Computing Vision Original Video Content Analysis

OVERVIEW

This project profers four data driven recommendations for Computing Vision to support their diversification into the original-video content market. Computing vision can use the recommendations made within this analysis to influence their budgeting decisions, genre selection, and market entry strategy

BUSINESS PROBLEM

Computing Vision is diversifying into the original movie content market.We have been tasked with creating data driven recommendations that will influence the outcome of the newly formed studio. Some business questions include: What kind of movies do people want to watch? How much should be spent?

The goal of this project is to provide data driven recommendations that could potentially influence the outcome of the Computing Vision newly formed studio.

DATA UNDERSTANDING

The data used for this analysis came from Box Office, IMDB, Rotten Tomatoes, Movie DB, and Numbers. The datasets contain thousands of information describing different details about movies released within the past decade years. Some of the descriptors include, ratings, genres, budget, movie title, directors, movie revenue amongst others.

This project leverages on the dataset from IMDB (called movie in this analysis) database and the budgets dataset. The IMDB database contains an array of relational tables with information on movie genres, production date, movie directors, runtime among others. The budgets dataset contains financial information about movies made from 1915 to 2020

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

In [2]:
```
movie_budget = pd.read_csv("Datasets/tn.movie_budgets.csv.gz")
```

In [3]:

```python
import sqlite3

from zipfile import ZipFile

with ZipFile("Datasets/im.db.zip", 'r') as zObject:
    zObject.extractall(
        path="IMDB/")
conn = sqlite3.connect("IMDB/im.db")
cur = conn.cursor()
imdb= pd.read_sql('''
    SELECT *
    FROM persons;

''', conn)
data =  """
SELECT *
FROM movie_ratings AS "mr"
LEFT JOIN movie_basics AS "m"
    ON m.movie_ID=mr.movie_ID

"""
movie_data = pd.read_sql(data, conn)
```

In [4]:

```python
movie_budget.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   id                 5782 non-null   int64 
 1   release_date       5782 non-null   object
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   object
 4   domestic_gross     5782 non-null   object
 5   worldwide_gross    5782 non-null   object
dtypes: int64(1), object(5)
memory usage: 271.2+ KB
```

In [5]:
```
movie_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   movie_id         73856 non-null  object 
 1   averagerating    73856 non-null  float64
 2   numvotes         73856 non-null  int64  
 3   movie_id         73856 non-null  object 
 4   primary_title    73856 non-null  object 
 5   original_title   73856 non-null  object 
 6   start_year       73856 non-null  int64  
 7   runtime_minutes  66236 non-null  float64
 8   genres           73052 non-null  object 
dtypes: float64(2), int64(2), object(5)
memory usage: 5.1+ MB
```

MOVIE BUDGET DATA

This dataset consist information about movies from 1915 to 2020 with information about their domestic, foreign and worldwide gross

In [6]:
```
# This shows the fist five rows in the dataset

movie_budget.head()
```

Out[6]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,279
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
2	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
3	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747

In [7]:

```
# Fixing the date formate amd viewing the five summary statistics of the colu

movie_budget['Release_Date'] =  pd.to_datetime(movie_budget['release_date'])
movie_budget['Release_Date'].describe()
```

```
<ipython-input-7-80c3651d3905>:4: FutureWarning: Treating datetime data as
categorical rather than numeric in `.describe` is deprecated and will be re
moved in a future version of pandas. Specify `datetime_is_numeric=True` to
silence this warning and adopt the future behavior now.
  movie_budget['Release_Date'].describe()
```

Out[7]:

```
count                    5782
unique                   2418
top       2014-12-31 00:00:00
freq                       24
first     1915-02-08 00:00:00
last      2020-12-31 00:00:00
Name: Release_Date, dtype: object
```

In [8]:

```
# Viewing the five summary statistics of the domestic gross column

movie_budget['domestic_gross'].describe()
```

Out[8]:

```
count      5782
unique     5164
top          $0
freq        548
Name: domestic_gross, dtype: object
```

MOVIE DATA

The movie dataset contains information from movie basic and movie rating table with records about different genres from year 2012 to 2019

In [9]:

```
# This shows the fist five rows in the dataset

movie_data.head()
```

Out[9]:

	movie_id	averagerating	numvotes	movie_id	primary_title	original_title	start_year	run
0	tt10356526	8.3	31	tt10356526	Laiye Je Yaarian	Laiye Je Yaarian	2019	
1	tt10384606	8.9	559	tt10384606	Borderless	Borderless	2019	
2	tt1042974	6.4	20	tt1042974	Just Inès	Just Inès	2010	
3	tt1043726	4.2	50352	tt1043726	The Legend of Hercules	The Legend of Hercules	2014	
4	tt1060240	6.5	21	tt1060240	Até Onde?	Até Onde?	2011	

In [10]:
```python
# Counting the number of unique values in the start year column

movie_data['start_year'].value_counts()
```

Out[10]:
```
2016    8721
2017    8713
2015    8494
2014    8371
2013    7990
2012    7680
2018    7526
2011    7389
2010    6792
2019    2180
Name: start_year, dtype: int64
```

In [11]:
```python
# To extract the five summary stat

movie_data['runtime_minutes'].describe()
```

Out[11]:
```
count    66236.000000
mean        94.654040
std        208.574111
min          3.000000
25%         81.000000
50%         91.000000
75%        104.000000
max      51420.000000
Name: runtime_minutes, dtype: float64
```

In [12]:
```python
# To find how many unique values are in the genre column

movie_data['genres'].describe()
```

Out[12]:
```
count      73052
unique       923
top        Drama
freq       11612
Name: genres, dtype: object
```

DATA PREPARATION

Both data sets are cleaned for maximun optimization. This include dropping unnecessary rows, adding new columns and editting values within columns

In [13]:

```
# Changing the columns from string to integer

movie_budget["domestic_gross"]= movie_budget["domestic_gross"].str.replace("$
movie_budget["domestic_gross"]= movie_budget["domestic_gross"].str.replace(",
movie_budget["domestic_gross"]= movie_budget["domestic_gross"].astype('int64'

movie_budget["worldwide_gross"]= movie_budget["worldwide_gross"].str.replace(
movie_budget["worldwide_gross"]= movie_budget["worldwide_gross"].str.replace(
movie_budget["worldwide_gross"]= movie_budget["worldwide_gross"].astype('int6

movie_budget["production_budget"]= movie_budget["production_budget"].str.repl
movie_budget["production_budget"]= movie_budget["production_budget"].str.repl
movie_budget["production_budget"]= movie_budget["production_budget"].astype('
```

In [14]:

```
# Dropping unneccessary records

movie = movie_data.dropna()

# Splitting grouped genres
movie['genres'] = movie["genres"].apply(lambda word: word.split(",")[0]);
```

```
<ipython-input-14-ac56bf4162cf>:6: 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-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  movie['genres'] = movie["genres"].apply(lambda word: word.split(",")[0]);
```

In [15]:

```
# Adding new columns

movie_budget["foreign_gross"] = movie_budget["worldwide_gross"]  - movie_budg

movie_budget["domestic_profit"] = movie_budget["domestic_gross"] - movie_budg
movie_budget["foreign_profit"] = movie_budget["foreign_gross"] - movie_budget
movie_budget["worldwide_profit"] = movie_budget["worldwide_gross"] - movie_bu

movie_budget['domestic_profit_percentages'] = movie_budget['production_budget
movie_budget['foreign_profit_percentages'] = movie_budget['production_budget'
movie_budget['worldwide_profit_percentages'] = movie_budget['production_budge
```

DATA ANALYSIS

In [16]:

```
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
```

1. **AVERAGE PROFIT PERCENTAGE FOR ALL MOVIES**

Movies with a small production budget have higher average percentage mean

In [17]:
```
# Defining production budget sizes and separating categories

def m_categories(budget):
    if budget <= 5000000:
        return 'small'
    elif budget > 5000000 and budget < 17000000:
        return 'medium'
    else:
        return 'large'

movie_budget['budget_size'] = movie_budget['production_budget'].map(m_categor
```

In [18]:
```

dc = movie_budget.groupby('budget_size')

small_films = dc.get_group('small')
med_films = dc.get_group('medium')
lg_films = dc.get_group('large')
```

In [19]:
```
def m_classify(profit):
    if profit <= 0:
        return 'loser'
    elif profit > 0 and profit < .5:
        return 'scraper'
    else:
        return 'winner'

movie_budget['winners & losers'] = movie_budget['worldwide_profit_percentages
```

In [20]:
```
ec = movie_budget.groupby('winners & losers')

wins = ec.get_group('winner')
loses = ec.get_group('loser')
scrapes = ec.get_group('scraper')
```

In [21]:
```
fc = wins.groupby('budget_size')

gc = loses.groupby('budget_size')
```

In [22]:

```python
# Upon getting the means using the .describe()

data = {'film_sizes': ['small', 'medium', 'large'],
        'percentage_means': [1.769723, 0.120114, 0.700102]}
film_comp = pd.DataFrame(data)
film_comp
```

Out[22]:

	film_sizes	percentage_means
0	small	1.769723
1	medium	0.120114
2	large	0.700102

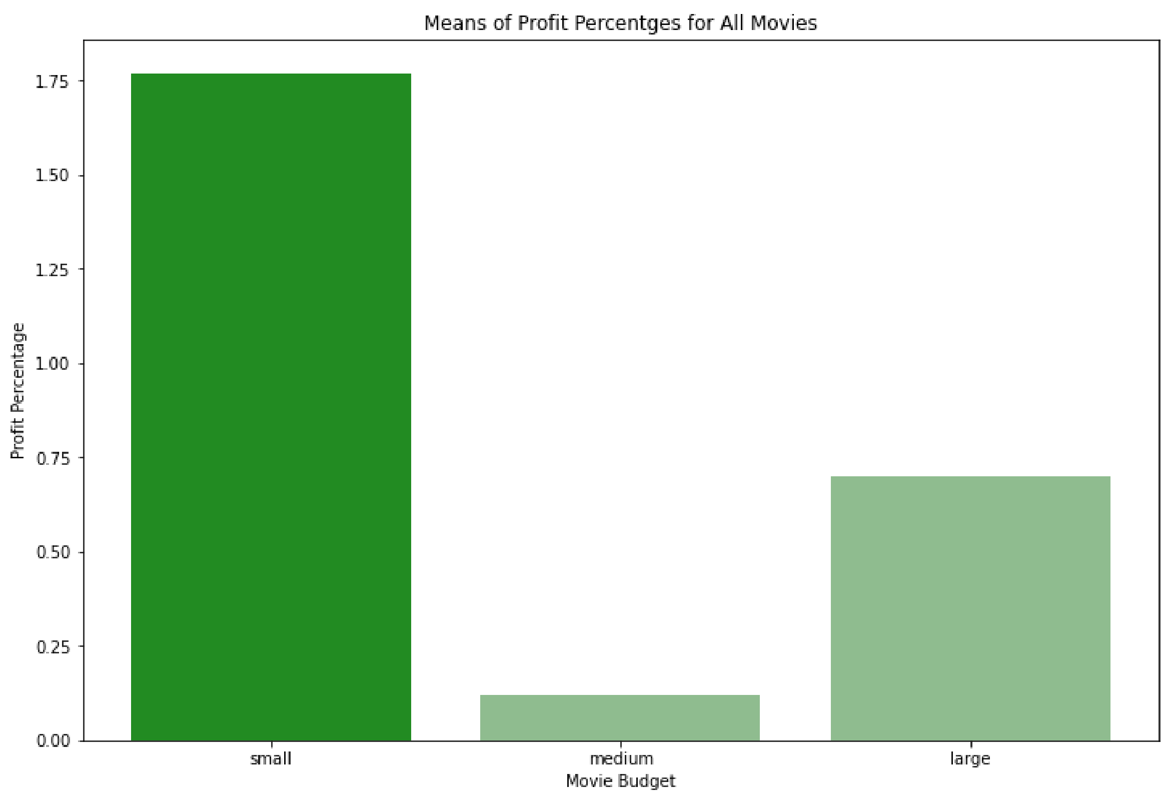
In [23]:

```python
# Creating a bar plot

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

ax.bar(film_comp['film_sizes'], film_comp['percentage_means'], color=['forest

ax.set_xlabel("Movie Budget")
ax.set_ylabel("Profit Percentage")
ax.set_title("Means of Profit Percentges for All Movies")
plt.show()
```



2. **DOMESTIC AND WORLDWIDE GROSS COMPARISON**

Domestic gross has a greater impact on the overall worldwide gross than foreign gross. When comparing both grosses sets Domestic gross had a sharper rate of change.

1 billion of domestic gross impacts just about 2.4 Billion of worldwide gross. However, 1 billion of foreign gross impacts only 1.5 Billion of worldwide gross.

In [24]:

```
# Creating a plot that compares domestic gross with worldwide gross

import plotly.express as px

fig = px.scatter(movie_budget, x="domestic_gross", y= "worldwide_gross" ,colc
                 color_continuous_scale= 'emrld',trendline="ols",
                 labels=dict(domestic_gross="Domestic Gross ($Billions)",
                             worldwide_gross="World Wide Gross ($Billions)"))

fig.update_xaxes(range=[0, 2000000000])
fig.update_yaxes(range=[0, 3000000000])
fig.update_layout(title_text='Domestic Gross vs. Worldwide Gross', title_x =
fig.update_layout(showlegend=False)
fig.show()
```

This distribution describes the difference in how domestic gross impacts the worldwide gross overall. From the graph below we see that there is a strong positive correlation between domestic gross and worldwide gross.

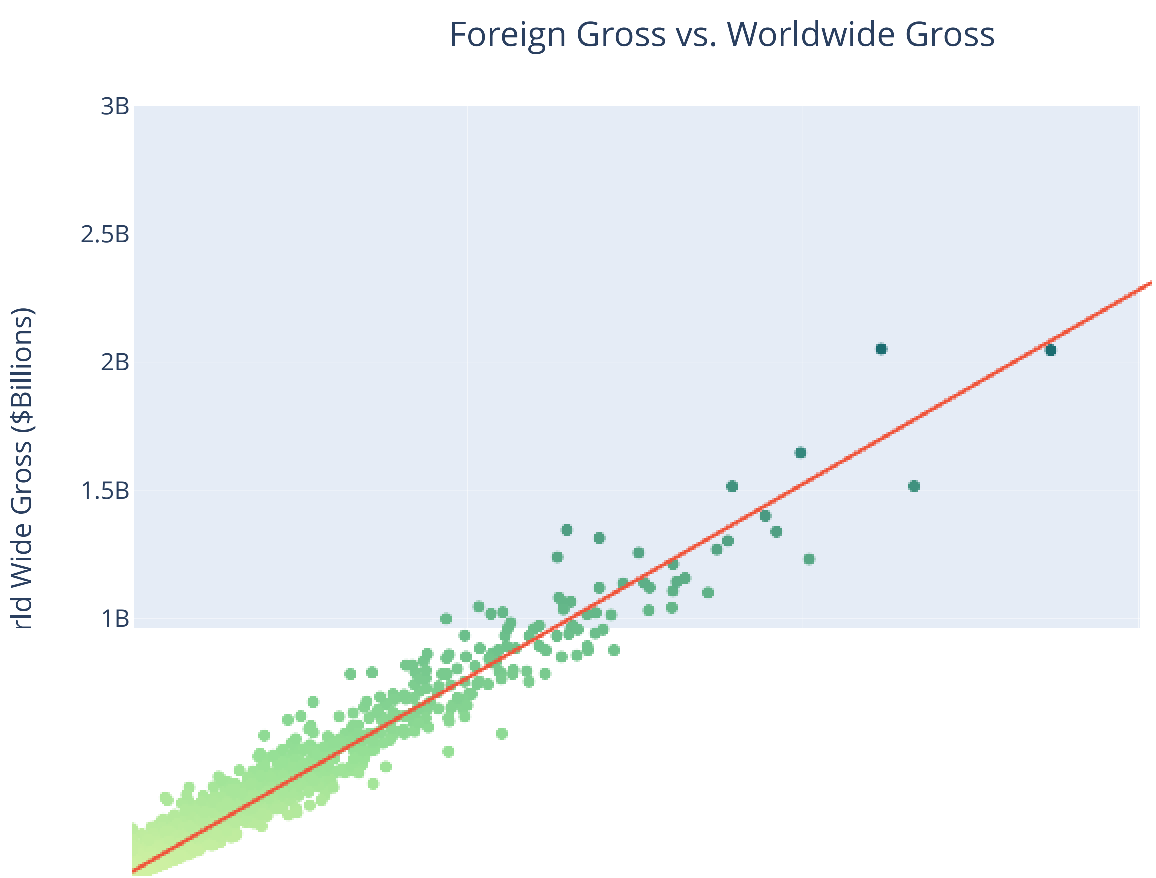
In addition, there is a strong rate of change as the domestiXc gross increases so do the worldwide gross.

In [25]:

```
# Creating a plot that compares foreign gross with worldwide gross

fig = px.scatter(movie_budget, x="foreign_gross", y= "worldwide_gross" ,color
                 labels=dict(foreign_gross="Foreign Gross ($Billions)",
                             worldwide_gross="World Wide Gross ($Billions)"))

fig.update_xaxes(range=[0, 2000000000])
fig.update_yaxes(range=[0, 3000000000])
fig.update_layout(title_text='Foreign Gross vs. Worldwide Gross', title_x = .
fig.show()
```



This distribution describes the difference in how foreign gross impacts the worldwide gross overall. From the graph below we see that there is a strong positive correlation between foreign gross and worldwide gross.

In addition, there is a steady rate of change as the foreign gross increases so do the worldwide gross.

3. **GENRES AND RATING**

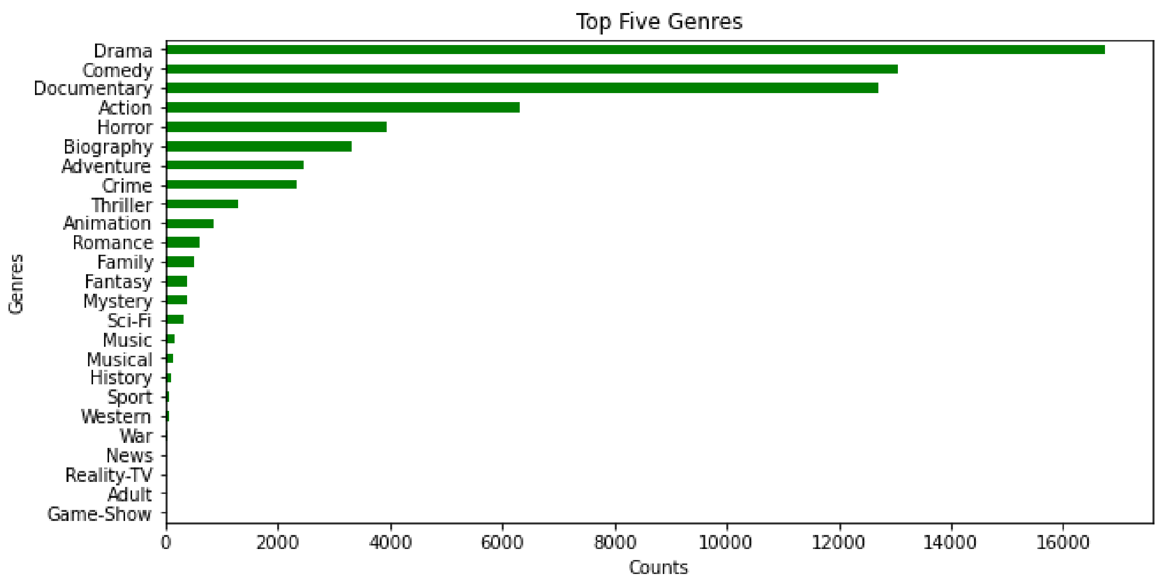
Most movies produced are Drama, Comedy, Documentary, Action and Horror genres. Documentary, Drama and Comedy genres are among the most rated genres of the top five genres.

In [26]:
```
# Create a plot showing the number of genres

fig, ax = plt.subplots(figsize=(10,5))

ax = movie["genres"].value_counts(ascending= True).plot(kind = 'barh',
    color = ["green"] , grid=False)

plt.grid(False)
ax.set_title("Top Five Genres")
ax.set(xlabel="Counts" , ylabel="Genres");
```



Now that we know the top five genres, let's look at these top 5 genres produced and deduce which of the genres has the highest rating

In [27]:
```
top_5 = ["Drama", "Comedy", "Documentary", "Action","Horror"]
mv_new = movie[movie.genres.isin(top_5)]
```

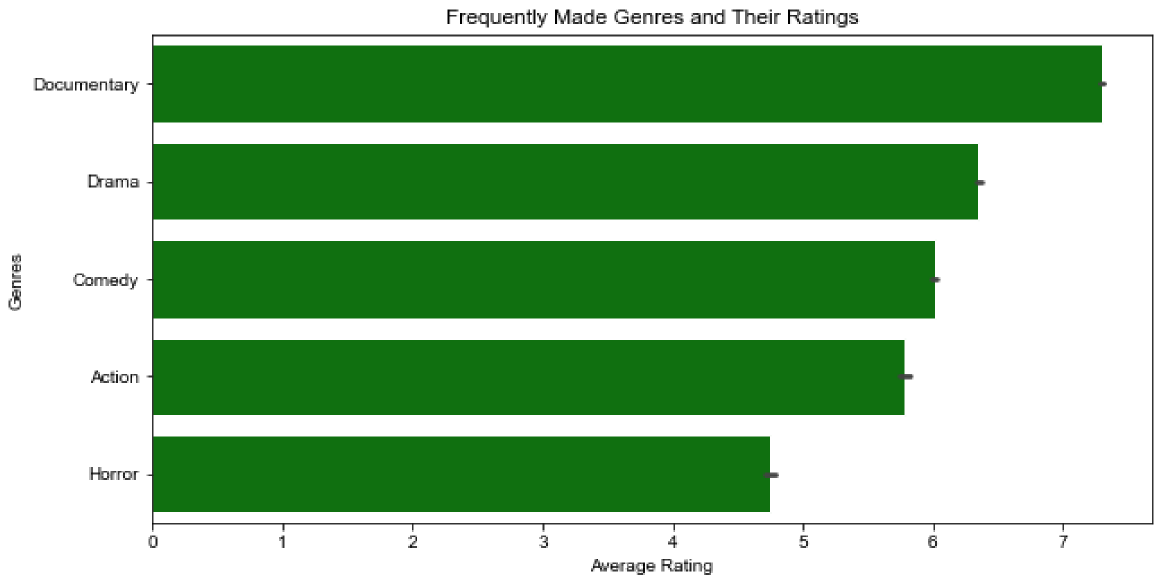
In [28]:

```
fig, ax = plt.subplots(figsize=(10,5))

sns.set(style="whitegrid", color_codes=True)
order= ["Documentary", "Drama", "Comedy", "Action", 'Horror']

sns.barplot(data=mv_new, x="averagerating", y="genres",
              order = order, color = "green")

plt.grid(False)
ax.set(xlabel="Average Rating", ylabel="Genres")
ax.set_title("Frequently Made Genres and Their Ratings");
```



4. **MOVIE RUNTIME**

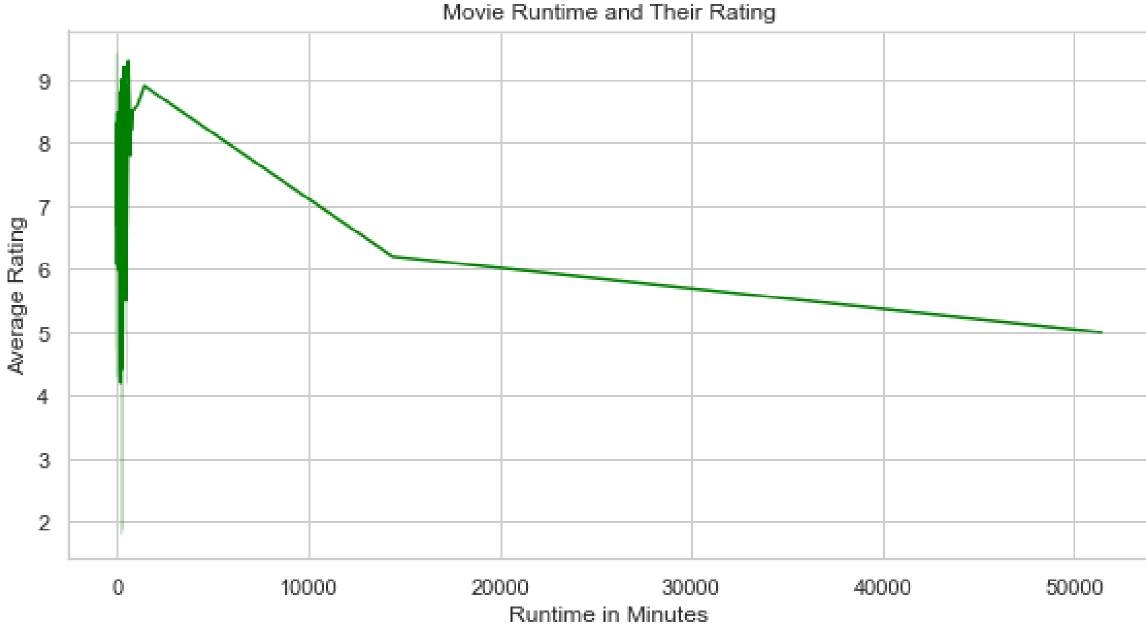
Short movies (movies with rutime of 56 minutes and lower) have higher rating than longer movies

In [29]:

```
# Create a plot

fig, ax = plt.subplots(figsize=(10,5))
g = sns.lineplot(data=movie, color = "green", x='runtime_minutes', y='average

ax.set_title("Movie Runtime and Their Rating")
ax.set(xlabel="Runtime in Minutes", ylabel="Average Rating");
```



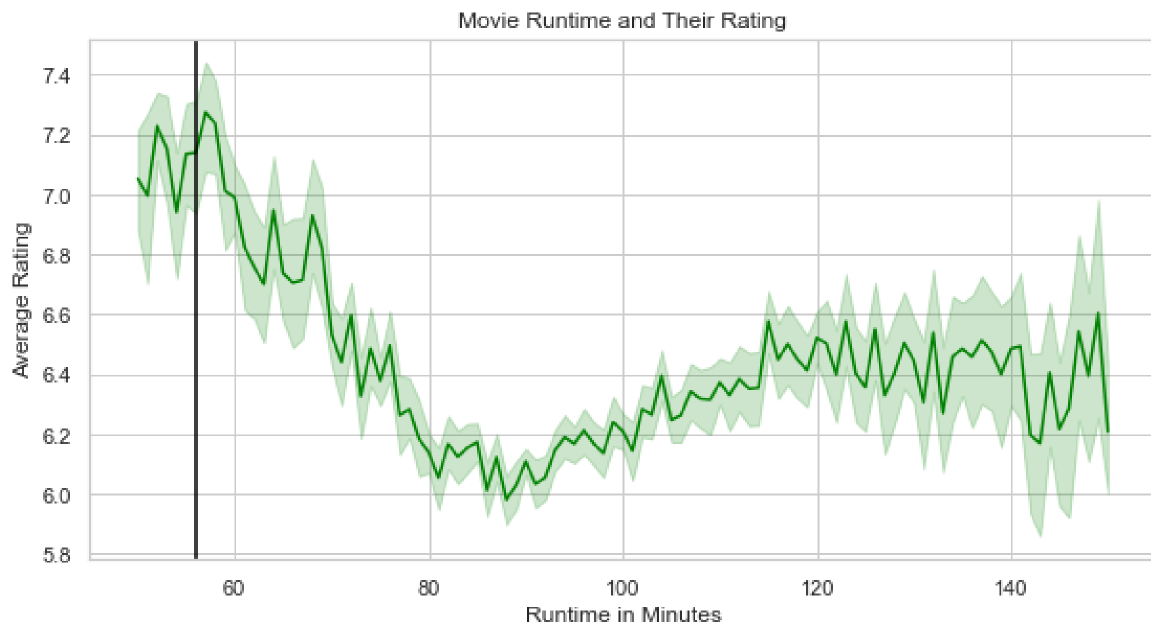
In [30]:

```
# Defining area of interest

interest = movie[(movie["runtime_minutes"] >=50.0) & (movie["runtime_minutes"

# Plotting area of interest

fig, ax = plt.subplots(figsize=(10,5))
g = sns.lineplot(data=interest, color = "green", x=interest['runtime_minutes'
g.axvline(x = 56, color = "black")
ax.set_title("Movie Runtime and Their Rating")
ax.set(xlabel="Runtime in Minutes", ylabel="Average Rating");
```



Here, it seems movies with runtime at 56 minutes and lower have higher ratings than movies with longer runtime.

This is clearly a bold statement to make so lets test the alternative hypothesis to see if movies with 56 minutes runtime actually differ in rating than the rest of the population.

Ho: Movies that have runtime less than 56 minutes do not have higher average rating

Ha: Movies that have runtime less than 56 minutes have higher average rating

In [31]:

```python
# Movie dataframe equals population data

movie.describe()
```

Out[31]:

	averagerating	numvotes	start_year	runtime_minutes
count	65720.000000	6.572000e+04	65720.000000	65720.000000
mean	6.320902	3.954674e+03	2014.258065	94.732273
std	1.458878	3.208823e+04	2.600143	209.377017
min	1.000000	5.000000e+00	2010.000000	3.000000
25%	5.500000	1.600000e+01	2012.000000	81.000000
50%	6.500000	6.200000e+01	2014.000000	91.000000
75%	7.300000	3.520000e+02	2016.000000	104.000000
max	10.000000	1.841066e+06	2019.000000	51420.000000

In [32]:

```python
# Creating the sample from the population

sample= movie[(movie["runtime_minutes"] >=0) & (movie["runtime_minutes"] <=56

# Stat summary of sample

sample.describe()
```

Out[32]:

	averagerating	numvotes	start_year	runtime_minutes
count	3036.000000	3036.000000	3036.000000	3036.000000
mean	7.144730	58.434124	2013.805007	47.829051
std	1.427123	546.470661	2.514968	9.537272
min	1.000000	5.000000	2010.000000	3.000000
25%	6.500000	7.000000	2012.000000	46.000000
50%	7.300000	12.000000	2014.000000	51.000000
75%	8.200000	26.000000	2016.000000	53.000000
max	10.000000	25596.000000	2019.000000	56.000000

Where average rating = population mean of 6.32

```
Ho: MUo <= 6.32
Ha: MUo > 6.32
```

This is a one-tailed Z test because we know the sample size is more than 30, and we know the population standard deviation.

In [33]:

```python
import numpy as np
import scipy.stats as stats

x_bar = 7.14 #Sample mean
MU = 6.32 #Population mean
n = 3036 #Sample size
sigma = 1.46 #Population SD
alpha = 0.05

# Calculating the z-statistic and the p-value

z = (x_bar - MU)/(sigma/np.sqrt(n))
p_value = stats.norm.sf(z)
p_value
```

Out[33]:

```
1.4147365585113178e-210
```

The p_value is less than alpha of 0.05, this means we reject the null hypothesis. The movies with runtime less tham 56 minutes have significantly higher rating.

CONCLUSION

Upon completion of the following analysis:

1. Analysing the amount of money made per dollar spent on making movies
2. Analysing the relationship between where a movie is launched with how much gross income it provides
3. Analysing the genres that are most frequently made in the market and the few most rated by the audience
4. The effect of average viewer rating on the length of movie (runtime)

We conculde with the list of recommendations below to influence the outcome of Computing Vision original video content creation endeavour.

- Invest in movies that cost lower than 5 million because these movies return a higher return per dollar spent in production budget. This means for every dollar spent producing a short movie, you should expect a higher pecentage return compared to the return for a medium or large sized budget movie.
- Launch movies domestically, before expanding to foreign markets because the worldwide gross correlates more strongly with domestic gross than foreign gross.
- Focus on documentary, drama, and comedy genres because they are the frequently made genres with the most ratings. The logic behind this is that frequently made genres are genres the audience wants to see (thus produced by other studios). Computing Vision can level up above competition by not anly producing the top genres but also producing genres within the top category that has sthe highest average viewer rating.
- Make short movies (defined as movies that are about 56 minutes) because they are rated favorably than longer movies. Viewers tend to rate short movies higher than longer movies and this was confirmed via hypothesis testing.

NEXT STEPS

Further analysis could yield insights on

- How much money should be spent per genre, i.e; cause the most amount of money
- What kind of genres have the highest screentime
- How muchtime it takes to produce movies based on genres