In [1]:
```python
import pandas as pd
import requests
import time
```

In [2]:
```python
# checking the size/type of the data
tmdb_df = pd.read_csv('../data/zippedData/tmdb.movies.csv.gz')
tmdb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26517 entries, 0 to 26516
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         26517 non-null  int64  
 1   genre_ids          26517 non-null  object 
 2   id                 26517 non-null  int64  
 3   original_language  26517 non-null  object 
 4   original_title     26517 non-null  object 
 5   popularity         26517 non-null  float64
 6   release_date       26517 non-null  object 
 7   title              26517 non-null  object 
 8   vote_average       26517 non-null  float64
 9   vote_count         26517 non-null  int64  
dtypes: float64(2), int64(3), object(5)
memory usage: 2.0+ MB
```

In [ ]:
```python
# testing the api key from The Movie Database
# the 3 or 4 after '.org/' determines the version, and therefore the key to use

API_KEY_V3 = 'd15ab8aefa8ddede64de44721f315562'
API_KEY_V4 = 'eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkMTVhYjhhZWZhOGRkZWRlNjRkZTQ0NzIxZjMxNTU2

url = "https://api.themoviedb.org/4/auth/request_token"

payload = "{\"redirect_to\":\"http://www.themoviedb.org/\"}"
headers = {
    'content-type': "application/json;charset=utf-8",
    'authorization': "Bearer " + API_KEY_V4
    }

response = requests.request("POST", url, data=payload, headers=headers)

print(response.text)
```

In [ ]:
```python
# using the sample search string in the api documentation
# checking to see if breaking it down gives a valid response

search_url = 'https://api.themoviedb.org/3/search/movie?api_key='

string_ = 'Jack+Reacher'
string_query = string_.replace(' ', '+')

# search format
# https://api.themoviedb.org/3/search/movie?api_key={api_key}&query=Jack+Reacher
search_response = search_url + API_KEY_V3 + '&query=' + string_query


r = requests.get(search_response)
print(r)
```

In [ ]:
```
# using one of the last movies in the provided dataframe
# checking that we can get the output we need

by_title_url = 'https://api.themoviedb.org/3/movie/'
mv_id = '488143'
get_response = by_title_url + mv_id + '?api_key=' + API_KEY_V3 + '&language=en-US'

r = requests.get(get_response)
r.json()
```

In [ ]:
```
# sample for loop

for movie in tmdb_df.head()['id']:
    print(movie)
```

In [ ]:
```
# getting the data from the api
# it took over two hours to run
# activate at your own peril
def tmdb_api(sleeper=5):
    '''
    this retrieves all of the movie data based on the previously provided data.
    sleeper: the amount of time (in milliseconds) to wait between requests
    '''
    dict_list = []

    # grabs the data via request based on movie id from provided data
    # has a 1/20th second delay to avoid getting rejected by server
    # adds data as a json dict to the above list
    for id_num in tmdb_df['id']:
        time.sleep(sleeper/100)
        mv_id = str(id_num)
        get_response = by_title_url + mv_id + '?api_key=' + API_KEY_V3 + '&language=en-
        r = requests.get(get_response)
        dict_list.append(r.json())

    # converts list of dicts to dataframe for ease of access
    test_df = pd.DataFrame(dict_list)


# tmbd_api()
```

In [ ]:
```
# creating a list of all the columns
remove_list = list(test_df)

# creating a list of all the columns we want to keep
# then removing those from the list above
keep_list = ['budget', 'genres', 'id', 'imdb_id', 'original_title', 'release_date', 're
for column in keep_list:
    remove_list.remove(column)

# removing unwanted columns
test_df = test_df.drop(remove_list, axis=1)
test_df
```

In [ ]:
```
# saving the data as a csv in our project folder
# test_df.to_csv('the_movie_db_filtered.csv')
```

```
In [8]:  tmdb_df = pd.read_csv('../data/tmdb_filtered.csv',index_col=0)
          tmdb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26517 entries, 0 to 26516
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   budget          26013 non-null  float64
 1   genres          26013 non-null  object 
 2   id              26013 non-null  float64
 3   imdb_id         24615 non-null  object 
 4   original_title  26013 non-null  object 
 5   release_date    26010 non-null  object 
 6   revenue         26013 non-null  float64
 7   vote_average    26013 non-null  float64
 8   vote_count      26013 non-null  float64
dtypes: float64(5), object(4)
memory usage: 2.0+ MB
```

```
In [9]:  zero_r = tmdb_df.revenue > 0
          phase1_df = tmdb_df.loc[zero_r]
          phase1_df.shape
```

```
Out[9]:  (3560, 9)
```

```
In [10]:  zero_b = phase1_df.budget > 0
           phase2_df = phase1_df.loc[zero_b]
           phase2_df.shape
```

```
Out[10]:  (2485, 9)
```

the movie industry is pay-to-win.

so we only *really* have 2485 useful entries.

```
In [ ]:
```