film?

3) What is the ideal runtime of a film?

2 Cleaning the Data:

2.1 Importing the libraries and various data files

In [51]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
# these are the various libraries we will need
```

In [52]:
```
# here is the raw data I will be using

movie_budgets_df = pd.read_csv('zippedData/tn.movie_budgets.csv.gz')
title_basics = pd.read_csv('zippedData/imdb.title.basics.csv.gz')
title_ratings = pd.read_csv('zippedData/imdb.title.ratings.csv.gz')
tmdb = pd.read_csv('zippedData/tmdb.movies.csv.gz')
```

2.2 Assigning genres to the genres column

In [53]:
```
# firstly, the tmbd database has it's genre's encoded so we must deco
# genre's using the below dictionary

genres={'28':"Action",
'12':"Adventure",
'16':"Animation",
'35':"Comedy",
'80':"Crime",
'99':"Documentary",
'18':"Drama",
'10751':"Family",
'14':"Fantasy",
'36':"History",
'27':"Horror",
'10402':"Music",
'9648':"Mystery",
'10749':"Romance",
'878':"Science Fiction",
'10770':"TV Movie",
'53':"Thriller",
'10752':"War",
'37':"Western"}

tmdb['genre_ids'] = tmdb['genre_ids'].replace(pd.Series(genres).astyp
                                              regex=True)
tmdb['genre_ids']=tmdb['genre_ids'].str.strip('[]').str.split(', ')

# creating a dictionary and using the key, values pairs to decode the
```

2.3 Making merges to bring the data together.

In [54]:

```
# secondly, we must merge the two IMDB dataframes

title_master = title_basics.merge(title_ratings, how='inner')
title_master.head()
```

	tconst	primary_title	original_title	start_year	runtime_minutes	genres	av
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama	7.0
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama	7.2
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama	6.9
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama	6.1
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Fantasy	6.5

In [55]:

```
# making the title and tconst ID part of the same object so I can dro
# douplicates effectively

title_master['title_and_id'] = title_master['primary_title'] +' id '
title_master['tconst']
```

In [56]:

```
# we must merge the tmdb database with the imdb database.
# we chose to do an inner merge, here because we
# are only interested in the movies that these dataframes have in com

tmdb_and_imdb = tmdb.merge(title_master, how = 'inner')
tmdb_and_imdb.head()
```

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	
0	0	[Adventure, Fantasy, Family]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Har Pot the Dea Hal Par
1	1	[Fantasy, Adventure, Animation, Family]	10191	en	How to Train Your Dragon	28.734	2010-03-26	Hov Trai Dra
2	2	[Adventure, Action, Science Fiction]	10138	en	Iron Man 2	28.515	2010-05-07	Iror
3	4	[Action, Science Fiction, Adventure]	27205	en	Inception	27.920	2010-07-16	Inc
4	5	[Adventure, Fantasy, Family]	32657	en	Percy Jackson & the Olympians: The Lightning T...	26.691	2010-02-11	Per Jacl the Oly The Lig T...

In [57]:

```
# the next dataframe we want to fold in has the column "movie" but we
# to rename it to title so the merge can happen on the title of the f
# so we must rename the column

movie_budgets_df = movie_budgets_df.rename(columns={'movie':'title'})
```

In [58]:

```
# now we want to merge the fincial dataframe from The Numbers
# with all the others based on Title, so we specify that
# with the on = 'title'

master_unclean = tmdb_and_imdb.merge\
(movie_budgets_df, on = 'title', how = 'inner')
master_unclean.head()
```

	Unnamed: 0	genre_ids	id_x	original_language	original_title	popularity	release_date_x	
0	1	[Fantasy, Adventure, Animation, Family]	10191	en	How to Train Your Dragon	28.734	2010-03-26	H T D
1	2	[Adventure, Action, Science Fiction]	10138	en	Iron Man 2	28.515	2010-05-07	I
2	4	[Action, Science Fiction, Adventure]	27205	en	Inception	27.920	2010-07-16	I
3	5	[Adventure, Fantasy, Family]	32657	en	Percy Jackson & the Olympians: The Lightning T...	26.691	2010-02-11	P J t C T L T
4	7	[Animation, Family, Comedy]	10193	en	Toy Story 3	24.445	2010-06-17	T

5 rows × 23 columns

In [59]:

```
# let's take a look at the amount of information we have now and the
# of information in our columns we can see where we have duplicate co
# we now have x and y versions to distinguish them from each other
# runtime_minutes and genres are the only columns that do not have
# 16669 non null values after this merge

master_unclean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2959 entries, 0 to 2958
Data columns (total 23 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         2959 non-null   int64  
 1   genre_ids          2959 non-null   object 
 2   id_x               2959 non-null   int64  
 3   original_language  2959 non-null   object 
 4   original_title     2959 non-null   object 
 5   popularity         2959 non-null   float64
 6   release_date_x     2959 non-null   object 
 7   title              2959 non-null   object 
 8   vote_average       2959 non-null   float64
 9   vote_count         2959 non-null   int64  
 10  tconst             2959 non-null   object 
 11  primary_title      2959 non-null   object 
 12  start_year         2959 non-null   int64  
 13  runtime_minutes    2864 non-null   float64
 14  genres             2952 non-null   object 
 15  averagerating      2959 non-null   float64
 16  numvotes           2959 non-null   int64  
 17  title_and_id       2959 non-null   object 
 18  id_y               2959 non-null   int64  
 19  release_date_y     2959 non-null   object 
 20  production_budget  2959 non-null   object 
 21  domestic_gross     2959 non-null   object 
 22  worldwide_gross    2959 non-null   object 
dtypes: float64(4), int64(6), object(13)
memory usage: 554.8+ KB
```

2.4 Limiting the data on runtime

In [60]:
```
# now we want to exclude movies from this master dataframe which are
# than 45 min and more than three hours, because we want to base the
# recomendations on films that are typically seen in the movie theate
# and put that information on a new dataframe called run_time

run_time = master_unclean.loc[(master_unclean['runtime_minutes'] > 45
                               & (master_unclean['runtime_minutes'] <
run_time.head()
```

	Unnamed: 0	genre_ids	id_x	original_language	original_title	popularity	release_date_x	
0	1	[Fantasy, Adventure, Animation, Family]	10191	en	How to Train Your Dragon	28.734	2010-03-26	H T D
1	2	[Adventure, Action, Science Fiction]	10138	en	Iron Man 2	28.515	2010-05-07	I
2	4	[Action, Science Fiction, Adventure]	27205	en	Inception	27.920	2010-07-16	I
3	5	[Adventure, Fantasy, Family]	32657	en	Percy Jackson & the Olympians: The Lightning T...	26.691	2010-02-11	P J t C T L T
4	7	[Animation, Family, Comedy]	10193	en	Toy Story 3	24.445	2010-06-17	T

5 rows × 23 columns

2.5 Dropping unnecessary columns

In [61]:

```
# here, we want to drop some columns where the information is irreliv
# or duplicated. I'm dripping id_x and id_y since I don't need anothe
# identifyer,  title, original title since I already have the title c
# need,  start year, year and release_date_x since I already have a d
# column, genres since the genre_ids column I already have is more ro
# and all the rating a popularity information since I won't be using
# for my analysis

df = master_unclean.drop\
(['Unnamed: 0', 'id_x', 'original_title', 'start_year', 'id_y',\
'genres', 'release_date_x', 'primary_title', 'original_language', \
'popularity', 'vote_average', 'vote_count', 'numvotes', 'averageratin
  'title', 'tconst'],
 axis = 1)
```

In [62]:

```
#let's take a look at where our dataframe is right now

df.head()
```

	genre_ids	runtime_minutes	title_and_id	release_date_y	production_budget	domestic_gross
0	[Fantasy, Adventure, Animation, Family]	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	$165,000,000	$217,581,232
1	[Adventure, Action, Science Fiction]	124.0	Iron Man 2 id tt1228705	May 7, 2010	$170,000,000	$312,433,331
2	[Action, Science Fiction, Adventure]	148.0	Inception id tt1375666	Jul 16, 2010	$160,000,000	$292,576,195
3	[Adventure, Fantasy, Family]	118.0	Percy Jackson & the Olympians: The Lightning T...	Feb 12, 2010	$95,000,000	$88,768,303
4	[Animation, Family, Comedy]	103.0	Toy Story 3 id tt0435761	Jun 18, 2010	$200,000,000	$415,004,880

2.6 Formatting the Remaining Data

In [63]:
```
# this function will take a string value with commas and dollar signs
# return an integer value in a format that I can do math upon

def reformat(string_to_money):
    string_to_money = string_to_money.replace(',', '').replace('$', '
    string_to_money = int(string_to_money)
    return string_to_money
```

In [64]:
```
#reformatting the production budget, domestic gross and worldwide gro

df['production_budget'] = df['production_budget'].map(reformat)
df['domestic_gross'] = df['domestic_gross'].map(reformat)
df['worldwide_gross'] =df['worldwide_gross'].map(reformat)
```

In [65]:
```
#checking to make sure my finacial data is in integer form

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2959 entries, 0 to 2958
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   genre_ids          2959 non-null   object
 1   runtime_minutes    2864 non-null   float64
 2   title_and_id       2959 non-null   object
 3   release_date_y     2959 non-null   object
 4   production_budget  2959 non-null   int64
 5   domestic_gross     2959 non-null   int64
 6   worldwide_gross    2959 non-null   int64
dtypes: float64(1), int64(3), object(3)
memory usage: 184.9+ KB
```

In [66]:
```
# creating total gross, net_profit and ROI as finacial metrics. I wil
# net profit as my main metric but I wanted to create total gross and
# for any additional analysis in my conclusion

df['total_gross'] = df['domestic_gross'] + df['worldwide_gross']
df['net_profit']= df['total_gross'] - df['production_budget']
df['ROI'] = (df['net_profit'] / df['production_budget']) * 100
```

2.7 Checking for Null Values

In [67]:
```
# seeing how many columns have null values

df.isna().sum()
```

```
genre_ids             0
runtime_minutes      95
title_and_id          0
release_date_y        0
production_budget     0
domestic_gross        0
worldwide_gross       0
total_gross           0
net_profit            0
ROI                   0
dtype: int64
```

I'm going to drop the rows where the runtime data is missing, it's a small section of my dataset and since I will be making recommendations based on runtime, I don't want to throw the data off.

In [68]:
```
# dropping rows with null values

df.dropna(axis=0, inplace= True)
```

In [69]:
```
#to check if the 34 rows dropped and we have no null values

df.isna().sum()
```

```
genre_ids            0
runtime_minutes      0
title_and_id         0
release_date_y       0
production_budget    0
domestic_gross       0
worldwide_gross      0
total_gross          0
net_profit           0
ROI                  0
dtype: int64
```

In [70]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2864 entries, 0 to 2958
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   genre_ids          2864 non-null   object 
 1   runtime_minutes    2864 non-null   float64
 2   title_and_id       2864 non-null   object 
 3   release_date_y     2864 non-null   object 
 4   production_budget  2864 non-null   int64  
 5   domestic_gross     2864 non-null   int64  
 6   worldwide_gross    2864 non-null   int64  
 7   total_gross        2864 non-null   int64  
 8   net_profit         2864 non-null   int64  
 9   ROI                2864 non-null   float64
dtypes: float64(2), int64(5), object(3)
memory usage: 246.1+ KB
```

2.8 Removing duplicates

In [71]:
```
1 # to show how many duplicates we have as a slice of the current df
2
3 df[df.duplicated(subset = df.drop('genre_ids', axis = 1).columns,
4                  )].sort_values(by = 'title_and_id')
```

	genre_ids	runtime_minutes	title_and_id	release_date_y	production_budget	domest
1529	[Horror]	84.0	24 Hours to Die id tt3747550	Sep 18, 2015	2000000	2583301
1525	[Crime, Drama, Thriller]	84.0	24 Hours to Die id tt3747550	Sep 18, 2015	2000000	2583301
2667	[Comedy]	104.0	A Bad Moms Christmas id tt6359956	Nov 1, 2017	28000000	7211065
656	[Documentary]	98.0	A Better Life id tt1554091	Jun 24, 2011	10000000	1759252
2432	[Drama, Fantasy]	108.0	A Monster Calls id tt3416532	Jan 6, 2017	43000000	3740823
...	...	...	...	...	...	...

In [72]:
```
# dropping duplicate rows that are about the same film by using the a
# subset (which checks for everything except matching values in the
# 'genre_ids' columns.)

df.drop_duplicates(subset = df.drop('genre_ids', axis = 1).columns,
                   inplace = True)
```

In [73]:
```
# checking to make sure the duplicates were dropped.

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2201 entries, 0 to 2955
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   genre_ids          2201 non-null   object 
 1   runtime_minutes    2201 non-null   float64
 2   title_and_id       2201 non-null   object 
 3   release_date_y     2201 non-null   object 
 4   production_budget  2201 non-null   int64  
 5   domestic_gross     2201 non-null   int64  
 6   worldwide_gross    2201 non-null   int64  
 7   total_gross        2201 non-null   int64  
 8   net_profit         2201 non-null   int64  
 9   ROI                2201 non-null   float64
dtypes: float64(2), int64(5), object(3)
memory usage: 189.1+ KB
```

In [74]:
```
1 df
```

	genre_ids	runtime_minutes	title_and_id	release_date_y	production_budget	domestic_g
0	[Fantasy, Adventure, Animation, Family]	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
1	[Adventure, Action, Science Fiction]	124.0	Iron Man 2 id tt1228705	May 7, 2010	170000000	312433331
2	[Action, Science Fiction, Adventure]	148.0	Inception id tt1375666	Jul 16, 2010	160000000	292576195
3	[Adventure, Fantasy, Family]	118.0	Percy Jackson & the Olympians: The Lightning T...	Feb 12, 2010	95000000	88768303
4	[Animation, Family, Comedy]	103.0	Toy Story 3 id tt0435761	Jun 18, 2010	200000000	415004880
...	...	...	...	...	...	...
2951	[Horror, Mystery, Thriller]	145.0	3 id tt2123129	Sep 16, 2011	7200000	59774
2952	[Horror, Mystery, Thriller]	115.0	3 id tt2379653	Sep 16, 2011	7200000	59774
2953	[Horror, Mystery, Thriller]	80.0	3 id tt5618160	Sep 16, 2011	7200000	59774
2954	[Action, Science Fiction]	91.0	Molly id tt6522982	Oct 22, 1999	21000000	17396
2955	[]	89.0	The Box id tt1728200	Nov 6, 2009	25000000	15051977

2201 rows × 10 columns

After cleaning and removing null values and duplicates, this analysis will use a dataset the size of 2,864 films.

3 Recommendations:

3.1 Genre recommendations

In [75]:
```
#first we need to use .explode to make a row for each genre that each
#falls into and so we can compare genres side by side based on total
#net profit and ROI

df_exploded = df.explode('genre_ids', ignore_index=False)
df_exploded.head()
```

	genre_ids	runtime_minutes	title_and_id	release_date_y	production_budget	domestic_gross
0	Fantasy	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Adventure	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Animation	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Family	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
1	Adventure	124.0	Iron Man 2 id tt1228705	May 7, 2010	170000000	312433331

In [76]:
```
# seeing the shape of the exploded dataframe

df_exploded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5118 entries, 0 to 2955
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   genre_ids          5118 non-null   object 
 1   runtime_minutes    5118 non-null   float64
 2   title_and_id       5118 non-null   object 
 3   release_date_y     5118 non-null   object 
 4   production_budget  5118 non-null   int64  
 5   domestic_gross     5118 non-null   int64  
 6   worldwide_gross    5118 non-null   int64  
 7   total_gross        5118 non-null   int64  
 8   net_profit         5118 non-null   int64  
 9   ROI                5118 non-null   float64
dtypes: float64(2), int64(5), object(3)
memory usage: 439.8+ KB
```

In [77]:

```
# slicing out the rows where the genre was left blank
# (these rows didn't pop as null values although they are equivilant

null_genre = df_exploded.loc[df_exploded['genre_ids'] == '']
null_genre
```

	genre_ids	runtime_minutes	title_and_id	release_date_y	production_budget	domestic_gı
360		80.0	Plastic id tt1691450	Sep 26, 2014	10000000	0
361		78.0	Plastic id tt2554700	Sep 26, 2014	10000000	0
362		102.0	Plastic id tt2556874	Sep 26, 2014	10000000	0
373		85.0	Monster id tt1617136	Dec 24, 2003	5000000	34469210
374		90.0	Monster id tt6840904	Dec 24, 2003	5000000	34469210
...	...	...	...	...	...	...
1954		90.0	Red id tt8851190	Oct 15, 2010	60000000	90380162
1957		88.0	Breaking In id tt7137846	May 11, 2018	6000000	46840590
2280		82.0	Diamond Ruff id tt2111292	Dec 31, 2014	500000	0
2304		75.0	Graduation Day id tt2616818	May 1, 1981	250000	23894000
2955		89.0	The Box id tt1728200	Nov 6, 2009	25000000	15051977

62 rows × 10 columns

In [78]:

```
# removing the rows we sliced above

df_exploded.drop(labels = null_genre.index, inplace = True)
```

In [79]:
```
# checking to make sure those rows were removed

df_exploded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5056 entries, 0 to 2954
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   genre_ids          5056 non-null   object 
 1   runtime_minutes    5056 non-null   float64
 2   title_and_id       5056 non-null   object 
 3   release_date_y     5056 non-null   object 
 4   production_budget  5056 non-null   int64  
 5   domestic_gross     5056 non-null   int64  
 6   worldwide_gross    5056 non-null   int64  
 7   total_gross        5056 non-null   int64  
 8   net_profit         5056 non-null   int64  
 9   ROI                5056 non-null   float64
dtypes: float64(2), int64(5), object(3)
memory usage: 434.5+ KB
```

In [80]:

```
# making a table that shows all the genres and their adverage values,
# sorted by net profit

by_genre_net = df_exploded.groupby(by='genre_ids', axis=0).mean()\
.sort_values('net_profit', ascending=False)
by_genre_net
```

genre_ids	runtime_minutes	production_budget	domestic_gross	worldwide_gross	total_gro
Animation	95.351351	9.040991e+07	1.328418e+08	3.590243e+08	4.918661e+
Adventure	110.701538	1.010752e+08	1.124018e+08	3.239660e+08	4.363677e+
Fantasy	108.731959	9.148041e+07	1.071063e+08	2.980314e+08	4.051378e+
Family	98.774869	8.051963e+07	1.074798e+08	2.838668e+08	3.913466e+
Science Fiction	107.185185	6.833975e+07	7.924039e+07	2.139633e+08	2.932037e+
Action	108.844130	7.340315e+07	7.896383e+07	2.152858e+08	2.942496e+
Comedy	100.841244	3.535768e+07	5.192640e+07	1.169176e+08	1.688441e+
War	112.522727	3.870682e+07	4.464108e+07	9.413630e+07	1.387774e+
Crime	107.146245	3.452103e+07	3.941484e+07	9.185459e+07	1.312694e+
Western	109.148148	5.058519e+07	4.334205e+07	1.039884e+08	1.473305e+
Music	107.239130	2.490000e+07	4.031065e+07	8.124665e+07	1.215573e+
TV Movie	98.000000	1.791667e+07	3.478072e+07	7.832768e+07	1.131084e+
Thriller	102.034539	2.954697e+07	3.376534e+07	8.486340e+07	1.186287e+
Romance	105.089069	2.428330e+07	3.439347e+07	7.555306e+07	1.099465e+
Mystery	101.191011	2.669610e+07	3.235570e+07	7.815538e+07	1.105111e+
Drama	107.205941	2.650487e+07	3.213199e+07	7.203307e+07	1.041651e+
Horror	94.375000	1.627641e+07	2.450560e+07	5.524297e+07	7.974857e+
History	121.541667	2.824583e+07	2.866987e+07	5.874817e+07	8.741805e+
Documentary	92.471264	1.268425e+07	1.667170e+07	3.203684e+07	4.870855e+

In [81]:

```
# making a plot to visualize the above table so we can easily see whi
# genres dominate net profits

values = by_genre_net['net_profit']
colors = ['grey' if (x < 2.838668e+08   ) else 'blue' for x in values

sns.set_style(style = 'white')

plt.figure(figsize=(8,8))

ax = sns.barplot(data = df_exploded, x = "net_profit", y = "genre_ids
           order = by_genre_net.index, palette=colors)
ax.set(xlabel="Average Net Profit in Hundred Millions", ylabel = "Gen
       title = "Net Profit by Genre")

# making sure my labels don't get cut-off

plt.tight_layout();

# specifying the scientific notation to units in regular English
# investigate this blank bar
```

The takeaway here is that animation, adventure, fantasy and family are the top four genres by close to 70 million dollars.

3.1.1 CITATION: color coding

adapted from: https://stackoverflow.com/questions/31074758/how-to-set-a-different-color-to-the-largest-bar-in-a-seaborn-barplot (https://stackoverflow.com/questions/31074758/how-to-set-a-different-color-to-the-largest-bar-in-a-seaborn-barplot)

3.2 Release Date

In [82]:
```
# making a new series called release_date that only contains films if
# in the Animation, Adventure, Fantasy or Family genres for further a

release_date = df_exploded['genre_ids'].isin(['Animation', 'Adventure
                                              'Fantasy', 'Family'])

release_date.value_counts()
```

```
False    4235
True      821
Name: genre_ids, dtype: int64
```

In [83]:
```
# slicing out the series from df_exploded and making it it's own data

release_date = df_exploded[release_date]
release_date.head()
```

	genre_ids	runtime_minutes	title_and_id	release_date_y	production_budget	domestic_gross
0	Fantasy	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Adventure	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Animation	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Family	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
1	Adventure	124.0	Iron Man 2 id tt1228705	May 7, 2010	170000000	312433331

In [84]:

```
# turning release_date_y into datetime format

release_date['date_time'] = pd.to_datetime(release_date['release_date
release_date.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 821 entries, 0 to 2947
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   genre_ids          821 non-null    object
 1   runtime_minutes    821 non-null    float64
 2   title_and_id       821 non-null    object
 3   release_date_y     821 non-null    object
 4   production_budget  821 non-null    int64
 5   domestic_gross     821 non-null    int64
 6   worldwide_gross    821 non-null    int64
 7   total_gross        821 non-null    int64
 8   net_profit         821 non-null    int64
 9   ROI                821 non-null    float64
 10  date_time          821 non-null    datetime64[ns]
dtypes: datetime64[ns](1), float64(2), int64(5), object(3)
memory usage: 77.0+ KB



<ipython-input-84-6f9bb6a9cb62>: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: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  release_date['date_time'] = pd.to_datetime(release_date['release_date_y'])
```

In [85]:
```
#isolating the release month from that datetime column

release_date['month_num'] = release_date['date_time'].dt.month
release_date.head()
```

```
<ipython-input-85-11899e521ef7>: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: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  release_date['month_num'] = release_date['date_time'].dt.month
```

	genre_ids	runtime_minutes	title_and_id	release_date_y	production_budget	domestic_gross
0	Fantasy	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Adventure	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Animation	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Family	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
1	Adventure	124.0	Iron Man 2 id tt1228705	May 7, 2010	170000000	312433331

In [86]:

```
# putting a column for the name of the month for readability

release_date['month_name'] = release_date['date_time'].dt.month_name(
release_date.head()
```

```
<ipython-input-86-152d7827001d>: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: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  release_date['month_name'] = release_date['date_time'].dt.month_name()
```

	genre_ids	runtime_minutes	title_and_id	release_date_y	production_budget	domestic_gross
0	Fantasy	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Adventure	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Animation	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Family	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
1	Adventure	124.0	Iron Man 2 id tt1228705	May 7, 2010	170000000	312433331

In [87]:

```
#making a pivot table to aggrigate the data based on genre

pivot = pd.pivot_table(release_date, values='net_profit', index='mont
                       columns='genre_ids')
pivot
```

genre_ids	Adventure	Animation	Family	Fantasy
month_name				
April	3.639112e+08	3.594065e+08	3.652238e+08	4.168243e+08
August	1.785756e+08	5.650822e+07	8.087768e+07	2.192492e+08
December	3.378595e+08	2.264141e+08	1.731402e+08	3.304541e+08
February	1.855495e+08	3.756855e+08	2.598824e+08	1.908724e+08
January	1.220691e+08	1.907414e+08	1.482089e+08	7.999303e+07
July	3.783961e+08	6.644173e+08	3.952794e+08	2.014382e+08
June	5.011279e+08	7.614652e+08	5.118351e+08	4.908190e+08
March	2.832314e+08	5.556345e+08	4.403053e+08	3.973046e+08
May	5.635206e+08	5.934604e+08	3.419126e+08	3.425121e+08
November	4.324922e+08	4.142573e+08	3.082870e+08	4.424158e+08
October	1.695030e+08	1.735661e+08	1.292997e+08	9.471312e+07
September	1.169505e+08	2.287723e+08	2.049232e+08	2.056765e+08

In [88]:

```
# I need to make a "total" column temporarily so that I can use those
# values to order the months in the pivot table

pivot['total'] = pivot['Adventure'] + pivot['Animation'] + \
                    pivot['Family'] +  pivot['Fantasy']
pivot
```

genre_ids	Adventure	Animation	Family	Fantasy	total
month_name					
April	3.639112e+08	3.594065e+08	3.652238e+08	4.168243e+08	1.505366e+09
August	1.785756e+08	5.650822e+07	8.087768e+07	2.192492e+08	5.352107e+08
December	3.378595e+08	2.264141e+08	1.731402e+08	3.304541e+08	1.067868e+09
February	1.855495e+08	3.756855e+08	2.598824e+08	1.908724e+08	1.011990e+09
January	1.220691e+08	1.907414e+08	1.482089e+08	7.999303e+07	5.410124e+08
July	3.783961e+08	6.644173e+08	3.952794e+08	2.014382e+08	1.639531e+09
June	5.011279e+08	7.614652e+08	5.118351e+08	4.908190e+08	2.265247e+09
March	2.832314e+08	5.556345e+08	4.403053e+08	3.973046e+08	1.676476e+09
May	5.635206e+08	5.934604e+08	3.419126e+08	3.425121e+08	1.841406e+09
November	4.324922e+08	4.142573e+08	3.082870e+08	4.424158e+08	1.597452e+09
October	1.695030e+08	1.735661e+08	1.292997e+08	9.471312e+07	5.670819e+08
September	1.169505e+08	2.287723e+08	2.049232e+08	2.056765e+08	7.563226e+08

In [89]:

```
# sorting by those totalled values

pivot.sort_values('total', inplace = True, ascending=False)
pivot
```

genre_ids month_name	Adventure	Animation	Family	Fantasy	total
June	5.011279e+08	7.614652e+08	5.118351e+08	4.908190e+08	2.265247e+09
May	5.635206e+08	5.934604e+08	3.419126e+08	3.425121e+08	1.841406e+09
March	2.832314e+08	5.556345e+08	4.403053e+08	3.973046e+08	1.676476e+09
July	3.783961e+08	6.644173e+08	3.952794e+08	2.014382e+08	1.639531e+09
November	4.324922e+08	4.142573e+08	3.082870e+08	4.424158e+08	1.597452e+09
April	3.639112e+08	3.594065e+08	3.652238e+08	4.168243e+08	1.505366e+09
December	3.378595e+08	2.264141e+08	1.731402e+08	3.304541e+08	1.067868e+09
February	1.855495e+08	3.756855e+08	2.598824e+08	1.908724e+08	1.011990e+09
September	1.169505e+08	2.287723e+08	2.049232e+08	2.056765e+08	7.563226e+08
October	1.695030e+08	1.735661e+08	1.292997e+08	9.471312e+07	5.670819e+08
January	1.220691e+08	1.907414e+08	1.482089e+08	7.999303e+07	5.410124e+08
August	1.785756e+08	5.650822e+07	8.087768e+07	2.192492e+08	5.352107e+08

In [90]:

```
# removing the total column once it's served it's purpose so I can pr
# graph the information within this aggregated data

pivot.drop(columns = 'total', inplace = True)
pivot
```

genre_ids	Adventure	Animation	Family	Fantasy
month_name				
June	5.011279e+08	7.614652e+08	5.118351e+08	4.908190e+08
May	5.635206e+08	5.934604e+08	3.419126e+08	3.425121e+08
March	2.832314e+08	5.556345e+08	4.403053e+08	3.973046e+08
July	3.783961e+08	6.644173e+08	3.952794e+08	2.014382e+08
November	4.324922e+08	4.142573e+08	3.082870e+08	4.424158e+08
April	3.639112e+08	3.594065e+08	3.652238e+08	4.168243e+08
December	3.378595e+08	2.264141e+08	1.731402e+08	3.304541e+08
February	1.855495e+08	3.756855e+08	2.598824e+08	1.908724e+08
September	1.169505e+08	2.287723e+08	2.049232e+08	2.056765e+08
October	1.695030e+08	1.735661e+08	1.292997e+08	9.471312e+07
January	1.220691e+08	1.907414e+08	1.482089e+08	7.999303e+07
August	1.785756e+08	5.650822e+07	8.087768e+07	2.192492e+08

In [91]:

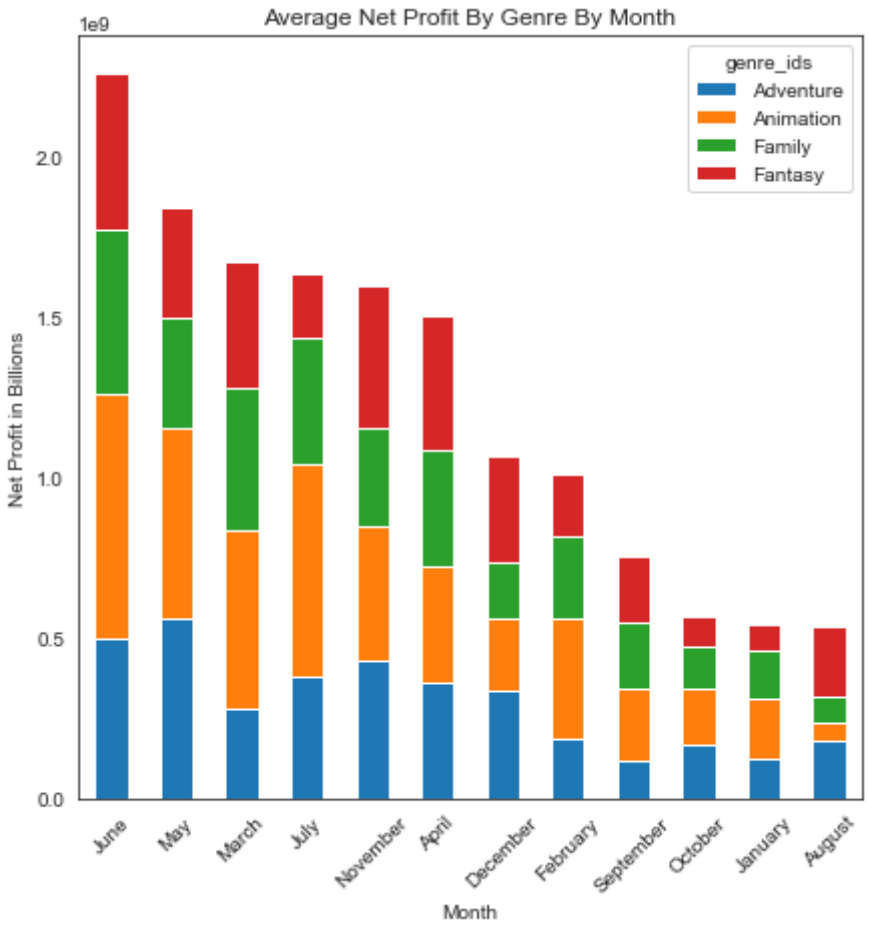
```
# making a plot to show the above pivot table as a stacked bar chart

ax = pivot.plot(kind='bar', figsize=(7, 7), stacked=True)

#ax.axhline(1500000000, color = "blue")
#ax.axhline(1900000000, color = "green")

ax.set(xlabel = "Month", ylabel = "Net Profit in Billions",
       title = "Average Net Profit By Genre By Month")

plt.xticks(rotation = 45);
```



The takeaway here is to release films in June, where the average film in this genre made over 1.9 billion dollars (shown by the green line.) If there is a delay in production due to Covid-19, I'd recommend these months as backup choices: May, March, July, November or April.

Interesting find: Fantasy films do not do well in July compared to other high earning months.

3.3 Runtime of Film

In [92]:
```
# taking a look at the shape of release_date

release_date.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 821 entries, 0 to 2947
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   genre_ids          821 non-null    object        
 1   runtime_minutes    821 non-null    float64       
 2   title_and_id       821 non-null    object        
 3   release_date_y     821 non-null    object        
 4   production_budget  821 non-null    int64         
 5   domestic_gross     821 non-null    int64         
 6   worldwide_gross    821 non-null    int64         
 7   total_gross        821 non-null    int64         
 8   net_profit         821 non-null    int64         
 9   ROI                821 non-null    float64       
 10  date_time          821 non-null    datetime64[ns]
 11  month_num          821 non-null    int64         
 12  month_name         821 non-null    object        
dtypes: datetime64[ns](1), float64(2), int64(6), object(4)
memory usage: 89.8+ KB
```

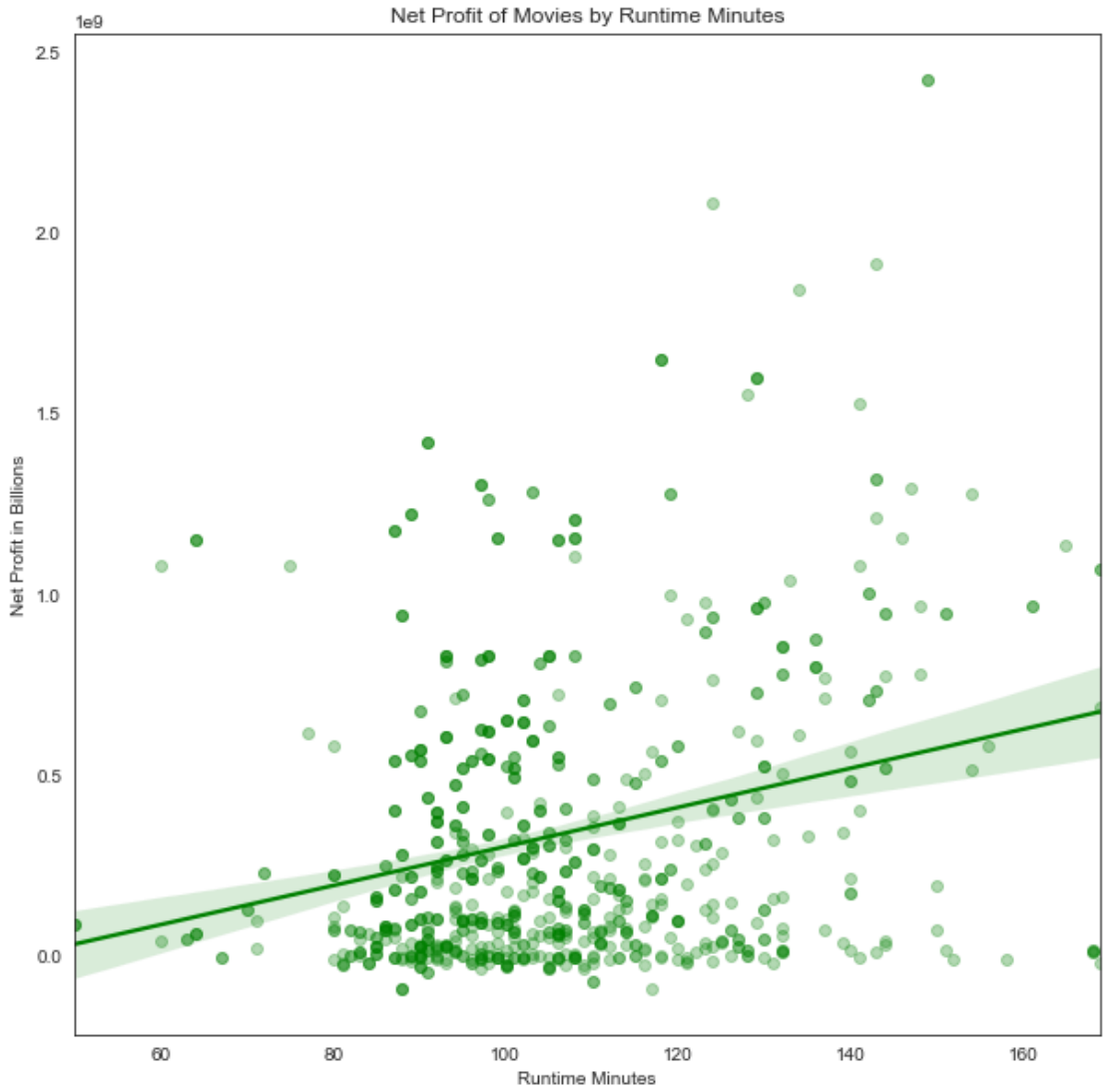
In [102]:

```
# all movies in this data set are between 30 and 180
# making a visualization to represent the films in this dataset based
# on Net Profit in billions and runtime.

plt.figure(figsize=(10,10))

runtime_plot = sns.regplot(data = release_date, x = 'runtime_minutes'
                 y = 'net_profit', color = 'green',
                 scatter_kws={'alpha':0.3})

runtime_plot.set(xlabel = 'Runtime Minutes', ylabel = 'Net Profit in
      title = "Net Profit of Movies by Runtime Minutes");
```



This doesn't tell us very much so instead let's sort runtime minutes into

categorical data instead

In [94]:
```
# Making a list of bin edges

bins = [ 60, 90, 120, 150, 180]
bins
```

```
[60, 90, 120, 150, 180]
```

In [95]:
```
# Making a list of labels to go with those bins

labels = ['Sixty', 'One hundred twenty',
          'One hundred fifty', 'One hundred eighty']
```

In [103]:
```
# cutting the data into those bins, using both bins and labels

cut_data, bin_edges = pd.cut(release_date['runtime_minutes'], bins =
                             labels = labels, retbins = True)
bin_edges
```

```
array([ 60,  90, 120, 150, 180])
```

In [105]:

```
# Taking a look to see if our bins are a new column and labeled

release_date['runtime_bins']= cut_data
release_date.head()
```

```
<ipython-input-105-1b6b290c6298>:1: 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: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  release_date['runtime_bins']= cut_data
```

	genre_ids	runtime_minutes	title_and_id	release_date_y	production_budget	domestic_gross
0	Fantasy	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Adventure	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Animation	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
0	Family	98.0	How to Train Your Dragon id tt0892769	Mar 26, 2010	165000000	217581232
1	Adventure	124.0	Iron Man 2 id tt1228705	May 7, 2010	170000000	312433331

In [97]:
```
runtime_df = release_date.groupby(by='runtime_bins').mean()
runtime_df = runtime_df.reset_index()
runtime_df
```

	runtime_bins	runtime_minutes	production_budget	domestic_gross	worldwide_gross	tota
0	Sixty	84.415584	5.798247e+07	7.235175e+07	2.091152e+08	2.8146
1	One hundred twenty	102.746032	8.905279e+07	1.043638e+08	2.870365e+08	3.9140
2	One hundred fifty	132.417266	1.387950e+08	1.827653e+08	4.979914e+08	6.8075
3	One hundred eighty	161.157895	1.377368e+08	1.543482e+08	5.215815e+08	6.7592

In [98]:
```
# taking a look at the average net profit in hundred-thousands by bin

runtime_df['net_profit']
```

```
0    2.234845e+08
1    3.023476e+08
2    5.419617e+08
3    5.381929e+08
Name: net_profit, dtype: float64
```

In [99]:
```
# sorting this aggregated data from highest to lowest by Net Profit

runtime_df.sort_values(by = 'net_profit', inplace = True)
runtime_df
```

	runtime_bins	runtime_minutes	production_budget	domestic_gross	worldwide_gross	tota
0	Sixty	84.415584	5.798247e+07	7.235175e+07	2.091152e+08	2.8146
1	One hundred twenty	102.746032	8.905279e+07	1.043638e+08	2.870365e+08	3.9140
3	One hundred eighty	161.157895	1.377368e+08	1.543482e+08	5.215815e+08	6.7592
2	One hundred fifty	132.417266	1.387950e+08	1.827653e+08	4.979914e+08	6.8075

In [100]:

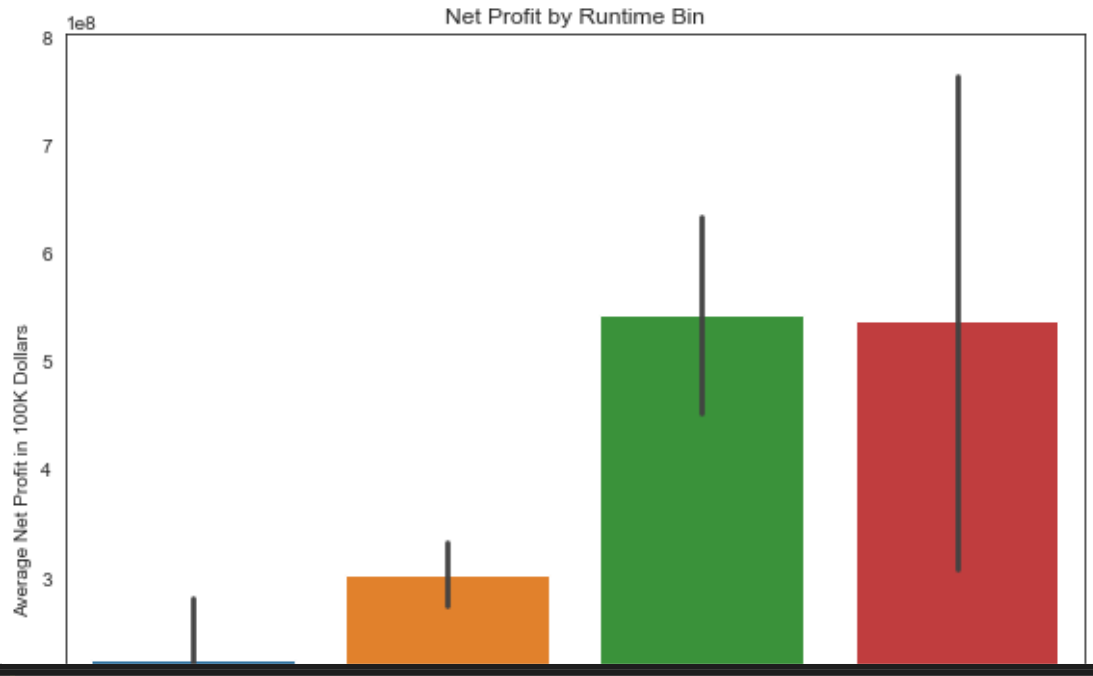
```
# visualizing the above data

plt.figure(figsize=(8,8))

ax = sns.barplot(data = release_date, x = 'runtime_bins', y = 'net_pr
plt.xticks(rotation=45)

ax.set(xlabel = 'Runtime Bins', ylabel = 'Average Net Profit in 100K
       title = "Net Profit by Runtime Bin")

plt.tight_layout();
# this will help my labels not get cut-off
```



Here the important piece of information is that films between 120-150 minutes do better consistently, and movies between 150-180 movies do about as well on average but have a much higher rate of deviation. (It's more likely that films in that range will do either much better or much worse than films in the previous bin.

4 Conclusion:

4.1 Three Business Recommendations:

My three business recommendations to Microsoft Studios would be:

1) Focus on making films that are a combination of animation, fantasy, adventure, and family. There are many examples of the combination of these genres already, (Pixar Studios being the best example of having luck here.)

2) Release this film in June, May, March of any genre, or July with an emphasis on non-Fantasy films (which, interestingly do much better in November.)

3) Make the movie somewhere within 120-150 minutes long since movies in this range tend to make the most net profit on average with the most consistency.

4.2 Future Work

Future work: I'd love to, if the project parameters were a bit larger, take a look at what studios tend to do best within the three recommendations I've already given. I'd also love an opportunity to work with review data and see how films that fit these three categories are normally rated and reviewed.