Netflix Movie Recommendation System

Netflix is all about connecting people to the movies they love. To help customers find those movies, they developed world-class movie recommendation system: CinematchSM. Its job is to predict whether someone will enjoy a movie based on how much they liked or disliked other movies. Netflix use those predictions to make personal movie recommendations based on each customer's unique tastes. And while Cinematch is doing pretty well, it can always be made better.

Now there are a lot of interesting alternative approaches to how Cinematch works that netflix haven't tried. Some are described in the literature, some aren't. We're curious whether any of these can beat Cinematch by making better predictions. Because, frankly, if there is a much better approach it could make a big difference to our customers and our business.

Credits: https://www.netflixprize.com/rules.html


```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import warnings
warnings.filterwarnings('ignore')
from datetime import datetime
from scipy import sparse
import dask.dataframe as dd
import dask.bag as db
from dask.delayed import delayed
dir_path = '/content/gdrive/My Drive/netflix/'
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.decomposition import TruncatedSVD
!pip install --upgrade surprise
from surprise import Reader, Dataset
import xgboost as xgb
from surprise import BaselineOnly, KNNBaseline , SVD, SVDpp
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting surprise
  Downloading surprise-0.1-py2.py3-none-any.whl (1.8 kB)
Collecting scikit-surprise (from surprise)
  Downloading scikit-surprise-1.1.3.tar.gz (771 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 772.0/772.0 kB 42.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->surprise) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->surprise) (1.22.4)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->surprise) (1.10.1)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.3-cp310-cp310-linux_x86_64.whl size=3095482 sha256=0756c9f43640d3
  Stored in directory: /root/.cache/pip/wheels/a5/ca/a8/4e28def53797fdc4363ca4af740db15a9c2f1595ebc51fb445
Successfully built scikit-surprise
Installing collected packages: scikit-surprise, surprise
Successfully installed scikit-surprise-1.1.3 surprise-0.1
```

```
start_time = datetime.now()
if not os.path.isfile(dir_path + 'netflix_data.csv'):
    data = open(dir_path + 'netflix_data.csv', mode='w')
    row = []
```

```
    files = [
            dir_path + 'combined_data_1.txt',
            dir_path + 'combined_data_2.txt',
            dir_path + 'combined_data_3.txt',
            dir_path + 'combined_data_4.txt'
        ]
    for file in files:
        print("Reading the file {}...".format(file))
        with open(file) as f:
            for line in f:
                del row[:]
                line = line.strip()
                if line.endswith(':'):
                    movie_id = line.replace(':','')
                else:
                    row = [x for x in line.split(',')]
                    row.insert(0,movie_id)
                    data.write(','.join(row))
                    data.write('\n')
        print("Done...")
    data.close()
print("Time Taken =" ,datetime.now()-start_time )
```

```
Time Taken = 0:00:00.002004
```

```
df = pd.read_csv(dir_path + 'netflix_data.csv')
df.shape
```

```
(100480506, 4)
```

```
# Only run this block when you dont have sorted data by 'time'.
if not os.path.isfile(dir_path + "netflix_sorted_data"):
    start_time = datetime.now()

    # creating the dataframe with 4 columns; namely 'movies_id','customer','rating','date'.
    print("Creating the dataframe")
    df = pd.read_csv(dir_path + "netflix_data.csv", sep=',', names=['movies_id','customer','rating','date'])

    # Checking the null value and then delete them if any.
    print("Checking Null values")
    print("Number of nan values: ",sum(df.isnull().any()))
    df = df.dropna(axis=0, inplace= False)

    # Checking the duplicate entries and remove them.
    dups_bool = df.duplicated(['movies_id','customer','rating'])
    dups = sum(dups_bool)
    print("Total number of duplicate entries: {}".format(dups))
    df = df.drop_duplicates()

    # Sorting the dataframe by date column.
    print("Sorting the dataframe")
    df = df.sort_values(by='date', inplace=False)
    print("Done")
    print("Saving... Please wait")
    df.to_csv(dir_path + "netflix_sorted_data", index=False)
    print("Saved")
    print("Time Taken: ", datetime.now()-start_time)
else:
    df = pd.read_csv(dir_path + 'netflix_sorted_data', sep=',' )
```

```
print("Total number of ratings in whole dataset: ", df.shape[0])
print("Total number of Users in whole dataset: ", len(np.unique(df.customer)))
print("Total number of Movies in whole dataset: ",len(np.unique(df.movies_id)))
```

```
Total number of ratings in whole dataset:  100480507
Total number of Users in whole dataset:  480189
Total number of Movies in whole dataset:  17770
```

```
df.customer.describe()
```

```
count    1.004805e+08
mean     1.322489e+06
std      7.645368e+05
min      6.000000e+00
25%      6.611980e+05
50%      1.319012e+06
75%      1.984455e+06
max      2.649429e+06
Name: customer, dtype: float64
```

```
df.rating.describe()
```

```
count    1.004805e+08
mean     3.604290e+00
std      1.085219e+00
min      1.000000e+00
25%      3.000000e+00
50%      4.000000e+00
75%      4.000000e+00
max      5.000000e+00
Name: rating, dtype: float64
```

Spliting the data into Train, Cross Validate and Test dataset.

```
#saving the datasets and loading them.
start = datetime.now()
if not os.path.isfile(dir_path + "train.csv"):
  df.iloc[:int(df.shape[0]*0.6)].to_csv(dir_path + "train.csv")

if not os.path.isfile(dir_path + "cv.csv"):
  df.iloc[int(df.shape[0]*0.6):int(df.shape[0]*0.8)].to_csv(dir_path + "cv.csv")

if not os.path.isfile(dir_path + "test.csv"):
  df.iloc[int(df.shape[0]*0.8):].to_csv(dir_path + "test.csv")
print("Time Taken: ",datetime.now()-start)

train_df = pd.read_csv(dir_path + "train.csv", parse_dates=['date'])
cv_df = pd.read_csv(dir_path + "cv.csv", parse_dates=['date'])
test_df = pd.read_csv(dir_path + "test.csv", parse_dates=['date'])
print("Datasets loaded successfully.")
```

```
Time Taken:  0:00:00.001880
```

Statistic on dataset

```
# This block tells you no. of datapoints in a dataset ,no. of rating, no. of unique users and movies
print("Train Dataset")
print("-"*50)
print("Number of datapoints: ",train_df.shape[0],"(", np.round((train_df.shape[0]/(train_df.shape[0]+cv_df.shape[0]+test_df.shape[0]))*10
print("\nNumber of ratings: {}".format(train_df.rating.shape[0]))
print("Number of users: {}".format(len(np.unique(train_df.customer))))
print("Number of movies: {}".format(len(np.unique(train_df.movies_id))))
print("\n")
print("Cross Validate Dataset")
print("-"*50)
print("Number of datapoints: ",cv_df.shape[0],"(", np.round((cv_df.shape[0]/(train_df.shape[0]+cv_df.shape[0]+test_df.shape[0]))*100),3,"
print("\nNumber of ratings: {}".format(cv_df.rating.shape[0]))
print("Number of users: {}".format(len(np.unique(cv_df.customer))))
print("Number of movies: {}".format(len(np.unique(cv_df.movies_id))))
print("\n")
print("Test Dataset")
print("-"*50)
print("Number of datapoints: ",test_df.shape[0],"(", np.round((test_df.shape[0]/(train_df.shape[0]+cv_df.shape[0]+test_df.shape[0]))*100)
print("\nNumber of ratings: {}".format(test_df.rating.shape[0]))
print("Number of users: {}".format(len(np.unique(test_df.customer))))
print("Number of movies: {}".format(len(np.unique(test_df.movies_id))))
```

```
Train Dataset
--------------------------------------------------
Number of datapoints:  60288304 ( 60.0 3 % )

Number of ratings: 60288304
Number of users: 328767
Number of movies: 16464


Cross Validate Dataset
--------------------------------------------------
Number of datapoints:  20096101 ( 20.0 3 % )

Number of ratings: 20096101
Number of users: 327155
Number of movies: 17386


Test Dataset
--------------------------------------------------
Number of datapoints:  20096102 ( 20.0 3 % )
```

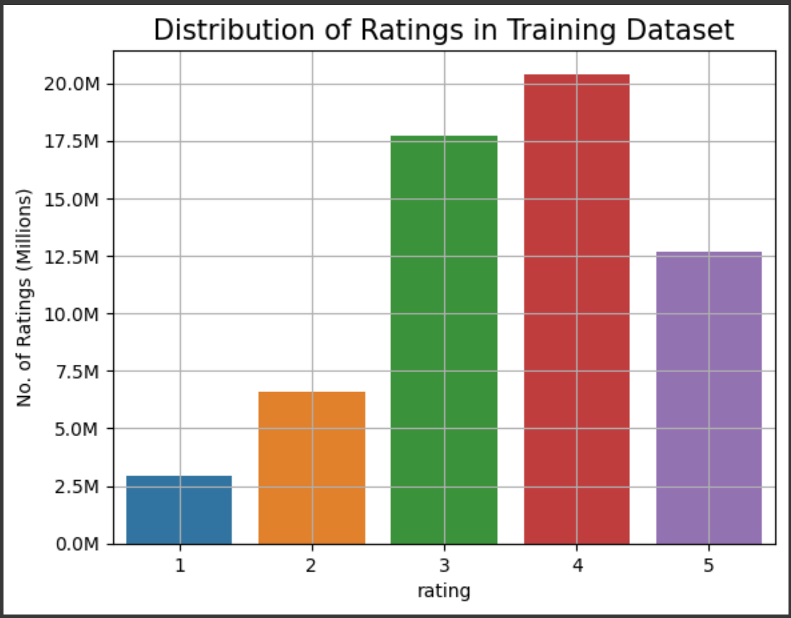
```
Number of ratings: 20096102
Number of users: 349312
Number of movies: 17757
```

```
def human(num , units = 'M'):
  units = units.lower()
  num = float(num)
  if units == 'k':
    return str(num/10**3) + 'K'
  if units == 'm':
    return str(num/10**6) + 'M'
  if units == 'b':
    return str(num/10**9) + 'B'
```

```
print("Occurance of each rating in train dataset.\n")
train_df.rating.value_counts()
```

```
Occurance of each rating in train dataset.

4    20394785
3    17726270
5    12650808
2     6595195
1     2921246
Name: rating, dtype: int64
```

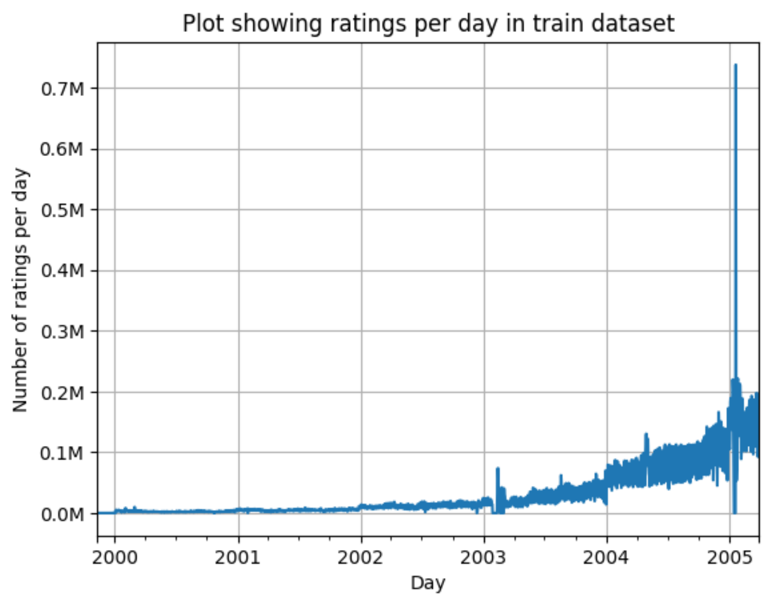
```
fig, ax = plt.subplots()
plt.title('Distribution of Ratings in Training Dataset', fontsize=15)
sns.countplot(x= train_df.rating)
ax.set_ylabel('No. of Ratings (Millions)')
ax.set_yticklabels([human(item, 'M') for item in ax.get_yticks()])
plt.grid()
plt.show()
```



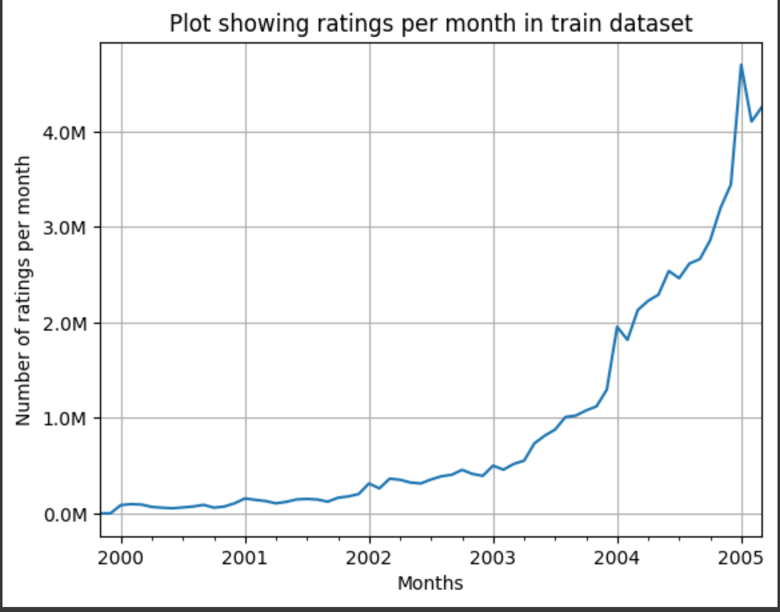

```
pd.options.mode.chained_assignment = None
train_df['days_of_week'] = train_df.date.dt.day_name()
train_df.tail(10)
```

	Unnamed: 0	movies_id	customer	rating	date	days_of_week
60288294	60288294	1305	1329779	2	2005-03-29	Tuesday

```
ax = train_df.resample('d' , on='date')['rating'].count().plot()
ax.set_yticklabels([human(item,'M') for item in ax.get_yticks()])
ax.set_title("Plot showing ratings per day in train dataset")
plt.xlabel("Day")
plt.ylabel("Number of ratings per day")
plt.grid()
plt.show()
```



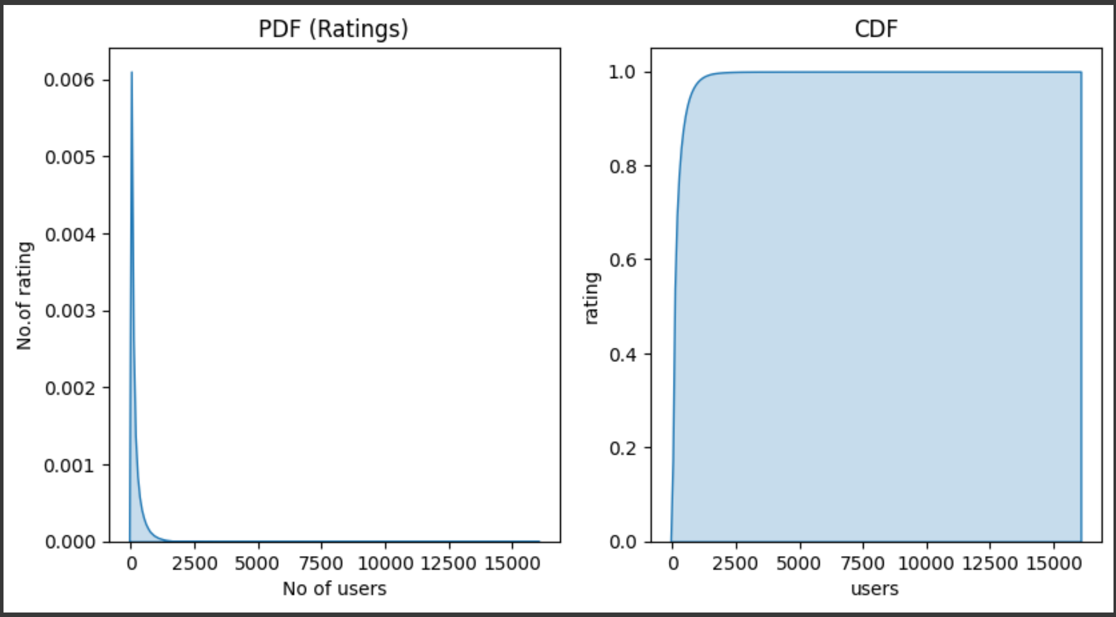

```
ax = train_df.resample('m' , on='date')['rating'].count().plot()
ax.set_yticklabels([human(item,'M') for item in ax.get_yticks()])
ax.set_title("Plot showing ratings per month in train dataset")
plt.xlabel("Months")
plt.ylabel("Number of ratings per month")
plt.grid()
plt.show()
```




```
print("Number of ratings given by top 5 users.")
no_of_rating_per_user = train_df.groupby(by='customer')['rating'].count().sort_values(ascending = False)
no_of_rating_per_user.head()
```

```
Number of ratings given by top 5 users.
customer
305344     15998
2439493    14733
387418     14124
1639792     9748
1932594     7398
Name: rating, dtype: int64
```

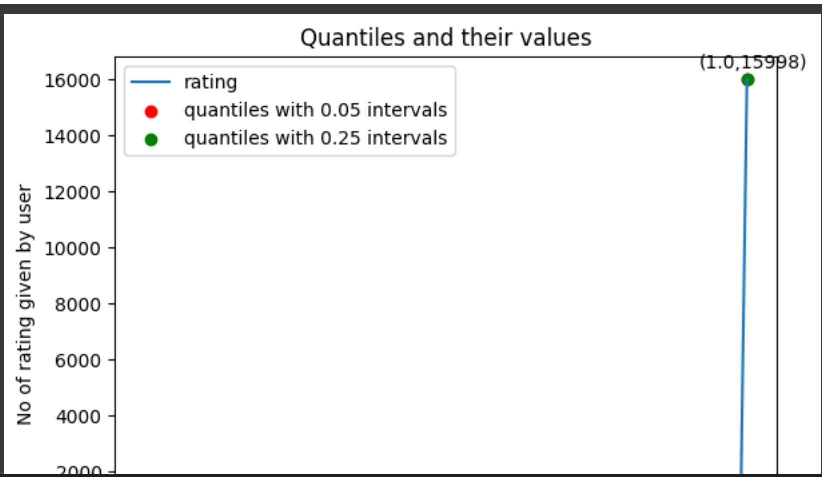
```
fig = plt.figure(figsize = plt.figaspect(.5))
ax1 = plt.subplot(1,2,1)
plt.title("PDF (Ratings)")
plt.xlabel("No of users")
plt.ylabel("No.of rating")
sns.kdeplot(no_of_rating_per_user, ax=ax1,shade= True)

ax2= plt.subplot(1,2,2)
plt.title("CDF")
plt.xlabel('users')
plt.ylabel('rating')
sns.kdeplot(no_of_rating_per_user,cumulative = True, ax = ax2, shade = True)
plt.show()
```




```
no_of_rating_per_user.describe()
```

```
count    328767.000000
mean        183.376993
std         276.787794
min           1.000000
25%          26.000000
50%          81.000000
75%         226.000000
max       15998.000000
Name: rating, dtype: float64
```

```
quantiles = no_of_rating_per_user.quantile(np.arange(0,1.01,0.01), interpolation = 'higher')
quantiles.plot()
plt.title("Quantiles and their values")
plt.xlabel("Values of quantile")
plt.ylabel("No of rating given by user")
# quantiles with 0.05 intervals
plt.scatter(x=quantiles.index[::5], y=quantiles.values[::5], color='red',label= "quantiles with 0.05 intervals")
# quantiles with 0.25 intervals
plt.scatter(x=quantiles.index[::25], y=quantiles.values[::25],color='green',label='quantiles with 0.25 intervals')
plt.legend(loc = 'best')

for x,y in zip(quantiles.index[::25], quantiles.values[::25]):
  ''' s="({} , {})".format(x, y) sets the content of the annotation to a formatted string that includes the values of x and y.
      For example, if x is 0.25 and y is 500, the annotation will display "(0.25, 500)".

      xy=(x, y) specifies the coordinates on the plot where the annotation arrow will point to. It corresponds to the x and y values
      obtained from the quantiles array.

      xytext=(x-0.05, y+500) sets the coordinates where the annotation text will be placed. It determines the position of the text
      relative to the annotation point. In this case, x-0.05 shifts the text slightly to the left of the annotation point, and y+500
      moves the text upward.'''
  plt.annotate(text="({},{})".format(x,y), xy=(x,y), xytext=(x-0.08,y+400))
```




```
print("Quantiles of ratings from 1 to 100 with interval of 5")
quantiles[::5]
```

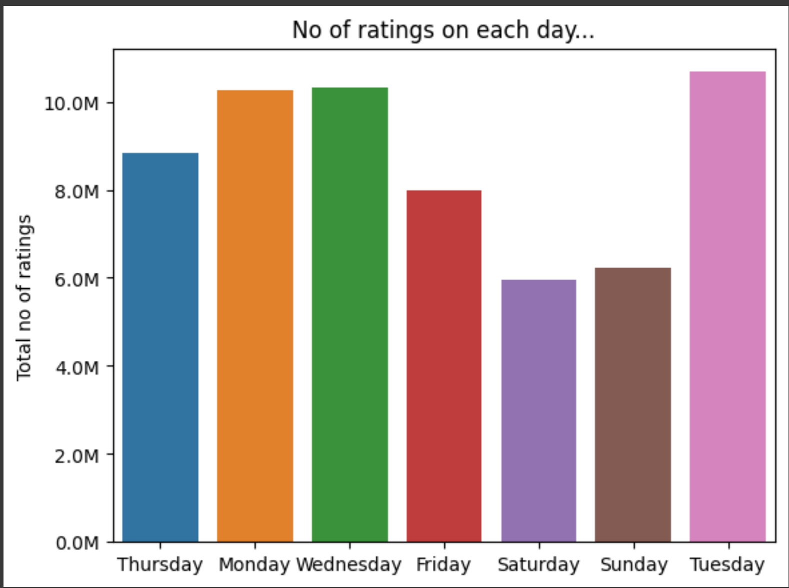
```
Quantiles of ratings from 1 to 100 with interval of 5
0.00        1
0.05        4
0.10        9
0.15       15
0.20       20
0.25       26
0.30       33
0.35       42
0.40       53
0.45       66
0.50       81
0.55      100
0.60      122
0.65      149
0.70      183
0.75      226
0.80      284
0.85      364
0.90      486
0.95      707
1.00    15998
Name: rating, dtype: int64
```

```
print("No of ratinga at last 5 percentile: ",sum(no_of_rating_per_user>707))
```

```
No of ratinga at last 5 percentile:  16419
```

```
no_of_movies_per_rate = train_df.groupby(by ='movies_id')['rating'].count().sort_values(ascending=False)
fig = plt.figure(figsize=plt.figaspect(.5))
ax = plt.gca() #Get Current Axes
plt.plot(no_of_movies_per_rate.values)
plt.title("Plot showing movies that have higher ratings")
plt.xlabel("Movies")
plt.ylabel("Ratings")
ax.set_xticklabels([])
plt.show()
```

```
fig, ax = plt.subplots()
sns.countplot(x='days_of_week', data=train_df, ax=ax)
plt.title('No of ratings on each day...')
plt.ylabel('Total no of ratings')
plt.xlabel('')
ax.set_yticklabels([human(item, 'M') for item in ax.get_yticks()])
plt.show()
```




```
avg_week_df = train_df.groupby(by=['days_of_week'])['rating'].mean()
print(" Average ratings")
print("-"*30)
print(avg_week_df)
print("\n")
```

```
 Average ratings
------------------------------
days_of_week
Friday       3.551549
Monday       3.547009
Saturday     3.559087
Sunday       3.562834
Thursday     3.551846
Tuesday      3.542370
Wednesday    3.554788
Name: rating, dtype: float64
```

```
start = datetime.now()
if os.path.isfile(dir_path + 'train_sparse_matrix.npz'):
    print("It is present in your netflix folder on google drive, getting it from drive....")
    # just get it from the disk instead of computing it
    train_sparse_matrix = sparse.load_npz(dir_path + 'train_sparse_matrix.npz')
    print("DONE..")
else:
    print("We are creating sparse_matrix from the dataframe..")
    # create sparse_matrix and store it for after usage.
    # csr_matrix(data_values, (row_index, col_index), shape_of_matrix)
    # It should be in such a way that, MATRIX[row, col] = data
    train_sparse_matrix = sparse.csr_matrix((train_df.rating.values, (train_df.customer.values,train_df.movie.values)),)

    print('Saving it into disk for furthur usage..')
    # save it into disk
    sparse.save_npz(dir_path + "train_sparse_matrix.npz", train_sparse_matrix)
    print('Done..\n')
print('It\'s shape is : (user, movie) : ',train_sparse_matrix.shape)
print(datetime.now() - start)
```

```
It is present in your netflix folder on google drive, getting it from drive....
DONE..
It's shape is : (user, movie) :  (2649430, 17771)
0:00:05.400190
```

```
tr_us,tr_mv = train_sparse_matrix.shape
tr_element = train_sparse_matrix.count_nonzero()
```

```
print("Sparsity Of Train matrix : {} % ".format(  (1-(tr_element/(tr_us*tr_mv))) * 100) )
```

```
Sparsity Of Train matrix : 99.87195319390865 %
```

```
start = datetime.now()
if os.path.isfile(dir_path + 'cv_sparse_matrix.npz'):
    print("It is present in your netflix folder on google drive, getting it from drive....")
    # just get it from the disk instead of computing it
    cv_sparse_matrix = sparse.load_npz(dir_path + 'cv_sparse_matrix.npz')
    print("DONE..")
else:
    print("We are creating sparse_matrix from the dataframe..")
    # create sparse_matrix and store it for after usage.
    # csr_matrix(data_values, (row_index, col_index), shape_of_matrix)
    # It should be in such a way that, MATRIX[row, col] = data
    cv_sparse_matrix = sparse.csr_matrix((cv_df.rating.values, (cv_df.customer.values,cv_df.movie.values)),)

    print('Saving it into disk for furthur usage..')
    # save it into disk
    sparse.save_npz(dir_path + "cv_sparse_matrix.npz", cv_sparse_matrix)
    print('Done..\n')
print("It\'s shape is : (user, movie) : ",cv_sparse_matrix.shape)
print(datetime.now() - start)
```

```
It is present in your netflix folder on google drive, getting it from drive....
DONE..
It's shape is : (user, movie) :  (2649430, 17771)
0:00:02.386528
```

```
cv_us,cv_mv = cv_sparse_matrix.shape
cv_element = cv_sparse_matrix.count_nonzero()

print("Sparsity Of Train matrix : {} % ".format(  (1-(tr_element/(cv_us*cv_mv))) * 100) )
```

```
Sparsity Of Train matrix : 99.87195319390865 %
```

```
start = datetime.now()
if os.path.isfile(dir_path + 'test_sparse_matrix.npz'):
    print("It is present in your netflix folder on google drive, getting it from drive....")
    # just get it from the disk instead of computing it
    test_sparse_matrix = sparse.load_npz(dir_path + 'test_sparse_matrix.npz')
    print("DONE..")
else:
    print("We are creating sparse_matrix from the dataframe..")
    # create sparse_matrix and store it for after usage.
    # csr_matrix(data_values, (row_index, col_index), shape_of_matrix)
    # It should be in such a way that, MATRIX[row, col] = data
    test_sparse_matrix = sparse.csr_matrix((test_df.rating.values, (test_df.customer.values,test_df.movie.values)),)

    print('Saving it into disk for furthur usage..')
    # save it into disk
    sparse.save_npz(dir_path + "test_sparse_matrix.npz", test_sparse_matrix)
    print('Done..\n')
print('It\'s shape is : (user, movie) : ',test_sparse_matrix.shape)
print(datetime.now() - start)
```

```
It is present in your netflix folder on google drive, getting it from drive....
DONE..
It's shape is : (user, movie) :  (2649430, 17771)
0:00:02.478079
```

```
te_us,te_mv = test_sparse_matrix.shape
te_element = test_sparse_matrix.count_nonzero()

print("Sparsity Of Train matrix : {} % ".format(  (1-(te_element/(te_us*te_mv))) * 100) )
```

```
Sparsity Of Train matrix : 99.95731772988694 %
```

```
def get_average_ratings(sparse_matrix , of_user):
  # chose ax=1 if of_user is True, 0 other wise.
  ax=1 if of_user else 0
  # A1 is used to convert the column matrix into 1-d array.
  # Represents the sum of ratings for that user or movie,
  sum_of_ratings = sparse_matrix.sum(axis=ax).A1
  # Create the boolean matrix denoting whether the user rated the movie or not.
  is_rated = sparse_matrix != 0
  # Represents the number of ratings received by that user or movie.
  no_of_ratings = is_rated.sum(axis=ax).A1
  u, m = sparse_matrix.shape
```

```
  # This line of code creates a dictionary where the keys are user or movie IDs, and the values are the corresponding average ratings.
  # The comprehension filters out IDs with zero ratings and calculates the average rating for the remaining IDs by dividing the
  # sum of ratings by the number of ratings.
  average_rating = {i: sum_of_ratings[i] / no_of_ratings[i] for i in range(u if of_user else m) if(no_of_ratings[i] != 0)}
  return average_rating
```

Global rating for train dataset

```
train_average = dict()
train_global_rating = train_sparse_matrix.sum()/ train_sparse_matrix.count_nonzero()
train_average['global_rating'] = train_global_rating
train_average
```

```
{'global_rating': 3.551661131485802}
```

User rating for train dataset

```
user_id = 29
train_average['user_rating'] = get_average_ratings(train_sparse_matrix, of_user = True)
if user_id in train_average['user_rating']:
  print("Average rating of User: ",user_id," is ",train_average['user_rating'][user_id])
else:
  print("There is no rating with user_id {}".format(user_id))
```

```
There is no rating with user_id 29
```

```
user_id = 10
# train_average['user_rating'] = get_average_ratings(train_sparse_matrix, of_user = True)
if user_id in train_average['user_rating']:
  print("Average rating of User: ",user_id," is ",train_average['user_rating'][user_id])
else:
  print("There is no rating with user_id {}".format(user_id))
```

```
Average rating of User:  10  is  3.3846153846153846
```

Movie rating for train dataset

```
movie_id = 29
train_average['movie_rating'] = get_average_ratings(train_sparse_matrix, of_user = False)
if movie_id in train_average['movie_rating']:
  print("Average rating of Movie: ",movie_id," is ",train_average['movie_rating'][movie_id])
else:
  print("There is no rating with movie_id {}".format(movie_id))
```

```
Average rating of Movie:  29  is  3.5313351498637604
```

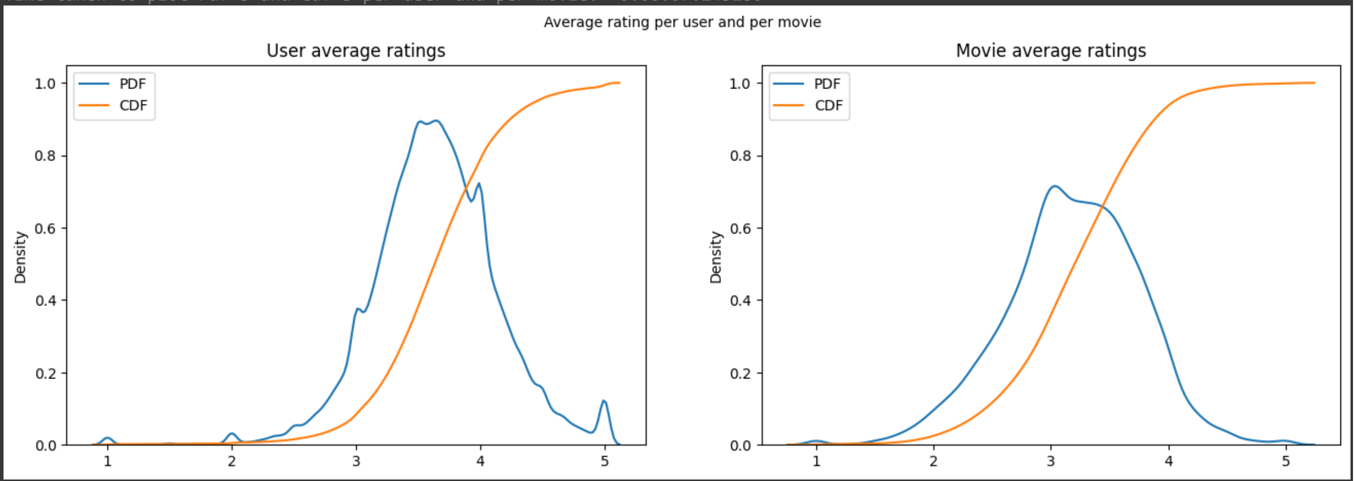
Observation: By this we observe that the movie with id 29 has an not superhit but average.

```
movie_id = 10
train_average['movie_rating'] = get_average_ratings(train_sparse_matrix, of_user = False)
if movie_id in train_average['movie_rating']:
  print("Average rating of Movie: ",movie_id," is ",train_average['movie_rating'][movie_id])
else:
  print("There is no rating with movie_id {}".format(movie_id))
```

```
Average rating of Movie:  10  is  3.193877551020408
```

```
start_time = datetime.now()
figure , (ax1,ax2) = plt.subplots(nrows = 1, ncols=2, figsize= plt.figaspect(.3))
figure.suptitle("Average rating per user and per movie", fontsize=10)
ax1.set_title("User average ratings")
user_average = [ratio for ratio in train_average['user_rating'].values()]
sns.kdeplot(user_average , ax = ax1 , label= 'PDF' )
sns.kdeplot(user_average , cumulative=True , ax= ax1, label= 'CDF')

ax2.set_title("Movie average ratings")
movie_average = [ratio for ratio in train_average['movie_rating'].values()]
sns.kdeplot(movie_average , ax = ax2 , label= 'PDF' )
sns.kdeplot(movie_average , cumulative=True , ax= ax2, label= 'CDF')

ax1.legend()
ax2.legend()

print("Time taken to plot Pdf's and Cdf's per user and per movie: ",datetime.now() - start_time\
      )
```

```
Time taken to plot Pdf's and Cdf's per user and per movie:  0:00:07.143280
```



Cold Start Problem

Cold start problem with users

```
total_user_r = len(np.unique(df['customer']))
user_r = len(train_average['user_rating'])
print("Total number of Users in dataset: ",total_user_r)
print("Number of users present in train dataset: ", user_r)
print("Number of users NOT present in train dataset: ",total_user_r - user_r , ((total_user_r-user_r)/total_user_r)*100, "%")
```

```
Total number of Users in dataset:  480189
Number of users present in train dataset:  328767
Number of users NOT present in train dataset:  151422 31.53383355303849 %
```

Cold start problem with movies

```
total_movie_r = len(np.unique(df['movies_id']))
movie_r = len(train_average['movie_rating'])
print("Total number of Movie in dataset: ",total_movie_r)
print("Number of movie present in train dataset: ", movie_r)
print("Number of movie NOT present in train dataset: ",total_movie_r - movie_r , ((total_movie_r-movie_r)/total_user_r)*100, "%")
```

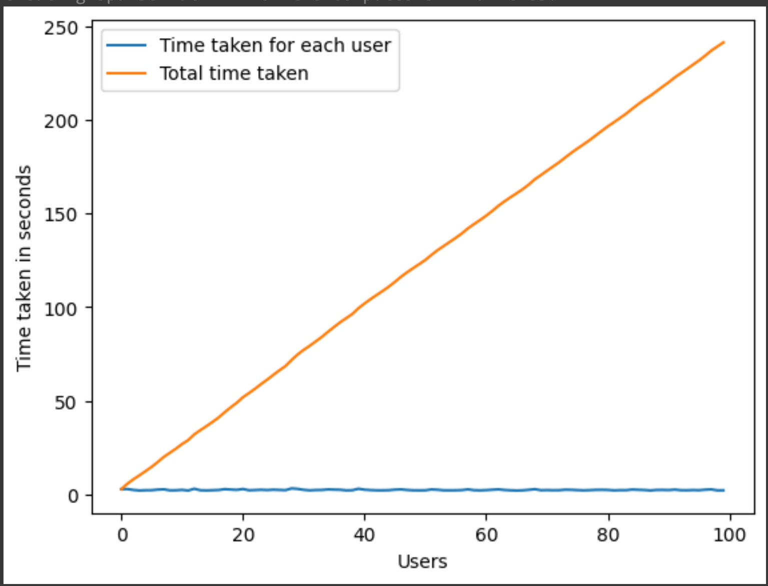
```
Total number of Movie in dataset:  17770
Number of movie present in train dataset:  16464
Number of movie NOT present in train dataset:  1306 0.27197624268777504 %
```

```
def compute_user_similarity(sparse_matrix, verbose=False, compute_for_few=False, top=100,verbose_for_n_rows=20,draw_time_taken=True):
  no_of_user, _ = sparse_matrix.shape
  row_ind, col_ind = sparse_matrix.nonzero()
  row_ind = sorted(set(row_ind))
  time_taken = list()
  rows, cols, data = list(), list(), list()
  if verbose: print("Computing top {} similar users".format(top))
```

```
    start_time = datetime.now()
    temp = 0
    for row in row_ind[:top] if compute_for_few else row_ind:
      temp = temp+1
      prev = datetime.now()

      sim = cosine_similarity(sparse_matrix.getrow(row),sparse_matrix).ravel()  # (X)^T.X
      top_sim_ind = sim.argsort()[-top:]
      top_sim_val = sim[top_sim_ind]

      rows.extend([row]*top)
      cols.extend(top_sim_ind)
      data.extend(top_sim_val)
      time_taken.append(datetime.now().timestamp() - prev.timestamp())
      # The purpose of this condition is typically to provide periodic progress updates or print intermediate results during a long computa
      if verbose:
        if temp % verbose_for_n_rows == 0 :
          print("Computing done for {} users [time elapsed : {} ]".format(temp, datetime.now() -start_time))
    if verbose: print("Creating sparse matrix from the computes similarities.")

    if draw_time_taken:
        plt.plot(time_taken, label= 'Time taken for each user')
        plt.plot(np.cumsum(time_taken), label = 'Total time taken')
        plt.legend(loc = 'best')
        plt.grid()
        plt.xlabel("Users")
        plt.ylabel("Time taken in seconds")
        plt.show()
    return sparse.csr_matrix((data,(rows,cols)) , shape=(no_of_user , no_of_user)), time_taken
```

```
start = datetime.now()
u_u_sim, _ =compute_user_similarity(train_sparse_matrix , compute_for_few=True, verbose = True, top=100)
print("*"*100)
print("Time Taken : {}".format(datetime.now()-start))
```

```
Computing top 100 similar users
Computing done for 20 users [time elapsed : 0:00:49.004448 ]
Computing done for 40 users [time elapsed : 0:01:39.480072 ]
Computing done for 60 users [time elapsed : 0:02:26.607492 ]
Computing done for 80 users [time elapsed : 0:03:14.221261 ]
Computing done for 100 users [time elapsed : 0:04:01.479311 ]
Creating sparse matrix from the computes similarities.
```




```
****************************************************************************************************
Time Taken : 0:04:07.023896
```

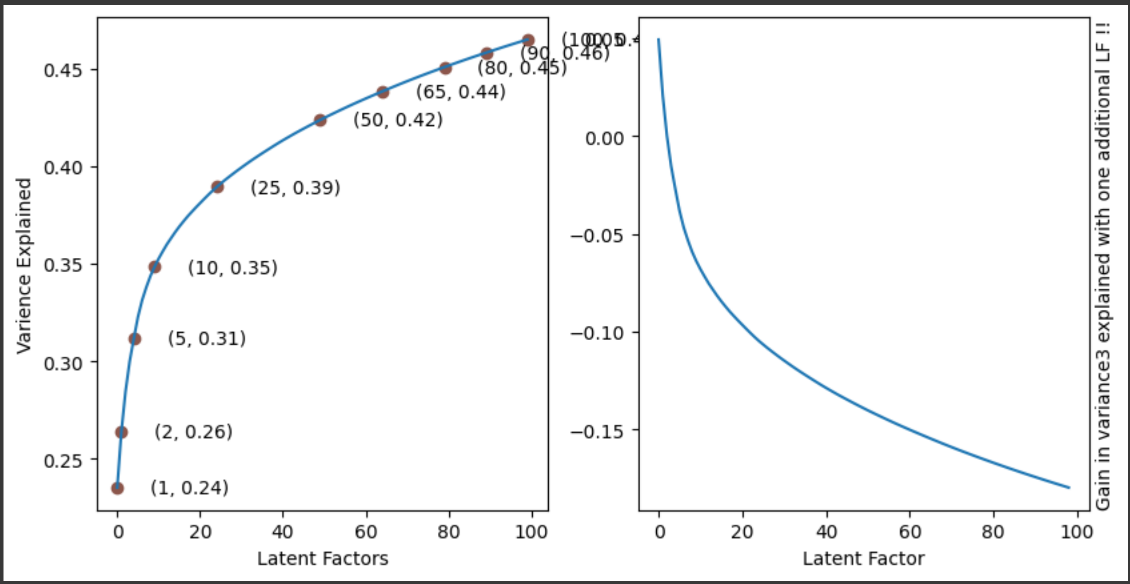
```
start = datetime.now()
svd = TruncatedSVD(n_components= 100, algorithm='randomized', random_state=15)
trunSvd = svd.fit_transform(train_sparse_matrix)
print("Time Taken: ", datetime.now()-start)
```

```
Time Taken:  0:07:33.430947
```

Overall, the below code appears to plot the variance explained (expl_var) on the first subplot and the gain in variance explained with one additional latent factor (change_in_expl_var) on the second subplot. Additionally, it adds annotations to specific points on the first subplot to provide additional information about the (latentfactors, expl_var) values at those points.

```
# An attribute that represents the ratio of the explained variance for each latent factor or component obtained from the SVD.
# It returns an array-like object containing the explained variance ratios in descending order.
exp_var = np.cumsum(svd.explained_variance_ratio_)
```

```
# This line creates a figure and two subplots arranged horizontally. ax1 and ax2 are the axes objects representing the two subplots.
fig, (ax1,ax2) = plt.subplots(nrows=1, ncols=2, figsize= plt.figaspect(.5))
ax1.set_ylabel("Varience Explained")
ax1.set_xlabel("Latent Factors")
ax1.plot(exp_var)
# Defines a list of indices to annotate on the plot.
index = [1,2,5,10,25,50,65,80,90,100]
#  Plots scatter points on the first subplot at the specified indices, with a custom color.
ax1.scatter(x= [i-1 for i in index] , y= exp_var[[i-1 for i in index]], c='#8c564b' )

# Annotates text on the plot to label specific points. It iterates over the indices in ind and adds an annotation with the corresponding
for i in index:
  text = "({}, {})".format(i, np.round(exp_var[i-1],2))
  # xytext = used to add annotations to specific points (Position) xytext = (i+7, exp_var[i-1] - 0.003)) play with these value for adjust
  ax1.annotate(text = text, xy = (i-1, exp_var[i-1]), xytext = (i+7, exp_var[i-1] - 0.003))

#  Calculates the change in expl_var for each adjacent pair of elements.
change_in_exp_var = [exp_var[i+1] - exp_var[i] for i in range(len(exp_var)-1)]


ax2.plot(change_in_exp_var)
ax2.set_xlabel("Latent Factor")
ax2.set_ylabel("Gain in variance3 explained with one additional LF !!")
ax2.yaxis.set_label_position("right")
plt.show()
```



More specifically, expl_var represents the cumulative sum of the explained variance ratios for each latent factor or component. The explained variance ratio quantifies the proportion of the total variance in the data that is explained by each latent factor. By summing up these ratios cumulatively, expl_var provides insight into the cumulative amount of variance explained as more latent factors or components are considered.

For example, if expl_var has values [0.2, 0.4, 0.6, 0.8, 1.0], it means that the first latent factor explains 20% of the total variance, the second latent factor explains an additional 20% (40% in total), the third factor explains another 20% (60% in total), and so on. The final value of 1.0 indicates that all the variance in the data has been accounted for by the latent factors considered.

```
for i in index:
  print("For Latent Factor {} , {}% of variance covered.".format(i, np.round(exp_var[i-1]*100, 2)))
```

```
For Latent Factor 1 , 23.54% of variance covered.
For Latent Factor 2 , 26.4% of variance covered.
For Latent Factor 5 , 31.17% of variance covered.
For Latent Factor 10 , 34.84% of variance covered.
For Latent Factor 25 , 38.93% of variance covered.
For Latent Factor 50 , 42.36% of variance covered.
For Latent Factor 65 , 43.82% of variance covered.
For Latent Factor 80 , 45.07% of variance covered.
For Latent Factor 90 , 45.8% of variance covered.
For Latent Factor 100 , 46.48% of variance covered.
```

```
start = datetime.now()
trun_matrix = train_sparse_matrix.dot(svd.components_.T)
print("Time taken: {}".format(datetime.now()- start))
```

```
Time taken: 0:00:13.458206
```

```
type(trun_matrix) , trun_matrix.shape
```

```
(numpy.ndarray, (2649430, 100))
```

Double-click (or enter) to edit

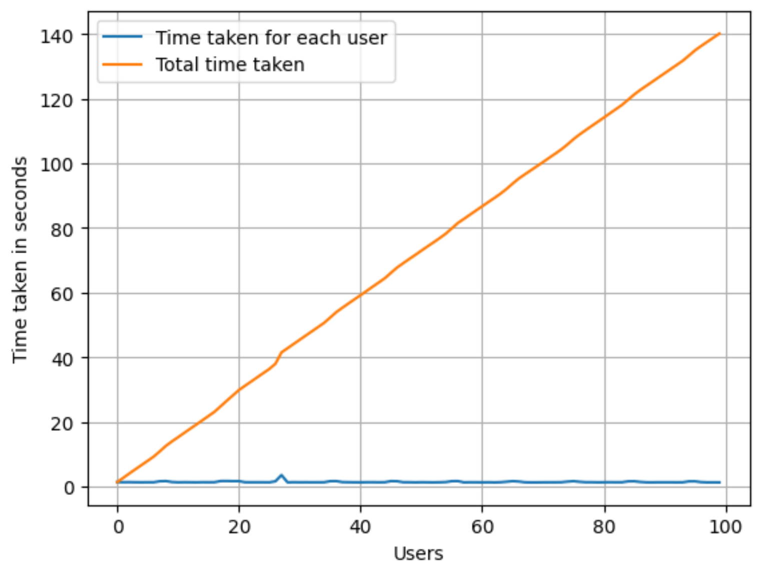
```
if os.path.isfile(dir_path + "trun_sparse_matrix.npz"):
  trun_sparse_matrix = sparse.load_npz(dir_path + "trun_sparse_matrix.npz")
else:
  trun_sparse_matrix = sparse.csr_matrix(trun_matrix)
  sparse.save_npz(dir_path + "trun_sparse_matrix.npz", trun_sparse_matrix)
```

```
print("The shape of Truncated SpareseMatrix is : ", trun_sparse_matrix.shape)
```

```
The shape of Truncated SpareseMatrix is :  (2649430, 100)
```

```
start = datetime.now()
trun_u_u_sim = compute_user_similarity(trun_sparse_matrix, verbose=True, compute_for_few = True, top=100, verbose_for_n_rows=10)
print("*"*100)
print("TimeTaken: ",datetime.now()-start)
```

```
Computing top 100 similar users
Computing done for 10 users [time elapsed : 0:00:13.957204 ]
Computing done for 20 users [time elapsed : 0:00:28.180195 ]
Computing done for 30 users [time elapsed : 0:00:44.112566 ]
Computing done for 40 users [time elapsed : 0:00:57.917799 ]
Computing done for 50 users [time elapsed : 0:01:11.672057 ]
Computing done for 60 users [time elapsed : 0:01:25.433140 ]
Computing done for 70 users [time elapsed : 0:01:39.119038 ]
Computing done for 80 users [time elapsed : 0:01:52.862884 ]
Computing done for 90 users [time elapsed : 0:02:06.608463 ]
Computing done for 100 users [time elapsed : 0:02:20.160029 ]
Creating sparse matrix from the computes similarities.
```




```
****************************************************************************************************
TimeTaken:  0:02:26.693408
```

Movie-Movie Similarity

```
start = datetime.now()
if not os.path.isfile(dir_path + "m_m_similarity_sparse.npz"):
  print("File not forund...")
  for i in range(4):
    print(".")
  print("Wait... We are computing file for you")
  m_m_similarity_sparse = cosine_similarity(train_sparse_matrix.T, dense_output=False)
  print("Computed Successfully")
  print("Saving the file in googledrive under netflix folder. ")
  sparse.save_npz(dir_path + "m_m_similarity_sparse.npz", m_m_similarity_sparse)
  print("Saved Successfully")
```

```
  print("The shape of movie-movie similarity sparse matrix is: ", m_m_similarity_sparse.shape)
  print("Time Taken to compute: ",datetime.now()-start)
else:
  m_m_similarity_sparse = sparse.load_npz(dir_path + "m_m_similarity_sparse.npz")
  print("File loaded successfully")
  print("The shape of movie-movie similarity sparse matrix is: ", m_m_similarity_sparse.shape)
  print("Time Taken to load: ",datetime.now()-start)
```

```
File loaded successfully
The shape of movie-movie similarity sparse matrix is:  (17771, 17771)
Time Taken to load:  0:00:46.278728
```

This line of code calculates the similarity scores for the current movie with all other movies, sorts the movies based on similarity (in descending order), and stores the indices of the most similar movies in the sim_movies array.

```
movie_ids = np.unique(m_m_similarity_sparse.nonzero()[1])
```

```
start = datetime.now()
top_sim_mvi = dict()
for movie in movie_ids:
  # .toarray() converts the selected row to a dense array format. This step is necessary because the subsequent operations require an arr
  # .ravel() flattens the 2D array to a 1D array, ensuring that the similarity scores are in a one-dimensional format.
  # .argsort() returns the indices that would sort the similarity scores in ascending order. Since we want the most similar movies, we ne
  # [::-1] reverses the sorted indices, effectively sorting the similarity scores in descending order.
  # [1:] slices the sorted indices starting from index 1, excluding the first element. This is done to exclude the current movie itself f
  similar_movies = m_m_similarity_sparse[movie].toarray().ravel().argsort()[: :-1][1:]
  top_sim_mvi[movie] = similar_movies[:100]
print("Time taken: ",datetime.now()-start)
# checking the top 100 similar movies for movie_id 29
top_sim_mvi[29]
```

```
Time taken:  0:00:29.067522
array([15405,  8162, 13625,  7843, 12824, 16071,  2513,  6155, 14249,
       11533,  9988,  5707, 12067,  5336,  1774,  2561,  1694, 14263,
        1460, 10405, 13608, 16033,  8562, 11906,  4599, 10423,  5351,
        6157, 16706,  6072,  5977,  7673,  9581, 11982, 11125, 13863,
       11588,  9643, 16250,  7242, 10125,  4366, 12138,  8855,  1234,
         522, 10613, 11858, 11245, 15604,  6005,  5118,  7579,  4219,
       10910, 11065,  7808,  2773, 12452, 13620, 14115,  4144,   202,
       15872,  3276, 17066,   517, 10846, 10448,  3515, 15625, 15515,
        9224,   883,  5400, 14457, 12313, 17614,  7972, 14221,   582,
        2383,  5016,   709,  5357, 17319, 15628,  7820,  8754,  2493,
         632, 14634,  7194,  9355, 16705, 15817, 12389, 13951,  1160,
       17248])
```

Reading the Movies titles

```
if os.path.isfile(dir_path + "movie_titles.csv"):
  print("Reading the data from google drive")
  movies_title = pd.read_csv(dir_path + "movie_titles.csv" , sep=',' , header=None, verbose=True,
                             names=['Movie_id', 'Movie_release_year', 'Movie_title'], index_col='Movie_id', encoding = 'ISO-8859-1')
else:
  print("movie_titles.csv not found kindly go to kaggle and download it.")
movies_title.head()
```

```
Reading the data from google drive
Tokenization took: 8.12 ms
Type conversion took: 10.75 ms
Parser memory cleanup took: 0.01 ms
```

	Movie_release_year	Movie_title
Movie_id		
1	2003.0	Dinosaur Planet
2	2004.0	Isle of Man TT 2004 Review
3	1997.0	Character
4	1994.0	Paula Abdul's Get Up & Dance
5	2004.0	The Rise and Fall of ECW

```
movies_title[movies_title['Movie_title'] == 'Titanic']
```

	Movie_release_year	Movie_title
Movie_id		
3124	1953.0	Titanic

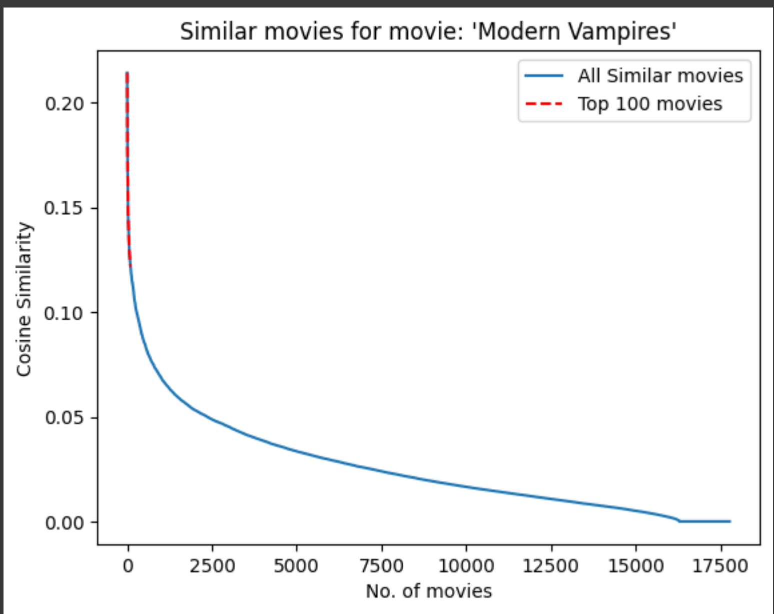
```
mvi_id = 323
print("Movie : {}\n".format(movies_title.loc[323].values[1]))
print("The movie have {} ratings from users.\n".format(train_sparse_matrix[:,mvi_id].getnnz()))
print("There are total {} movies that are similar with '{}' movie but we will see the top fews similar movies.".format(m_m_similarity_spa
```

```
Movie : Modern Vampires

The movie have 583 ratings from users.

There are total 16285 movies that are similar with 'Modern Vampires' movie but we will see the top fews similar movies.
```

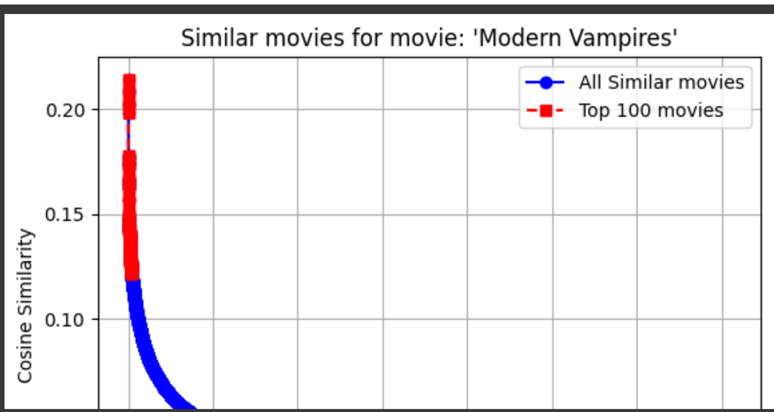
```
similarities = m_m_similarity_sparse[mvi_id].toarray().ravel()
similar_index = similarities.argsort()[: : -1] [1:]
similarities[similar_index]
sim_indices = similarities.argsort()[ : : -1][1:]
```

Double-click (or enter) to edit

```
mvi_id = 323
plt.plot(similarities[sim_indices], label="All Similar movies", linestyle='solid')
plt.plot(similarities[sim_indices[:100]], label="Top 100 movies", color='red', linestyle='dashed')
plt.xlabel("No. of movies")
plt.ylabel("Cosine Similarity")
plt.title("Similar movies for movie: '{}'".format(movies_title.loc[mvi_id].values[1]))
plt.legend()
plt.show()
```




```
mvi_id = 323
plt.plot(similarities[sim_indices], label="All Similar movies", linestyle='-', marker='o', color='blue')
plt.plot(similarities[sim_indices[:100]], label="Top 100 movies", linestyle='--', marker='s', color='red')
plt.xlabel("No. of movies")
plt.ylabel("Cosine Similarity")
plt.title("Similar movies for movie: '{}'".format(movies_title.loc[mvi_id].values[1]))
plt.legend()
plt.grid()
plt.show()
```



Thig plot will show that the top 100 movies for movie title "Modern Vampires" have 23% to 13% similarities.

```
print("Top 10 movies similar to ' Modern Vampires'")
movies_title.loc[sim_indices[:10]]
```

```
Top 10 movies similar to ' Modern Vampires'
```

Movie_id	Movie_release_year	Movie_title
4667	1996.0	Vampirella
15237	2001.0	The Forsaken
67	1997.0	Vampire Journals
16279	2002.0	Vampires: Los Muertos
13873	2001.0	The Breed
4173	1998.0	From Dusk Till Dawn 2: Texas Blood Money
1900	1997.0	Club Vampire
13962	2001.0	Dracula: The Dark Prince
15867	2003.0	Dracula II: Ascension
3496	1998.0	Vampires

```
def get_sample_sparse_matrix(sparse_matrix, no_user, no_movie, path ,verbose=True):
  row_index, col_index, rating = sparse.find(sparse_matrix)
  users = np.unique(row_index)
  movies = np.unique(col_index)
  print("Shape of matrix before sampling: ({} X {})".format(len(users) , len(movies)))
  print("Ratings before sampling: ",len(rating))

  np.random.seed(20)
  sample_users = np.random.choice(users, no_user, replace=False)
  sample_movies= np.random.choice(movies, no_movie , replace=False)
  mask = np.logical_and(np.isin(row_index , sample_users), np.isin(col_index, sample_movies))

  sample_sparse_matrix = sparse.csr_matrix((rating[mask] , (row_index[mask] , col_index[mask])), shape= (max(sample_users)+1, max(sample_

  if verbose:
    print("Shape of matrix after sampling: ({} X {})".format(len(sample_users), len(sample_movies)))
    print("Ratings after sampling: ",rating[mask].shape[0])

  print("Saving the matrix into drive.")
  sparse.save_npz(path , sample_sparse_matrix)
  if verbose:
    print("Save successfully")

  return sample_sparse_matrix
```

```
path = dir_path + "train_sample_sparse_matrix.npz"
if not os.path.isfile(path):
  print("Creating the Train sample sparse matrix. \n\n Please Wait")
  train_sample_sparse_matrix = get_sample_sparse_matrix(train_sparse_matrix, no_user = 10000, no_movie=1000, path=path)
else:
  print("Loading the Sample Matrix. \n\n Please Wait")
  train_sample_sparse_matrix = sparse.load_npz(path)
```

```
Loading the Sample Matrix.

 Please Wait
```

```
path = dir_path + "cv_sample_sparse_matrix.npz"
if not os.path.isfile(path):
  print("Creating the CV sample sparse matrix. \n\n Please Wait")
  cv_sample_sparse_matrix = get_sample_sparse_matrix(cv_sparse_matrix, no_user = 5000, no_movie=500, path=path)
else:
  print("Loading the Sample Matrix. \n\n Please Wait")
  cv_sample_sparse_matrix = sparse.load_npz(path)
```

```
Loading the Sample Matrix.

 Please Wait
```

```
path = dir_path + "test_sample_sparse_matrix.npz"
if not os.path.isfile(path):
  print("Creating the Test sample sparse matrix. \n\n Please Wait")
  test_sample_sparse_matrix = get_sample_sparse_matrix(test_sparse_matrix, no_user = 5000, no_movie=500, path=path)
else:
  print("Loading the Sample Matrix. \n\n Please Wait")
  test_sample_sparse_matrix = sparse.load_npz(path)
```

```
Loading the Sample Matrix.

 Please Wait
```

Featurizing

```
sample_train_dataset = dict()
average_global_rating = train_sample_sparse_matrix.sum()  / train_sample_sparse_matrix.count_nonzero()
sample_train_dataset['global'] = average_global_rating
sample_train_dataset
```

```
{'global': 3.545324133472072}
```

```
user = 311465
sample_train_dataset['user'] = get_average_ratings(train_sample_sparse_matrix, of_user=True)
print("Average rating of user {} is: {}".format(user,sample_train_dataset['user'][user]))
```

```
Average rating of user 311465 is: 4.0
```

```
'''Movie_release_year    1964.0
   Movie_title           Marnie
   Name: 17109, dtype: object'''
movieid= 17109
sample_train_dataset['movie'] = get_average_ratings(train_sample_sparse_matrix, of_user=False)
print("Average rating for movie '{}' is: {}".format(movies_title.iloc[17109 -1].values[1] ,sample_train_dataset['movie'][movieid] ))
```

```
Average rating for movie 'Marnie' is: 3.4927536231884058
```

```
train_sample_user, train_sample_movie,train_sample_rating = sparse.find(train_sparse_matrix)
```

```
#user_sim = cosine_similarity(sample_train_sparse_matrix[user], sample_train_sparse_matrix).ravel():
#This line calculates the cosine similarity between the user (represented by the index user) and all other users in the
#sample_train_sparse_matrix. The result is flattened using ravel() to create a 1D array.

# top_sim_users = user_sim.argsort()[::-1][1:]: Here, argsort() sorts the similarity scores in ascending order and returns the indices
 # that would sort the array. By using [::-1], it reverses the order, resulting in indices that sort the array in descending order. [1:]
 # is used to exclude the first element, which corresponds to the user itself. Thus, top_sim_users contains the indices of users that are

#top_ratings = sample_train_sparse_matrix[top_sim_users, movie].toarray().ravel(): This line retrieves the ratings of the most similar us
 #for the given movie. It uses the top_sim_users indices and the movie index to access the corresponding ratings from the
 #sample_train_sparse_matrix. The toarray() method converts the sparse matrix slice to a dense array, and ravel() flattens it to a 1D arr

#top_sim_users_ratings = list(top_ratings[top_ratings != 0][:5]): Here, top_ratings[top_ratings != 0] filters out any zero ratings from
#the top_ratings array, and [:5] selects at most the first five non-zero ratings. This ensures that we have a maximum of five ratings fro
#similar users for the movie.

#top_sim_users_ratings.extend([sample_train_averages['movie'][movie]]*(5 - len(top_sim_users_ratings))): This line extends the
#top_sim_users_ratings list by appending the average rating of the movie (sample_train_averages['movie'][movie])
#repeated 5 - len(top_sim_users_ratings) times. It ensures that if there are fewer than five ratings from similar users, the remaining sl
#are filled with the average rating.
start = datetime.now()
if os.path.isfile(dir_path + "train_regression_data.csv"):
  print("Train Regression Dataset found in the google drive.\nReading...")
  train_regression_data = pd.read_csv(dir_path + "train_regression_data.csv")
  print("Load Successfully and time taken: {}".format(datetime.now()-start))
else:
  print("Preparing {} tuples\nKindly wait...".format(len(train_sample_rating)))
```

```
  with open(dir_path + "train_regression_data.csv" , mode = 'w') as train_regression_data_file:
    count = 0

    for user , movie, rating in zip(train_sample_user, train_sample_movie, train_sample_rating):

      st = datetime.now()
      # Finding 5 most similar users.
      user_similarity = cosine_similarity(train_sample_sparse_matrix[user],train_sample_sparse_matrix).ravel()
      top_sim_user = user_similarity.argsort()[::-1][1:]
      # top_sim_user:  (2648546,)
      top_rating   = train_sample_sparse_matrix[top_sim_user , movie].toarray().ravel()
      top_sim_user_rating = list(top_rating[top_rating != 0][:5])
      top_sim_user_rating.extend([sample_train_dataset['movie'][movie]]* (5- len(top_sim_user_rating)))

      # Finding 5 most similar movies.
      movie_similarity = cosine_similarity(train_sample_sparse_matrix[:,movie].T,train_sample_sparse_matrix.T).ravel()
      top_sim_movies = movie_similarity.argsort()[::-1][1:]
      # top_sim_movies:  (17749,)
      top_rating  = train_sample_sparse_matrix[ user ,top_sim_movies].toarray().ravel()
      top_sim_movie_rating = list(top_rating[top_rating != 0][:5])
      top_sim_movie_rating.extend([sample_train_dataset['user'][user]]* (5-len(top_sim_movie_rating)))

      # Preparing the row by inserting values.
      row = []
      row.append(user)
      row.append(movie)
      row.append(sample_train_dataset['global'])
      row.extend(top_sim_user_rating)
      row.extend(top_sim_movie_rating)
      row.append(sample_train_dataset['user'][user])
      row.append(sample_train_dataset['movie'][movie])
      row.append(rating)
      count += 1

      train_regression_data_file.write(','.join(map(str,row)))
      train_regression_data_file.write('\n')

      if count% 10000 == 0:
        print("Done for {} rows and time elapse: {}".format(count,datetime.now()-start))

print("Total time taken: {}".format(datetime.now()-start))
```

```
Preparing 98987 tuples
Kindly wait...
Done for 10000 rows and time elapse: 0:46:04.334337
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-36-6970ebdb1461> in <cell line: 22>()
     34       # Finding 5 most similar users.
     35       user_similarity = cosine_similarity(train_sample_sparse_matrix[user],train_sample_sparse_matrix).ravel()
---> 36       top_sim_user = user_similarity.argsort()[::-1][1:]
     37       # top_sim_user:  (2648546,)
     38       top_rating   = train_sample_sparse_matrix[top_sim_user , movie].toarray().ravel()

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

1. The prepare_regression_data function takes three lists as input: train_sample_user, train_sample_movie, and train_sample_rating. These lists represent the user, movie, and rating data, respectively.
2. The function first checks if the train_regression_data.csv file already exists. If it does, it reads the file using pd.read_csv and skips the processing step.
3. If the train_regression_data.csv file doesn't exist, it proceeds with the data preparation. The function creates a partial function called partial_process_tuple using the partial function from the functools module. This partial function fixes the first three arguments of the partial_process_tuple function, which are user, movie, and rating.
4. The partial_process_tuple function performs the data processing for each tuple of user, movie, and rating. It calculates the cosine similarity between the user and all other users (user_similarity) and selects the top similar users (top_sim_users). It then retrieves the ratings of the top similar users for the given movie (top_ratings) and filters out zero ratings. Similarly, it calculates the cosine similarity between the movie and all other movies (movie_similarity) and selects the top similar movies (top_sim_movies). It retrieves the ratings of the top similar movies for the given user and filters out zero ratings.
5. The processed values are appended to the row list, which represents a row in the train_regression_data.csv file. The values in the row list are written to the file using the write method.

6. The prepare_regression_data function then creates a Pool object, which represents a pool of worker processes. By default, the number of worker processes is determined by the number of CPU cores available.
7. The starmap method of the Pool object is used to apply the partial_process function to each tuple of train_sample_user, train_sample_movie, and train_sample_rating. The starmap function distributes the processing of tuples among the worker processes in parallel.
8. After the parallel processing is completed, the close method is called on the Pool object to prevent any new tasks from being submitted to the pool. Then, the join method is called to wait for all the worker processes to complete.
9. Finally, the total time taken for the data preparation process is printed.

By using parallel processing with the multiprocessing module, the code distributes the processing of tuples across multiple CPU cores, potentially speeding up the overall execution time compared to sequential processing.

```
import os
import pandas as pd
from datetime import datetime
from sklearn.metrics.pairwise import cosine_similarity
from functools import partial
from multiprocessing import Pool



def partial_process_tuple(user, movie, rating):
    # Open the file within the process
    with open(dir_path + "train_regression_data.csv", mode='a') as train_regression_data_file:
        # Finding 5 most similar users.
        user_similarity = cosine_similarity(train_sample_sparse_matrix[user], train_sample_sparse_matrix).ravel()
        top_sim_users = user_similarity.argsort()[::-1][1:]
        top_ratings = train_sample_sparse_matrix[top_sim_users, movie].toarray().ravel()
        top_sim_users_ratings = list(top_ratings[top_ratings != 0][:5])
        top_sim_users_ratings.extend([sample_train_dataset['movie'][movie]] * (5 - len(top_sim_users_ratings)))

        # Finding 5 most similar movies.
        movie_similarity = cosine_similarity(train_sample_sparse_matrix[:, movie].T, train_sample_sparse_matrix.T).ravel()
        top_sim_movies = movie_similarity.argsort()[::-1][1:]
        top_ratings = train_sample_sparse_matrix[user, top_sim_movies].toarray().ravel()
        top_sim_movies_ratings = list(top_ratings[top_ratings != 0][:5])
        top_sim_movies_ratings.extend([sample_train_dataset['user'][user]] * (5 - len(top_sim_movies_ratings)))

        # Preparing the row by inserting values.
        row = []
        row.append(user)
        row.append(movie)
        row.append(sample_train_dataset['global'])
        row.extend(top_sim_users_ratings)
        row.extend(top_sim_movies_ratings)
        row.append(sample_train_dataset['user'][user])
        row.append(sample_train_dataset['movie'][movie])
        row.append(rating)

        train_regression_data_file.write(','.join(map(str, row)))
        train_regression_data_file.write('\n')

def prepare_regression_data(train_sample_user, train_sample_movie, train_sample_rating):
    if os.path.isfile(dir_path + "train_regression_data.csv"):
        print("Train Regression Dataset found in the Google Drive.\nReading...")
        train_regression_data = pd.read_csv(dir_path + "train_regression_data.csv")
        print("Load Successful and time taken: {}".format(datetime.now() - start))
    else:
        print("Preparing {} tuples\nKindly wait...".format(len(train_sample_rating)))

        # Define the partial function
        partial_process = partial(partial_process_tuple)

        # Create a pool of workers
        pool = Pool()

        # Apply parallel processing
        pool.starmap(partial_process, zip(train_sample_user, train_sample_movie, train_sample_rating))

        pool.close()
        pool.join()

        print("Total time taken: {}".format(datetime.now() - start))

start = datetime.now()
prepare_regression_data(train_sample_user, train_sample_movie, train_sample_rating)
```

```
Preparing 98987 tuples
Kindly wait...
```

```
test_sample_user, test_sample_movie, test_sample_rating = sparse.find(test_sample_sparse_matrix)
```

```
start = datetime.now()
if os.path.isfile(dir_path + "test_regression_data.csv"):
  print("Test Regression Data found in google drive.\n reading data...")
  test_regression_data = pd.read_csv(dir_path + "test_regression_data.csv")
  print("Data loading successfully")

else:
  print("Test Regression Data not found in google drive.\nPreparing tuple {}\nKindly Wait...".format(len(test_sample_rating)))
  with open(dir_path + "test_regression_data.csv", mode='w') as regression_data_file:
    count = 0
```

```
    for user, movie, rating in zip(test_sample_user, test_sample_movie, test_sample_rating):
      st = datetime.now()
      try:
        user_similarity = cosine_similarity(train_sample_sparse_matrix[user], train_sample_sparse_matrix).ravel()
        top_sim_user = user_similarity.argsort()[::-1][1:]
        top_rating = train_sample_sparse_matrix[top_sim_user, movie].toarray().ravel()
        top_sim_user_rating = list(top_rating[top_rating != 0][:5])
        top_sim_user_rating.extend([sample_train_dataset['movie'][movie]]*(5 - len(top_sim_user_rating)))

      except(IndexError, KeyError):
        top_sim_user_rating.extend([sample_train_dataset['global']] * (5 - len(top_sim_user_rating)))

      except:
        print(user,movie)
        raise

      try:
        movie_similarity = cosine_similarity(train_sample_sparse_matrix[:,movie].T, train_sample_sparse_matrix.T).ravel()
        top_sim_movie = movie_similarity.argsort()[::-1][1:]
        top_rating = train_sample_sparse_matrix[user, top_sim_movie].toarray().ravel()
        top_sim_movie_rating = list(top_rating[top_rating!= 0][:5])
        top_sim_movie_rating.extend([sample_train_dataset['user'][user]] * (5 - len(top_sim_movie_rating)))

      except(IndexError, KeyError):
        top_sim_movie_rating.extend([sample_train_dataset['global']] * (5- len(top_sim_movie_rating)))

      except:
        print(user,rating)
        raise

      row =[]
      row.append(user)
      row.append(movie)
      row.append(sample_train_dataset['global'])
      row.extend(top_sim_user_rating)
      row.extend(top_sim_movie_rating)

      try:
        row.append(sample_train_dataset['user'][user])
      except(KeyError):
        row.append(sample_train_dataset['global'])

      try:
        row.append(sample_train_dataset['movie'][movie])
      except(KeyError):
        row.append(sample_train_dataset['global'])

      row.append(rating)
      count += 1
      regression_data_file.write(','.join(map(str,row)))
      regression_data_file.write('\n')
      if count%1000 == 0:
        print("Done for {} row... Time elapse {}".format(count, datetime.now()- start))
  print("Total time taken {}".format(datetime.now()-start))
```

```
Test Regression Data not found in google drive.
Preparing tuple 7257
Kindly Wait...
Done for 1000 row... Time elapse 0:04:15.966295
Done for 2000 row... Time elapse 0:08:25.964561
Done for 3000 row... Time elapse 0:12:32.473567
Done for 4000 row... Time elapse 0:16:42.830608
Done for 5000 row... Time elapse 0:20:49.653053
Done for 6000 row... Time elapse 0:24:59.295137
Done for 7000 row... Time elapse 0:29:08.424005
Total time taken 0:30:13.720584
```

For Cross Validate

```
cv_sample_user, cv_sample_movie, cv_sample_rating = sparse.find(cv_sample_sparse_matrix)
```

```
start = datetime.now()
if os.path.isfile(dir_path + "cv_regression_data.csv"):
  print("CrossValidate Regression Data found in google drive.\n reading data...")
  test_regression_data = pd.read_csv(dir_path + "cv_regression_data.csv")
  print("Data loading successfully")

else:
  print("CrossValidate Regression Data not found in google drive.\nPreparing tuple {}\nKindly Wait...".format(len(cv_sample_rating)))
  with open(dir_path + "cv_regression_data.csv", mode='w') as regression_data_file:
    count = 0
```

```
    for user, movie, rating in zip(cv_sample_user, cv_sample_movie, cv_sample_rating):
      st = datetime.now()
      try:
        user_similarity = cosine_similarity(train_sample_sparse_matrix[user], train_sample_sparse_matrix).ravel()
        top_sim_user = user_similarity.argsort()[::-1][1:]
        top_rating = train_sample_sparse_matrix[top_sim_user, movie].toarray().ravel()
        top_sim_user_rating = list(top_rating[top_rating != 0][:5])
        top_sim_user_rating.extend([sample_train_dataset['movie'][movie]]*(5 - len(top_sim_user_rating)))

      except(IndexError, KeyError):
        top_sim_user_rating.extend([sample_train_dataset['global']] * (5 - len(top_sim_user_rating)))

      except:
        print(user,movie)
        raise

      try:
        movie_similarity = cosine_similarity(train_sample_sparse_matrix[:,movie].T, train_sample_sparse_matrix.T).ravel()
        top_sim_movie = movie_similarity.argsort()[::-1][1:]
        top_rating = train_sample_sparse_matrix[user, top_sim_movie].toarray().ravel()
        top_sim_movie_rating = list(top_rating[top_rating!= 0][:5])
        top_sim_movie_rating.extend([sample_train_dataset['user'][user]] * (5 - len(top_sim_movie_rating)))

      except(IndexError, KeyError):
        top_sim_movie_rating.extend([sample_train_dataset['global']] * (5- len(top_sim_movie_rating)))

      except:
        print(user,rating)
        raise

      row =[]
      row.append(user)
      row.append(movie)
      row.append(sample_train_dataset['global'])
      row.extend(top_sim_user_rating)
      row.extend(top_sim_movie_rating)

      try:
        row.append(sample_train_dataset['user'][user])
      except(KeyError):
        row.append(sample_train_dataset['global'])

      try:
        row.append(sample_train_dataset['movie'][movie])
      except(KeyError):
        row.append(sample_train_dataset['global'])

      row.append(rating)
      count += 1
      regression_data_file.write(','.join(map(str,row)))
      regression_data_file.write('\n')
      if count%1000 == 0:
        print("Done for {} row... Time elapse {}".format(count, datetime.now()- start))
  print("Total time taken {}".format(datetime.now()-start))
```

```
CrossValidate Regression Data not found in google drive.
Preparing tuple 11926
Kindly Wait...
Done for 1000 row... Time elapse 0:03:58.587465
Done for 2000 row... Time elapse 0:07:55.910899
Done for 3000 row... Time elapse 0:11:52.203224
Done for 4000 row... Time elapse 0:15:43.544197
Done for 5000 row... Time elapse 0:19:33.789241
Done for 6000 row... Time elapse 0:23:24.642801
Done for 7000 row... Time elapse 0:27:13.917863
Done for 8000 row... Time elapse 0:31:03.895687
Done for 9000 row... Time elapse 0:34:53.763637
Done for 10000 row... Time elapse 0:38:43.891820
Done for 11000 row... Time elapse 0:42:30.181282
Total time taken 0:45:57.811517
```

Reading Regression dataset for Train

```
train_regression_data = pd.read_csv(dir_path + "train_regression_data.csv", names=['user','movie','GAvg', 'sur1','sur2','sur3','sur4','su
                                                                                'smr1','smr2','smr3','smr4','smr5','UAvg','MAvg','rating']
train_regression_data.head()
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	5.0	3.0	1.0	3.370370	4.092437	4
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	4.0	3.0	5.0	3.555556	4.092437	3
2	99865	33	3.581679	5.0	5.0	4.0	5.0	3.0	5.0	4.0	4.0	5.0	4.0	3.714286	4.092437	5

Reading Regression dataset for CrossValidate

```
cv_regression_data = pd.read_csv(dir_path + "cv_regression_data.csv", names=['user','movie','GAvg', 'sur1','sur2','sur3','sur4','sur5',
                                                                               'smr1','smr2','smr3','smr4','smr5','UAvg','MAvg','rating'
cv_regression_data.head()
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg
0	543665	91	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324
1	1932594	91	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324
2	2321468	91	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324
3	14936	241	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324
4	44518	241	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324

Reading Regression dataset for Test

```
test_regression_data = pd.read_csv(dir_path + "test_regression_data.csv", names=['user','movie','GAvg', 'sur1','sur2','sur3','sur4','sur5
                                                                               'smr1','smr2','smr3','smr4','smr5','UAvg','MAvg','rating'
test_regression_data.head()
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg
0	256033	24	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324
1	1735827	24	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324
2	1804811	24	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324
3	2485642	24	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324
4	2308181	38	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324

Overview of Surprise Library

The Surprise library uses a specific data format called the Surprise Dataset format. This format is designed to facilitate the use of various recommender algorithms provided by the library.

The Surprise Dataset format represents the user-item ratings data in a structured manner that can be easily processed by the library's algorithms. It consists of three main components: users, items, and ratings.

Here's a brief overview of the format:

1. Users: Users are identified by unique user IDs. Each user ID corresponds to a set of ratings given by that user.
2. Items: Items (or items being recommended) are identified by unique item IDs. Each item ID corresponds to a set of ratings received by that item.
3. Ratings: Ratings represent the user-item interactions or preferences. They typically indicate how a user rates or interacts with an item. Ratings can be numerical, binary (e.g., liked/disliked), or explicit/implicit.

The Surprise Dataset format is often constructed using the Dataset class provided by the library. It offers methods like load_from_df, load_from_file, or load_builtin to load data from dataframes, files, or built-in datasets, respectively. These methods convert the data into the Surprise Dataset format.

Once the data is loaded into the Surprise Dataset format, it can be further processed, split into train and test sets, and used as input for training and evaluating recommender algorithms provided by the Surprise library.

```
!pip install surprise
from surprise import Reader, Dataset
reader    = Reader(rating_scale=(1,5))
dataset   = Dataset.load_from_df(train_regression_data[['user','movie','rating']], reader)
train_set = dataset.build_full_trainset()
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: surprise in /usr/local/lib/python3.10/dist-packages (0.1)
Requirement already satisfied: scikit-surprise in /usr/local/lib/python3.10/dist-packages (from surprise) (1.1.3)
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->surprise) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->surprise) (1.22.4)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->surprise) (1.10.1)
```

```
cv_set = list(zip(cv_regression_data.user.values , cv_regression_data.movie.values , cv_regression_data.rating.values))
cv_set[:3]
```

```
[(543665, 91, 2), (1932594, 91, 2), (2321468, 91, 5)]
```

```
test_set = list(zip(test_regression_data.user.values , test_regression_data.movie.values , test_regression_data.rating.values))
test_set[:3]
```

```
[(256033, 24, 4), (1735827, 24, 4), (1804811, 24, 3)]
```

Applying Machine Learning Models

These dictionaries are used as the set of RMSE in Train, cv, test sets for all the machine learning model.

```
train_model_evaluation = {}
cv_model_evaluation = {}
test_model_evaluation = {}
train_model_evaluation, cv_model_evaluation, test_model_evaluation
```

```
({}, {}, {})
```

Utility functions for XGBoost Regression

```
def get_error_matrix(y_true, y_predict):
  rmse = np.sqrt(np.mean([ (y_true[i] - y_predict[i])**2 for i in range(len(y_predict))])) # Root Mean Square Error
  mape = np.mean(np.abs((y_true - y_predict)/y_true)) *100    # Mean Absolute Percentage Error
  return rmse , mape

def run_xgboost(algo, x_train, y_train, x_test, y_test, verbose=True):
  train_result = {}
  test_result  = {}

  print("Training the model")
  start = datetime.now()
  algo.fit(x_train, y_train , eval_metric = 'rmse')
  print("Done\n Time Taken: ",datetime.now()-start)

  print("Evaluating the model on Train data")
  start = datetime.now()
  y_train_predict = algo.predict(x_train)
  train_rmse, train_mape  = get_error_matrix(y_train.values,y_train_predict)

  train_result = { 'rmse': train_rmse ,
                   'mape': train_mape ,
                   'y_predict': y_train_predict}

  print("Evaluating the model on Test data")
  y_test_predict = algo.predict(x_test)
  test_rmse, test_mape = get_error_matrix(y_test, y_test_predict)

  test_result = { 'rmse': test_rmse ,
                  'mape': test_mape ,
                  'y_predict':y_test_predict}


  if verbose:
    print("*"*100)
    print("Train Data")
    print("RMSE: ",train_rmse)
    print("MAPE: ",train_mape)
    print("\n")
    print("*"*100)
    print("Test Data")
    print("RMSE: ",test_rmse)
    print("MAPE: ",test_mape)

    return train_result , test_result
```

Utility Function for Surprise Model

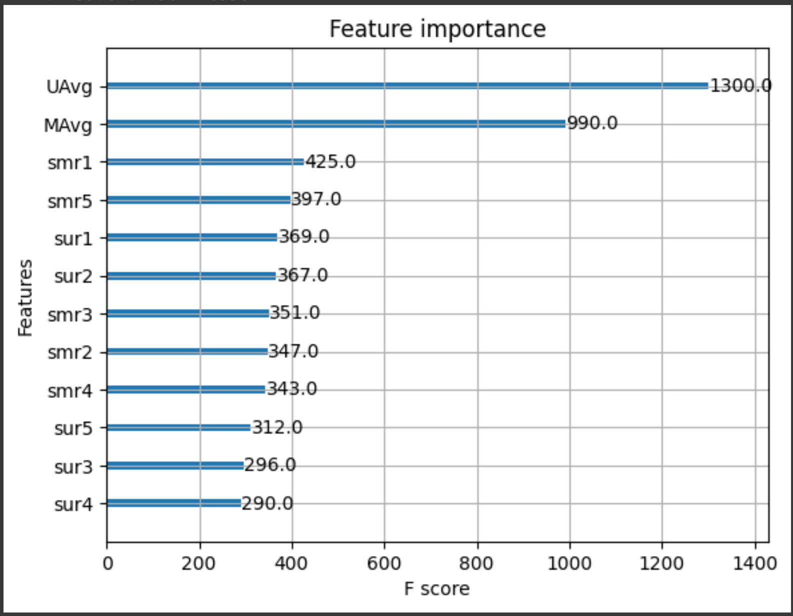
```
my_seed = 15
# random.seed(my_seed)
np.random.seed(my_seed)

def get_rating(prediction):
  actual = np.array([pred.r_ui for pred in prediction])
  predict = np.array([pred.est for pred in prediction])
  return actual,predict

def get_errors(prediction, print_them=False):
  actual , predict = get_rating(prediction)
  rmse = np.sqrt(np.mean((predict - actual)**2))
  mape = np.mean(np.abs(predict - actual)/actual)*100
  return rmse , mape

def run_surprise(algo, train_set , test_set, verbose=True):
  start = datetime.now()

  train_result = {}
  test_result = {}
  print("Training the surprise model")
  algo.fit(train_set)
  print("Done\n Time Taken: ",datetime.now()-start)

  # Evaluating Train set
  st = datetime.now()
  print("Evaluting the Train set")
  train_predict = algo.test(train_set.build_testset())
  train_actual_rating , train_predict_rating = get_rating(train_predict)
  train_rmse , train_mape = get_errors(train_predict)
  print("Time Taken: ",datetime.now()-st)

  train_result = { 'rmse': train_rmse,
                   'mape': train_mape,
                   'y_predict': train_predict_rating}

  if verbose:
    print("*"*100)
    print("Train Data")
    print("RMSE: ",train_rmse)
    print("MAPE: ",train_mape)

  # Evaluating Test set
  st = datetime.now()
  print("Evaluting the Test set")
  test_predict = algo.test(test_set)
  test_actual_rating , test_predict_rating = get_rating(test_predict)
  test_rmse , test_mape = get_errors(test_predict)
  print("Time Taken: ",datetime.now()-st)

  test_result = { 'rmse': test_rmse,
                  'mape': test_mape,
                  'y_predict': test_predict_rating}

  if verbose:
    print("*"*100)
    print("Test Data")
    print("RMSE: ",test_rmse)
    print("MAPE: ",test_mape)
  print("\n")
  print("~"*100)
  print("Total Time taken to run the algo: ",datetime.now()-start)
  return train_result, test_result
```

First XGBoost model with 13 handcraft features

```
x_train = train_regression_data.drop(['user', 'movie','rating'], axis = 1)
y_train  = train_regression_data['rating']

x_test = test_regression_data.drop(['user','movie','rating'], axis = 1)
y_test = test_regression_data['rating']

first_xgb = xgb.XGBRegressor(n_job = -1, random_state=15, n_estimator=100, silence= False)
train_result, test_result = run_xgboost(first_xgb, x_train, y_train, x_test, y_test)
train_model_evaluation['first_algo'] = train_result
```

```
test_model_evaluation['first_algo'] = test_result
xgb.plot_importance(first_xgb)
plt.show()
```

```
Training the model
[17:04:05] WARNING: ../src/learner.cc:767:
Parameters: { "n_estimator", "n_job", "silence" } are not used.

Done
 Time Taken:  0:01:12.762199
Evaluating the model on Train data
Evaluating the model on Test data
*****************************************************************************************************
Train Data
RMSE:  0.7933884025295052
MAPE:  23.188514838567027


*****************************************************************************************************
Test Data
RMSE:  1.2192864215756742
MAPE:  33.67817369218556
```



Baseline Surprise model

```
bsl_options = {'method':'sgd', 'learning_rate': 0.001}
bsl_algo = BaselineOnly(bsl_options=bsl_options)
bsl_train_result , bsl_test_result = run_surprise(bsl_algo, train_set, test_set, verbose=True)
train_model_evaluation['bsl_algo'] = train_result
test_model_evaluation['bsl_algo'] = test_result
```

```
Training the surprise model
Estimating biases using sgd...
Done
 Time Taken:  0:00:00.906051
Evaluting the Train set
Time Taken:  0:00:01.425750
*****************************************************************************************************
Train Data
RMSE:  0.9347153928678286
MAPE:  29.389572652358183
Evaluting the Test set
Time Taken:  0:00:00.060378
*****************************************************************************************************
Test Data
RMSE:  1.1060300803275174
MAPE:  35.85404178528604


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Total Time taken to run the algo:  0:00:02.396515
```

Building model with 13 Features + BaselineOnly

```
train_regression_data['bslpre'] = train_model_evaluation['bsl_algo']['y_predict']
train_regression_data.head(3)
```

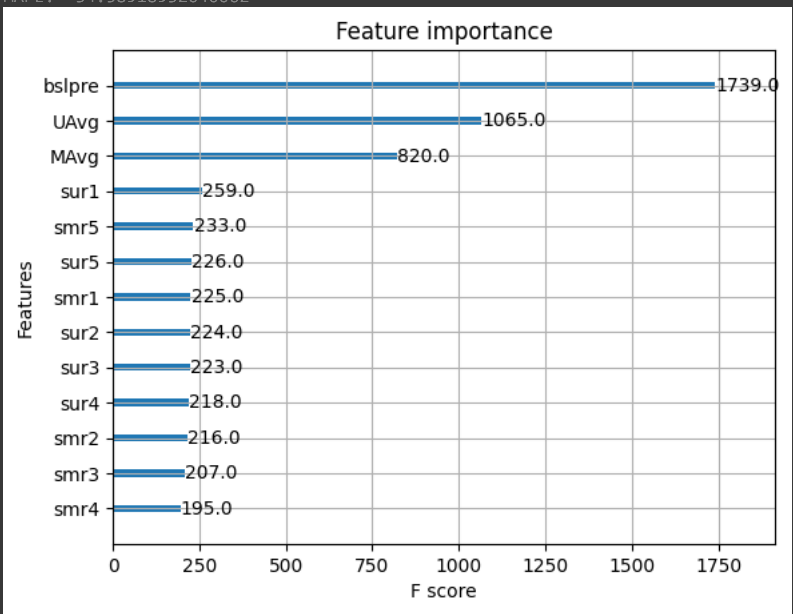
	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	bslpre
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	5.0	3.0	1.0	3.370370	4.092437	4	3.919560
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	4.0	3.0	5.0	3.555556	4.092437	3	4.078834
2	99865	33	3.581679	5.0	5.0	4.0	5.0	3.0	5.0	4.0	4.0	5.0	4.0	3.714286	4.092437	5	4.376114

```
test_regression_data['bslpre'] = test_model_evaluation['bsl_algo']['y_predict']
test_regression_data.head(3)
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg
0	256033	24	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324
1	1735827	24	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324
2	1804811	24	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324

```
from xgboost.sklearn import XGBRegressor
x_train = train_regression_data.drop(['user', 'movie', 'rating'],axis=1)
y_train = train_regression_data['rating']

x_test = test_regression_data.drop(['user', 'movie', 'rating'],axis=1)
y_test = test_regression_data['rating']

second_xgb = xgb.XGBRegressor(n_job=-1, n_estimator=100, silence=False, random_state=15)
train_result, test_result = run_xgboost(second_xgb, x_train, y_train, x_test, y_test)

train_model_evaluation['second_xgb_bsl'] = train_result
test_model_evaluation['second_xgb_bsl']  = test_result

xgb.plot_importance(second_xgb)
plt.show()
```

```
Training the model
[17:05:21] WARNING: ../src/learner.cc:767:
Parameters: { "n_estimator", "n_job", "silence" } are not used.

Done
 Time Taken:  0:00:42.783785
Evaluating the model on Train data
Evaluating the model on Test data
********************************************************************************************
Train Data
RMSE:  0.7373148503260653
MAPE:  20.878823995033898


********************************************************************************************
Test Data
RMSE:  1.3532714723511179
MAPE:  34.58918952040062
```



Surprise KNN with user-user similarity

```
sim_options = { 'user_based':True, 'shrikage':100, "name":'pearson_baseline', 'min_support':2}
bsl_options = { 'method':'sgd'}
knn_baseline = KNNBaseline(sim_options = sim_options, bsl_options=bsl_options, k=20)
knn_u_bsl_train_result , knn_u_bsl_test_result = run_surprise(knn_baseline, train_set, test_set, verbose=True)

train_model_evaluation['knn_bsl_u'] = knn_u_bsl_train_result
test_model_evaluation['knn_bsl_u']  = knn_u_bsl_test_result
```

```
Training the surprise model
Estimating biases using sgd...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Done
 Time Taken:  0:00:14.746729
Evaluting the Train set
Time Taken:  0:01:38.492308
****************************************************************************************************
Train Data
RMSE:  0.2761839099722744
MAPE:  7.391408141050905
Evaluting the Test set
Time Taken:  0:00:00.055750
****************************************************************************************************
Test Data
RMSE:  1.1058524900205533
MAPE:  35.840822583307116


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Total Time taken to run the algo:  0:01:53.300548
```

Surprise KNN with item-item similarity model

```
sim_options = { 'user_based':False, 'shrikage':100, "name":'pearson_baseline', 'min_support':2}
bsl_options = { 'method':'sgd'}
knn_baseline = KNNBaseline(sim_options = sim_options, bsl_options=bsl_options, k=20)
knn_m_bsl_train_result , knn_m_bsl_test_result = run_surprise(knn_baseline, train_set, test_set, verbose=True)

train_model_evaluation['knn_bsl_m'] = knn_m_bsl_train_result
test_model_evaluation['knn_bsl_m']  = knn_m_bsl_test_result
```

```
Training the surprise model
Estimating biases using sgd...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Done
 Time Taken:  0:00:01.066893
Evaluting the Train set
Time Taken:  0:00:08.502476
****************************************************************************************************
Train Data
RMSE:  0.3166747918437742
MAPE:  8.234166602083588
Evaluting the Test set
Time Taken:  0:00:00.055548
****************************************************************************************************
Test Data
RMSE:  1.1059045973935064
MAPE:  35.842238488246274


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Total Time taken to run the algo:  0:00:09.629935
```

```
sim_options = { 'user_based':False,
                'name':'pearson_baseline',
                'min_support':2}
shrinkage = [20,40,60,80,100]
k = [2,4,6,8,10,15,20,30,40]
bsl_options = { 'method':'sgd'}
avg_score = {}
for kval in k:
  for sval in shrinkage:
    print("KNeighbour: ",kval,"shrinkage: ",sval)
    knn_baseline = KNNBaseline(sim_options = sim_options, bsl_options=bsl_options, k=kval, shrinkage=sval)
    knn_m_bsl_train_result , knn_m_bsl_test_result = run_surprise(knn_baseline, train_set, test_set, verbose=True)
```

```
train_regression_data['knn_bsl_u'] = train_model_evaluation['knn_bsl_u']['y_predict']
train_regression_data['knn_bsl_m'] = train_model_evaluation['knn_bsl_m']['y_predict']
train_regression_data.head(2)
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	bslpre	knn_bsl_u
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	5.0	3.0	1.0	3.370370	4.092437	4	3.919560	3.890735
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	4.0	3.0	5.0	3.555556	4.092437	3	4.078834	3.177330

```
test_regression_data['knn_bsl_u'] = test_model_evaluation['knn_bsl_u']['y_predict']
test_regression_data['knn_bsl_m'] = test_model_evaluation['knn_bsl_m']['y_predict']
test_regression_data.head(2)
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg
0	256033	24	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324
1	1735827	24	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324

```
x_train = train_regression_data.drop(['user', 'movie', 'rating'], axis=1)
y_train = train_regression_data['rating']

x_test = test_regression_data.drop(['user', 'movie', 'rating'], axis=1)
y_test = test_regression_data['rating']

third_xgb = xgb.XGBRegressor(n_estimator = 100, n_job = -1, silence= False, randon_state=15)
train_result , test_result = run_xgboost(third_xgb, x_train, y_train, x_test, y_test)

train_model_evaluation['xgb_bsl_knn'] = train_result
test_model_evaluation['xgb_bsl_knn'] = test_result

xgb.plot_importance(third_xgb)
plt.show()
```

```
Training the model
[17:08:08] WARNING: ../src/learner.cc:767:
```

```python
svd = SVD(n_factors = 50, n_epochs = 20, biased= True, random_state= 15, verbose=True)
train_svd_result, test_svd_result = run_surprise(svd, train_set, test_set, verbose=True)

train_model_evaluation['svd'] = train_svd_result
test_model_evaluation['svd']  = test_svd_result
```

```
Training the surprise model
Processing epoch 0
Processing epoch 1
Processing epoch 2
Processing epoch 3
Processing epoch 4
Processing epoch 5
Processing epoch 6
Processing epoch 7
Processing epoch 8
Processing epoch 9
Processing epoch 10
Processing epoch 11
Processing epoch 12
Processing epoch 13
Processing epoch 14
Processing epoch 15
Processing epoch 16
Processing epoch 17
Processing epoch 18
Processing epoch 19
Done
 Time Taken:  0:00:03.823376
Evaluting the Train set
Time Taken:  0:00:05.871097
****************************************************************************************************
Train Data
RMSE:  0.7321358234779256
MAPE:  22.076350745298114
Evaluting the Test set
Time Taken:  0:00:00.254105
****************************************************************************************************
Test Data
RMSE:  1.1058922174782135
MAPE:  35.83563206357407


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Total Time taken to run the algo:  0:00:09.957217
```

```python
svdpp = SVDpp(n_factors= 50, n_epochs= 20, random_state= 15, verbose=True)
train_svdpp_result, test_svdpp_result = run_surprise(svdpp, train_set, test_set, verbose= True)

train_model_evaluation['svdpp'] = train_svdpp_result
test_model_evaluation['svdpp'] = test_svdpp_result
```

```
Training the surprise model
 processing epoch 0
 processing epoch 1
 processing epoch 2
 processing epoch 3
 processing epoch 4
 processing epoch 5
 processing epoch 6
 processing epoch 7
 processing epoch 8
 processing epoch 9
 processing epoch 10
 processing epoch 11
 processing epoch 12
 processing epoch 13
 processing epoch 14
 processing epoch 15
 processing epoch 16
 processing epoch 17
 processing epoch 18
 processing epoch 19
Done
 Time Taken:  0:00:23.167086
Evaluting the Train set
Time Taken:  0:00:10.573294
****************************************************************************************************
Train Data
RMSE:  0.6032438403305899
MAPE:  17.49285063490268
Evaluting the Test set
Time Taken:  0:00:00.056293
****************************************************************************************************
Test Data
RMSE:  1.1059761773758738
```

```
MAPE:  35.830226335646685


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Total Time taken to run the algo:  0:00:33.805700
```

XGB with 13 features + bslpre + knn_bsl_u + knn_bsl_m + SVD + SVDPP

```
train_regression_data['svd']   = train_model_evaluation['svd']['y_predict']
train_regression_data['svdpp'] = train_model_evaluation['svdpp']['y_predict']
train_regression_data.head(2)
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	UAvg	MAvg	rating	bslpre	knn_bsl_u
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	...	3.0	1.0	3.370370	4.092437	4	3.919560	3.890735
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	...	3.0	5.0	3.555556	4.092437	3	4.078834	3.177330

2 rows × 21 columns

```
test_regression_data['svd']   = test_model_evaluation['svd']['y_predict']
test_regression_data['svdpp'] = test_model_evaluation['svdpp']['y_predict']
test_regression_data.head(2)
```

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	UAvg
0	256033	24	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	...	3.545324	3.545324	3.545324
1	1735827	24	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	3.545324	...	3.545324	3.545324	3.545324

2 rows × 21 columns

```
x_train = train_regression_data.drop(['user' , 'movie' , 'rating'], axis=1)
y_train = train_regression_data['rating']

x_test  = test_regression_data.drop(['user', 'movie', 'rating'], axis = 1)
y_test  = test_regression_data['rating']

forth_xgb = xgb.XGBRegressor(n_job=-1, random_state=15)
train_result, test_result = run_xgboost(forth_xgb, x_train, y_train, x_test, y_test)

train_model_evaluation['xgb_bsl_knn_svd_svdpp'] = train_result
test_model_evaluation['xgb_bsl_knn_svd_svdpp']  = test_result
```