Lab 10 - ML Programming

January 29, 2022

1 EXERCISE 1

1.1 Exploring Movie Recommendation Dataset

[55]:

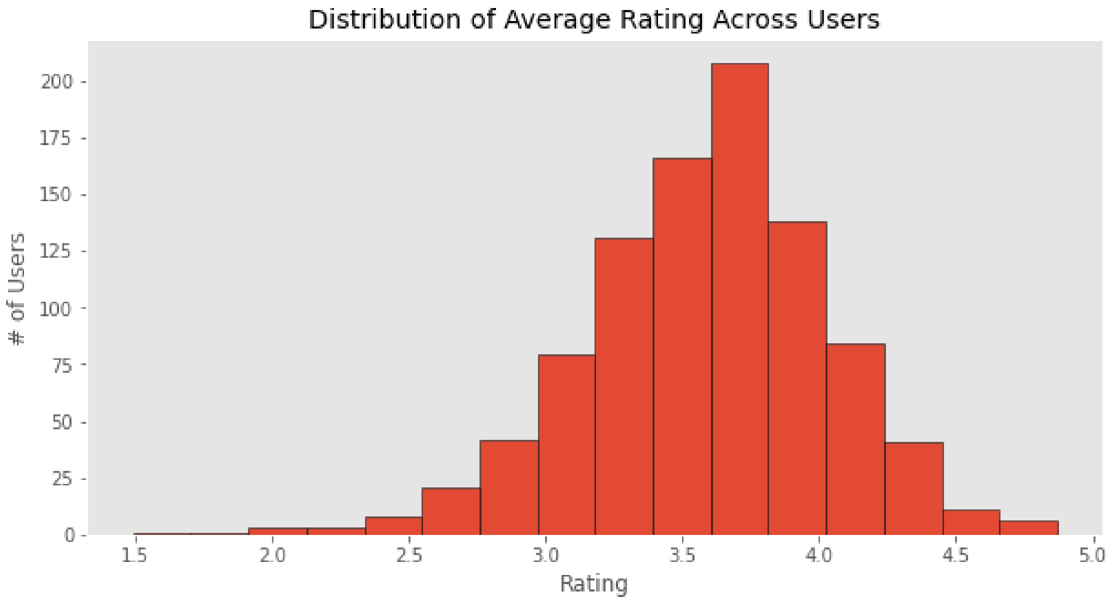
```
## Import data + necessary libraries

import matplotlib.pyplot as plt
import matplotlib as mp
import numpy as np
import pandas as pd
import math
plt.style.use('ggplot')

data = pd.read_csv('u.data',delimiter="\t", names=['user id','movie␣
 ↪id','rating', 'timestamp'])
user = pd.read_csv('u.user',delimiter="|",names=['user␣
 ↪id','age','gender','occupation','zip code'])
item = pd.read_csv('u.item',delimiter="|",names=['movie id','movie␣
 ↪title','release date','video release date',
              'IMDb URL','unknown','Action','Adventure','Animation',
              "Children's",'Comedy','Crime','Documentary','Drama','Fantasy',
              'Film-Noir','Horror','Musical','Mystery','Romance','Sci-Fi',
              'Thriller','War','Western'])
item.drop(['video release date','IMDb URL'], axis=1, inplace= True)
## Just lists of genre and occupation, not as relevant
genre = pd.read_csv('u.genre',delimiter="|",header=None)
genre.drop(genre.columns[1], axis=1, inplace=True)
genre.columns = ['Genres']
genre_list = list(genre['Genres'])
occupation = pd.read_csv('u.occupation',delimiter="|",header=None)
```

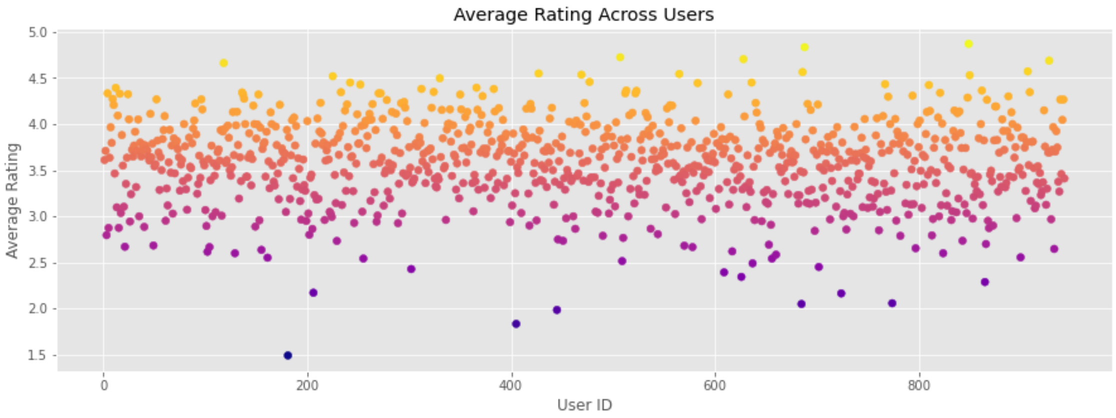
[2]:

```
## Showcase how the ratings vary across users
## Consider whether the plot is able to tell if most ratings are only from a␣
 ↪handful of users

## Merge datasets into one dataframe
```

```
data_user = pd.merge(data[['user id','movie id','rating','timestamp']],␣
↪user[['user id','age','gender','occupation']], on='user id')
full = pd.merge(data_user[['user id','movie␣
↪id','rating','timestamp','age','gender','occupation']], item[['movie␣
↪id','movie title','release date',
              'unknown','Action','Adventure','Animation',
              "Children's",'Comedy','Crime','Documentary','Drama','Fantasy',
              'Film-Noir','Horror','Musical','Mystery','Romance','Sci-Fi',
              'Thriller','War','Western']], on='movie id')

## Merged the data so that we get each users average rating and it's not just a␣
↪handful of users
## Shows us that most users have similar average ratings
## Color graph in a gradient in order to differentiate between high and low␣
↪averages
dftmp = full[['user id','rating']].groupby('user id').mean()
users = list(range(1,944))
dftmp['user id'] = users
dftmp['rating_dup'] = dftmp['rating']

dftmp2 = full[['user id','rating']].groupby('user id').mean()
dftmp2.hist(bins=16, grid=False, edgecolor='black', figsize=(10,5))
plt.xlabel('Rating')
plt.ylabel('# of Users')
plt.title('Distribution of Average Rating Across Users')
plt.show

plt.figure(figsize=(15, 5))
plt.scatter(dftmp['user id'],y=dftmp['rating'],c=dftmp['rating_dup'],cmap=plt.
↪cm.plasma)
plt.xlabel('User ID')
plt.ylabel('Average Rating')
plt.title('Average Rating Across Users')
plt.show
```

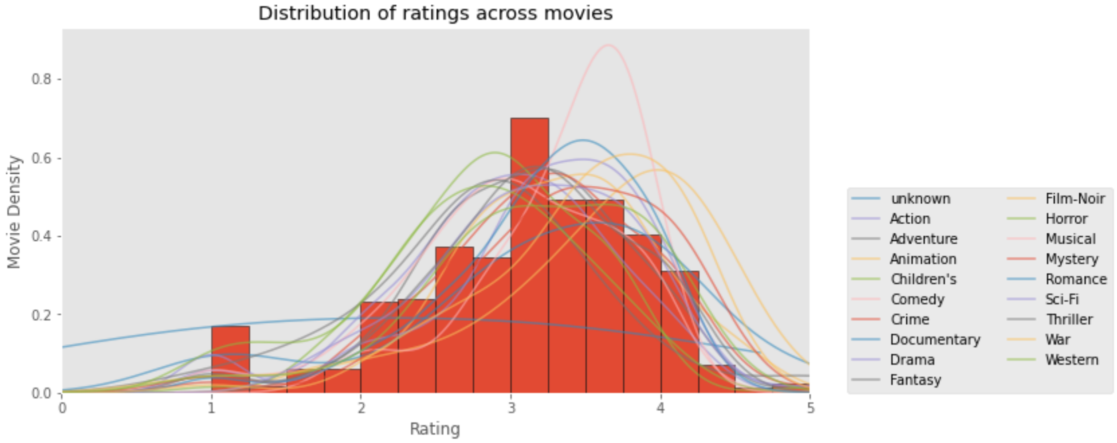
```
[2]: <function matplotlib.pyplot.show(close=None, block=None)>
```





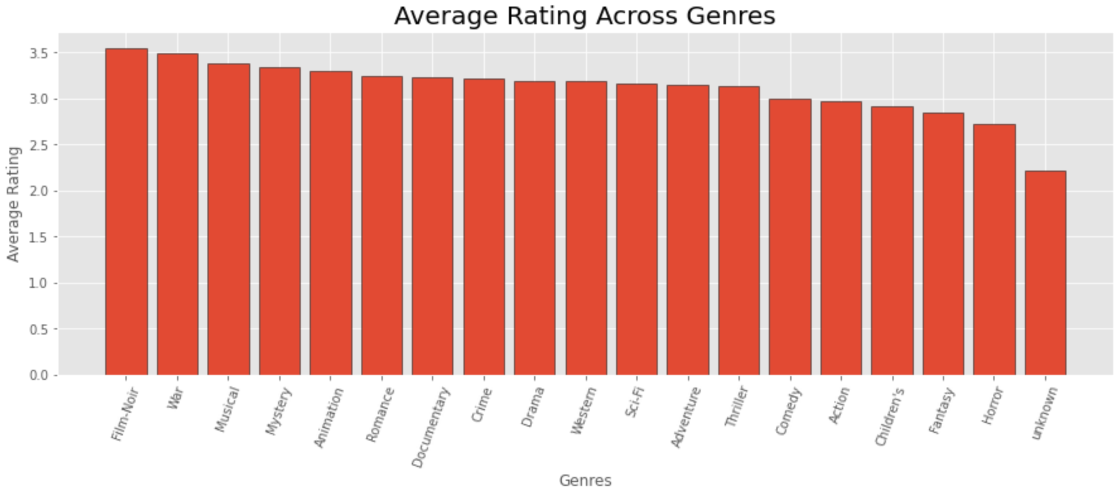
[3]:

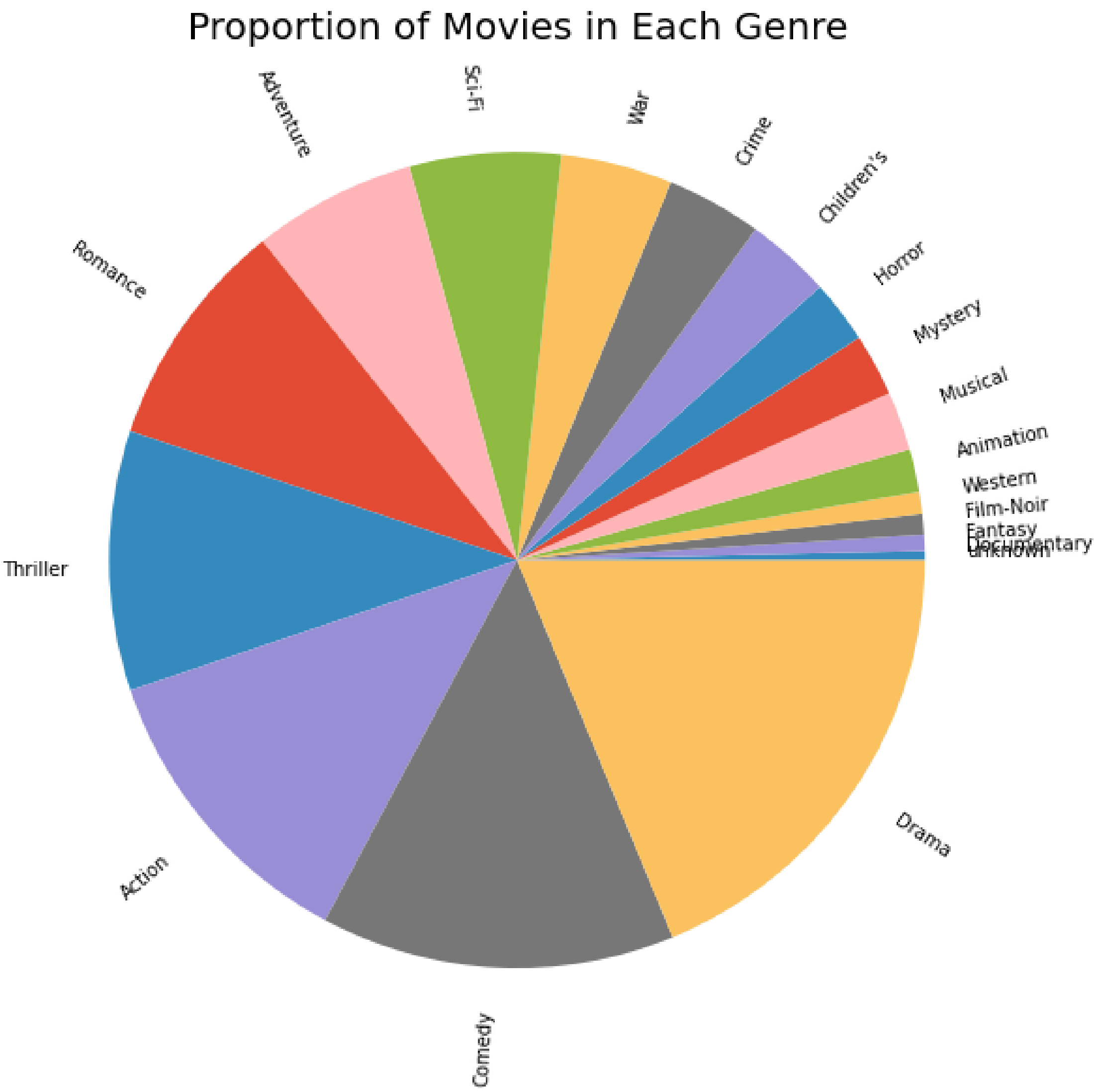
```
## Showcase how the ratings vary across items

dftmp = full[['movie id','rating']].groupby('movie id').mean()

dftmp.hist(bins=16, grid=False, density=True, edgecolor='black', figsize=(10,5))
for gen in genre_list:
    dftmp = full[full[gen]==1]
    dftmp = dftmp[['movie id','rating']].groupby('movie id').mean()
    dftmp['rating'].plot(grid=False, kind='kde', alpha=0.6, label=gen)
plt.legend(loc=(1.05,0), ncol=2)
plt.xlim(0,5)
plt.xlabel('Rating')
plt.ylabel('Movie Density')
plt.title("Distribution of ratings across movies")
plt.show()

## We see that movies overall are rated around 3
## Film-noir genre is often rated at 3.5 and unknown doesn't have enough data␣
 ↪to generate a normal bell curve
## However we can see that the distribution across genres is quite similar␣
 ↪besides these 2 categories
```



[43]:

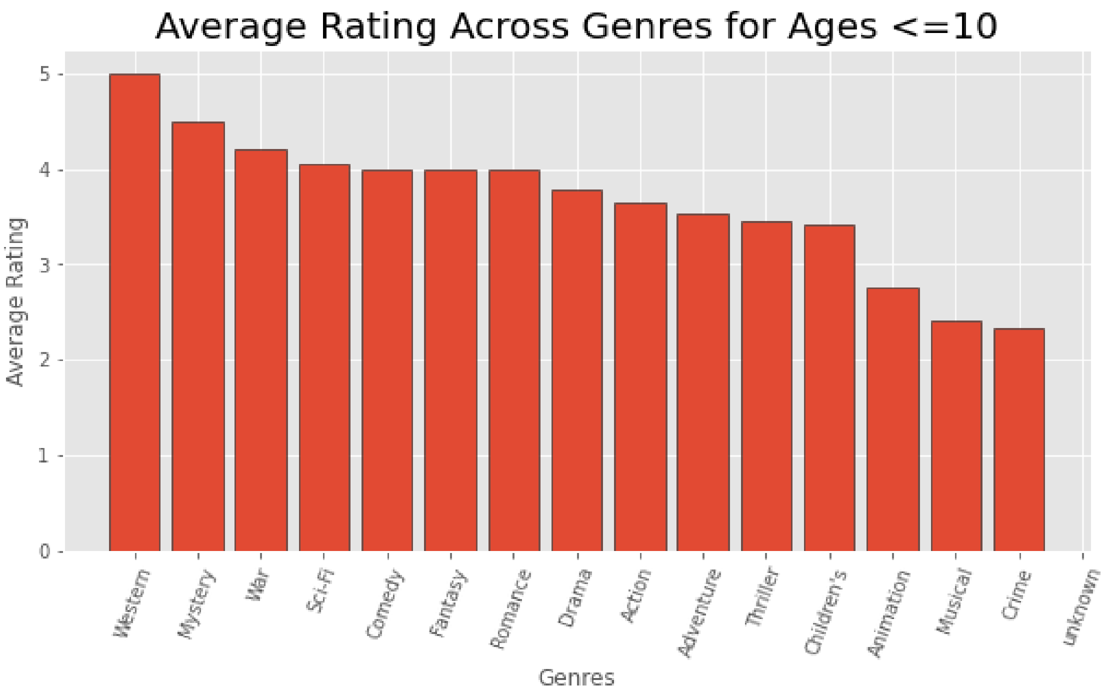
```
## Are there genres that are more highly rated than others?

dftmp_list = []
for gen in genre_list:
    dftmp = full[full[gen] == 1]
    dftmp = dftmp[['movie id','rating']].groupby('movie id').mean()
    dftmp[gen] = dftmp['rating']
    dftmp.drop(['rating'], axis=1, inplace= True)
    dftmp_list.append(dftmp)

dftmp = pd.concat(dftmp_list,ignore_index=True)
dicts = {}
for gen in genre_list:
     dicts[gen] = dftmp[gen].mean()
genre_avgrating = pd.DataFrame.from_dict(dicts,orient='index')
genre_avgrating['genre'] = genre_list
genre_avgrating.sort_values(by=[0],inplace=True,ascending=False)
plt.figure(figsize=(15, 5))
plt.bar(genre_avgrating['genre'],genre_avgrating[0],edgecolor='black')
plt.xlabel('Genres')
plt.xticks(rotation=70)
plt.ylabel('Average Rating')
plt.title('Average Rating Across Genres',fontsize=20)
plt.show

dictz = {}
for gen in genre_list:
     dictz[gen] = full[gen].mean()
proportion = pd.DataFrame.from_dict(dictz,orient='index')
proportion['genre'] = genre_list
proportion.sort_values(by=[0],inplace=True,ascending=True)
plt.figure(figsize=(10, 10))
plt.pie(proportion[0], labels=proportion['genre'], startangle=0,␣
 ↪rotatelabels=True)
plt.title('Proportion of Movies in Each Genre',fontsize=20)
plt.show()

## We see that film noir is the highest rated but it makes up an extremely␣
 ↪small proportion of the movies overall
## The more movies in a genre have been rated, the more reliable the average␣
 ↪rating
## This is why it is important to analyze these two graphs in conjunction
```

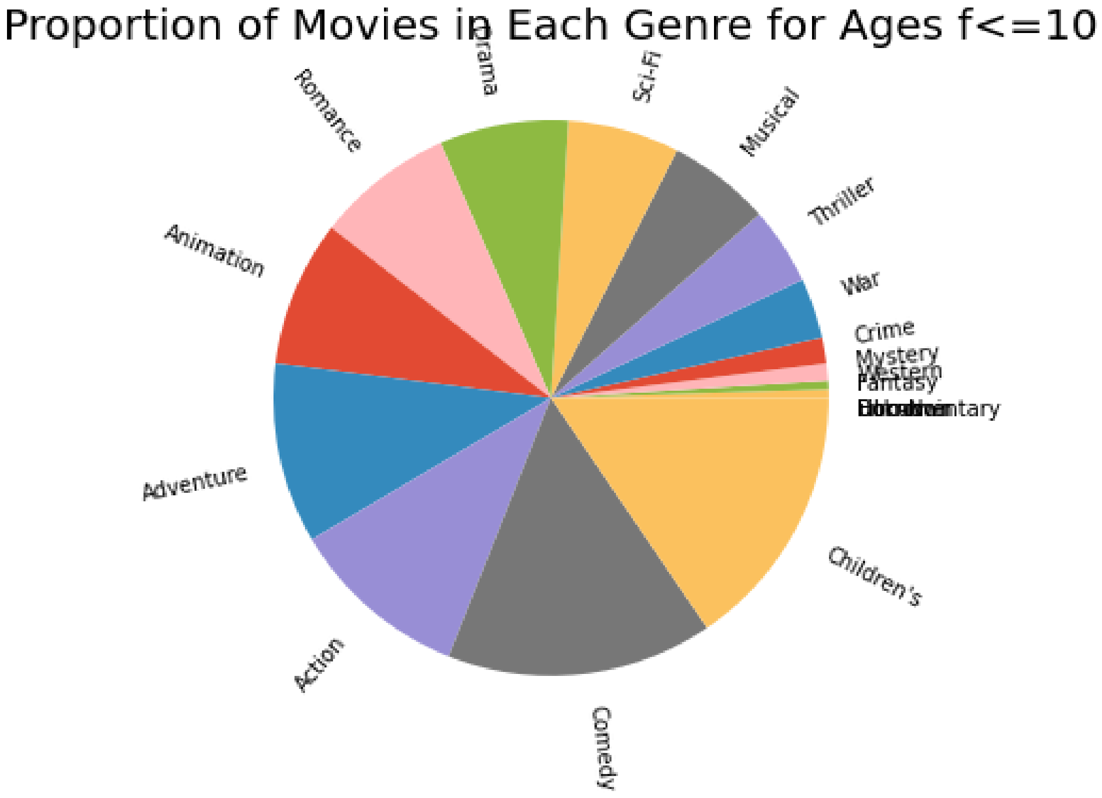


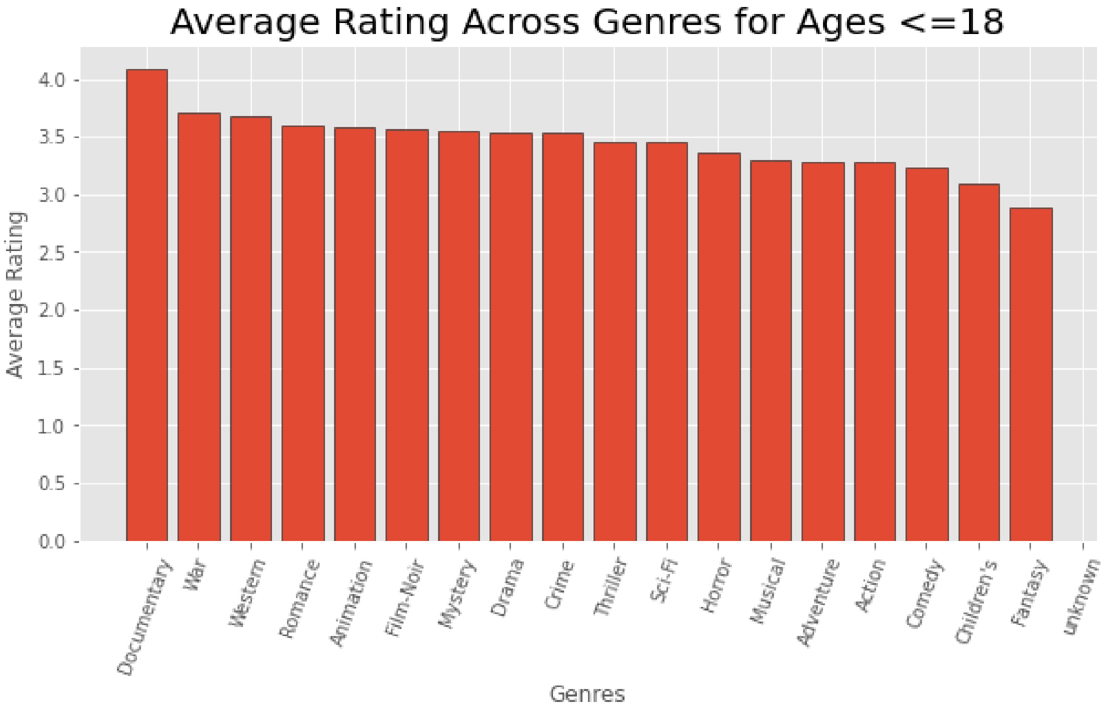


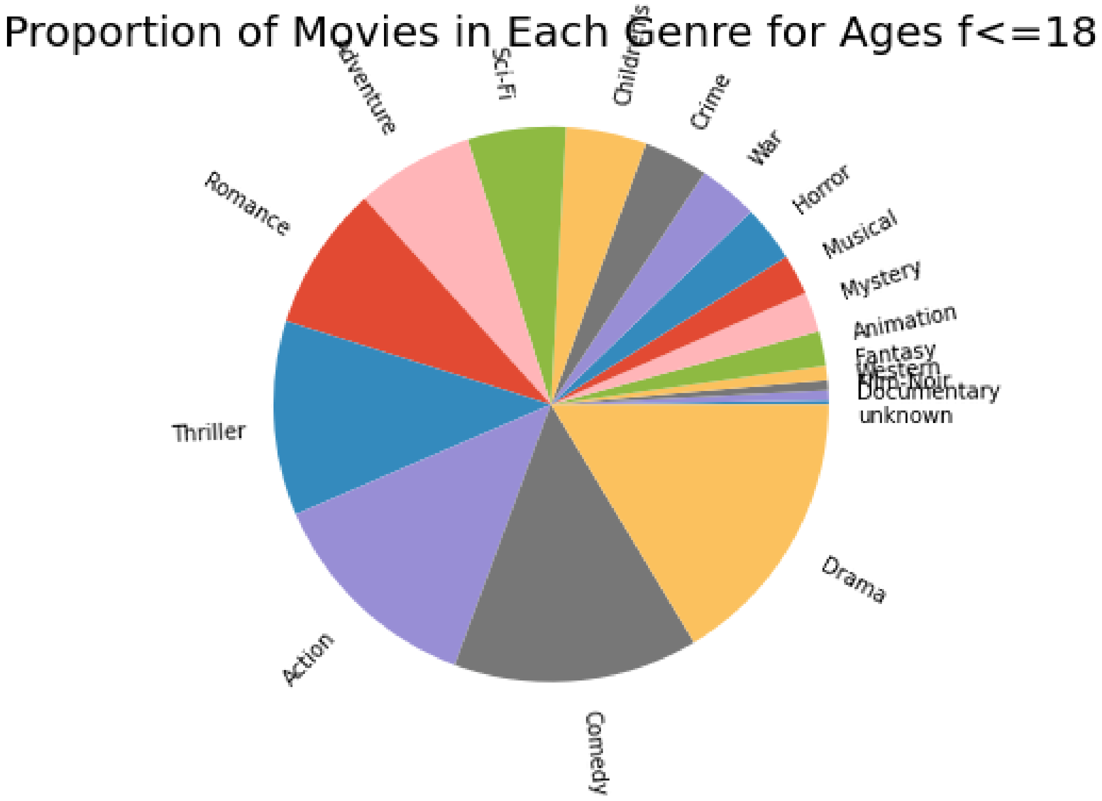
[54]:

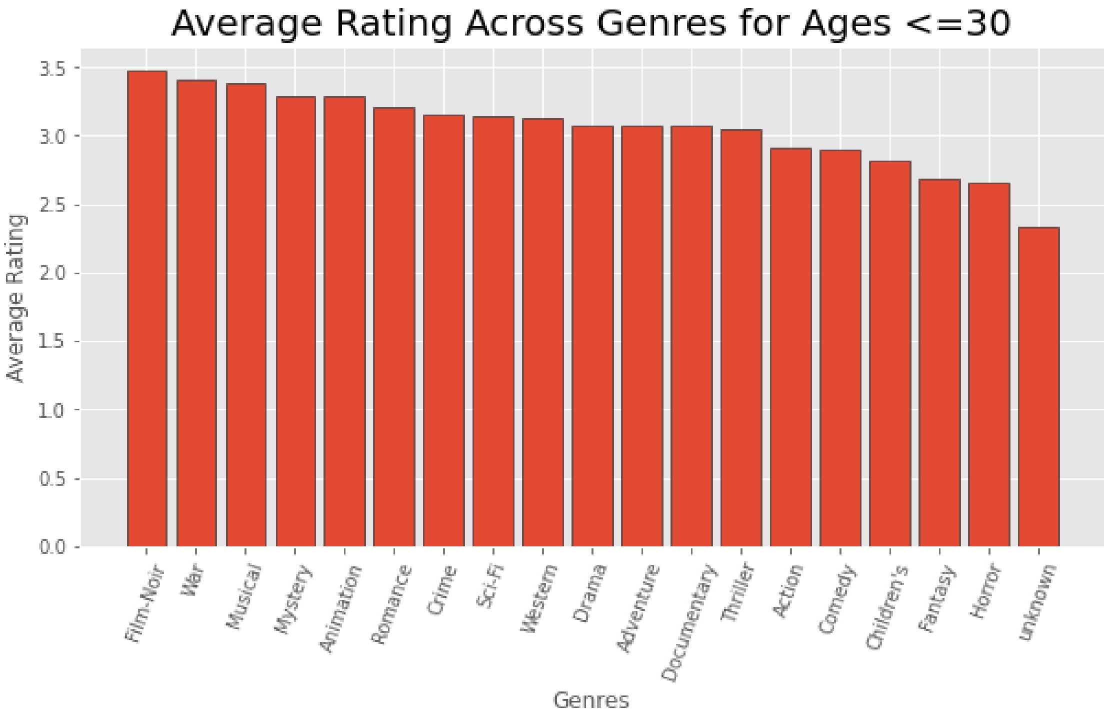
```
## What age groups prefer what genres based on ratings?
## You can bin respective ages to your preference

bins= [0,10,18,30,50,200]
labels = ['<=10','<=18','<=30','<=50', '50+']
full['age group'] = pd.cut(full['age'], bins=bins, labels=labels, right=True)

for l in labels:
    age_grouped = full[full['age group'] == l]
    dftmp_list = []
    for gen in genre_list:
        dftmp = age_grouped[age_grouped[gen] == 1]
        dftmp = dftmp[['movie id','rating']].groupby('movie id').mean()
        dftmp[gen] = dftmp['rating']
        dftmp.drop(['rating'], axis=1, inplace= True)
        dftmp_list.append(dftmp)
    dftmp = pd.concat(dftmp_list,ignore_index=True)
    dicts = {}
    for gen in genre_list:
        dicts[gen] = dftmp[gen].mean()
    genre_avgrating = pd.DataFrame.from_dict(dicts,orient='index')
    genre_avgrating['genre'] = genre_list
    genre_avgrating.sort_values(by=[0],inplace=True,ascending=False)
    plt.figure(figsize=(10, 5))
    plt.bar(genre_avgrating['genre'],genre_avgrating[0],edgecolor='black')
    plt.xlabel('Genres')
    plt.xticks(rotation=70)
    plt.ylabel('Average Rating')
    plt.title(f'Average Rating Across Genres for Ages {l}',fontsize=20)
    plt.show
    dictz = {}
    for gen in genre_list:
        dictz[gen] = age_grouped[gen].mean()
    proportion = pd.DataFrame.from_dict(dictz,orient='index')
    proportion['genre'] = genre_list
    proportion.sort_values(by=[0],inplace=True,ascending=True)
    plt.figure(figsize=(6, 6))
    plt.pie(proportion[0], labels=proportion['genre'], startangle=0,␣
↪rotatelabels=True)
    plt.title(f'Proportion of Movies in Each Genre for Ages f{l}',fontsize=20)
    plt.show()

## This shows us the highest average rated genres for each age group first
## Then shows us the most frequently watched genres for each age group
## Should try to combine these measurements into one for most useful insights
## (i.e. normalize for amount of ratings per genre)
```

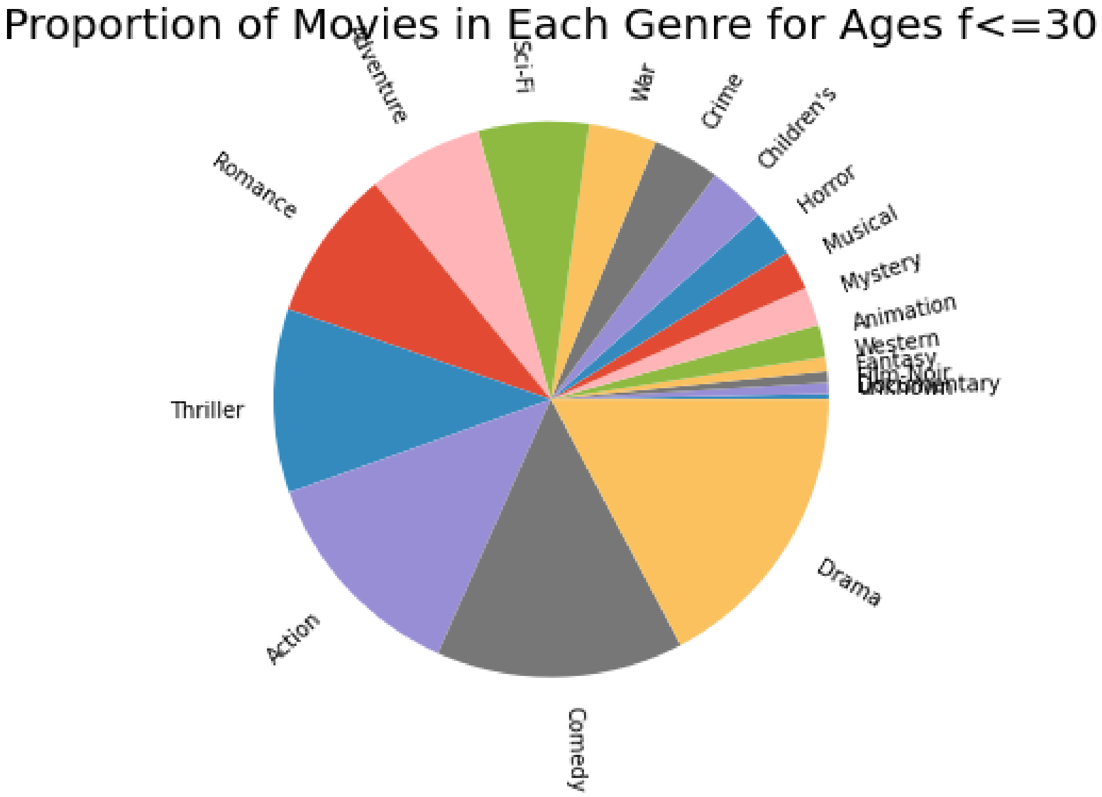

Average Rating Across Genres for Ages <=10
Average Rating
0
1
2
3
4
5
Western
Mystery
War
Sci-Fi
Comedy
Fantasy
Romance
Drama
Action
Adventure
Thriller
Children's
Animation
Musical
Crime
unknown
Genres

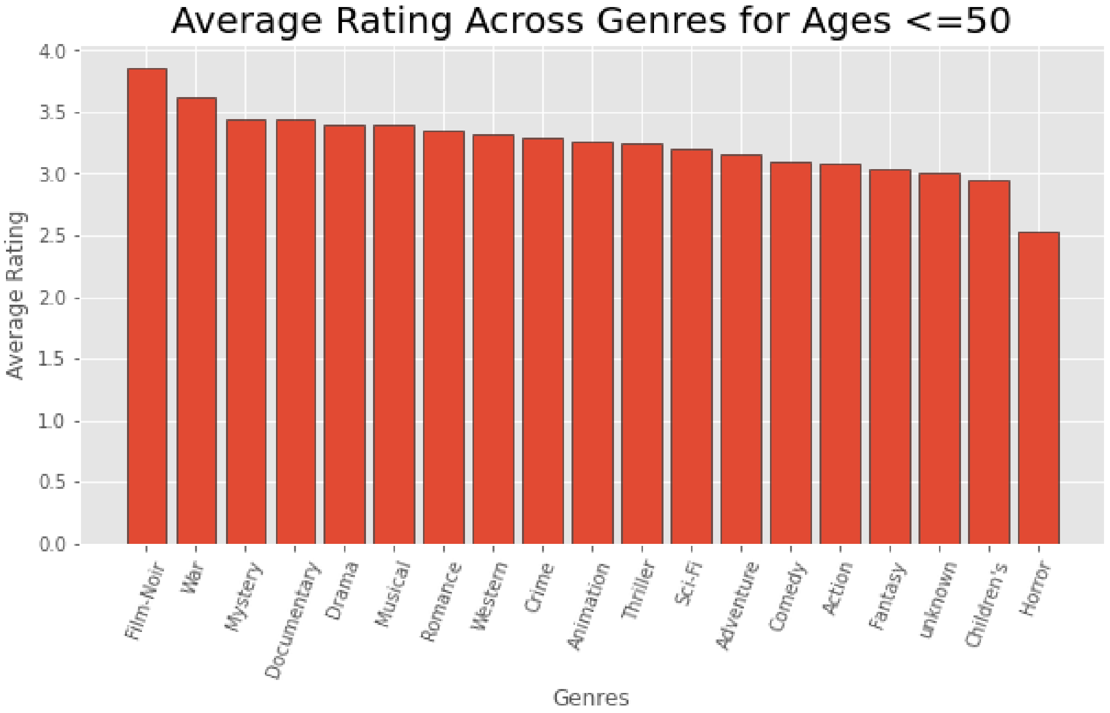

Proportion of Movies in Each Genre for Ages f<=10
Drama
Sci-Fi
Musical
Thriller
War
Crime
Mystery
Western
Fantasy
Documentary
Children's
Comedy
Action
Adventure
Animation
Romance

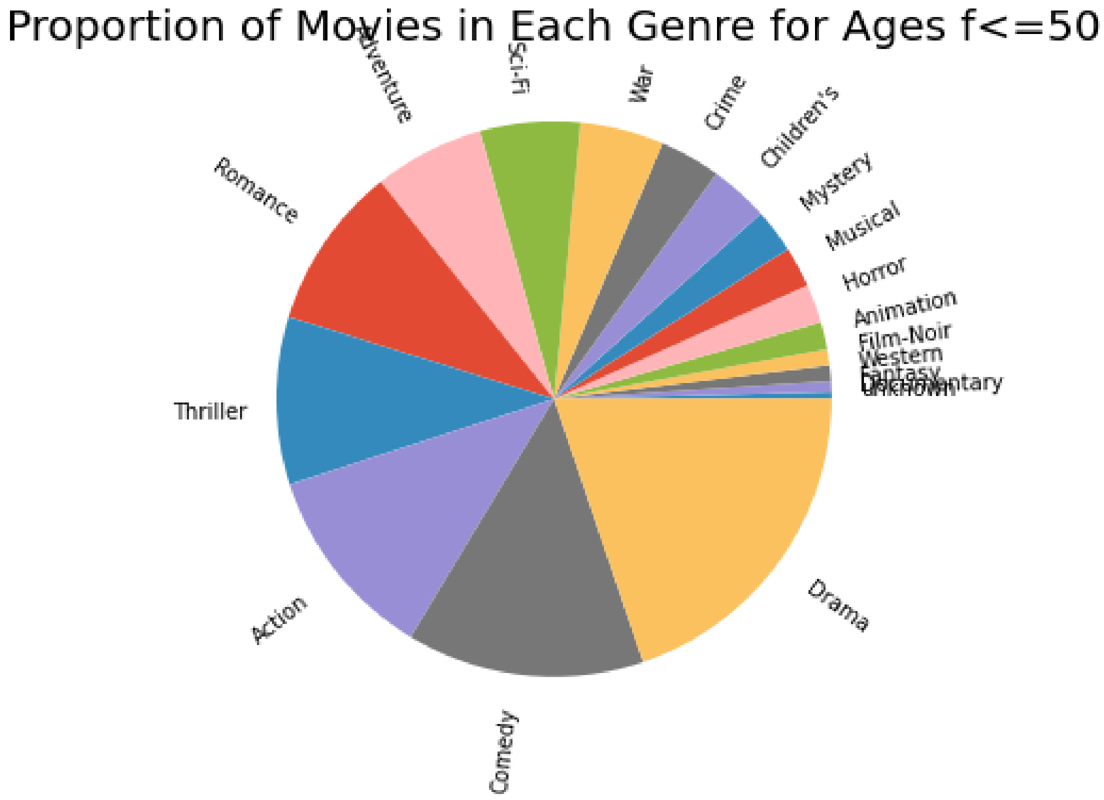

Average Rating Across Genres for Ages <=18
Average Rating
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Documentary
War
Western
Romance
Animation
Film-Noir
Mystery
Drama
Crime
Thriller
Sci-Fi
Horror
Musical
Adventure
Action
Comedy
Children's
Fantasy
unknown
Genres

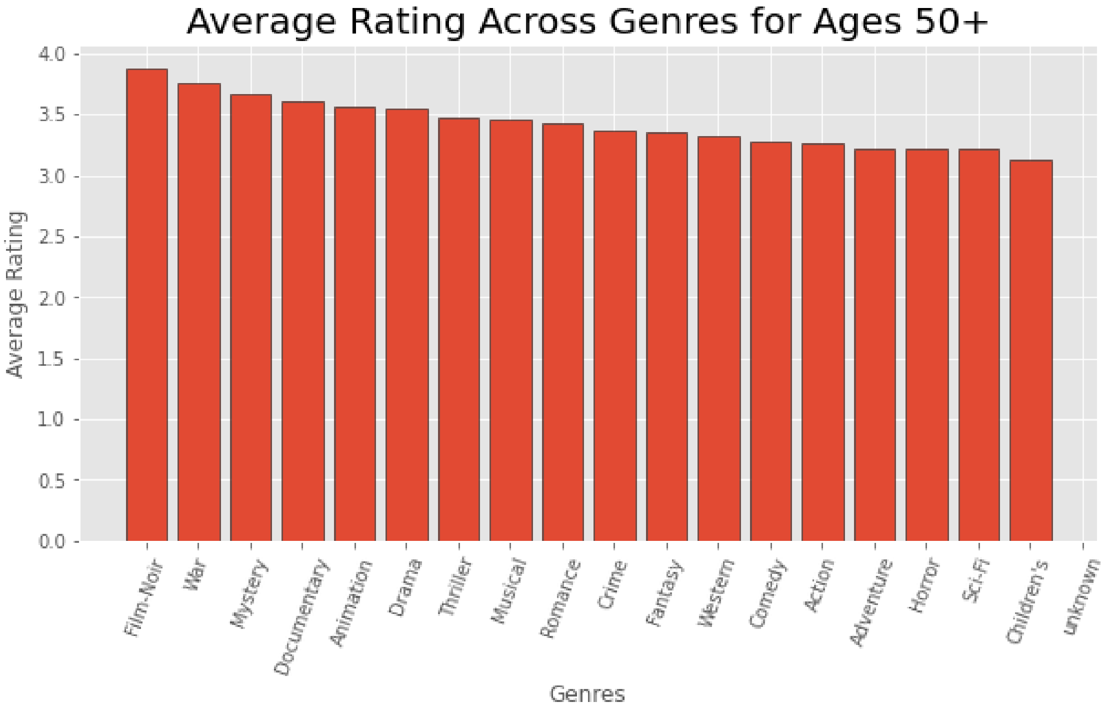

Proportion of Movies in Each Genre for Ages f<=18
Adventure
Sci-Fi
Children's
Crime
War
Horror
Musical
Mystery
Animation
Fantasy
Western
Film-Noir
Documentary
unknown
Romance
Thriller
Drama
Action
Comedy

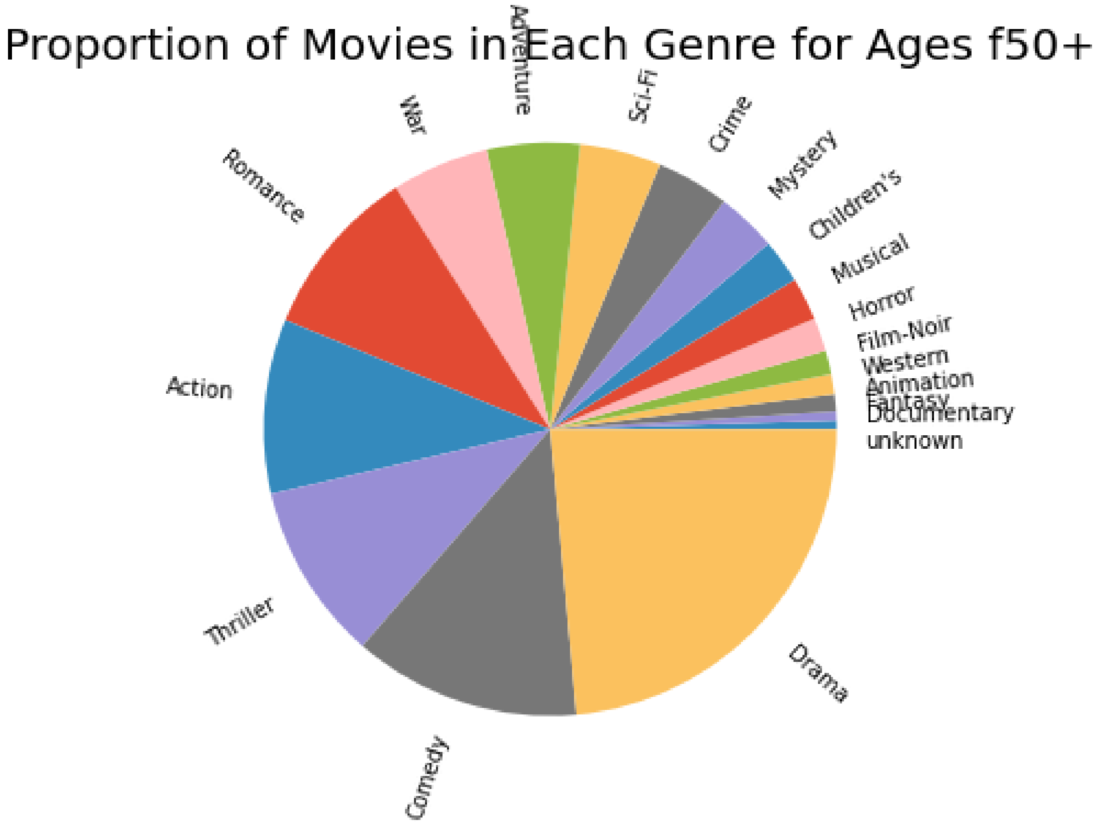

Average Rating Across Genres for Ages <=30
Average Rating
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Film-Noir
War
Musical
Mystery
Animation
Romance
Crime
Sci-Fi
Western
Drama
Adventure
Documentary
Thriller
Action
Comedy
Children's
Fantasy
Horror
unknown
Genres


Proportion of Movies in Each Genre for Ages f<=30
Adventure
Sci-Fi
War
Crime
Children's
Horror
Musical
Mystery
Animation
Western
Fantasy
Film-Noir
Documentary
Unknown
Romance
Thriller
Drama
Action
Comedy


Average Rating Across Genres for Ages <=50
Average Rating
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Film-Noir
War
Mystery
Documentary
Drama
Musical
Romance
Western
Crime
Animation
Thriller
Sci-Fi
Adventure
Comedy
Action
Fantasy
unknown
Children's
Horror
Genres


Proportion of Movies in Each Genre for Ages f<=50
Adventure
Sci-Fi
War
Crime
Children's
Mystery
Romance
Musical
Horror
Animation
Film-Noir
Western
Fantasy
Documentary
unknown
Thriller
Action
Drama
Comedy


Average Rating Across Genres for Ages 50+
Average Rating
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Film-Noir
War
Mystery
Documentary
Animation
Drama
Thriller
Musical
Romance
Crime
Fantasy
Western
Comedy
Action
Adventure
Horror
Sci-Fi
Children's
unknown
Genres



2 EXERCISE 2

2.1 Implementing basic matrix factorization (MF) technique for recommender systems

[78]:

```
## Split data into 80-10-10

train = full.sample(frac=0.8,random_state=3116)
ytrain = train['rating']
xtrain = train.drop(['rating'], axis=1, inplace= True)
leftover = full.drop(train.index)
validation = leftover.sample(frac=0.5,random_state=3116)
yval = validation['rating']
xval = validation.drop(['rating'],axis=1, inplace= True)
test = leftover.drop(validation.index)
ytest = test['rating']
xtest = test.drop(['rating'],axis=1, inplace= True)
```

```
print(train.shape,validation.shape,test.shape)
```

```
(80000, 28) (10000, 28) (10000, 28)
```

[74]:

```
def RMSE(y,y_hat):
    rmse = ((np.sum((y-y_hat)**2))/(len(y)))**0.5
    return rmse

## Based on https://github.com/Quang-Vinh/matrix-factorization
def obtain_rating(P,Q,users,items,ubias,ibias,bias):
    pred = bias+ubias.loc[users]+ibias.loc[item]+P.loc[users].dot(Q.loc[items].
 ↪T)
    return pred

def sgd(sample,P,Q,ubias,ibias,bias,alpha,beta):
    for i,j,r in sample:
        pred = obtain_rating(P,Q,i,j,ubias,ibias,bias)
        error = r-pred
        ubias.loc[i] += alpha*(e[0]-beta*ubias.loc[i])
        ibias.loc[j] += alpha*(e[0]-beta*ibias.loc[i])
        P.loc[i] += alpha*(e[0]*Q.loc[j]-beta*P.loc[i])
        Q.loc[j] += alpha*(e[0]*P.loc[i]-beta*Q.loc[j])
    return P,Q,ubias,ibias

def mf(matrix,K,alpha,beta,epoch,sample,tsample):
    users,items = matrix.shape
    P = pd.DataFrame(np.random.normal(scale=1./K,␣
 ↪size=(users,k)),columns=[0,1],index=sample['user id'].unique()
    Q = pd.DataFrame(np.random.normal(scale=1./K,␣
 ↪size=(items,k)),columns=[0,1],index=sample['movie id'].unique()
    ubias = pd.DataFrame(0,columns=[0],index=sample['user id'].unique())
    ibias = pd.DataFrame(0,columns=[0],index=sample['movie id'].unique())
    sample = sample.values
    matrix = matrix.values
    bias = np.mean(matrix[np.where(matrix!=0)])
    rmse = []
    for i in range(epoch):
        np.random.shuffle(sample)
        P,Q,ubias,ibias = sgd(sample,P,Q,ubias,ibias,bias,alpha,beta)
        err = RMSE(y,y_hat)
        rmse.append(err)
    return rmse,P,Q,ubias,ibias,bias
```

[82]:

```
def predict(matrix,test):
    y_hat = []
    values = []
    for i in range(0,len(test)):
        marker = 1
        if(test.iloc[i]['user id'] in matrix.index):
            if(test.iloc[i]['movie id'] in matrix.columns):
                marker = 0
        if(marker):
            values.append(i)
        else:
            y_hat.append(matrix[test.iloc[i]['movie id']][test.iloc[i]['user␣
 ↪id']])
    return y_hat,test.drop(test.index[values])

rmse,P,Q,ubias,ibias,bias = mf()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-82-297f64c16c54> in <module>
     20     return y_hat,test.drop(test.index[values])
     21 
---> 22 nP, nQ = matrix_factorization(train_array, P, Q, k_latent)
     23 y_hat,new = predict(validation,xval)
     24 rmse_validation = rmse(yval,y_hat)

<ipython-input-74-cbfad11f3244> in matrix_factorization(R, P, Q, K, steps,␣
 ↪alpha, beta)
      8         for i in range(len(R)):
      9             for j in range(len(R[i])):
---> 10                 if R[i][j] > 0:
     11                     eij = R[i][j] - np.dot(P[i,:],Q[:,j])
     12                     for k in range(K):

TypeError: '>' not supported between instances of 'str' and 'int'
```

3 EXERCISE 3

3.1 Recommender Systems using matrix factorization sckitlearn

[ ]:

```
def divide(data,k):
    size_of_k = math.floor(len(data/k))
    data_kdivided = []
    c = 0
    for i in range(0,k):
        dataset = pd.DataFrame(data.head(0))
        for j in range(i*size_of_k,(i*size_of_k)+size_of_k):
            dataset = dataset.append(data.iloc[j])
            c = c+1
        data_kdivided.append(dataset)
    for j in range(c,len(data)):
        data_kdivided[k-1]=data_kdivided[k-1].append(data.iloc[j])
    return data_kdivided

def k_traintest(x,k):
    kfold_data = []
    for i in range(0,k):
        xtest = x[i]
        xtrain = pd.DataFrame()
        for j in range(0,k):
            if i != j:
                xtrain = xtrain.append(x[j])
        final = dict([('xtrain',xtrain),('xtest',xtest)])
        kfold_data.append(final)
    return kfold_data

def kfold(xtrain,k,alpha,lamda):
    errors = []
    xtrain_k = divide(xtrain,k)
    kdata = k_traintest(xtrain_k,k)
    for i in range(0,k):
        matrixx = kdata[i]['xtrain'].pivot(index='user id',columns='movie␣
 ↪id',values='rating')
        matrixx = matrixx.replace(np.nan,0)
        rmse,P,Q,ubias,ibias,bias =␣
 ↪mf(matrixx,2,alpha,lamda,1,kdata[i]['xtrain'][['user id','movie␣
 ↪id','rating']],kdata[i]['xtest'][['user id','movie id','rating']])
        errors.append(sum(rmse)/len(rmse))
    return sum(errors)/k
```

[72]:

```
rmse = kfold(train,3,0.01,0.01)
rmse
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\anaconda3\lib\site-packages\pandas\core\ops\array_ops.py in␣
 ↪_na_arithmetic_op(left, right, op, is_cmp)
    141     try:
--> 142         result = expressions.evaluate(op, left, right)
    143     except TypeError:

~\anaconda3\lib\site-packages\pandas\core\computation\expressions.py in␣
 ↪evaluate(op, a, b, use_numexpr)
    234             # error: "None" not callable
--> 235             return _evaluate(op, op_str, a, b)  # type: ignore[misc]
    236     return _evaluate_standard(op, op_str, a, b)
```

```
~\anaconda3\lib\site-packages\pandas\core\computation\expressions.py in␣
 ↪_evaluate_numexpr(op, op_str, a, b)
    119     if result is None:
--> 120         result = _evaluate_standard(op, op_str, a, b)
    121

~\anaconda3\lib\site-packages\pandas\core\computation\expressions.py in␣
 ↪_evaluate_standard(op, op_str, a, b)
     68     with np.errstate(all="ignore"):
---> 69         return op(a, b)
     70

TypeError: unsupported operand type(s) for /: 'str' and 'int'

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
<ipython-input-72-cc06344e3055> in <module>
----> 1 rmse = kfold(train,3,0.01,0.01)
      2 rmse

<ipython-input-71-75fa16f74136> in kfold(xtrain, k, alpha, lamda)
     27 def kfold(xtrain,k,alpha,lamda):
     28     errors = []
---> 29     xtrain_k = divide(xtrain,k)
     30     kdata = k_traintest(xtrain_k,k)
     31     for i in range(0,k):

<ipython-input-71-75fa16f74136> in divide(data, k)
      1 def divide(data,k):
----> 2     size_of_k = math.floor(len(data/k))
      3     data_kdivided = []
      4     c = 0
      5     for i in range(0,k):

~\anaconda3\lib\site-packages\pandas\core\ops\common.py in new_method(self,␣
 ↪other)
     63         other = item_from_zerodim(other)
     64
---> 65         return method(self, other)
     66
     67     return new_method

~\anaconda3\lib\site-packages\pandas\core\arraylike.py in __truediv__(self,␣
 ↪other)
    111     @unpack_zerodim_and_defer("__truediv__")
    112     def __truediv__(self, other):
--> 113         return self._arith_method(other, operator.truediv)
```

```
    114
    115     @unpack_zerodim_and_defer("__rtruediv__")

~\anaconda3\lib\site-packages\pandas\core\frame.py in _arith_method(self, other ␣
 ↪op)
   5980         self, other = ops.align_method_FRAME(self, other, axis,␣
 ↪flex=True, level=None)
   5981
-> 5982         new_data = self._dispatch_frame_op(other, op, axis=axis)
   5983         return self._construct_result(new_data)
   5984

~\anaconda3\lib\site-packages\pandas\core\frame.py in _dispatch_frame_op(self,␣
 ↪right, func, axis)
   6006         if not is_list_like(right):
   6007             # i.e. scalar, faster than checking np.ndim(right) == 0
-> 6008             bm = self._mgr.apply(array_op, right=right)
   6009             return type(self)(bm)
   6010

~\anaconda3\lib\site-packages\pandas\core\internals\managers.py in apply(self,␣
 ↪f, align_keys, ignore_failures, **kwargs)
    423             try:
    424                 if callable(f):
--> 425                     applied = b.apply(f, **kwargs)
    426                 else:
    427                     applied = getattr(b, f)(**kwargs)

~\anaconda3\lib\site-packages\pandas\core\internals\blocks.py in apply(self,␣
 ↪func, **kwargs)
    376         """
    377         with np.errstate(all="ignore"):
--> 378             result = func(self.values, **kwargs)
    379
    380         return self._split_op_result(result)

~\anaconda3\lib\site-packages\pandas\core\ops\array_ops.py in␣
 ↪arithmetic_op(left, right, op)
    187     else:
    188         with np.errstate(all="ignore"):
--> 189             res_values = _na_arithmetic_op(lvalues, rvalues, op)
    190
    191     return res_values

~\anaconda3\lib\site-packages\pandas\core\ops\array_ops.py in␣
 ↪_na_arithmetic_op(left, right, op, is_cmp)
    147             #  will handle complex numbers incorrectly, see GH#32047
    148             raise
```

```
--> 149         result = _masked_arith_op(left, right, op)
    150 
    151     if is_cmp and (is_scalar(result) or result is NotImplemented):

~\anaconda3\lib\site-packages\pandas\core\ops\array_ops.py in␣
 ↪_masked_arith_op(x, y, op)
    109         if mask.any():
    110             with np.errstate(all="ignore"):
--> 111                 result[mask] = op(xrav[mask], y)
    112 
    113     result, _ = maybe_upcast_putmask(result, ~mask, np.nan)

TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

```
[ ]:
```