¿Qué son los Sistemas ó Motores de Recomendación?

> Los sistemas de recomendación, a veces llamados en inglés «recommender systems» son algoritmos que intentan «predecir» los siguientes ítems (productos, canciones, etc.) que querrá adquirir un usuario en particular.

Los Sistemas de Recomendación intentan personalizar al máximo lo que ofrecerán a cada usuario. Esto es ahora posible por la cantidad de información individual que podemos recabar de las personas y nos da la posibilidad de tener una mejor tasa de aciertos, mejorando la experiencia del internauta sin ofrecer productos a ciegas.

Ejercicio en Python: «Sistema de Recomendación de Repositorios Github»

Vamos a crear un **motor de recomendación de repositorios Github**. Es la propuesta que hago en el blog… porque los recomendadores de música, películas y libros ya están muy vistos!.
La idea es que si este recomendador le parece de interés a los lectores, en un futuro, publicarlo online para extender su uso.
Inicialmente contaremos con un set de datos limitado (pequeño), pero que como decía, podremos llevar a producción e ir agregando usuarios y repositorios para mejorar las sugerencias.

Codigo!

Cargando las librerias

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.metrics import mean_squared_error
4 from sklearn.model_selection import train_test_split
5 from sklearn.neighbors import NearestNeighbors
6 import matplotlib.pyplot as plt
7 import sklearn
```

Cargamos y previsualizamoás los 3 archivos de datos csv que utilizaremos:

```
1 df_users = pd.read_csv("users.csv")
2 df_repos = pd.read_csv("repos.csv")
3 df_ratings = pd.read_csv("ratings.csv")
4 print(df_users.head())
5 print(df_repos.head())
6 print(df_ratings.head())
```

	userId	username	name
0	1	iris9112	Isabel Ruiz Buriticá
1	2	dianaclarke	Diana
2	3	nateprewitt	Nate Prewitt
3	4	oldani	Ordanis Sanchez
4	5	waflessnet	waflessnet

	repoId	title	
0	1	airbnb / javascript	
1	2	kamranahmedse / developer-roadmap	Roadmap to becomin
2	3	microsoft / vscode	
3	4	torvalds / linux	
4	5	ytdl-org / youtube-dl	Command-line program to

	userId	repoId	rating
0	1	1	2
1	1	2	3
2	1	3	4
3	1	4	5
4	1	5	3

Vemos que tenemos un archivo con la información de los usuarios y sus identificadores, un archivo con la información de los repositorios y finalmente el archivo «ratings» que contiene la valoración por usuario de los repositorios. Como no tenemos REALMENTE una valoración

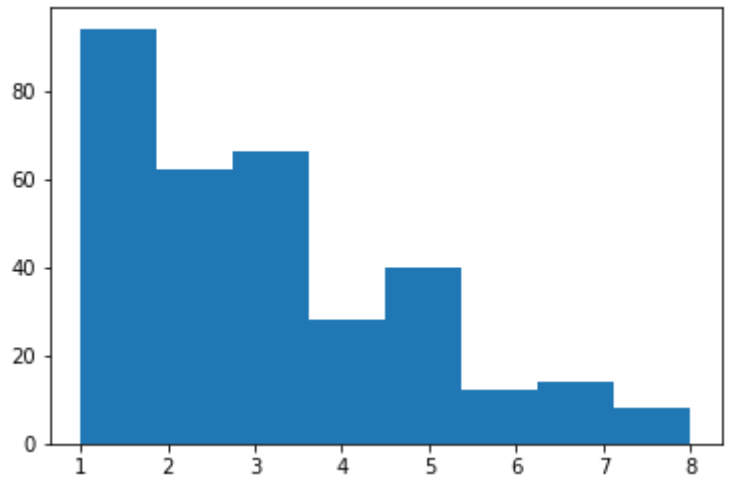
del 1 al 5 -como podríamos tener por ejemplo al valorar películas-, la columna rating **es el número de usuarios que tienen ese mismo repositorio** dentro de nuestra base de datos. Sigamos explorando para comprende un poco mejor:

```
1 n_users = df_ratings.userId.unique().shape[0]
2 n_items = df_ratings.repoId.unique().shape[0]
3 print (str(n_users) + ' users')
4 print (str(n_items) + ' items')
```

```
30 users
167 items
```

Vemos que es un dataset reducido, pequeño. Tenemos 30 usuarios y 167 repositorios valorados.

```
1 plt.hist(df_ratings.rating,bins=8)
```

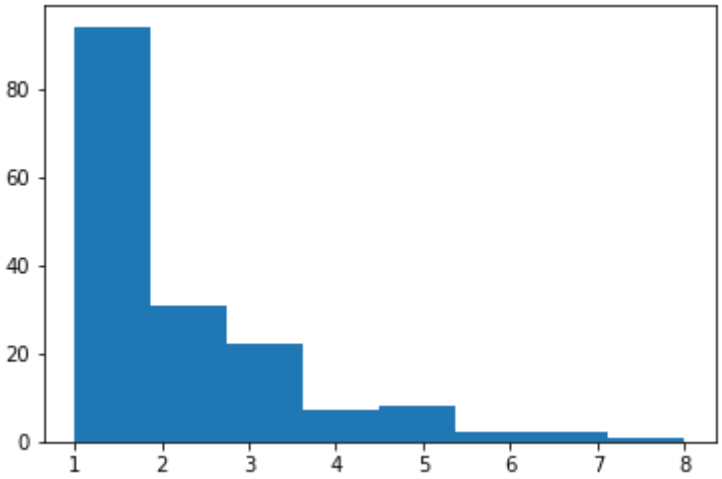


Tenemos más de 80 valoraciones con una puntuación de 1 y unas 40 con puntuación en 5. Veamos las cantidades exactas:

```
1 df_ratings.groupby(["rating"])["userId"].count()
```

```
rating
1 94
2 62
3 66
4 28
5 40
6 12
7 14
8 8
Name: userId, dtype: int64
```

```
1 plt.hist(df_ratings.groupby(["repoId"])["repoId"].count(),bins=8)
```



Aquí vemos la cantidad de repositorios y cuantos usuarios «los tienen». La mayoría de repos los tiene 1 sólo usuario, y no los demás. Hay unos 30 que los tienen 2 usuarios y unos 20 que coinciden 3 usuarios. La suma total debe dar 167.

Creamos la matriz usuarios/ratings

Ahora crearemos la matriz en la que cruzamos todos los usuarios con todos los repositorios.

```
1 df_matrix = pd.pivot_table(df_ratings, values='rating', index='userId', columns='repoId').fillna(0)
2 df_matrix
```

repoId	1	2	3	4	5	6	7	8	9	10
userId										
1	2.0	3.0	4.0	5.0	3.0	1.0	5.0	1.0	0.0	4.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0
5	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0

Vemos que rellenamos los «huecos» de la matriz con ceros. Y esos ceros serán los que deberemos reemplazar con las recomendaciones.

Sparcity

Veamos el porcentaje de sparcity que tenemos:

```
ratings = df_matrix.values
sparsity = float(len(ratings.nonzero()[0]))
sparsity /= (ratings.shape[0] * ratings.shape[1])
sparsity *= 100
print('Sparsity: {:4.2f}%'.format(sparsity))
```

```
Sparsity: 6.43%
```

Esto serán muchos «ceros» que rellenar (predecir)…

Dividimos en Train y Test set

Separamos en train y test para -más adelante- poder medir la calidad de nuestras recomendaciones.

¿Porqué es tan importante dividir en Train, Test y Validación del Modelo?

```
ratings_train, ratings_test = train_test_split(ratings, test_size = 0.2, random_state=42)
print(ratings_train.shape)
print(ratings_test.shape)
```

```
(24, 167)
(6, 167)
```

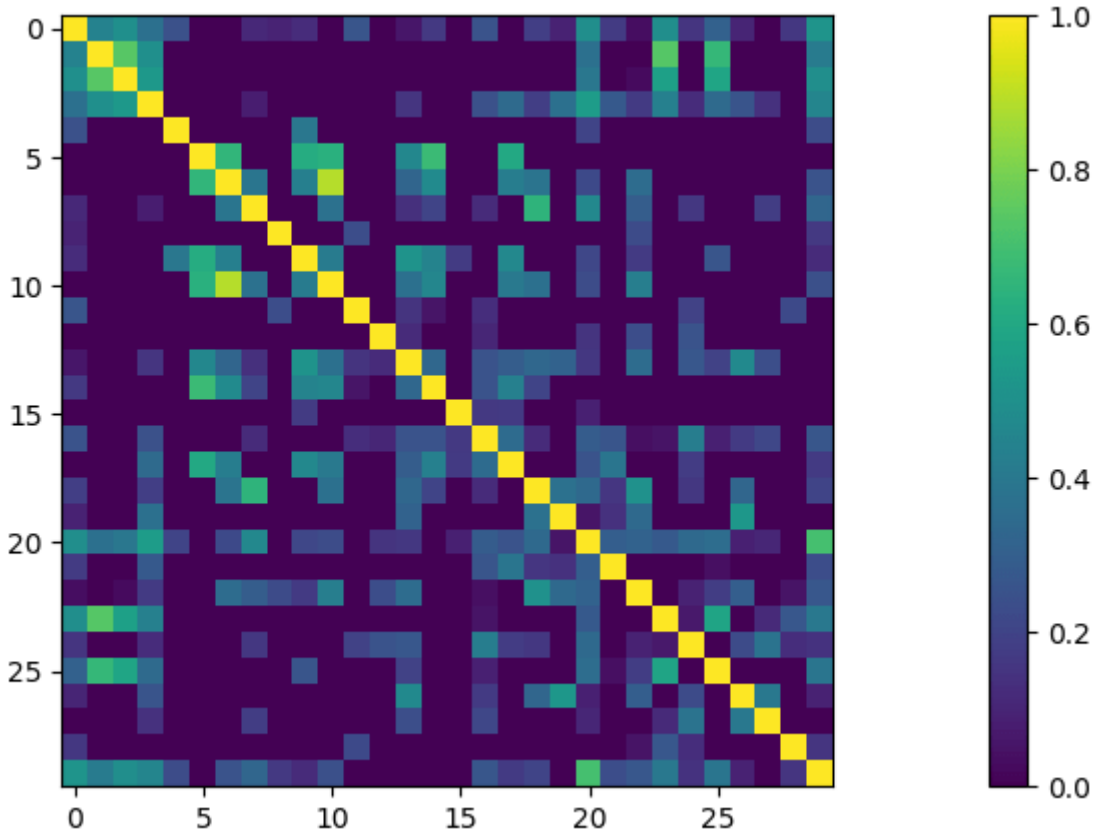
Matriz de Similitud: Distancias por Coseno

Ahora calculamos en una nueva matriz la similitud entre usuarios.

```
sim_matrix = 1 - sklearn.metrics.pairwise.cosine_distances(ratings)
print(sim_matrix.shape)
```

```
(30, 30)
```

```
plt.imshow(sim_matrix);
plt.colorbar()
plt.show()
```



Cuanto más cercano a 1, mayor similitud entre esos usuarios.