Homework -5

CSCE 633: Machine Learning

Manav Gurumoorthy

830000011

1. Introduction:

The goal of this Homework assignment was to build a machine learning model with relation to a real-world problem. This involved finding a suitable dataset, formulating a problem statement to be solved using a machine learning model and to apply some of the theory that was covered in class.

2. Motivation and Data:

The motivation for this work is from applications in entertainment. The idea was to use data collected about user generated movie ratings and create a system that could recommend new movies to a user based on the ratings of other users.

The data set used is called movielens. It was created by a research group called grouplens. GroupLens Research Project is a research group in the Department of Computer Science at the University of Minnesota.

The dataset consisted of three files, the first called Ratings, this contained the ratings that a user had assigned to a movie. It was in the following format, "UserID::MovieID::Rating::Timestamp", where

1. UserID: was a number between 1 to 6040
2. MovieID: was a number between 1 and 3952
3. Ratings: Whole-star ratings on a scale of 5

Each user in this dataset has rated 20 movies at the least.

The second file was "users' it consisted of some demographic information of the users in the following format UserID::Gender::Age::Occupation::Zip-code.

1. Gender is M or F for male or female.
2. Age is kept in the following ranges:

```
*  1:  "Under 18"
* 18:  "18-24"
* 25:  "25-34"
* 35:  "35-44"
* 45:  "45-49"
* 50:  "50-55"
* 56:  "56+"
```

3. Occupation is chosen from the following choices:

```
*  0:  "other" or not specified
*  1:  "academic/educator"
*  2:  "artist"
*  3:  "clerical/admin"
*  4:  "college/grad student"
*  5:  "customer service"
*  6:  "doctor/health care"
*  7:  "executive/managerial"
*  8:  "farmer"
*  9:  "homemaker"
* 10:  "K-12 student"
* 11:  "lawyer"
* 12:  "programmer"
* 13:  "retired"
* 14:  "sales/marketing"
* 15:  "scientist"
* 16:  "self-employed"
* 17:  "technician/engineer"
* 18:  "tradesman/craftsman"
* 19:  "unemployed"
```

```
* 20:  "writer"
```

The third file was called “Movies”, and contains description of the movies, it is in the following format, “MovieID::Title::Genres”.

1. Titles are the movie titles from IMDB (with year of release).
2. Genres are pipe-separated and are selected from the following genres:

```
* Action
* Adventure
* Animation
* Children's
* Comedy
* Crime
* Documentary
* Drama
* Fantasy
* Film-Noir
* Horror
* Musical
* Mystery
* Romance
* Sci-Fi
* Thriller
* War
* Western
```

3. Problem Formulation

The idea that I had was to use this dataset to make a movie recommendation system. My plan was to make movie recommendations using the concept of Mode-Based Collaborative filtering. I made a mistake in this formulation as this is a kind of a semi-supervised learning algorithm, while the task was to use a supervised method. I did carry on with this initial idea, even though it doesn’t exactly fit the requirement of the problem. Model-based collaborative filtering is based on PCA. Where we use PCA’s matrix factorization to find latent attributes of the inputs while also reducing the dimensionality of the data.

I did try to reformulate this problem into a fully supervised task. Unfortunately, I was unable to do so successfully in the stipulated time.

4. Data pre-processing

The data was in the ‘*.dat’ format. So, the first step was to convert the data from the 3 files into pandas data frames. Also, since the data was separated across 3 files, we merged the data into a single large dataset that contained all the information.
Also, I replaced all missing entries with ‘0’ so that the dataset was complete. I also tried to vectorise some of the features, like the genres, occupation description and the movie title using sklearn’s TfidfVectorizer to keep track of the frequency of commonly used words.

5. Data Exploration

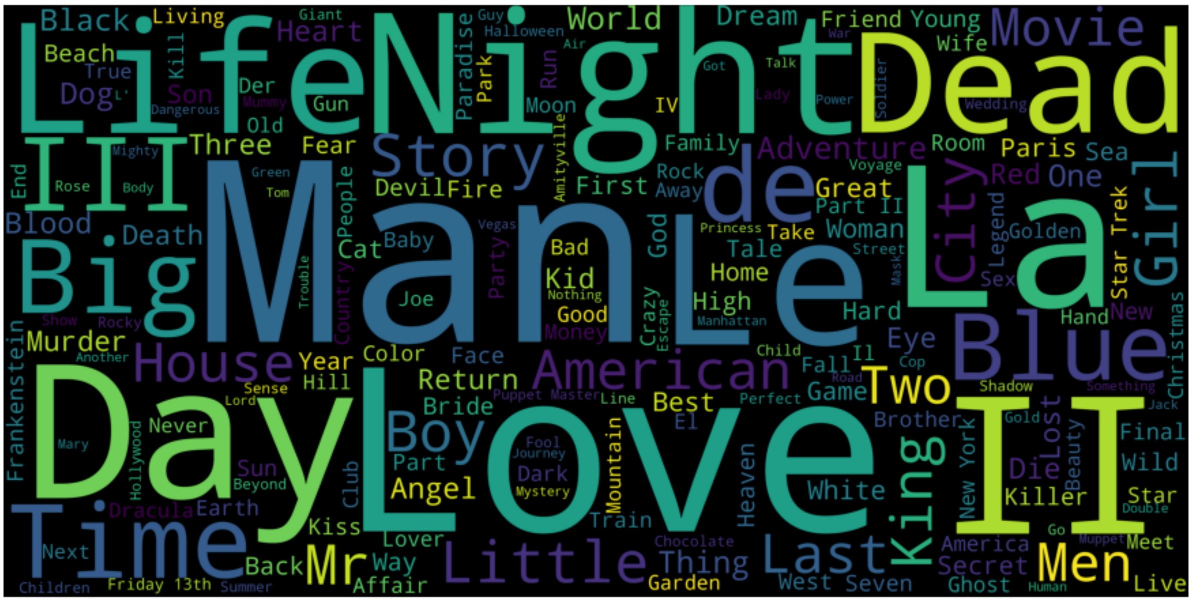
In this part, I explored the dataset. First I we found how many unique movie and user entries were in the dataset.

```
The no. of unique users is : 6040
The no. of unique movies is: 3706
```

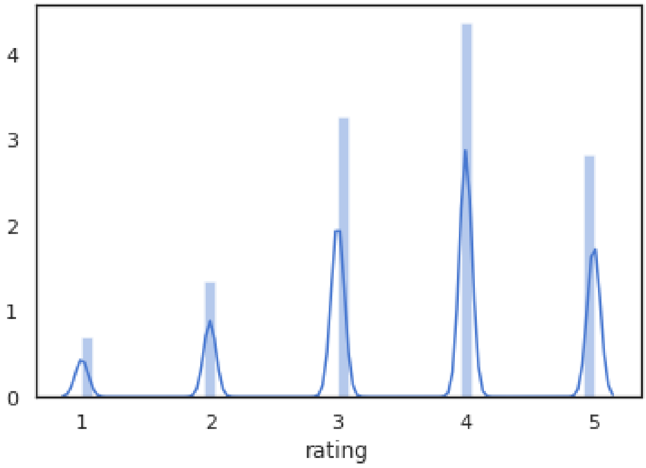
Also some, statistics about the data from dataset:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   user_id     6040 non-null   int64
 1   gender      6040 non-null   object
 2   age         6040 non-null   int64
 3   occupation  6040 non-null   int64
 4   zipcode     6040 non-null   object
 5   age_desc    6040 non-null   object
 6   occ_desc    6040 non-null   object
dtypes: int64(3), object(4)
memory usage: 330.4+ KB
None
```

To get an understanding of the frequencies of some of the terms in the dataset I used a word cloud representation. The larger the word is in the word cloud means more frequent is the word.



The following plot shows how the users have rated the movies. It is clear that the users are quite generous with their ratings. Because a large percentage of the movies have 4 to 5 start ratings.



```
USER RATINGS
_____________

1 Star:56174
2 Star:107557
3 Star:261197
4 Star:348971
5 Star:226310
_____________
Total:1000209 ratings
```

I also plotted the top 20 movies that have the highest rating,

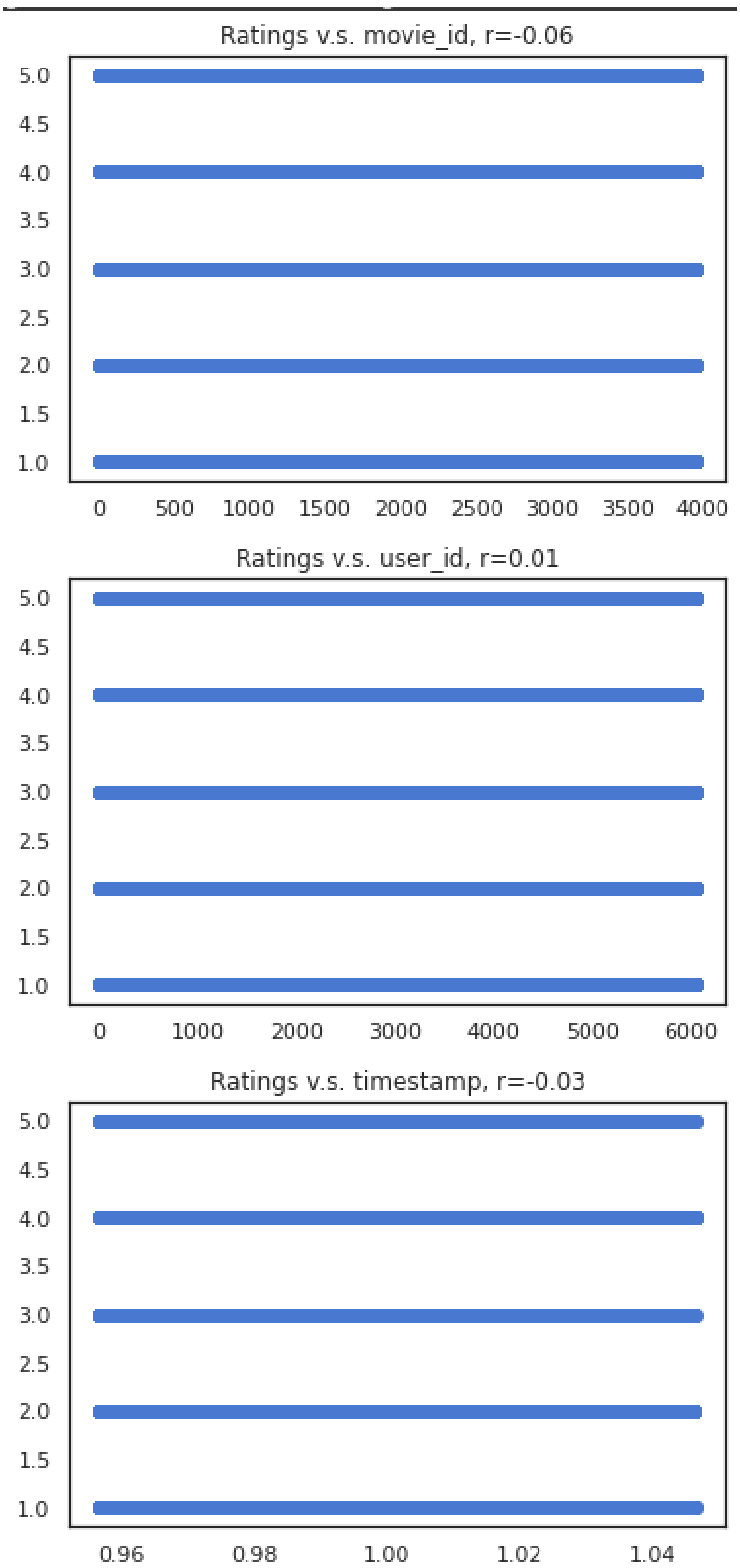
	title	genres	rating
0	Toy Story (1995)	Animation|Children's|Comedy	5
489283	American Beauty (1999)	Comedy|Drama	5
489259	Election (1999)	Comedy	5
489257	Matrix, The (1999)	Action|Sci-Fi|Thriller	5
489256	Dead Ringers (1988)	Drama|Thriller	5
489237	Rushmore (1998)	Comedy	5
489236	Simple Plan, A (1998)	Crime|Thriller	5
489226	Hands on a Hard Body (1996)	Documentary	5
489224	Pleasantville (1998)	Comedy	5
489212	Say Anything... (1989)	Comedy|Drama|Romance	5
489207	Beetlejuice (1988)	Comedy|Fantasy	5
489190	Roger & Me (1989)	Comedy|Documentary	5
489172	Buffalo 66 (1998)	Action|Comedy|Drama	5
489171	Out of Sight (1998)	Action|Crime|Romance	5
489170	I Went Down (1997)	Action|Comedy|Crime	5
489168	Opposite of Sex, The (1998)	Comedy|Drama	5
489157	Good Will Hunting (1997)	Drama	5
489152	Fast, Cheap & Out of Control (1997)	Documentary	5
489149	L.A. Confidential (1997)	Crime|Film-Noir|Mystery|Thriller	5
489145	Contact (1997)	Drama|Sci-Fi	5

Also, this was the distribution of the movies based on the various genres.

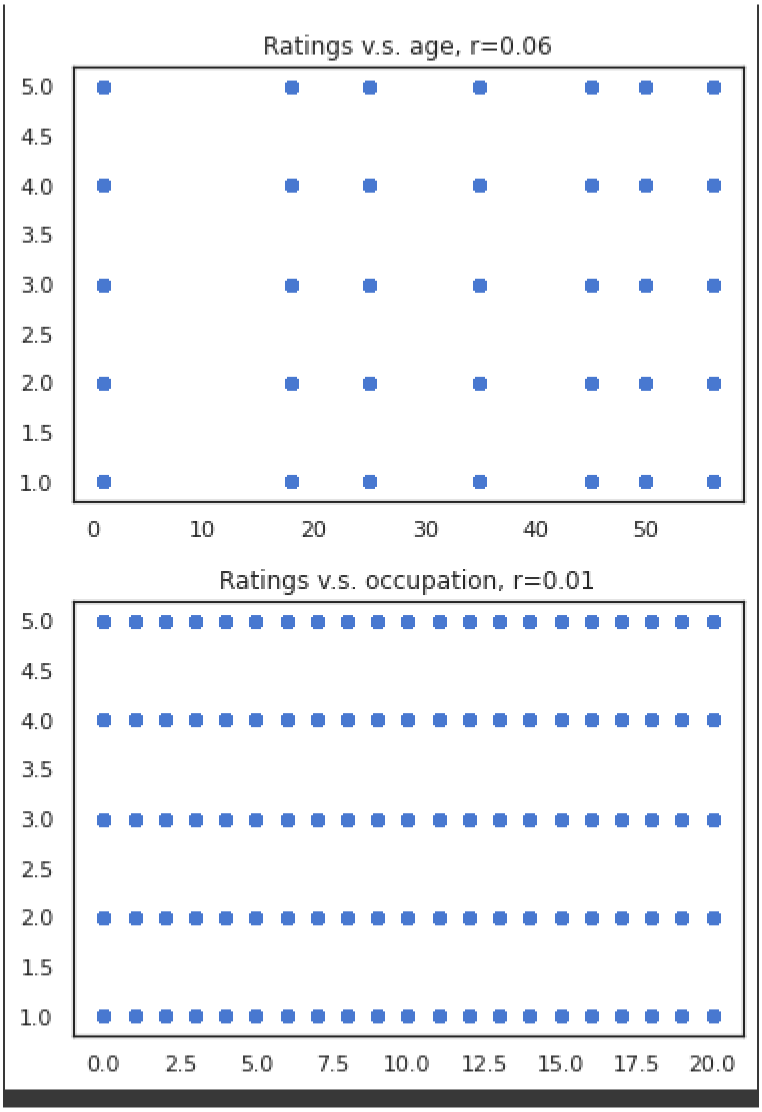
```
Movie          Number of Entries
---------      ---------------------
Drama                  1603

Comedy                 1200

Action                 503

Thriller                       492

Romance                471

Horror                 343

Adventure                      283

Sci-Fi                 276

Children's                     251

Crime                  211

War                    143

Documentary                    127

Musical                114

Mystery                106

Animation                      105

Western                68

Fantasy                68

Film-Noir                      44
```

6. Feature Selection

Used Pearson’s r to find the correlation between the features with the target and the correlation between the features themselves. As it is visible from below there were no real correlations between the data and the selected target. Which showed me further that this task may not be suitable for a supervised learning algorithm.



Ratings v.s. movie_id, r=-0.06
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
500
1000
1500
2000
2500
3000
3500
4000
Ratings v.s. user_id, r=0.01
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
1000
2000
3000
4000
5000
6000
Ratings v.s. timestamp, r=-0.03
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.96
0.98
1.00
1.02
1.04



The correlation numbers between the features:

```
1.00,-0.02,0.04,0.03,0.01,
-0.02,1.00,-0.49,0.03,-0.03,
0.04,-0.49,1.00,-0.06,0.02,
0.03,0.03,-0.06,1.00,0.08,
0.01,-0.03,0.02,0.08,1.00,
```

7. Feature Transformation

I have used Principal Component Analysis in this section to find latent associations in the data and also to reduce the dimensionality of the data. In this portion I created a matrix between user Id and movie Id with each entry representing the rating given by that user for that movie.

Using the reconstructed matrix I make predictions for a user based on how well the movie was rated by other users.

From the dataset I find the movies that the target user has not watched and use that information to recommend the highest rated movie that the user hasn't seen. While this model doesn't use any other information you can see that it does make some good prediction. For example, in one case, a user had liked and seen Star Wars Episode 1 and the system recommended Episode 3 and Episode 6 to the user. Now qualitatively this shows that the system is "working" as expected.

Here is where I was faced with another challenge, for the evaluation of the model, since this was basically an unsupervised model, I was unable to come up with any powerful quantitative measures of the system performance.

Luckily, I did find a library called Surprise online that was built to test such recommendation systems, using the surprise model, I found the model to have a Root Mean Square Error to be 0.87 over a 5 fold cross validation.

The details of this can be found in Appendix A as a part of the code.

8. Conclusion

The purpose of this homework was to go through the entire process of building a machine learning model to solve a real-world problem. In this I looked at creating a movie recommendation system, very similar to the algorithms that run on YouTube and Netflix. While I may have not exactly solved all the requirements of the homework, I have learned a lot about the concepts involved in this area of work. I feel this work can be extended to make more holistic recommendation by taking the demographic information of the users into consideration. There may also be better methods in order to create such a recommendation system.

APPENDIX A
COLAB NOTEBOOK

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.a

Enter your authorization code:
··········
Mounted at /content/drive
```

```
#IMPORTING THE DATA FROM THE DATASETS.
import pandas as pd
import numpy as np
import csv
import matplotlib.pyplot as plt
from skimage import data, io, filters
# Import the necessary modules and libraries
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold


path_users = "/content/drive/My Drive/ML Project /users.dat"
path_ratings = "/content/drive/My Drive/ML Project /ratings.dat"
path_movies = "/content/drive/My Drive/ML Project /movies.dat"
```

Data Preparation

```
AGES = { 1: "Under 18", 18: "18-24", 25: "25-34", 35: "35-44", 45: "45-49", 50: "50-55", 56: "56+" }
OCCUPATIONS = { 0: "other or not specified", 1: "academic/educator", 2: "artist", 3: "clerical/admin",
                4: "college/grad student", 5: "customer service", 6: "doctor/health care",
                7: "executive/managerial", 8: "farmer", 9: "homemaker", 10: "K-12 student", 11: "lawyer",
                12: "programmer", 13: "retired", 14: "sales/marketing", 15: "scientist", 16: "self-employed",
```

```
                 12: "programmer", 13: "retired", 14: "sales/marketing", 15: "scientist", 16: "self-employed",
                 17: "technician/engineer", 18: "tradesman/craftsman", 19: "unemployed", 20: "writer" }

ratings = pd.read_csv(path_ratings,sep = '::', engine='python', names=['user_id', 'movie_id', 'rating', 'timestamp'])
max_userid = ratings['user_id'].drop_duplicates().max()
# Set max_movieid to the maximum movie_id in the ratings
max_movieid = ratings['movie_id'].drop_duplicates().max()

# Process ratings dataframe for Keras Deep Learning model
# Add user_emb_id column whose values == user_id - 1
ratings['user_emb_id'] = ratings['user_id'] - 1
# Add movie_emb_id column whose values == movie_id - 1
ratings['movie_emb_id'] = ratings['movie_id'] - 1

users = pd.read_csv(path_users,
                    sep='::',
                    engine='python',
                    names=['user_id', 'gender', 'age', 'occupation', 'zipcode'])
users['age_desc'] = users['age'].apply(lambda x: AGES[x])
users['occ_desc'] = users['occupation'].apply(lambda x: OCCUPATIONS[x])

movies = pd.read_csv(path_movies,sep='::',
                     engine ='python',
                     names = ['movie_id', 'title', 'genres'])

dataset = pd.merge(pd.merge(movies,ratings),users)

```

EXPLORATION

```
#computing the number of unique users and movies in this dataset:
noOfUsers = ratings.user_id.unique().shape[0]
noOfMovies = ratings.movie_id.unique().shape[0]
print("The no. of unique users is : {0}\nThe no. of unique movies is: {1}".format(noOfUsers,noOfMovies))
```

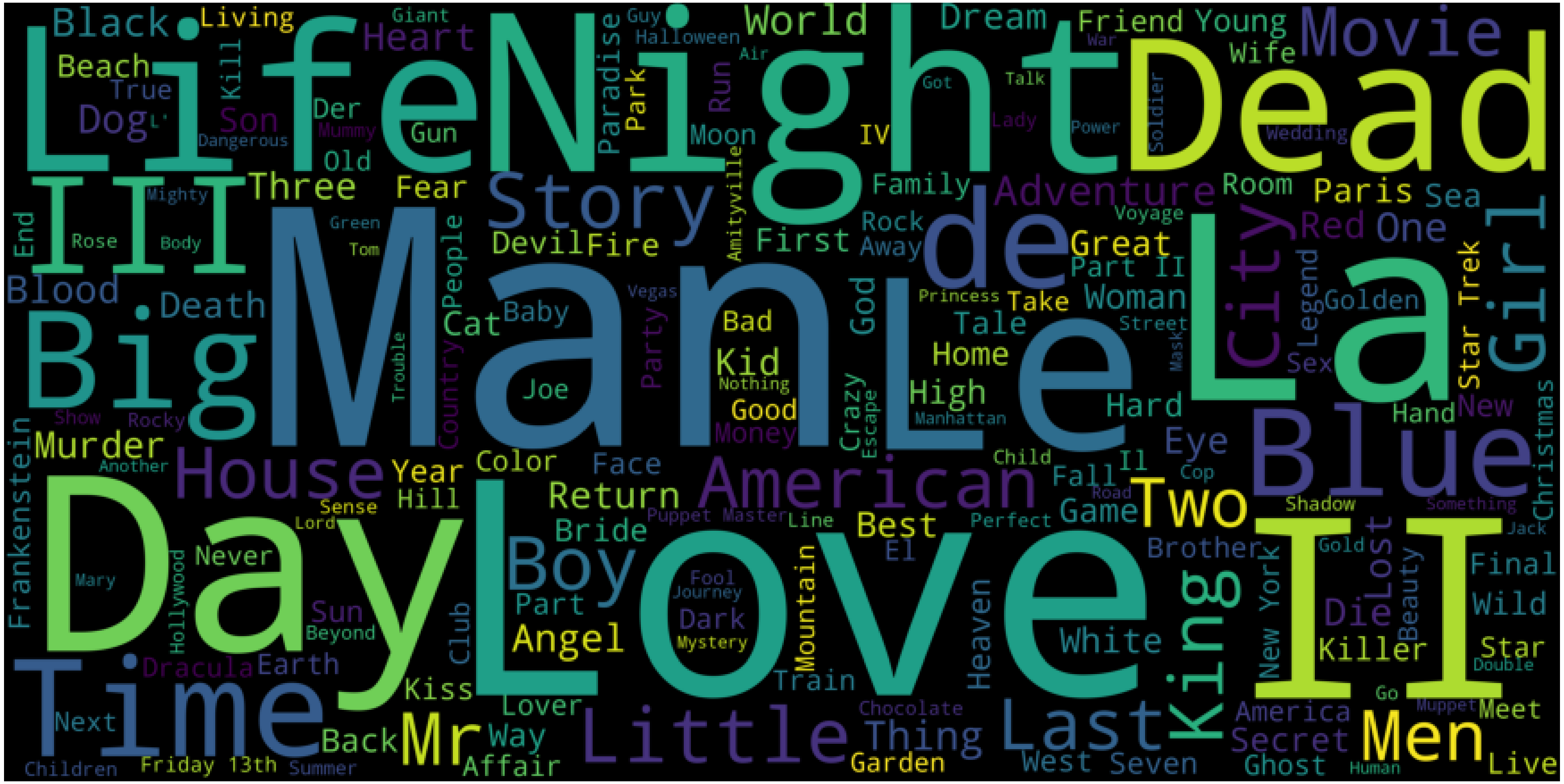
```
The no. of unique users is : 6040
The no. of unique movies is: 3706
```

```
print(users.info())
print(users.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   user_id     6040 non-null   int64 
 1   gender      6040 non-null   object
 2   age         6040 non-null   int64 
 3   occupation  6040 non-null   int64 
 4   zipcode     6040 non-null   object
 5   age_desc    6040 non-null   object
 6   occ_desc    6040 non-null   object
dtypes: int64(3), object(4)
memory usage: 330.4+ KB
None
   user_id gender  age  occupation zipcode  age_desc              occ_desc
0        1      F    1          10   48067  Under 18          K-12 student
1        2      M   56          16   70072       56+         self-employed
2        3      M   25          15   55117     25-34             scientist
3        4      M   45           7   02460     45-49  executive/managerial
4        5      M   25          20   55455     25-34                writer
```

```
import wordcloud
from wordcloud import WordCloud, STOPWORDS

movies['title'] = movies['title'].fillna("").astype('str')
title_list = ' '.join(movies['title'])
title_cloud = WordCloud(stopwords=STOPWORDS, height=2000, width=4000).generate(title_list)

plt.figure(figsize=(16,8))
plt.axis('off')
plt.imshow(title_cloud)
```

```
plt.show()
```

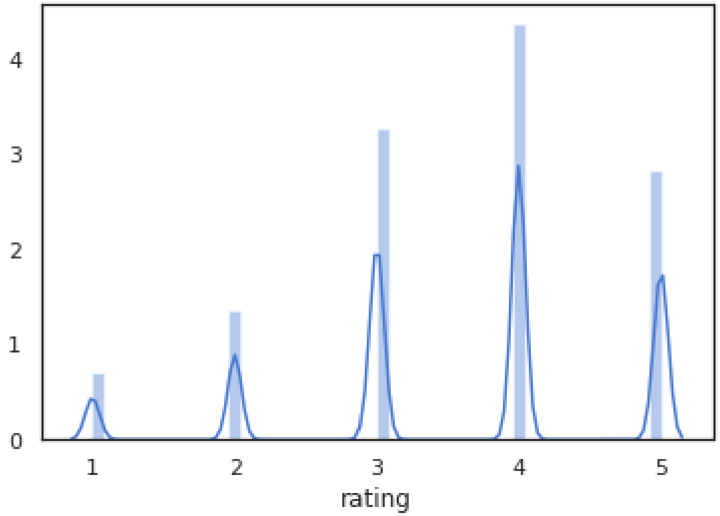



```
import seaborn as sns
import pandas.util.testing as tm
ratings['rating'].describe()
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use t
  import pandas.util.testing as tm
count    1.000209e+06
mean     3.581564e+00
std      1.117102e+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
```

```
1 sns.set(style="white", palette="muted", color_codes=True)
2 sns.distplot(ratings['rating'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb783d2e0f0>
```




```
1 dataset[['title','genres','rating']].sort_values('rating', ascending=False).head(20)
```

	title	genres	rating
0	Toy Story (1995)	Animation|Children's|Comedy	5
489283	American Beauty (1999)	Comedy|Drama	5
489259	Election (1999)	Comedy	5
489257	Matrix, The (1999)	Action|Sci-Fi|Thriller	5
489256	Dead Ringers (1988)	Drama|Thriller	5
489237	Rushmore (1998)	Comedy	5
489236	Simple Plan, A (1998)	Crime|Thriller	5
489226	Hands on a Hard Body (1996)	Documentary	5
489224	Pleasantville (1998)	Comedy	5
489212	Say Anything... (1989)	Comedy|Drama|Romance	5
489207	Beetlejuice (1988)	Comedy|Fantasy	5
489190	Roger & Me (1989)	Comedy|Documentary	5
489172	Buffalo 66 (1998)	Action|Comedy|Drama	5
489171	Out of Sight (1998)	Action|Crime|Romance	5
489170	I Went Down (1997)	Action|Comedy|Crime	5
489168	Opposite of Sex, The (1998)	Comedy|Drama	5
489157	Good Will Hunting (1997)	Drama	5
489152	Fast, Cheap & Out of Control (1997)	Documentary	5
489149	L.A. Confidential (1997)	Crime|Film-Noir|Mystery|Thriller	5
489145	Contact (1997)	Drama|Sci-Fi	5

```
count = [0,0,0,0,0]
```

```
count = [0,0,0,0,0]
for i in dataset['rating']:
  count[i-1] = count[i-1]+1

print("USER RATINGS\n_____________")
print("\n1 Star:{0}\n2 Star:{1}\n3 Star:{2}\n4 Star:{3}\n5 Star:{4}\n_____________\nTotal:{5} ratings".
      format(count[0],count[1],count[2],count[3],count[4],sum(count)))
```

```
USER RATINGS
_____________

1 Star:56174
2 Star:107557
3 Star:261197
4 Star:348971
5 Star:226310
_____________
Total:1000209 ratings
```

```
genre_labels = set()
for s in movies['genres'].str.split('|').values:
    genre_labels = genre_labels.union(set(s))

def count_word(dataset, ref_col, census):
    keyword_count = dict()
    for s in census:
        keyword_count[s] = 0
    for census_keywords in dataset[ref_col].str.split('|'):
        if type(census_keywords) == float and pd.isnull(census_keywords):
            continue
        for s in [s for s in census_keywords if s in census]:
            if pd.notnull(s):
                keyword_count[s] += 1
    keyword_occurences = []
    for k,v in keyword_count.items():
        keyword_occurences.append([k,v])
    keyword_occurences.sort(key = lambda x:x[1], reverse = True)
    return keyword_occurences, keyword_count
```

```
keyword_occurences, dum = count_word(movies, 'genres', genre_labels)
print("Movie\t\tNumber of Entries\n---------\t--------------------- ")
keyword_occurences = np.array(keyword_occurences)
for i in range (len(keyword_occurences)):
  print("{0}\t\t\t{1}\n".format(keyword_occurences[i][0],keyword_occurences[i][1]).ljust(10))
```

Movie	Number of Entries
Drama	1603
Comedy	1200
Action	503
Thriller	492
Romance	471
Horror	343
Adventure	283
Sci-Fi	276
Children's	251
Crime	211
War	143
Documentary	127
Musical	114
Mystery	106
Animation	105
Western	68
Fantasy	68
Film-Noir	44

Data pre-processing

```python
# Creating a train, test and validation dataset
import pandas as pd
from sklearn import datasets, linear_model
from sklearn.model_selection import train_test_split

data_train, data_test = train_test_split(ratings, test_size=0.2)
data_test, data_valid = train_test_split(ratings, test_size = 0.5)
```

```python
# Fill NaN values in user_id and movie_id column with 0
ratings['user_id'] = ratings['user_id'].fillna(0)
ratings['movie_id'] = ratings['movie_id'].fillna(0)

# Replace NaN values in rating column with average of all values
ratings['rating'] = ratings['rating'].fillna(ratings['rating'].mean())
print(ratings.info())
data_subset = ratings.sample(frac= 0.02)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 6 columns):
 #   Column        Non-Null Count    Dtype
---  ------        --------------    -----
 0   user_id       1000209 non-null  int64
 1   movie_id      1000209 non-null  int64
 2   rating        1000209 non-null  int64
 3   timestamp     1000209 non-null  int64
 4   user_emb_id   1000209 non-null  int64
 5   movie_emb_id  1000209 non-null  int64
dtypes: int64(6)
memory usage: 45.8 MB
None
```

```python
ratings = ratings.drop(labels=[ 'user_emb_id',
```

```
      'movie_emb_id'],axis =1)
```

```
dataset_in = dataset.drop(labels=['title', 'genres','user_emb_id', 'movie_emb_id', 'gender','age_desc','occ_desc','zipcode'],axis=
```

```
# Convert titles to string value
dataset['title'] = dataset['title'].fillna("").astype('str')
# Convert genres to string value
dataset['genres'] = dataset['genres'].fillna("").astype('str')
# Convert occupation description to string value
dataset['occ_desc'] = dataset['occ_desc'].fillna("").astype('str')
```

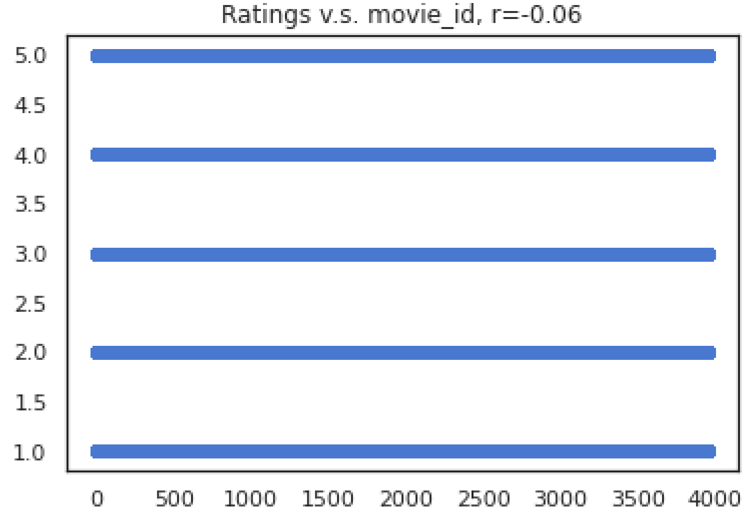
```
from sklearn.feature_extraction.text import TfidfVectorizer

tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix_genres = tf.fit_transform(dataset['genres'])
print(tfidf_matrix_genres.shape)

tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix_occ = tf.fit_transform(dataset['occ_desc'])
print(tfidf_matrix_occ.shape)

tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix_title = tf.fit_transform(dataset['occ_desc'])
print(tfidf_matrix_title.shape)
```

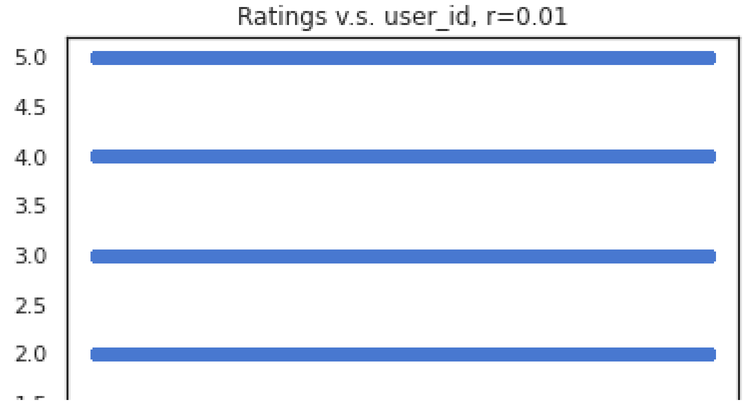
```
(1000209, 127)
(1000209, 46)
(1000209, 46)
```

```
#pearson's r between features vs ratings
from scipy.stats import pearsonr

print(dataset_in)
dataset_in = dataset_in.drop(labels=['rating'],axis=1)
target = dataset.rating
for i in dataset_in.columns:
  plt.scatter(dataset_in[i],target)
```

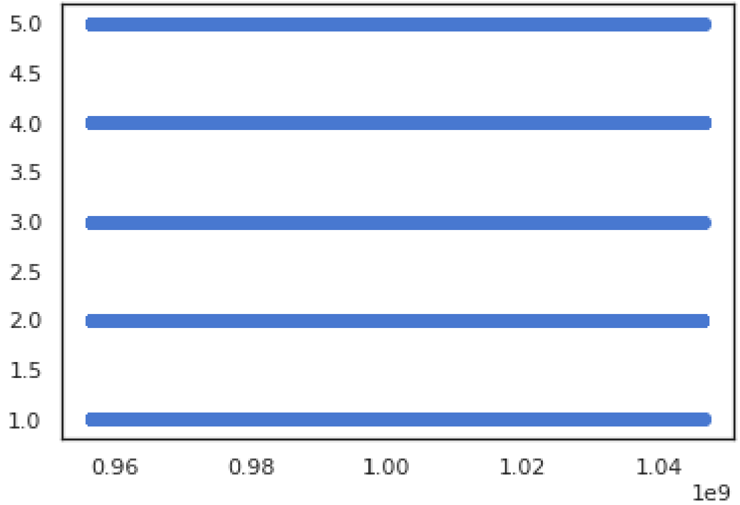
```
plt.scatter(dataset_in[i],target)
corr,p = pearsonr(dataset_in[i],target)
plt.title("Ratings v.s. " + i + ", r=" + str(format(corr, '.2f')))
plt.show()
```

```
         movie_id  user_id  rating  timestamp  age  occupation
0               1        1       5  978824268    1          10
1              48        1       5  978824351    1          10
2             150        1       5  978301777    1          10
3             260        1       4  978300760    1          10
4             527        1       5  978824195    1          10
...           ...      ...     ...        ...  ...         ...
1000204      3513     5727       4  958489970   25           4
1000205      3535     5727       2  958489970   25           4
1000206      3536     5727       5  958489902   25           4
1000207      3555     5727       3  958490699   25           4
1000208      3578     5727       5  958490171   25           4

[1000209 rows x 6 columns]
```

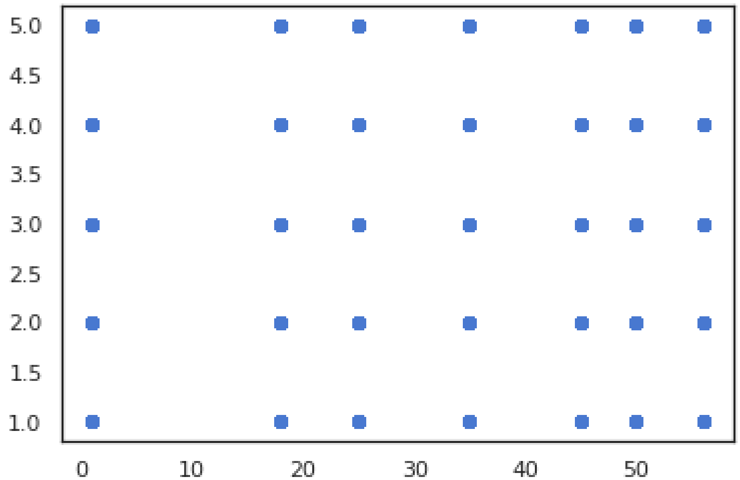




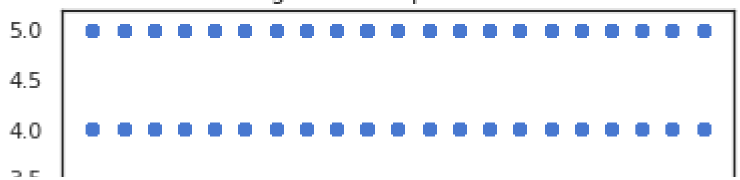
Ratings v.s. timestamp, r=-0.03

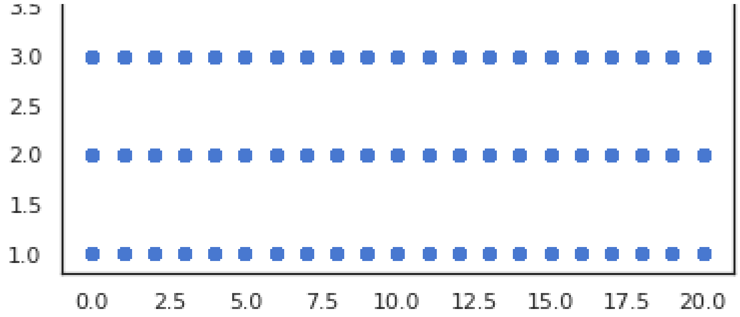


Ratings v.s. age, r=0.06



Ratings v.s. occupation, r=0.01






```
for i in dataset_in.columns:
  for j in dataset_in.columns:
    corr, p = pearsonr(dataset_in[i], dataset_in[j])
    print(format(corr, '.2f'), end = ',')
  print()

```

```
1.00,-0.02,0.04,0.03,0.01,
-0.02,1.00,-0.49,0.03,-0.03,
0.04,-0.49,1.00,-0.06,0.02,
0.03,0.03,-0.06,1.00,0.08,
0.01,-0.03,0.02,0.08,1.00,
```

```
data_train, data_test, target_train, target_test = train_test_split(dataset_in,target, test_size=0.2)
data_test, data_valid, target_test, target_valid = train_test_split(data_test,target_test,test_size = 0.5)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge

lr = LinearRegression()
lr.fit(data_train,target_train)

train_score, train_score_p = pearsonr(lr.predict(data_train),target_train)
print("r=" + format(train_score, '.2f') + ", p=" + format(train_score_p, '.2f'))

test_score, test_score_p = pearsonr(lr.predict(data_test),target_test)
print("r=" + format(test_score, '.2f') + ", p=" + format(test_score_p, '.2f'))
```

```
r=0.09, p=0.00
r=0.09, p=0.00
```

```
# Randomly sample 1% of the ratings dataset
small_data = dataset_in.sample(frac=0.02)
# Check the sample info
print(small_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20004 entries, 671785 to 527272
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   movie_id    20004 non-null  int64
 1   user_id     20004 non-null  int64
 2   timestamp   20004 non-null  int64
 3   age         20004 non-null  int64
 4   occupation  20004 non-null  int64
dtypes: int64(5)
memory usage: 937.7 KB
None
```

```
data_train, data_test= train_test_split(small_data, test_size=0.2)
# data_test, data_valid= train_test_split(data_test, test_size = 0.5)
```

```
# Create two user-item matrices, one for training and another for testing
train_data_matrix = np.array(data_train.drop(labels=['age', 'occupation'],axis=1))
test_data_matrix = np.array(data_test.drop(labels=['age', 'occupation'],axis=1))

# Check their shape
print(train_data_matrix.shape)
print(test_data_matrix.shape)
```

```
(16003, 3)
(4001, 3)
```

```
from sklearn.metrics.pairwise import pairwise_distances
```

```
# Item Similarity Matrix
item_correlation = 1 - pairwise_distances(train_data_matrix.T, metric='correlation')
item_correlation[np.isnan(item_correlation)] = 0
print(item_correlation)
```

```
[[ 1.         -0.01479626  0.03406019]
 [-0.01479626  1.         -0.48371933]
 [ 0.03406019 -0.48371933  1.        ]]
```

```
# Function to predict ratings
def predict(ratings, similarity):
    pred = ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)])
    return pred
```

```
from sklearn.metrics import mean_squared_error
from math import sqrt

# Function to calculate RMSE
def rmse(pred, actual):
    # Ignore nonzero terms.
    pred = pred[actual.nonzero()].flatten()
    actual = actual[actual.nonzero()].flatten()
    return sqrt(mean_squared_error(pred, actual))
```

```
item_prediction = predict(train_data_matrix, item_correlation)
```

```
# RMSE on the test data
print('Item-based CF RMSE: ' + str(rmse(item_prediction, test_data_matrix)))
```

```
Item-based CF RMSE: 264455589.246584
```

```
# RMSE on the train data
print('Item-based CF RMSE: ' + str(rmse(item_prediction, train_data_matrix)))
```

```
Item-based CF RMSE: 264269704.02283144
```

PCA Implementation

```
Ratings = ratings.pivot(index = 'user_id', columns ='movie_id', values = 'rating').fillna(0)
```

```
# Data_Preprocessing for zero mean
R = Ratings.to_numpy()
user_ratings_mean = np.mean(R, axis = 1)
Ratings_demeaned = R - user_ratings_mean.reshape(-1, 1)
```

```
from numpy import mean
from numpy import cov
from numpy.linalg import eig

V = cov(Ratings_demeaned.T)
values, vectors = eig(V)
new_vectors=np.argsort(values)[-50:]
new_values = []
Vt = []
U = []

for i in new_vectors:
  new_values.append(values[i])
  Vt.append(vectors[i])
  U.append(R.T[i])
sigma= np.diag(new_values)

Vt = np.array(Vt)
U = np.array(U)
P = np.dot(np.dot(U.T, sigma), Vt)
P = P + user_ratings_mean.reshape(-1, 1)
P = pd.DataFrame(P, columns = Ratings.columns)
P.head()
```

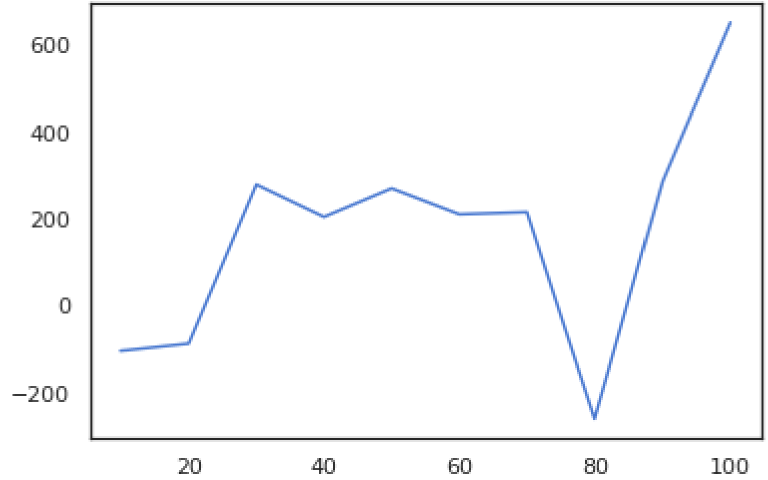
movie_id	1	2	3	4	5	6
0	-54.567852+0.000000j	15.089488+0.000000j	28.505553+0.000000j	48.413456+0.000000j	47.102674+0.000000j	51.276238+0.000000j
1	-0.210675+0.000000j	0.192335+0.000000j	0.314633+0.000000j	-0.025535+0.000000j	-0.025579+0.000000j	0.014184+0.000000j
2	0.053697+0.000000j	0.053697+0.000000j	0.053697+0.000000j	0.053697+0.000000j	0.053697+0.000000j	0.053697+0.000000j
3	0.023745+0.000000j	0.023745+0.000000j	0.023745+0.000000j	0.023745+0.000000j	0.023745+0.000000j	0.023745+0.000000j
4	-7.530086+0.000000j	0.809984+0.000000j	6.014066+0.000000j	-9.423351+0.000000j	0.393749+0.000000j	1.785977+0.000000j

5 rows × 3706 columns

```
# SVD for dimensionality reduction
sigma_list = []
U_list = []
Vt_list = []
for i in range (1,11):
  from scipy.sparse.linalg import svds
  U, sigma, Vt = svds(Ratings_demeaned, k=i*10)
  sigma_list.append(sigma)
  U_list.append(U)
  Vt_list.append(Vt)
s= []
for i in range(len(sigma_list)):
  Lambda_d_sum = sum(sum(U_list[i]))
  Lambda_k_sum = sum(sigma_list[i])
  sk = Lambda_k_sum/Lambda_d_sum
  s.append(sk)
```

```
k =[10,20,30,40,50,60,70,80,90,100]
plt.plot(k,s)
```

```
[<matplotlib.lines.Line2D at 0x7fb7921a35c0>]
```




```
def reconstruct_data(U,sigma,Vt):
  sigma = np.diag(sigma)
  all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_ratings_mean.reshape(-1, 1)
  preds = pd.DataFrame(all_user_predicted_ratings, columns = Ratings.columns)
```

```
sigma = np.diag(sigma)
```

```
all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_ratings_mean.reshape(-1, 1)
```

```
preds = pd.DataFrame(all_user_predicted_ratings, columns = Ratings.columns)
```

```
preds.head()
```

movie_id	1	2	3	4	5	6	7	8	9	10	11	1
0	5.157608	0.184833	0.348341	-0.022609	0.139622	-0.156937	-0.061122	0.072117	0.018278	-0.372566	-0.275992	-0.06915
1	0.557186	0.296927	0.078853	-0.013888	0.028675	1.092160	-0.054492	0.114191	0.090106	1.695371	0.611882	0.08798
2	2.176318	0.396428	0.302057	-0.117164	-0.006330	0.077833	0.000836	0.064654	-0.018309	1.062417	-0.231946	0.04593
3	0.194185	0.155507	0.046863	0.047477	-0.014495	0.247765	-0.057580	-0.006338	0.007387	-0.423240	-0.267654	0.00019

```
ratings.head()
```

	user_id	movie_id	rating	timestamp	user_emb_id	movie_emb_id
0	1	1193	5	978300760	0	1192
1	1	661	3	978302109	0	660
2	1	914	3	978301968	0	913
3	1	3408	4	978300275	0	3407
4	1	2355	5	978824291	0	2354

```
def recommend_movies(predictions, userID, movies, original_ratings, num_recommendations):

    # Get and sort the user's predictions
    user_row_number = userID - 1 # User ID starts at 1, not 0
    sorted_user_predictions = preds.iloc[user_row_number].sort_values(ascending=False)

    # Get the user's data and merge in the movie information.
    user_data = original_ratings[original_ratings.user_id == (userID)]
    #adding information about the movie. This is not part of the prediction just for visualisation
    user_full = (user_data.merge(movies, how = 'left', left_on = 'movie_id', right_on = 'movie_id').
                     sort_values(['rating'], ascending=False)
                 )

    print('User {0} has already rated {1} movies.'.format(userID, user_full.shape[0]))
    print('Recommending highest {0} predicted ratings movies not already rated.'.format(num_recommendations))

```

```
    # Recommend the highest predicted rating movies that the user hasn't seen yet.
    recommendations = (movies[~movies['movie_id'].isin(user_full['movie_id'])].
         merge(pd.DataFrame(sorted_user_predictions).reset_index(), how = 'left',
               left_on = 'movie_id',
               right_on = 'movie_id').
         rename(columns = {user_row_number: 'Predictions'}).
         sort_values('Predictions', ascending = False).
                       iloc[:num_recommendations, :-1]
                      )

    return user_full, recommendations
```

```
already_rated, predictions = recommend_movies(preds, 65, movies, ratings, 50)
```

```
User 65 has already rated 121 movies.
Recommending highest 50 predicted ratings movies not already rated.
```

```
already_rated.head(10)
```

	user_id	movie_id	rating	timestamp	user_emb_id	movie_emb_id	title	genres
120	65	1246	5	987383453	64	1245	Dead Poets Society (1989)	Drama
100	65	1124	5	983853171	64	1123	On Golden Pond (1981)	Drama
54	65	3252	5	977888608	64	3251	Scent of a Woman (1992)	Drama
53	65	1573	5	986615241	64	1572	Face/Off (1997)	Action|Sci-Fi|Thriller
94	65	500	5	977888587	64	499	Mrs. Doubtfire (1993)	Comedy
51	65	969	5	986615227	64	968	African Queen, The (1951)	Action|Adventure|Romance|War
95	65	1036	5	986615095	64	1035	Die Hard (1988)	Action|Thriller

```
predictions.head(50)
```

3397	3578	Gladiator (2000)	Action|Drama
2485	2640	Superman (1978)	Action|Adventure|Sci-Fi
1866	2002	Lethal Weapon 3 (1992)	Action|Comedy|Crime|Drama
3232	3408	Erin Brockovich (2000)	Drama
8	11	American President, The (1995)	Comedy|Drama|Romance
2260	2405	Jewel of the Nile, The (1985)	Action|Adventure|Comedy|Romance
569	589	Terminator 2: Judgment Day (1991)	Action|Sci-Fi|Thriller
3638	3827	Space Cowboys (2000)	Action|Sci-Fi
1176	1234	Sting, The (1973)	Comedy|Crime
353	364	Lion King, The (1994)	Animation|Children's|Musical
3264	3441	Red Dawn (1984)	Action|War
3272	3450	Grumpy Old Men (1993)	Comedy
2719	2881	Double Jeopardy (1999)	Action|Thriller
154	161	Crimson Tide (1995)	Drama|Thriller|War
57	62	Mr. Holland's Opus (1995)	Drama
2915	3082	World Is Not Enough, The (1999)	Action|Thriller
2158	2302	My Cousin Vinny (1992)	Comedy
1212	1275	Highlander (1986)	Action|Adventure
1077	1127	Abyss, The (1989)	Action|Adventure|Sci-Fi|Thriller
1488	1584	Contact (1997)	Drama|Sci-Fi
1745	1876	Deep Impact (1998)	Action|Drama|Sci-Fi|Thriller
1047	1092	Basic Instinct (1992)	Mystery|Thriller
1577	1682	Truman Show, The (1998)	Drama

2275	2424	You've Got Mail (1998)	Comedy|Romance
2934	3101	Fatal Attraction (1987)	Thriller
3188	3363	American Graffiti (1973)	Comedy|Drama
2822	2985	Robocop (1987)	Action|Crime|Sci-Fi
2632	2791	Airplane! (1980)	Comedy
887	924	2001: A Space Odyssey (1968)	Drama|Mystery|Sci-Fi|Thriller
283	292	Outbreak (1995)	Action|Drama|Thriller
43	47	Seven (Se7en) (1995)	Crime|Thriller
459	474	In the Line of Fire (1993)	Action|Thriller

```python
#Evaluation of the model
from surprise import SVD
from surprise import Dataset, Reader
from surprise.model_selection import cross_validate

reader = Reader()

# Load the movielens-100k dataset (download it if needed),
data = Dataset.load_from_df(ratings[['user_id', 'movie_id', 'rating']], reader)

# We'll use the famous SVD algorithm.
algo = SVD()

# Run 5-fold cross-validation and print results
cross_validate(algo, data, measures=['RMSE'], cv=5, verbose=True)
```

```
Evaluating RMSE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8753  0.8748  0.8714  0.8723  0.8741  0.8736  0.0015
Fit time          59.14   59.48   59.90   57.42   58.90   58.97   0.85
Test time         2.76    2.99    2.78    2.45    2.76    2.75    0.17
{'fit_time': (59.143911838531494,
  59.48141026496887,
  59.902074337005615,
  57.41506791114807,
  58.90307545661926),
 'test_rmse': array([0.87532321, 0.87482425, 0.87136265, 0.87230161, 0.87410427]),
 'test_time': (2.758392333984375,
  2.9852092266082764,
  2.7836737632751465,
  2.4511756896972656,
  2.761986017227173)}
```