In [4]:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

In [5]:

```
#Load the dataset
df = pd.read_csv("IMDB Top 250 Movies.csv")
df.head()
```

Out[5]:

	rank	name	year	rating	genre	certificate	run_time	tagline	
0	1	The Shawshank Redemption	1994	9.3	Drama	R	2h 22m	Fear can hold you prisoner. Hope can set you f...	2
1	2	The Godfather	1972	9.2	Crime,Drama	R	2h 55m	An offer you can't refuse.	
2	3	The Dark Knight	2008	9.0	Action,Crime,Drama	PG-13	2h 32m	Why So Serious?	18
3	4	The Godfather Part II	1974	9.0	Crime,Drama	R	3h 22m	All the power on earth can't change destiny.	1
4	5	12 Angry Men	1957	9.0	Crime,Drama	Approved	1h 36m	Life Is In Their Hands -- Death Is On Their Mi...	

Getting basic info about the data

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   rank         250 non-null    int64  
 1   name         250 non-null    object 
 2   year         250 non-null    int64  
 3   rating       250 non-null    float64
 4   genre        250 non-null    object 
 5   certificate  250 non-null    object 
 6   run_time     250 non-null    object 
 7   tagline      250 non-null    object 
 8   budget       250 non-null    object 
 9   box_office   250 non-null    object 
 10  casts        250 non-null    object 
 11  directors    250 non-null    object 
 12  writers      250 non-null    object 
dtypes: float64(1), int64(2), object(10)
memory usage: 25.5+ KB
```

In [7]:

```
df.describe()
```

Out[7]:

	rank	year	rating
count	250.000000	250.000000	250.000000
mean	125.500000	1986.360000	8.307200
std	72.312977	25.125356	0.229081
min	1.000000	1921.000000	8.000000
25%	63.250000	1966.250000	8.100000
50%	125.500000	1994.000000	8.200000
75%	187.750000	2006.000000	8.400000
max	250.000000	2022.000000	9.300000

Feature Engineering

Budget

In [8]:

```
set(df['budget'])
```

Out[8]:

```
{'$3300000',
 '$8240000',
 '1000000',
 '10000000',
 '100000000',
 '102000000',
 '103000000',
 '11000000',
 '11400000',
 '11500000',
 '115000000',
 '1200000',
 '12000000',
 '125000000',
 '1288000',
 '1300000',
 '13000000',
 '133000',
 '13500000',
 '1400000',
 '140000000',
 '14400000',
 '14600000',
 '150000',
 '1500000',
 '15000000',
 '150000000',
 '160000000',
 '16400000',
 '165000000',
 '170000000',
 '17500000',
 '175000000',
 '1752000',
 '1800000',
 '18000000',
 '180000000',
 '18500000',
 '185000000',
 '1900000',
 '19000000',
 '2000000',
 '20000000',
 '200000000',
 '2100000',
 '2200000',
 '22000000',
 '229575',
 '23000000',
 '24000000',
 '2400000000',
 '2479000',
 '250000',
 '2500000',
 '25000000',
 '250000000',
 '2540800',
 '2777000',
 '2800000',
 '28000000',
 '2883848',
 '3000000',
 '30000000',
 '3101000',
 '31500000',
 '3180000',
 '3200000',
 '321000000',
 '325000',
 '32500000',
 '3300000',
 '33000000',
 '350000',
 '35000000',
 '356000000',
 '3700000',
 '3800000',
 '38000000',
 '3977000',
 '4000000',
 '40000000',
 '4500000',
 '45000000',
 '46000000',
 '48000000',
 '500000',
 '5000000',
 '52000000',
 '54000000',
 '5500000',
 '55000000',
 '550000000',
 '58000000',
 '600000',
 '6000000',
 '60000000',
 '63000000',
 '6400000',
 '6800000',
 '7000000',
 '70000000',
 '72000000',
 '750000',
 '800000',
 '8000000',
 '80000000',
 '806947',
 '8200000',
 '839727',
 '9000000',
 '90000000',
 '910000',
 '92000000',
 '923000',
 '927262',
 '93000000',
 '935000',
 '94000000',
 '950000',
 '9500000',
 '95000000',
 '960000',
 '97000000',
 '97600000',
 'EM\xa032000000',
 'EM\xa06000000',
 'Not Available',
 'RF\xa0115000000'}
```

In [9]:

```
#converting all the currency in dollar and only keeping numeric value

def convert_currency(value):
  if "DEM" in value:
    value = value.replace('DEM\xa0','')
    value = float(value)*0.54
  elif "FRF" in value:
    value = value.replace('FRF\xa0','')
    value = float(value)*0.16
  return value
```

In [10]:

```
#removing the $ sign from the value

df['budget'] = df['budget'].str.replace('$','')
df['budget'] = df['budget'].apply(convert_currency)
df['budget'] = pd.to_numeric(df['budget'],
                             errors='coerce',
                             downcast='float')
```

```
/tmp/ipykernel_138/4123476812.py:3: FutureWarning: The default value of r
egex will change from True to False in a future version. In addition, sin
gle character regular expressions will *not* be treated as literal string
s when regex=True.
  df['budget'] = df['budget'].str.replace('$','')
```

Box Office Collection

In [11]:

```
set(df['box_office'])
```

Out[11]:

```
{'1006234167',
 '101209702',
 '104880868',
 '106285522',
 '1067316101',
 '1074458282',
 '1081169825',
 '109114817',
 '109676311',
 '1098',
 '1109802321',
 '1146457748',
 '11477',
 '11487676',
 '116112375',
 '1166717',
 '117250402',
 '120072577',
 '12180',
 '122126687',
 '1226507',
 '126216940',
 '127244',
 '13019063',
 '131060248',
 '1342359942',
 '134686457',
 '1349711',
 '135342',
 '138545632',
 '141603197',
 '142319',
 '144738',
 '14480',
 '14651',
 '1488732821',
 '15000000',
 '151052',
 '15421226',
 '156000000',
 '159167799',
 '159428329',
 '162861289',
 '1645133',
 '16767475',
 '170005875',
 '171627434',
 '173082189',
 '1740429',
 '174122191',
 '180563636',
 '180906076',
 '187436818',
 '187733202',
 '18778738',
 '1921847111',
 '1940906',
 '195088',
 '19569225',
 '19632715',
 '199632',
 '2000288',
 '2052415039',
 '20745728',
 '20908467',
 '213216216',
 '213928762',
 '216639112',
 '216763646',
 '21897373',
 '225508210',
 '228178',
 '22926076',
 '230098753',
 '23308615',
 '23335817',
 '23341568',
 '23402427',
 '235860116',
 '23661347',
 '23690757',
 '237536126',
 '23875127',
 '24427162',
 '250341816',
 '25253887',
 '25637669',
 '26023860',
 '262676096',
 '264118201',
 '26960374',
 '269958228',
 '27200000',
 '272742922',
 '2799439100',
 '28570902',
 '286801374',
 '28884504',
 '2913644',
 '291480452',
 '29225935',
 '29328',
 '294805697',
 '300073',
 '30680793',
 '311212',
 '31207',
 '316791257',
 '32052925',
 '321457747',
 '321752656',
 '322161245',
 '327333559',
 '33882243',
 '346258',
 '35401758',
 '35566',
 '355822319',
 '356296601',
 '37034579',
 '373672993',
 '3753929',
 '383336762',
 '384479940',
 '389925971',
 '394436586',
 '40047078',
 '402382193',
 '406878233',
 '415261382',
 '4164283',
 '41722424',
 '41960',
 '424208848',
 '426074373',
 '426588510',
 '436655',
 '441306145',
 '45720631',
 '4626532',
 '46358827',
 '463618',
 '467222728',
 '46808',
 '46969409',
 '47036784',
 '47335804',
 '474171806',
 '475106177',
 '476512065',
 '47961919',
 '482349603',
 '49074379',
 '49396747',
 '494879471',
 '5014000',
 '503162313',
 '50419',
 '516962',
 '52066791',
 '520881154',
 '521311890',
 '52287414',
 '5252',
 '52767889',
 '538375067',
 '5435024',
 '5473337',
 '56675895',
 '579707738',
 '5987386',
 '599146',
 '60262836',
 '60418',
 '60611975',
 '619179950',
 '623726085',
 '631607053',
 '654264015',
 '65884703',
 '65889846',
 '67',
 '672806432',
 '678226465',
 '6788659',
 '71108591',
 '72275',
 '735099102',
 '7390108',
 '74036715',
 '74437720',
 '750000 (estimated)',
 '76182388',
 '7693',
 '77356942',
 '773867216',
 '775398007',
 '7798146',
 '78371200',
 '800000 (estimated)',
 '81379',
 '814337054',
 '83557872',
 '836848102',
 '83862032',
 '8574081',
 '858848019',
 '898204420',
 '910000 (estimated)',
 '92181574',
 '941637960',
 '947944270',
 '9523464',
 '955',
 '962002',
 '96302',
 '968511805',
 '96983009',
 '970263',
 '98690254',
 'Not Available'}
```

In [12]:

```
#removing the estimated and taking float value

df['box_office'] = df['box_office'].str.replace('(estimated)','')
df['box_office'] = pd.to_numeric(df['box_office'],
                                errors='coerce',
                                downcast='integer')
```

```
/tmp/ipykernel_138/744310834.py:3: FutureWarning: The default value of re
gex will change from True to False in a future version.
  df['box_office'] = df['box_office'].str.replace('(estimated)','')
```

Run Time

In [13]:

```
#converting the run-time of movies into minutes

def convert_to_minutes(time):
  try:
    total_minutes = 0
    if 'h' in time:
      time = time.split('h')
      hours = int(time[0])
      mins = int(time[1][:-1]) if time[1][:-1].strip().isdigit() else 0
      total_minutes = hours*60 + mins
    else:
        total_minutes = int(time[:-1])
    return total_minutes
  except ValueError:
    return float('NaN')
```

In [14]:

```
df['run_time'] = df['run_time'].apply(convert_to_minutes)
```

In [15]:

```
#splitting genre, directors, writers and casts and made a list

df['genre'] = df['genre'].str.split(",")   #genre
df['directors'] = df['directors'].str.split(",")   #directors
df['writers'] = df['writers'].str.split(",")    #writers
df['casts'] = df['casts'].str.split(",")       #casts
```

In [16]:

```
df.head()
```

Out[16]:

	rank	name	year	rating	genre	certificate	run_time	tagline	budget
0	1	The Shawshank Redemption	1994	9.3	[Drama]	R	142.0	Fear can hold you prisoner. Hope can set you f...	25000000.0
1	2	The Godfather	1972	9.2	[Crime, Drama]	R	175.0	An offer you can't refuse.	6000000.0
2	3	The Dark Knight	2008	9.0	[Action, Crime, Drama]	PG-13	152.0	Why So Serious?	185000000.0
3	4	The Godfather Part II	1974	9.0	[Crime, Drama]	R	202.0	All the power on earth can't change destiny.	13000000.0
4	5	12 Angry Men	1957	9.0	[Crime, Drama]	Approved	96.0	Life Is In Their Hands -- Death Is On Their Mi...	350000.0

In [17]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   rank         250 non-null    int64  
 1   name         250 non-null    object 
 2   year         250 non-null    int64  
 3   rating       250 non-null    float64
 4   genre        250 non-null    object 
 5   certificate  250 non-null    object 
 6   run_time     249 non-null    float64
 7   tagline      250 non-null    object 
 8   budget       208 non-null    float32
 9   box_office   217 non-null    float64
 10  casts        250 non-null    object 
 11  directors    250 non-null    object 
 12  writers      250 non-null    object 
dtypes: float32(1), float64(3), int64(2), object(7)
memory usage: 24.5+ KB
```

In [18]:

```
df.describe()
```

Out[18]:

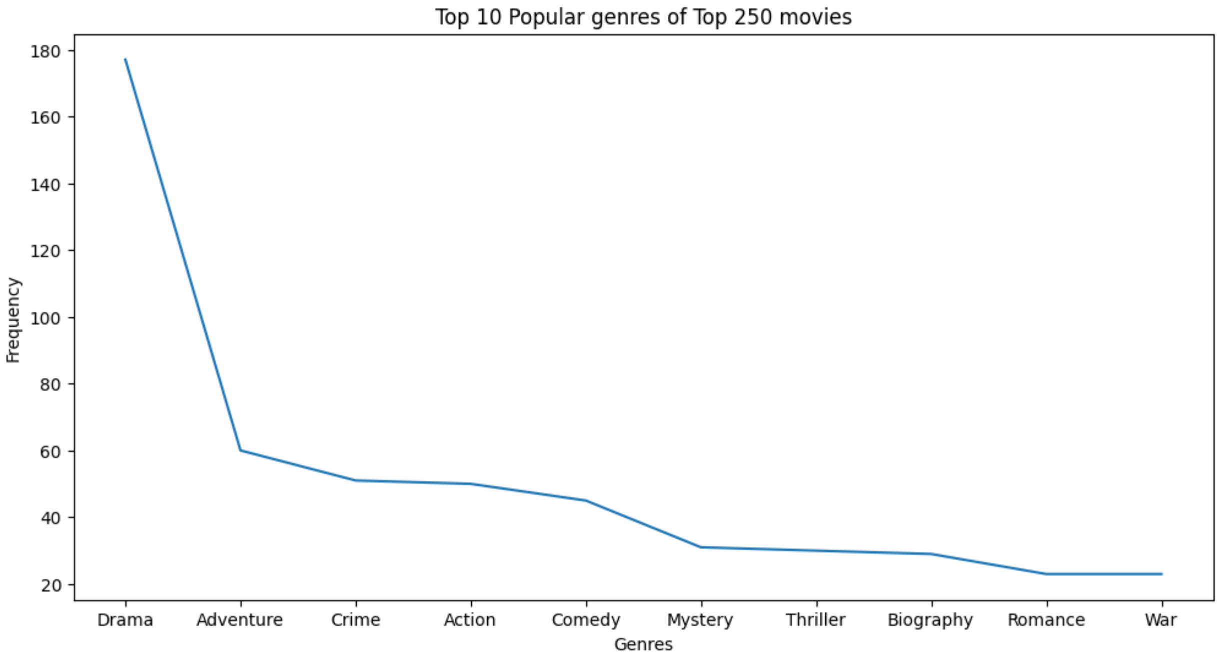
	rank	year	rating	run_time	budget	box_office
count	250.000000	250.000000	250.000000	249.000000	2.080000e+02	2.170000e+02
mean	125.500000	1986.360000	8.307200	129.052209	5.245898e+07	2.382076e+08
std	72.312977	25.125356	0.229081	30.223383	1.770686e+08	3.825392e+08
min	1.000000	1921.000000	8.000000	45.000000	1.330000e+05	6.700000e+01
25%	63.250000	1966.250000	8.100000	107.000000	3.000000e+06	8.574081e+06
50%	125.500000	1994.000000	8.200000	126.000000	1.500000e+07	7.403672e+07
75%	187.750000	2006.000000	8.400000	145.000000	4.900000e+07	3.217527e+08
max	250.000000	2022.000000	9.300000	238.000000	2.400000e+09	2.799439e+09

EDA (Exploratory Data Analysis)

What are the popular genres of Top 250 movies?

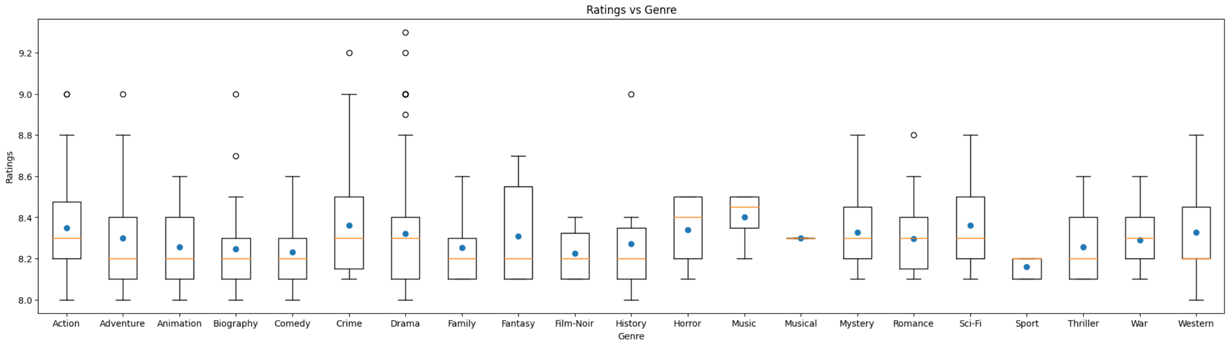
In [19]:

```
diff_genres_count = df['genre'].explode().value_counts()
plt.figure(figsize=(12, 6))
plt.plot(diff_genres_count[:10])
plt.xlabel("Genres")
plt.ylabel("Frequency")
plt.title("Top 10 Popular genres of Top 250 movies")
plt.show()
```



How does ratings vary across the genres?

In [20]:
```
rating_per_genre = df[['genre', 'rating']].explode('genre').groupby('genr
genre_rating_dict = rating_per_genre['rating'].apply(list).to_dict()
grouped_data = list(genre_rating_dict.values())
mean_rating_per_genre = rating_per_genre.mean()
plt.figure(figsize=(24, 6))
plt.boxplot(grouped_data)
plt.plot(range(1, len(genre_rating_dict) + 1), mean_rating_per_genre, 'o'
plt.xticks(range(1, len(genre_rating_dict) + 1), list(genre_rating_dict.k
plt.xlabel("Genre")
plt.ylabel("Ratings")
plt.title("Ratings vs Genre")
plt.show()
```



```
     Q1-1.5IQR   Q1       median mean   Q3   Q3+1.5IQR
                  |---------:------------|
  o      |--------|         :      o     |--------|    o  o
                  |---------:------------|
flier             <---------------------->
fliers
                            IQR
```

What were the categories of Top 250 Movies?

There are several box office categories but there is no specific formula or threshold for determining the categories. Instead, the categories are based on industry standards and expectations, as well as the production budget and marketing costs of the movie. For the sake of simplicity, we're using the following formula in comparision to their budget.

- **All time Blockbuster**: Above 300%
- **Blockbuster**: 200%-300%
- **Super Hit**: 175%-200%
- **Hit**: 125%-175%
- **Average**: 100%-125%
- **Flop**: 75%-100%
- **Super Flop**: 50%-75%
- **Disaster**: Below 50%

In [21]:

```
#categorising on the basis of ROI
#applying conditions to roi
def convert_to_categories(roi):
  if roi > 3.0:
    return "All Time Blockbuster"
  elif roi > 2.0:
    return "Blockbuster"
  elif roi > 1.75:
    return "Super Hit"
  elif roi > 1.25:
    return "Hit"
  elif roi > 1.0:
    return "Average"
  elif roi > 0.75:
    return "Flop"
  elif roi > 0.5:
    return "Super Flop"
  else:
    return "Disaster"
```

In [22]:

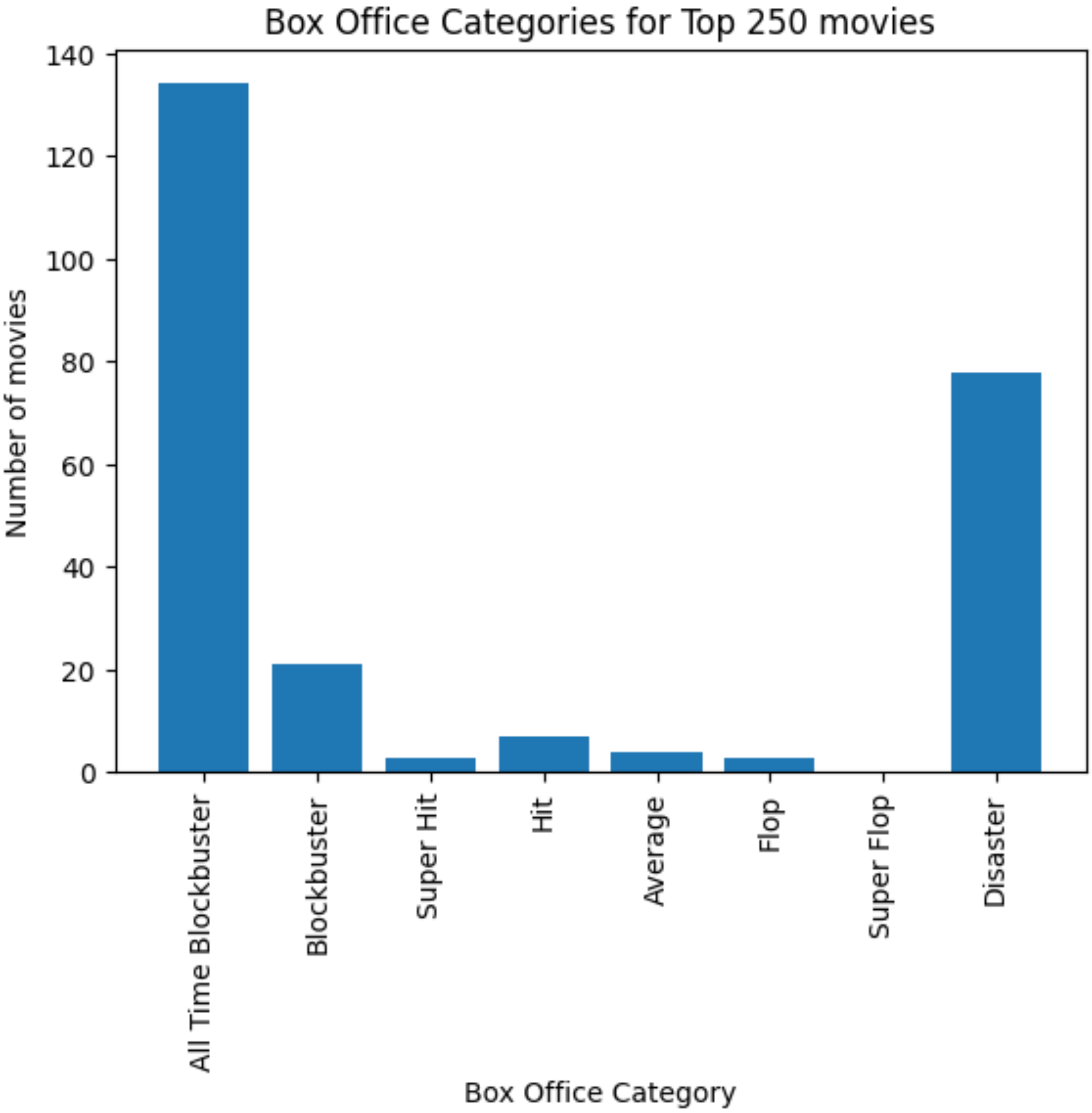
```python
#calculated roi(return of investment)
df['roi'] = df['box_office']/df['budget']         #formula for ROI
df['box-office-status'] = df['roi'].apply(convert_to_categories)
df.head()
```

Out[22]:

	rank	name	year	rating	genre	certificate	run_time	tagline	budget
0	1	The Shawshank Redemption	1994	9.3	[Drama]	R	142.0	Fear can hold you prisoner. Hope can set you f...	25000000.0
1	2	The Godfather	1972	9.2	[Crime, Drama]	R	175.0	An offer you can't refuse.	6000000.0
2	3	The Dark Knight	2008	9.0	[Action, Crime, Drama]	PG-13	152.0	Why So Serious?	185000000.0
3	4	The Godfather Part II	1974	9.0	[Crime, Drama]	R	202.0	All the power on earth can't change destiny.	13000000.0
4	5	12 Angry Men	1957	9.0	[Crime, Drama]	Approved	96.0	Life Is In Their Hands -- Death Is On Their Mi...	350000.0

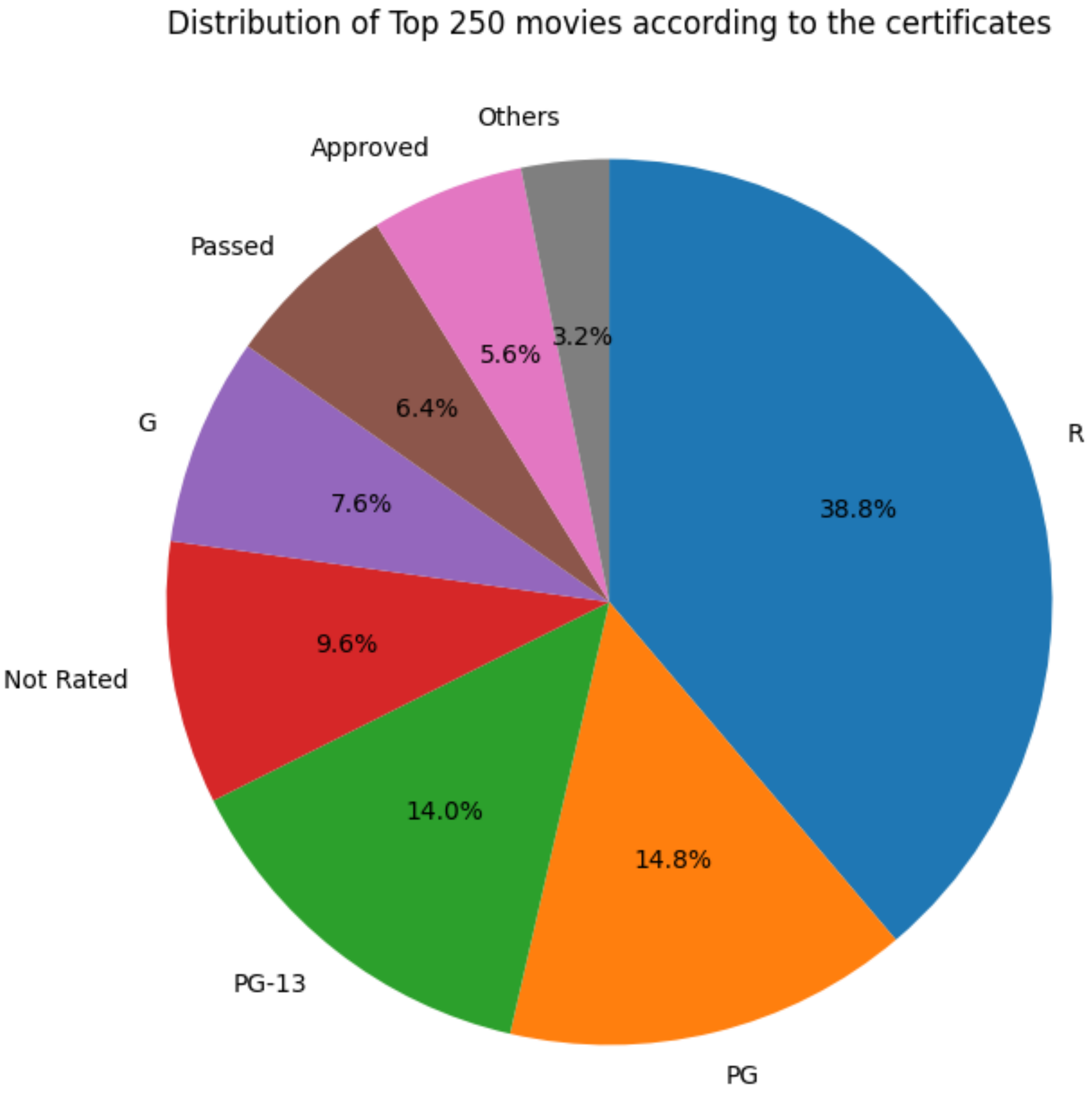
In [23]:

```
#plotting according to its success rate

xticks = ["All Time Blockbuster", "Blockbuster", "Super Hit", "Hit", "Ave
box_office_status = df['box-office-status'].value_counts()
box_office_status = box_office_status.reindex(xticks)
plt.bar(box_office_status.keys(), box_office_status.values)
plt.xlabel("Box Office Category")
plt.ylabel("Number of movies")
plt.xticks(rotation="vertical")
plt.title("Box Office Categories for Top 250 movies")
plt.show()
```

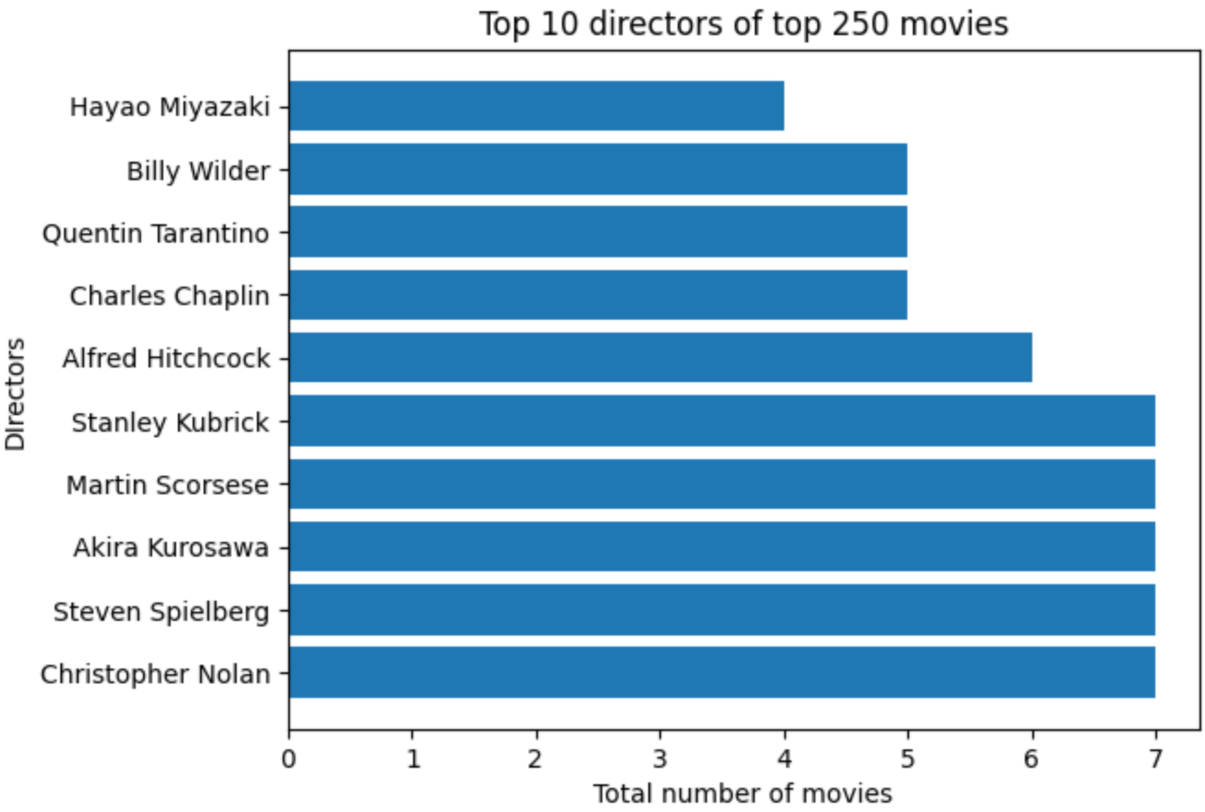


In [24]:

```
#distribution according to the certificates

movie_certs = df['certificate'].value_counts()
others_count = movie_certs[movie_certs < 5].sum()
movie_certs = movie_certs.drop(movie_certs[movie_certs < 5].index.tolist(
movie_certs['Others'] = others_count
plt.figure(figsize=(8, 8))
plt.pie(movie_certs, \
        labels=movie_certs.keys(), \
        startangle = 90, \
        autopct='%1.1f%%', \
        counterclock=False)
plt.title("Distribution of Top 250 movies according to the certificates")
plt.show()
```



In [25]:
```
#top-10 directors plotted

best_directors = df['directors'].explode().value_counts()
plt.barh(best_directors.keys()[:10], best_directors.values[:10])
plt.ylabel("DIrectors")
plt.xlabel("Total number of movies")
plt.title("Top 10 directors of top 250 movies")
# plt.plot(best_directors[:5])
plt.show()
```



In [26]:
```
dir_probs = best_directors/250
cast_probs = df['casts'].explode().value_counts()/250
writer_probs = df['writers'].explode().value_counts()/250
certs_probs = movie_certs/250
genre_probs = diff_genres_count/250

#calculated probability ... applied regression

#for genres
def convert_genres(genres):
  genre_factor = -0.06
  genre_prob = 0
  for genre in genres:
    genre_prob += genre_probs[genre] if genre in genre_probs else genre_f

  return genre_prob

#for certificates
def convert_certificate(certificate):
  return certs_probs[certificate] if certificate in certs_probs\
   else certs_probs['Others']
```

```
#for directors

def convert_directors(directors):
  dir_factor = -0.02
  dir_prob = 0
  for director in directors:
    dir_prob += dir_probs[director] if director in dir_probs else dir_fac

  return dir_prob

#for casts

def convert_casts(casts):
  cast_factor = -0.015
  cast_prob = 0
  for cast in casts:
    cast_prob += cast_probs[cast] if cast in cast_probs else cast_factor

  return cast_prob

#for writers

def convert_writers(writers):
  writer_factor = -0.007
  writer_prob = 0
  for writer in writers:
    writer_prob += writer_probs[writer] if writer in writer_probs else wr

  return writer_prob

regression_df = df.copy()
regression_df['genre'] = regression_df['genre'].apply(convert_genres)
regression_df['certificate'] = regression_df['certificate'].apply(convert
regression_df['directors'] = regression_df['directors'].apply(convert_dir
regression_df['casts'] = regression_df['casts'].apply(convert_casts)
regression_df['writers'] = regression_df['writers'].apply(convert_writers

regression_df['run_time'] = regression_df['run_time'].fillna(regression_d
regression_df['budget'] = regression_df['budget'].fillna(regression_df['r
regression_df['box_office'] = regression_df['box_office'].fillna(regressi
regression_df['roi'] = regression_df['box_office']/regression_df['budget'
regression_df = regression_df.drop(columns=['name', 'rank', 'tagline', 'b

regression_df.head()
```

Out[26]:

	year	rating	genre	certificate	run_time	budget	box_office	casts	directo
0	1994	9.3	0.708	0.388	142.0	25000000.0	2.888450e+07	0.116	0.00
1	1972	9.2	0.912	0.388	175.0	6000000.0	2.503418e+08	0.144	0.0
2	2008	9.0	1.112	0.140	152.0	185000000.0	1.006234e+09	0.164	0.02
3	1974	9.0	0.912	0.388	202.0	13000000.0	4.796192e+07	0.172	0.0
4	1957	9.0	0.912	0.056	96.0	350000.0	9.550000e+02	0.112	0.00

In [27]:

```
regression_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   year         250 non-null    int64  
 1   rating       250 non-null    float64
 2   genre        250 non-null    float64
 3   certificate  250 non-null    float64
 4   run_time     250 non-null    float64
 5   budget       250 non-null    float32
 6   box_office   250 non-null    float64
 7   casts        250 non-null    float64
 8   directors    250 non-null    float64
 9   writers      250 non-null    float64
 10  roi          250 non-null    float64
dtypes: float32(1), float64(9), int64(1)
memory usage: 20.6 KB
```

In [28]:

```
#calculated the accuracy

def count_accuracy(y_true, y_pred):
  threshold = 0.2
  count = 0
  for i in range(len(y_true)):
    if abs(y_true[i]-y_pred[i]) < threshold:
      count += 1
  return count/len(y_true)
```

Test

In [29]:

```
#linear regression

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
#Define the model
model = LinearRegression()
y = np.array(regression_df['rating'])
x = np.array(regression_df.loc[:, regression_df.columns!='rating'])
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.15,
reg = model.fit(X_train, y_train)
y_pred = reg.predict(X_test)
accuracy = count_accuracy(y_test, y_pred)
#Printed the accuracy
print(accuracy)
```

```
0.6578947368421053
```

In [30]:

```
#Support Vector Regression (SVR)

from sklearn.svm import SVR
#Define the model
model = SVR()
y = np.array(regression_df['rating'])
x = np.array(regression_df.loc[:, regression_df.columns!='rating'])
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.15,
reg = model.fit(X_train, y_train)
y_pred = reg.predict(X_test)
accuracy = count_accuracy(y_test, y_pred)
#Printed the accuracy
print(accuracy)
```

```
0.7631578947368421
```

In [ ]: