

Report Progetto Base

Mattia Foggetti

February 2023

Contents

1	Riassunto	3
2	Un'introduzione al problema	4
2.1	Obbiettivo Analisi Esplorativa	5
2.1.1	Distribuzione dei dati	5
2.1.2	Rating negli anni	9
2.1.3	Correlazioni	9
2.1.4	Correlazioni col rating	10
2.1.5	Correlazioni Taglie	11
3	Step del progetto	12
3.1	KNN	12
3.1.1	Elbow Method	15
3.1.2	GridSearch	16
3.2	Costruzione Modello	17
3.3	Filling della matrice di rating	18
3.4	Items da consigliare	19
3.5	K-Means	20
3.5.1	Elbow Method	22

3.5.2	K-Means nltk	23
3.5.3	K-Means sklearn	23
3.5.4	K-Means su Rating Predetti	26
3.6	Matrix Factorization	28
3.6.1	Algoritmo	29
3.6.2	Predizioni	29
4	Conclusioni	30
4.1	Analisi descrittiva	30
4.2	Correlazioni	30
4.3	Algoritmo di Predizione	31
4.4	Clustering	31

1 Riassunto

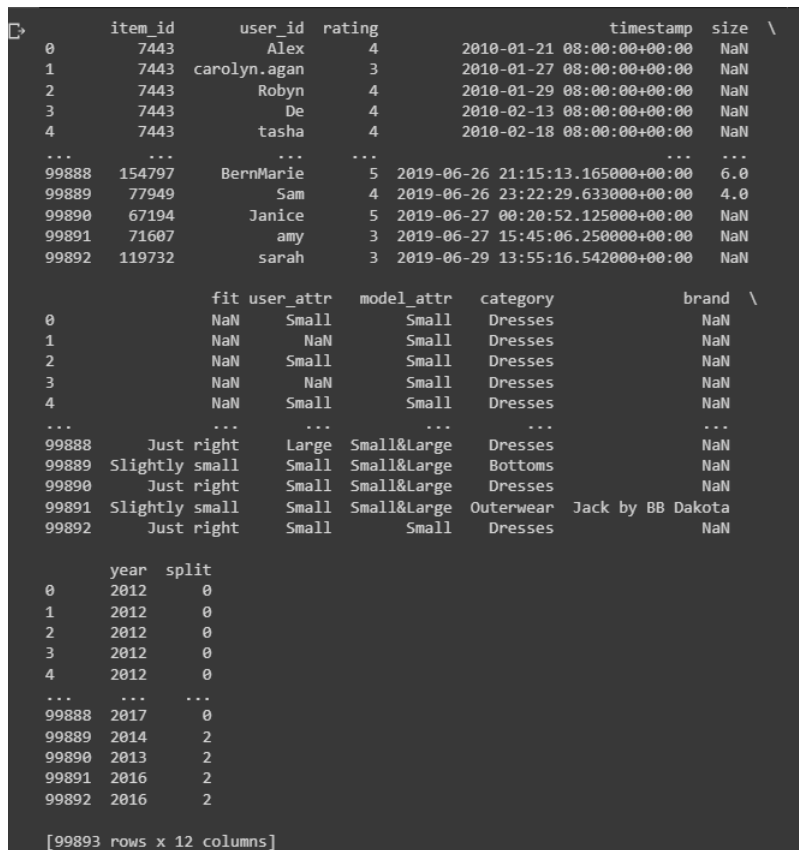
Il DataSet scelto é il seguente: Marketing Bias (ModCloth). Il dataset é un insieme di recensioni date da utenti ad item ognuna con un rating e vari attributi tra cui brand, la taglia (size), come calza il vestito (fit), ecc. I principali obiettivi usando il dataset sono:

- Fare Analisi esplorativa;
- Usare un algoritmo KNN per addestrare un modello usando parametri ottimali;
- Una volta addestrato il modello possiamo andare a predire i possibili rating che gli utenti potrebbero dare a uno o più item, infatti non tutti gli utenti recensiscono tutti gli item;
- Una volta che andiamo a prevedere possibili valori per gli item dati a utenti possiamo andare a raccomandare agli utenti degli item che loro non hanno recensito basandoci sul rating predetto;
- Tornando invece al nostro dataset usando un algoritmo per fare clustering come K-Means in modo tale da vedere come si suddividono gli utenti in base alle loro preferenze;
- Possiamo infine comparare la nostra accuracy, in modo particolare RMSE, ottenuta dal nostro modello KNN e compararlo a uno ottenuto usando un algoritmo SVD

Per ogni obiettivo i risultati ottenuti sono i seguenti: L'algoritmo KNN ha un rsme di circa 1, SVD é anche lui circa 1 anche se qualche millesimo migliore del KNN. Il clustering é stato diviso in 4, poiché dopo il wcss non migliorava praticamente più. I suoi silhouette score é 0.45 (approfondito più avanti).

2 Un'introduzione al problema

Il nostro Dataset come già detto é un insieme di recensioni di vestiti.



```
0      item_id  user_id  rating      timestamp  size \
1      7443      Alex      4      2010-01-21 08:00:00+00:00  NaN
2      7443  carolyn.agan      3      2010-01-27 08:00:00+00:00  NaN
3      7443      Robyn      4      2010-01-29 08:00:00+00:00  NaN
4      7443      De      4      2010-02-13 08:00:00+00:00  NaN
5      7443      tasha      4      2010-02-18 08:00:00+00:00  NaN
...      ...      ...      ...      ...      ...
99888  154797  BernMarie      5      2019-06-26 21:15:13.165000+00:00  6.0
99889  77949      Sam      4      2019-06-26 23:22:29.633000+00:00  4.0
99890  67194      Janice      5      2019-06-27 00:20:52.125000+00:00  NaN
99891  71607      amy      3      2019-06-27 15:45:06.250000+00:00  NaN
99892  119732      sarah      3      2019-06-29 13:55:16.542000+00:00  NaN

0      fit  user_attr  model_attr  category      brand \
1      NaN      Small      Small      Dresses      NaN
2      NaN      Small      Small      Dresses      NaN
3      NaN      Small      Small      Dresses      NaN
4      NaN      Small      Small      Dresses      NaN
...      ...      ...      ...      ...      ...
99888      Just right      Large  Small&Large      Dresses      NaN
99889  Slightly small      Small  Small&Large      Bottoms      NaN
99890      Just right      Small  Small&Large      Dresses      NaN
99891  Slightly small      Small  Small&Large  Outerwear  Jack by BB Dakota
99892      Just right      Small      Small      Dresses      NaN

0      year  split
1      2012      0
2      2012      0
3      2012      0
4      2012      0
...      ...      ...
99888  2017      0
99889  2014      2
99890  2013      2
99891  2016      2
99892  2016      2

[99893 rows x 12 columns]
```

Le colonne sono:

- Item_id, in questo campo abbiamo l'id dell'item recensito
- user_id, in questo campo abbiamo l'id dell'user che ha recensito l'item
- rating, qua abbiamo il voto che lo user ha dato all'item da 1 a 5
- timestamp, il momento in cui é stata fatta la recensione

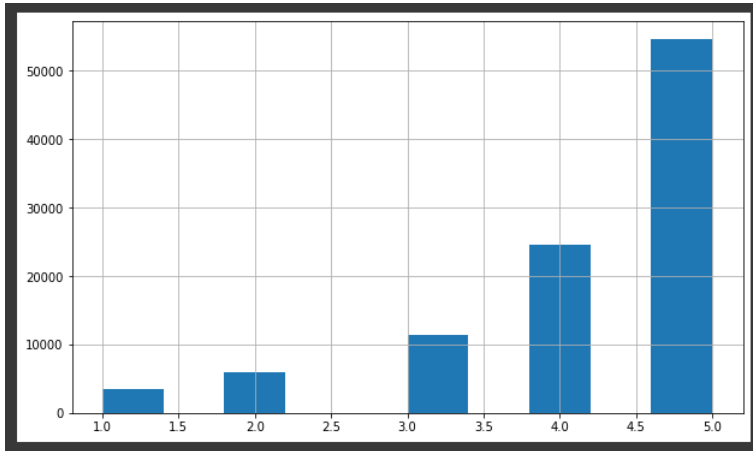
- size, la misura che l'utente ha preso per il suo item/vestito, vanno da 1 a 8
- fit, come gli calzava all'utente il capo. Variano da 'just right', 'Slightly Small', 'Very Small', 'Slightly Large', 'Very Large',
- user_attr, qua abbiamo la taglia dell'utente se esso sia Small oppure Large
- model_attr, qua abbiamo la taglia della modella dell'item, esso può essere Small oppure Small&Large
- category, esso può essere un Dresses, Outerwear, Top, Bottom. È appunto la categoria del vestito
- brand, qua abbiamo i vari brand che vende il sito

2.1 Obiettivo Analisi Esplorativa

L'obiettivo primario della nostra analisi esplorativa è quello di vedere la distribuzione dei nostri dati e le varie correlazioni che possono avere i nostri dati nell'avere un alto rating oppure se all'aumentare di un valore diminuisce il rating. Ci interessiamo anche alle correlazioni che possono avere gli altri dati tra di loro come il fit, size, ecc.

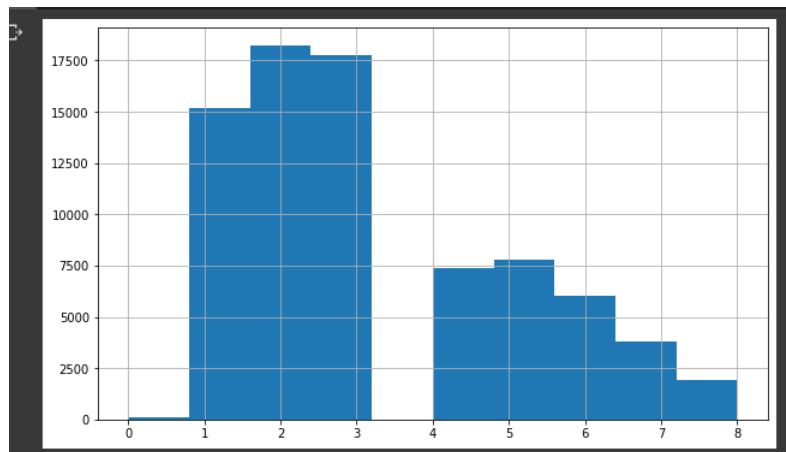
2.1.1 Distribuzione dei dati

La mia analisi esplorativa è partita dal vedere come fossero distribuiti i vari dati delle colonne, cioè capire se i valori venissero assunti in modo omogeneo o meno. Ho iniziato a fare un istogramma del rating per vedere come i valori da 1 a 5 fossero distribuiti, possiamo notare appunto che le istanze dove il rating assume valore 5 sono molto alti rispetto agli altri. In generale possiamo notare che i rating sono alti.

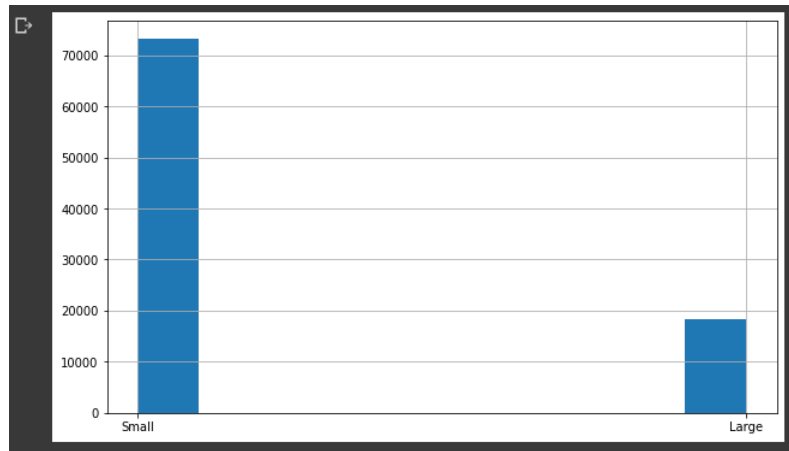


Andando a vedere altri grafici troviamo che:

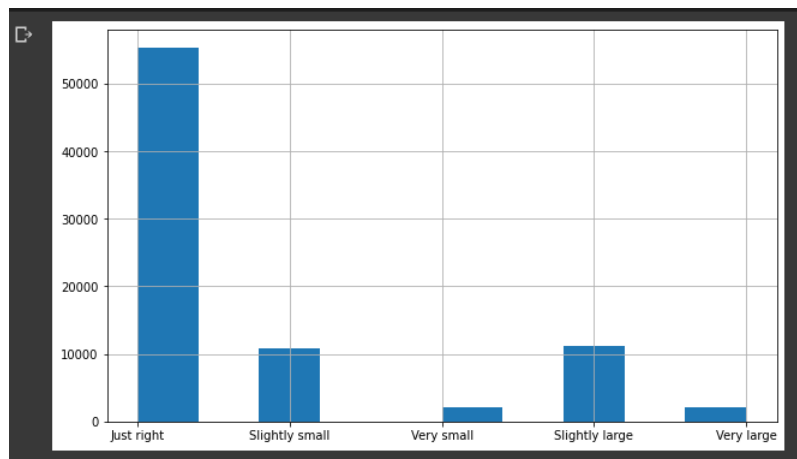
- Anche le taglie, cioè la size, é distribuita in modo non omogeneo in particolare le size minori o uguali a 3 vanno per la maggiore



- Possiamo trovare un forte sbilanciamento anche nello `user_attr` cioè di come viene identificato un utente se Small o Large. É intuibile il risultato dal fatto che la maggior parte delle taglie (size) siano nella parte di taglie "piccole"



- Infine abbiamo il fit, cioè se a una persona l'item calzava bene oppure no. Anche qui è piuttosto sbilanciata la distribuzione infatti alla maggior parte degli utenti l'item calza "Giusto", il risultato è sicuramente interessante e positivo per l'azienda poiché gli item venduti rispettano le taglie

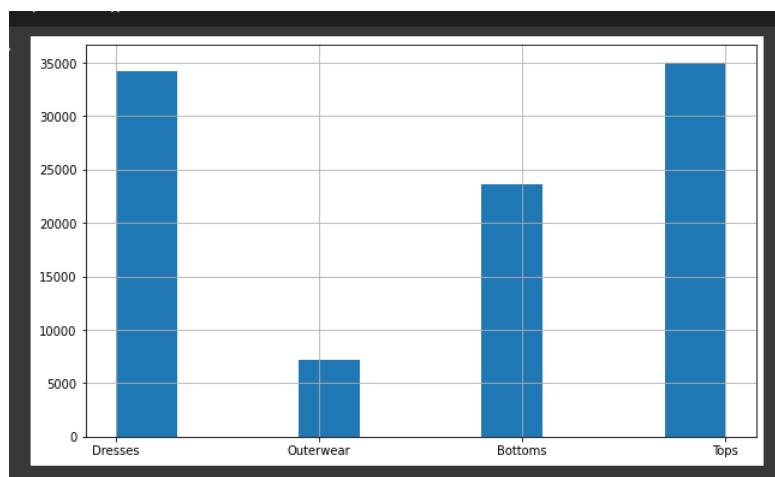


Ci sono anche delle distribuzioni omogenee come ad esempio :

- Come il model_attr dove appunto sono quasi pari Small e Small&Large

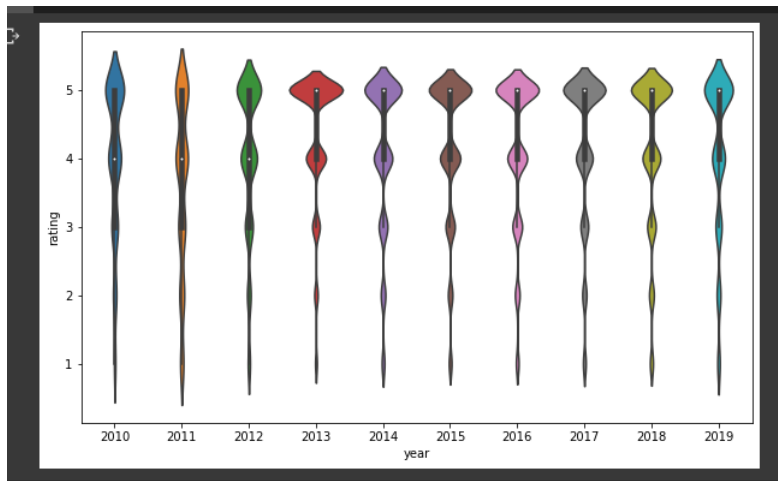


- Abbiamo anche la distribuzione delle category, troviamo che i Dresses e i Tops vanno per la maggiore, dopodiché abbiamo i Bottoms e gli Outerwear. Questi ultimi comparati molto meno rispetto ai dresses e tops.



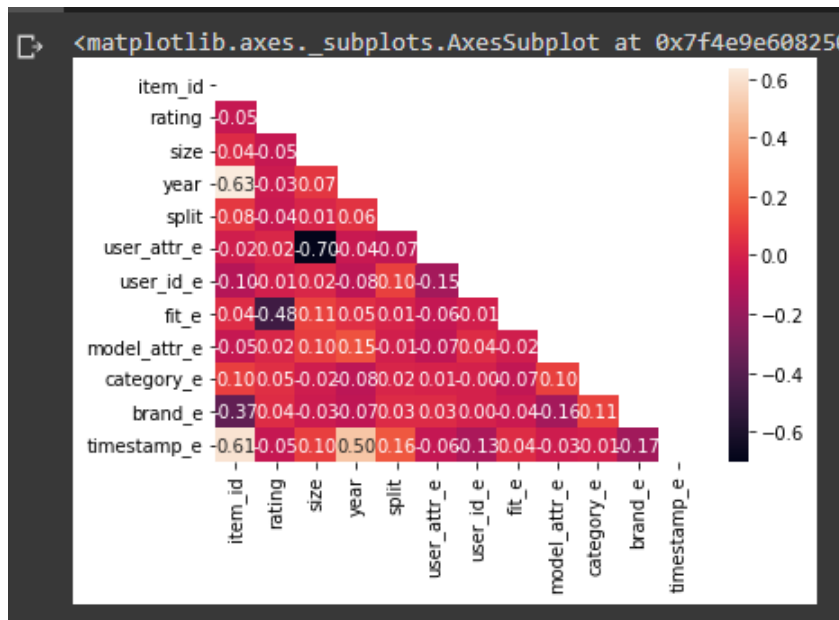
2.1.2 Rating negli anni

Possiamo notare che il rating negli anni é praticamente distribuito in modo uguale, aumentano soltanto il numeri di recensioni dovute al probabile aumento di vendite.



2.1.3 Correlazioni

Iniziamo subito da una heatmap, questa ci sar  utile per capire se esistono diverse correlazioni tra i vari attributi delle recensioni. In questo modo possiamo avere una idea generale iniziale.



Quello che possiamo vedere fin da subito é che non abbiamo forti correlazioni in generale.

Dopo aver avuto una idea generale andiamo nello specifico.

Usiamo un OneHotEncoder per cercare le correlazione nei valori stringa che una colonna puó assumere. Ad esempio il model.attr il quale puó essere Small oppure Small&Large per cui trasformiamo questi valori in colonne.

2.1.4 Correlazioni col rating

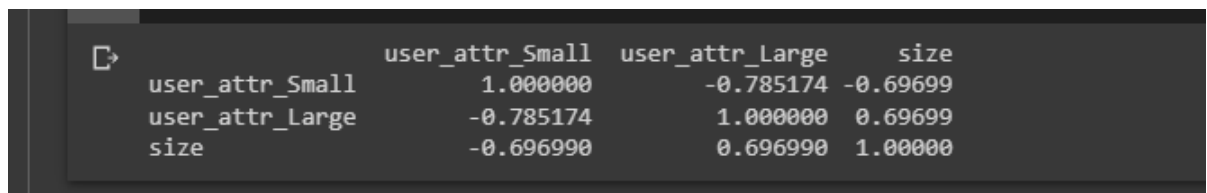
Il rating ha praticamente nessuna correlazione, l'unica piccola nota é questa:

	fit_Just right	rating
fit_Just right	1.000000	0.323832
rating	0.323832	1.000000

Il quale ci indica che se a un utente l'item calza perfetto "Just Right" allora può essere che il suo rating sia maggiore. Possiamo dire che il fatto che a una persona calzi il vestito bene invoglia ad aumentare il rating dato.

2.1.5 Correlazioni Taglie

Nella heatmap vediamo una correlazione negativa molto forte tra `user_attr` e `size`



```
↵ user_attr_Small user_attr_Large size
user_attr_Small 1.000000 -0.785174 -0.69699
user_attr_Large -0.785174 1.000000 0.69699
size -0.696990 0.696990 1.00000
```

The image shows a Jupyter Notebook interface with a dark theme. A code cell contains a correlation matrix for three variables: `user_attr_Small`, `user_attr_Large`, and `size`. The matrix is displayed as a text output, showing the pairwise correlations between these variables. The diagonal elements are all 1.000000, indicating perfect self-correlation. The correlation between `user_attr_Small` and `user_attr_Large` is -0.785174. The correlation between `user_attr_Small` and `size` is -0.69699. The correlation between `user_attr_Large` and `size` is 0.69699.

Possiamo notare che tra uno user Large ci sia un correlazione positiva per quanto riguarda le taglie e viceversa per gli utenti Small. Essa assume un valore di 0.69, ciò comporta a dire che se lo user é di una taglia large allora anche la sua size del vestito comprato aumenta. É una correlazione abbastanza banale ma é importante saperla.

É importante saperla perché potremmo pensare che gli utenti Large comprino quando vedono il `model_attr` Small&Large poiché si sentono rappresentati da esso invece non esiste nessuna correlazione con esso neanche per gli user Small che comprano solo se il `model_attr` é small

	user_attr_Small	user_attr_Large	model_attr_Small	\
user_attr_Small	1.000000	-0.785174	0.018680	
user_attr_Large	-0.785174	1.000000	-0.043957	
model_attr_Small	0.018680	-0.043957	1.000000	
model_attr_Small&Large	-0.018680	0.043957	-1.000000	
	model_attr_Small&Large			
user_attr_Small		-0.018680		
user_attr_Large		0.043957		
model_attr_Small		-1.000000		
model_attr_Small&Large		1.000000		

Non abbiamo altre correlazioni che ci possono aiutare in alcun modo oltre a quella che i timestamp aumentano negli anni come anche gli item id.

3 Step del progetto

Per ogni sezione del progetto, oltre all'analisi esplorativa descritta nella sezione precedente, verranno descritti in modo dettagliato i passaggi nelle varie sottosezioni

3.1 KNN

L'obiettivo della sezione é quello di creare un modello il quale va a predire dato un utente e un item se quest'ultimo potrebbe piacere, in particolare andiamo a predire il rating che un utente potrebbe dare ad un item.

L'algoritmo scelto per fare queste predizioni é il KNN, il suo funzionamento é il seguente:

1. Per prima cosa dobbiamo scegliere la libreria di python che ci fornisce questi algoritmi e tutti i mezzi per utilizzarli. Abbiamo

principalmente due librerie: surprise e sklearn, entrambe hanno algoritmi KNN, nel progetto sono state usate entrambe ma il modello il quale fa predizioni é nella libreria surprise

2. Scegliamo dalla libreria uno dei vari algoritmi KKN, ne abbiamo a disposizione vari: KNNBasic, KNNWithMeans, KNNWithZScore, KNNBaseline. Per questo progetto é stato scelto l'algoritmo KNNBasic.
3. Ora dobbiamo prendere il nostro dataset e renderlo adatto al training e al testing. Abbiamo diversi modi per farlo tra cui:
 - Possiamo imputare tutti i dati mancanti nel nostro dataset con un imputer, sklearn ne offre alcuni come KNNImputer, SimpleImputer. Il primo usa lo stesso principio del KNN per imputare dati mancanti nel dataset, il secondo imputa i dati con la media della colonna.
 - Il modo da me usato é stato eliminare tutte le righe con valori mancanti in modo tale che il modello potesse essere addestrato su recensioni complete. Così facendo però perdiamo molti dati infatti su 100k di righe ne andiamo ad ottenere 15k che é solo in 15%. Possiamo migliorare in verità il numero totale di righe su cui addestrare il modello a circa 70k ma su colab la computazione in fase di training non riesce poiché la ram viene riempita e si blocca il tutto, il modo per ottenere queste 70k di righe era dropare la colonna brand essa infatti presenta molte missing values e poi dropare le righe che hanno altri valori nulli al loro interno.

```

[99893 rows x 12 columns]
item_id      0
user_id      1
rating       0
timestamp    0
size        21760
fit         18506
user_attr    8367
model_attr   0
category     0
brand       73980
year         0
split        0

```

(Somma le missing values per ogni colonna)

4. Una volta ottenuto un dataset su cui addestrare il modello possiamo iniziare a dividere il dataset in due parti, una parte é il training set sui cui il nostro algoritmo KNN si addestra ed é l'80% del dataset e l'altro é il testset sui cui il nostro algoritmo farà dei predict di rating e vedrà quanto sono accurati, é il 20% del dataset. Questa divisione la fa una funzione chiamata `train_test_split` integrata nella libreria `surprise`

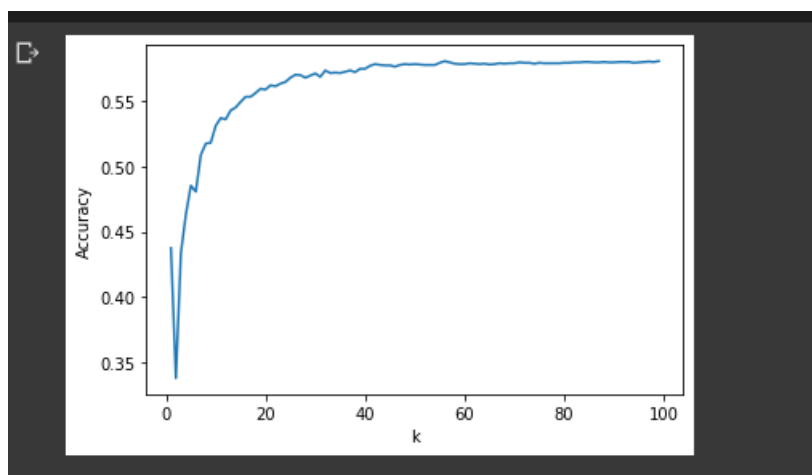
Prima di far andare il nostro modello ci serve ottimizzare dei parametri tra cui :

- Il numero di K neighbors che vogliamo andare a prendere
- La distanza che vogliamo usare per l'algoritmo, possiamo scegliere tra cosine e pearson. La cosine si basa sulla similitudine tra due vettori effettuata calcolando il coseno tra di loro. La pearson é definita come la covarianza tra due variabili divisa per il prodotto delle deviazioni standard delle due variabili.

- Infine se basare il nostro modello User Based o Item Based la differenza sta nel fatto che prendiamo i neighbors negli users o negli items.

3.1.1 Elbow Method

Un metodo per trovare il miglior k neighbors é appunto l'elbow method, in pratica iteriamo a partire da k uguale ad 1 fino a quanto vogliamo, per esempio 100, e vediamo come l'accuracy come é all'aumentare di questo valore



(Grafico Elbow Method)

Come possiamo notare all'aumentare di k aumenta l'accuracy. La curva però é praticamente logaritmica quindi all'incirca dopo il 40 essa aumenta di un numero insignificante, dobbiamo infatti considerare che prendere un k molto alto vuol dire che il nostro algoritmo andrà a cercare tanti utenti (oppure item) simili a quello preso in considerazione aumentando il carico computazionale. Questo é un ottimo metodo per visualizzare al variare di parametri dai valori continui come k come varia anche l'accuracy.

Anche se ora abbiamo una idea di come scegliere k, soprattutto da

che valore partire a sceglierlo dobbiamo anche considerare che distanza usare e se fare un modello user based o item based. Andiamo quindi ad usare un GridSearch.

3.1.2 GridSearch

Il Gridsearch é una tecnica che ci permette di prendere il nostro dataset e lo divide appunto in trainset e testset facendo poi cross-validation, cioè la divisione trainset e testset viene fatto un numero di volte specificato nella funzione per poi vedere per ogni volta l'rmse calcolato. Ogni volta che viene effettuata la divisione train/test logicamente essi cambiano non sono mai uguale. A questo punto abbiamo infatti il miglior k, la distanza da usare tra quelle che gli abbiamo proposto e se avere un modello user based o item based.

```
Done computing similarity matrix.  
Best RMSE = 1.0217  
Best configuration = {'k': 70, 'sim_options': {'name': 'cosine', 'user_based': True}}  
ALGO KNN Rating
```

(Risultato GridSearch)

Come possiamo vedere il nostro miglior RMSE con la migliore configurazione e circa 1, di seguito abbiamo appunto i dati della miglior configurazione, $k = 70$, distanza = cosine, user based = true. Notiamo che se il gridsearch verrà eseguito molteplici volte potremmo avere risultati leggermente diversi per qualche millesimo per quanto riguarda RMSE e potrebbe anche cambiare il k, il quale però sarà sempre maggiore di 40.

Una volta che abbiamo eseguito il nostro gridsearch possiamo passare ad fare il nostro modello con l'algoritmo KNNBasic avendo come parametri i migliori consigliati dal gridsearch.

3.2 Costruzione Modello

Possiamo ora passare a costruire il modello. Abbiamo suddiviso il nostro dataset in train e test quindi ora possiamo fare il training e il testing del nostro modello e vedere che RMSE abbiamo, ricordiamo che stiamo cercando di predire rating che va da 1 a 5.

```
# Initialize the algorithm
from surprise.model_selection import train_test_split

sim_options = {
    "name": "cosine",
    "user_based": True, # compute similarities between items
}

algor = KNNBasic(k=70, sim_options=sim_options)
trainset, testset = train_test_split(data, test_size=0.2)

# Train and test the algorithm
algor.fit(trainset)
predictions = algor.test(testset)
# Compute metrics
mse = accuracy.mse(predictions)
rmse = accuracy.rmse(predictions)

Computing the cosine similarity matrix...
Done computing similarity matrix.
MSE: 1.0187
RMSE: 1.0093
```

(Risultato KNNBasic)

Possiamo notare che sia RMSE E MSE sono circa 1, ciò vuol dire che il rating predetto in media é sbagliato di più o meno 1. É difficile stabilire se esso é valore accettabile o meno poiché dipende anche da quanto grande é il nostro dataset. Nel nostro caso infatti i dati sono abbastanza pochi.

3.3 Filling della matrice di rating

Una volta addestrato il nostro modello possiamo usarlo per creare una matrice di rating dove abbiamo 3 colonne:

- userId
- itemId
- rating predetto

Vogliamo quindi per ogni user predire che voto darebbe a tutti gli item, in totale quindi abbiamo circa 44k di user e 1k di item, ci dovrebbero venire all'incirca 44M di righe. Sfortunatamente colab non permette di fare tutto ciò in tempi ragionevoli, infatti facendo un test considerando appunto tutti gli item e user dopo 2 ore la matrice non era ancora stata completata. Ho deciso quindi di considerare un sottoinsieme di utenti precisamente 1% (447 users) di essi poiché provando a considerare il 10% anche lì non computava in tempi ragionevoli. Gli item invece sono stati considerati tutti.

```
# Create a new dataframe for all combinations of user_id and item_id
import random
users = df_r['user_id_e'].unique()
users = random.sample(list(users), 447)
items = df_r['item_id'].unique()

# Initialize the predictions dataframe
df_predictions = pd.DataFrame(columns=['user_id', 'item_id', 'rating'])

# Predict the ratings
for user in users:
    for item in items:
        pred = algor.predict(user, item)
        df_predictions = df_predictions.append({'user_id': user, 'item_id': item, 'rating': pred.est}, ignore_index=True)

# Reset the index of the predictions dataframe
df_predictions = df_predictions.reset_index(drop=True)

[ ] print(df_predictions)

   user_id  item_id  rating
0    39841.0  153445.0  4.306708
1    39841.0  129113.0  4.306708
2    39841.0  152990.0  4.306708
3    39841.0  152868.0  4.306708
4    39841.0  138414.0  4.306708
...      ...      ...      ...
455935  36523.0  32134.0  4.306708
455936  36523.0  142398.0  4.306708
455937  36523.0  154762.0  4.306708
455938  36523.0  152544.0  4.306708
455939  36523.0  50405.0  4.306708

[455940 rows x 3 columns]
```

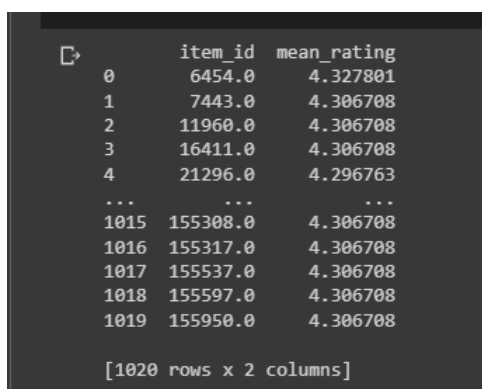
(Codice Matrice di Rating)

In modo particolare i sample degli user sono presi in modo randomico dai 44k. Il motivo per la quale gli item invece sono stati presi tutti nessuno escluso é perché il punto seguente chiede di raccomandare degli item ai vari utenti quindi eliminando item si eliminano dal raccomandarli.

3.4 Items da consigliare

Una volta costruito il dataframe `userId, itemId, rating` stimato possiamo andare a consigliare per ogni utente (44k totali) 10 item. Possiamo usare diverse metriche per consigliare un item, quella piú semplice é il rating.

Prima di tutto ho preso il dataframe con tutti i rating predetti per tutti gli item dati dai 447 utenti considerati, li ho raggruppati e di tutti i rating dati a un item ho fatto la media cosí facendo ho costruito un dataframe `item, mean_rating`.



	item_id	mean_rating
0	6454.0	4.327801
1	7443.0	4.306708
2	11960.0	4.306708
3	16411.0	4.306708
4	21296.0	4.296763
...
1015	155308.0	4.306708
1016	155317.0	4.306708
1017	155537.0	4.306708
1018	155597.0	4.306708
1019	155950.0	4.306708

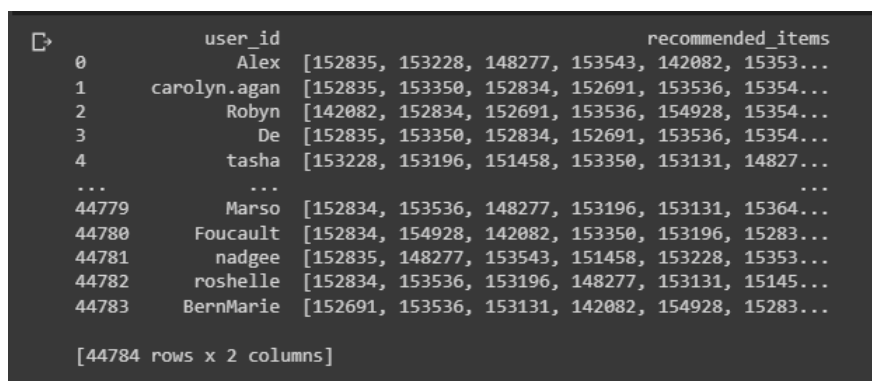
[1020 rows x 2 columns]

(Dataframe `ItemId`, Media dei rating di quell'item)

Dopodiché ho inizializzato una lista dove andranno messe le nostre raccomandazioni, dopo questa lista verrà convertita in un dataframe. Ho preso tutti gli unique user nel mio dataset circa 1k e per ognuno di questi ho fatto le seguenti azioni:

- Prima di tutto ho voluto creare varietà nelle mie raccomandazioni per cui ho preso tutti gli item che il nostro user avesse votato e li ho sottratti alla lista degli item, in modo tale da ottenere items che il nostro user non avesse mai recensito.
- Una volta avuto la lista di items che il nostro utente non avesse recensito li ho ordinati in modo decrescente, quindi dal rating più alto al più basso e preso i primi 10
- Appendo il risultato cioè un array di 10 elementi alla lista di raccomandazioni

Infine una volta fatto questo lavoro per ogni user creo il mio dataframe con id e la lista di item raccomandati



```

0      Alex [152835, 153228, 148277, 153543, 142082, 15353...
1  carolyn.agan [152835, 153350, 152834, 152691, 153536, 15354...
2      Robyn [142082, 152834, 152691, 153536, 154928, 15354...
3        De [152835, 153350, 152834, 152691, 153536, 15354...
4     tasha [153228, 153196, 151458, 153350, 153131, 14827...
...      ...
44779   Marso [152834, 153536, 148277, 153196, 153131, 15364...
44780 Foucault [152834, 154928, 142082, 153350, 153196, 15283...
44781   nadgee [152835, 148277, 153543, 151458, 153228, 15353...
44782 roshelle [152834, 153536, 153196, 148277, 153131, 15145...
44783 BernMarie [152691, 153536, 153131, 142082, 154928, 15283...

[44784 rows x 2 columns]

```

(Dataframe User, item raccomandati)

L'id degli user sono presi direttamente dal dataset iniziale per cui sono di tipo stringa.

3.5 K-Means

In questo step del progetto viene richiesto di suddividere i nostri utenti in cluster, cioè partizioni, in base alle loro preferenze cioè in base al rat-

ing che danno agli item. Per fare ciò usiamo un algoritmo di clustering K-Means.

L'algoritmo K-means è un algoritmo di analisi dei gruppi partizionale che permette di suddividere un insieme di oggetti in k gruppi sulla base dei loro attributi. Una libreria in python che ci permette di usare questo algoritmo è sklearn, il K-Means della libreria sklearn non ci permette di cambiare la distanza che vogliamo usare per fare clustering, di default usa la distanza euclidea. Invece il K-Means della libreria nltk ci permette di selezionare che distanza vogliamo utilizzare. Io ho provato entrambi gli algoritmi per vedere quale dei due avesse il miglior siluette score. È venuto fuori che il k-means della libreria sklearn ha un siluette score migliore.

Il siluette score è un valore che va da -1 a 1. un valore molto vicino a 1 indica un buon clustering, cioè: gli elementi dello stesso cluster sono molto vicini e elementi di cluster differenti sono molto distanti. In generale una buona siluette score è maggiore di 0.6.

PREMESSA: In teoria per il clustering avrei dovuto prendere tutti gli user e tutti gli item con i rispettivi voti, alcuni di essi già presenti nel dataset e altri no. Quelli non presenti avrei potuto:

- Fare un predict tramite KNN del rating user, item
- Fare la media dei voti dati a quell'item e metterlo come rating
- Mettere uno 0 dove non ho un rating

Sotto consiglio imputare 0 dove non avessi rating era il miglior modo per fare ciò poiché usando una cosine similarity non avrei avuto problemi.

Il **PROBLEMA** però sta nel fatto che in totale avrei avuto 44 milioni di righe poiché il dataset ha 44k di utenti e 1k di item. Era

impossibile computare tutto ciò, per cui ho iniziato con un sottoinsieme di utenti e item, precisamente 25k di utenti e 300 item, dopo 5 ore non é riuscito a costruire il dataframe user, item, rating. Anche riducendo ulteriormente gli utenti a 10k non riusciva a computare in meno di 5 ore.

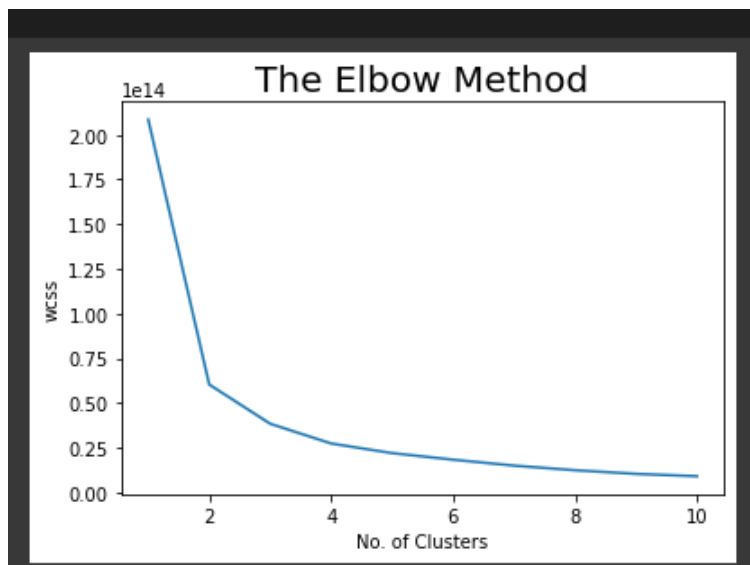
SOLUZIONE: Dato che non aveva senso ridurre ancora utenti e item mi sono basato sul dataset iniziale dove avevo già tutti i valori di rating per cui non dovevo imputarne alcuni. Ho quindi fillato tutte le colonne dove avessi missing values con degli 0 e fatto clustering su di esso. Le uniche colonne che ho droppato sono brand, timestamp, year poiché non volevo che i cluster si basassero su queste correlazioni.

Ho testato anche usando solo le colonne user, item, rating e il risultato non cambia.

3.5.1 Elbow Method

Per prima cosa dobbiamo ottimizzare i parametri del nostro cluster in modo particolare quanti cluster vogliamo andare a creare. Il modo più semplice per fare ciò é usare come per il KNNBasic l'elbow method. Il quale tramite il wcss ci fa capire il numero ottimale di cluster.

Per WCSS (Within-Cluster Sum of Squares) intendiamo la somma dei quadrati delle distanze tra ogni punto di un cluster e il centroide di quel cluster. In altre parole, WCSS misura quanto i punti in un cluster si allontanano dal centroide del cluster.



(Grafico Elbow Method)

Notiamo subito che dopo il 4 il wcss non si abbassa di molto per cui possiamo usare il 4 come numero di cluster, nessuno ci vieta di usare 8 o 10 ad esempio ma il miglioramento sarebbe minimo e la computazione più lunga.

3.5.2 K-Means nltk

Ho testato questo algoritmo con la distanza cosine ma in generale i risultati del cluster erano confusionari a livello visivo, cioè facendo un grafico i cluster erano molto sparpagliati. Il silhouette score veniva di circa 0. Ho quindi optato per usare l'algoritmo k-means di sklearn.

3.5.3 K-Means sklearn

- Per prima cosa creo un dataframe apposta per il clustering, copiando il dataframe iniziale trasformo tutte le variabili come fit, user_attr in numerici, poiché il mio dataset ha correlazioni molto basse lascio tutte le colonne a parte:

- il brand poiché aveva troppe missing values, aveva poco senso lasciarlo anche perché non c'erano correlazioni utili al clustering
 - year e timestamp, il motivo é che logicamente all'aumentare di uno dei due aumenta anche l'altro, non volevo che i miei cluster si basassero su questi due fattori
- Come seconda cosa inizializzo il mio algoritmo di clustering, i parametri più importanti sono il numero di cluster che nel nostro caso sono 4 e il modo in cui scegliamo i centroidi del cluster. L'algoritmo ci propone di sceglierli a caso e quindi in modo random oppure nel modo "k-means++", secondo quanto riportato nella documentazione si basa sul campionamento basato su una distribuzione di probabilità empirica del contributo dei punti all'inerzia complessiva. In generale usando quest'ultimo modo i risultati in media sono migliori.
 - Una volta addestrato l'algoritmo per ogni tripla user, item, rating associa un cluster, possiamo aggiungere la colonna al dataframe avendo come risultato:

	item_id	rating	size	split	user_attr_e	fit_e	model_attr_e	\
0	7443	4	0.0	0	1	5	0	
1	7443	3	0.0	0	2	5	0	
2	7443	4	0.0	0	1	5	0	
3	7443	4	0.0	0	2	5	0	
4	7443	4	0.0	0	1	5	0	
...	
99888	154797	5	6.0	0	0	0	1	
99889	77949	4	4.0	2	1	2	1	
99890	67194	5	0.0	2	1	0	1	
99891	71607	3	0.0	2	1	2	1	
99892	119732	3	0.0	2	1	0	0	
	category_e	user_id_e	label					
0	1	309	2					
1	1	13009	2					
2	1	5534	2					
3	1	1716	2					
4	1	42071	2					
...					
99888	1	843	1					
99889	0	5706	0					
99890	1	2808	0					
99891	2	9022	0					
99892	1	38602	3					

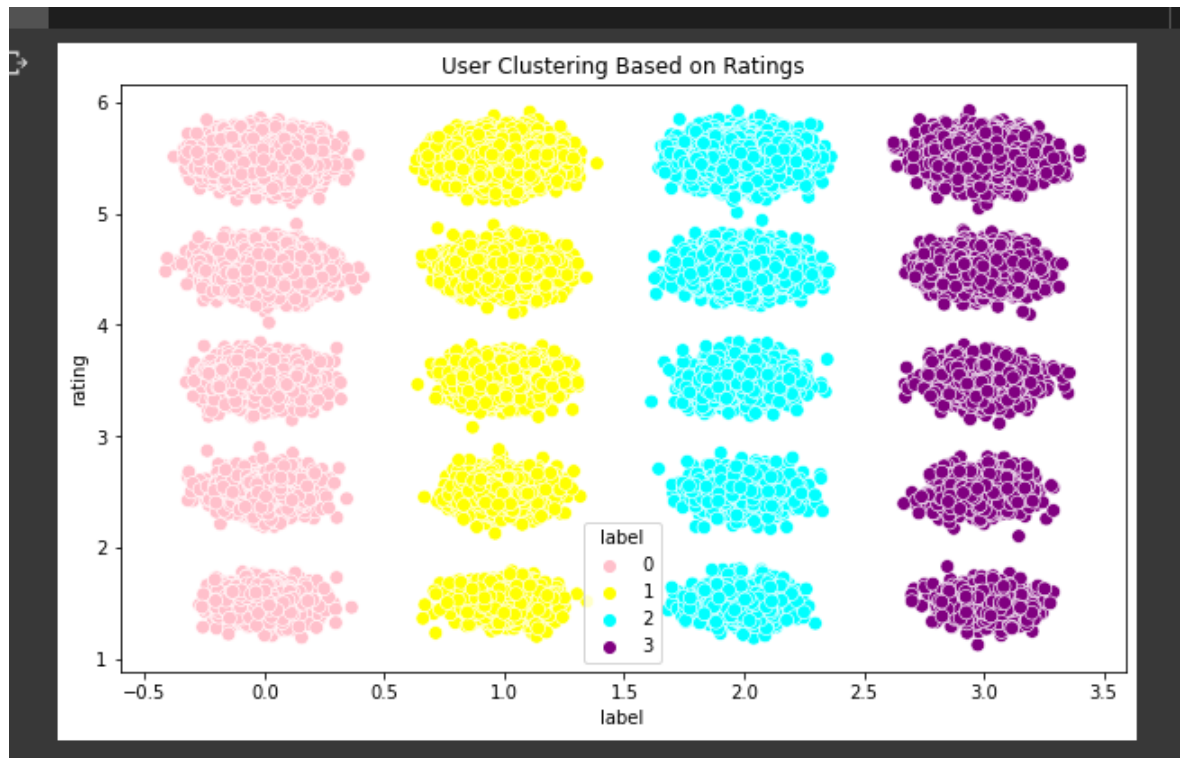
(df_means con l'aggiunta dei labels)

- Possiamo ora valutare il suo siluette score

```
Silhouette score = 0.4543181075836887
```

Il risultato é di 0.45 il che ci indica che non é un ottimo risultato ma un risultato decente.

- Andando a visualizzare la distribuzione dei rating nei valori label usando un grafico otteniamo questo



Notiamo appunto che nel nostro grafico in ogni label la distribuzione dei rating é molto uniforme.

3.5.4 K-Means su Rating Predetti

Ho voluto testare se il silhouette score e il grafico venissero meglio. Logicamente questa matrice di rating predetti si basa sulla matrice costruita in precedenza per cui é un sottoinsieme di user e item.

- Per prima cosa ho copiato la matrice in una nuova in modo tale da poterci lavorare sopra in modo da non danneggiare la matrice originale, perché il tempo di computazione per farla é abbastanza lungo.

- Il procedimento eseguito é identico al precedente con anche lo stesso numero di cluster e il modo in cui vengono inizializzati i cluster. Otteniamo quindi questo dataframe

```

✎

```

	user_id	item_id	rating	label
0	9019.0	7443.0	4.309867	1
1	9019.0	11960.0	4.309867	1
2	9019.0	16411.0	4.309867	1
3	9019.0	21296.0	4.266667	1
4	9019.0	22563.0	4.309867	1
...
455935	23437.0	153853.0	4.309867	3
455936	23437.0	153866.0	4.309867	3
455937	23437.0	149062.0	4.309867	3
455938	23437.0	54062.0	4.309867	1
455939	23437.0	153228.0	4.309867	3

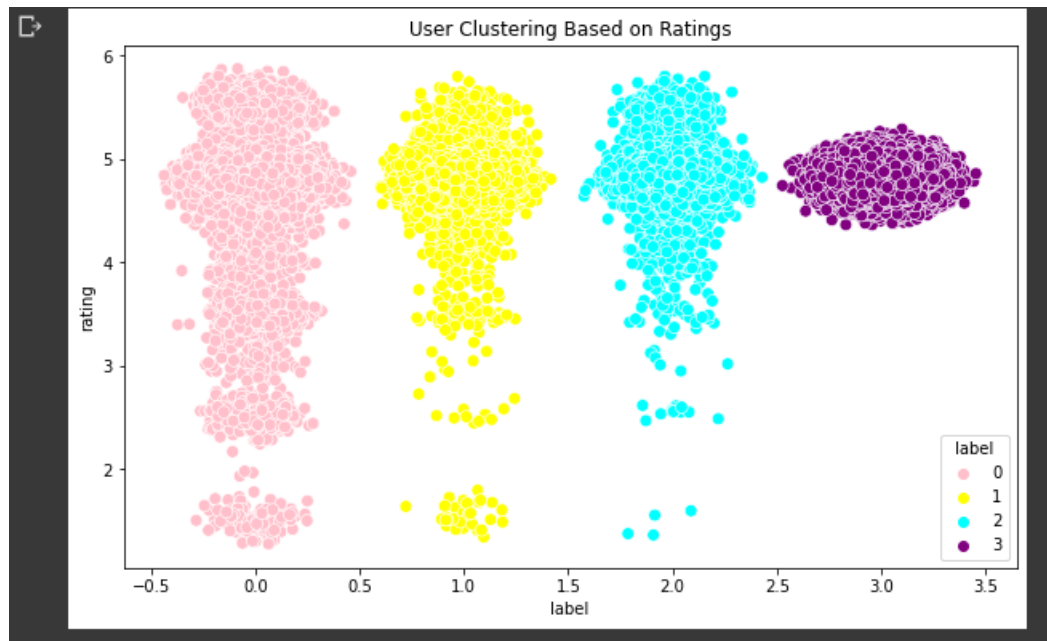
[455940 rows x 4 columns]

- Dopodiché ho calcolato il silhouette score il quale é migliorato rispetto alla matrice di rating passata

```
Silhouette score = 0.5439173022340751
```

Il silhouette score é di 0.54 che si avvicina molto al valore 0.6 dove appunto sarebbe stato un buon cluster. É un buon risultato

- Infine ho fatto un grafico uguale all'altro e cioé con i label e il rating per vedere se ogni cluster avesse differenze



Da questo clustering già possiamo vedere un paio di cose in più tra cui il fatto che il cluster 3 (viola) sia quello dove sono raggruppati rating molto alti e il cluster 0 (rosa) invece è quello con il maggior numero di elementi. Anche qui però la distribuzione dei rating per i primi 3 cluster è distribuita in modo abbastanza omogeneo.

3.6 Matrix Factorization

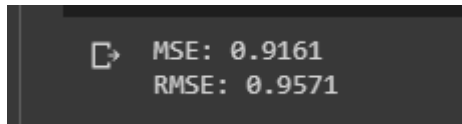
Questa parte di progetto prevede di addestrare un altro algoritmo della famiglia "Collaborative Filtering" come il KNN. Questo però usa una filosofia completamente diversa dal KNN, si basa infatti sulla fattorizzazione matriciale.

La libreria surprise ci permette di utilizzare l'algoritmo SVD il quale fa parte degli algoritmi matrix factorization. Utilizza la fattorizzazione matriciale per ridurre la complessità del problema e per generare una

matrice approssimata che può essere utilizzata per effettuare previsioni di valutazione.

3.6.1 Algoritmo

- Come per il knn dobbiamo dividere il nostro dataset in trainingset e testset. usiamo sempre la libreria surprise.
- Ora inizializziamo il nostro algoritmo SVD. In questo non dobbiamo fare nessuno studio sui parametri che andremmo ad utilizzare.
- Andiamo poi ad addestrare l'algoritmo sul trainset e a testarlo sul test
- Infine andiamo a calcolare RMSE e MSE

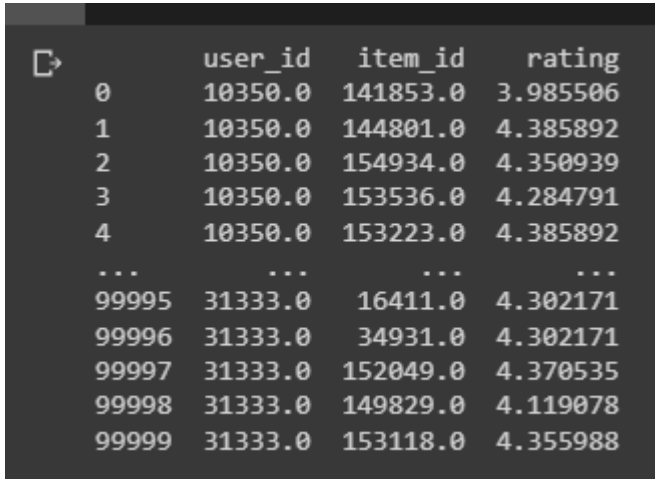


```
➤ MSE: 0.9161
  RMSE: 0.9571
```

Possiamo notare di come rispetto al KNN non abbiamo una grande riduzione delle errore che rimane sempre intorno a 1

3.6.2 Predizioni

Infine ho usato lo stesso numero di user e item della matrice di rating predetto col knn per crearne una nuova usando la predizione dell'algoritmo SVD.

A terminal window with a dark background and light gray text. It displays a table with four columns: an index, user_id, item_id, and rating. The data is as follows:

	user_id	item_id	rating
0	10350.0	141853.0	3.985506
1	10350.0	144801.0	4.385892
2	10350.0	154934.0	4.350939
3	10350.0	153536.0	4.284791
4	10350.0	153223.0	4.385892
...
99995	31333.0	16411.0	4.302171
99996	31333.0	34931.0	4.302171
99997	31333.0	152049.0	4.370535
99998	31333.0	149829.0	4.119078
99999	31333.0	153118.0	4.355988

4 Conclusioni

Un breve riassunto di tutti i risultati a cui si é arrivati per ogni step

4.1 Analisi descrittiva

Dai grafici si é notato che i rating da 1 a 4 sono distribuiti in modo abbastanza omogeneo mentre i rating a 5 sono maggiori rispetto al numero degli altri. Un'altra analisi é il fatto che la maggior parte degli utenti hanno taglie Small.

4.2 Correlazioni

Le correlazioni sono molto basse, le piú importanti sono:

- Il rating con il fatto che il fit just right e cioé che l'item calzi giusto vada ad aumentare il rating
- Un'altra correlazione la troviamo nel fatto che se uno user ha una taglia Large essa vada ad aumentare la taglia

- Infine non troviamo correlazione nel fatto che gli utenti comprino item se la modella é della loro stessa taglia

4.3 Algoritmo di Predizione

Sia l'algoritmo KNN che il SVD hanno un RMSE di circa 1, anche se SVD é lievemente migliore. Possiamo concludere con fatto che in generale usando un algoritmo piú complesso come l'SVD non abbiamo migliorie notevoli.

4.4 Clustering

I risultati del clustering non sono ottimi ma neanche pessimi, nel complesso sono decenti. I silhouette score sulla matrice di rating é di 0.45 e su quella di rating predetto é 0.54. Soprattutto su quest'ultima il clustering oltre ad avere un buon silhouette score anche la rappresentazione grafica di come i nostri cluster sono suddivisi é migliore.