Project 2: Investigate the TMDB movies dataset:

Table of Contents

Introduction

In this project, we will explore the data gathered from Kaggle (the cleaned version provided by Udacity). The data has information about more than 10000 movies, the number of votes, budget, and revenue of the movies, director and cast of the movies,..etc. The goal of this project is to get an idea of what factors can play a role in making movies more profitable and likable from the viewers. During my EDA I tried to answer the question below :

> - Which type of movies is the most-watched from year to year?
> - What types of movies are associated with a higher rating?
> - What kind of properties are associated with movies that have high revenue?
> - Which actors had made the most number of appearances

First, let's start by importing all the needed packages and of course the 'magic word' so that our visualizations are plotted.

In [92]:
```
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
sns.set_style('darkgrid')
```

Data Wrangling

General Properties

In [95]:
```
# Loading up the data and printing out the first 5 lines.
df_movies= pd.read_csv(r"C:\Users\hp\Desktop\Akram Folder\DATABASE\UDACITY\Project 2 - Analyze Experiment Results\tmdb-movies.csv")
df_movies.head()
```

Out[95]:

	id	imdb_id	popularity	budget	revenue	original_title	cast	
0	135397	tt0369610	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	
1	76341	tt1392190	28.419936	150000000	378436354	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic...	
2	262500	tt2908446	13.112507	110000000	295238201	Insurgent	Shailene Woodley|Theo James|Kate Winslet|Ansel...	http://ww
3	140607	tt2488496	11.173104	200000000	2068178225	Star Wars: The Force Awakens	Harrison Ford|Mark Hamill|Carrie Fisher|Adam D...	htt
4	168259	tt2820852	9.335014	190000000	1506249360	Furious 7	Vin Diesel|Paul Walker|Jason Statham|Michelle ...	

5 rows × 21 columns

In [93]:
```
df_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10866 entries, 0 to 10865
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   id                    10866 non-null  int64  
 1   imdb_id               10856 non-null  object 
 2   popularity            10866 non-null  float64
 3   budget                10866 non-null  int64  
 4   revenue               10866 non-null  int64  
 5   original_title        10866 non-null  object 
 6   cast                  10790 non-null  object 
 7   homepage              2936 non-null   object 
 8   director              10822 non-null  object 
 9   tagline               8042 non-null   object 
 10  keywords              9373 non-null   object 
 11  overview              10862 non-null  object 
 12  runtime               10866 non-null  int64  
 13  genres                10843 non-null  object 
 14  production_companies  9836 non-null   object 
 15  release_date          10866 non-null  object 
 16  vote_count            10866 non-null  int64  
 17  vote_average          10866 non-null  float64
 18  release_year          10866 non-null  int64  
 19  budget_adj            10866 non-null  float64
 20  revenue_adj           10866 non-null  float64
dtypes: float64(4), int64(6), object(11)
memory usage: 1.7+ MB
```

Data Cleaning

> We Have 10866 rows and 21 columns in this dataset, with multiple missing data in many columns, we will start our wrangling process by dropping unnecessary columns for our EDA, then, removing any duplicated and null data, and by the end, we convert data type for some of the columns in order to make the processing phase much easier.

1.Deleting unecessary columns

In [97]:
```
#droping all the colums not needed for my EDA
#Release_date column is not necessary since we have the release year in differ
ent column
Columns_to_Drop = ['imdb_id', 'homepage', 'tagline', 'overview', 'production_c
ompanies','director', 'keywords', 'release_date']
df_movies.drop(Columns_to_Drop, axis=1, inplace=True)
```

In [98]:
```
#cheking the results
df_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10866 entries, 0 to 10865
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              10866 non-null  int64  
 1   popularity      10866 non-null  float64
 2   budget          10866 non-null  int64  
 3   revenue         10866 non-null  int64  
 4   original_title  10866 non-null  object 
 5   cast            10790 non-null  object 
 6   runtime         10866 non-null  int64  
 7   genres          10843 non-null  object 
 8   vote_count      10866 non-null  int64  
 9   vote_average    10866 non-null  float64
 10  release_year    10866 non-null  int64  
 11  budget_adj      10866 non-null  float64
 12  revenue_adj     10866 non-null  float64
dtypes: float64(4), int64(6), object(3)
memory usage: 1.1+ MB
```

2. Removing duplicated data

In [68]:
```
#checking for any duplicated data
df_movies.duplicated().sum()
```

Out[68]: 1

In [92]:
```
#checking the results after droping the duplicated row
df_movies.drop_duplicates(inplace=True)
df_movies.duplicated().sum()
```

Out[92]: 0

Note: It came to my attention to check if there are any duplicated movies, and I found 294 duplicates that for better use of my data I will get rid of them.

In [95]:
```
df_movies['original_title'].duplicated().sum()
```

Out[95]: 294

In [97]:
```
#Droping the duplicates in 'Original_Title' column
df_movies['original_title'].drop_duplicates(inplace= True)
```

In [98]:
```
df_movies['original_title'].duplicated().sum()
```

Out[98]: 0

3. Deleting Null data

In [99]:
```
# droping missing values rows
df_movies.dropna(inplace=True)
#Checking the results
df_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10768 entries, 0 to 10865
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              10768 non-null  int64  
 1   popularity      10768 non-null  float64
 2   budget          10768 non-null  int64  
 3   revenue         10768 non-null  int64  
 4   original_title  10768 non-null  object 
 5   cast            10768 non-null  object 
 6   runtime         10768 non-null  int64  
 7   genres          10768 non-null  object 
 8   vote_count      10768 non-null  int64  
 9   vote_average    10768 non-null  float64
 10  release_year    10768 non-null  int64  
 11  budget_adj      10768 non-null  float64
 12  revenue_adj     10768 non-null  float64
dtypes: float64(4), int64(6), object(3)
memory usage: 1.2+ MB
```

4. Changing datatype:

> release_year and runtime columns contain fixed values, so changing the datatype to 'String' will make the processing of the data much easier.

In [101]:
```
#converting the datatype of release_year and runtime columns
df_movies['release_year']=df_movies['release_year'].astype(str)
df_movies['runtime']=df_movies['runtime'].astype(str)
#checking the results
df_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10768 entries, 0 to 10865
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              10768 non-null  int64  
 1   popularity      10768 non-null  float64
 2   budget          10768 non-null  int64  
 3   revenue         10768 non-null  int64  
 4   original_title  10768 non-null  object 
 5   cast            10768 non-null  object 
 6   runtime         10768 non-null  object 
 7   genres          10768 non-null  object 
 8   vote_count      10768 non-null  int64  
 9   vote_average    10768 non-null  float64
 10  release_year    10768 non-null  object 
 11  budget_adj      10768 non-null  float64
 12  revenue_adj     10768 non-null  float64
dtypes: float64(4), int64(4), object(5)
memory usage: 1.2+ MB
```

So After cleaning the dataset and keeping only what we will need for our analysis, we opted to keep a copy of the cleaned data in a separate file that we will use later for our EDA.

In [108]:
```
#saving a copy of the cleaned data
df_movies.to_excel("TMDB Clean DATA.xlsx",sheet_name="Clean_data", index=False
, index_label=True)
```

Exploratory Data Analysis

In this section we will try to answer our questions by processing our cleaned data and using matplotlib and seaborn libraries to visualize the results.

Research Question 1 (which genre of movies is popular from year to year ?)

In [2]:
```
# Loading up the saved cleaned data
df_tmdb1 = pd.read_excel(r"C:\Users\hp\Desktop\Akram Folder\DATABASE\UDACITY\P
roject 2 - Analyze Experiment Results\TMDB_Clean_DATA.xlsx")
```

In [3]:
```
df_tmdb1.head()
```

Out[3]:

	id	popularity	budget	revenue	original_title	cast	runtime	
0	135397	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	124	Action|Adv
1	76341	28.419936	150000000	378436354	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic...	120	Action|Adv
2	262500	13.112507	110000000	295238201	Insurgent	Shailene Woodley|Theo James|Kate Winslet|Ansel...	119	Adv
3	140607	11.173104	200000000	2068178225	Star Wars: The Force Awakens	Harrison Ford|Mark Hamill|Carrie Fisher|Adam D...	136	Action|Adv F
4	168259	9.335014	190000000	1506249360	Furious 7	Vin Diesel|Paul Walker|Jason Statham|Michelle ...	137	Actior

The column 'genres' seems to have multiple data inside every row separated by '|' , so in order to make the use of these data much easier, we will covert every cell in 'genres' column to a list.

> **Note** : we will do the same for 'cast' column later on.

In [5]:
```
#changing the data type of the column 'genres' to a list
df_tmdb1['genres'] = df_tmdb1.genres.apply(lambda x: x[:].split('|'))
```

In [6]:
```
#checking the results
df_tmdb1.genres[0]
```

Out[6]:
```
['Action', 'Adventure', 'Science Fiction', 'Thriller']
```

In [7]:
```
#genres_list will contain all the types of movies mentioned in the dataset
genres_list = []
for genre in df_tmdb1['genres']:
    genres_list.extend(genre)
print(genres_list)
```

```
['Action', 'Adventure', 'Science Fiction', 'Thriller', 'Action', 'Adventure',
'Science Fiction', 'Thriller', 'Adventure', 'Science Fiction', 'Thriller', 'A
ction', 'Adventure', 'Science Fiction', 'Fantasy', 'Action', 'Crime', 'Thrill
er', 'Western', 'Drama', 'Adventure', 'Thriller', 'Science Fiction', 'Actio
n', 'Thriller', 'Adventure', 'Drama', 'Adventure', 'Science Fiction', 'Famil
y', 'Animation', 'Adventure', 'Comedy', 'Comedy', 'Animation', 'Family', 'Act
ion', 'Adventure', 'Crime', 'Science Fiction', 'Fantasy', 'Action', 'Adventur
e', 'Drama', 'Science Fiction', 'Action', 'Comedy', 'Science Fiction', 'Actio
n', 'Adventure', 'Science Fiction', 'Crime', 'Drama', 'Mystery', 'Western',
'Crime', 'Action', 'Thriller', 'Science Fiction', 'Action', 'Adventure', 'Rom
ance', 'Fantasy', 'Family', 'Drama', 'War', 'Adventure', 'Science Fiction',
'Action', 'Family', 'Science Fiction', 'Adventure', 'Mystery', 'Action', 'Dra
ma', 'Action', 'Drama', 'Thriller', 'Drama', 'Romance', 'Comedy', 'Drama', 'A
ction', 'Comedy', 'Crime', 'Comedy', 'Action', 'Adventure', 'Drama', 'Thrille
r', 'History', 'Action', 'Science Fiction', 'Thriller', 'Mystery', 'Drama',
'Crime', 'Action', 'Science Fiction', 'Comedy', 'Music', 'Thriller', 'Drama',
'Adventure', 'Horror', 'Comedy', 'Drama', 'Thriller', 'Crime', 'Drama', 'Myst
ery', 'Adventure', 'Animation', 'Comedy', 'Family', 'Fantasy', 'Action', 'Cri
me', 'Drama', 'Mystery', 'Thriller', 'Drama', 'Romance', 'Drama', 'Music', 'F
antasy', 'Action', 'Adventure', 'History', 'Drama', 'Comedy', 'Action', 'Adve
nture', 'Fantasy', 'Drama', 'Romance', 'Action', 'Adventure', 'Science Fictio
n', 'Fantasy', 'Comedy', 'Animation', 'Science Fiction', 'Family', 'Drama',
'Mystery', 'Romance', 'Thriller', 'Crime', 'Drama', 'Thriller', 'Comedy', 'Dr
ama', 'Romance', 'Science Fiction', 'Romance', 'Drama', 'Comedy', 'Adventur
e', 'Drama', 'Comedy', 'Drama', 'Action', 'Crime', 'Thriller', 'Drama', 'Scie
nce Fiction', 'Mystery', 'Thriller', 'Comedy', 'Adventure', 'Drama', 'Myster
y', 'Crime', 'Action', 'Thriller', 'Drama', 'Action', 'Crime', 'Drama', 'Myst
ery', 'Thriller', 'Action', 'Adventure', 'Science Fiction', 'Mystery', 'Horro
r', 'Action', 'Comedy', 'Crime', 'Romance', 'Comedy', 'Crime', 'Drama', 'Acti
on', 'Crime', 'Thriller', 'Thriller', 'Drama', 'Adventure', 'Action', 'Histor
y', 'Crime', 'Thriller', 'Action', 'Drama', 'Comedy', 'Drama', 'Thriller', 'W
ar', 'Crime', 'Thriller', 'Thriller', 'Adventure', 'Family', 'Fantasy', 'Acti
on', 'Adventure', 'Fantasy', 'Comedy', 'Drama', 'Adventure', 'Animation', 'Co
medy', 'Family', 'Drama', 'Comedy', 'Drama', 'Horror', 'Thriller', 'Romance',
'Drama', 'Animation', 'Comedy', 'Family', 'Family', 'Comedy', 'Adventure', 'D
rama', 'Thriller', 'Action', 'Crime', 'Drama', 'Adventure', 'Comedy', 'Horro
r', 'Thriller', 'Horror', 'Drama', 'Romance', 'Science Fiction', 'Crime', 'Th
riller', 'Thriller', 'Mystery', 'Comedy', 'Fantasy', 'Action', 'Adventure',
'Thriller', 'Science Fiction', 'Action', 'Adventure', 'Adventure', 'Animatio
n', 'Fantasy', 'Adventure', 'Animation', 'Comedy', 'Family', 'Drama', 'Romanc
e', 'Comedy', 'Horror', 'Action', 'Adventure', 'Comedy', 'Family', 'Adventur
e', 'Animation', 'Family', 'Action', 'Drama', 'Science Fiction', 'Thriller',
'Thriller', 'Action', 'Comedy', 'Comedy', 'Comedy', 'Horror', 'Horror', 'Thri
ller', 'Crime', 'Drama', 'Crime', 'Action', 'Thriller', 'Horror', 'Comedy',
'Fantasy', 'Drama', 'Mystery', 'Thriller', 'Action', 'Thriller', 'Crime', 'Dr
ama', 'Comedy', 'Comedy', 'Action', 'Drama', 'Action', 'Fantasy', 'Adventur
e', 'Comedy', 'Science Fiction', 'Thriller', 'Comedy', 'Science Fiction', 'Th
riller', 'Action', 'Crime', 'Mystery', 'Thriller', 'Fantasy', 'Horror', 'Dram
a', 'Thriller', 'Action', 'Comedy', 'Drama', 'Music', 'Horror', 'Thriller',
'Romance', 'Thriller', 'Western', 'Drama', 'Crime', 'Drama', 'Mystery', 'Come
dy', 'Family', 'Animation', 'Crime', 'Drama', 'Mystery', 'Adventure', 'Dram
a', 'Family', 'Family', 'Animation', 'Comedy', 'Adventure', 'Drama', 'Comed
y', 'Drama', 'Action', 'Drama', 'Crime', 'Horror', 'Thriller', 'Action', 'Cri
me', 'Comedy', 'Drama', 'Horror', 'Thriller', 'Drama', 'Science Fiction', 'Th
riller', 'Action', 'Adventure', 'Fantasy', 'Comedy', 'Drama', 'Drama', 'Scien
ce Fiction', 'Thriller', 'Adventure', 'Drama', 'Family', 'Animation', 'Comed
y', 'Drama', 'Romance', 'Horror', 'Western', 'Adventure', 'Drama', 'Horror',
```

In [8]:
```
pd.Series(genres_list).value_counts()
```

Out[8]:
```
Drama              4752
Comedy             3785
Thriller           2905
Action             2381
Romance            1712
Horror             1637
Adventure          1469
Crime              1354
Science Fiction    1227
Family             1219
Fantasy             911
Mystery             809
Animation           669
Documentary         478
Music               405
History             331
War                 268
Foreign             187
TV Movie            167
Western             165
dtype: int64
```

> **Drama** has been mentioned as type of movie in 4752 movies, therefore, we can say that the **Drama Movies** are the most-watched type in the period between 1960 and 2015.

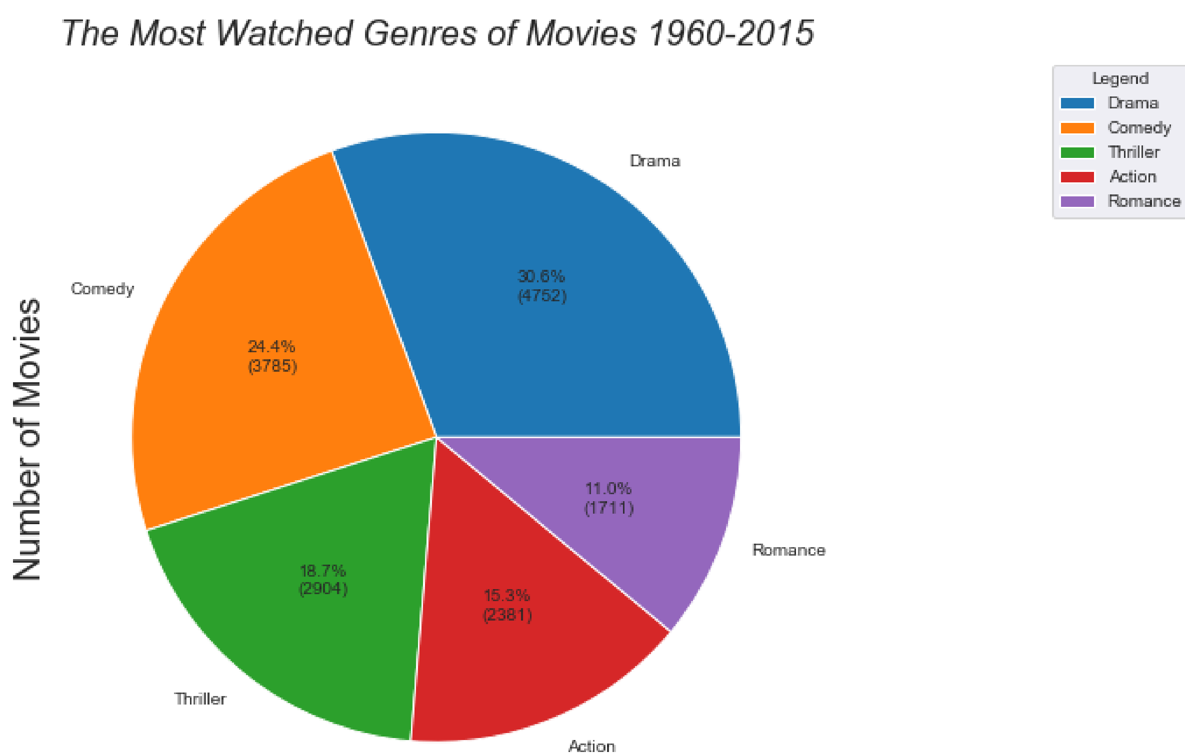
Below is a pie chart that visualizes the data obtained.

In [103]:
```
#Plotting the 5 most watched types of movies
import matplotlib.pyplot as plt

def func(pct, allvals):        #Fuction copied from Matplotlib documentaion 'h
ttps://matplotlib.org/3.1.1/gallery/pie_and_polar_charts/pie_and_donut_labels.
html'
    absolute = int(pct/100.*np.sum(allvals))
    return "{:.1f}%\n({:d})".format(pct, absolute)

data=pd.Series(genres_list).value_counts()[:5]
data.plot.pie(figsize=(8,8), autopct=lambda pct: func(pct, data))
plt.title("The Most Watched Genres of Movies 1960-2015", fontsize=20, style='i
talic')
plt.ylabel("Number of Movies", fontsize=20)
plt.legend(loc='best', title='Legend', bbox_to_anchor=(1, 0, 0.5, 1))
```

Out[103]: <matplotlib.legend.Legend at 0x2ec1592f708>

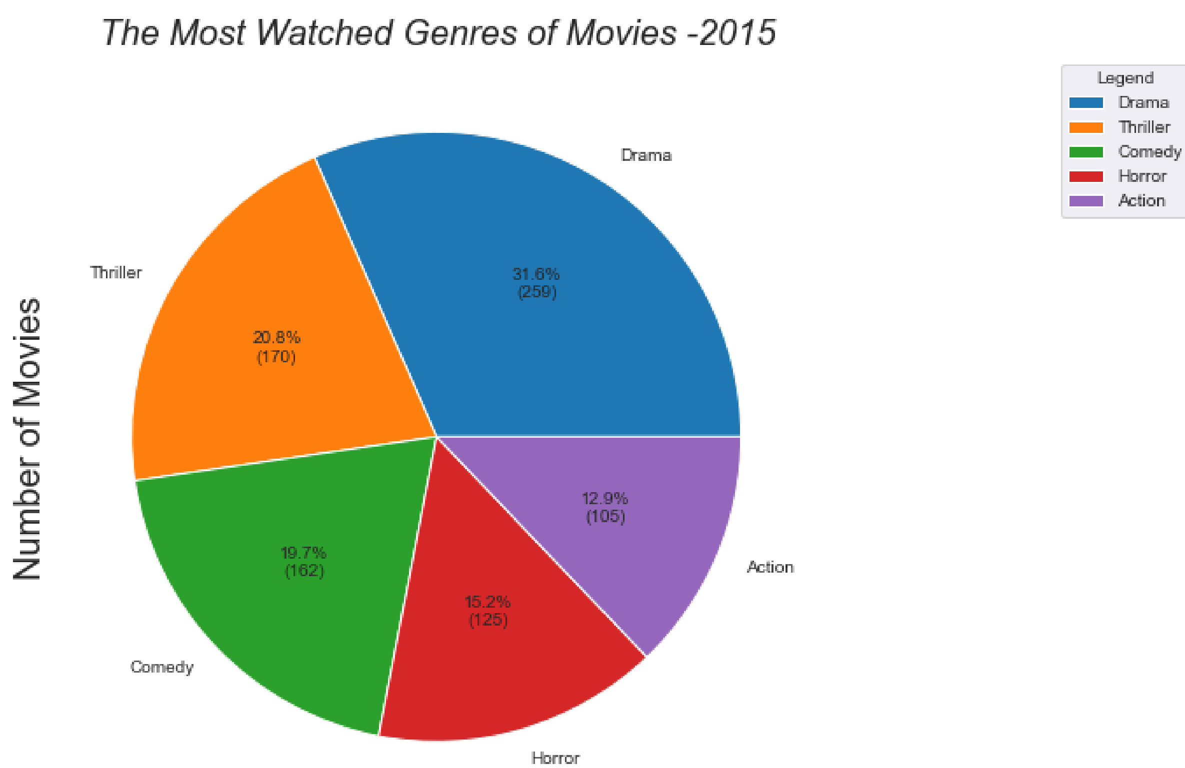


In order to check the genre of movies that had a lot of viewers per year, we created a function that takes 'year' as an argument and return a list of the popular genre of movies.

In [10]:
```
# Function that return the genre of the movie
def genres_year (year):
    df_test= df_tmdb1[df_tmdb1.release_year == str(year)]
    list1 = []
    for genre in df_test['genres']:
        list1.extend(genre)
    return pd.Series(list1).value_counts()
```

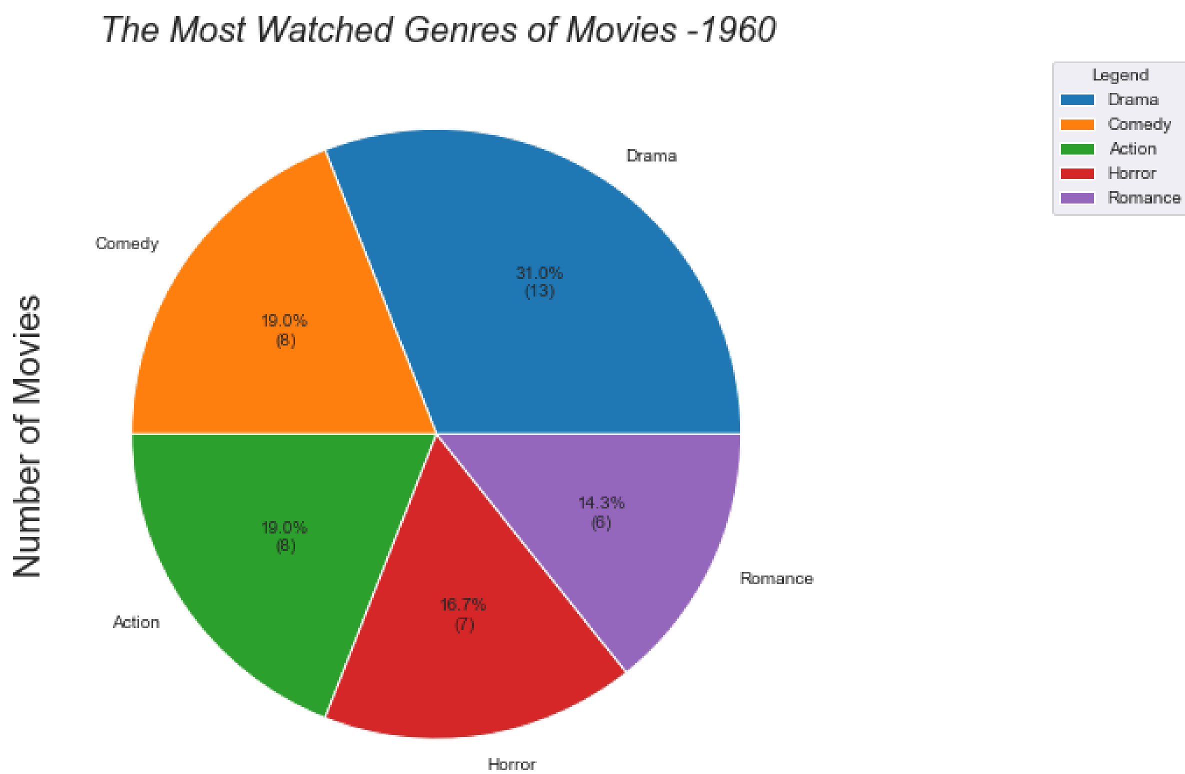
In [104]:
```
# The Most Watched Genres of Movies -2015
data_2015 = genres_year(2015)[:5]
data_2015.plot.pie(figsize=(8,8), autopct=lambda pct: func(pct, data_2015))
plt.title("The Most Watched Genres of Movies -2015", fontsize=20, style='itali
c')
plt.ylabel("Number of Movies", fontsize=20)
plt.legend(loc='best', title='Legend', bbox_to_anchor=(1, 0, 0.5, 1))
```

Out[104]: <matplotlib.legend.Legend at 0x2ec15823ac8>



In [112]:
```
# The Most Watched Genres of Movies -1960
data_1960 = genres_year(1960)[:5]
data_1960.plot.pie(figsize=(8,8), autopct=lambda pct: func(pct, data_1960))
plt.title("The Most Watched Genres of Movies -1960", fontsize=20, style='itali
c')
plt.ylabel("Number of Movies", fontsize=20)
plt.legend(loc='best', title='Legend', bbox_to_anchor=(1, 0, 0.5, 1))
```

Out[112]:
```
<matplotlib.legend.Legend at 0x2ec13b6df48>
```



As we can see (even comparing just two years is not a factor to make a judgment), **Drama** had been in the top list of the most-watched genres over the period between 1960 and 2015.

Note : In order to check the other years, you just need to change the 'year' in the function part (**data_1960 = genres_year(1960)[:5]**) of the cell above, and it will plot a pie chart, visualizing the top 5 genres of that year.

Research Question 2 (what type of Movies is associated with the highest rating ?)

To Answer this question, we will load up the cleaned data again and save it in a different data frame, so whatever processing will be made on this data won't affect the use of it to answer the rest of the question.

In [76]:
```
#we will start by loading up the data again and save it in 'df_votes' then, de
let every column not needed for this EDA.
df_votes = pd.read_excel(r"C:\Users\hp\Desktop\Akram Folder\DATABASE\UDACITY\P
roject 2 - Analyze Experiment Results\TMDB_Clean_DATA.xlsx")
Columns_to_Drop = ['id', 'popularity', 'budget', 'revenue','cast', 'runtime',
'release_year']
#checking the result
df_votes.drop(Columns_to_Drop, axis=1, inplace=True)
df_votes.head(1)
```

Out[76]:

	original_title	genres	vote_count	vote_average	budget_adj	revenue_adj
0	Jurassic World	Action|Adventure|Science Fiction|Thriller	5562	6.5	1.379999e+08	1.392446e+09

After keeping what we need in our dataframe, we separate the genres of the movies again, and then we will group the mean of 'vote_average' data by genres.

In [77]:
```
#separating the data in the column 'genres'
new_votes = pd.DataFrame(df_votes.genres.str.split('|').tolist(),index=df_vote
s.vote_average).stack().reset_index([0, 'vote_average'])
new_votes.columns=['vote_average', 'genres']
new_votes
```

Out[77]:

	vote_average	genres
0	6.5	Action
1	6.5	Adventure
2	6.5	Science Fiction
3	6.5	Thriller
4	7.1	Action
...	...	...
26826	6.5	Mystery
26827	6.5	Comedy
26828	5.4	Action
26829	5.4	Comedy
26830	1.5	Horror

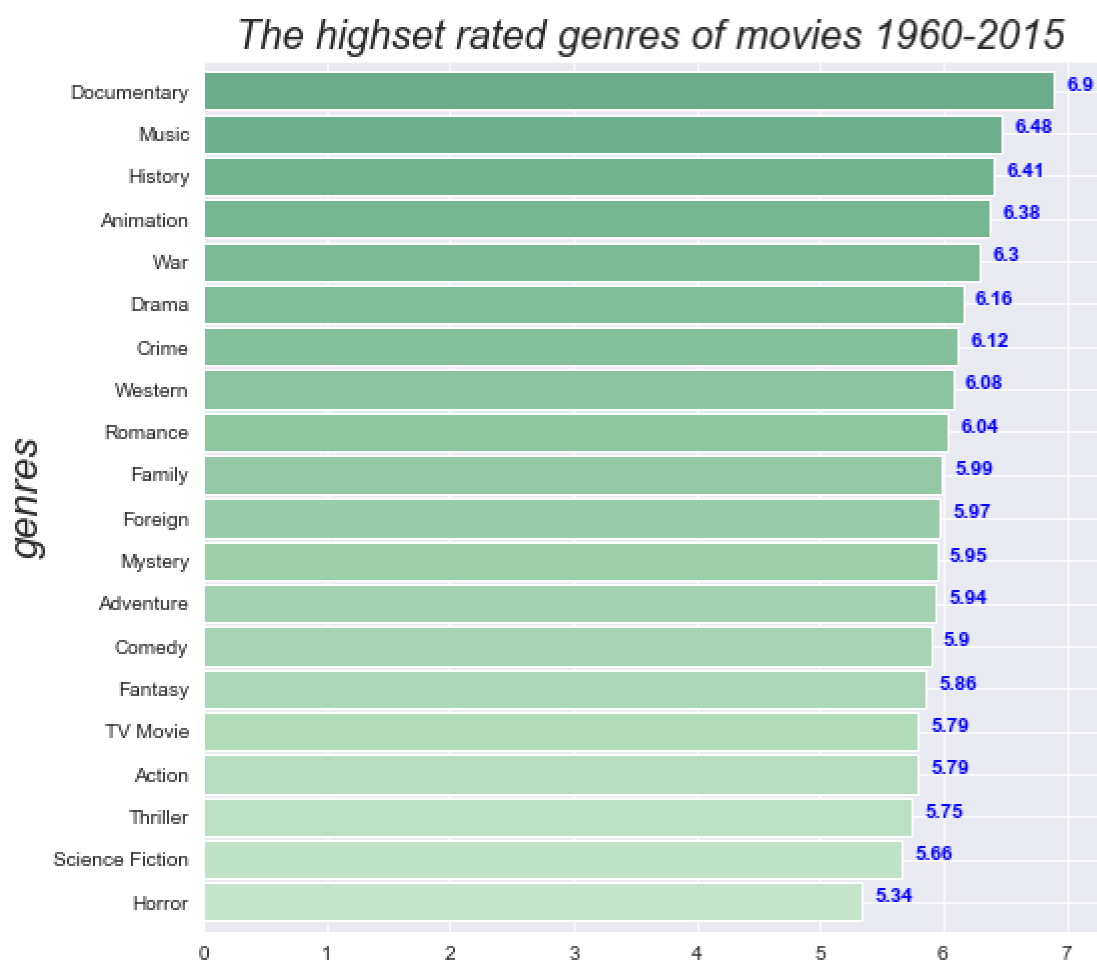
26831 rows × 2 columns

In [78]:
```
#grouping the mean of 'votes_average' data by the column 'genres'
data_votes = new_votes.groupby('genres')['vote_average'].mean().round(2).sort_
values(ascending=True)
data_votes
```

Out[78]:
```
genres
Horror             5.34
Science Fiction    5.66
Thriller           5.75
Action             5.79
TV Movie           5.79
Fantasy            5.86
Comedy             5.90
Adventure          5.94
Mystery            5.95
Foreign            5.97
Family             5.99
Romance            6.04
Western            6.08
Crime              6.12
Drama              6.16
War                6.30
Animation          6.38
History            6.41
Music              6.48
Documentary        6.90
Name: vote_average, dtype: float64
```

In [79]:
```
#visualizing the results using barh plot
data_votes.plot.barh(width=0.9,color=sns.color_palette("ch:2.5,-.2,dark=.3", 4
0), figsize=(8,8))
for ind, value in enumerate(new_votes.groupby('genres')['vote_average'].mean()
.round(2).sort_values(ascending=True).values):
    plt.text(value + 0.1 , ind , str(value), color='blue', fontweight='bold')
plt.title("The highset rated genres of movies 1960-2015", fontsize=20, style=
'italic')
plt.ylabel("genres", fontsize=20, style='italic')
```

Out[79]: Text(0, 0.5, 'genres')



- Using the average votes of the movies as a factor to rank the highest rated genre seems a little unfair since, after a quick inspection of the dataset, we noticed that some of the movies have a low number of votes and high score as average votes, meanwhile there'are other movies with a high number of votes but the average score of votes is lower than the previous category, so we don't think it is efficient to take the vote_average in consideration but calculating the **weighted rating** as explained below, will get us a better comparison of the genres, and the ranking will be more accurate.

How do you calculate the rank of movies and TV shows on the Top Rated Movies and Top Rated TV Show lists?

The following formula is used to calculate the Top Rated 250 titles. This formula provides a true 'Bayesian estimate', which takes into account the number of votes each title has received, minimum votes required to be on the list, and the mean vote for all titles:

> > **weighted rating (WR) = (v ÷ (v+m)) × R + (m ÷ (v+m)) × C**

Where:

- R = average for the movie (mean) = (rating)
- v = number of votes for the movie = (votes)
- m = minimum votes required to be listed in the Top Rated list
- C = the mean vote across the whole report

Link: https://help.imdb.com/article/imdb/track-movies-tv/ratings-faq/G67Y87TFYYP6TWAV# (https://help.imdb.com/article/imdb/track-movies-tv/ratings-faq/G67Y87TFYYP6TWAV#)

In [113]:
```
# Calculatin the mean of 'vote_average'
C= df_tmdb1['vote_average'].mean()
#In order to be in the chart the movie need to have votes more than 95% of the list
m= df_tmdb1['vote_count'].quantile(0.95)
C,m
```

Out[113]:
```
(5.967548992291278, 1034.3999999999978)
```

> - So the mean of average votes is almost 6 on a scale of 10.
> - we opted to chose 95% as the threshold for the value of minimum votes required to be listed in the Top Rated list 'm', that means, only movies with a number of votes higher than the 95% of the list will be in the list. The purpose is to minimize the number of movies on the list since we are looking for the top genres based on the higher votes.

In [81]:
```
#Storing the filterd data in new data frame
wr_votes = df_votes[df_votes.vote_count >= m]
wr_votes.shape
```

Out[81]:
```
(539, 6)
```

In [82]:
```
#weighted rating function
def weighted_rating_IMDB(df, m=m, C=C):
    v = df['vote_count']
    R = df['vote_average']
    return round((v/(v+m) * R) + (m/(m+v) * C), 2)
```

In [83]:
```
# We will add a new column to the new dataframe, so we can compare the votes t
o the genres
wr_votes['weighted_rating'] = wr_votes.apply(weighted_rating_IMDB, axis= 1)
```

```
C:\Users\hp\anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithC
opyWarning:
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/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

We can Also check which movie is associated with the highest rating, and below is the result. As we can see, **The Shawshank Redemption** has the highest rating of the dataset.

In [114]:
```
#The top 10 of the highest-rated movies between 1960 and 2015
wr_votes.sort_values('weighted_rating', ascending=False).head(10)
```

Out[114]:

	original_title	genres	vote_count	vote_average	budget_adj	revenue_
4127	The Shawshank Redemption	Drama|Crime	5754	8.4	3.677779e+07	4.169346e·
2838	The Dark Knight	Drama|Action|Crime|Thriller	8432	8.1	1.873655e+08	1.014733e·
7192	The Godfather	Drama|Crime	3970	8.3	3.128737e+07	1.277914e·
2373	Fight Club	Drama	5923	8.1	8.247033e+07	1.320229e·
4126	Pulp Fiction	Thriller|Crime	5343	8.1	1.176889e+07	3.147131e·
4128	Forrest Gump	Comedy|Drama|Romance	4856	8.1	8.091114e+07	9.973333e·
620	Interstellar	Adventure|Drama|Science Fiction	6498	8.0	1.519800e+08	5.726906e·
1891	Inception	Action|Thriller|Science Fiction|Mystery|Adventure	9767	7.9	1.600000e+08	8.255000e·
621	Guardians of the Galaxy	Action|Science Fiction|Adventure	5612	7.9	1.565855e+08	7.122911e·
4886	The Lord of the Rings: The Return of the King	Adventure|Fantasy|Action	5636	7.9	1.114231e+08	1.326278e·

In [85]:
```
# Spliting the data in 'genres' column in order to compare it to the 'weighted
_rating'
new_votes2 = pd.DataFrame(wr_votes.genres.str.split('|').tolist(),index=wr_vot
es.weighted_rating).stack().reset_index([0, 'weighted_rating'])
new_votes2.columns=['weighted_rating', 'genres']
new_votes2
```

Out[85]:

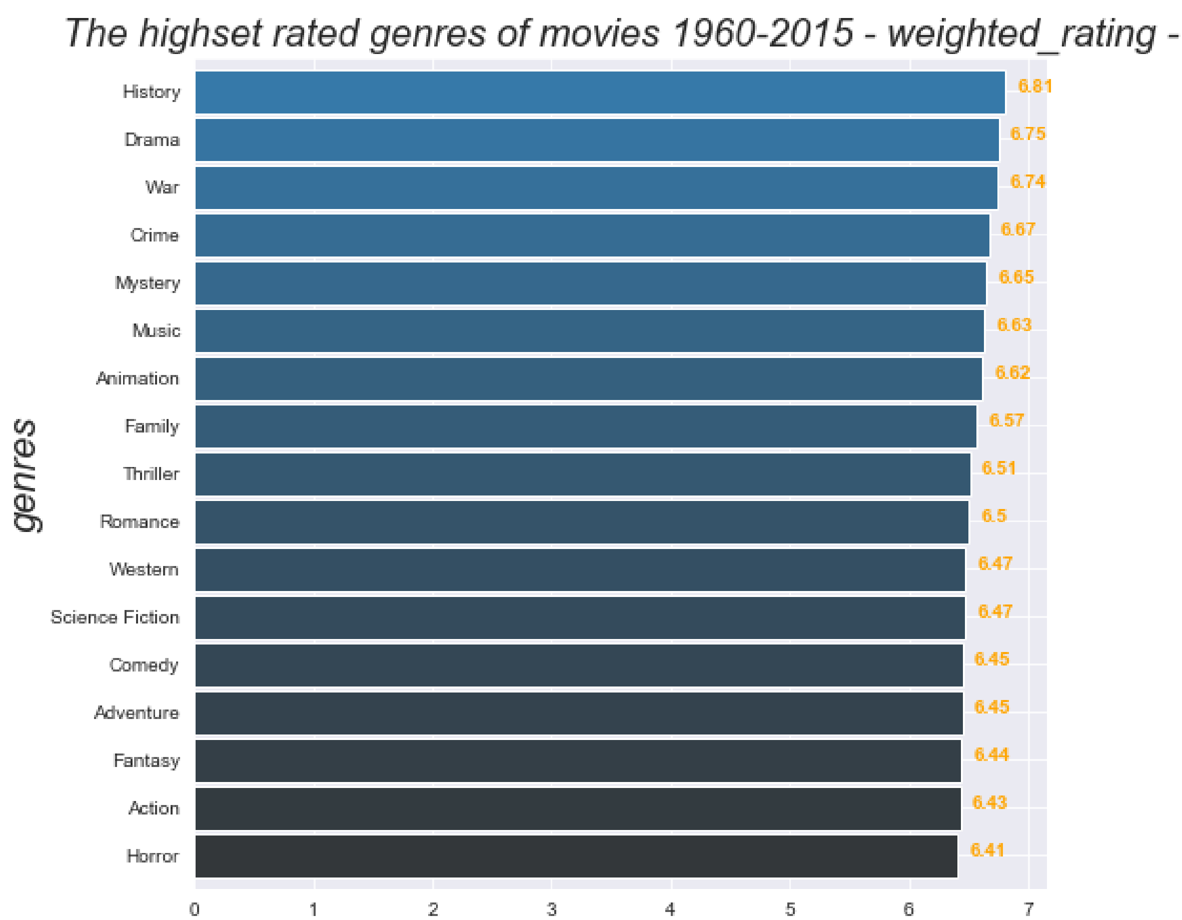
	weighted_rating	genres
0	6.42	Action
1	6.42	Adventure
2	6.42	Science Fiction
3	6.42	Thriller
4	6.94	Action
...	...	...
1599	6.29	Action
1600	6.29	Romance
1601	6.61	Drama
1602	6.61	Horror
1603	6.61	Thriller

1604 rows × 2 columns

In [115]:
```
#grouping the mean of 'weighted_rating' data by the column 'genres'
data_votes2 = new_votes2.groupby('genres')['weighted_rating'].mean().round(2).
sort_values(ascending=True)
```

In [116]:
```
#visualization of the data using the barh plot
data_votes2.plot.barh(width=0.9,color=sns.color_palette("Blues_d", 40), figsiz
e=(8,8))
for ind, value in enumerate(new_votes2.groupby('genres')['weighted_rating'].me
an().round(2).sort_values(ascending=True).values):
    plt.text(value +0.1, ind , str(value), color='orange', fontweight='bold')
plt.title("The highset rated genres of movies 1960-2015 - weighted_rating -",
fontsize=20, style='italic')
plt.ylabel("genres", fontsize=20, style='italic')
```

Out[116]: Text(0, 0.5, 'genres')



- We personally thought that **Drama** will the highest rated genre but the **DATA** had a different say. So **History** is the highest rated genre by a score of 6.81/10, followed by **Drama** by a score of 6.75/10 and then in the third-place **War** scoring 6.74/10.
- The result obtained may be explained by the large period(1960-2015 (55 years)) we use to do our analysis, we're assuming that back the days the audience had a different taste of movies than the actual one.

Research Question (What properties are associated with movies that have the highest revenue ?)

First of all, let's start by adding a column that shows the profit of each movie based on the budget and revenue of the associated movie in terms of 2010 dollars, accounting for inflation over time. we will use the last data frame created **'wr_votes'** to compare all the properties of the top 10 movies.

In [89]:
```
# getting the Profit by subtracting the budget from the revenue
wr_votes['profit'] = wr_votes['budget_adj'] - wr_votes['revenue_adj']
#Checking the result (sorted in a descending way)
wr_votes.sort_values('profit', ascending=False).head(10)
```

```
C:\Users\hp\anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithC
opyWarning:
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/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

Out[89]:

	original_title	genres	vote_count	vote_average	budget_adj	revenu
5444	The Lone Ranger	Action|Adventure|Western	1607	6.0	2.386885e+08	8.357833
5406	RED 2	Action|Comedy|Crime|Thriller	1109	6.3	7.862680e+07	0.000000
5171	Gattaca	Thriller|Science Fiction|Mystery|Romance	1117	7.3	4.890457e+07	1.702528
647	The Interview	Action|Comedy	1657	6.1	4.052801e+07	1.041310
5393	The World's End	Comedy|Action|Science Fiction	1143	6.6	1.872067e+07	0.000000
5974	The Internship	Comedy	1174	6.1	5.428993e+07	4.118547
1916	Scott Pilgrim vs. the World	Action|Adventure|Comedy	1258	7.2	6.000000e+07	4.766456
4353	Dredd	Action|Science Fiction	1350	6.5	4.748721e+07	3.897536
6495	Children of Men	Drama|Action|Thriller|Science Fiction	1211	7.3	8.220686e+07	7.567330
2600	Donnie Darko	Fantasy|Drama|Mystery	1777	7.5	7.388929e+06	1.564633

From the table above we can conclude that the votes are not a factor in deciding if the movie will make a good profit or not, but in the other side, we can see that the genre has an effect on the profit, **Action** Movies showed that they're profitable, followed by the **Comedy** movies.

Research Question 3 (which Actor has the most number of appearances ?)

In [17]:
```
#changing the data type of the column 'cast' to a list
df_tmdb1['cast'] = df_tmdb1.cast.apply(lambda x: x[:].split('|'))
```

In [18]:
```
#checking the results
df_tmdb1.cast[0]
```

Out[18]:
```
['Chris Pratt',
 'Bryce Dallas Howard',
 'Irrfan Khan',
 "Vincent D'Onofrio",
 'Nick Robinson']
```

In [19]:
```
#cast_list will gather every actor's name mentioned in the dataset
cast_list = []
for actor in df_tmdb1['cast']:
    cast_list.extend(actor)
print(cast_list)
```

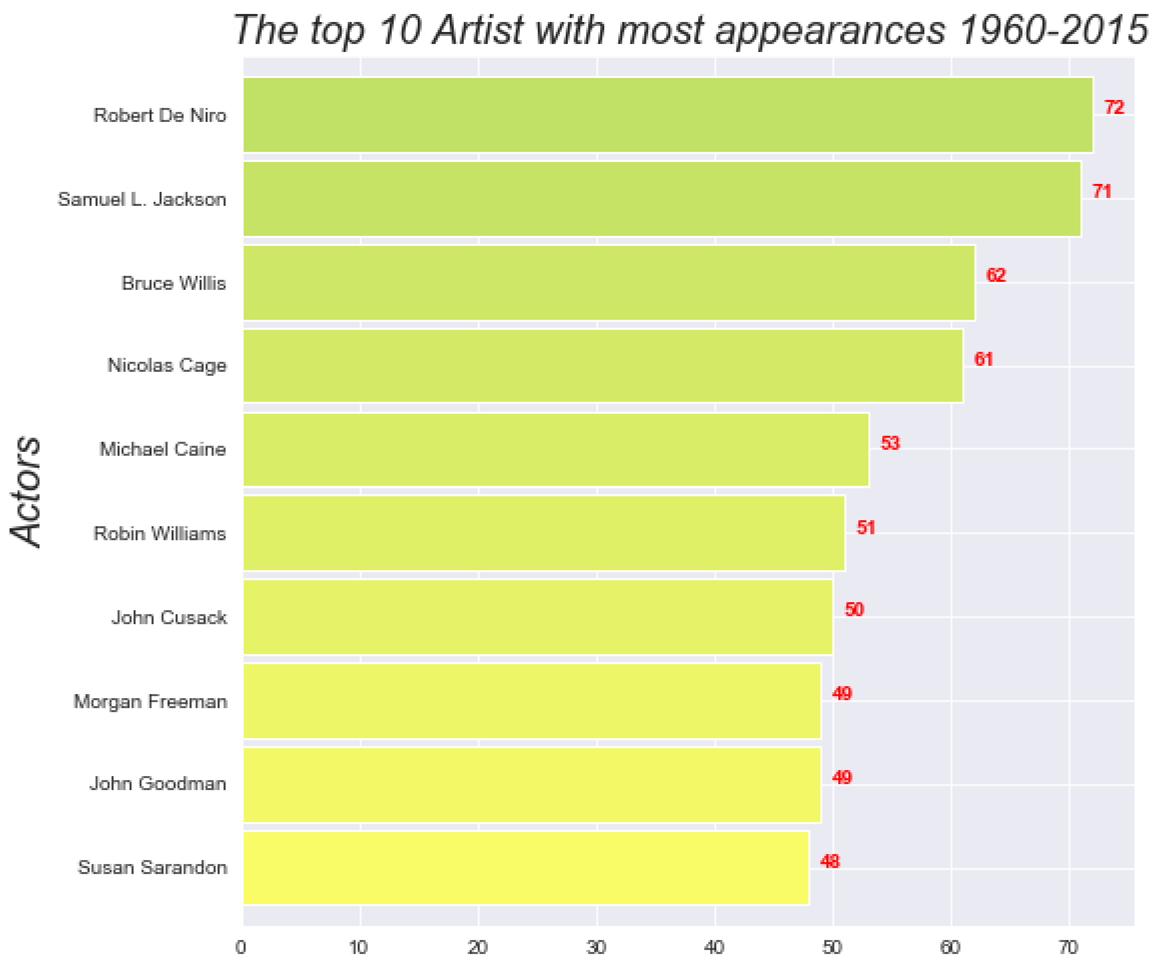
```
['Chris Pratt', 'Bryce Dallas Howard', 'Irrfan Khan', "Vincent D'Onofrio", 'N
ick Robinson', 'Tom Hardy', 'Charlize Theron', 'Hugh Keays-Byrne', 'Nicholas
Hoult', 'Josh Helman', 'Shailene Woodley', 'Theo James', 'Kate Winslet', 'Ans
el Elgort', 'Miles Teller', 'Harrison Ford', 'Mark Hamill', 'Carrie Fisher',
'Adam Driver', 'Daisy Ridley', 'Vin Diesel', 'Paul Walker', 'Jason Statham',
'Michelle Rodriguez', 'Dwayne Johnson', 'Leonardo DiCaprio', 'Tom Hardy', 'Wi
ll Poulter', 'Domhnall Gleeson', 'Paul Anderson', 'Arnold Schwarzenegger', 'J
ason Clarke', 'Emilia Clarke', 'Jai Courtney', 'J.K. Simmons', 'Matt Damon',
'Jessica Chastain', 'Kristen Wiig', 'Jeff Daniels', 'Michael PeÃ±a', 'Sandra
Bullock', 'Jon Hamm', 'Michael Keaton', 'Allison Janney', 'Steve Coogan', 'Am
y Poehler', 'Phyllis Smith', 'Richard Kind', 'Bill Hader', 'Lewis Black', 'Da
niel Craig', 'Christoph Waltz', 'LÃ©a Seydoux', 'Ralph Fiennes', 'Monica Bell
ucci', 'Mila Kunis', 'Channing Tatum', 'Sean Bean', 'Eddie Redmayne', 'Dougla
s Booth', 'Domhnall Gleeson', 'Alicia Vikander', 'Oscar Isaac', 'Sonoya Mizun
o', 'Corey Johnson', 'Adam Sandler', 'Michelle Monaghan', 'Peter Dinklage',
'Josh Gad', 'Kevin James', 'Robert Downey Jr.', 'Chris Hemsworth', 'Mark Ruff
alo', 'Chris Evans', 'Scarlett Johansson', 'Samuel L. Jackson', 'Kurt Russel
l', 'Jennifer Jason Leigh', 'Walton Goggins', 'DemiÃ¡n Bichir', 'Liam Neeso
n', 'Forest Whitaker', 'Maggie Grace', 'Famke Janssen', 'Dougray Scott', 'Pau
l Rudd', 'Michael Douglas', 'Evangeline Lilly', 'Corey Stoll', 'Bobby Cannava
le', 'Lily James', 'Cate Blanchett', 'Richard Madden', 'Helena Bonham Carte
r', 'Holliday Grainger', 'Jennifer Lawrence', 'Josh Hutcherson', 'Liam Hemswo
rth', 'Woody Harrelson', 'Elizabeth Banks', 'Britt Robertson', 'George Cloone
y', 'Raffey Cassidy', 'Thomas Robinson', 'Hugh Laurie', 'Jake Gyllenhaal', 'R
achel McAdams', 'Forest Whitaker', 'Oona Laurence', '50 Cent', 'Dwayne Johnso
n', 'Alexandra Daddario', 'Carla Gugino', 'Ioan Gruffudd', 'Archie Panjabi',
'Dakota Johnson', 'Jamie Dornan', 'Jennifer Ehle', 'Eloise Mumford', 'Victor
Rasuk', 'Christian Bale', 'Steve Carell', 'Ryan Gosling', 'Brad Pitt', 'Melis
sa Leo', 'Tom Cruise', 'Jeremy Renner', 'Simon Pegg', 'Rebecca Ferguson', 'Vi
ng Rhames', 'Mark Wahlberg', 'Seth MacFarlane', 'Amanda Seyfried', 'Jessica B
arth', 'Giovanni Ribisi', 'Taron Egerton', 'Colin Firth', 'Samuel L. Jackso
n', 'Michael Caine', 'Mark Strong', 'Mark Ruffalo', 'Michael Keaton', 'Rachel
McAdams', 'Liev Schreiber', 'John Slattery', "Dylan O'Brien", 'Kaya Scodelari
o', 'Thomas Brodie-Sangster', 'Giancarlo Esposito', 'Aidan Gillen', 'Ian McKe
llen', 'Milo Parker', 'Laura Linney', 'Hattie Morahan', 'Patrick Kennedy', 'S
harlto Copley', 'Dev Patel', 'Ninja', 'Yolandi Visser', 'Jose Pablo Cantill
o', 'Anna Kendrick', 'Rebel Wilson', 'Hailee Steinfeld', 'Brittany Snow', 'Sk
ylar Astin', 'Tom Hanks', 'Mark Rylance', 'Amy Ryan', 'Alan Alda', 'Sebastian
Koch', 'Jack Black', 'Dylan Minnette', 'Odeya Rush', 'Amy Ryan', 'Jillian Bel
l', 'Brie Larson', 'Jacob Tremblay', 'Joan Allen', 'Sean Bridgers', 'William
H. Macy', 'Abbie Cornish', 'Jeffrey Dean Morgan', 'Colin Farrell', 'Anthony H
opkins', 'Marley Shelton', 'Raymond Ochoa', 'Jack Bright', 'Jeffrey Wright',
'Frances McDormand', 'Maleah Nipay-Padilla', 'Liam Neeson', 'Ed Harris', 'Joe
l Kinnaman', 'Boyd Holbrook', 'Bruce McGill', 'Saoirse Ronan', 'Domhnall Glee
son', 'Emory Cohen', 'Emily Bett Rickards', "Eileen O'Higgins", "O'Shea Jacks
on Jr.", 'Corey Hawkins', 'Jason Mitchell', 'Neil Brown Jr.', 'Aldis Hodge',
'Vin Diesel', 'Rose Leslie', 'Michael Caine', 'Elijah Wood', 'Ã“lafur Darri
Ã“lafsson', 'Michael Fassbender', 'Kate Winslet', 'Seth Rogen', 'Katherine Wa
terston', 'Jeff Daniels', 'Henry Cavill', 'Armie Hammer', 'Alicia Vikander',
'Elizabeth Debicki', 'Luca Calvani', 'Blake Lively', 'Michiel Huisman', 'Harr
ison Ford', 'Ellen Burstyn', 'Kathy Baker', 'Sharlto Copley', 'Haley Bennet
t', 'Danila Kozlovskiy', 'Tim Roth', 'Andrei Dementiev', 'Jim Parsons', 'Riha
nna', 'Steve Martin', 'Jennifer Lopez', 'Matt Jones', 'Nat Wolff', 'Cara Dele
vingne', 'Halston Sage', 'Justice Smith', 'Austin Abrams', 'Jason Statham',
'Michael Angarano', 'Milo Ventimiglia', 'Dominik GarcÃ\xada-Lorido', 'Anne He
che', 'Colin Farrell', 'Rachel Weisz', 'LÃ©a Seydoux', 'John C. Reilly', 'Ben
Whishaw', 'Cate Blanchett', 'Rooney Mara', 'Kyle Chandler', 'Sarah Paulson',
```

In [117]:
```
# Actors with the highest number of appearances are as following:
pd.Series(cast_list).value_counts()
```

Out[117]:
```
Robert De Niro          72
Samuel L. Jackson       71
Bruce Willis            62
Nicolas Cage            61
Michael Caine           53
                        ..
Christopher Buchholz     1
Michael Coleman          1
Lois Maxwell             1
Knut Joner               1
Gethin Anthony           1
Length: 18983, dtype: int64
```

In [118]:
```
#Visualization of the data
data_cast=pd.Series(cast_list).value_counts()[:10].sort_values(ascending=True)
data_cast.plot.barh(width=0.9,color=sns.color_palette("summer_r", 40), figsize
=(8,8))
for ind, value in enumerate(pd.Series(cast_list).value_counts()[:10].sort_valu
es(ascending=True).values):
    plt.text(value + 1, ind , str(value), color='red', fontweight='bold')
plt.title("The top 10 Artist with most appearances 1960-2015", fontsize=20, st
yle='italic')
plt.ylabel("Actors", fontsize=20, style='italic')
```

Out[118]:
```
Text(0, 0.5, 'Actors')
```



No surprise that one of our favorite actors appears in many movies, **Robert De Niro** appeared in 72 movies in the period between 1960 and 2015, followed by another great actor **Samuel L. Jackson** that was in 71 movies during the same period.

Conclusions

> As summary of our analysis on the dataset collected between 1960 and 2015 :
>
> > - **Drama** has been the most watched type of movies
> > - **The Shawshank Redemption** is the highest rated movie
> > - **Robert De Niro** has the highest number of appearances in movies
> > - **Action** and **Comedy** tend to be the most profitable genre of movies
> > - **Votes** don't affect the profitability of the movie
>
> The finding mentioned abvove is basic to understand what factors can play major roles in either recommending a movie or anticipating if a movie will be profitable or not, in addition to other analytics that required statistical tests.

In [ ]: