MovieLensProject

June 1, 2022

[1]:
```
# Load libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
```

[2]:
```
movie_data = pd.read_csv("movies.dat",
                         sep="::", header=None,␣
 ↪names=['MovieID','Title','Genres'],
                         dtype={'MovieID': np.int32, 'Title': np.str_, 'Genres':
 ↪ np.str_}, engine='python')
users_data = pd.read_csv("users.dat",
                       sep="::", header=None,␣
 ↪names=['UserID','Gender','Age','Occupation','Zip-code'],
    dtype={'UserID': np.int32, 'Gender': np.str_, 'Age': np.int32, 'Occupation'␣
 ↪: np.int32, 'Zip-code' : np.str_}, engine='python')
ratings_data = pd.read_csv("ratings.dat",
                       sep="::", header=None,␣
 ↪names=['UserID','MovieID','Rating','Timestamp'],
                dtype={'UserID': np.int32, 'MovieID': np.int32, 'Rating': np.
 ↪int32, 'Timestamp' : np.str_}, engine='python')
```

[3]:
```
movie_data.head()
```

[3]:

	MovieID	Title	Genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama

```
4        5  Father of the Bride Part II (1995)                      Comedy
```

[4]:
```
movie_data.shape
```

[4]:
```
(3883, 3)
```

[5]:
```
movie_data.isnull().sum()
# Results show that no columns are empty or null
```

[5]:
```
MovieID    0
Title      0
Genres     0
dtype: int64
```

[6]:
```
movie_data.describe()
```

[6]:
```
           MovieID
count  3883.000000
mean   1986.049446
std    1146.778349
min       1.000000
25%     982.500000
50%    2010.000000
75%    2980.500000
max    3952.000000
```

[7]:
```
movie_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3883 entries, 0 to 3882
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   MovieID  3883 non-null   int32
 1   Title    3883 non-null   object
 2   Genres   3883 non-null   object
dtypes: int32(1), object(2)
memory usage: 76.0+ KB
```

[8]:
```
# On users data
users_data.shape
```

[8]:
```
(6040, 5)
```

[9]:
```
users_data.head()
```

[9]:

	UserID	Gender	Age	Occupation	Zip-code
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117
3	4	M	45	7	02460
4	5	M	25	20	55455

[10]:
```
users_data.describe()
```

[10]:

	UserID	Age	Occupation
count	6040.000000	6040.000000	6040.000000
mean	3020.500000	30.639238	8.146854
std	1743.742145	12.895962	6.329511
min	1.000000	1.000000	0.000000
25%	1510.750000	25.000000	3.000000
50%	3020.500000	25.000000	7.000000
75%	4530.250000	35.000000	14.000000
max	6040.000000	56.000000	20.000000

[11]:
```
users_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   UserID      6040 non-null   int32 
 1   Gender      6040 non-null   object
 2   Age         6040 non-null   int32 
 3   Occupation  6040 non-null   int32 
 4   Zip-code    6040 non-null   object
dtypes: int32(3), object(2)
memory usage: 165.3+ KB
```

[12]:
```
users_data.isnull().sum()
# Results show that no columns are empty or null
```

[12]:
```
UserID        0
Gender        0
Age           0
Occupation    0
Zip-code      0
dtype: int64
```

[13]:
```
# On Ratings data
ratings_data.head()
```

[13]:

	UserID	MovieID	Rating	Timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291

[14]:
```
ratings_data.shape
```

[14]:
```
(1000209, 4)
```

[15]:
```
ratings_data.describe()
```

[15]:

	UserID	MovieID	Rating
count	1.000209e+06	1.000209e+06	1.000209e+06
mean	3.024512e+03	1.865540e+03	3.581564e+00
std	1.728413e+03	1.096041e+03	1.117102e+00
min	1.000000e+00	1.000000e+00	1.000000e+00
25%	1.506000e+03	1.030000e+03	3.000000e+00
50%	3.070000e+03	1.835000e+03	4.000000e+00
75%	4.476000e+03	2.770000e+03	4.000000e+00
max	6.040000e+03	3.952000e+03	5.000000e+00

[16]:
```
ratings_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 4 columns):
 #   Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0   UserID     1000209 non-null  int32
 1   MovieID    1000209 non-null  int32
 2   Rating     1000209 non-null  int32
 3   Timestamp  1000209 non-null  object
dtypes: int32(3), object(1)
memory usage: 19.1+ MB
```

[17]:
```
ratings_data.isnull().sum()
# Results show that no columns are empty or null
```

[17]:
```
UserID       0
MovieID      0
Rating       0
Timestamp    0
dtype: int64
```

[18]:
```
#merge ratings and user data
df_user_rating = users_data.merge(ratings_data, how='left', left_on=['UserID'],␣
↪right_on=['UserID'])
```

[19]:
```
#merge with movie data to create Master data
Master_Data = df_user_rating.merge(movie_data, how='left', left_on=['MovieID'],␣
↪right_on=['MovieID'])
```

[20]:
```
Master_Data.head()
```

[20]:

	UserID	Gender	Age	Occupation	Zip-code	MovieID	Rating	Timestamp	Title	Genres
0	1	F	1	10	48067	1193	5	978300760	One Flew Over the Cuckoo's Nest (1975)	Drama
1	1	F	1	10	48067	661	3	978302109	James and the Giant Peach (1996)	Animation|Children's|Musical
2	1	F	1	10	48067	914	3	978301968	My Fair Lady (1964)	Musical|Romance
3	1	F	1	10	48067	3408	4	978300275	Erin Brockovich (2000)	Drama
4	1	F	1	10	48067	2355	5	978824291	Bug's Life, A (1998)	Animation|Children's|Comedy

[21]:
```
Master_Data.columns
```

[21]:
```
Index(['UserID', 'Gender', 'Age', 'Occupation', 'Zip-code', 'MovieID',
       'Rating', 'Timestamp', 'Title', 'Genres'],
      dtype='object')
```

[22]:
```
#create a dataframe with required column
col = ['MovieID','Title', 'UserID', 'Age','Gender', 'Occupation', 'Rating']
Master_Data = Master_Data[col]
```

[23]:
```
Master_Data.head()
```

[23]:

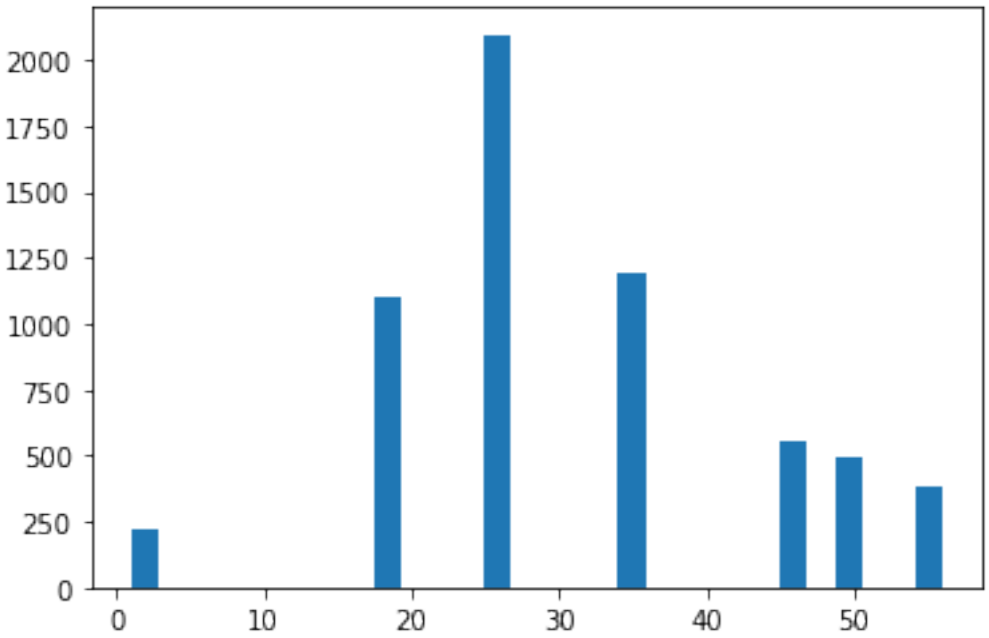
	MovieID	Title	UserID	Age	Gender	Occupation	Rating
0	1193	One Flew Over the Cuckoo's Nest (1975)	1	1	F	10	5
1	661	James and the Giant Peach (1996)	1	1	F	10	3
2	914	My Fair Lady (1964)	1	1	F	10	3
3	3408	Erin Brockovich (2000)	1	1	F		
4	2355	Bug's Life, A (1998)	1	1	F		

```
3           10        4
4           10        5
```

[24]:
```
#User Age Distribution
age_group = users_data.groupby('Age').size()
age_group
```

[24]:
```
Age
1       222
18     1103
25     2096
35     1193
45      550
50      496
56      380
dtype: int64
```

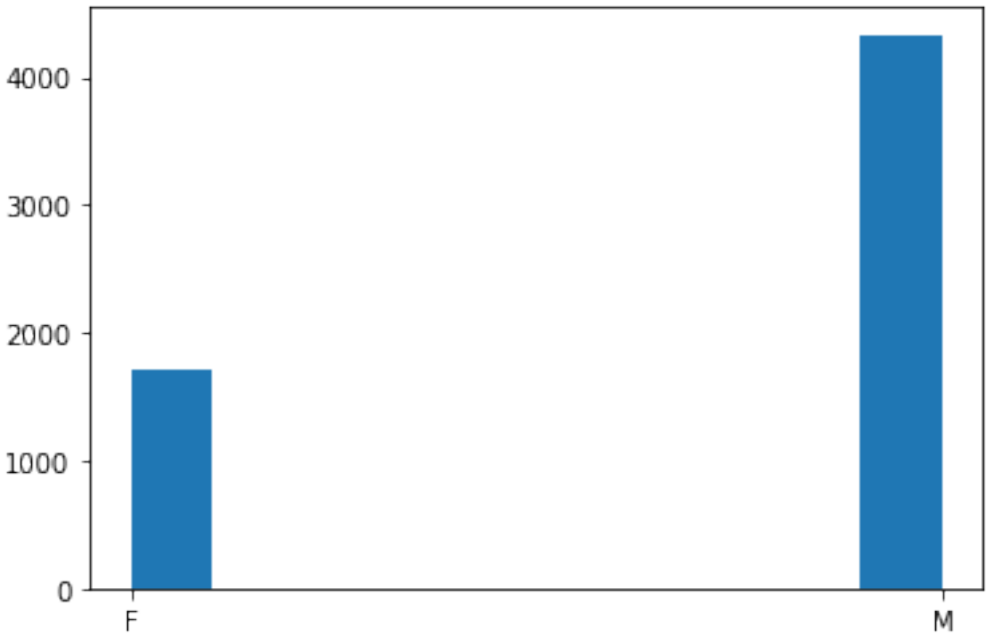
[25]:
```
plt.hist(data=age_group,x=[users_data.Age], bins=30)
plt.show()
```



[26]:
```
# Gender Distribution
gender_group = users_data.groupby('Gender').size()
gender_group
```

[26]:
```
Gender
F    1709
M    4331
dtype: int64
```

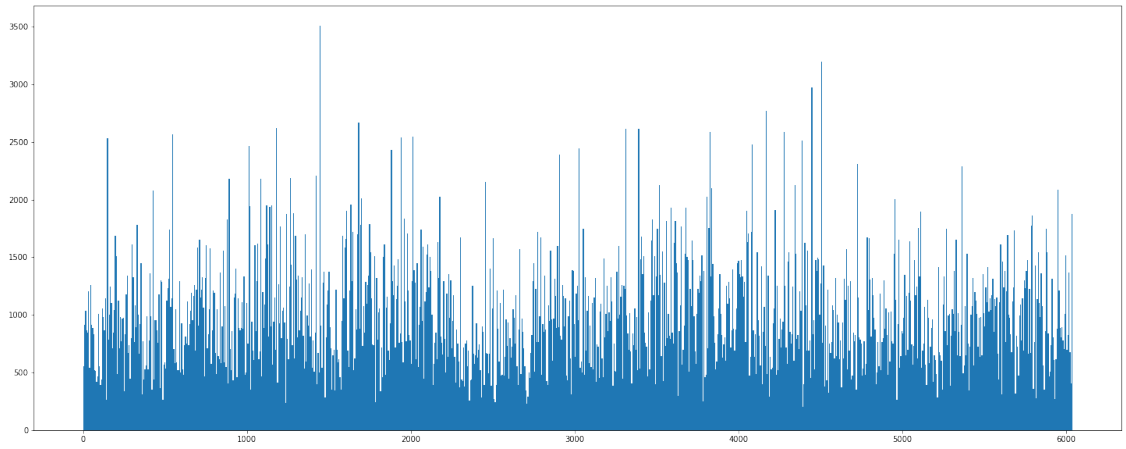
[27]:
```
plt.hist(data=gender_group,x=[users_data.Gender])
plt.show()
```



[28]:
```
#User ratings
user_group = ratings_data.groupby(['UserID']).size()
user_group.head(10)
```

[28]:
```
UserID
1      53
2     129
3      51
4      21
5     198
6      71
7      31
8     139
9     106
10    401
dtype: int64
```

[29]:
```
plt.figure(figsize=(25,10))
plt.hist(x=[ratings_data.UserID], bins=1000)
plt.show()
```



[30]:
```
# ToyStory Data
toystory_data = ratings_data[ratings_data.MovieID==1]
toystory_data.head(10)
```

[30]:

	UserID	MovieID	Rating	Timestamp
40	1	1	5	978824268
469	6	1	4	978237008
581	8	1	4	978233496
711	9	1	5	978225952
837	10	1	5	978226474
1966	18	1	4	978154768
2276	19	1	5	978555994
2530	21	1	3	978139347
2870	23	1	4	978463614
3405	26	1	3	978130703

[31]:
```
toystory_data.groupby('Rating').size()
```

[31]:
```
Rating
1     16
2     61
3    345
4    835
5    820
dtype: int64
```

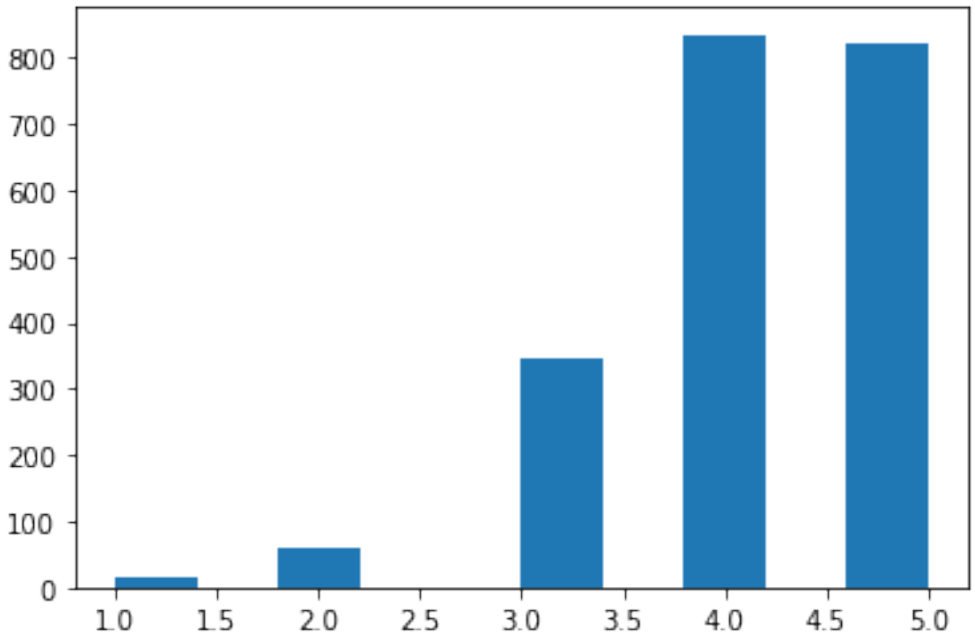
Toy story is mostly rated in range 4-5

[32]:
```
toystory_data_group = toystory_data.groupby('Rating')
toystory_data_group.agg({'Rating':'mean'})
```

[32]:

Rating	Rating
1	1
2	2
3	3
4	4
5	5

[33]:
```
plt.hist(x=toystory_data['Rating'])
plt.show()
```



[34]:
```
# Viewership by Age for Toystory
viewership = pd.merge(ratings_data, users_data, how='left', left_on=['UserID'],␣
↪right_on=['UserID'])
```

[35]:
```
viewership.shape
```

[35]: (1000209, 8)

[36]:
```
ratings_data.shape
```

[36]: (1000209, 4)

[37]:
```
viewership.head()
```

[37]:

	UserID	MovieID	Rating	Timestamp	Gender	Age	Occupation	Zip-code
0	1	1193	5	978300760	F	1	10	48067
1	1	661	3	978302109	F	1	10	48067
2	1	914	3	978301968	F	1	10	48067
3	1	3408	4	978300275	F	1	10	48067
4	1	2355	5	978824291	F	1	10	48067

[38]:
```
#select only 'Toystory' data
viewership_of_toystory = viewership[viewership['MovieID'] == 1]
viewership_of_toystory.shape
```

[38]:
```
(2077, 8)
```

[39]:
```
viewership_of_toystory.head()
```

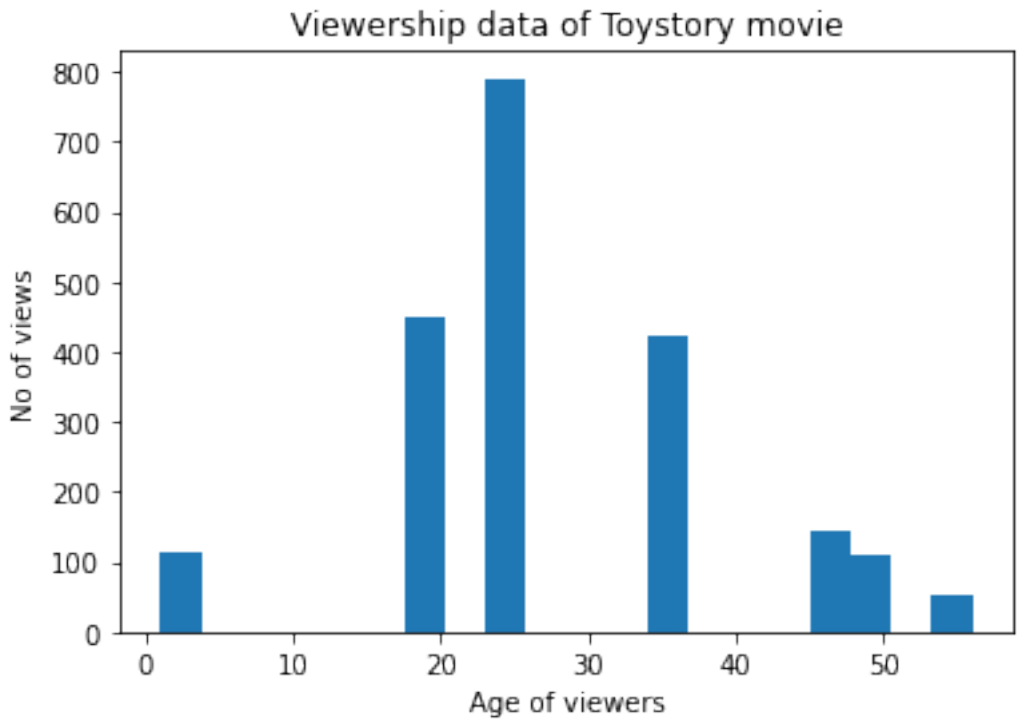
[39]:

	UserID	MovieID	Rating	Timestamp	Gender	Age	Occupation	Zip-code
40	1	1	5	978824268	F	1	10	48067
469	6	1	4	978237008	F	50	9	55117
581	8	1	4	978233496	M	25	12	11413
711	9	1	5	978225952	M	25	17	61614
837	10	1	5	978226474	F	35	1	95370

[40]:
```
viewership_of_toystory.groupby('Age').size()
```

[40]:
```
Age
1     112
18    448
25    790
35    423
45    143
50    108
56     53
dtype: int64
```

[41]:
```
plt.hist(x=viewership_of_toystory['Age'], bins=20)
plt.xlabel("Age of viewers")
plt.ylabel("No of views")
plt.title("Viewership data of Toystory movie")
plt.show()
```



0.0.1 The above plot shows that the Toystory movie is more popular for viewers between Age group 20-25 years

Top 25 movies by viewership rating

[42]:
```
movie_rating = ratings_data.groupby(['MovieID'], as_index=False)
average_movie_ratings = movie_rating.agg({'Rating':'mean'})
top_25_movies = average_movie_ratings.sort_values('Rating', ascending=False).
↪head(25)
top_25_movies
```

[42]:

	MovieID	Rating
926	989	5.000000
3635	3881	5.000000
1652	1830	5.000000
3152	3382	5.000000
744	787	5.000000
3054	3280	5.000000
3367	3607	5.000000
3010	3233	5.000000
2955	3172	5.000000
3414	3656	5.000000
3021	3245	4.800000

```
51          53  4.750000
2309      2503  4.666667
2698      2905  4.608696
1839      2019  4.560510
309        318  4.554558
802        858  4.524966
708        745  4.520548
49          50  4.517106
513        527  4.510417
1066      1148  4.507937
2117      2309  4.500000
1626      1795  4.500000
2287      2480  4.500000
425        439  4.500000
```

[43]:
```
#The below list shows top 25 movies by viewership data
pd.merge(top_25_movies, movie_data, how='left', left_on=['MovieID'],␣
↪right_on=['MovieID'])
```

[43]:
```
    MovieID    Rating                                                Title  \
0       989  5.000000                Schlafes Bruder (Brother of Sleep) (1995)
1      3881  5.000000                                Bittersweet Motel (2000)
2      1830  5.000000                                 Follow the Bitch (1998)
3      3382  5.000000                                  Song of Freedom (1936)
4       787  5.000000                       Gate of Heavenly Peace, The (1995)
5      3280  5.000000                                           Baby, The (1973)
6      3607  5.000000                                One Little Indian (1973)
7      3233  5.000000                                    Smashing Time (1967)
8      3172  5.000000                                 Ulysses (Ulisse) (1954)
9      3656  5.000000                                           Lured (1947)
10     3245  4.800000                       I Am Cuba (Soy Cuba/Ya Kuba) (1964)
11       53  4.750000                                         Lamerica (1994)
12     2503  4.666667                                 Apple, The (Sib) (1998)
13     2905  4.608696                                       Sanjuro (1962)
14     2019  4.560510  Seven Samurai (The Magnificent Seven) (Shichin…
15      318  4.554558                          Shawshank Redemption, The (1994)
16      858  4.524966                                    Godfather, The (1972)
17      745  4.520548                                    Close Shave, A (1995)
18       50  4.517106                               Usual Suspects, The (1995)
19      527  4.510417                                  Schindler's List (1993)
20     1148  4.507937                               Wrong Trousers, The (1993)
21     2309  4.500000               Inheritors, The (Die Siebtelbauern) (1998)
22     1795  4.500000                       Callej n de los milagros, El (1995)
23     2480  4.500000                    Dry Cleaning (Nettoyage  sec) (1997)
24      439  4.500000                                 Dangerous Game (1993)

                        Genres
```

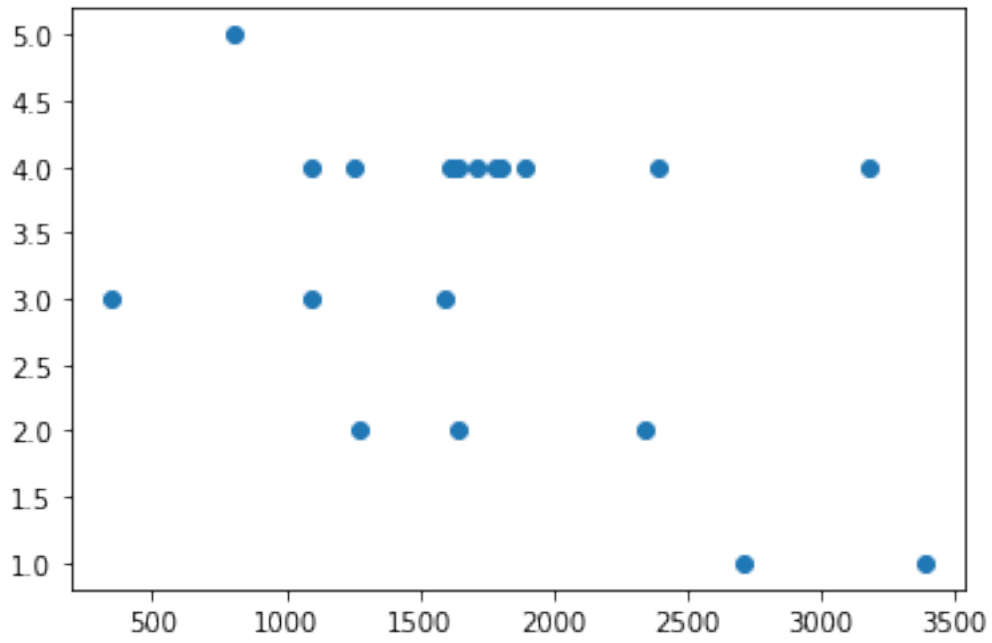
```
0                       Drama
1                 Documentary
2                      Comedy
3                       Drama
4                 Documentary
5                      Horror
6       Comedy|Drama|Western
7                      Comedy
8                   Adventure
9                       Crime
10                      Drama
11                      Drama
12                      Drama
13          Action|Adventure
14              Action|Drama
15                      Drama
16        Action|Crime|Drama
17 Animation|Comedy|Thriller
18            Crime|Thriller
19                 Drama|War
20          Animation|Comedy
21                      Drama
22                      Drama
23                      Drama
24                      Drama
```

[44]:
```
#Rating of userid = 2696
user_rating_data = ratings_data[ratings_data['UserID']==2696]
user_rating_data.head()
```

[44]:

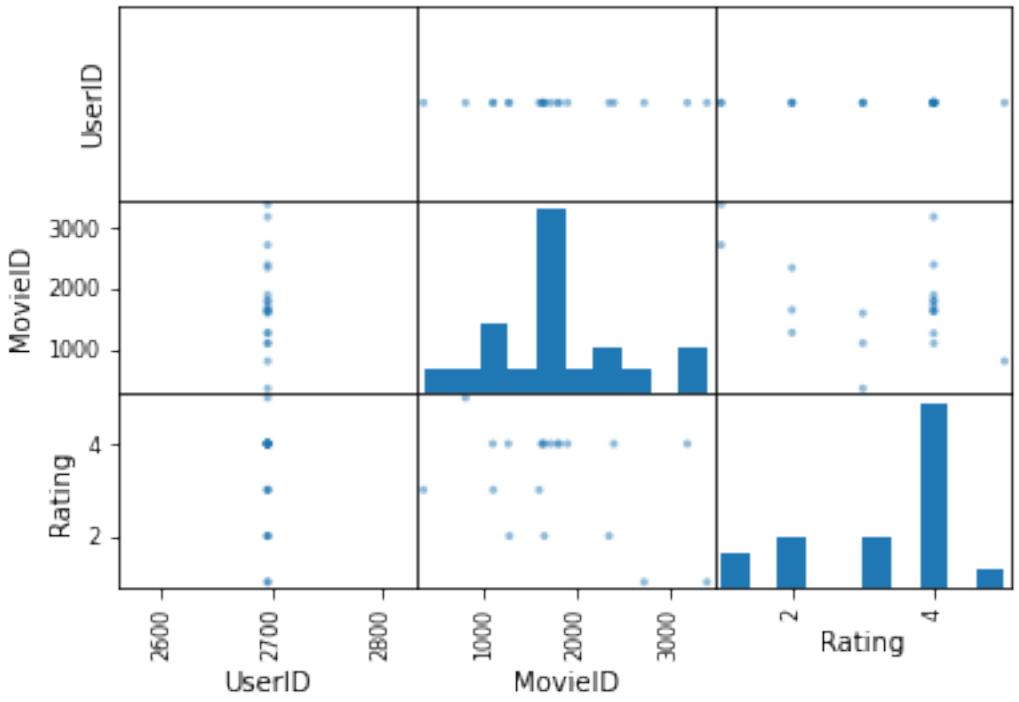
	UserID	MovieID	Rating	Timestamp
440667	2696	1258	4	973308710
440668	2696	1270	2	973308676
440669	2696	1617	4	973308842
440670	2696	1625	4	973308842
440671	2696	1644	2	973308920

[45]:
```
# plotting the above data
plt.scatter(x=user_rating_data['MovieID'], y=user_rating_data['Rating'])
plt.show()
```



[46]:
```
from pandas.plotting import scatter_matrix
scatter_matrix(user_rating_data)
plt.show()
```

```
/usr/local/lib/python3.7/site-packages/pandas/plotting/_matplotlib/misc.py:71:
UserWarning: Attempting to set identical left == right == 2696.0 results in
singular transformations; automatically expanding.
  ax.set_xlim(boundaries_list[i])
/usr/local/lib/python3.7/site-packages/pandas/plotting/_matplotlib/misc.py:81:
UserWarning: Attempting to set identical bottom == top == 2696.0 results in
singular transformations; automatically expanding.
  ax.set_ylim(boundaries_list[i])
/usr/local/lib/python3.7/site-packages/pandas/plotting/_matplotlib/misc.py:80:
UserWarning: Attempting to set identical left == right == 2696.0 results in
singular transformations; automatically expanding.
  ax.set_xlim(boundaries_list[j])
```



0.0.2 Split Genre

[47]:
```
#Split genre to get value of genre
list_genre = movie_data.Genres.str.split('|').values
list_genre
```

[47]:
```
array([list(['Animation', "Children's", 'Comedy']),
       list(['Adventure', "Children's", 'Fantasy']),
       list(['Comedy', 'Romance']), …, list(['Drama']), list(['Drama']),
       list(['Drama', 'Thriller'])], dtype=object)
```

[48]:
```
# Unique genre keys
genre_labels = set()
for s in movie_data.Genres.str.split('|').values:
    genre_labels = genre_labels.union(set(s))
genre_labels
```

[48]:
```
{'Action',
 'Adventure',
 'Animation',
 "Children's",
 'Comedy',
```

```
 'Crime',
 'Documentary',
 'Drama',
 'Fantasy',
 'Film-Noir',
 'Horror',
 'Musical',
 'Mystery',
 'Romance',
 'Sci-Fi',
 'Thriller',
 'War',
 'Western'}
```

0.0.3 One hot encoding

[51]:
```
# One hot encoding
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder

# define data for one hot encoding
genre_labels = list(genre_labels)
values = array(genre_labels)
print(values)
```

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

[53]:
```
# integer encode
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(values)
print(integer_encoded)
```

```
[ 5 11 17  3  8  6  4 13 16 14  1  2 10  0  9 12  7 15]
```

[54]:
```
# binary encode
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
print(onehot_encoded)
```

```
[[0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
```

0.0.4 Features affecting the rating and development of model

[55]:
```
#prepare data
few_viewership = viewership.head(500)
few_viewership.shape
```

[55]:
```
(500, 8)
```

[56]:
```
few_viewership.head()
```

[56]:

	UserID	MovieID	Rating	Timestamp	Gender	Age	Occupation	Zip-code
0	1	1193	5	978300760	F	1	10	48067
1	1	661	3	978302109	F	1	10	48067
2	1	914	3	978301968	F	1	10	48067
3	1	3408	4	978300275	F	1	10	48067
4	1	2355	5	978824291	F	1	10	48067

[57]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

le.fit(few_viewership['Age'])
x_age = le.transform(few_viewership['Age'])
x_age
```

[57]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
```

```
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3])
```

[58]:
```
le.fit(few_viewership['Occupation'])
x_occ = le.transform(few_viewership['Occupation'])
x_occ
```

[58]:
```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

[59]:
```
le.fit(few_viewership['MovieID'])
x_movieid = le.transform(few_viewership['MovieID'])
x_movieid
```

[59]:
```
array([130,  78,  95, 374, 280, 132, 156, 321,  71,  96,  72,  98, 287,
       330, 107, 318, 304, 251, 355, 319, 274,  80, 154,  61, 278,  12,
       119, 211, 186,  84, 271, 364, 189,  67, 231,  86, 226, 103, 316,
        18,   0, 243, 244, 305,  29, 104, 105, 135, 252,  62, 359,  74,
       145, 161, 346, 184,  75, 264,  76, 266, 302, 121, 329, 379, 136,
       222, 205, 137, 392, 326, 342, 139, 355,  49, 260, 356, 357, 343,
       148, 194,  33, 265, 347,  92,  44, 149, 360, 185, 158, 127, 366,
       367, 368,  17, 267, 293, 225, 380,  68, 207, 398, 323, 237, 100,
       227, 324, 140, 252,  60,  50, 272,  30, 170, 113, 403,  54, 173,
       255, 151, 162, 130, 224, 163, 279, 372, 289,  69, 131, 187,  83,
       133,  70, 281,  15, 308, 297, 234, 286, 407, 239, 193, 413, 240,
       241,  28, 122, 242,  20,   3,  21, 274, 115,  46, 294,  39,  51,
       118,  97,  52, 181, 376, 166, 378, 353,  85,  56, 312, 247, 244,
       220, 331, 248,  36, 135, 246, 400, 143,  41, 144, 145, 415, 146,
       377, 198,  76, 169, 389,  16, 314, 136, 172, 414, 112, 338, 195,
       157, 149,  77, 262, 191, 396,  29, 324, 359, 111, 150,  64,  54,
       151, 152,  82, 131,  69, 280, 132, 133, 164,  70, 165, 391, 160,
       154, 292, 362, 301, 243, 399, 248, 325, 259, 246, 124, 257, 379,
       136, 333, 138, 108,  29, 252,  54, 131, 133, 240, 119, 376, 404,
       282, 167, 388, 134, 305, 332, 141, 337, 276, 126,   9,  32, 277,
       183, 168, 266, 175,  89, 203,  90, 204, 329, 317,  25, 219,  57,
       392,  58, 147, 411,  59,  10, 194, 254, 412, 338,  11, 306,  34,
        66, 196,  81,  35, 350, 296, 232,  18,  26, 406,  27,   1, 339,
       324, 110,  60,  87,  13,  14, 128, 129,  82, 351,  45, 279, 153,
       352, 289, 385, 290, 280, 268, 386, 188, 233,  70, 281, 307, 176,
       308, 297, 269,  19, 123, 340, 256, 208, 361, 291, 270, 197, 155,
       309, 310, 235, 311, 236, 298, 373, 408,  99,  91, 341, 221,  36,
        22,  53,  74, 171, 261, 209, 199, 365, 363, 210, 322, 313, 200,
        37, 249, 354, 334, 137, 223, 299, 177, 355, 335, 178, 211, 212,
       201,  93, 191, 202, 283, 213, 381, 327, 358, 252,   2,  30, 253,
        23,  63, 179, 344, 273, 180, 114, 369,  94, 214, 374, 375, 300,
       174, 215, 216, 284, 217, 370, 371,  96, 397, 285, 192, 303,  48,
       315, 101, 228, 229,  31,   4, 345,  38, 275, 116,   5,  65, 117,
       181, 182, 376, 263,  55,  24, 218, 328, 316, 305, 245, 382,  40,
         6, 142, 230, 125, 410,   7,  41,  42,   8,  79, 288, 120, 405,
       106, 206, 107, 392,  43, 383, 348,  44,  34,  12, 349, 127, 384,
        67,   0, 109, 324,  45,  69,  72,  73,  47, 295, 387, 189, 190,
       248, 257, 258, 393, 394, 320, 389, 390, 136, 409, 401, 250, 402,
       395, 159,  88, 102, 238, 336])
```

[60]:
```
few_viewership['New Age'] = x_age
few_viewership['New Occupation'] = x_occ
few_viewership['New MovieID'] = x_movieid
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[61]:
```
# Feature Selection
x_input = few_viewership[['New Age','New Occupation','New MovieID']]
y_target = few_viewership['Rating']
```

[62]:
```
x_input.head()
```

[62]:

	New Age	New Occupation	New MovieID
0	0	2	130
1	0	2	78
2	0	2	95
3	0	2	374
4	0	2	280

[63]:
```
# Split-out validation dataset
x_train, x_test, y_train, y_test = train_test_split(x_input, y_target,␣
 ↪test_size=0.25)
```

[64]:
```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

[64]:
```
((375, 3), (125, 3), (375,), (125,))
```

[65]:
```
from sklearn.linear_model import LogisticRegression
logitReg = LogisticRegression()
lm = logitReg.fit(x_train, y_train)
```

```
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

[66]:
```
result = logitReg.predict(x_test)
```

[67]:
```
estimated = pd.Series(result, name='Estimated Values')
```

[68]:
```
final_result = pd.concat([y_test, estimated], axis=1)
```

[71]:
```
# Test options and evaluation metric
print (accuracy_score(y_test, result))
print (confusion_matrix(y_test, result))
print (classification_report(y_test, result))
```

```
0.36
[[ 0  0  2  6  0]
 [ 0  0  4  7  0]
 [ 0  0 14 18  3]
 [ 0  0 18 28  6]
 [ 0  0  8  8  3]]
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         8
           2       0.00      0.00      0.00        11
           3       0.30      0.40      0.35        35
           4       0.42      0.54      0.47        52
           5       0.25      0.16      0.19        19

    accuracy                           0.36       125
   macro avg       0.19      0.22      0.20       125
weighted avg       0.30      0.36      0.32       125


/usr/local/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1248:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
```

```
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1248:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1248:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Accuracy of the above matrix is 36 %

[72]:
```
#import KNN from sklearn
from sklearn.neighbors import KNeighborsClassifier
```

[73]:
```
#instantiate KNN estimator
knn = KNeighborsClassifier(n_neighbors=1)
```

[74]:
```
print(knn)
```

```
KNeighborsClassifier(n_neighbors=1)
```

[75]:
```
# fit data in KNN estimator with movie and userid as feature
knn.fit(x_train,y_train)
```

[75]:
```
KNeighborsClassifier(n_neighbors=1)
```

[76]:
```
# Testing
predicted = knn.predict(x_test)
predicted
```

[76]:
```
array([3, 4, 3, 3, 2, 3, 3, 5, 4, 3, 4, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4,
       3, 2, 4, 1, 3, 3, 2, 5, 3, 4, 4, 1, 4, 4, 3, 4, 5, 3, 4, 3, 1, 2,
       3, 3, 3, 4, 3, 1, 2, 2, 2, 4, 4, 3, 5, 1, 4, 5, 5, 3, 5, 3, 5, 4,
       4, 1, 3, 4, 3, 5, 5, 5, 4, 1, 2, 3, 4, 3, 2, 4, 1, 5, 2, 2, 5, 2,
       5, 1, 4, 5, 5, 1, 4, 3, 3, 4, 3, 4, 4, 3, 5, 4, 5, 5, 5, 3, 3, 5,
       5, 4, 2, 5, 5, 4, 4, 1, 5, 4, 3, 3, 4, 4, 4], dtype=int32)
```

[77]:
```
#Evaluate the accuracy of model
from sklearn import metrics
x = metrics.accuracy_score(y_test,predicted)
print(x)
```

```
0.288
```

0.0.5 KNN gives 28.8 % accuracy

[78]:
```
# Confusion Matrix
from sklearn import metrics
print(metrics.confusion_matrix(y_test,predicted))
```

```
[[ 1  3  2  2  0]
 [ 3  1  2  3  2]
 [ 2  4 11 11  7]
 [ 4  5 13 18 12]
 [ 1  1  4  8  5]]
```

[79]:
```
# Classification report
print(metrics.classification_report(y_test,predicted))
```

```
              precision    recall  f1-score   support

           1       0.09      0.12      0.11         8
           2       0.07      0.09      0.08        11
           3       0.34      0.31      0.33        35
           4       0.43      0.35      0.38        52
           5       0.19      0.26      0.22        19

    accuracy                           0.29       125
   macro avg       0.23      0.23      0.22       125
weighted avg       0.32      0.29      0.30       125
```

[ ]:
```

```