```
%%HTML
<style type="text/css">

div.h2 {
    background-color: #3E5AE6;
    background-image: linear-gradient(120deg, #3E5AE6, #A37CE6);
    text-align: left;
    color: white;
    padding:9px;
    padding-right: 100px;
    font-size: 20px;
    max-width: 1500px;
    margin: auto;
    margin-top: 40px;
}

body {
  font-size: 12px;
}

div.h3 {
    color: #3E5AE6;
    font-size: 18px;
    margin-top: 20px;
    margin-bottom:4px;
}

div.h4 {
    color: #159957;
    font-size: 15px;
    margin-top: 20px;
    margin-bottom: 8px;
}

span.note {
    font-size: 5;
    color: gray;
    font-style: italic;
}

hr {
    display: block;
    color: gray
    height: 1px;
    border: 0;
    border-top: 1px solid;
}


hr.light {
    display: block;
    color: lightgray
    height: 1px;
    border: 0;
    border-top: 1px solid;
}

table.dataframe th
{
    border: 1px darkgray solid;
    color: black;
    background-color: white;
}

table.dataframe td
{
    border: 1px darkgray solid;
    color: black;
    background-color: white;
    font-size: 14px;
    text-align: center;
}

table.rules th
{
    border: 1px darkgray solid;
    color: black;
    background-color: white;
    font-size: 14px;
}
```

```
table.rules td
{
    border: 1px darkgray solid;
    color: black;
    background-color: white;
    font-size: 13px;
    text-align: center;
}


table.rules tr.best
{
    color: green;
}

.output {
    align-items: center;
}

.output_png {
    display: table-cell;
    text-align: center;
    margin:auto;
}

</style>
```

```
!pip install -U vega_datasets notebook vega
```

```
Requirement already satisfied: vega_datasets in /usr/local/lib/python3.10/dist-packages (0.9.0)
Requirement already satisfied: notebook in /usr/local/lib/python3.10/dist-packages (6.4.8)
Collecting notebook
  Downloading notebook-6.5.4-py3-none-any.whl (529 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 529.8/529.8 kB 10.4 MB/s eta 0:00:00
Collecting vega
  Downloading vega-4.0.0-py3-none-any.whl (3.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 80.3 MB/s eta 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from vega_datasets) (1.5.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from notebook) (3.1.2)
Requirement already satisfied: tornado>=6.1 in /usr/local/lib/python3.10/dist-packages (from notebook) (6.3.1)
Requirement already satisfied: pyzmq>=17 in /usr/local/lib/python3.10/dist-packages (from notebook) (23.2.1)
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-packages (from notebook) (21.3.0)
Requirement already satisfied: traitlets>=4.2.1 in /usr/local/lib/python3.10/dist-packages (from notebook) (5.7.1)
Requirement already satisfied: jupyter-core>=4.6.1 in /usr/local/lib/python3.10/dist-packages (from notebook) (5.3.1)
Requirement already satisfied: jupyter-client>=5.3.4 in /usr/local/lib/python3.10/dist-packages (from notebook) (6.1.12)
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-packages (from notebook) (0.2.0)
Requirement already satisfied: nbformat in /usr/local/lib/python3.10/dist-packages (from notebook) (5.9.0)
Requirement already satisfied: nbconvert>=5 in /usr/local/lib/python3.10/dist-packages (from notebook) (6.5.4)
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3.10/dist-packages (from notebook) (1.5.6)
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-packages (from notebook) (5.5.6)
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from notebook) (1.8.2)
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from notebook) (0.17.1)
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.10/dist-packages (from notebook) (0.17.0)
Collecting nbclassic>=0.4.7 (from notebook)
  Downloading nbclassic-1.0.0-py3-none-any.whl (10.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.0/10.0 MB 128.5 MB/s eta 0:00:00
Collecting ipytablewidgets<0.4.0,>=0.3.0 (from vega)
  Downloading ipytablewidgets-0.3.1-py2.py3-none-any.whl (190 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 190.2/190.2 kB 27.9 MB/s eta 0:00:00
Collecting jupyter<2.0.0,>=1.0.0 (from vega)
  Downloading jupyter-1.0.0-py2.py3-none-any.whl (2.7 kB)
Requirement already satisfied: ipywidgets<9,>=7.5.0 in /usr/local/lib/python3.10/dist-packages (from ipytablewidgets<0.4.0,>=0.3
Collecting traittypes>=0.0.6 (from ipytablewidgets<0.4.0,>=0.3.0->vega)
  Downloading traittypes-0.2.1-py2.py3-none-any.whl (8.6 kB)
Requirement already satisfied: numpy<2.0.0,>=1.10.4 in /usr/local/lib/python3.10/dist-packages (from ipytablewidgets<0.4.0,>=0.3
Collecting lz4 (from ipytablewidgets<0.4.0,>=0.3.0->vega)
  Downloading lz4-4.3.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 56.7 MB/s eta 0:00:00
Collecting qtconsole (from jupyter<2.0.0,>=1.0.0->vega)
  Downloading qtconsole-5.4.3-py3-none-any.whl (121 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 121.9/121.9 kB 18.1 MB/s eta 0:00:00
Requirement already satisfied: jupyter-console in /usr/local/lib/python3.10/dist-packages (from jupyter<2.0.0,>=1.0.0->vega) (6.
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.10/dist-packages (from jupyter-client>=5.3.4->note
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.6.1->notebook)
Requirement already satisfied: jupyter-server>=1.8 in /usr/local/lib/python3.10/dist-packages (from nbclassic>=0.4.7->notebook)
Collecting notebook-shim>=0.2.3 (from nbclassic>=0.4.7->notebook)
  Downloading notebook_shim-0.2.3-py3-none-any.whl (13 kB)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook) (4.9.2)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook) (4.11.2)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook) (6.0.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook) (0.7.1)
```

```
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook) (0.4)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook) (0.2
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook) (2.1.3)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert>=5->notebook) (0.8.4
```

```
!pip install ujson
```

```
Collecting ujson
  Downloading ujson-5.8.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (53 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.9/53.9 kB 3.1 MB/s eta 0:00:00
Installing collected packages: ujson
Successfully installed ujson-5.8.0
```

```
%env JOBLIB_TEMP_FOLDER=/tmp
!pip install pyspark
```

```
env: JOBLIB_TEMP_FOLDER=/tmp
Collecting pyspark
  Downloading pyspark-3.4.1.tar.gz (310.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 310.8/310.8 MB 4.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.4.1-py2.py3-none-any.whl size=311285398 sha256=b292e7a2b75d82cc027f92b4aeee0117d74f
  Stored in directory: /root/.cache/pip/wheels/0d/77/a3/ff2f74cc9ab41f8f594dabf0579c2a7c6de920d584206e0834
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.4.1
```

```
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 8)
import os
import gc
import ujson as json
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.patches as patches
import seaborn as sns

import plotly as py
import plotly.express as px
import plotly.graph_objs as go
from plotly.subplots import make_subplots
from plotly.offline import download_plotlyjs
from plotly.offline import init_notebook_mode
from plotly.offline import plot,iplot

init_notebook_mode(connected=True)

import altair as alt
from altair.vega import v5
from IPython.display import HTML
alt.renderers.enable('notebook')

from IPython.display import HTML
from IPython.display import Image
from IPython.display import display
from IPython.core.display import display
from IPython.core.display import HTML
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import warnings
warnings.filterwarnings('ignore')

plt.style.use('seaborn')
color_pal = [x['color'] for x in plt.rcParams['axes.prop_cycle']]
%config InlineBackend.figure_format = 'svg'
th_props = [('font-size', '13px'), ('background-color', 'white'), ('color', '#666666')]
td_props = [('font-size', '15px'), ('background-color', 'white')]
styles = [dict(selector="td", props=td_props), dict(selector="th", props=th_props)]

SMALL_SIZE = 8
MEDIUM_SIZE = 10
BIGGER_SIZE = 12
plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
```

```
plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title
```

```
/usr/local/lib/python3.10/dist-packages/altair/vega/v5/__init__.py:18: AltairDeprecationWarning:

The module altair.vega.v5 is deprecated and will be removed in Altair 5.
```

```
# using ideas from this kernel: https://www.kaggle.com/notslush/altair-visualization-2018-stackoverflow-survey
def prepare_altair():
    """
    Helper function to prepare altair for working.
    """

    vega_url = 'https://cdn.jsdelivr.net/npm/vega@' + v5.SCHEMA_VERSION
    vega_lib_url = 'https://cdn.jsdelivr.net/npm/vega-lib'
    vega_lite_url = 'https://cdn.jsdelivr.net/npm/vega-lite@' + alt.SCHEMA_VERSION
    vega_embed_url = 'https://cdn.jsdelivr.net/npm/vega-embed@3'
    noext = "?noext"

    paths = {
        'vega': vega_url + noext,
        'vega-lib': vega_lib_url + noext,
        'vega-lite': vega_lite_url + noext,
        'vega-embed': vega_embed_url + noext
    }

    workaround = f"""    requirejs.config({{
        baseUrl: 'https://cdn.jsdelivr.net/npm/',
        paths: {paths}
    }});
    """

    return workaround

def add_autoincrement(render_func):
    # Keep track of unique <div/> IDs
    cache = {}
    def wrapped(chart, id="vega-chart", autoincrement=True):
        if autoincrement:
            if id in cache:
                counter = 1 + cache[id]
                cache[id] = counter
            else:
                cache[id] = 0
            actual_id = id if cache[id] == 0 else id + '-' + str(cache[id])
        else:
            if id not in cache:
                cache[id] = 0
            actual_id = id
        return render_func(chart, id=actual_id)
    # Cache will stay outside and
    return wrapped

@add_autoincrement
def render(chart, id="vega-chart"):
    """
    Helper function to plot altair visualizations.
    """
    chart_str = """
    <div id="{id}"></div><script>
    require(["vega-embed"], function(vg_embed) {{
        const spec = {chart};
        vg_embed("#{id}", spec, {{defaultStyle: true}}).catch(console.warn);
        console.log("anything?");
    }});
    console.log("really...anything?");
    </script>
    """
    return HTML(
        chart_str.format(
            id=id,
            chart=json.dumps(chart) if isinstance(chart, dict) else chart.to_json(indent=None)
        )
    )

# setting up altair
workaround = prepare_altair()
```

```
HTML("".join((
    "<script>",
    workaround,
    "</script>",
)))
```

It's time to take a look in all files provided by the dataset.

```
#print('Data Files in Directory')
#print(os.listdir(DATA_PATH))
```

For now, I will ignore all small dataset versions.
Time to import relvant (Ratings, Links and Metadata) files and check the data.

```
ratings = pd.read_csv('/content/ratings.csv')
links = pd.read_csv('/content/links.csv')
metadata = pd.read_csv('/content/movies_metadata.csv')
```

```
# Function that I wrote to print all relevant infos in dataset
import io

def get_df_info(df):
    display(df.head(3))
    buf = io.StringIO()
    df.info(buf=buf)
    info = buf.getvalue().split('\n')[-2]
    display(f'Number of Rows: {df.shape[0]}, Number of Columns: {df.shape[1]}')
    display('Data Types')
    df_types = df.dtypes
    df_types = pd.DataFrame({'Column':df_types.index, 'Type':df_types.values})
    display(df_types)
    display(info)
    missing = df.isnull().sum().sort_values(ascending=False)
    display('Missing Values')
    if missing.values.sum() == 0:
        display('No Missing Values')
    else:
        missing = missing[missing > 0]
        missing = pd.DataFrame({'Column' : missing.index, 'Missing Values' : missing.values})
        display(missing)
```

Ratings Content

```
get_df_info(ratings)
```

	userId	movieId	rating	timestamp
0	1	110	1.0	1.425942e+09
1	1	147	4.5	1.425942e+09
2	1	858	5.0	1.425942e+09

```
'Number of Rows: 1534062, Number of Columns: 4'
'Data Types'
```

	Column	Type
0	userId	int64
1	movieId	int64
2	rating	float64
3	timestamp	float64

```
'memory usage: 46.8 MB'
'Missing Values'
```

	Column	Missing Values
0	timestamp	1

Links Content

```
get_df_info(links)
```

	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0

```
'Number of Rows: 45843, Number of Columns: 3'
'Data Types'
```

	Column	Type
0	movieId	int64
1	imdbId	int64
2	tmdbId	float64

```
'memory usage: 1.0 MB'
'Missing Values'
```

	Column	Missing Values
0	tmdbId	219

Metadata Content

```
get_df_info(metadata)
```

In all the data info displayed, we can see that only ratings have a large memory usage, and I will use this dataset to make the recommendation system based on user ratings, the dataset have the relevant data like userId, movieId and ratings.

It's important to say that the ratings dataset doesn't have any missing value, therefore, will not needed any treatment like data imputation or drop NA rows.

The other datasets will be used for Exploratory Data Analysis.

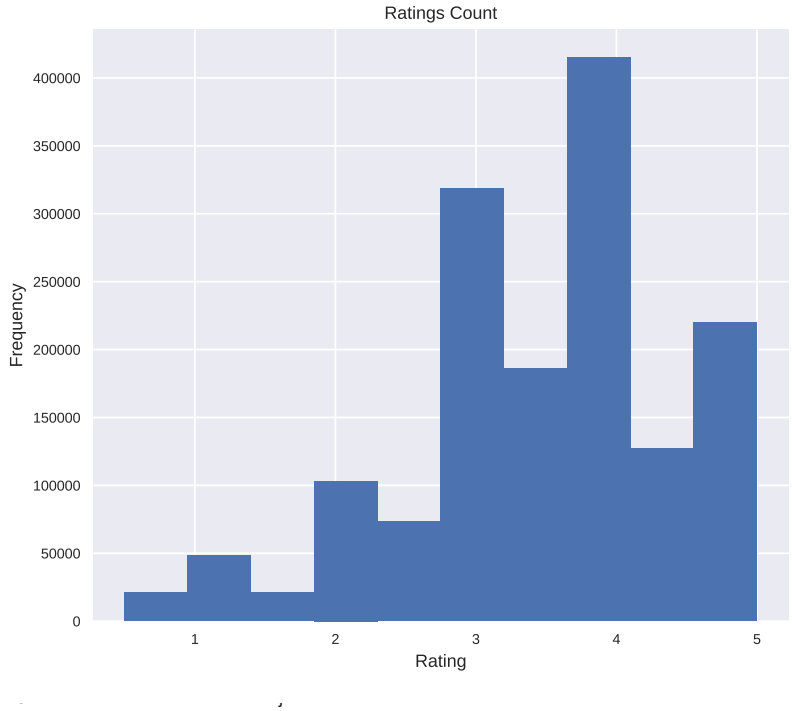
Exploratory Data Analysis

Let's start with the following approaches

* Rating Frequency. * Analysis of most rated movies. * World cloud with most common words.

Let's start plotting an Histogram to see the rating distribution.

```
plt.rcParams['figure.figsize'] = (7, 6)
plt.hist(ratings['rating'], bins=10);
plt.title('Ratings Count', size=10)
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show();
```



And for a better visualization, let's represent by a pie chart with the percent representation.

```
values = ratings.rating.value_counts()
labels = values.index
colors = ['red', 'blue', 'green', 'yellow', 'black']
trace = go.Pie(labels=labels,
               values=values,
                marker=dict(colors=colors)
              )
layout = go.Layout(title='Ratings by Total Percent')
fig = go.Figure(data=trace, layout=layout)
iplot(fig)
```

We can see most movies were rated with 4, on a scale of 1 to 5. A fewer movies (compared to the total dataset) were rated with low grades.
Let's see which movies were rated most times, taking the 10 most rated.

```
df_aux = ratings['movieId'].value_counts().reset_index().head(10).rename(columns={'index': 'movieId', 'movieId': 'count'})
df_aux['movieId'] = df_aux['movieId'].astype(str)


render(alt.Chart(df_aux).mark_bar().encode(
    x=alt.X('movieId:N', axis=alt.Axis(title='Movie ID'), sort=list(df_aux['movieId'].values)),
    y=alt.Y('count:Q', axis=alt.Axis(title='Total Count')),
    tooltip=['movieId', 'count']
).properties(title='Movie Count', height=300, width=800).interactive())
```

Time to discover which movies have this IDs.
Let's check the IDs on IMDB and get some info.

```
# Get the Movie on metadata
def get_movie_metadata(movieId):
    metadata['imdb_id'] = metadata['imdb_id'].astype('category')
    imdb_id = links[links['movieId'] == movieId]
    imdb_id = imdb_id.imdbId.values[0]
    if len(str(imdb_id)) == 7:
        movie_rated = metadata[metadata['imdb_id'] == 'tt'+imdb_id.astype(str)]
        df = movie_rated.loc[:,['title', 'overview', 'vote_average', 'release_date']]
        return df.reset_index(drop=True)
    elif len(str(imdb_id)) == 6:
        movie_rated = metadata[metadata['imdb_id'] == 'tt0'+imdb_id.astype(str)]
        df = movie_rated.loc[:,['title', 'overview', 'vote_average', 'release_date']]
        return df.reset_index(drop=True)
    elif len(str(imdb_id)) == 5:
        movie_rated = metadata[metadata['imdb_id'] == 'tt00'+imdb_id.astype(str)]
        df = movie_rated.loc[:,['title', 'overview', 'vote_average', 'release_date']]
        return df.reset_index(drop=True)
    elif len(str(imdb_id)) == 4:
        movie_rated = metadata[metadata['imdb_id'] == 'tt000'+imdb_id.astype(str)]
        df = movie_rated.loc[:,['title', 'overview', 'vote_average', 'release_date']]
        return df.reset_index(drop=True)
    elif len(str(imdb_id)) == 3:
        movie_rated = metadata[metadata['imdb_id'] == 'tt0000'+imdb_id.astype(str)]
        df = movie_rated.loc[:,['title', 'overview', 'vote_average', 'release_date']]
        return df.reset_index(drop=True)
    elif len(str(imdb_id)) == 2:
        movie_rated = metadata[metadata['imdb_id'] == 'tt00000'+imdb_id.astype(str)]
        df = movie_rated.loc[:,['title', 'overview', 'vote_average', 'release_date']]
        return df.reset_index(drop=True)
    elif len(str(imdb_id)) == 1:
        movie_rated = metadata[metadata['imdb_id'] == 'tt000000'+imdb_id.astype(str)]
        df = movie_rated.loc[:,['title', 'overview', 'vote_average', 'release_date']]
        return df.reset_index(drop=True)
    else:
        pass
# Get Movie List
def get_movie(df):
    movieIdIdx = df['movieId'].values.astype(int)
    df_aux_b = pd.DataFrame({'title': ['aaa'],
                             'overview': ['bbb'],
                             'vote_average': [1.7],
                             'release_date': ['1999-01-01']
        })
    for i in movieIdIdx:
        df_aux_b = df_aux_b.append(get_movie_metadata(i), ignore_index=True)

    df_aux_b.drop(0, inplace=True)
    df_aux_b['release_date'] = df_aux_b['release_date'].apply(lambda x : x.split('-')[0])
```

```
    df_aux_b['release_date'] = df_aux_b['release_date'].astype(int)
    df_aux_b.rename(columns={'release_date' : 'release_year'}, inplace=True)
    return df_aux_b.reset_index(drop=True)


df_movies = get_movie(df_aux)
df_movies
```

	title	overview	vote_average	release_year
0	Forrest Gump	A man with a low IQ has accomplished great thi...	8.2	1994
1	The Shawshank Redemption	Framed in the 1940s for the double murder of h...	8.5	1994
2	Pulp Fiction	A burger-loving hit man, his philosophical par...	8.3	1994
3	The Silence of the Lambs	FBI trainee, Clarice Starling ventures into a ...	8.1	1991
4	The Matrix	Set in the 22nd century, The Matrix tells the ...	7.9	1999
5	Star Wars	Princess Leia is captured and held hostage by ...	8.1	1977
6	Jurassic Park	A wealthy entrepreneur secretly creates a them...	7.6	1993
7	Schindler's List	The true story of how businessman Oskar Schind...	8.3	1993
8	Toy Story	Led by Woody, Andy's toys live happily in his ...	7.7	1995
9	Braveheart	Enraged at the slaughter of Murron, his new br...	7.7	1995

Nice, we have good movies in the list, I'm a great fan of Forrest Gump and The Silence of the Lambs!
Most of the movies listed were released in 90's, the only exception is Star Wars (1977).
Maybe most people are always viewing 90's movies? It's a interesting information to take note.

Another curios information is that in the most rated movies, none have a rate above 9.0.
Let's expand it, let's get all the 1000 most rated movies and examine what words are frequent in they overviews.

```
df_aux = ratings['movieId'].value_counts().reset_index().head(1001).rename(columns={'index': 'movieId', 'movieId': 'count'})
df_aux['movieId'] = df_aux['movieId'].astype(str)
df_aux = get_movie(df_aux)
get_df_info(df_aux)
```

	title	overview	vote_average	release_year
0	Forrest Gump	A man with a low IQ has accomplished great thi...	8.2	1994
1	The Shawshank Redemption	Framed in the 1940s for the double murder of h...	8.5	1994
2	Pulp Fiction	A burger-loving hit man, his philosophical par...	8.3	1994

```
'Number of Rows: 1000, Number of Columns: 4'
'Data Types'
```

	Column	Type
0	title	object
1	overview	object
2	vote_average	float64
3	release_year	int64

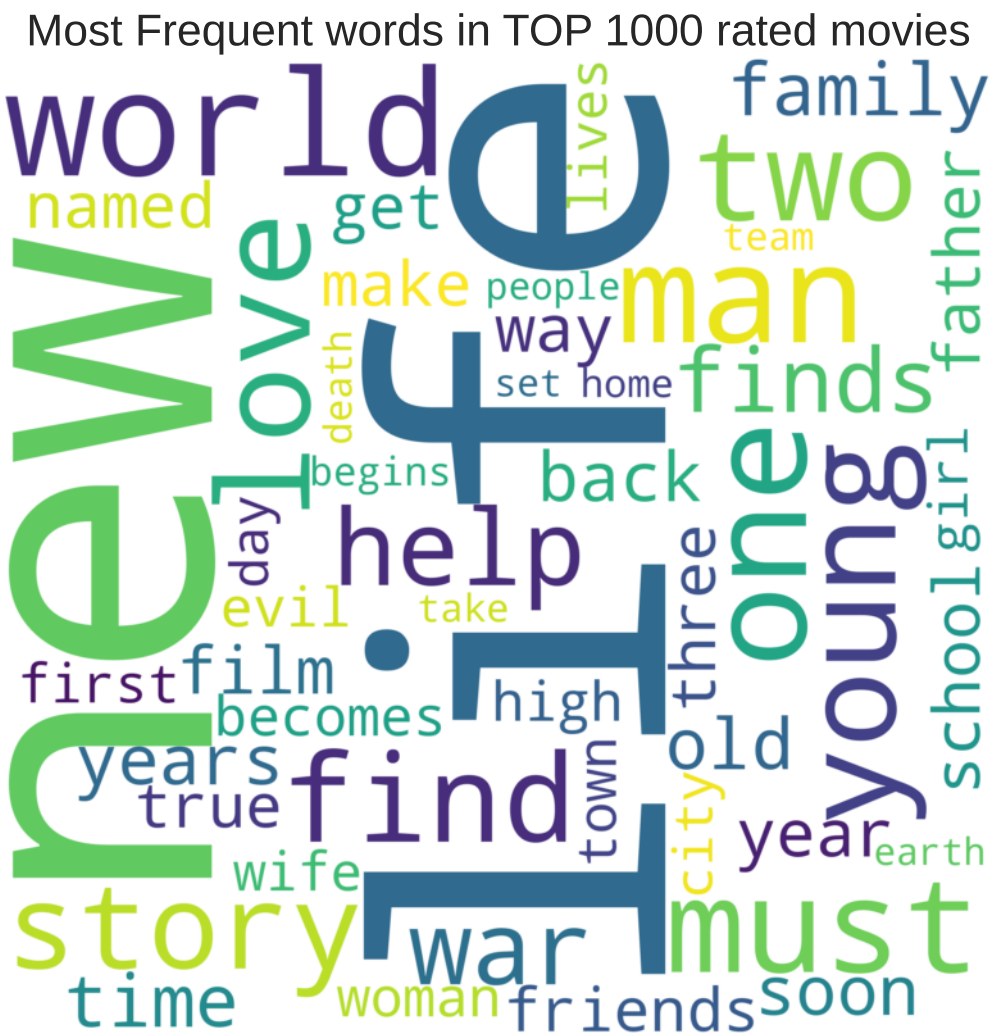
```
'memory usage: 31.4+ KB'
'Missing Values'
'No Missing Values'
```

Time to use Natural Language Processing (NLP) with NLTK module and transform everything in overview for lower case, word tokens and remove stopwords and make the Word Cloud.

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from wordcloud import WordCloud

stop_words = set(stopwords.words('english'))
tokenizer = RegexpTokenizer(r'\w+')

df_aux['overview'] = df_aux.overview.apply(lambda x : x.lower())
df_aux['overview'] = df_aux.overview.apply(lambda x : tokenizer.tokenize(x))
df_aux['overview'] = df_aux.overview.apply(lambda x : [w for w in x if w not in stop_words])
df_aux['overview'] = df_aux.overview.apply(lambda x : ' '.join(x))
```

```
word_count = df_aux.overview.apply(lambda x: pd.value_counts(x.split(" "))).sum(axis = 0).sort_values(ascending=False)
word_count = pd.DataFrame({'word' : word_count.index, 'count': word_count.values})
# Plot the WordCloud
d = {}
for a, x in word_count.values:
    d[a] = x

wordcloud = WordCloud(background_color = 'white',
                      max_words = 50,
                      width = 2000,
                      height = 2000)
wordcloud.generate_from_frequencies(frequencies=d)
plt.rcParams['figure.figsize'] = (10, 10)
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title('Most Frequent words in TOP 1000 rated movies', fontsize = 25)
plt.show();
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```



The words "Life" and "World" are commons and connecting with other words like "Love", "Family", "Father" and "Wife", we caan conclude most of the movies are Family Friendly.
As we can see, the word "Man" have a big presence on the Word Cloud, and looking at the Top 10 Movies Rated most movies have a male protagonist.
This concludes the EDA and it's time to start to build the model.

```
del ratings, df_aux, df_movies
gc.collect()
```

```
2886
```

Model using PySpark

Let's start importing all needed models and setting Spark.

```
import pyspark.sql.functions as sql_func
from pyspark.sql.types import *
from pyspark.ml.recommendation import ALS, ALSModel
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from pyspark.mllib.evaluation import RegressionMetrics, RankingMetrics
from pyspark.ml.evaluation import RegressionEvaluator

sc = SparkContext('local')
spark = SparkSession(sc)
```

Create the Schema.

```
data_schema = StructType([
    StructField('userId', IntegerType(), False),
    StructField('movieId', IntegerType(), False),
    StructField('rating', FloatType(), False),
    StructField('timestamp',IntegerType(), False)
])
final_stat = spark.read.csv('/content/ratings.csv', header=True, schema=data_schema).cache()

ratings = (final_stat.select('userId','movieId','rating')).cache()
```

Split in Train (70%) and Test (30%).

```
(training, test) = ratings.randomSplit([0.7, 0.3], seed=42)
```

And train the model, the evaluation will be made on test set using Mean Absolute Error (MAE).

```
als = ALS(
          rank=30,
          maxIter=4,
          regParam=0.1,
          userCol='userId',
          itemCol='movieId',
          ratingCol='rating',
          coldStartStrategy='drop',
          implicitPrefs=False
         )
model = als.fit(training)

predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName='mae', labelCol='rating',
                                predictionCol='prediction')

mae = evaluator.evaluate(predictions)
print(f'MAE (Test) = {mae}')
```

```
MAE (Test) = 0.6673722472097843
```

And finally, generate the Best recommendation for each user (User Based Recommendation System). The movieId, the first element in recommendations vector, is the same of ratings dataframe.

```
model.recommendForAllUsers(1).show(5)
```

```
+------+------------------+
|userId|    recommendations|
+------+------------------+
|     1|[{60983, 6.104841}]|
|     2| [{95776, 4.65278}]|
|     3|[{171603, 4.00956}]|
|     4|[{51380, 5.337099}]|
|     5|[{60983, 5.843824}]|
+------+------------------+
only showing top 5 rows
```

Let's see which movie was recommended for a particular userId.

```
get_movie_metadata(156589)
```

	title	overview	vote_average	release_date
0	Hate Story 2	The movie is a revenge thriller with Surveen C...	4.5	2014-07-18

Show the most recommended user for each movie (Item Based Recommendation System). Again, the movieId is the same of ratings dataframe.

```
model.recommendForAllItems(1).show(5)
```

```
+-------+--------------------+
|movieId|     recommendations|
+-------+--------------------+
|      1| [{1729, 5.0524607}]|
|      2| [{5517, 4.7457924}]|
|      3| [{8535, 4.7531295}]|
|      4|[{10116, 4.4815817}]|
|      5|[{12290, 4.6578584}]|
+-------+--------------------+
only showing top 5 rows
```