GitHub Dataset

姓名：廖嘉琦

学号：1120200733

Step 0 导入相关库

包括 pandas，matplotlib，seaborn，并设置 matplotlib 的中文字体。

```python
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

Step 1 读入数据

从 repository_data.csv 中读入数据，解析后存入 pandas 的 DataFrame 中。

```python
repository_df = pd.read_csv('repository_data.csv')
print(repository_df.shape)
repository_df.head()
```

```
(2917951, 10)

                          name  stars_count  forks_count  watchers  \
0                  freeCodeCamp       359805        30814      8448
1                       996.ICU       264811        21470      4298
2         free-programming-books       262380        53302      9544
3   coding-interview-university       244927        65038      8539
4                       awesome       235223        24791      7446

   pull_requests primary_language  \
0          31867       TypeScript
1           1949              NaN
2           8235              NaN
3            867              NaN
4           1859              NaN

                                      languages_used  commit_count  \
0  ['TypeScript', 'JavaScript', 'CSS', 'Shell', '...      32231.00
1                                                NaN       3189.00
2                                                NaN       8286.00
3                                                NaN       2314.00
4                                                NaN       1074.00

             created_at                                           licence
0  2014-12-24T17:49:19Z          BSD 3-Clause "New" or "Revised" License
1  2019-03-26T07:31:14Z                                             Other
2  2013-10-11T06:50:37Z                                             Other
3  2016-06-06T02:34:12Z  Creative Commons Attribution Share Alike 4.0 I...
4  2014-07-11T13:42:37Z                Creative Commons Zero v1.0 Universal
```

Step 2 数据预处理

检查异常数据，发现部分字段存在缺失值，对缺失值进行处理。

去除掉数值属性的缺失值，对于 licence 字段的缺失值，采用填充的方法，填充为 Unknown；对于 language 字段的缺失值，填充为 None，同理 languages_used 字段的缺失值填充为 ['None']。

```python
import ast
```

```python
for column in repository_df.columns:
    cnt_na = repository_df[column].isna().sum()
    if cnt_na > 0:
        print(f'{column} has {cnt_na} missing values')

repository_df.dropna(subset=['name', 'stars_count', 'forks_count',
                     'watchers', 'commit_count', 'created_at'], inplace=True)
repository_df['primary_language'].fillna('None', inplace=True)
repository_df['languages_used'].fillna("['None']", inplace=True)
repository_df['licence'].fillna('Unknown', inplace=True)
repository_df['commit_count'] = repository_df['commit_count'].astype(int)
repository_df['created_at'] = pd.to_datetime(repository_df['created_at'])
repository_df['licence'] = repository_df['licence'].astype(str)
repository_df['languages_used'] = repository_df['languages_used'].apply(ast.literal_eval)
repository_df['primary_language'] = repository_df['primary_language'].astype(str)
print(repository_df.shape)
repository_df.head()
```

```
name has 13 missing values
primary_language has 218573 missing values
languages_used has 221984 missing values
commit_count has 1921 missing values
licence has 1378200 missing values
(2916017, 10)

                          name  stars_count  forks_count  watchers  \
0                 freeCodeCamp       359805        30814      8448
1                     996.ICU       264811        21470      4298
2          free-programming-books       262380        53302      9544
3  coding-interview-university       244927        65038      8539
4                      awesome       235223        24791      7446

   pull_requests primary_language  \
0          31867       TypeScript
1           1949             None
2           8235             None
3            867             None
4           1859             None

                                      languages_used  commit_count  \
0  [TypeScript, JavaScript, CSS, Shell, Dockerfil...         32231
1                                             [None]          3189
2                                             [None]          8286
3                                             [None]          2314
4                                             [None]          1074

                 created_at                                         licence
0 2014-12-24 17:49:19+00:00            BSD 3-Clause "New" or "Revised" License
1 2019-03-26 07:31:14+00:00                                            Other
2 2013-10-11 06:50:37+00:00                                            Other
3 2016-06-06 02:34:12+00:00  Creative Commons Attribution Share Alike 4.0 I...
4 2014-07-11 13:42:37+00:00                Creative Commons Zero v1.0 Universal
```

删除缺失值后，仍有 99.93% 的数据保留，因此对后续分析几乎不会产生影响。

Step 3 数据分析

1. 数值型数据的描述性统计

```python
pd.set_option('display.float_format', '{:.2f}'.format)
repository_df.describe()
```

```
       stars_count  forks_count   watchers  pull_requests  commit_count
```

```
count   2916017.00   2916017.00 2916017.00     2916017.00     2916017.00
mean         76.45        20.96       7.14          24.32         614.37
std         909.98       303.05      37.63         378.57       16808.04
min           2.00         0.00       0.00           0.00           1.00
25%           7.00         1.00       2.00           0.00           9.00
50%          12.00         4.00       3.00           1.00          27.00
75%          30.00        11.00       6.00           6.00          89.00
max      359805.00    242208.00    9544.00      301585.00     4314502.00
```

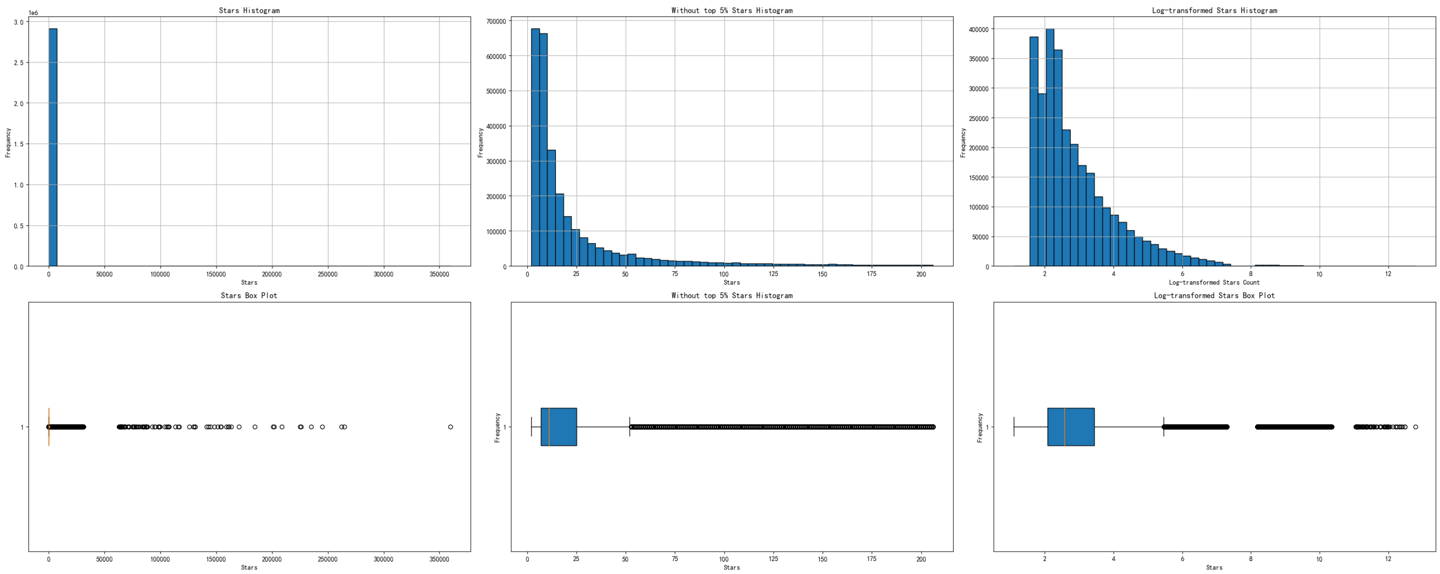
```python
columns = ['stars_count', 'forks_count',
           'watchers', 'pull_requests', 'commit_count']

for column in columns:
    avg = repository_df[column].mean()
    percentile_99 = repository_df[column].quantile(q=0.99)
    print(f'{column} has average {avg:.2f}, and 99th percentile {percentile_99:.2f}')
```

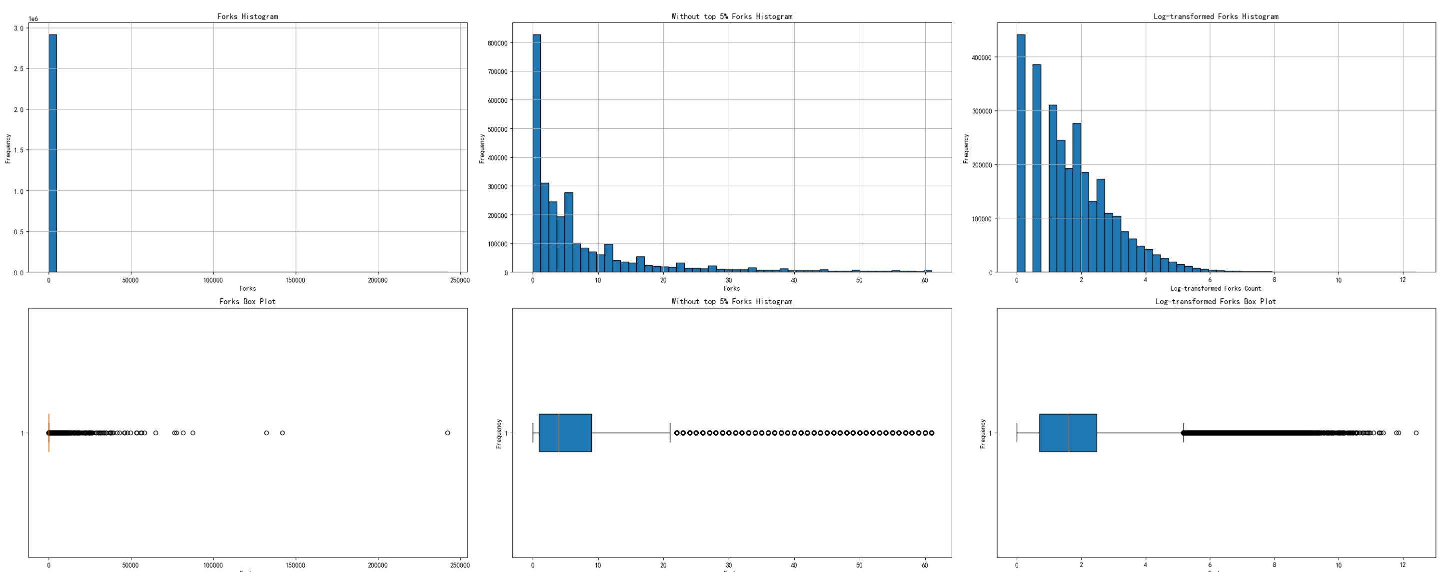
```
stars_count has average 76.45, and 99th percentile 823.00
forks_count has average 20.96, and 99th percentile 233.00
watchers has average 7.14, and 99th percentile 67.00
pull_requests has average 24.32, and 99th percentile 400.00
commit_count has average 614.37, and 99th percentile 2927.00
```

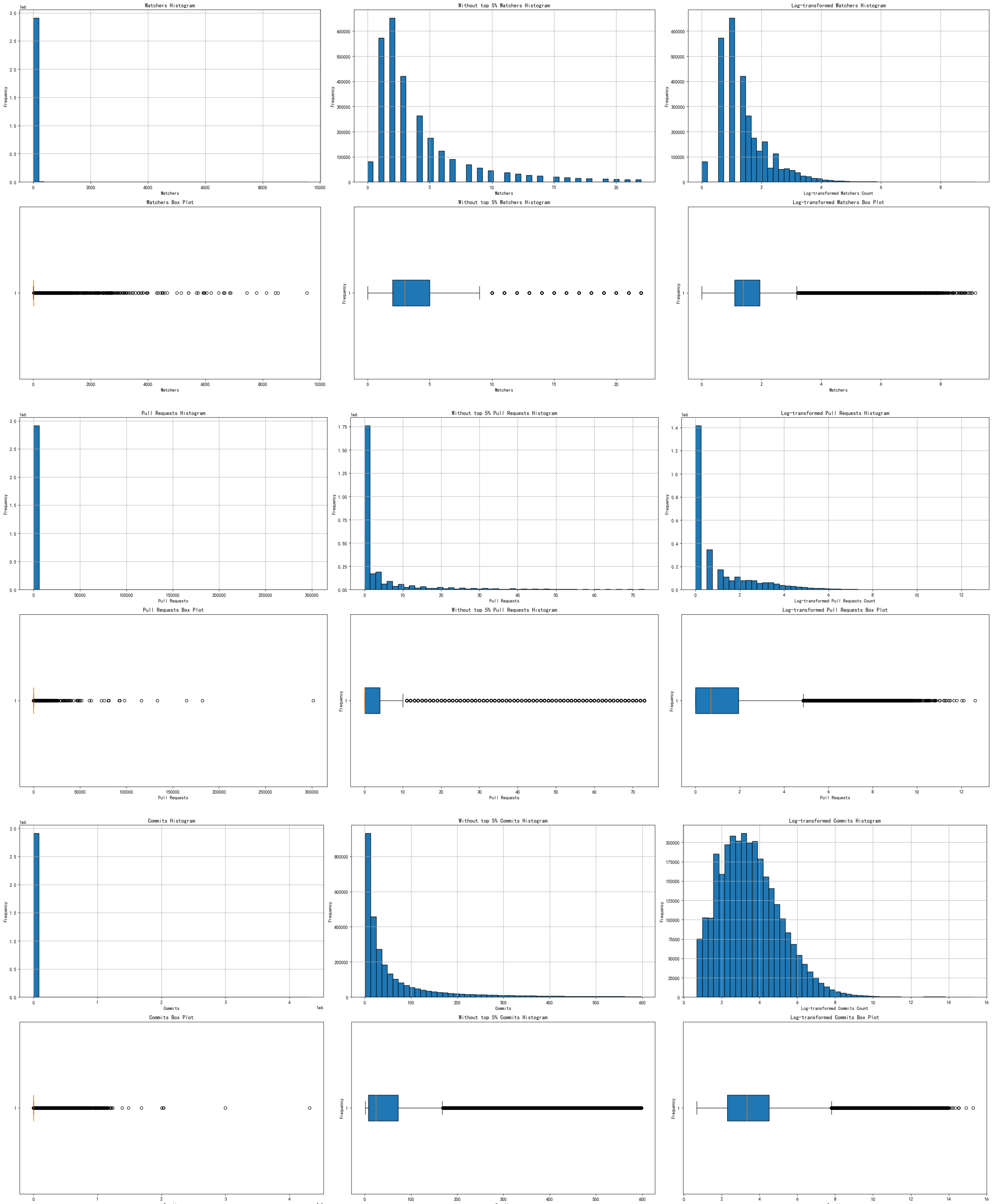
不难发现，数据中存在极少数的巨大值，因此在绘制图时，为了方便观察，选择了三种方式：所有数据、去除最大的 10% 后的数据、log 变换后的数据。

```python
def draw(column, name):
    plt.figure(figsize=(30, 12))
    plt.subplot(2, 3, 1)
    repository_df[column].hist(bins=50, edgecolor='k')
    plt.title(f'{name} Histogram')
    plt.xlabel(name)
    plt.ylabel('Frequency')

    plt.subplot(2, 3, 2)
    percentile_95 = repository_df[column].quantile(0.95)
    repository_df[repository_df[column] <= percentile_95][column].hist(
        bins=50, edgecolor='k')
    plt.title(f'Without top 5% {name} Histogram')
    plt.xlabel(name)
    plt.ylabel('Frequency')

    # 绘制对数变换后的直方图
    plt.subplot(2, 3, 3)
    np.log1p(repository_df[column]).hist(
        bins=50, edgecolor='k')
    plt.title(f'Log-transformed {name} Histogram')
    plt.xlabel(f'Log-transformed {name} Count')
    plt.ylabel('Frequency')

    plt.subplot(2, 3, 4)
    plt.boxplot(repository_df[column], vert=False, patch_artist=True)
    plt.title(f'{name} Box Plot')
    plt.xlabel(name)

    plt.subplot(2, 3, 5)
    plt.boxplot(repository_df[repository_df[column] <= percentile_95][column], vert=False, patch_artist=True)
    plt.title(f'Without top 5% {name} Histogram')
    plt.xlabel(name)
    plt.ylabel('Frequency')

    plt.subplot(2, 3, 6)
```

```
    plt.boxplot(np.log1p(repository_df[column]),
                vert=False, patch_artist=True)
    plt.title(f'Log-transformed {name} Box Plot')
    plt.xlabel(name)
    plt.ylabel('Frequency')
    plt.tight_layout()
    plt.show();

names = ['Stars', 'Forks', 'Watchers', 'Pull Requests', 'Commits']

for column, name in zip(columns, names):
    draw(column, name)
```





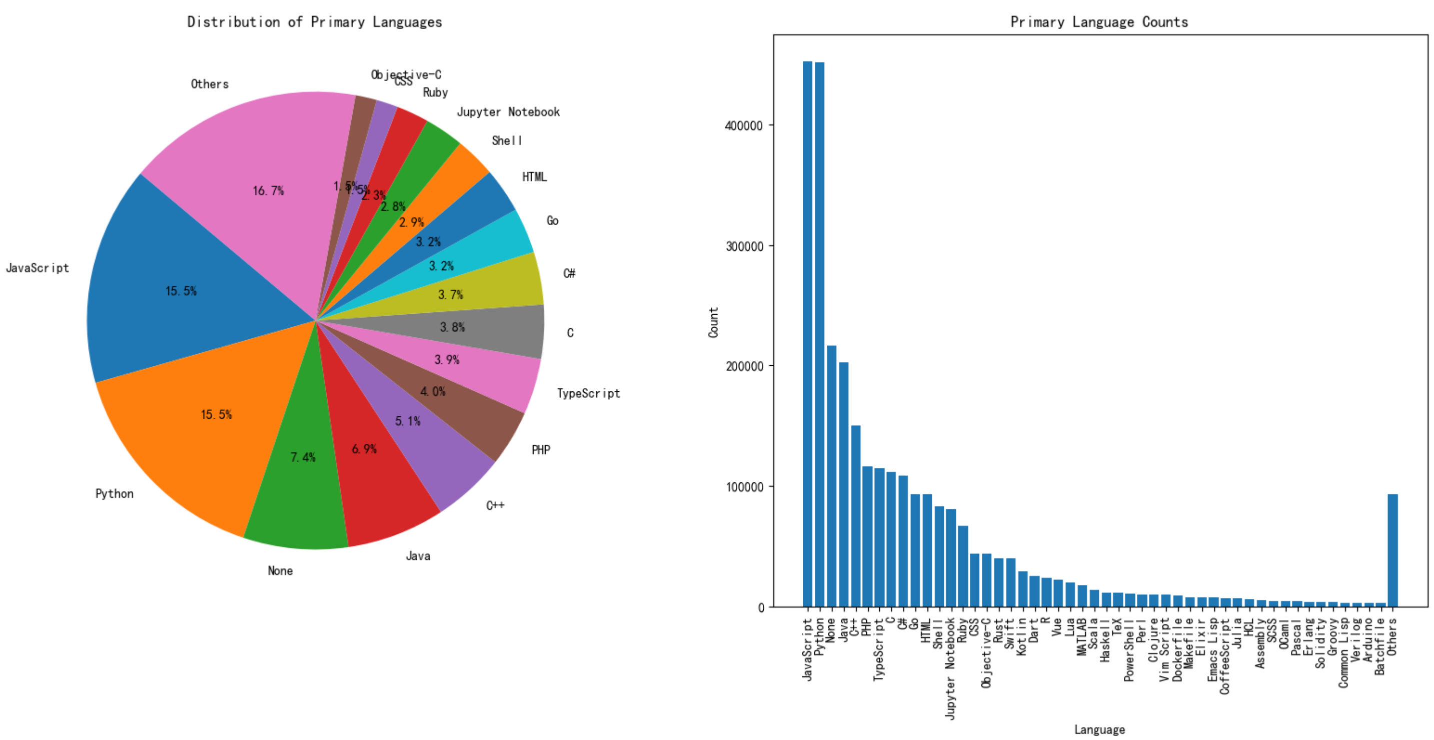


接下来，分析使用的首要语言和使用语言的分布情况。

```
# 计算每种语言出现的次数
language_counts = repository_df['primary_language'].value_counts()
```

```python
# 定义阈值，比如所有占总数小于 1.5% 的语言将被归入"Others"
threshold_percent = 1.5
threshold = threshold_percent / 100 * language_counts.sum()

# 将小于阈值的语言归类为"Others"
filtered_languages = language_counts[language_counts > threshold]
others_count = language_counts[language_counts <= threshold].sum()
if others_count > 0:
    filtered_languages['Others'] = others_count

plt.figure(figsize=(20, 8))
plt.subplot(1, 2, 1)
plt.pie(filtered_languages, labels=filtered_languages.index,
        autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Primary Languages');

plt.subplot(1, 2, 2)
# 定义阈值，比如所有占总数小于 1.5% 的语言将被归入"Others"
threshold_percent = 0.1
threshold = threshold_percent / 100 * language_counts.sum()

# 将小于阈值的语言归类为"Others"
filtered_languages = language_counts[language_counts > threshold]
others_count = language_counts[language_counts <= threshold].sum()
if others_count > 0:
    filtered_languages['Others'] = others_count
plt.bar(filtered_languages.index, filtered_languages)
plt.title('Primary Language Counts')
plt.xlabel('Language')
plt.ylabel('Count')
plt.xticks(rotation=90);
print(f'首要语言: 共有 {len(language_counts)} 种语言, 其中占比超过 {threshold_percent}% 的有 {len(filtered_languag
```

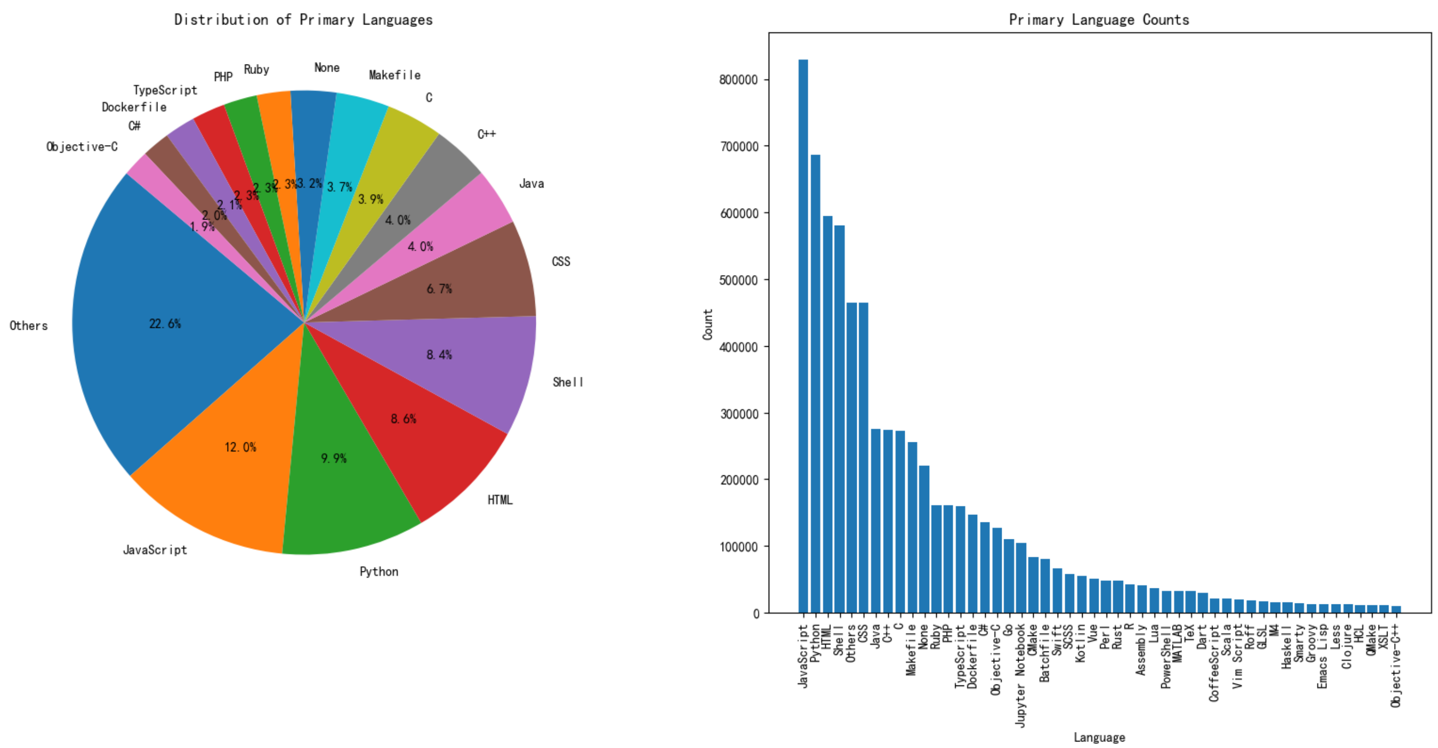
```
首要语言: 共有 498 种语言, 其中占比超过 0.1% 的有 50 种
```




```python
from collections import Counter
```

```
def filter(threshold_percent, df):
    threshold = threshold_percent / 100 * df['Count'].sum()
    filtered_data = df[language_counts_df['Count'] > threshold].copy()
    others_count = df[language_counts_df['Count'] <= threshold]['Count'].sum()
    filtered_data.loc[len(filtered_data)] = ['Others', others_count]
    filtered_data = filtered_data.sort_values(
        'Count', ascending=False)
    return filtered_data

language_counts = Counter(
    [lang for sublist in repository_df['languages_used'] for lang in sublist])

language_counts_df = pd.DataFrame(language_counts.items(), columns=['Language', 'Count'])

plt.figure(figsize=(20, 8))
plt.subplot(1, 2, 1)
filtered_languages = filter(1.5, language_counts_df)
plt.pie(filtered_languages['Count'], labels=filtered_languages['Language'],
        autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Primary Languages')

plt.subplot(1, 2, 2)
filtered_languages = filter(0.15, language_counts_df)
plt.bar(filtered_languages['Language'], filtered_languages['Count'])
plt.title('Primary Language Counts')
plt.xlabel('Language')
plt.ylabel('Count')
plt.xticks(rotation=90)
print(f'所有语言：共有 {len(language_counts)} 种语言，其中占比超过 {threshold_percent}% 的有 {len(filtered_languag
```

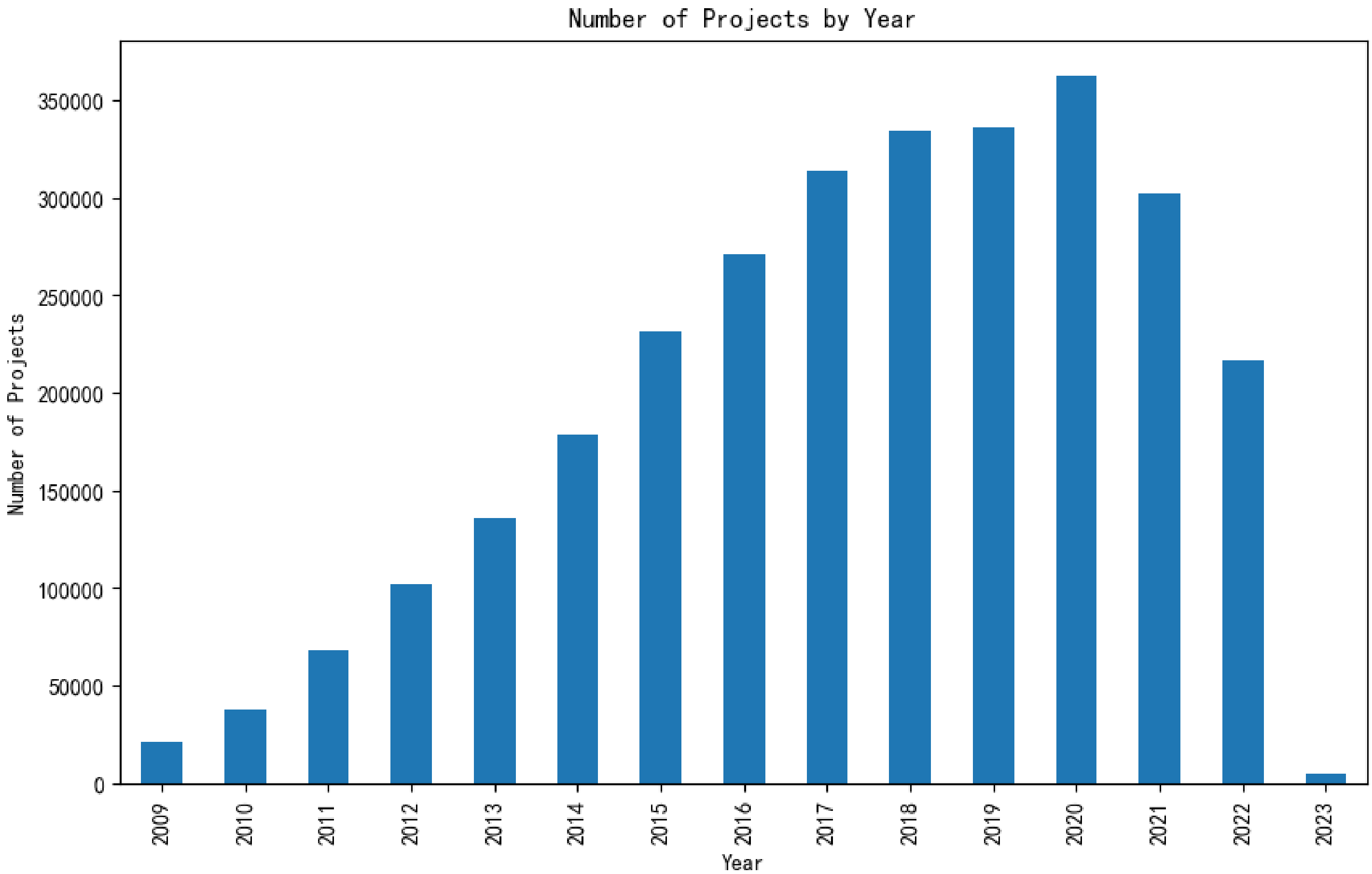
```
所有语言：共有 529 种语言，其中占比超过 0.1% 的有 50 种
```



再分析每年的新增仓库数的变化情况。

```
repository_df['year'] = repository_df['created_at'].dt.year
yearly_counts = repository_df.groupby('year').size()
plt.figure(figsize=(10, 6))
```

```python
yearly_counts.plot(kind='bar')
plt.title('Number of Projects by Year')
plt.xlabel('Year')
plt.ylabel('Number of Projects')
plt.show()
```



最后分析数据之间的相关性。

```python
def calculate_correlation(df, x, y):
    pearson_corr = df[x].corr(df[y], method='pearson')
    print(f"{x}与{y}的皮尔森相关系数: {pearson_corr:.2f}")

for i in range(len(columns)):
    for j in range(i + 1, len(columns)):
        calculate_correlation(repository_df, columns[i], columns[j])
```

```
stars_count与forks_count的皮尔森相关系数: 0.57
stars_count与watchers的皮尔森相关系数: 0.71
stars_count与pull_requests的皮尔森相关系数: 0.19
stars_count与commit_count的皮尔森相关系数: 0.02
forks_count与watchers的皮尔森相关系数: 0.49
forks_count与pull_requests的皮尔森相关系数: 0.21
forks_count与commit_count的皮尔森相关系数: 0.02
watchers与pull_requests的皮尔森相关系数: 0.16
watchers与commit_count的皮尔森相关系数: 0.02
pull_requests与commit_count的皮尔森相关系数: 0.05
```

皮尔森相关系数表明，stars_count 与 watchers 的线性关系较强，而 watchers 与 commit_count 几乎没有线性关系。

MovieLens 10M Dataset

姓名：廖嘉琦

学号：1120200733

Step 0 导入相关库

包括 pandas，matplotlib，seaborn，并设置 matplotlib 的中文字体。

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

Step 1 读入数据

从 *.dat 中读入数据，解析后存入对应的 pandas 的 DataFrame 中。

1. 读入 movies.dat，包括 MovieID::Title::Genres，将 Genres 拆分为多列，并从 Title 中解析出 Year。

```
dtype_spec = {
    'MovieID': int,
    'Title': str
}

# MovieID::Title::Genres
movies_df = pd.read_csv('movies.dat', sep='::', engine='python', header=None,
                        names=['MovieID', 'Title', 'Genres'],
                        dtype=dtype_spec)

# split Genres by '|'
movies_df['Genres'] = movies_df['Genres'].apply(lambda x: x.split('|'))
movies_df['Year'] = movies_df['Title'].apply(
    lambda x: int(x[-5:-1]) if x[-5:-1].isdigit() else -1)
movies_df['Title'] = movies_df['Title'].apply(
    lambda s: s[:-7])
print(movies_df.head())
```

```
   MovieID                        Title  \
0        1                    Toy Story
1        2                      Jumanji
2        3             Grumpier Old Men
3        4            Waiting to Exhale
4        5  Father of the Bride Part II

                                          Genres  Year
0  [Adventure, Animation, Children, Comedy, Fantasy]  1995
1                     [Adventure, Children, Fantasy]  1995
2                              [Comedy, Romance]  1995
3                       [Comedy, Drama, Romance]  1995
4                                       [Comedy]  1995
```

1. 读入 ratings.dat，包括 UserID::MovieID::Rating::Timestamp，将 Timestamp 转换为 Datetime 格式。

```
dtype_spec = {
    'UserID': int,
    'MovieID': int,
    'Rating': float
}

# UserID::MovieID::Rating::Timestamp
ratings_df = pd.read_csv('ratings.dat', sep='::', engine='python',
```

```
                            names=['UserID', 'MovieID', 'Rating', 'Timestamp'],
                            dtype=dtype_spec)

# convert Timestamp to datetime
ratings_df['Timestamp'] = pd.to_datetime(ratings_df['Timestamp'], unit='s')

print(ratings_df.head())
```

```
   UserID  MovieID  Rating           Timestamp
0       1      122     5.0 1996-08-02 11:24:06
1       1      185     5.0 1996-08-02 10:58:45
2       1      231     5.0 1996-08-02 10:56:32
3       1      292     5.0 1996-08-02 10:57:01
4       1      316     5.0 1996-08-02 10:56:32
```

```
dtype_spec = {
    'UserID': int,
    'MovieID': int,
    'Tag': str
}

# UserID::MovieID::Tag::Timestamp
tags_df = pd.read_csv('tags.dat', sep='::', engine='python',
                            names=['UserID', 'MovieID', 'Tag', 'Timestamp'],
                            dtype=dtype_spec)

# convert Timestamp to datetime
tags_df['Timestamp'] = pd.to_datetime(tags_df['Timestamp'], unit='s')

print(tags_df.head())
```

```
   UserID  MovieID              Tag           Timestamp
0      15     4973       excellent! 2008-07-04 15:17:10
1      20     1747         politics 2007-08-28 01:17:47
2      20     1747           satire 2007-08-28 01:17:47
3      20     2424  chick flick 212 2007-08-28 01:17:15
4      20     2424            hanks 2007-08-28 01:17:15
```

```
print('Movie: Num =', len(movies_df))
print('Ratings: Num =', len(ratings_df))
print('Tags: Num =', len(tags_df))
```

```
Movie: Num = 10681
Ratings: Num = 10000054
Tags: Num = 95580
```

此时，得到三个 DataFrame：

- movies_df：电影数据，共 10681 部。
- ratings_df：评分数据，共 10000054 条。
- tags_df：标签数据，共 95580 条。

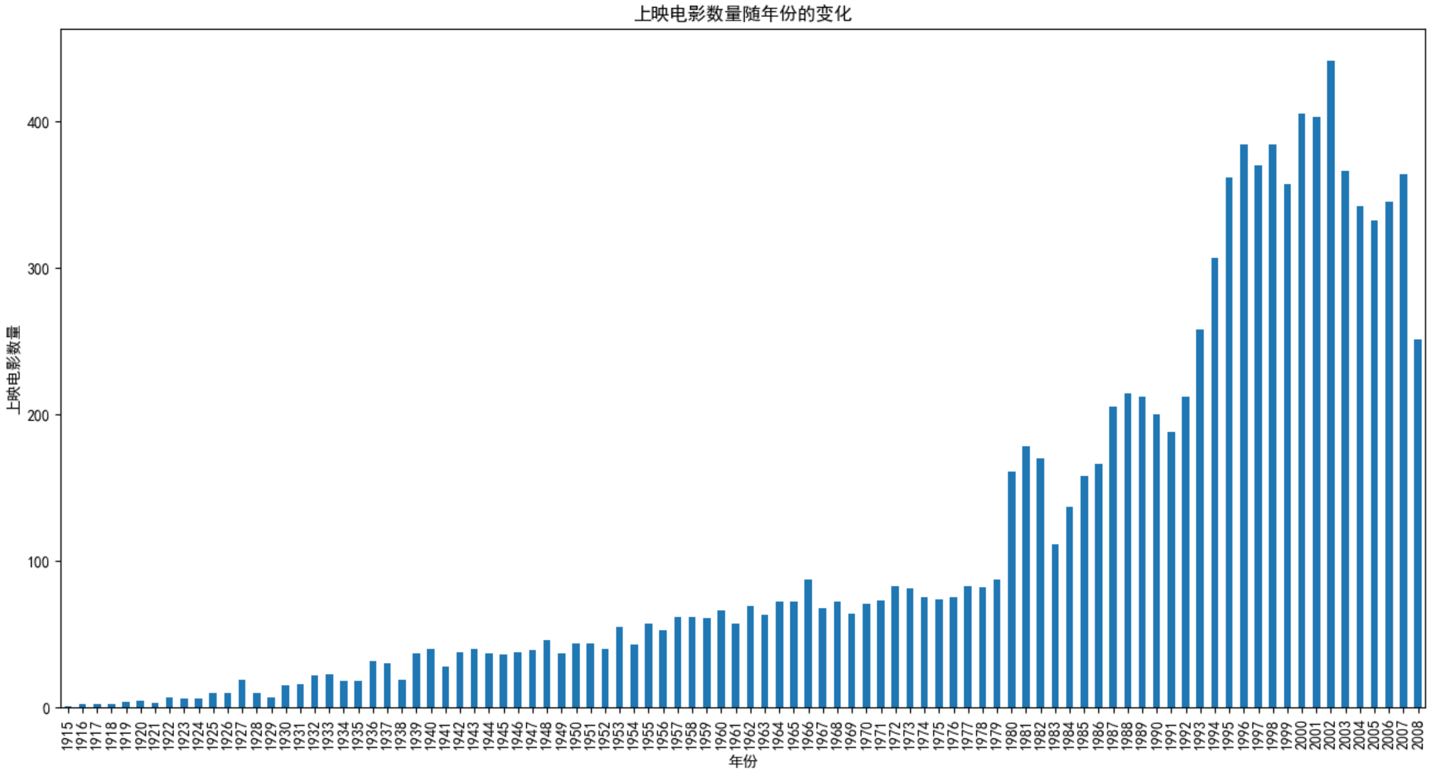
Step 3 分析电影数据

1. 统计电影的上映年份分布
2. 统计电影不同类型的比例
3. 剔除不合法的类型数据

```
print('异常年份的电影数量:', movies_df[movies_df['Year'] == -1].shape[0])
plt.figure(figsize=(16, 8))
movies_df[movies_df['Year'] != -1]['Year'].value_counts().sort_index().plot(kind='bar')
plt.xticks(rotation=90)
plt.xlabel('年份')
```

```python
plt.ylabel('上映电影数量')
plt.title('上映电影数量随年份的变化');
```

```
异常年份的电影数量: 0
```



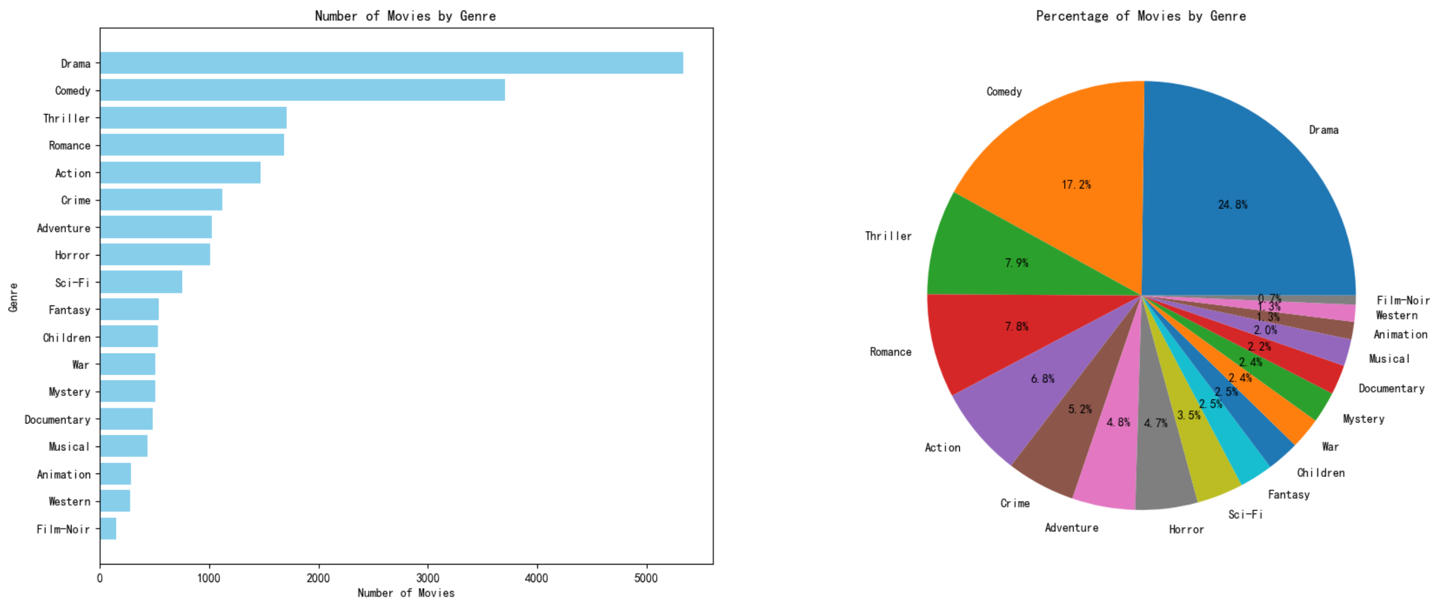

```python
genres = {'Action', 'Adventure', 'Animation', 'Children', 'Comedy', 'Crime', 'Documentary',
          'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi',
          'Thriller', 'War', 'Western'}

movies_df['NonStandardGenres'] = movies_df['Genres'].apply(
    lambda x: [genre for genre in x if genre not in genres])

non_standard_genres_movies = movies_df[movies_df['NonStandardGenres'].apply(
    len) > 0].drop('Genres', axis=1)
print(non_standard_genres_movies.head())
print('去除异常分类:', len(non_standard_genres_movies))
movies_df = movies_df.assign(Genres=lambda df: df['Genres'].apply(lambda x: [
                             genre for genre in x if genre in genres])).drop('NonStandardGenres', axis=1)
```

```
      MovieID                   Title  Year NonStandardGenres
32         33        Wings of Courage  1995            [IMAX]
36         37  Across the Sea of Time  1995            [IMAX]
1724     1797                 Everest  1998            [IMAX]
3074     3159           Fantasia 2000  1999            [IMAX]
4289     4382                  Wolves  1999            [IMAX]
去除异常分类: 30
```

接下来分析不同分类的电影的数量和占比。如果同一个电影有多个分类，那么会统计多次。

```python
from collections import Counter
genre_counts = Counter([genre for genres in movies_df['Genres']
                       for genre in genres])
genre_counts_df = pd.DataFrame(list(genre_counts.items()), columns=[
                               'Genre', 'Count']).sort_values('Count', ascending=False)

plt.figure(figsize=(20, 8))
```

```python
plt.subplot(1, 2, 1)
plt.barh(genre_counts_df['Genre'], genre_counts_df['Count'], color='skyblue')
plt.xlabel('Number of Movies')
plt.ylabel('Genre')
plt.title('Number of Movies by Genre')
plt.gca().invert_yaxis()

plt.subplot(1, 2, 2)
plt.pie(genre_counts_df['Count'],
        labels=genre_counts_df['Genre'], autopct='%1.1f%%')
plt.title('Percentage of Movies by Genre');
```



Step 4 分析评分数据

先统计评分数量的时间分布。

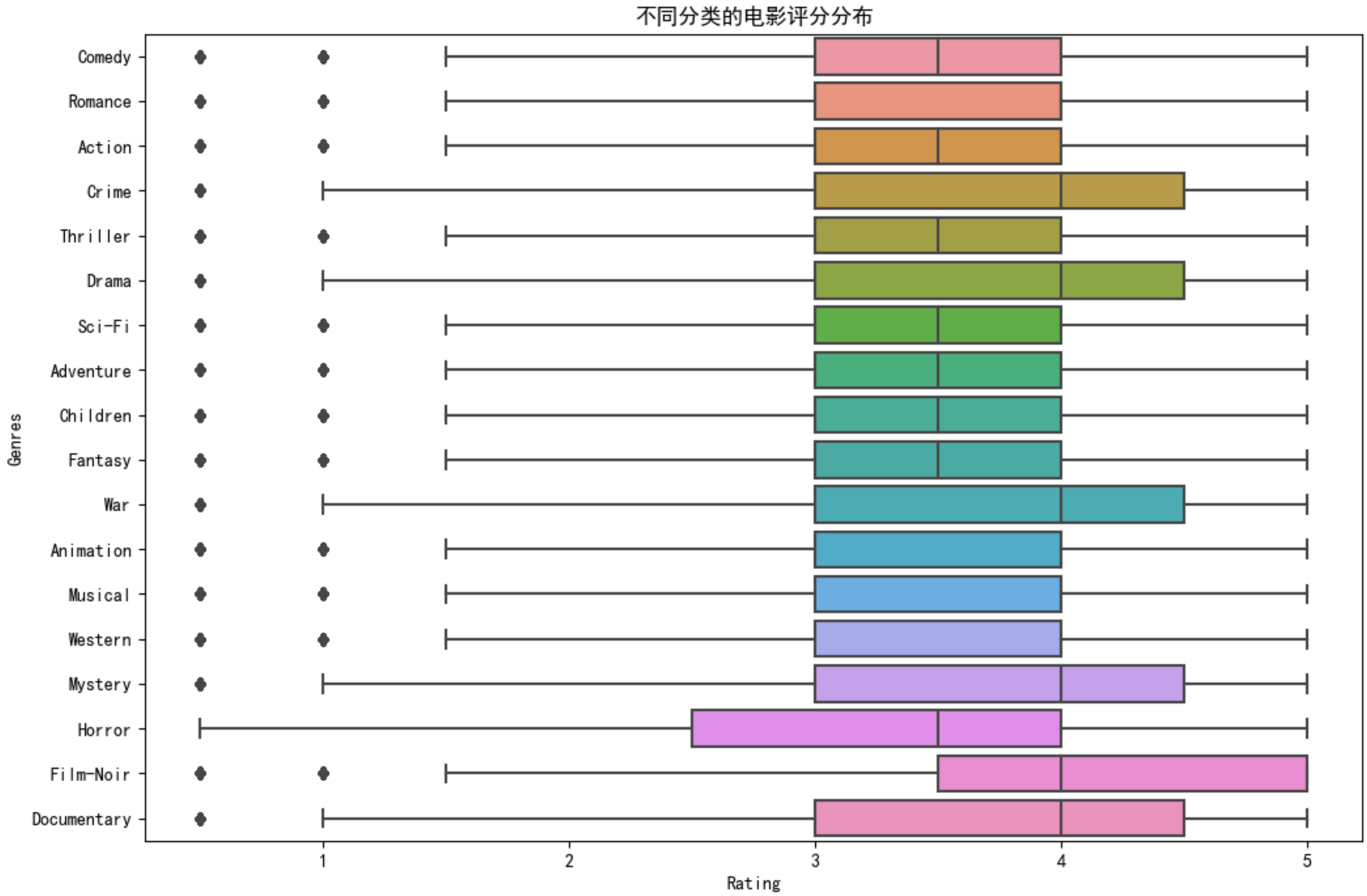
```python
ratings_df['Year'] = pd.to_datetime(ratings_df['Timestamp'], unit='s').dt.year
yearly_review_count = ratings_df.groupby(
    'Year')['Rating'].count().reset_index()
print(yearly_review_count)
ratings_df = ratings_df[~ratings_df['Year'].isin([1995, 2009])]
```

```
    Year   Rating
0   1995        3
1   1996  1047618
2   1997   459947
3   1998   202092
4   1999   788793
5   2000  1271623
6   2001   759141
7   2002   583409
8   2003   688694
9   2004   768168
10  2005  1177283
11  2006   765733
12  2007   699325
13  2008   773617
14  2009    14608
```

发现 1995 年和 2009 年的评分数据显著少于其他年份，可能是数据缺失，因此剔除这两年的数据。

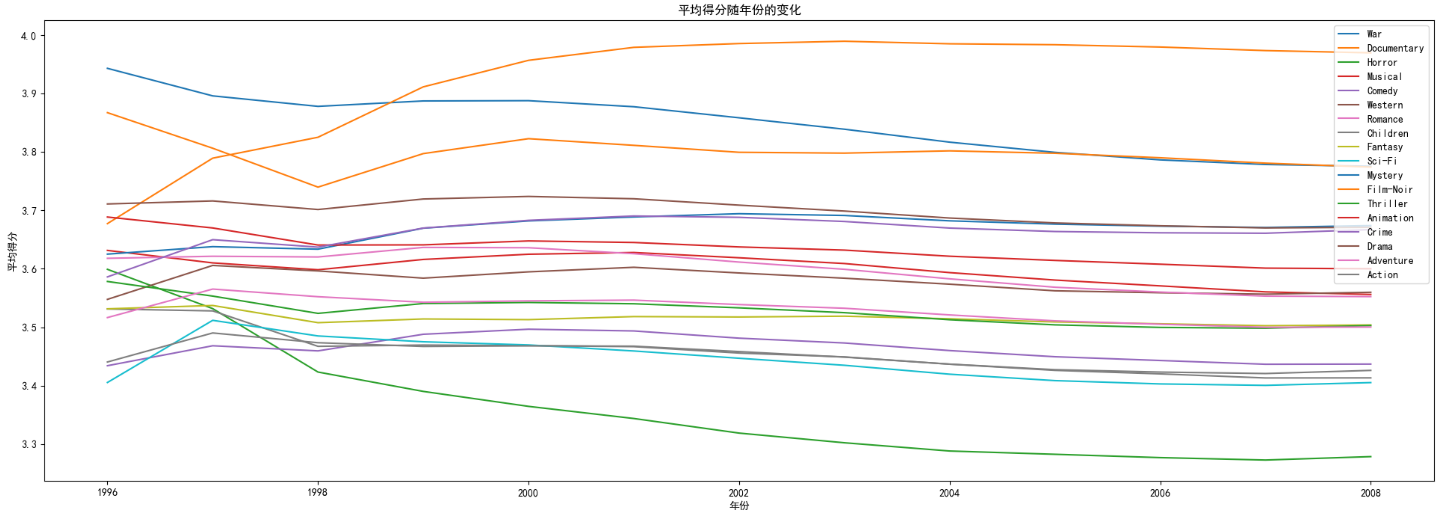
接下来统计不同分类的电影的评分和评价数量，以及其随着时间的变化。

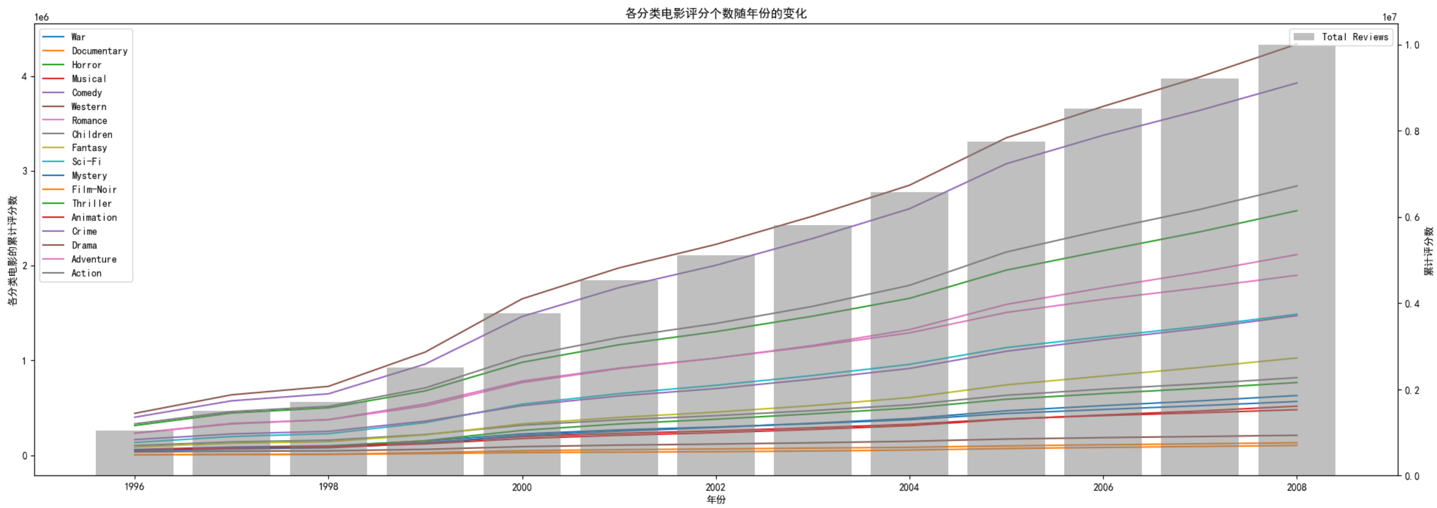
```
movies_df_exploded = movies_df.explode('Genres')
merged_df = pd.merge(ratings_df, movies_df_exploded, on='MovieID')

plt.figure(figsize=(12, 8))
sns.boxplot(x='Rating', y='Genres', data=merged_df)
plt.title('不同分类的电影评分分布');
```




```
movies_df_exploded = movies_df.explode('Genres')
merged_df = pd.merge(ratings_df, movies_df_exploded, on='MovieID')

average_ratings_per_year = merged_df.groupby(
    ['Genres', 'Year_x'])['Rating'].mean().reset_index()

average_ratings_per_year['Cumulative Average Rating'] = average_ratings_per_year.groupby(
    'Genres')['Rating'].apply(lambda x: x.expanding().mean()).reset_index(level=0, drop=True)

average_ratings_per_year['Yearly Review Count'] = merged_df.groupby(
    ['Genres', 'Year_x'])['Rating'].count().reset_index(drop=True)

average_ratings_per_year['Cumulative Review Count'] = average_ratings_per_year.groupby(
    'Genres')['Yearly Review Count'].cumsum()

average_ratings_per_year = average_ratings_per_year[[
    'Genres', 'Year_x', 'Rating', 'Cumulative Average Rating', 'Yearly Review Count', 'Cumulative Review Count

yearly_review_count = ratings_df.groupby(
    'Year')['Rating'].count().reset_index()

yearly_review_count['Cumulative Review Count'] = yearly_review_count['Rating'].cumsum()
```

```python
plt.figure(figsize=(24, 8))

for genre in genres:
    genre_data = average_ratings_per_year[average_ratings_per_year['Genres'] == genre]
    plt.plot(genre_data['Year_x'],
             genre_data['Cumulative Average Rating'], label=genre)

plt.xlabel('年份')
plt.ylabel('平均得分')
plt.title('平均得分随年份的变化')
plt.legend()

plt.figure(figsize=(24, 8))
ax1 = plt.gca()
for genre in genres:
    genre_data = average_ratings_per_year[average_ratings_per_year['Genres'] == genre]
    ax1.plot(genre_data['Year_x'],
             genre_data['Cumulative Review Count'], label=genre)

ax1.set_xlabel('年份')
ax1.set_ylabel('各分类电影的累计评分数')
ax1.tick_params(axis='y')
ax1.legend(loc='upper left')

ax2 = ax1.twinx()
ax2.bar(yearly_review_count['Year'], yearly_review_count['Cumulative Review Count'],
        color='grey', alpha=0.5, label='Total Reviews')
ax2.set_ylabel('累计评分数')
ax2.tick_params(axis='y')
ax2.legend(loc='upper right')

plt.title('各分类电影评分个数随年份的变化');
```



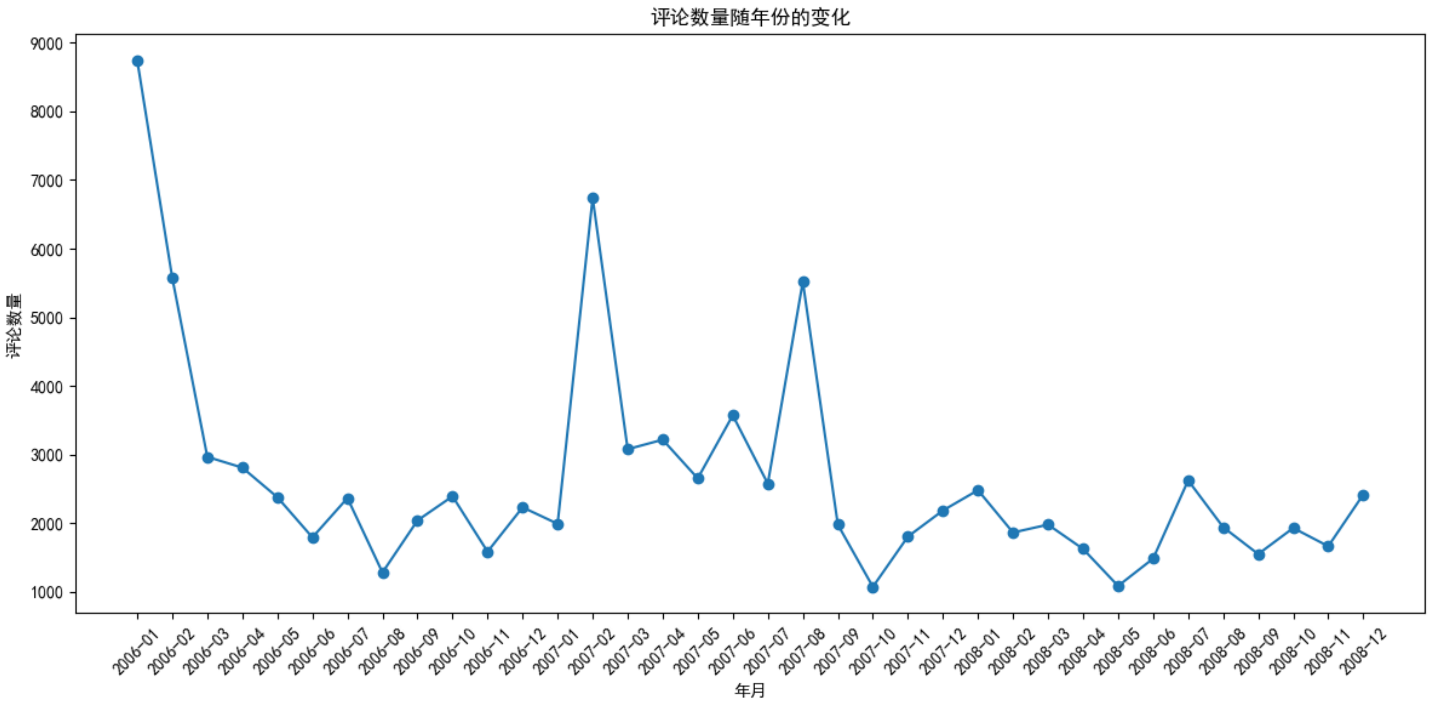


Step 5 分析标签数据

先分析标签数随着时间的变化。

```
tags_df['Year'] = pd.to_datetime(tags_df['Timestamp'], unit='s').dt.year
yearly_tag_count = tags_df.groupby(
    'Year')['Tag'].count().reset_index()
print(yearly_tag_count)
tags_df = tags_df[~tags_df['Year'].isin([2005, 2009])]
# plot
tags_df['YearMonth'] = pd.to_datetime(
    tags_df['Timestamp'], unit='s').dt.to_period('M')
yearly_monthly_tag_count = tags_df.groupby(
    'YearMonth')['Tag'].count().reset_index(name='Count')
plt.figure(figsize=(12, 6))
plt.plot(yearly_monthly_tag_count['YearMonth'].astype(
    str), yearly_monthly_tag_count['Count'], marker='o')
plt.title('评论数量随年份的变化')
plt.xlabel('年月')
plt.ylabel('评论数量')
plt.xticks(rotation=45)
plt.tight_layout();
```

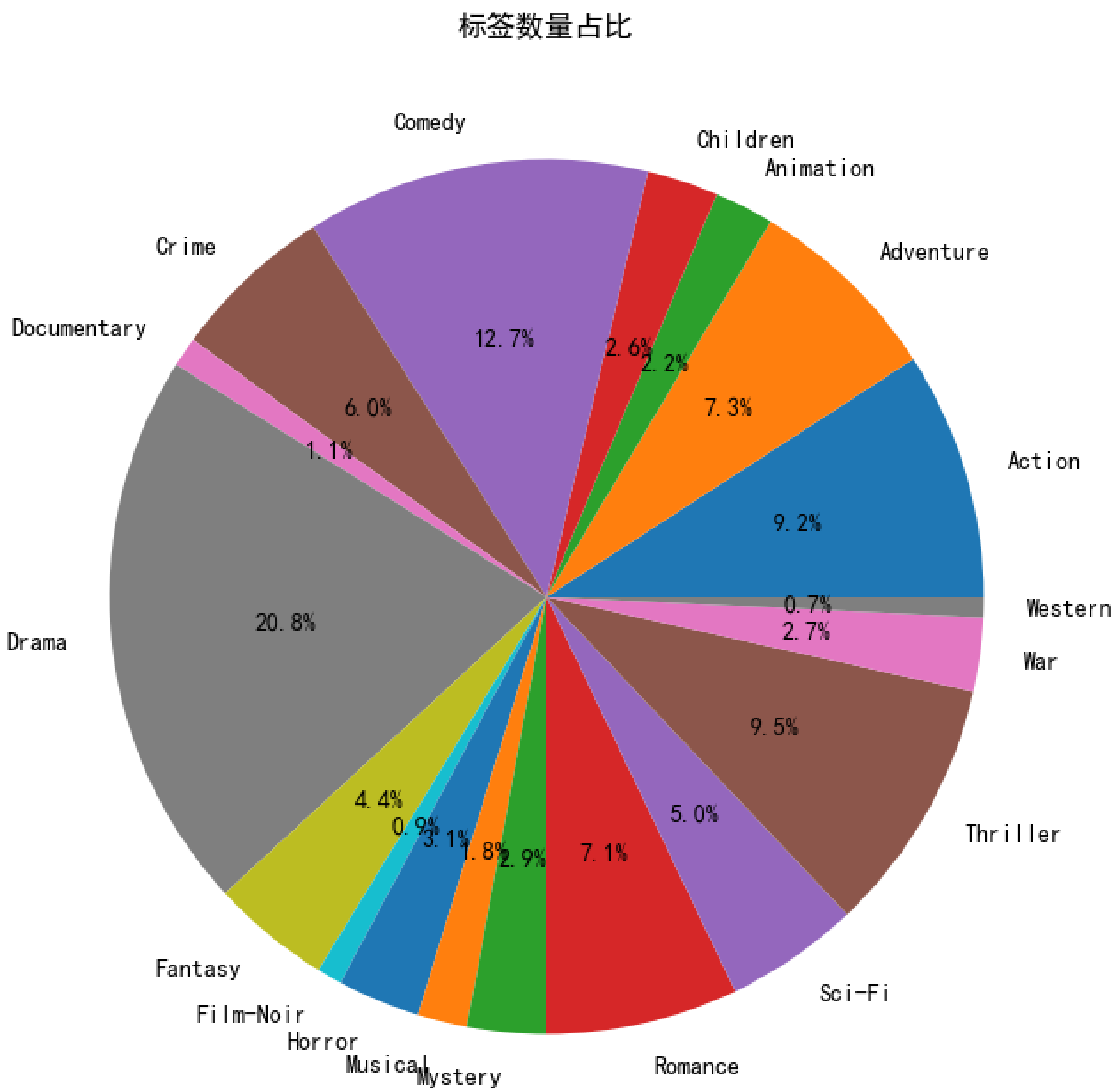
	Year	Tag
0	2005	38
1	2006	36163
2	2007	36395
3	2008	22658
4	2009	310



发现 2005 年和 2009 年只有极少数标签，属于异常数据，剔除。

再分析不同分类的电影的标签数量和占比。

```
movies_df_exploded = movies_df.explode('Genres')
merged_df = pd.merge(tags_df, movies_df_exploded, on='MovieID')

genre_tag_counts = merged_df.groupby(
    'Genres')['Tag'].count().reset_index(name='Tag Count')

total_tags = genre_tag_counts['Tag Count'].sum()

genre_tag_counts['Proportion'] = genre_tag_counts['Tag Count'] / total_tags

plt.figure(figsize=(10, 8))
plt.pie(genre_tag_counts['Proportion'], labels=genre_tag_counts['Genres'], autopct='%1.1f%%')
plt.title('标签数量占比');
```


标签数量占比
Comedy
12.7%
Children
2.6%
Animation
2.2%
Adventure
7.3%
Action
9.2%
Western
0.7%
War
2.7%
Thriller
9.5%
Sci-Fi
5.0%
Romance
7.1%
Mystery
2.9%
Musical
1.8%
Horror
3.1%
Film-Noir
0.9%
Fantasy
4.4%
Drama
20.8%
Documentary
1.1%
Crime
6.0%