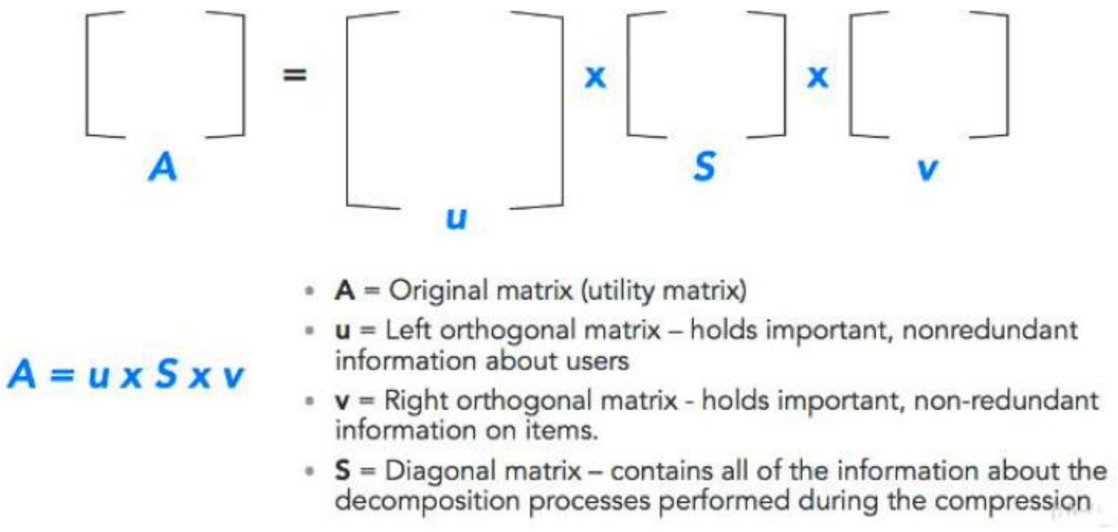
Model Based Collaborative Filtering

[zohreh Karimi]

With model-based collaborative filtering systems, you build a recommender model from user ratings, and then mke recommendations based on the model. For this segment we use these parameters:

- **Truncated singular value decomposition:** SVD is a linear algebra method that can decompose a utility matrix into three compressed matrices. Model based recommender use these compressed matrices to make recommendations without having to refer back tho the complete data set. Also with SVD, you uncover latent variables.



- **A** = Original matrix (utility matrix)
- **u** = Left orthogonal matrix – holds important, nonredundant information about users
- **v** = Right orthogonal matrix - holds important, non-redundant information on items.
- **S** = Diagonal matrix – contains all of the information about the decomposition processes performed during the compression

The Anatomy of SVD

- **Utility Matrix(user item matrix):** This matrices contain values for each user,each item and the rating each user gave to each item. Note that utility matrices are sparse, because every user does not review every item.

Let's go to use Python to build a model based collaborative filtering system.

Step 1: import Libraries

```
import pandas as pd
import numpy as np
import sklearn
from sklearn.decomposition import TruncatedSVD
```

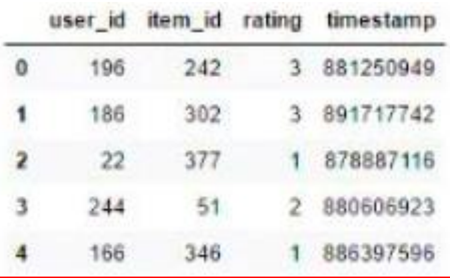
dataset: for dataset we are going to use the 100k MovieLens dataset. You can download that from the link i provited here: https://grouplens.org/datasets/movielens/100k/

We are going build a model-based movie recommender. So our **users** will **be movie goers** and our **items** will be **movies**.

Model Based Collaborative Filtering

Step 2: Preparing the data

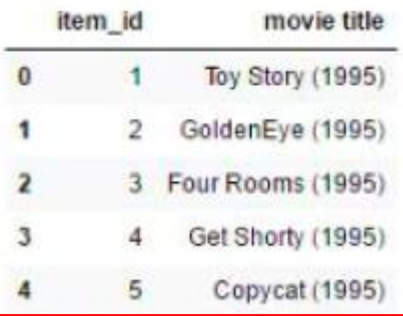
```
columns=['user_id','item_id','rating','timestamp']
data=pd.read_csv('u.data',sep='\t',names=columns)
data.head()
```



	user_id	item_id	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

This dataset contains records for each of the users and each of the movies they've reviewed,and the rating they gave each of the movies. Next import the item attributes like movie names, release dates, and genre into which the movies fall.

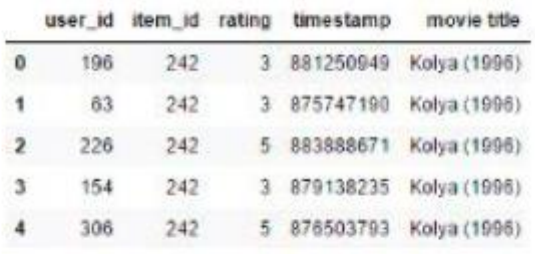
```
columns=['item_id', 'movie title', 'release date', 'video release date', 'IMDb URL', 'unknown',
'Action', 'Adventure','Animation','Childrens','Comedy','Crime','Documentary', 'Drama', 'Fantasy',
'Film-Noir', 'Horror','Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western']
movies=pd.read_csv('u.item',sep='|',names=columns, encoding='latin-1')
movie_names=pd.DataFrame(movies, columns=['item_id','movie title'])
print(movie_names)
```



	item_id	movie title
0	1	Toy Story (1995)
1	2	GoldenEye (1995)
2	3	Four Rooms (1995)
3	4	Get Shorty (1995)
4	5	Copycat (1995)

Next we are going to combine these two data frames into one:

```
combined_movies_data=pd.merge(data,movies, on='item_id')
print(combined_movies_data)
```



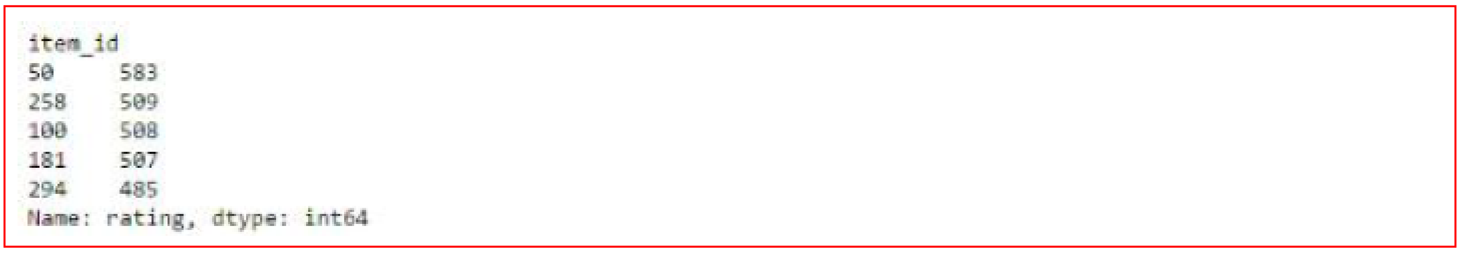
	user_id	item_id	rating	timestamp	movie title
0	196	242	3	881250949	Kolya (1996)
1	63	242	3	875747190	Kolya (1996)
2	226	242	5	883888671	Kolya (1996)
3	154	242	3	879138235	Kolya (1996)
4	306	242	5	876503793	Kolya (1996)

Model Based Collaborative Filtering

Step3: Find movie with most number of reviews

As you see, have duplicate data it's because more than one movie goer reviewed the same movie. Let's see what movie have the most number of reviews.

```
rating=pd.DataFrame(combined_movies_data.groupby('item_id')['rating'].count().sort_values(as
cending=False).head())
```



```
item_id
50     583
258    509
100    508
181    507
294    485
Name: rating, dtype: int64
```

Item 50 get 583 rating, so let's see what the name of that movie is:

```
Filter= combined_movies_data['item_id']==50
print(combined_movies_data[Filter]['movie title'].unique())
```

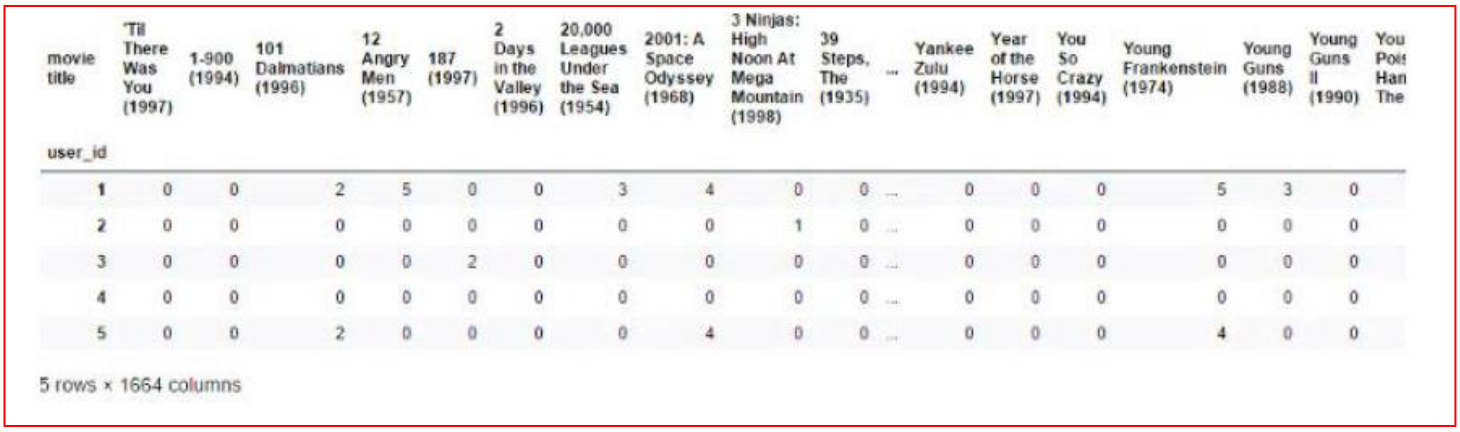
```
['Star Wars (1977)']
```

item ID 50 is **Star Wars,1977** which is a very popular movie.

Step4: Building a Utility Matrix

Now we are going to movie into building a utility matrix, this matrix is going to contain a value for each user and each movie. where the user did provide a movie review, that rating shows a numeric value all other user movie values will return as null.

```
rating_crosstab_mat=combined_movies_data.pivot_table(values='rating',index='user_id',
columns='movie title',fill_value=0)
print(rating_crosstab_mat)
```



movie title	'Til There Was You (1997)	1-900 (1994)	101 Dalmatians (1996)	12 Angry Men (1957)	187 (1997)	2 Days in the Valley (1996)	20,000 Leagues Under the Sea (1954)	2001: A Space Odyssey (1968)	3 Ninjas: High Noon At Mega Mountain (1998)	39 Steps, The (1935)	...	Yankee Zulu (1994)	Year of the Horse (1997)	You So Crazy (1994)	Young Frankenstein (1974)	Young Guns (1988)	Young Guns II (1990)	You Poi Han The
user_id																		
1	0	0	2	5	0	0	3	4	0	0	...	0	0	0	5	3	0	
2	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	
3	0	0	0	0	2	0	0	0	0	0	...	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
5	0	0	2	0	0	0	0	4	0	0	...	0	0	0	4	0	0	

5 rows × 1664 columns

No, since we want to recommend movies, we want to reserve the movie names as uncompressed rows. we want to use the similarities between users to decide which movies to recommend, so use Truncated SVD to compress all of the user ratings down to just 10 latent variables. These

variables are going to capture most of the information that was stored in the 943 user columns previously.
So the first thing we will do is to transpose out matrix, that movies are represented by rows, and users are represented by columns. Then use SVD to compress this matrix down to a 1664 by 10 matrix. All of the individual movie names will be retained along the rows. But the users will have been compressed down to 10 synthetic components, that represent a generalized view of user's tastes.

Step5: Transposing the Matrix

So we have a utility matrix, next thing is we are going to take this utility matrix and transpose it and then use SVD to decomposite it down to synthetic representations.

```
print(rating_crosstab_mat.shape)
```

```
(943,1664)
```

```
X=rating_crosstab_mat.values.T
print(X.shape)
```

```
(1664,943)
```

Step6: Decomposing the Matrix

```
SVD=TruncatedSVD(n_components=10,random_state=18)
result_mat=SVD.fit_transform(X)
print(result_mat.shape)
```

```
(1664, 10)
```

So we have 1664 by 10 matrix. Next let's movie into generating a correlation matrix. We 'll calculate the Pearson r correlation coefficient for every movie pair in the result_mat. With correlation being based on similarities between user preferences.

```
corr_mat=np.corrcoef(result_mat)
print(corr_mat.shape)
```

```
 (1664, 1664)
```

We'll take the array that represents Star Wars,and examine how well its user ratings correlate with the user ratings given to other movies in the dataset. For each movie pair in the matrix, we'll calculate how well they correlate, based on user preferences. The goal is to recommend the movie that has the highest correlation with our movie of interest, based on generalized user tastes.

Step7: Isolating Star Wars From the Correlation Matrix

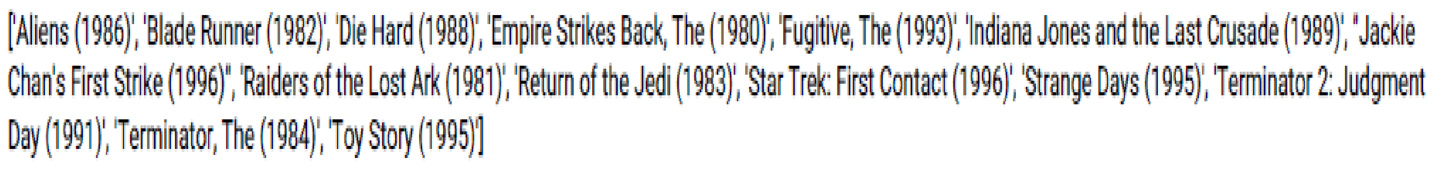
```
movies_names=rating_crosstab_mat.columns
movies_list=list(movies_names)
star_wars= movies_list.index('Star Wars (1977)')
print(star_wars)
corr_star_wars=corr_mat[star_wars]
print(corr_star_wars.shape)
```

```
1398
(1664,)
```

You can see we have a vertical array of 1664 rows each of the rows contains a Pearson r coefficient that indicate how well each movie in the dataset correlates with Star Wars, based on user preferences. Now, let's generate a list of movie names that exhibit a high degree of correlation with Star Wars.

Step8: Recommending a Highly Correlated Movie

```
print(list(movies_names[(corr_star_wars < 1.0) &  (corr_star_wars > 0.9)]))
```



```
['Aliens (1986)', 'Blade Runner (1982)', 'Die Hard (1988)', 'Empire Strikes Back, The (1980)', 'Fugitive, The (1993)', 'Indiana Jones and the Last Crusade (1989)', "Jackie Chan's First Strike (1996)", 'Raiders of the Lost Ark (1981)', 'Return of the Jedi (1983)', 'Star Trek: First Contact (1996)', 'Strange Days (1995)', 'Terminator 2: Judgment Day (1991)', 'Terminator, The (1984)', 'Toy Story (1995)']
```

Code

```
import pandas as pd
import numpy as np
import sklearn
from sklearn.decomposition import TruncatedSVD

columns=['user_id','item_id','rating','timestamp']
data=pd.read_csv('u.data',sep='\t',names=columns)
columns=['item_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']
movies=pd.read_csv('u.item',sep='|',names=columns, encoding='latin-1')
movie_names=pd.DataFrame(movies, columns=['item_id','movie title'])
combined_movies_data=pd.merge(data,movies, on='item_id')
rating=pd.DataFrame(combined_movies_data.groupby('item_id')['rating'].count().sort_values(as
cending=False).head())
Filter= combined_movies_data['item_id']==50
print(combined_movies_data[Filter]['movie title'].unique())
rating_crosstab_mat=combined_movies_data.pivot_table(values='rating',index='user_id',
columns='movie title',fill_value=0)
X=rating_crosstab_mat.values.T
SVD=TruncatedSVD(n_components=10,random_state=18)
result_mat=SVD.fit_transform(X)
corr_mat=np.corrcoef(result_mat)
movies_names=rating_crosstab_mat.columns
movies_list=list(movies_names)
star_wars= movies_list.index('Star Wars (1977)')
corr_star_wars=corr_mat[star_wars]
print(list(movies_names[(corr_star_wars < 1.0) &  (corr_star_wars > 0.9)]))
```