Movielens Case Study

In [1]:
```
#Import - Import required packages
import pandas as pd;
import numpy as np;
import matplotlib.pyplot as plt;


# Machine learning Imports
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
```

Part 1 :

Data acquisition of the movielens dataset

In [2]:
```
#Reading movies file into data frames - .head(100)
#Function read_csv() is working for .dat files.

#define column names
m_cols = ['movie_id', 'title', 'genre']

#read_csv movied.dat file items are separated using 'sep'
#mnetion column names, engine = pyhton ( for some reason we were getting error )
#header = none - as we do not have any header information on first line..
dfMovies = pd.read_csv('movies.dat',sep='::',engine='python', names=m_cols, header = None)
dfMovies.dropna(inplace=True)
dfMovies.head()
```

Out[2]:

	movie_id	title	genre
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

In [3]:
```
#Reading ratings file into data frames

r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']
dfRatings = pd.read_csv('ratings.dat',sep='::',engine='python',names=r_cols, header = None)
dfRatings.dropna(inplace=True)
dfRatings.head()
```

Out[3]:

	user_id	movie_id	rating	unix_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

In [4]:
```
#Reading users file into data frames

u_cols = ['user_id', 'sex', 'age', 'occupation', 'zip_code']
dfUsers = pd.read_csv('users.dat',sep='::',engine='python',names=u_cols, header = None)
dfUsers.dropna(inplace=True)
dfUsers.head()
```

Out[4]:

	user_id	sex	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

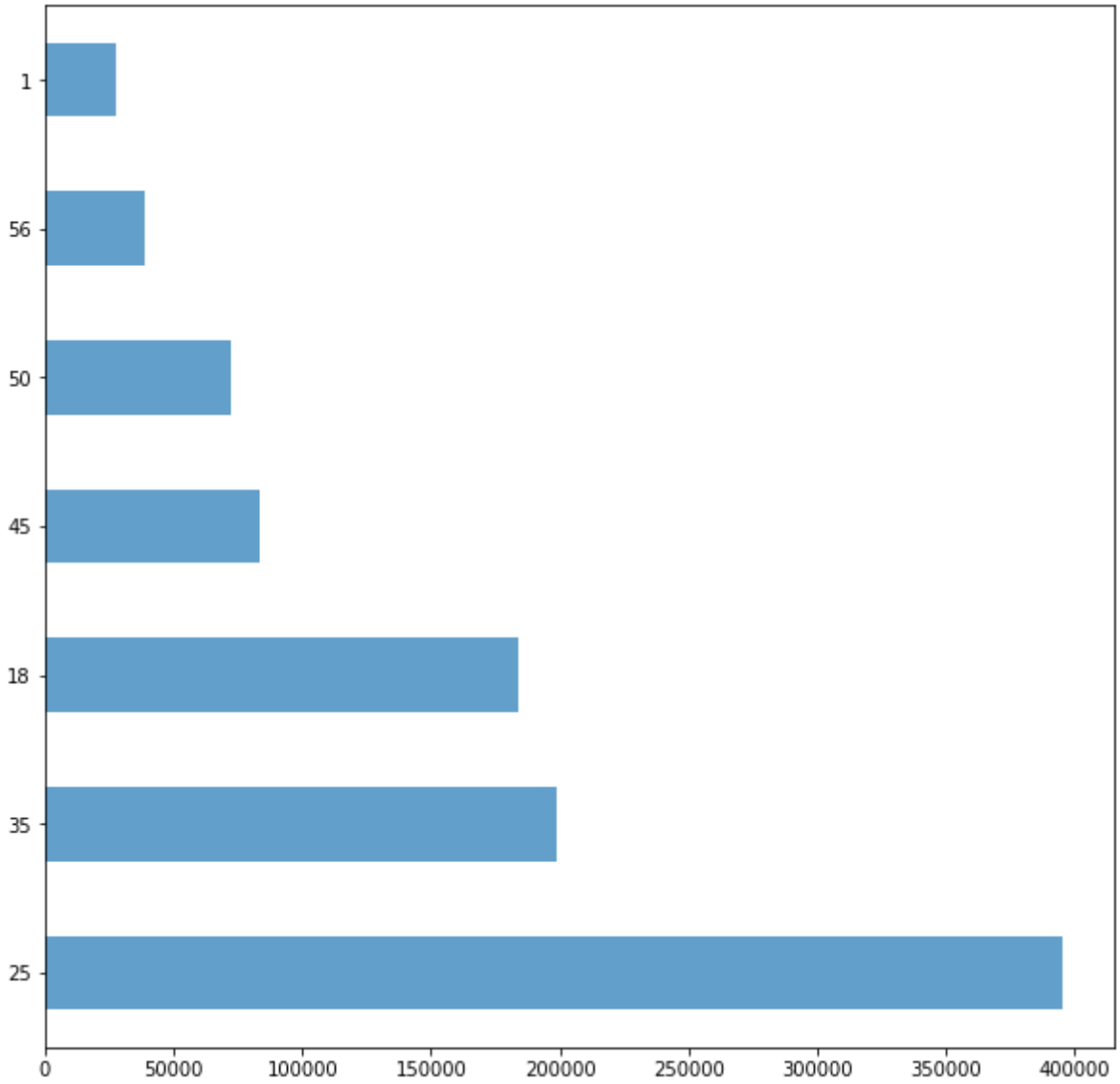
In [5]:
```
#Merge data frames to create a master data frame

#We can conert the fields in integer if we want, as they all are in string..
#dfMovies['movie_id'] = dfMovies['movie_id'].astype(int)
#dfRatings['movie_id'] = dfRatings['movie_id'].astype(int)

movie_ratings = pd.merge(dfMovies, dfRatings)

# *** Final Master Data. ***
master_data = pd.merge(movie_ratings, dfUsers)
master_data.head()

# Some MovieIDs do not correspond to a movie due to accidental duplicate entries and/or test entries
# This issue has been handled in the above merge condition as it returns only those matching records.
```

Out[5]:

	movie_id	title	genre	user_id	rating	unix_timestamp	sex	age	occupation	zip_code
0	1	Toy Story (1995)	Animation|Children's|Comedy	1	5	978824268	F	1	10	48067
1	48	Pocahontas (1995)	Animation|Children's|Musical|Romance	1	5	978824351	F	1	10	48067
2	150	Apollo 13 (1995)	Drama	1	5	978301777	F	1	10	48067
3	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Fantasy|Sci-Fi	1	4	978300760	F	1	10	48067
4	527	Schindler's List (1993)	Drama|War	1	5	978824195	F	1	10	48067

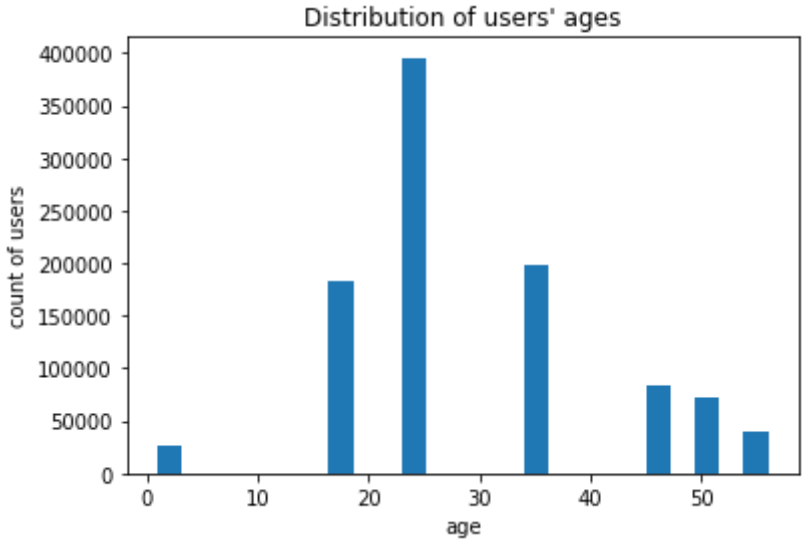
2. Perform the Exploratory Data Analysis (EDA) for the users dataset

In [6]:
```
#Visualize user age distribution
master_data['age'].value_counts().plot(kind='barh',alpha=0.7,figsize=(10,10))
plt.show()
```



In [7]:
```
master_data.age.plot.hist(bins=25)
plt.title("Distribution of users' ages")
plt.ylabel('count of users')
plt.xlabel('age')
```

Out[7]:
```
Text(0.5, 0, 'age')
```

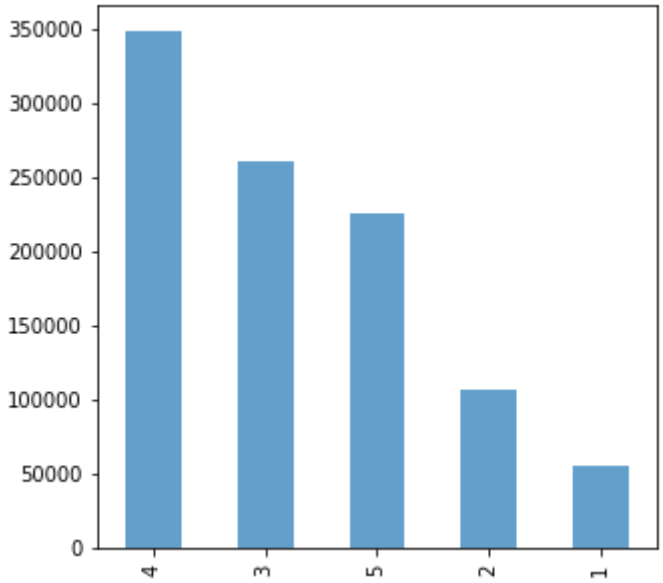


In [8]:
```
labels = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79']
master_data['age_group'] = pd.cut(master_data.age, range(0, 81, 10), right=False, labels=labels)
master_data[['age', 'age_group']].drop_duplicates()[:10]
```

Out[8]:

	age	age_group
0	1	0-9
53	50	50-59
124	25	20-29
369	35	30-39
770	18	10-19
2778	45	40-49
5001	56	50-59

In [9]:
```
#Visualize overall rating by users
master_data['rating'].value_counts().plot(kind='bar',alpha=0.7,figsize=(5,5))
plt.show()
```

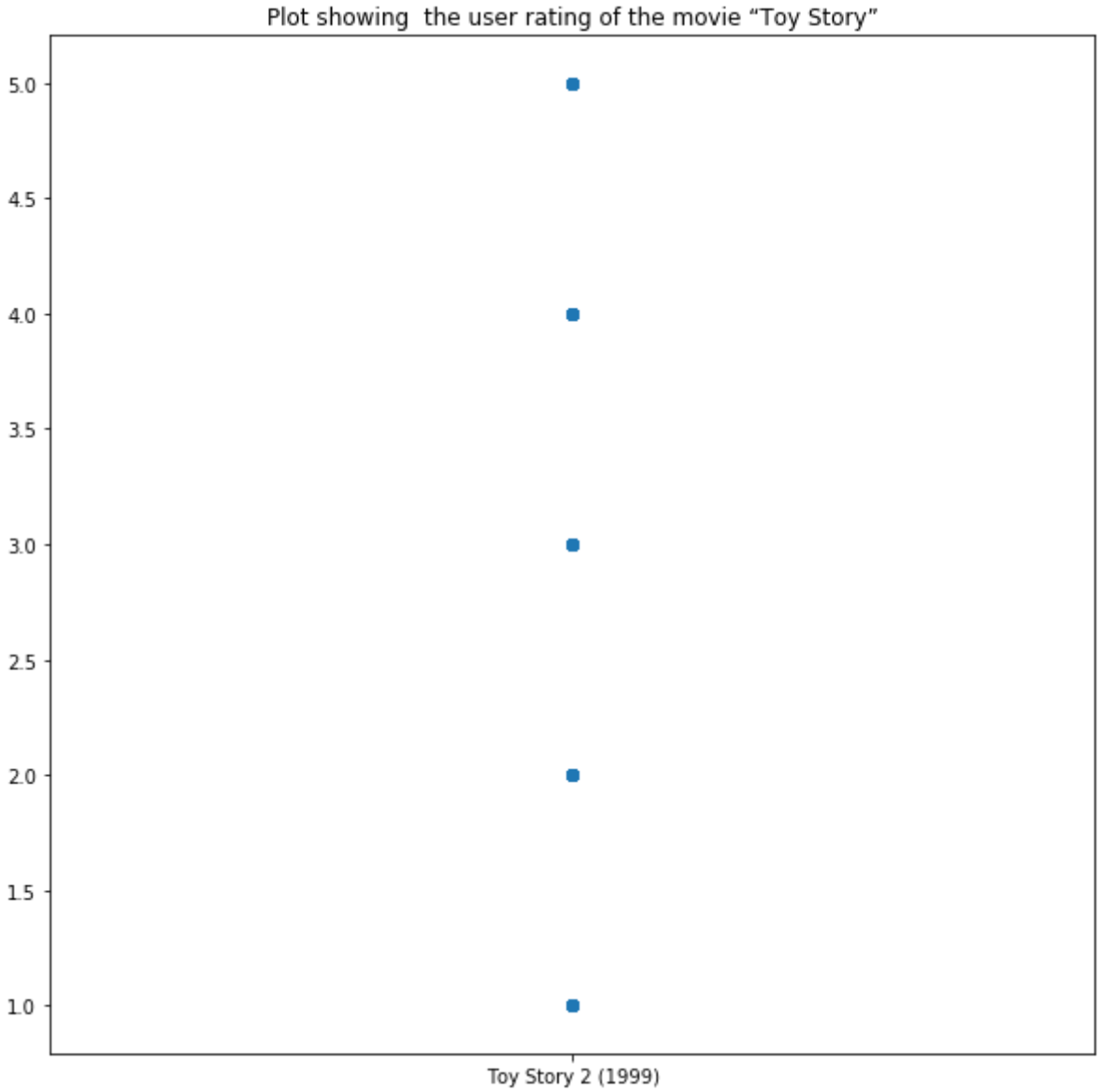


In [10]:
```
groupedby_movieName = master_data.groupby('title')
groupedby_rating = master_data.groupby('rating')
groupedby_uid = master_data.groupby('user_id')
```

In [11]:
```
# 2. User rating of the movie “Toy Story”

ToyStory_data = groupedby_movieName.get_group('Toy Story 2 (1999)')
ToyStory_data.shape
```

Out[11]:
```
(1585, 11)
```

In [12]:
```
# Visualize the user rating of the movie “Toy Story”

plt.figure(figsize=(10,10))
plt.scatter(ToyStory_data['title'],ToyStory_data['rating'])
plt.title('Plot showing  the user rating of the movie “Toy Story”')
plt.show()
```

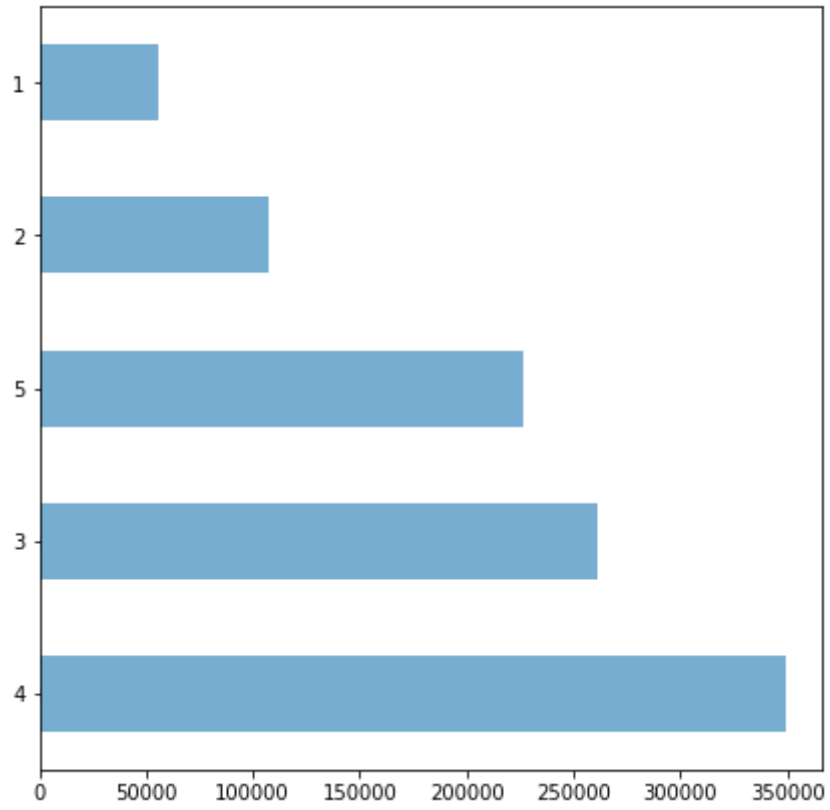

Plot showing the user rating of the movie “Toy Story”
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Toy Story 2 (1999)

In [13]:
```
ToyStory_data[['title','age_group']][:3]
```

Out[13]:

	title	age_group
50	Toy Story 2 (1999)	0-9
346	Toy Story 2 (1999)	20-29
715	Toy Story 2 (1999)	30-39

In [14]:
```
# 3. Find and visualize the top 25 movies by viewership rating
top_25 = master_data[25:]
top_25['rating'].value_counts().plot(kind='barh',alpha=0.6,figsize=(7,7))
plt.show()
```



In [15]:
```
# 4. Find the ratings for all the movies reviewed by for a particular user of user id = 2696

userid_2696 = groupedby_uid.get_group(2696)
userid_2696[['user_id','title','rating']]
```

Out[15]:

	user_id	title	rating
991035	2696	Client, The (1994)	3
991036	2696	Lone Star (1996)	5
991037	2696	Basic Instinct (1992)	4
991038	2696	E.T. the Extra-Terrestrial (1982)	3
991039	2696	Shining, The (1980)	4
991040	2696	Back to the Future (1985)	2
991041	2696	Cop Land (1997)	3
991042	2696	L.A. Confidential (1997)	4
991043	2696	Game, The (1997)	4
991044	2696	I Know What You Did Last Summer (1997)	2
991045	2696	Devil's Advocate, The (1997)	4
991046	2696	Midnight in the Garden of Good and Evil (1997)	4
991047	2696	Palmetto (1998)	4
991048	2696	Wild Things (1998)	4
991049	2696	Perfect Murder, A (1998)	4
991050	2696	I Still Know What You Did Last Summer (1998)	2
991051	2696	Psycho (1998)	4
991052	2696	Lake Placid (1999)	1
991053	2696	Talented Mr. Ripley, The (1999)	4
991054	2696	JFK (1991)	1

Part 2 :

Feature Engineering

In [16]:
```
# Categorize movies genres properly. Working later with +20MM rows of strings proved very resource consuming
# 1 . Find out all the unique genres (Hint: split the data in column genre making a list and then
# process the data to find out only the unique categories of genres)

genres_unique = pd.DataFrame(dfMovies.genre.str.split('|').tolist()).stack().unique()
genres_unique = pd.DataFrame(genres_unique, columns=['genre']) # Format into DataFrame to store later
genres_unique
```

Out[16]:

	genre
0	Animation
1	Children's
2	Comedy
3	Adventure
4	Fantasy
5	Romance
6	Drama
7	Action
8	Crime
9	Thriller
10	Horror
11	Sci-Fi
12	Documentary
13	War
14	Musical
15	Mystery
16	Film-Noir
17	Western

In [17]:
```
# using one-hot encoder from pandas -
# 2. Create a separate column for each genre category with a one-hot encoding ( 1 and 0)
# whether or not the movie belongs to that genre.
one_hot_encoded_genre_predictors = pd.get_dummies(genres_unique)
one_hot_encoded_genre_predictors.head()
```

Out[17]:

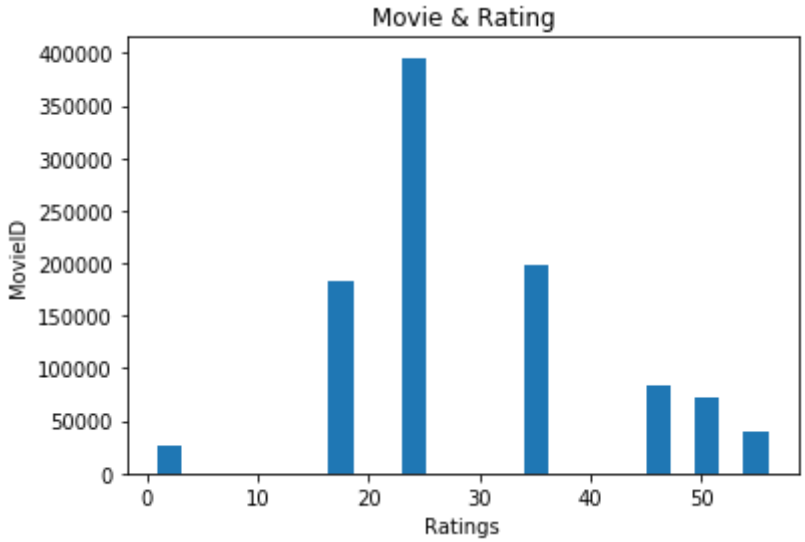
	genre_Action	genre_Adventure	genre_Animation	genre_Children's	genre_Comedy	genre_Crime	genre_Documentary	genre_Drama	genre_Fantas
0	0	0	1	0	0	0	0	0	
1	0	0	0	1	0	0	0	0	
2	0	0	0	0	1	0	0	0	
3	0	1	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	

In [18]:
```
#First 500 extracted records

top_500 = master_data[500:]
```

In [19]:
```
#Creating Features and Labels
#Creating Train and Test Data

features = top_500[['movie_id','age','occupation']].values
labels = top_500[['rating']].values
train, test, train_labels, test_labels = train_test_split(features,labels,test_size=0.33,random_state=42)
```

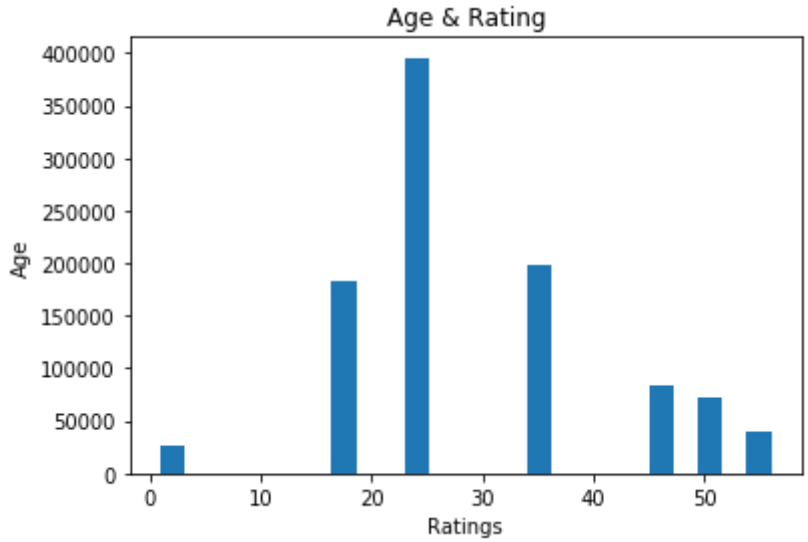
In [20]:
```
#Create a histogram for movie

master_data.age.plot.hist(bins=25)
plt.title("Movie & Rating")
plt.ylabel('MovieID')
plt.xlabel('Ratings')
```

Out[20]: Text(0.5, 0, 'Ratings')



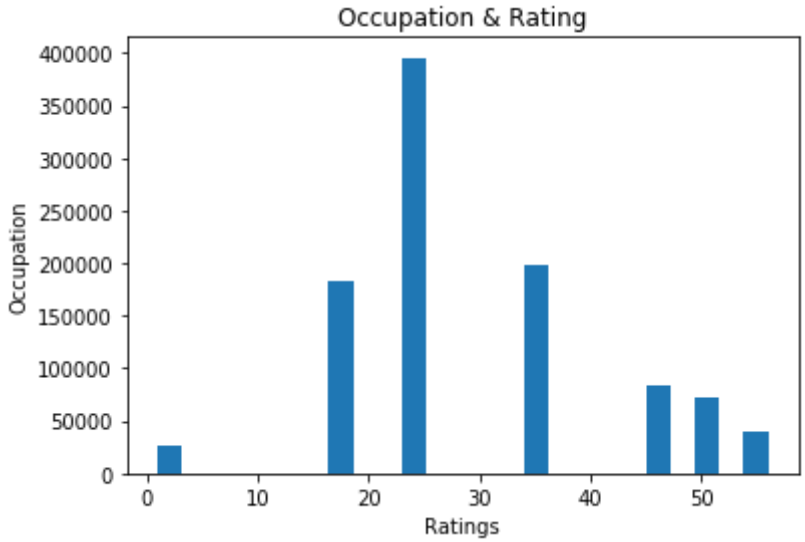
In [21]:
```
#Create a histogram for age

master_data.age.plot.hist(bins=25)
plt.title("Age & Rating")
plt.ylabel('Age')
plt.xlabel('Ratings')
```

Out[21]: Text(0.5, 0, 'Ratings')



In [22]:
```
#Create a histogram for occupation

master_data.age.plot.hist(bins=25)
plt.title("Occupation & Rating")
plt.ylabel('Occupation')
plt.xlabel('Ratings')
```

Out[22]: Text(0.5, 0, 'Ratings')



In [23]:
```
# Logistic Regression

logreg = LogisticRegression()
logreg.fit(train, train_labels)
Y_pred = logreg.predict(test)
acc_log = round(logreg.score(train, train_labels) * 100, 2)
acc_log
```

```
C:\Users\Sameer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be
changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\Sameer\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Sameer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:460: FutureWarning: Default multi_class wi
ll be changed to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

Out[23]: 34.9

In [24]:
```
# K Nearest Neighbors Classifier

knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(train, train_labels)
Y_pred = knn.predict(test)
acc_knn = round(knn.score(train, train_labels) * 100, 2)
acc_knn
```

```
C:\Users\Sameer\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: DataConversionWarning: A column-vector y was passe
d when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  after removing the cwd from sys.path.
```

Out[24]: 44.92

In [25]:
```
# Support Vector Machines

#svc = SVC()
#svc.fit(train, train_labels)
#Y_pred = svc.predict(test)
#acc_svc = round(svc.score(train, train_labels) * 100, 2)
#acc_svc
```

In [26]:
```
# Gaussian Naive Bayes

gaussian = GaussianNB()
gaussian.fit(train, train_labels)
Y_pred = gaussian.predict(test)
acc_gaussian = round(gaussian.score(train, train_labels) * 100, 2)
acc_gaussian
```

```
C:\Users\Sameer\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[26]:
```
34.9
```

In [27]:
```
# Perceptron

perceptron = Perceptron()
perceptron.fit(train, train_labels)
Y_pred = perceptron.predict(test)
acc_perceptron = round(perceptron.score(train, train_labels) * 100, 2)
acc_perceptron
```

```
C:\Users\Sameer\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:166: FutureWarning: max_iter an
d tol parameters have been added in Perceptron in 0.19. If both are left unset, they default to max_iter=5 and tol=Non
e. If tol is not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol wi
ll be 1e-3.
  FutureWarning)
C:\Users\Sameer\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[27]: 11.35

In [28]:
```
# Linear SVC

linear_svc = LinearSVC()
linear_svc.fit(train, train_labels)
Y_pred = linear_svc.predict(test)
acc_linear_svc = round(linear_svc.score(train, train_labels) * 100, 2)
acc_linear_svc
```

```
C:\Users\Sameer\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Sameer\Anaconda3\lib\site-packages\sklearn\svm\base.py:931: ConvergenceWarning: Liblinear failed to converge,
increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[28]: 24.34

In [29]:
```
# Stochastic Gradient Descent

sgd = SGDClassifier()
sgd.fit(train, train_labels)
Y_pred = sgd.predict(test)
acc_sgd = round(sgd.score(train, train_labels) * 100, 2)
acc_sgd
```

```
C:\Users\Sameer\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:166: FutureWarning: max_iter an
d tol parameters have been added in SGDClassifier in 0.19. If both are left unset, they default to max_iter=5 and tol=N
one. If tol is not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol
will be 1e-3.
  FutureWarning)
C:\Users\Sameer\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[29]: 33.65

In [30]:
```
# Decision Tree

decision_tree = DecisionTreeClassifier()
decision_tree.fit(train, train_labels)
Y_pred = decision_tree.predict(test)
acc_decision_tree = round(decision_tree.score(train, train_labels) * 100, 2)
acc_decision_tree
```

Out[30]: 56.6

In [31]:
```
# Random Forest

random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(train, train_labels)
Y_pred = random_forest.predict(test)
random_forest.score(train, train_labels)
acc_random_forest = round(random_forest.score(train, train_labels) * 100, 2)
acc_random_forest
```

```
C:\Users\Sameer\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: DataConversionWarning: A column-vector y was passe
d when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  after removing the cwd from sys.path.
```

Out[31]:
```
56.59
```

In [32]:
```
models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent', 'Linear SVC',
              'Decision Tree'],
    'Score': [acc_knn, acc_log,
              acc_random_forest, acc_gaussian, acc_perceptron,
              acc_sgd, acc_linear_svc, acc_decision_tree]})
models.sort_values(by='Score', ascending=False)
```

Out[32]:

	Model	Score
7	Decision Tree	56.60
2	Random Forest	56.59
0	KNN	44.92
1	Logistic Regression	34.90
3	Naive Bayes	34.90
5	Stochastic Gradient Decent	33.65
6	Linear SVC	24.34
4	Perceptron	11.35

END OF PROJECT

Additional Analysis

In [33]:
```
#  Top 10 most rated movies ( Note - Most rated moveies, not high rated .. )

most_rated_movies = master_data.groupby('title').size().sort_values(ascending=False)[:10]
#most_rated_movies
```

In [34]:
```
#  Top 3 most rated Genre

most_rated_genre = master_data.groupby('genre').size().sort_values(ascending=False)[:3]
# most_rated_genre
```

In [35]:
```
# Top 10 most rated Users
most_rated_users = master_data.groupby('user_id').size().sort_values(ascending=False)[:10]
most_rated_users
```

Out[35]:
```
user_id
4169    2314
1680    1850
4277    1743
1941    1595
1181    1521
889     1518
3618    1344
2063    1323
1150    1302
1015    1286
dtype: int64
```

In [36]:
```
# Top 3 most movie rated regions
most_rated_region = master_data.groupby('zip_code').size().sort_values(ascending=False)[:3]
most_rated_region
```

Out[36]:
```
zip_code
94110    3802
60640    3430
98103    3204
dtype: int64
```

In [37]:
```
# Top most rated user groups for movie genre, age range

most_rated_genre_usergrp = master_data.groupby(['genre','age']).size().sort_values(ascending=False)[:5]
most_rated_genre_usergrp
```

Out[37]:
```
genre   age
Comedy  25     48444
Drama   25     42834
Comedy  18     24204
Drama   35     22442
Comedy  35     21868
dtype: int64
```

In [38]:
```
#Top 3 most rated occupation users
# Users working under 'X' occupation rated movies.
# Ex. 4 is "college/grad student" and 7 is "executive/managerial"

most_rated_occupation = master_data.groupby('occupation').size().sort_values(ascending=False)[:3]
most_rated_occupation
```

Out[38]:
```
occupation
4    131032
0    130499
7    105425
dtype: int64
```

In [39]:
```
#Top 3 most rated occupation users under genre
# Users working under 'X' occupation rated 'Y' genre

most_rated_occupation_genre = master_data.groupby(['occupation','genre']).size().sort_values(ascending=False)[:3]
most_rated_occupation_genre
```

Out[39]:
```
occupation  genre
4           Comedy    16591
0           Comedy    15260
            Drama     14408
dtype: int64
```

In [40]:
```
# Most highly rated movies.

high_rated_movies = master_data.groupby('title').agg({'rating': [np.size, np.mean]})[:3]
high_rated_movies.sort_values([('rating', 'mean')], ascending=False).head()
```

Out[40]:

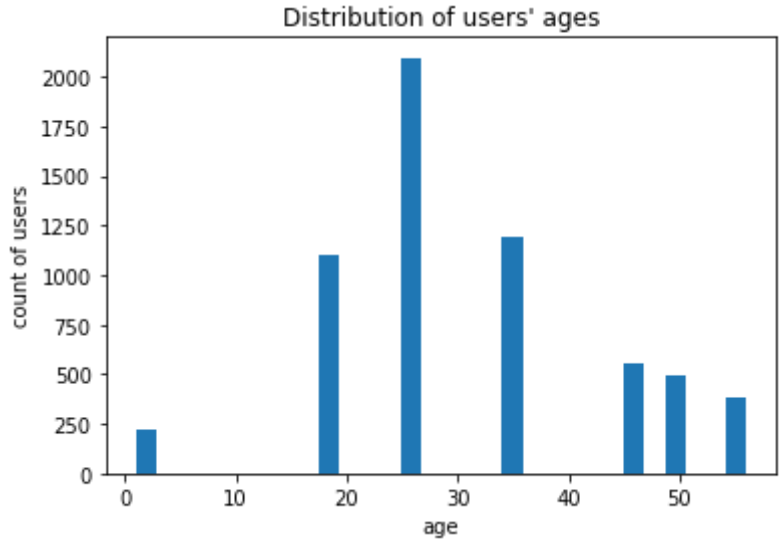
	rating	
	size	mean
title		
'Night Mother (1986)	70	3.371429
$1,000,000 Duck (1971)	37	3.027027
'Til There Was You (1997)	52	2.692308

In [41]:
```
# Following are the movies that have been rated at least 30 times. ( These results look realistic)
atleast_30 = high_rated_movies['rating']['size'] >= 30
high_rated_movies[atleast_30].sort_values([('rating', 'mean')], ascending=False)[:5]
```

Out[41]:

	rating	
	size	mean
title		
'Night Mother (1986)	70	3.371429
$1,000,000 Duck (1971)	37	3.027027
'Til There Was You (1997)	52	2.692308

In [42]:
```
# Problem Statement -

#User Age Distribution
#User rating of the movie “Toy Story”
#Top 25 movies by viewership rating
#Find the ratings for all the movies reviewed by for a particular user of user id = 2696

# User Age Distribution
dfUsers.age.plot.hist(bins=30)
plt.title("Distribution of users' ages")
plt.ylabel('count of users')
plt.xlabel('age');
```



End Of Additional Analysis

In [ ]: