SVKM'S NARSEE MONJEE MUKESH PATEL SCHOOL OF TECHNOLOGY MANAGEMENT & ENGINEERING, NMIMS



IMPLEMENTATION

FOR

**DATA ANALYSIS**

**for**

**MovieLens Dataset**

**(Predictive Modelling)**

**By,**

**Aditi Anup (B001), Samreen Reyaz (B005), Dishant Bhatt (B013) and Gabriel Francis (B027)**

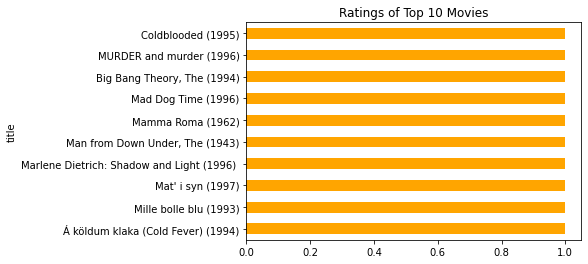
**Case Title: Analysis of 10,000 samples of MovieLens Data**

**College: Mukesh Patel School of Technology Management & Engineering, NMIMS**

# DATA ANALYSIS CODE:

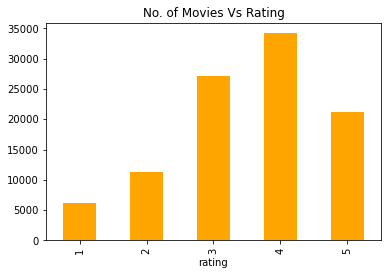
most\_rated=movielens.groupby('title').size().sort\_values(ascending=True)[:10]

most\_rated.plot(kind="barh",title="Ratings of Top 10 Movies",label="count",color="orange")



most\_rated=movielens.groupby('rating').size()

most\_rated.plot(kind="bar",title="No. of Movies Vs Rating",label="count",color="orange")



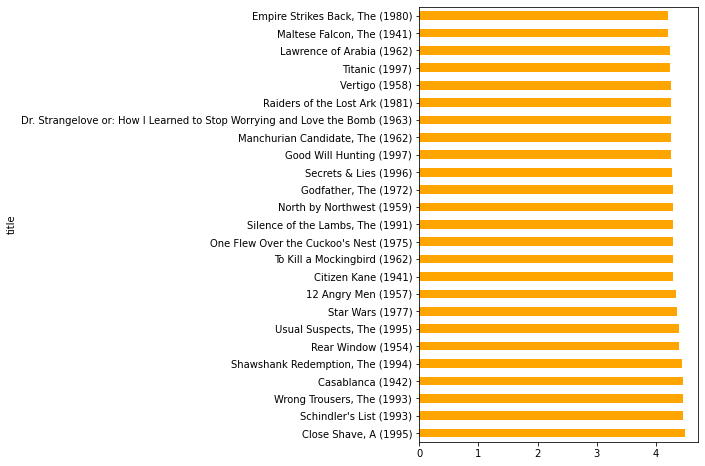
movie\_stat = movielens.groupby('title').agg({'rating':[np.size,np.mean]})

movie\_stat.sort\_values([('rating','mean')],ascending=False).head()

most100=movie\_stat['rating']['size'] >= 100

most\_rated\_mean=movie\_stat[most100].sort\_values([('rating', 'mean')], ascending=False)

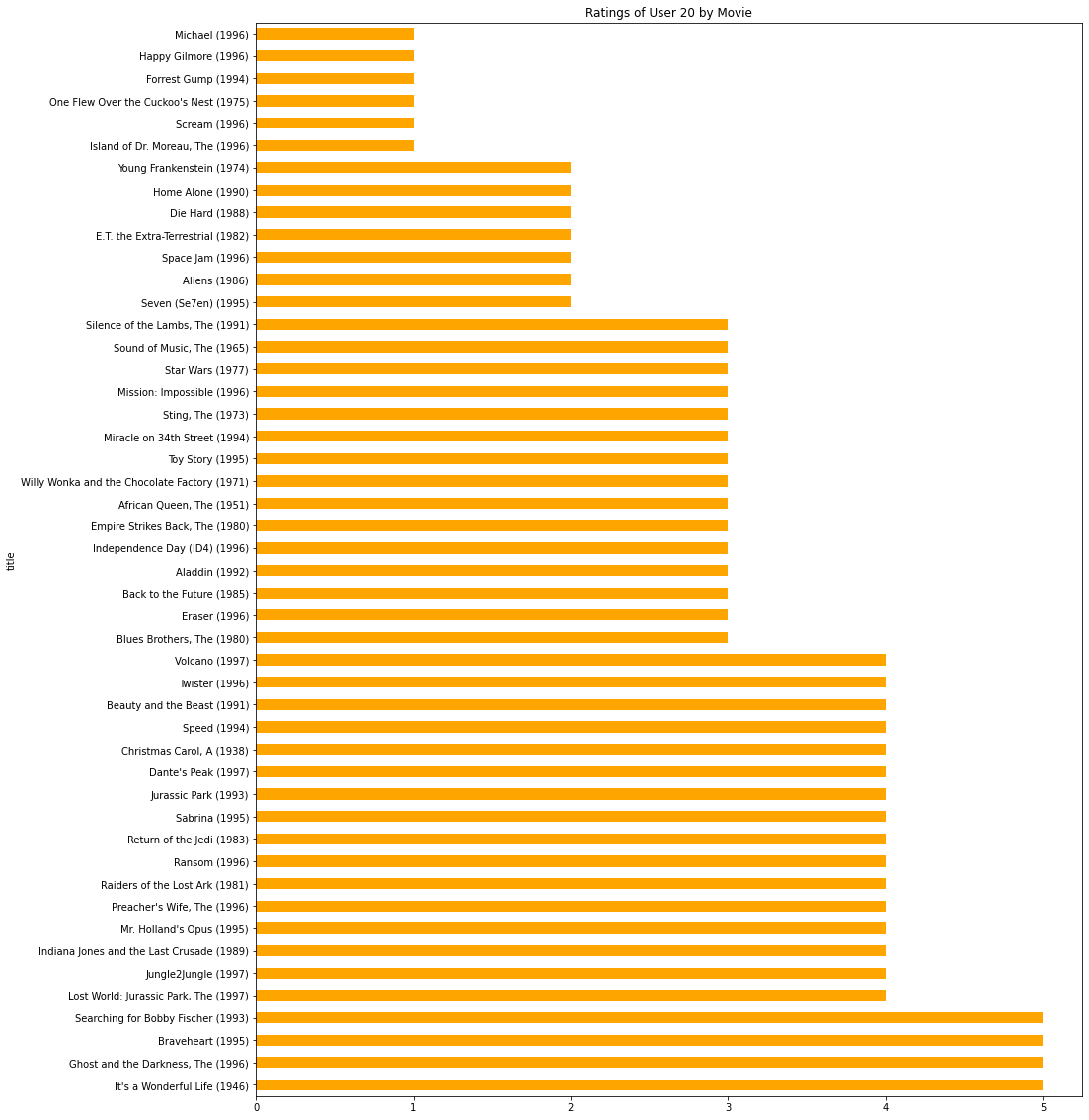
most\_rated\_mean['rating']['mean'].sort\_values(ascending=False)[:25].plot(kind="barh",color="orange",figsize=(5,8))



user1=movielens[movielens.user\_id==20]

user1=user1.groupby('title').agg([np.size,np.mean])

user1['rating']['mean'].sort\_values(ascending=False).plot(kind="barh",color = "orange",figsize=(15,20),title="Ratings of User 20 by Movie",label="Movie Name")

****

words = dict()

for s in t:

words[s[0]] = s[1]

tone = 11# define the color of the words

f, ax = plt.subplots(figsize=(14, 6))

wordcloud = WordCloud(width=800,height=300, background\_color='black',

max\_words=1628,relative\_scaling=0.7,

color\_func = random\_color\_func,

normalize\_plurals=False)

wordcloud.generate\_from\_frequencies(words)

plt.imshow(wordcloud, interpolation="bilinear")

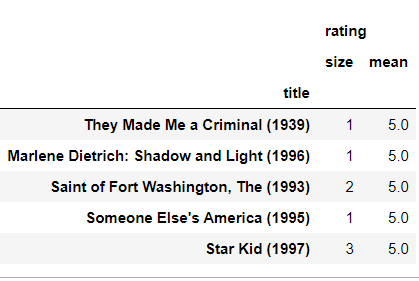
plt.axis('off')

plt.show()



movie\_stat = movielens.groupby('title').agg({'rating':[np.size,np.mean]})

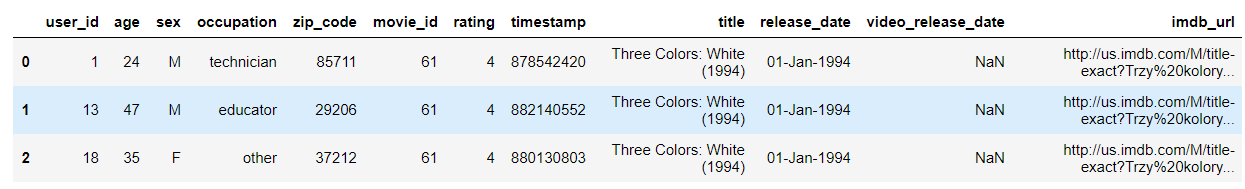
movie\_stat.sort\_values([('rating','mean')],ascending=False).head()



movielens=pd.merge(users,ratings)

movielens=pd.merge(movielens,movies)

movielens.head(3)



**Favorite Genre based on Occupation:**

import pandas as pd

#import of review data

cols = ["user id","item id","rating","timestamp"]

#encoding using ISO-8859-1 is used because utf-8 does not support all the characters in movie names

df\_data = pd.read\_csv("ml-100k/u.data",sep="\t",names=cols,header=None,encoding="ISO-8859-1")

#verifying the sucessful import of review data

print(df\_data.head())

#import of moviedata

cols = ["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"]

df\_movie = pd.read\_csv("ml-100k/u.item",sep="|",names=cols,header=None,encoding="ISO-8859-1")

#verifying the sucessful import of movie data

print(df\_movie.head())

#import of user data

cols = ["user id","age","gender","occupation","zip code"]

df\_user = pd.read\_csv("ml-100k/u.user",sep="|",names=cols,header=None,encoding="ISO-8859-1")

#verifying the sucessful import of user data

print(df\_user.head())

#frequency binning the ages into age groups as it will be easier for future analysis

df\_user['age\_group'] = pd.qcut(df\_user['age'],q=10,precision=0)

#the bins are of unequal size due to repeating values in a bin

df\_user['age\_group'].value\_counts()

df\_movie.drop(["movie id",

"movie title",

"release date",

"video release date",

"IMDb URL",

"unknown"],axis=1).sum(axis = 0, skipna = True)

df = pd.merge(pd.merge(df\_data,

df\_user[["user id",

"age",

"gender",

"occupation"]],

on='user id',

how='left'),

df\_movie.drop(['IMDb URL'],axis=1),

left\_on = 'item id',

right\_on = 'movie id',

how ='left')

df.head()

def release\_year(row):

return str(row['release date'])[-4:]

def release\_month(row):

return str(row['release date'])[3:6]

def release\_date(row):

return str(row['release date'])[:2]

df['release\_year'] = df.apply(lambda row: release\_year(row), axis=1)

df['release\_month'] = df.apply(lambda row: release\_month(row), axis=1)

df['release\_day'] = df.apply(lambda row: release\_date(row), axis=1)

df = df.drop(['release date'])

df\_job\_genre = df[['occupation',

'rating',

"Action",

"Adventure",

"Animation",

"Children's",

"Comedy",

"Crime",

"Documentary",

"Drama",

"Fantasy",

"Film-Noir",

"Horror",

"Musical",

"Mystery",

"Romance",

"Sci-Fi",

"Thriller",

"War",

"Western"]]

def select\_genre(row):

for key,value in row.items():

if value==1:

return key

df\_job\_genre['genre']= df\_job\_genre.apply(lambda row: select\_genre(row.iloc[2:]),axis=1)

df\_job\_genre.drop(["Action",

"Adventure",

"Animation",

"Children's",

"Comedy",

"Crime",

"Documentary",

"Drama",

"Fantasy",

"Film-Noir",

"Horror",

"Musical",

"Mystery",

"Romance",

"Sci-Fi",

"Thriller",

"War",

"Western"],

inplace=True,

axis=1)

df\_job\_genre\_grouped = df\_job\_genre.groupby(['occupation','genre']).mean()

df\_favorite\_genre = df\_job\_genre\_grouped.reset\_index(level='occupation').groupby('occupation')['rating'].idxmax().reset\_index(name='favorite\_genre')

df\_favorite\_genre

We also found the most popular genre of movies based on the occupation of a user



# MODELS IMPLEMENTED:

## MODEL 1, 2 AND 3: LOGISTIC REGRESSION, DECISION TREE AND RANDOM FOREST CLASSIFICATION

import pandas as pd

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

#import of review data

cols = ["user id",

"item id",

"rating",

"timestamp"]

#encoding using ISO-8859-1 is used because utf-8 does not support all the characters in movie names

df\_data = pd.read\_csv("ml-100k/u.data",sep="\t",names=cols,header=None,encoding="ISO-8859-1")

#import of moviedata

cols = ["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"]

df\_movie = pd.read\_csv("ml-100k/u.item",sep="|",names=cols,header=None,encoding="ISO-8859-1")

#import of user data

cols = ["user id",

"age",

"gender",

"occupation",

"zip code"]

df\_user = pd.read\_csv("ml-100k/u.user",sep="|",names=cols,header=None,encoding="ISO-8859-1")

#frequency binning the ages into age groups as it will be easier for future analysis

df\_user['age\_group'] = pd.qcut(df\_user['age'],q=10,precision=0)

df = pd.merge(pd.merge(df\_data,

df\_user[["user id",

"age\_group",

"gender",

"occupation"]],

on='user id',

how='left'),

df\_movie.drop(['IMDb URL'],axis=1),

left\_on = 'item id',

right\_on = 'movie id',

how ='left')

def release\_year(row):

return str(row['release date'])[-4:]

def release\_month(row):

return str(row['release date'])[3:6]

def release\_date(row):

return str(row['release date'])[:2]

df['release\_year'] = df.apply(lambda row: release\_year(row), axis=1)

df = df.query("release\_year != 'nan'")

df['release\_year'] = df['release\_year'].astype(int)

df['release\_month'] = df.apply(lambda row: release\_month(row), axis=1)

df['release\_day'] = df.apply(lambda row: release\_date(row), axis=1)

df = df.drop(['release date'],axis=1)

df['release\_year'] = pd.cut(df['release\_year'],bins=20)

df.drop(['user id',

'item id',

'timestamp',

'movie id',

'movie title',

'video release date',

'release\_day',

'release\_month'

# 'release\_year'

],

axis=1,

inplace=True)

# df = df.query("release\_month != ''")

# df = df.query("release\_month != 'eb-'")

df = df.query("unknown != '1'")

df\_dummies = pd.get\_dummies(df['age\_group'], prefix = 'age\_group')

df = pd.concat([df, df\_dummies], axis=1)

df\_dummies = pd.get\_dummies(df['gender'], prefix = 'gender')

df = pd.concat([df, df\_dummies], axis=1)

df\_dummies = pd.get\_dummies(df['occupation'], prefix = 'occupation')

df = pd.concat([df, df\_dummies], axis=1)

df\_dummies = pd.get\_dummies(df['release\_year'], prefix = 'release\_year')

df = pd.concat([df, df\_dummies], axis=1)

# df\_dummies = pd.get\_dummies(df['release\_month'], prefix = 'release\_month')

# df = pd.concat([df, df\_dummies], axis=1)

df.drop(['age\_group',

'gender',

'occupation',

'unknown',

'release\_year',

# 'release\_month'

],

axis=1,

inplace=True)

train, test = train\_test\_split(df, test\_size=0.1)

y\_train = train.rating

x\_train = train.drop(['rating'],axis=1)

y\_test = test.rating

x\_test = test.drop(['rating'],axis=1)

lr = LogisticRegression(max\_iter=1000)

lr.fit(x\_train,y\_train)

y\_pred= lr.predict(x\_test)

total = 0

correct = 0

for i in range(len(y\_pred)):

if y\_pred[i]==y\_test.iloc[i]:

correct+=1

total+=1

print('Accuracy: ',correct/total)

rfm = RandomForestClassifier(n\_estimators=50, oob\_score=70, n\_jobs=-1,

random\_state=101,max\_features=None, min\_samples\_leaf=30)

rfm.fit(x\_train,y\_train)

y\_pred=rfm.predict(x\_test)

total = 0

correct = 0

for i in range(len(y\_pred)):

if y\_pred[i]==y\_test.iloc[i]:

correct+=1

total+=1

print('Accuracy: ',correct/total)

dtree = DecisionTreeClassifier(max\_depth=100, random\_state=101,

max\_features=None, min\_samples\_leaf=200)

dtree.fit(x\_train,y\_train)

y\_pred=dtree.predict(x\_test)

total = 0

correct = 0

for i in range(len(y\_pred)):

if y\_pred[i]==y\_test.iloc[i]:

correct+=1

total+=1

print('Accuracy: ',correct/total)

## MODEL 4: MULTIPLE LINEAR REGRESSION:

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

import numpy as np

#import of review data

cols = ["user id"," movie id ","rating","timestamp"]

#encoding using ISO-8859-1 is used because utf-8 does not support all the characters in movie names

df\_data = pd.read\_csv("ml-100k/u.data",sep="\t",names=cols,header=None,encoding="ISO-8859-1")

#import of moviedata

cols = [" 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 "]

df\_movie = pd.read\_csv("ml-100k/u.item",sep="|",names=cols,header=None,encoding="ISO-8859-1")

#import of user data

cols = ["user id","age","gender","occupation","zip code"]

df\_user = pd.read\_csv("ml-100k/u.user",sep="|",names=cols,header=None,encoding="ISO-8859-1")

#frequency binning the ages into age groups as it will be easier for future analysis

df\_user['age\_group'] = pd.qcut(df\_user['age'],q=10,precision=0)

df\_movie=df\_movie.drop([" video release date "],axis=1)

df\_mdata=pd.merge(df\_user,df\_data)

df\_edata=pd.merge(df\_movie,df\_mdata)

df\_edata = df\_edata.drop([" movie title "," release date ","IMDb URL ","age","zip code"],axis=1)

df\_edata = pd.concat([df\_edata,pd.get\_dummies(df\_edata['gender'], prefix='gender')],axis=1)

df\_edata.drop(['gender'],axis=1, inplace=True)

df\_edata = pd.concat([df\_edata,pd.get\_dummies(df\_edata['occupation'], prefix='occupation')],axis=1)

df\_edata.drop(['occupation'],axis=1, inplace=True)

df\_edata = pd.concat([df\_edata,pd.get\_dummies(df\_edata['age\_group'], prefix='age\_group')],axis=1)

df\_edata.drop(['age\_group'],axis=1, inplace=True)

columns = [" unknown "," Action "," Adventure "," Animation ","Children's "," Comedy "," Crime "," Documentary ",

" Drama "," Fantasy ","Film-Noir "," Horror "," Musical "," Mystery "," Romance "," Sci-Fi ", "Thriller ",

" War "," Western ","gender\_M","gender\_F","occupation\_administrator","occupation\_artist","occupation\_doctor",

"occupation\_educator","occupation\_engineer","occupation\_entertainment","occupation\_executive","occupation\_healthcare",

"occupation\_homemaker","occupation\_lawyer","occupation\_librarian","occupation\_marketing","occupation\_none",

"occupation\_other","occupation\_programmer","occupation\_retired","occupation\_salesman","occupation\_scientist",

"occupation\_student","occupation\_technician","occupation\_writer","age\_group\_(6.0, 20.0]","age\_group\_(20.0, 23.0]",

"age\_group\_(23.0, 26.0]","age\_group\_(26.0, 29.0]","age\_group\_(29.0, 31.0]","age\_group\_(31.0, 35.0]","age\_group\_(35.0, 40.0]",

"age\_group\_(40.0, 46.0]","age\_group\_(46.0, 51.0]","age\_group\_(51.0, 73.0]","rating"]

df\_edata = pd.DataFrame(data=df\_edata, columns=columns)

a = df\_edata.groupby('rating').head(3000)

X = df\_edata.drop(['rating'],axis=1)

Y = df\_edata['rating']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size = 0.3, random\_state = 100)

regr = LinearRegression()

regr.fit(x\_train, y\_train)

y\_pred = regr.predict(x\_test)

r\_score = r2\_score(y\_test,y\_pred)

y\_test=y\_test.values.reshape(1,4500)

df = pd.DataFrame({'Actual':y\_test.flatten(),'Predicted':y\_pred.flatten()})

df = df.round(decimals=0)

count = df\_edata['rating'].value\_counts()

total = 0

correct = 0

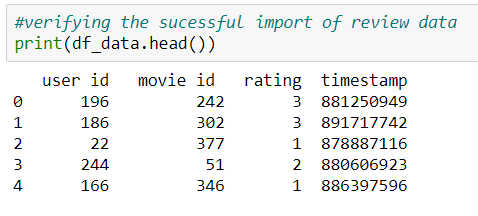
for i in range(len(y\_pred)):

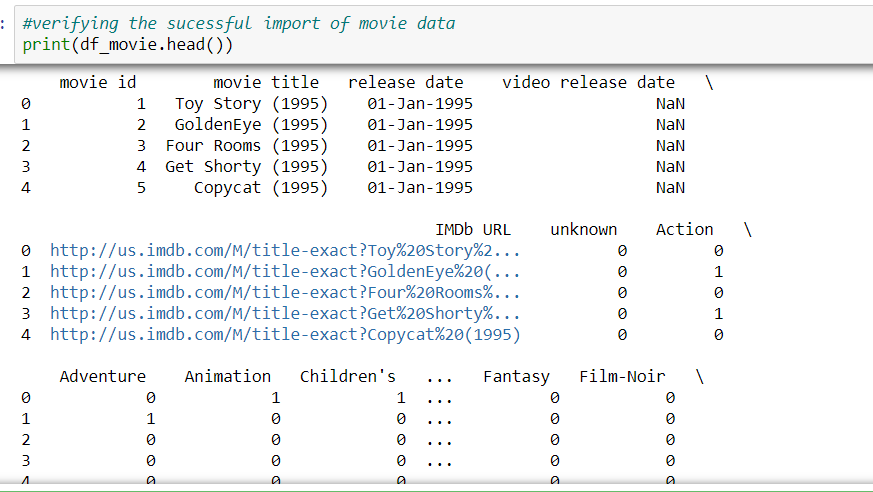
if round(y\_pred[i])==y\_test[0][i]:

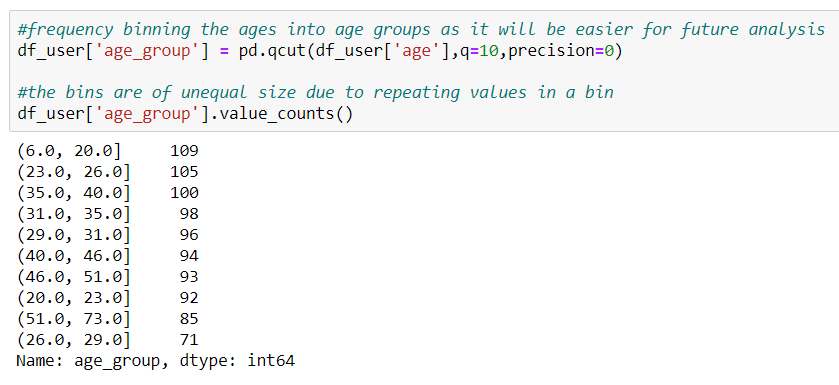
correct+=1

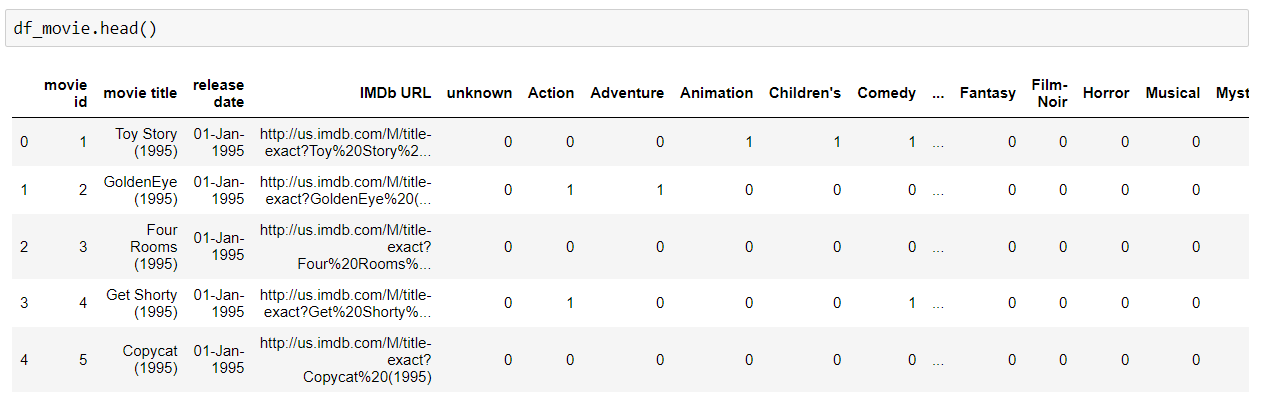
total+=1

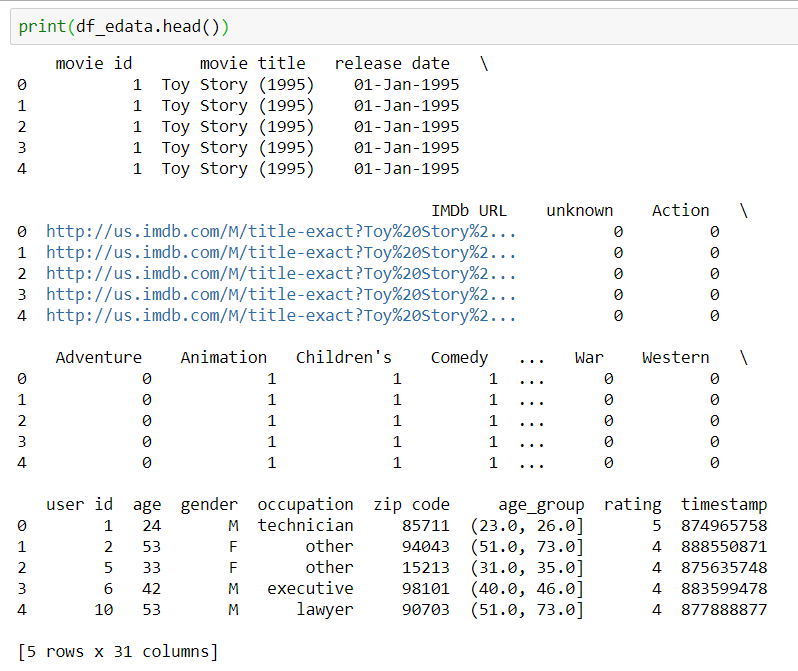
print('Accuracy: ',correct/total)

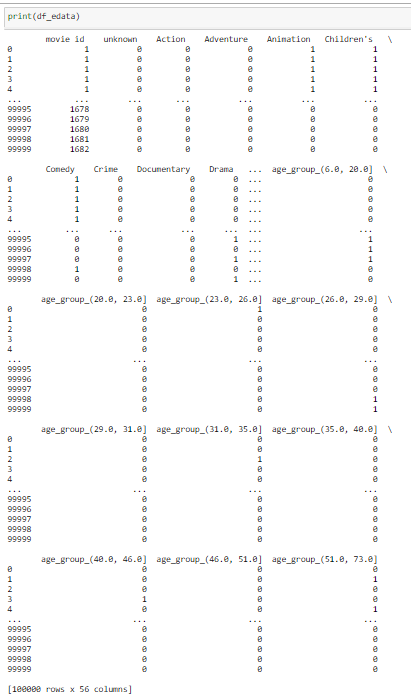


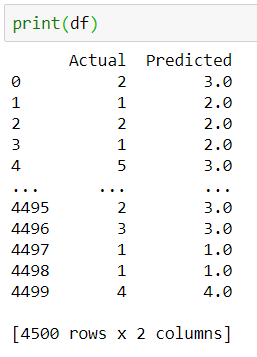


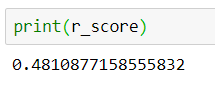


****

****

****

****

****

## MODEL 5: NEURAL NETWORKS

# # Get test data ready

import pandas as pd

import torch

import torchvision as tv

from torchvision import datasets, transforms

from torch.utils.data import DataLoader as DL

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

from sklearn.utils import resample

#import of moviedata

cols = ["movie id","movie title","release date","video release date","IMDb URL","unknown",

"Action","Adventure","Animation","Childrens","Comedy","Crime","Documentary",

"Drama","Fantasy","Film-Noir","Horror","Musical","Mystery","Romance","Sci-Fi",

"Thriller","War","Western"]

df\_movie = pd.read\_csv("ml-100k/u.item",sep="|",names=cols,header=None,encoding="ISO-8859-1")

df\_movie.head()

#import of user data

cols = ["user id","age","gender","occupation","zip code"]

df\_user = pd.read\_csv("ml-100k/u.user",sep="|",names=cols,header=None,encoding="ISO-8859-1")

df\_user.head()

#frequency binning the ages into age groups as it will be easier for future analysis

df\_user['age\_group'] = pd.qcut(df\_user['age'],q=10,precision=0)

#the bins are of unequal size due to repeating values in a bin

df\_user['age\_group'].value\_counts()

# # Check with test sets

#import of review data

cols = ["user id","item id","rating","timestamp"]

#encoding using ISO-8859-1 is used because utf-8 does not support all the characters in movie names

u\_train = pd.read\_csv("ml-100k/u5.base",sep="\t",names=cols,header=None,encoding="ISO-8859-1")

u\_test = pd.read\_csv("ml-100k/u5.test",sep="\t",names=cols,header=None,encoding="ISO-8859-1")

##keep changing above files to u2.base, u2.test, etc.

u\_train.head()

u\_test.head()

#join all three dataframes

training = pd.merge(pd.merge(u\_train,

df\_user[["user id",

"age\_group",

"gender",

"occupation"]],

on='user id',

how='left'),

df\_movie,

left\_on = 'item id',

right\_on = 'movie id',

how ='left')

testing = pd.merge(pd.merge(u\_test,

df\_user[["user id",

"age\_group",

"gender",

"occupation"]],

on='user id',

how='left'),

df\_movie,

left\_on = 'item id',

right\_on = 'movie id',

how ='left')

#categorize age\_group, gender and occupation using 1-hot encoder

training['age\_group'] = pd.Categorical(training['age\_group'])

training['gender'] = pd.Categorical(training['gender'])

training['occupation'] = pd.Categorical(training['occupation'])

age\_group\_dummies = pd.get\_dummies(training['age\_group'])

gender\_dummies = pd.get\_dummies(training['gender'])

occupation\_dummies = pd.get\_dummies(training['occupation'])

training = pd.concat([training,

age\_group\_dummies,

gender\_dummies,

occupation\_dummies], axis=1)

training.drop(['age\_group',

'gender',

'occupation'], axis=1, inplace=True)

#verify categorization

training.head()

#categorize age\_group, gender and occupation using 1-hot encoder

testing['age\_group'] = pd.Categorical(testing['age\_group'])

testing['gender'] = pd.Categorical(testing['gender'])

testing['occupation'] = pd.Categorical(testing['occupation'])

age\_group\_dummies = pd.get\_dummies(testing['age\_group'])

gender\_dummies = pd.get\_dummies(testing['gender'])

occupation\_dummies = pd.get\_dummies(testing['occupation'])

testing = pd.concat([testing,

age\_group\_dummies,

gender\_dummies,

occupation\_dummies], axis=1)

testing.drop(['age\_group',

'gender',

'occupation'], axis=1, inplace=True)

#verify categorization

testing.head()

#drop unneccessary features

training.drop(["movie id",

"movie title",

"release date",

"video release date",

"IMDb URL",

"unknown",

"user id",

"item id",

"timestamp"],axis=1, inplace=True)

testing.drop(["movie id",

"movie title",

"release date",

"video release date",

"IMDb URL",

"unknown",

"user id",

"item id",

"timestamp"],axis=1, inplace=True)

print(training.rating.value\_counts())

print(testing.rating.value\_counts())

#balance the training ratings using upsampling

ns = 25000

training\_1 = training[training.rating == 1]

training\_2 = training[training.rating == 2]

training\_3 = training[training.rating == 3]

training\_4 = training[training.rating == 4]

training\_5 = training[training.rating == 5]

training\_1\_upsampled = resample(training\_1,

replace = True,

n\_samples = ns,

random\_state=123)

training\_2\_upsampled = resample(training\_2,

replace = True,

n\_samples = ns,

random\_state=123)

training\_3\_upsampled = resample(training\_3,

replace = True,

n\_samples = ns,

random\_state=123)

training\_4\_upsampled = resample(training\_4,

replace = True,

n\_samples = ns,

random\_state=123)

training\_5\_upsampled = resample(training\_5,

replace = True,

n\_samples = ns,

random\_state=123)

training = pd.concat([training\_1\_upsampled,

training\_2\_upsampled,

training\_3\_upsampled,

training\_4\_upsampled,

training\_5\_upsampled])

training.rating.value\_counts()

#balance the testing ratings using upsampling

ns = 6500

testing\_1 = testing[testing.rating == 1]

testing\_2 = testing[testing.rating == 2]

testing\_3 = testing[testing.rating == 3]

testing\_4 = testing[testing.rating == 4]

testing\_5 = testing[testing.rating == 5]

testing\_1\_upsampled = resample(testing\_1,

replace = True,

n\_samples = ns,

random\_state=123)

testing\_2\_upsampled = resample(testing\_2,

replace = True,

n\_samples = ns,

random\_state=123)

testing\_3\_upsampled = resample(testing\_3,

replace = True,

n\_samples = ns,

random\_state=123)

testing\_4\_upsampled = resample(testing\_4,

replace = True,

n\_samples = ns,

random\_state=123)

testing\_5\_upsampled = resample(testing\_5,

replace = True,

n\_samples = ns,

random\_state=123)

testing = pd.concat([testing\_1\_upsampled,

testing\_2\_upsampled,

testing\_3\_upsampled,

testing\_4\_upsampled,

testing\_5\_upsampled])

testing.rating.value\_counts()

#resetting the index

training.reset\_index(inplace = True, drop = True)

testing.reset\_index(inplace = True, drop = True)

print(training.shape)

print(testing.shape)

#prepare data for PyTorch

n\_input = training.shape[1] - 1

rank\_train = training['rating'].values

training\_input = training.drop(["rating"], axis=1)

train = []

for index,row in training\_input.iterrows():

t = (torch.tensor(row.values), rank\_train[index])

train.append(t)

train = tuple(train)

rank\_test = testing['rating'].values

testing\_input = testing.drop(["rating"], axis=1)

test = []

for index,row in testing\_input.iterrows():

t = (torch.tensor(row.values), rank\_test[index])

test.append(t)

test = tuple(test)

# # Neural Network

#create class for the neural network

'''

fully connected layer = fc

nn.Linear(input, ouput)

initial input = number of columns = 51

middle layers = 3 layers of 64 neurons

final output = number of ratings (0-5) = 6

'''

n\_hidden\_neurons = int((2\*n\_input/3)+6)

class Net(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.fc1 = nn.Linear(n\_input, n\_hidden\_neurons)

self.fc2 = nn.Linear(n\_hidden\_neurons, n\_hidden\_neurons)

self.fc3 = nn.Linear(n\_hidden\_neurons, n\_hidden\_neurons)

self.fc4 = nn.Linear(n\_hidden\_neurons, 6)

#ReLU activation function on hidden layers

#Use log\_softmax for output to get probability for classes

def forward(self, x):

x = torch.tanh(self.fc1(x))

x = torch.tanh(self.fc2(x))

x = torch.tanh(self.fc3(x))

x = self.fc4(x)

return F.log\_softmax(x, dim=1)

#view created network

net = Net()

net

#divide into batches

'''

batch\_size = how many inputs to pass to model at a time

shuffle = to shuffle inputs or not

'''

trainset = DL(train, batch\_size=64, shuffle=True)

testset = DL(test, batch\_size=64, shuffle=True)

print(trainset)

#lr = learning rate = 0.001

opt = optim.Adam(net.parameters(), lr = 0.001)

#EPOCHS = number of times to iterate over dataset

EPOCHS = 10

#train the network

'''

loss = error

zero\_grad() = makes gradient zero after batch

nll\_loss = calculates loss to update weights

if data is 1 hot vector, use mean squared error

backward() = propogate the weights backward

opt.step() = adjusts the weights

'''

for epoch in range(EPOCHS):

for data in trainset:

X, y = data

net.zero\_grad()

output = net(X.view(-1, n\_input).float())

loss = F.nll\_loss(output, y)

loss.backward()

opt.step()

print(loss)

#check the model

'''

no\_grad() = as test data will not be used for optimization,

we do not need to calculate gradient for it

'''

correct = 0

total = 0

with torch.no\_grad():

for data in testset:

X, y = data

output = net(X.view(-1, n\_input).float())

for idx, i in enumerate(output):

if torch.argmax(i) == y[idx]:

correct += 1

total += 1

print("Accuracy: ",round(correct/total, 4))

