In [ ]:
```
working_df = pd.read_csv('post_wiki_scrape.csv')
working_df = working_df.set_index(['start_year', 'title'])
working_df.head()
```

Imports

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import os
import seaborn as sns
import requests
from bs4 import BeautifulSoup
import re
import json
from inspect import currentframe, getframeinfo
```

Read in provided data

In [2]:
```
data_imports = {}

# Importing all the provided CSVs and TSVs to a dictionary
for file in os.listdir("zippedData"):
    print(file)
    if file[-3:] == 'csv':
        data_imports[file[:-4]] = pd.read_csv('zippedData/' + str(file))
    elif file[-3:] == 'tsv':
        data_imports[file[:-4]] = pd.read_csv('zippedData/' + str(file), sep

print("Finished import")
```

```
bom.movie_gross.csv
name.basics.csv
rt.movie_info.tsv
rt.reviews.tsv
title.akas.csv
title.basics.csv
title.crew.csv
title.principals.csv
title.ratings.csv
tmdb.movies.csv
tn.movie_budgets.csv
Finished import
```

Starting with Bom.Movie_Gross, I want to set the index as release year and then title

In [3]:
```
bom_movie_gross = data_imports['bom.movie_gross']
# bom_movie_gross[bom_movie_gross['title'].duplicated(keep=False)]
```

In [4]:
```
bom_movie_gross = bom_movie_gross.rename(columns = {'year': 'start_year'})
bom_movie_gross = bom_movie_gross.set_index(['start_year', 'title'])
bom_movie_gross.head()
```

Out[4]:

		studio	domestic_gross	foreign_gross
start_year	**title**			
2010	**Toy Story 3**	BV	415000000.0	652000000
	Alice in Wonderland (2010)	BV	334200000.0	691300000
	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000
	Inception	WB	292600000.0	535700000
	Shrek Forever After	P/DW	238700000.0	513900000

In an effort to explore the data, we'll merge title.basics with title.ratings into imdb_df

In [5]:
```
title_basics = data_imports['title.basics']
title_ratings = data_imports['title.ratings']

title_basics = title_basics.set_index('tconst')
title_ratings = title_ratings.set_index('tconst')
imdb_df = title_basics.join(title_ratings, on='tconst')

```

Merge imdb_df with tmdb.movies

In [6]:
```
tmdb_movies = data_imports['tmdb.movies']
tmdb_movies = tmdb_movies.drop_duplicates(subset = ['id'])
tmdb_movies['start_year'] = tmdb_movies['release_date'].apply(lambda x: int(
tmdb_movies = tmdb_movies.set_index(['start_year', 'title'])
tmdb_movies.head()
```

```
/home/stonehengee/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:
3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  This is separate from the ipykernel package so we can avoid doing imports unt
il
```

Out[6]:

		Unnamed: 0	genre_ids	id	original_language	original_title	popularity	rele
start_year	**title**							
2010	**Harry Potter and the Deathly Hallows: Part 1**	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2(
	How to Train Your Dragon	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2(
	Iron Man 2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2(
1995	**Toy Story**	3	[16, 35, 10751]	862	en	Toy Story	28.005	1(
2010	**Inception**	4	[28, 878, 12]	27205	en	Inception	27.920	2(

In [7]:
```
imdb_df = imdb_df.rename(columns = {'primary_title': "title"})
imdb_df = imdb_df.reset_index()
working_df = imdb_df.set_index(['start_year', 'title']).join(tmdb_movies, ho


working_df = working_df.drop(columns=['Unnamed: 0', 'vote_average', 'vote_co
# working_df = working_df.fillna(value = {'genres': 'Unknown'})

working_df.head()

```

Out[7]:

		tconst	original_title_imdb	runtime_minutes	genres	averagerating	numvotes
start_year	title						
1930	All Quiet on the Western Front	NaN	NaN	NaN	NaN	NaN	NaN
1933	The Vampire Bat	NaN	NaN	NaN	NaN	NaN	NaN
1936	Le Bonheur	NaN	NaN	NaN	NaN	NaN	NaN
1939	How Walt Disney Cartoons Are Made	NaN	NaN	NaN	NaN	NaN	NaN
1946	The Best Years of Our Lives	NaN	NaN	NaN	NaN	NaN	NaN

Add in bom_movie_gross

In [8]:
```
working_df = working_df.join(bom_movie_gross, how='outer')
```

Add in tn_movie_budgets

In [9]:
```
tn_movie_budgets = data_imports['tn.movie_budgets']
```

In [11]:
```
working_df['release_date'] = pd.to_datetime(working_df['release_date'])
tn_movie_budgets['start_year'] = pd.DatetimeIndex(tn_movie_budgets['release_
tn_movie_budgets = tn_movie_budgets.rename(columns={'movie':'title'})
tn_movie_budgets = tn_movie_budgets.set_index(['start_year', 'title'])
```

In [12]:
```
# tn_movie_budgets.head()
working_df = working_df.join(tn_movie_budgets, how='outer', lsuffix='_imdb',

```

Drop movies from before 2000

In [13]:
```
working_df = working_df.reset_index()
working_df = working_df[working_df['start_year'] > 2000]
working_df = working_df.set_index(['start_year', 'title'])
```

Drop NaN genres

In [14]:
```
working_df = working_df.dropna(subset=['genres'])
working_df = working_df[working_df['original_language'] == 'en']
working_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 11122 entries, (2010, * Cemetery) to (2018, eHero)
Data columns (total 17 columns):
tconst                  11122 non-null object
original_title_imdb     11122 non-null object
runtime_minutes         10864 non-null float64
genres                  11122 non-null object
averagerating           10761 non-null float64
numvotes                10761 non-null float64
original_language       11122 non-null object
original_title_mvdb     11122 non-null object
release_date_imdb       11122 non-null datetime64[ns]
studio                  1430 non-null object
domestic_gross_imdb     1424 non-null float64
foreign_gross           1127 non-null object
id                      1398 non-null float64
release_date_tn         1398 non-null object
production_budget       1398 non-null object
domestic_gross_tn       1398 non-null object
worldwide_gross         1398 non-null object
dtypes: datetime64[ns](1), float64(5), object(11)
memory usage: 2.6+ MB
```

Scraping wikipedia for production budgets

At this point I had to decide whether it was better to remove foreign films from the dataset or spend the time it would require to write code to account for each individual foreign currency found through scrapping. I ultimately decided, both due to the scope of the project (Microsoft being an American company) and the marginal number of foreign films remaining in my dataset that it was better to simply return None for budgets not in US dollars.

In [15]:

```
# Takes in the string, isolated as a budget, and returns a number of type ir


def convert_budget_to_int(budget, debug):

    #     Checks for various non-alphanumeric characters

    if type(budget) == int:
        return budget
    if type(budget) == float:
        return int(budget)
    if budget.startswith('$CAD'):
        return None
    if budget.startswith('$') or budget.startswith('US$'):
        if budget[0] == '<':
            budget = budget[1:]

#  Enable for Avatar 2009, breaks other movies


#         if '$' in budget[3:]:
#             temp = budget[3:]
#             budget = budget[:temp.index('$')+3]
        while '[' in budget:
            budget = budget.replace(
                budget[budget.index('['):budget.index(']')+1], '')
        if '–' in budget:
            currency = ''
            for i in budget:
                if not i.isnumeric():
                    currency = currency + i
                else:
                    break
            if debug:
                print(budget, "Middle of dash check")
            budget = budget[budget.index('–')+1:]
            budget = currency + budget
        if '-' in budget:
            currency = ''
            for i in budget:
                if not i.isnumeric():
                    currency = currency + i
                else:
                    break
            if debug:
                print(budget, "Middle of dash check")
            budget = budget[budget.index('-')+1:]
            budget = currency + budget
        if '—' in budget:
            currency = ''
            for i in budget:
                if not i.isnumeric():
                    currency = currency + i
                else:
                    break
            if debug:
```

```
57                  print(budget, "Middle of dash check")
58              budget = budget[budget.index('–')+1:]
59              budget = currency + budget
60          if debug:
61              print(budget)
62
63  #     Using regex because for some reason ' ' would not be recognized for ce
64          if bool(re.search(r"\s", budget)):
65              whitespace_index = re.search(r"\s", budget).start()
66              if budget[whitespace_index-1].isnumeric():
67                  number = budget[:whitespace_index]
68                  word = budget[whitespace_index+1:]
69              else:
70                  whitespace_index = re.search(
71                      r"\s", budget[:whitespace_index]+budget[whitespace_index
72                  number = budget[:whitespace_index+1]
73                  word = budget[whitespace_index+2:]
74              if debug:
75                  print(word)
76                  print(number)
77
78          else:
79              number, word = budget, ''
80          if debug:
81              print(number, 'Before \'.\' check')
82          if '.' in number:
83              left, right = number.split('.')
84              decimal_places = len(right)
85              number = number.replace('.', '')
86
87  #     Replacing instnaces of million and crore (Indian for ten million) with
88
89          if 'crore' in word.lower():
90              try:
91                  number = number + '0000000'[decimal_places:]
92              except:
93                  number = number + '0000000'
94          elif 'million' in word.lower():
95              try:
96                  number = number + '000000'[decimal_places:]
97              except:
98                  number = number + '000000'
99
100         if ',' in number:
101             number = number.replace(',', '')
102
103         budget = budget.strip()
104         if debug:
105             print(budget)
106
107         if budget[0] == '$':
108             number = number.replace('$', '')
109         elif budget[:3] == 'US$':
110             number = number.replace('US$', '')
111
112         return int(number)
113
```

```
# Replaces spaces in a URL with %20


def urlify(in_string):
    return "%20".join(in_string.split())


'''Uses Wikipedia's API to search for movies.
In practice I would search by the title and year to reduce the chance of an


def wiki_search(search):
    url = "https://en.wikipedia.org/w/api.php?action=query&format=json&prop=
        urlify(search))
    response = requests.get(url=url)

    try:
        return(response.json()['query']['search'][0]['pageid'])
    except IndexError:
        return None


'''The called function which managed the search for movies on Wikipedia,
the isolating of the budget string, and ultimately the return of the budget


def wiki_grab(search, debug=False):
    searches_to_ignore = ['#Stuck 2014',
                          'House of Black Wings 2010',
                          'Restoring a Masterpiece: The Renovation of Eastma
                          'Avatar: Special Edition 2010',
                          'The Forgotten Jewel 2010',
                          'Birth of a Party 2011'
                          ]
    if search[0] == '#':
        return None
    if search in searches_to_ignore:
        return None
    pageid = wiki_search(search)
    if debug:
        print(pageid)
    if pageid is None:
        return None
    url = 'https://en.wikipedia.org/w/api.php?action=parse&format=json&pagei
        pageid)
    if debug:
        print(url)
    response = requests.get(url=url)
    soup = BeautifulSoup(response.json()['parse']['text'])

    if 'Budget</th>' in str(soup):
        if soup.find(text='Budget').next.text:
            return convert_budget_to_int(soup.find(text='Budget').next.text,
        elif '(gross)' in soup.find(text='Budget').next.text:
            gross = soup.find('li', text=re.compile(r' .+(\(gross\))')).text
            gross = gross.replace(' (gross)', '')
            gross = convert_budget_to_int(gross, debug)
```

```
            return(gross)
        elif re.compile(r' \d') in soup.find(text='Budget').next.li.text:
            return(soup.find('li', text=re.compile(r' \d')))

# A test run
# print(wiki_grab("Habermann 2010", True))

```

In [ ]:

```
working_df['budget_wiki'] = np.nan
for year, title in working_df[working_df['budget_wiki'].isna()].index.values
    try:
#         Do not search again in the senario that we're running this code mu
        if working_df.loc[(year, title), 'budget_wiki'].values[0] == -1:
            continue
        print(title, year)
        budget = wiki_grab(title + ' ' + str(year))
        if budget == None:
            working_df.loc[(year, title), 'budget_wiki'] = -1
        else:
            working_df.loc[(year, title), 'budget_wiki'] = budget
        print(working_df.loc[(year, title), 'budget_wiki'])
    except:
        working_df.loc[(year, title), 'budget_wiki'] = -1
```

Save my work

In [16]:

```
# working_df.to_csv('post_wiki_scrape.csv')

working_df = pd.read_csv('post_wiki_scrape.csv')
working_df = working_df.set_index(['start_year', 'title'])
working_df.head()
```

Out[16]:

		tconst	original_title_imdb	runtime_minutes	genres	ave
start_year	**title**					
2010	*** Cemetery**	tt1598691	* Cemetery	80.0	Comedy,Horror,Thriller	
	127 Hours	tt1542344	127 Hours	94.0	Adventure,Biography,Drama	
	12th & Delaware	tt1548865	12th & Delaware	81.0	Documentary,Drama	
	13	tt0798817	13	91.0	Drama,Thriller	
	15 Till Midnight	tt1568798	15 Till Midnight	97.0	Drama,Mystery,Sci-Fi	

In [30]:
```
# Convert financial columns to int64 for comparison

def convert_columns_to_int(budget):
    try:
        return int(float(budget))
    except:
        return convert_budget_to_int(budget, False)

for column in ['domestic_gross_imdb', 'foreign_gross', 'production_budget',
    working_df[column] = working_df[column].fillna(-1)
    working_df[column] = working_df[column].apply(lambda x: convert_columns_
    working_df[column] = working_df[column].astype('int64')

working_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 11122 entries, (2010, * Cemetery) to (2018, eHero)
Data columns (total 21 columns):
tconst                 11122 non-null object
original_title_imdb    11122 non-null object
runtime_minutes        10864 non-null float64
genres                 11122 non-null object
averagerating          10761 non-null float64
numvotes               10761 non-null float64
original_language      11122 non-null object
original_title_mvdb    11122 non-null object
release_date_imdb      11122 non-null object
studio                 1430 non-null object
domestic_gross_imdb    11122 non-null int64
foreign_gross          11122 non-null int64
id                     1398 non-null float64
release_date_tn        1398 non-null object
production_budget      11122 non-null int64
domestic_gross_tn      11122 non-null int64
worldwide_gross        11122 non-null int64
budget_wiki            11122 non-null int64
genre1                 11122 non-null object
genre2                 11122 non-null object
genre3                 11122 non-null object
dtypes: float64(4), int64(6), object(11)
memory usage: 1.9+ MB
```

In [31]:
```
'''
Derive a working worldwide gross and working budget based on the data gather
preference for information from IMDB and Wikipedia based on token examinatio
'''

working_df['working_wwg'] = working_df.apply(
    lambda x: x['domestic_gross_imdb'] + x['foreign_gross'] if x['domestic_g
        else x['worldwide_gross'], axis = 1)
working_df['working_budget'] = working_df.apply(
    lambda x: x['budget_wiki'] if x['budget_wiki'] > -1 else x['production_b
```

In [32]:
```
working_df['roi'] = working_df.apply(lambda x: x['working_wwg'] - x['working
working_df = working_df.drop_duplicates()
```

Split 'genre' column into genre1, genre2, genre3

In [34]:
```
# Find the maximum amount of commas in the genres column
comma_counter = 0

for each in working_df['genres']:
    if type(each) == str:
        current_count = each.count(',')
        if current_count > comma_counter:
            comma_counter = current_count

comma_counter
```

Out[34]: 2

In [35]:
```
working_df[['genre1', 'genre2', 'genre3']] = working_df['genres'].str.split(

```

replacing NaN with "Unknown"

In [36]:
```
working_df = working_df.fillna(value = {
    'genres': 'Unknown', 'genre1': 'Unknown', 'genre3': 'Unknown', 'genre2':
})

analysis_df = working_df[[
    'runtime_minutes',
    'genres',
    'genre1',
    'genre2',
    'genre3',
    'working_wwg',
    'working_budget',
    'roi'
]].copy()
analysis_df.head()
```

Out[36]:

start_year	title	runtime_minutes	genres	genre1	genre2	genre3
2010	* Cemetery	80.0	Comedy,Horror,Thriller	Comedy	Horror	Thriller
	127 Hours	94.0	Adventure,Biography,Drama	Adventure	Biography	Drama
	12th & Delaware	81.0	Documentary,Drama	Documentary	Drama	Unknown
	13	91.0	Drama,Thriller	Drama	Thriller	Unknown
	15 Till Midnight	97.0	Drama,Mystery,Sci-Fi	Drama	Mystery	Sci-Fi

In [37]:
```
analysis_df = analysis_df[analysis_df['roi'] != 0]
analysis_df = analysis_df[analysis_df['working_wwg'] != -1]
analysis_df = analysis_df[analysis_df['working_budget'] != -1]
analysis_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 1512 entries, (2010, 127 Hours) to (2018, Winchester)
Data columns (total 8 columns):
runtime_minutes    1502 non-null float64
genres             1512 non-null object
genre1             1512 non-null object
genre2             1512 non-null object
genre3             1512 non-null object
working_wwg        1512 non-null int64
working_budget     1512 non-null int64
roi                1512 non-null int64
dtypes: float64(1), int64(3), object(4)
memory usage: 180.9+ KB
```

In [38]:
```
analysis_df = analysis_df.reset_index()
analysis_df= analysis_df.rename(columns = {'start_year': 'year'})
```

Analysis

We'll break up the dataframe along certain budget markers and create series through which we can analyse the performance of different genres

Note that this method double and tripple counts certain movies with more than one listed genre, which is why it's important for us to eliminate genres with too few examples to prevent skewing.

In [39]:
```
genre_roi = pd.concat([pd.Series(analysis_df['roi'].values, analysis_df['gen
                       pd.Series(analysis_df['roi'].values, analysis_df['gen
                       pd.Series(analysis_df['roi'].values, analysis_df['gen

analysis_df_1mil = analysis_df[analysis_df['working_budget'] >= 1000000].cop

genre_roi1 = pd.concat([pd.Series(analysis_df_1mil['roi'].values, analysis_d
                       pd.Series(analysis_df_1mil['roi'].values, analysis_df
                       pd.Series(analysis_df_1mil['roi'].values, analysis_df

analysis_df_10mil = analysis_df[analysis_df['working_budget'] >= 10000000].c

genre_roi10 = pd.concat([pd.Series(analysis_df_10mil['roi'].values, analysis
                       pd.Series(analysis_df_10mil['roi'].values, analysis_d
                       pd.Series(analysis_df_10mil['roi'].values, analysis_d
```

Let's check how many of each genre we have so we can prevent skewing

In [40]:
```
1 pd.DataFrame(genre_roi).reset_index()['index'].value_counts()
```

Out[40]:
```
Drama          735
Unknown        686
Comedy         519
Action         443
Adventure      358
Thriller       270
Crime          237
Romance        196
Horror         180
Biography      140
Sci-Fi         128
Mystery        128
Fantasy        125
Animation      104
Family          92
Music           47
Documentary     44
History         40
Sport           29
War             15
Western         11
Musical          8
Reality-TV       1
Name: index, dtype: int64
```

In [41]:
```
1 pd.DataFrame(genre_roi1).reset_index()['index'].value_counts()
```

Out[41]:
```
Drama          699
Unknown        627
Comedy         503
Action         437
Adventure      358
Thriller       255
Crime          233
Romance        185
Horror         167
Biography      137
Fantasy        122
Sci-Fi         121
Mystery        117
Animation      104
Family          90
Music           46
History         40
Documentary     36
Sport           29
War             15
Western         11
Musical          8
Reality-TV       1
Name: index, dtype: int64
```

In [42]:
```
1 pd.DataFrame(genre_roi10).reset_index()['index'].value_counts()
```

Out[42]:
```
Drama          529
Comedy         406
Unknown        405
Action         396
Adventure      336
Crime          195
Thriller       186
Romance        141
Fantasy        111
Biography      111
Sci-Fi         106
Animation       99
Horror          95
Mystery         79
Family          77
History         35
Music           34
Sport           24
Documentary     15
War             13
Western          8
Musical          7
Name: index, dtype: int64
```

In [43]:
```
1 genre_roi = genre_roi.drop(labels=['Reality-TV', 'Musical'])
2 genre_roi1 = genre_roi1.drop(labels=['Reality-TV', 'Musical'])
3 genre_roi10 = genre_roi10.drop(labels=['Western', 'Musical'])
4
5
```

Before we graph these sets, we're going to reset the index and properly name the roi column

In [44]:
```
1 genre_roi_df = pd.DataFrame(genre_roi).reset_index().rename(columns={0:'roi'
2 genre_roi1_df = pd.DataFrame(genre_roi1).reset_index().rename(columns={0:'ro
3 genre_roi10_df = pd.DataFrame(genre_roi10).reset_index().rename(columns={0:'
```

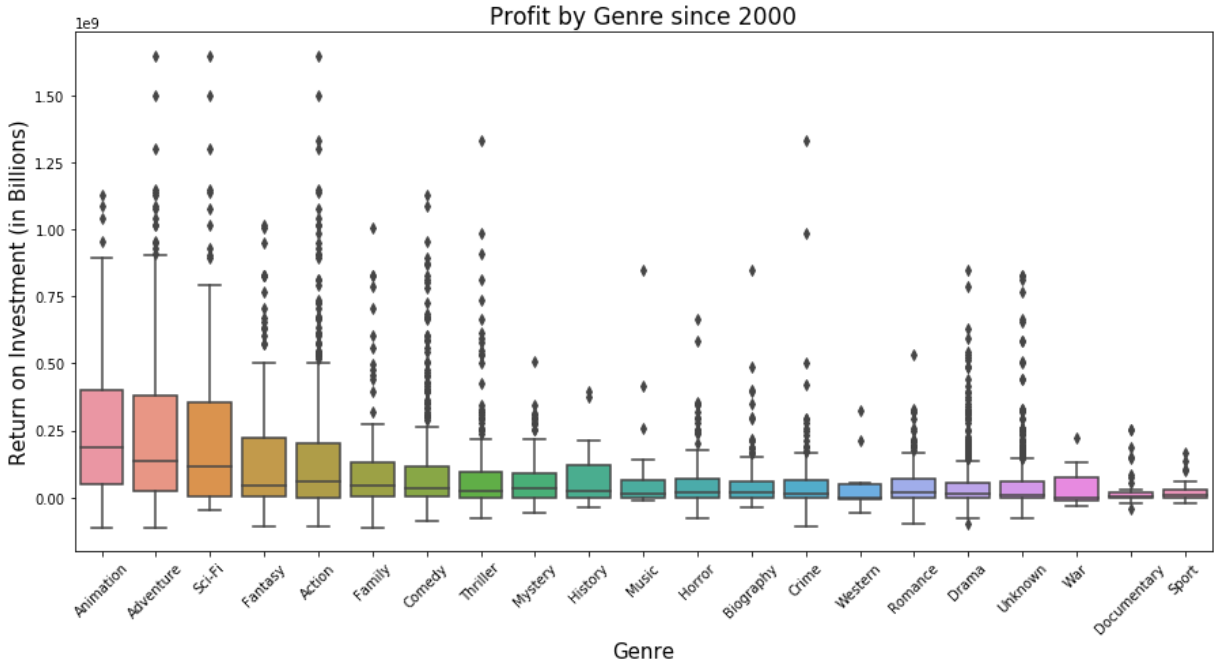
General genre return on investment breakdown (no budget floor, excluded genres with fewer than 10 films in dataset)

In [45]:
```
my_order = genre_roi_df.groupby("index")["roi"].mean().sort_values().iloc[::


plt.figure(figsize=(15, 7))

ax = sns.boxplot(x=genre_roi_df['index'], y=genre_roi_df['roi'], data=(genre
plt.setp(ax.get_xticklabels(), rotation=45, fontsize=10)
plt.setp(ax.get_yticklabels(), fontsize = 10)

ax.set_title('Profit by Genre since 2000', fontsize = 17)
ax.set_xlabel('Genre', fontsize=15);
ax.set_ylabel('Return on Investment (in Billions)', fontsize=15);
```



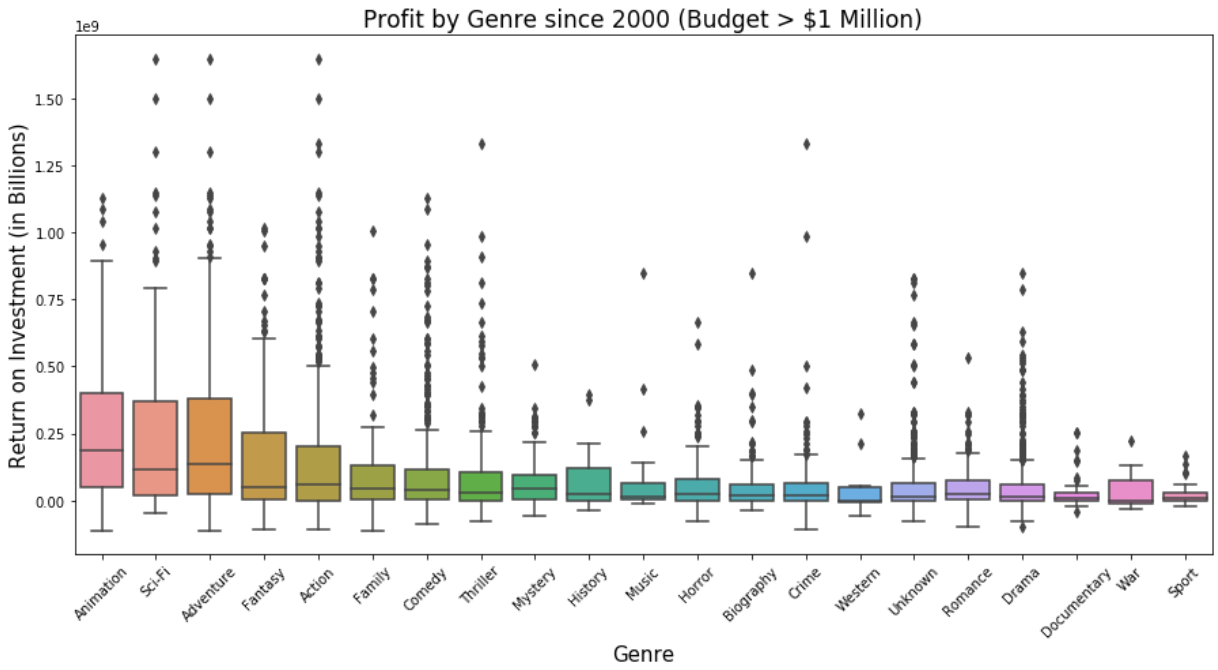
General genre return on investment breakdown (budget >$1 million, excluded genres with fewer than 10 films in dataset)

In [46]:
```
my_order = genre_roi1_df.groupby("index")["roi"].mean().sort_values().iloc[:


plt.figure(figsize=(15, 7))

ax = sns.boxplot(x=genre_roi1_df['index'], y=genre_roi1_df['roi'], data=(gen
plt.setp(ax.get_xticklabels(), rotation=45, fontsize=10)
plt.setp(ax.get_yticklabels(), fontsize = 10)

ax.set_title('Profit by Genre since 2000 (Budget > $1 Million)', fontsize =
ax.set_xlabel('Genre', fontsize=15);
ax.set_ylabel('Return on Investment (in Billions)', fontsize=15);
```



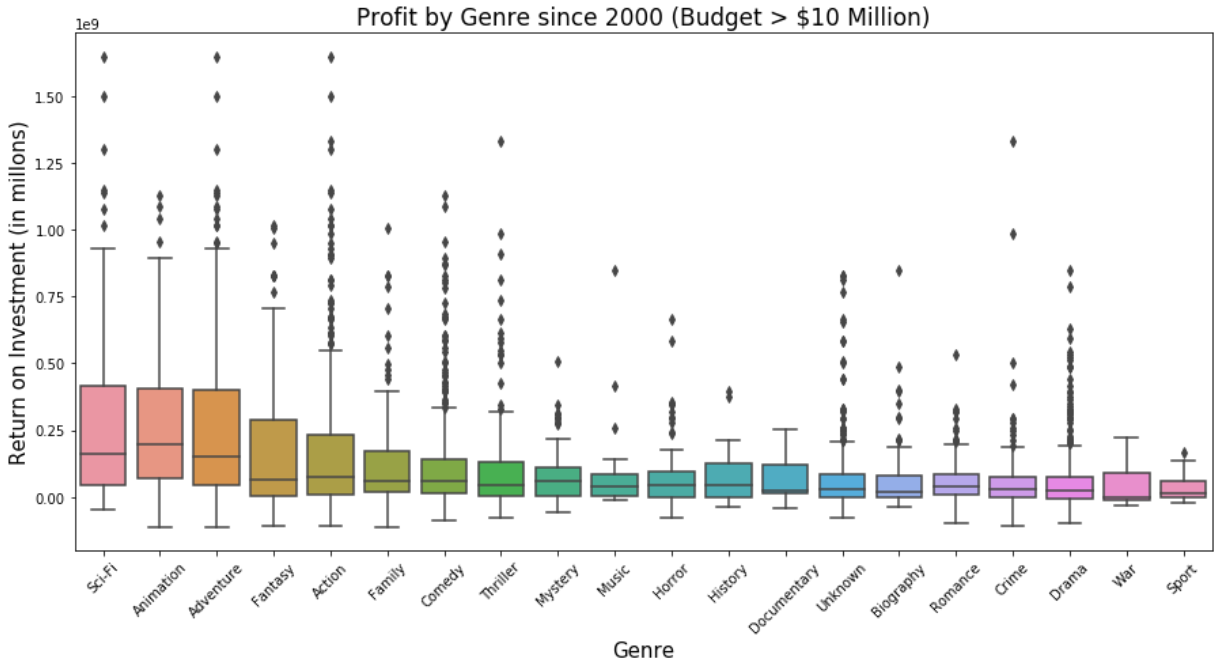
General genre return on investment breakdown (budget >$10 million, excluded genres with fewer than 10 films in dataset)

In [47]:
```
my_order = genre_roi10_df.groupby("index")["roi"].mean().sort_values().iloc[


plt.figure(figsize=(15, 7))

ax = sns.boxplot(x=genre_roi10_df['index'], y=genre_roi10_df['roi'], data=(g
plt.setp(ax.get_xticklabels(), rotation=45, fontsize=10)
plt.setp(ax.get_yticklabels(), fontsize = 10)

ax.set_title('Profit by Genre since 2000 (Budget > $10 Million)', fontsize =
ax.set_xlabel('Genre', fontsize=15);
ax.set_ylabel('Return on Investment (in millons)', fontsize=15);
```

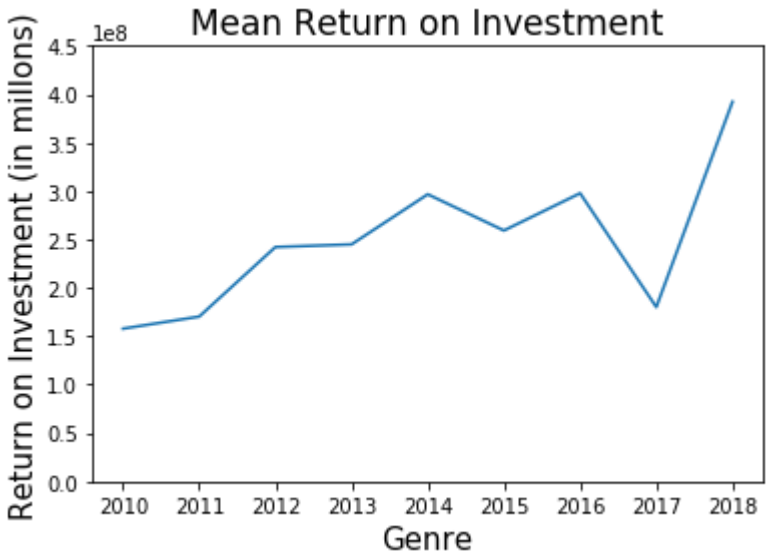


In [48]:

```
genre_roi2_scifi = pd.DataFrame(pd.concat([pd.Series(analysis_df_1mil.loc[an
                                pd.Series(analysis_df_1mil.loc[analysis_df_1m
                                pd.Series(analysis_df_1mil.loc[analysis_df_1m
                               )).reset_index().rename(columns={0:'year'})

grouped_mean = genre_roi2_scifi.groupby('year').mean()

ax = sns.lineplot(x= grouped_mean.index, y=grouped_mean.roi, data=grouped_me
plt.ylim(0, 450000000)
ax.set_title('Mean Return on Investment', fontsize = 17)
ax.set_xlabel('Genre', fontsize=15);
ax.set_ylabel('Return on Investment (in millons)', fontsize=15);
```



In [49]:
```
working_df[working_df['genres'].str.contains('Sci-Fi')]
```

Out[49]:

start_year	title	tconst	original_title_imdb	runtime_minutes	genres	av
2010	15 Till Midnight	tt1568798	15 Till Midnight	97.0	Drama,Mystery,Sci-Fi	
	Alien Vengeance II: Rogue Element	tt1637674	Alien Vengeance II: Rogue Element	78.0	Sci-Fi	
	Altitude	tt1407049	Altitude	90.0	Horror,Mystery,Sci-Fi	
	Ashes	tt1674769	Ashes	95.0	Horror,Sci-Fi,Thriller	
	Dark Metropolis	tt1825735	Dark Metropolis	97.0	Fantasy,Sci-Fi	
	Defcon 2012	tt1349646	Defcon 2012	92.0	Sci-Fi	
	Denizen	tt1194424	Denizen	83.0	Action,Horror,Sci-Fi	

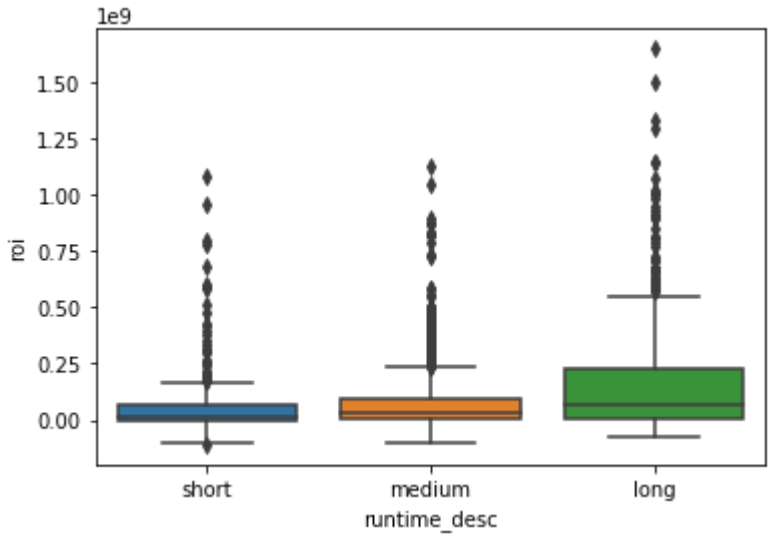
group by runtime_minutes

In [50]:
```
analysis_df_runtime_grouped = analysis_df_1mil.copy()
```

In [51]:
```
analysis_df_runtime_grouped['runtime_desc'] = analysis_df_runtime_grouped.ap
    lambda x: 'short' if x['runtime_minutes'] <= 95.0 else(
        'medium' if x['runtime_minutes'] <= 118.0 else 'long'), axis = 1
)
```

In [52]:
```
ax = sns.boxplot(x=analysis_df_runtime_grouped['runtime_desc'], y=analysis_d


```



Looking for trends in Adventure/Sci-Fi

In [53]:
```
analysis_df_1mil.loc[((analysis_df_1mil['genre3'] == 'Sci-Fi') |
                      (analysis_df_1mil['genre2'] == 'Sci-Fi') |
                      (analysis_df_1mil['genre1'] == 'Sci-Fi') )&(
                        (analysis_df_1mil['genre3'] == 'Adventure') |
                        (analysis_df_1mil['genre2'] == 'Adventure') |
                        (analysis_df_1mil['genre1'] == 'Adventure'))
                     ].sort_values(by = 'roi', ascending=False).reset_index().st
                    lambda x: ['background: #00a4ef' if x.title == 'Inception' o
```

Out[53]:

	index	year	title	runtime_minutes	genres	genre1	genre2	genre
0	1392	2018	Avengers: Infinity War	149	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
1	949	2015	Jurassic World	124	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
2	474	2012	The Avengers	143	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
3	1397	2018	Black Panther	134	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
4	1439	2018	Jurassic World: Fallen Kingdom	128	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
5	890	2015	Avengers: Age of Ultron	141	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
6	581	2013	Iron Man 3	130	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
7	344	2011	Transformers: Dark of the Moon	154	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
8	1086	2016	Captain America: Civil War	147	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
9	865	2014	Transformers: Age of Extinction	165	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
10	1179	2016	Rogue One: A Star Wars Story	133	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
11	664	2013	The Hunger Games: Catching Fire	146	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
12	1325	2017	Spider-Man: Homecoming	133	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
13	67	2010	Inception	148	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
14	840	2014	The Hunger Games: Mockingjay - Part 1	123	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
15	729	2014	Captain America: The Winter Soldier	136	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
16	472	2012	The Amazing Spider-Man	136	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F

	index	year	title	runtime_minutes	genres	genre1	genre2	genre3
17	877	2014	X-Men: Days of Future Past	132	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
18	1031	2015	The Martian	144	Adventure,Drama,Sci-Fi	Adventure	Drama	Sci-F
19	770	2014	Interstellar	169	Adventure,Drama,Sci-Fi	Adventure	Drama	Sci-F
20	1024	2015	The Hunger Games: Mockingjay - Part 2	137	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
21	69	2010	Iron Man 2	124	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
22	825	2014	The Amazing Spider-Man 2	142	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
23	592	2013	Man of Steel	143	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
24	1464	2018	Ready Player One	140	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
25	755	2014	Godzilla	123	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
26	1235	2016	X-Men: Apocalypse	144	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
27	1368	2017	Transformers: The Last Knight	154	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
28	1403	2018	Bumblebee	114	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
29	1463	2018	Rampage	107	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
30	681	2013	The Wolverine	126	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
31	637	2013	Star Trek Into Darkness	132	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
32	446	2012	Prometheus	124	Adventure,Mystery,Sci-Fi	Adventure	Mystery	Sci-F
33	1126	2016	Independence Day: Resurgence	120	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
34	566	2013	G.I. Joe: Retaliation	110	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
35	604	2013	Pacific Rim	131	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
36	489	2012	The Hunger Games	142	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
37	359	2011	X-Men: First Class	131	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
38	955	2015	Mad Max: Fury Road	120	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
39	598	2013	Oblivion	124	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
40	1188	2016	Star Trek Beyond	122	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F

	index	year	title	runtime_minutes	genres	genre1	genre2	genre3
41	208	2011	Captain America: The First Avenger	124	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
42	1075	2016	Assassin's Creed	115	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
43	1457	2018	Pacific Rim: Uprising	111	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
44	241	2011	I Am Number Four	111	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
45	105	2010	Predators	107	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
46	375	2012	Battleship	131	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
47	1494	2018	The Predator	107	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
48	627	2013	Riddick	119	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
49	275	2011	Paul	104	Adventure,Comedy,Sci-Fi	Adventure	Comedy	Sci-F
50	1316	2017	Power Rangers	124	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
51	745	2014	Earth to Echo	91	Adventure,Family,Sci-Fi	Adventure	Family	Sci-F
52	232	2011	Green Lantern	114	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
53	416	2012	John Carter	132	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
54	579	2013	Independence Daysaster	90	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F
55	667	2013	The Last Days on Mars	98	Adventure,Horror,Sci-Fi	Adventure	Horror	Sci-F
56	948	2015	Jupiter Ascending	127	Action,Adventure,Sci-Fi	Action	Adventure	Sci-F

Christopher Nolan movies are the only non-lisenced films on the list until number 49 - Paul

In [ ]:
```
1
```