2023-04-05

**Πανεπιστήμιο Πειραιώς**

**Προηγμένα Συστήματα Πληροφορικής – Ανάπτυξη Λογισμικού και Τεχνητής Νοημοσύνης**

Τελική Εργασία για το Μάθημα

**Αναγνώριση Προτύπων και Μηχανική Μάθηση**

των

**Αποστόλου Αθανάσιου (mpsp2203@unipi.gr)**

**Μπιρμπάκου Γεώργιου (mpsp2220@unipi.gr)**

**Ευαγγέλου Αλέξανδρου (mpsp2210@unipi.gr)**

**Θέμα: Αλγόριθμοι Σύστασης με Χρήση Τεχνητών Νευρωνικών Δικτύων και Τεχνικών Ομαδοποίησης Δεδομένων**

# Εισαγωγή

Στα πλαίσια της τελικής εργασίας του μαθήματος “Αναγνώριση Προτύπων και Μηχανική Μάθηση” αναπτύξαμε κώδικα όπου διαβάζει ένα αρχικό Dataset (το οποίο βρίσκεται στην διεύθυνση <https://www.dropbox.com/s/6bcfscyexabgy0w/Dataset.npy?dl=0>), αναλύει τα δεδομένα του και εξάγει τα επιθυμητά αποτελέσματα.

Ο κώδικας έχει γραφεί σε γλώσσα Python ως ένα jupyter notebook και βρίσκεται στο αρχείο “**IMDB.ipynb**”.

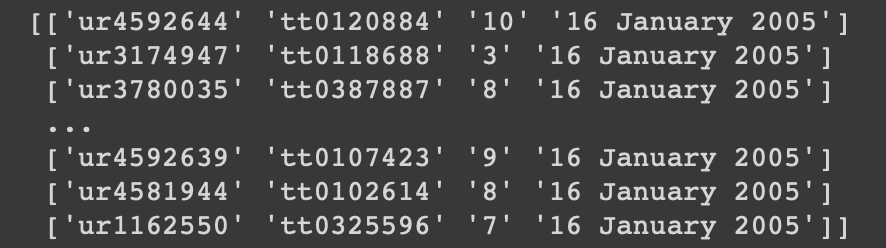
Ο κώδικας και αυτή η αναφορά βρίσκονται επίσης στο git repository <https://github.com/ThanosApostolou/aics-pattern-recognition>

# Προ-επεξεργασία Δεδομένων

### Να βρείτε το σύνολο των μοναδικών χρηστών και το σύνολο των μοναδικών αντικειμένων .

Αρχικά ξεκινάμε φορτώνοντας τα δεδομένα που έχουμε σε ένα numpy array.

Ελέγχοντας το αρχικό dataset που μας δίνεται, βλέπουμε ότι είναι της ακόλουθης μορφής:

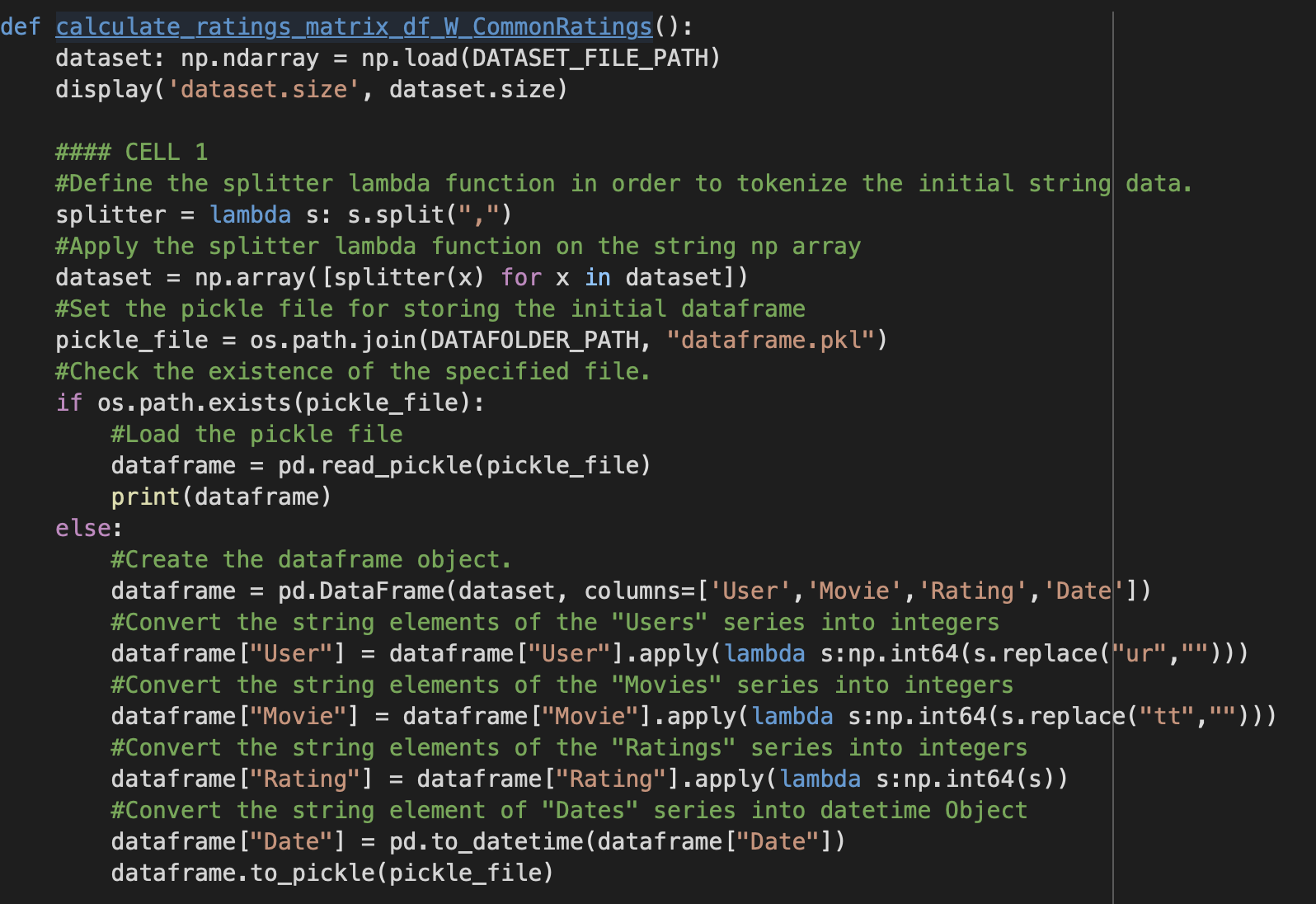


Όπου:

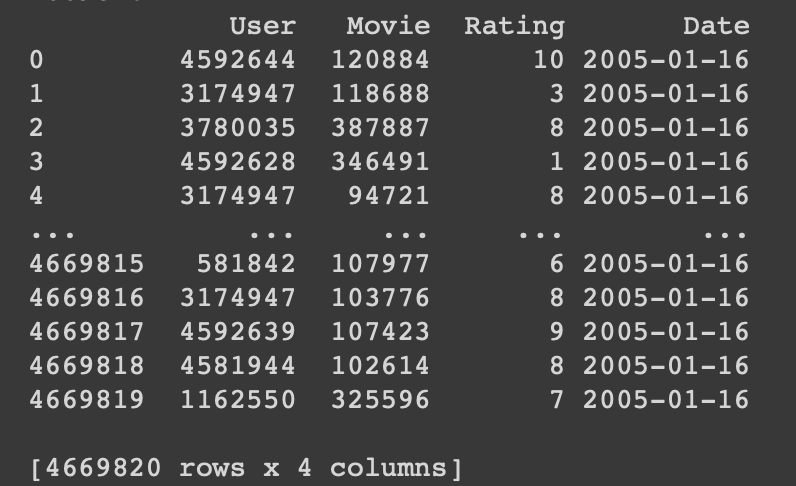
* urxxxxxx ένα πρόθεμα ur ακολουθούμενο από το ID κάθε χρήστη
* ttxxxxxx ένα πρόθεμα tt ακολουθούμενο από το ID κάθε ταινίας
* Η αξιολόγηση της ταινίας από το χρήστη
* Η ημερομηνία της αξιολόγησης

Επιθυμούμε στη συνέχεια να δημιουργήσουμε ένα καινούριο Pandas Dataframe από τα δεδομένα που μας δίνονται το οποίο θα έχει τη μορφή:

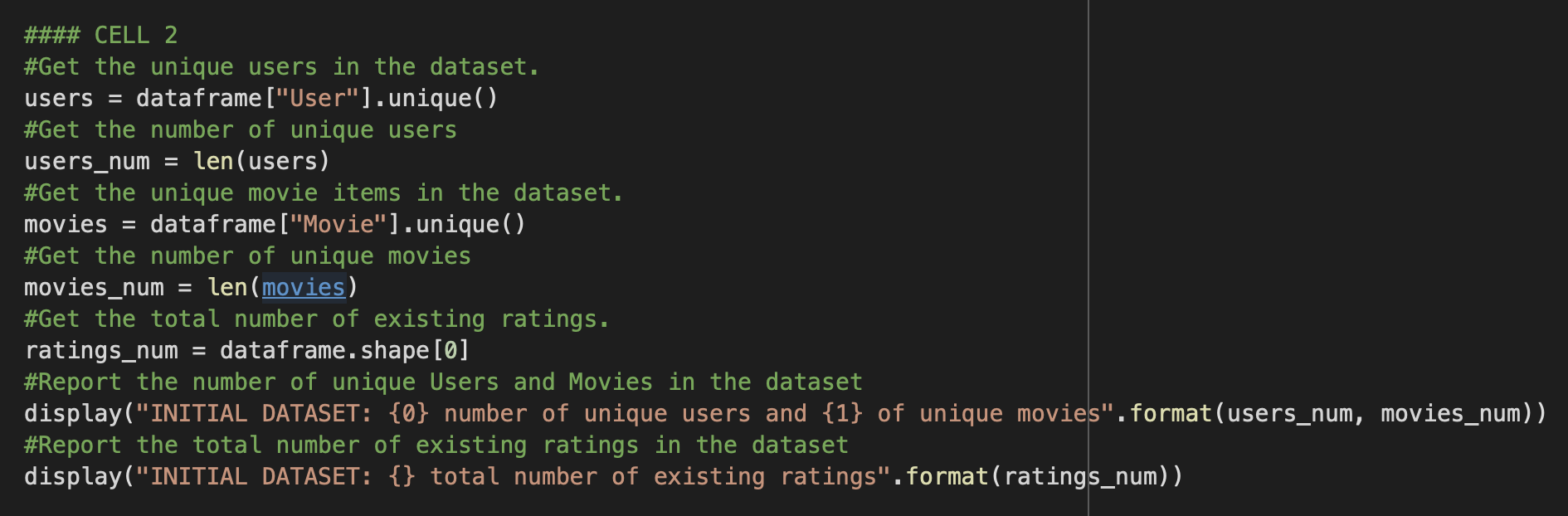
|  |  |  |  |
| --- | --- | --- | --- |
| User | Movie | Rating | Date |



Το αποτέλεσμα μετά τη δημιουργία του dataframe μας είναι το ακόλουθο:



Από αυτό το dataframe βρίσκουμε το σύνολο των μοναδικών χρηστών U και το σύνολο των μοναδικών ταινιών Ι:



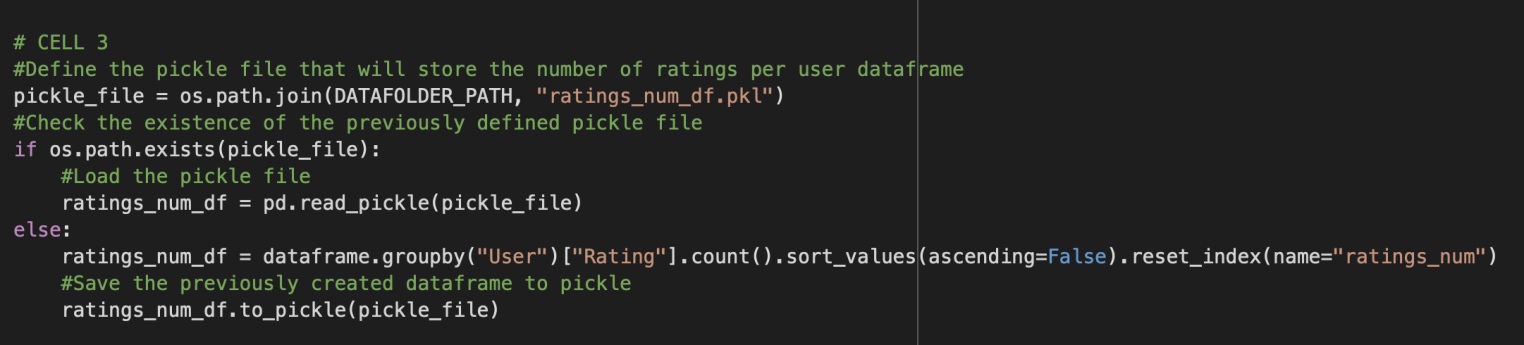
Όπου λαμβάνουμε τα αποτελέσματα:

**INITIAL DATASET: 1499238 number of unique users and 351109 of unique movies**

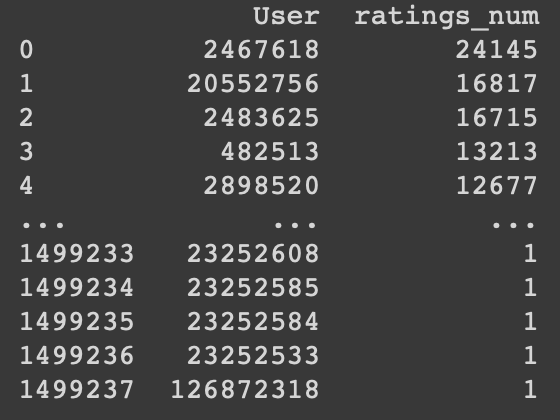
**INITIAL DATASET: 4669820 total number of existing ratings**

### Θεωρείστε την συνάρτηση (όπου το δυναμοσύνολο του ) η οποία επιστρέφει το σύνολο των αντικειμένων που αξιολογήθηκαν από τον χρήστη . Μπορούμε να γράψουμε, επομένως, ότι . Να περιορίσετε τα σύνολα των μοναδικών χρηστών και μοναδικών αντικειμένων στα αντίστοιχα σύνολα και έτσι ώστε όπου και ο ελάχιστος απαιτούμενος και ο μέγιστος επιτρεπτός αριθμός αξιολογήσεων ανά χρήστη. Θεωρήστε προφανώς ότι και *.*

Θέλουμε αρχικά να δημιουργήσουμε τη συνάρτηση Φ η οποία για κάθε χρήστη επιστρέφει το σύνολο των αντικειμένων Ι που αυτός έχει αξιολογήσει.

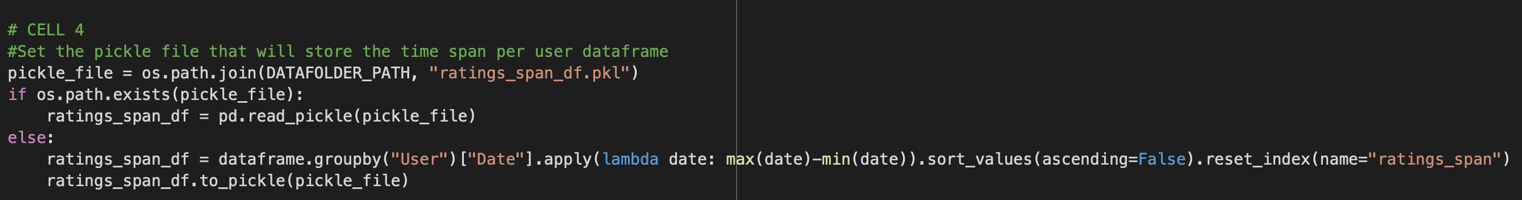


Τα αποτελέσματα αυτής της συνάρτησης είναι τα ακόλουθα:



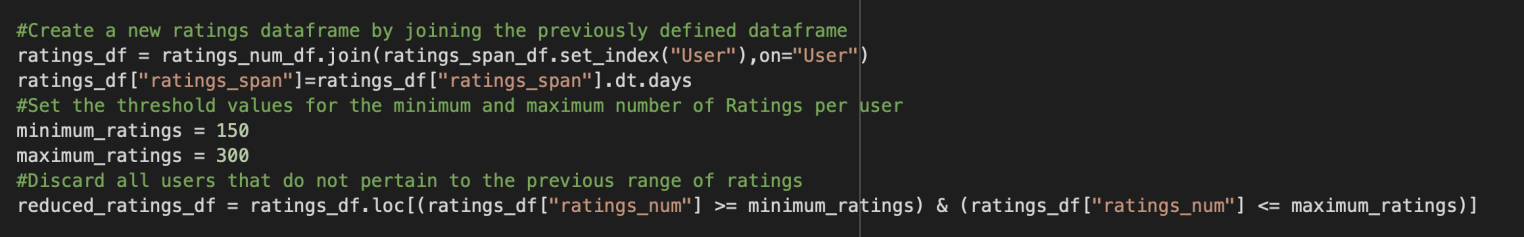
Τα σύνολα αυτά τα αποθηκεύουμε σε ένα pickle file.

Στη συνέχεια, δημιουργούμε το dataframe που περιέχει το χρονικό εύρος των αξιολογήσεων κάθε χρήστη σε ημέρες (θα χρησιμοποιηθεί για να δημιουργήσουμε το ιστόγραμμα συχνοτήτων που απαιτείται σε επόμενο ερώτημα).



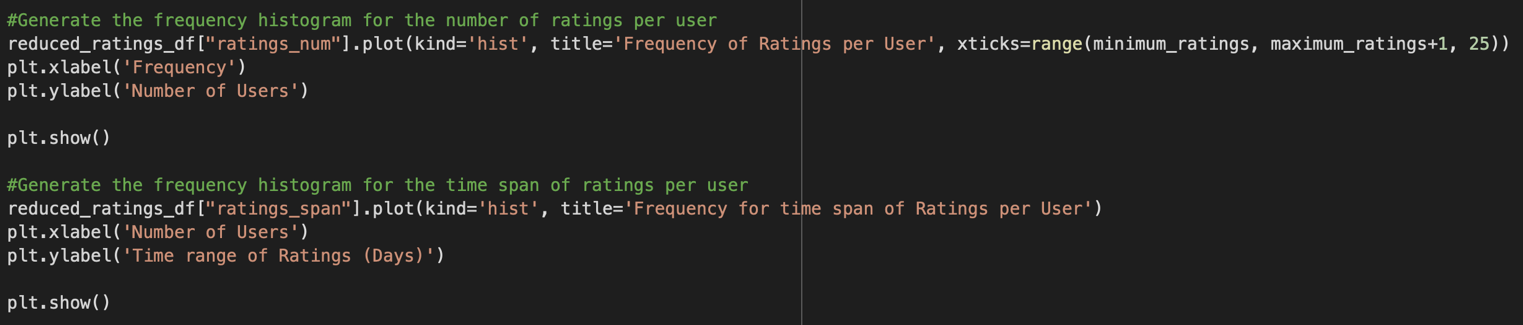
Χρησιμοποιώντας τώρα αυτά τα δύο dataframe, δημιουργούμε ένα καινούριο, το οποίο περιέχει τόσο τον αριθμό των αξιολογήσεων κάθε χρήστη αλλά και το χρονικό εύρος των αξιολογήσεών του.

Περιορίζουμε το συγκεκριμένο dataframe έτσι ώστε να περιέχει μόνο τους χρήστες των οποίων ο αριθμός των αξιολογήσεων είναι μεταξύ των 150 και 300.

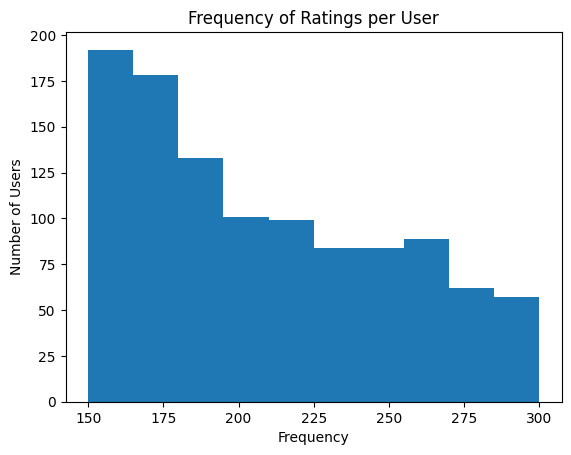


### Να δημιουργήσετε και να αναπαραστήσετε γραφικά τα ιστογράμματα συχνοτήτων για το πλήθος αλλά και για το χρονικό εύρος των αξιολογήσεων του κάθε χρήστη.

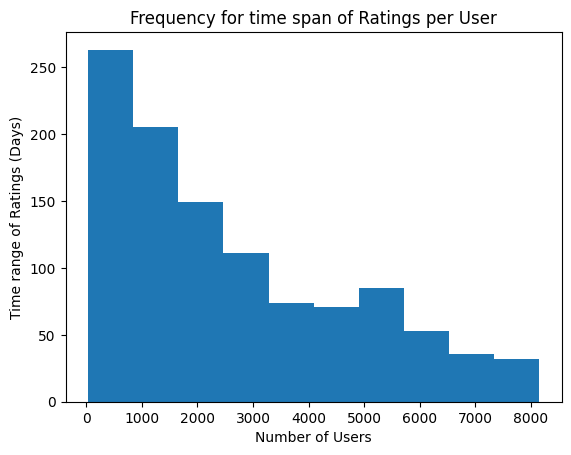
Από τα dataframe που δημιουργήσαμε στο προηγούμενο ερώτημα, δημιουργούμε τα ιστογράμματα των συχνοτήτων για το πλήθος και το χρονικό εύρος των αξιολογήσεων κάθε χρήστη χρησιμοποιώντας τη βιβλιοθήκη matplotlib.



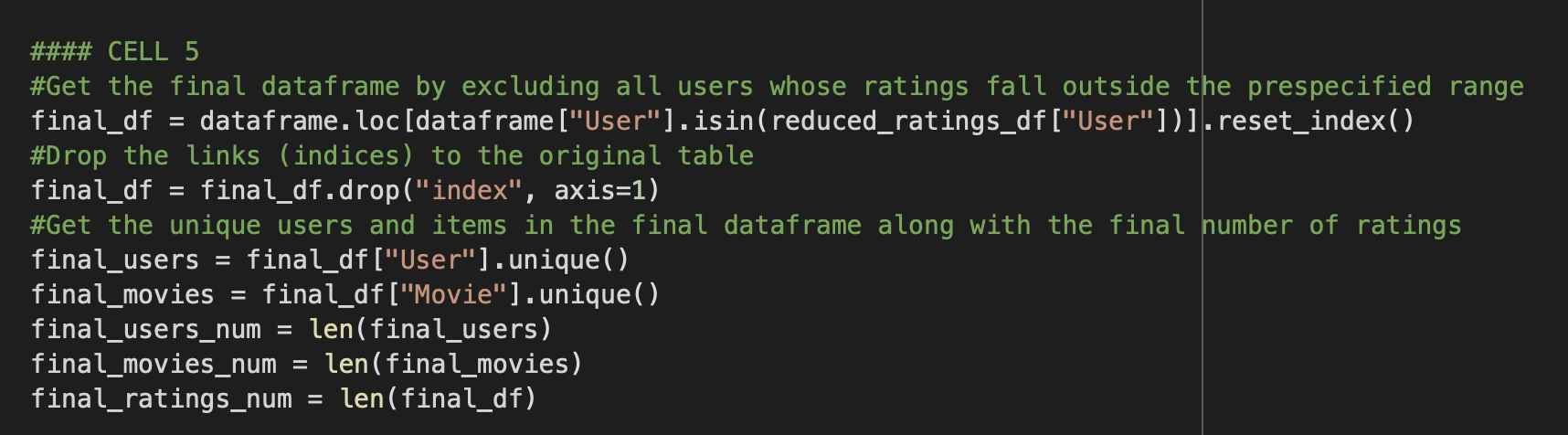
Ιστόγραμμα συχνοτήτων για το πλήθος των αξιολογήσεων:



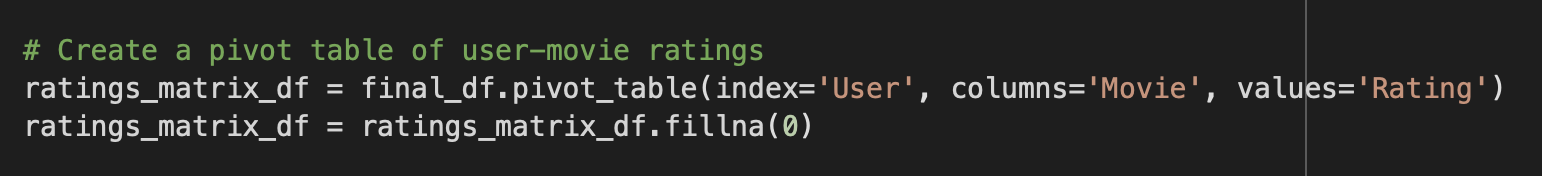
Ιστόγραμμα συχνοτήτων για το χρονικό εύρος μεταξύ των αξιολογήσεων κάθε χρήστη:



### 4.Δημιουργήστε μια εναλλακτική αναπαράσταση του συνόλου των δεδομένων ως ένα σύνολο διανυσμάτων προτιμήσεων με . Συγκεκριμένα, μπορούμε να γράψουμε ότι:



Αφού περιορίσουμε το dataframe στους χρήστες που ορίσαμε στο προηγούμενο ερώτημα, θέλουμε να μετασχηματίσουμε το dataframe έτσι ώστε οι γραμμές του να αναπαριστούν τους χρήστες, οι στήλες τις ταινίες και τα στοιχεία του dataframe να είναι η αξιολόγηση του χρήστη στην κάθε ταινία. Εάν ο χρήστης δεν έχει αξιολογήσει κάποια ταινία, το στοιχείο θα είναι μηδέν.



# Αλγόριθμοι Ομαδοποίησης Δεδομένων

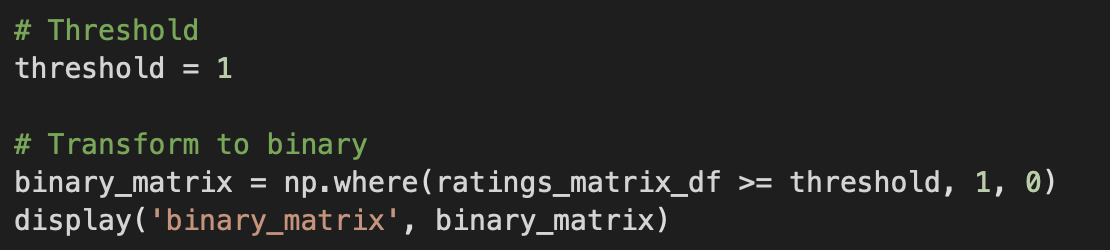
### Να οργανώσετε το περιορισμένο σύνολο των χρηστών σε συστάδες (clusters) της μορφής έτσι ώστε βασιζόμενοι στην διανυσματική αναπαράσταση των προτιμήσεών τους μέσω του συνόλου . Η διαδικασία ομαδοποίησης των διανυσμάτων προτιμήσεων των χρηστών επάνω στο περιορισμένο σύνολο των αντικειμένων , μπορεί να πραγματοποιηθεί με κατάλληλη παραμετροποίηση του αλγορίθμου k-means μεταβάλλοντας την μετρική που αποτιμά την απόσταση μεταξύ δύο διανυσμάτων προτιμήσεων για ένα ζεύγος χρηστών και ως . Συγκεκριμένα, μπορείτε να χρησιμοποιήσετε τις παρακάτω μετρικές (\*):

* 1. έτσι ώστε:

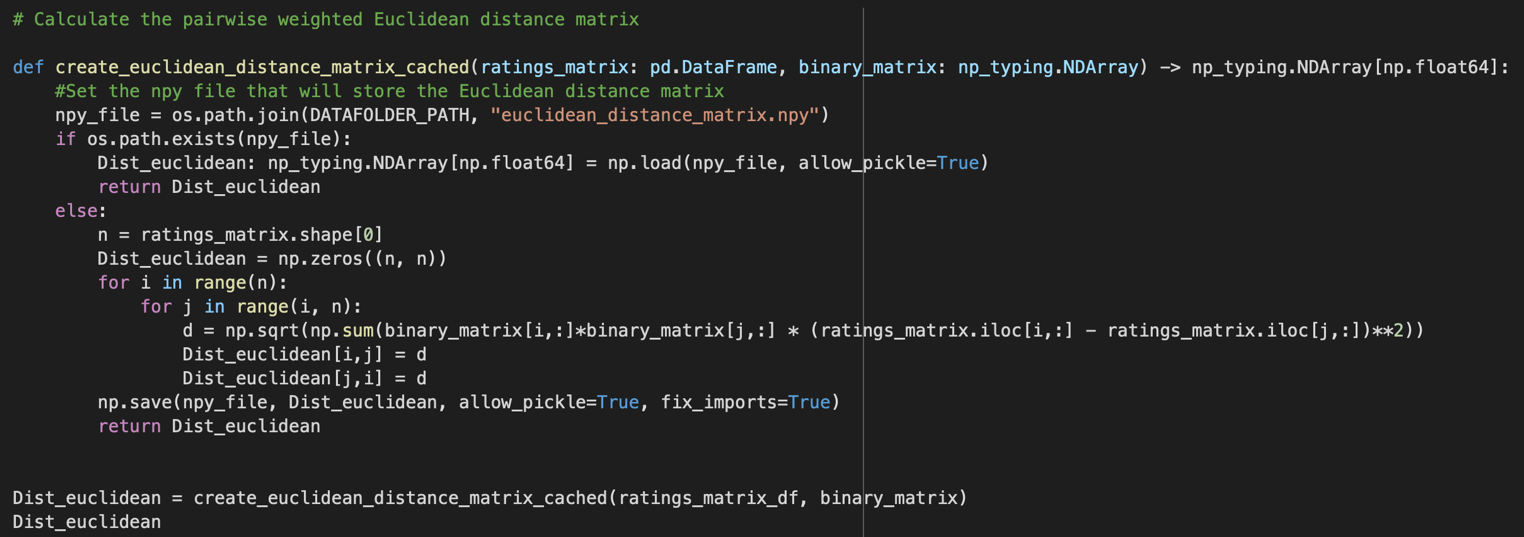
(**3**)

Η συνάρτηση αποτιμά το αν ο χρήστης έχει αξιολογήσει ή όχι το αντικείμενο . Προφανώς, μέσω της συγκεκριμένης συνάρτηση μπορούμε αν υπολογίσουμε την τιμή του που αποτιμά το πλήθος των αξιολογήσεων που παρείχε συνολικά ο χρήστης ως: .

Αρχικά δημιουργούμε τον πίνακα των «βαρών» που περιέχει τις τιμές του λ για τον κάθε χρήστη. Η τιμή των στοιχείων σε αυτόν τον πίνακα θα είναι 1 εάν ο χρήστης έχει αξιολογήσει την ταινία και 0 εάν δεν υπάρχει αξιολόγηση.

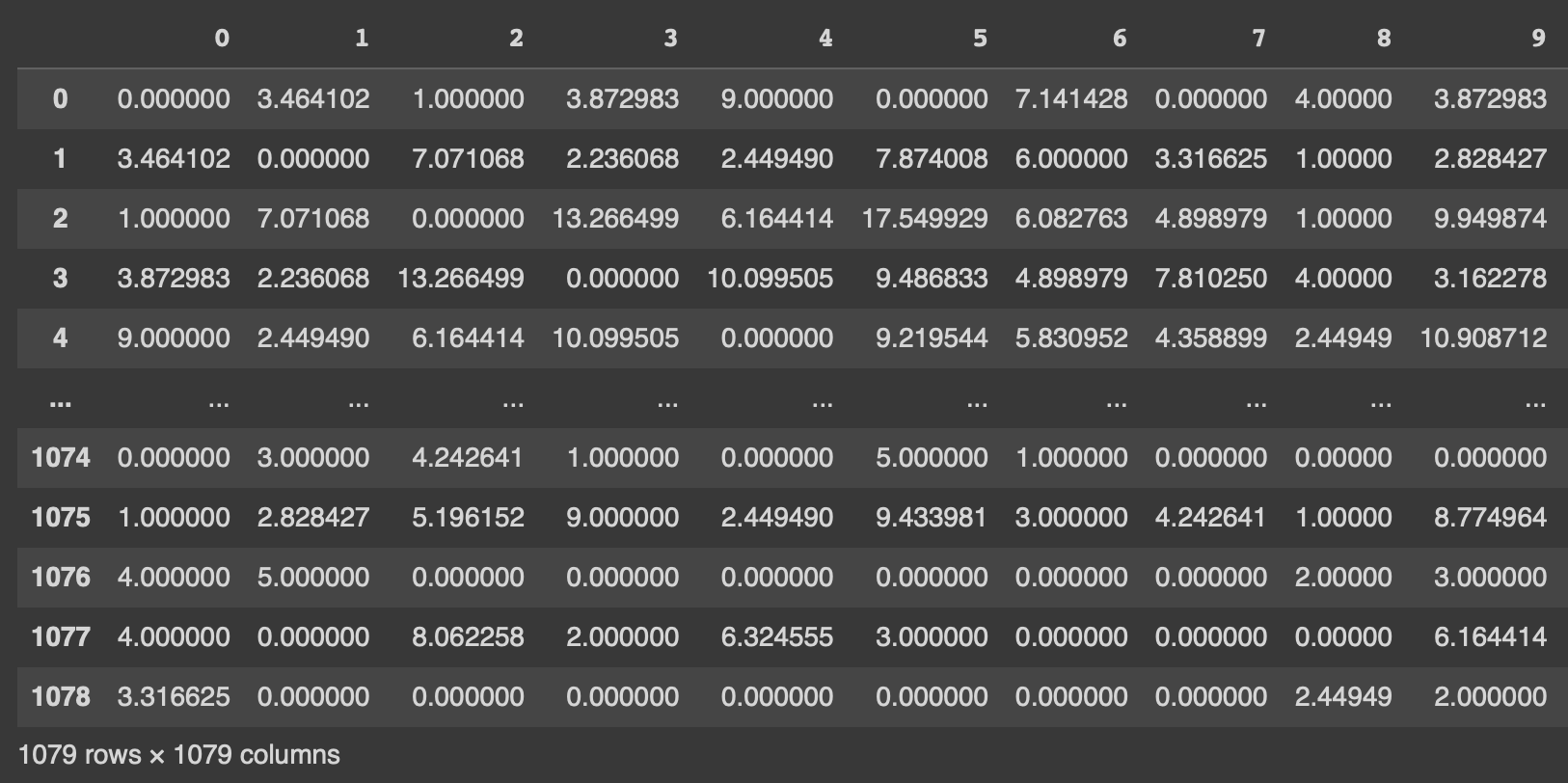


Θέλουμε τώρα να κατασκευάσουμε τον πίνακα των Ευκλείδειων αποστάσεων μεταξύ των αξιολογήσεων των χρηστών χρησιμοποιώντας την ανωτέρω μετρική:



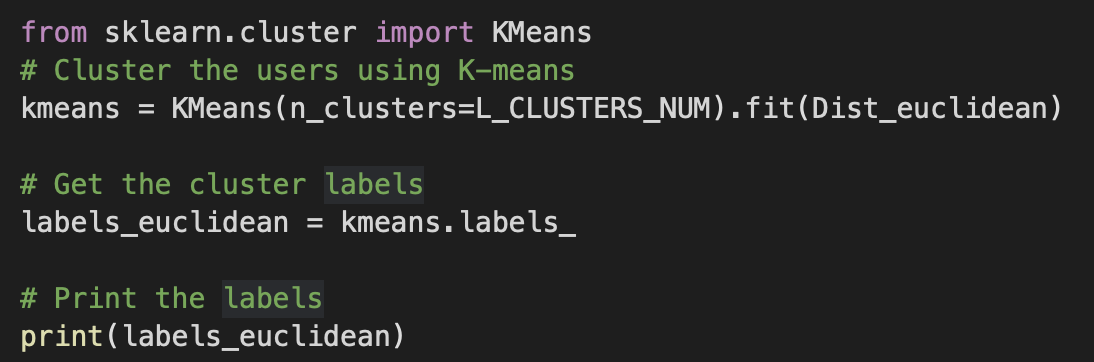
Για λόγους επίσπευσης των υπολογισμών, αναθέτουμε κατευθείαν την απόσταση μεταξύ των χρηστών [ i, j ] = [ j, i ] = d.

Το αποτέλεσμα που λαμβάνουμε είναι της ακόλουθης μορφής:



Οι γραμμές και οι στήλες αυτού του dataframe είναι οι μοναδικοί χρήστες ενώ το στοιχείο που υπάρχει σε κάθε περίπτωση είναι η μεταξύ τους απόσταση.

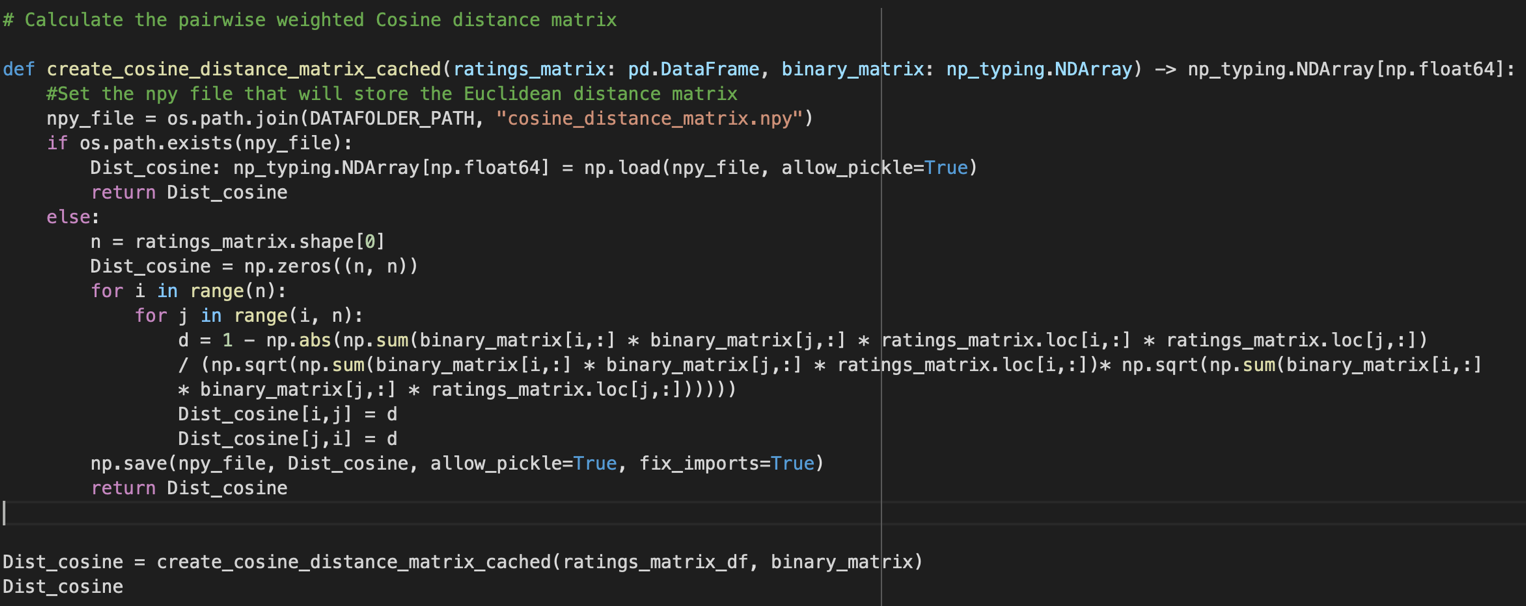
Στον πίνακα που έχουμε δημιουργήσει, θα χρησιμοποιήσουμε τον αλγόριθμο k-means έτσι ώστε να ομαδοποιήσουμε τις αποστάσεις μεταξύ των χρηστών. Θα δημιουργήσουμε δηλαδή συστάδες χρηστών (clusters) των οποίων οι μεταξύ τους αποστάσεις ανά δύο θα είναι οι μικρότερες δυνατές.



### 



Για να δημιουργήσουμε συστάδες χρησιμοποιώντας την ανωτέρω μετρική, θα λειτουργήσουμε με την ίδια μεθοδολογία όπως στο προηγούμενο ερώτημα.



Αρχικά δημιουργούμε τον πίνακα των αποστάσεων μεταξύ των ζευγών των χρηστών. Αυτός θα είναι της ακόλουθης μορφής:

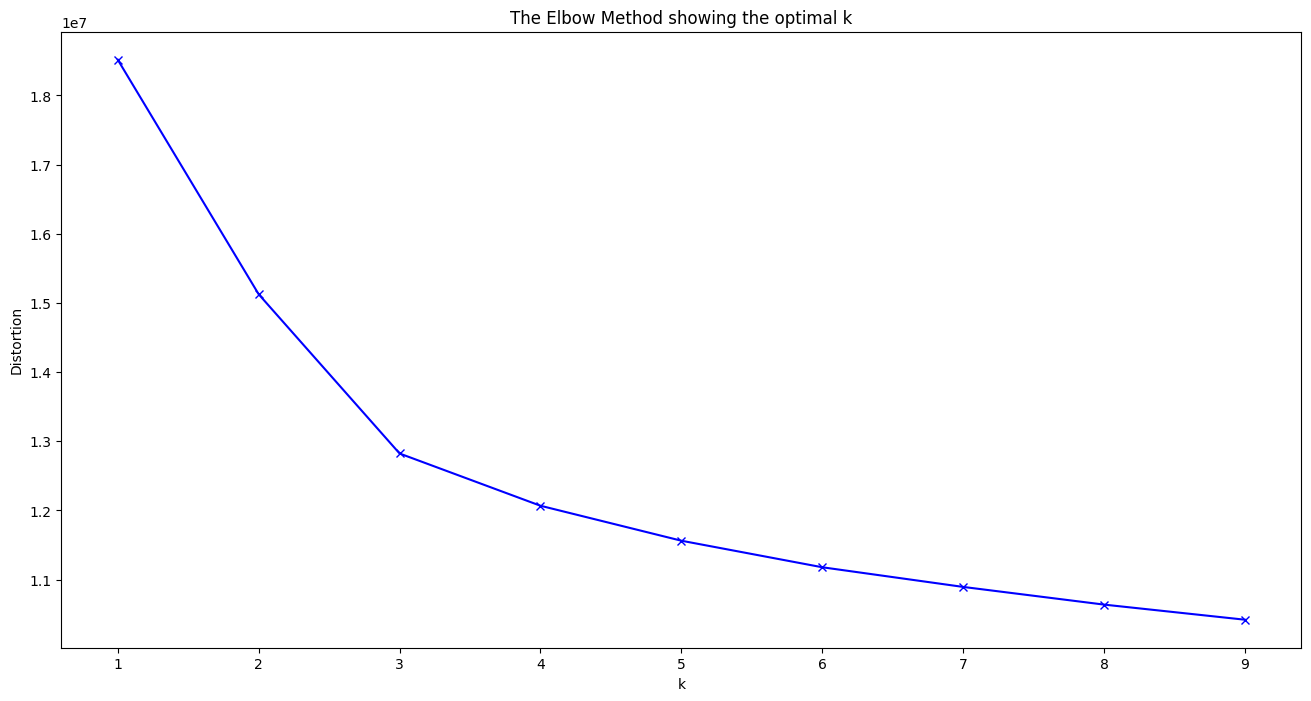
### 

Από τον ανωτέρω πίνακα θα δημιουργήσουμε τις συστάδες των χρηστών, χρησιμοποιώντας τον αλγόριθμο k-means.

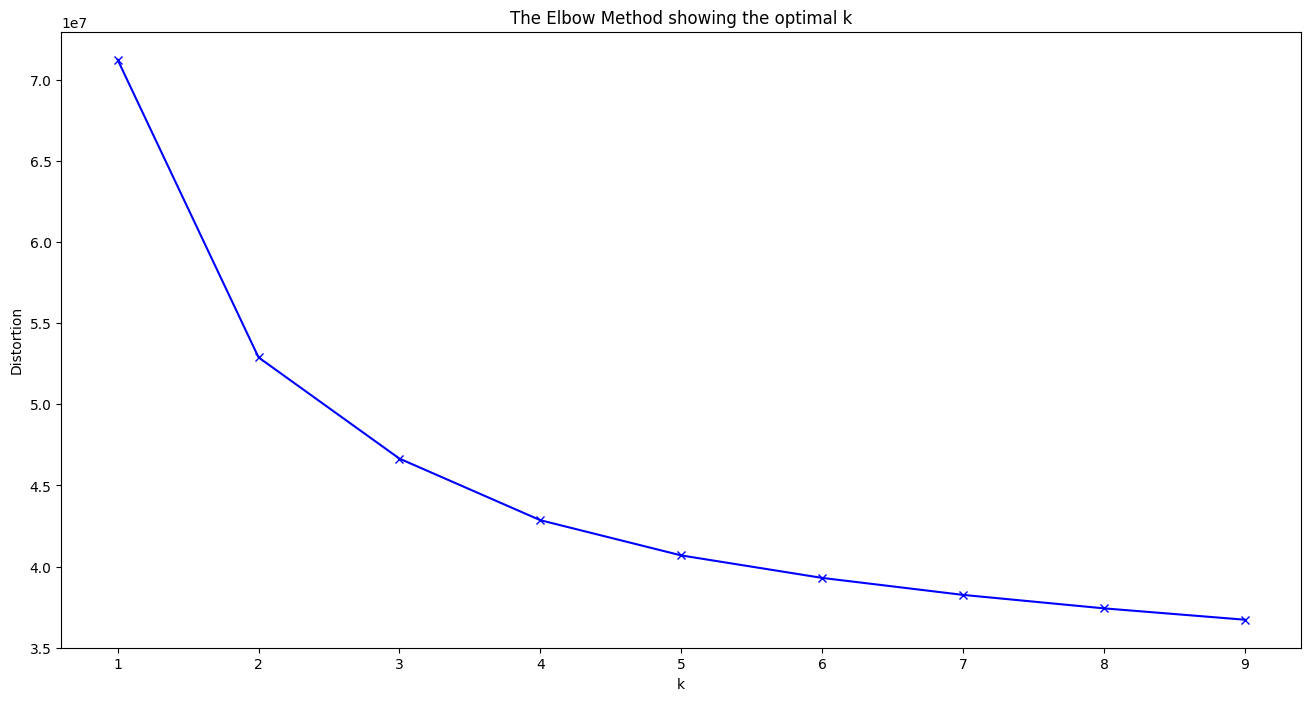
### 

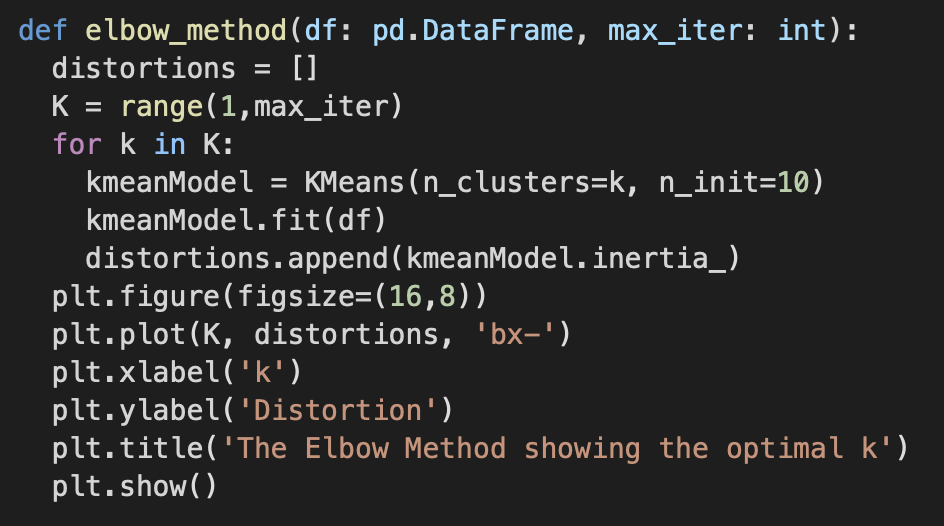
* 1. Να αναπαραστήσετε γραφικά τις συστάδες των χρηστών που αναγνωρίστηκαν από τον αλγόριθμο k-means για κάθε μία από τις παραπάνω μετρικές για διάφορες τιμές της παραμέτρου .

Για να επιλέξουμε τον βέλτιστο αριθμό των Κ διαφορετικών συστάδων χρηστών που θέλουμε να χρησιμοποιήσουμε στον αλγόριθμο k-means χρησιμοποιήσαμε την elbow method. Αυτή απέφερε τα ακόλουθα αποτελέσματα για τη μετρική της Ευκλείδειας απόστασης:



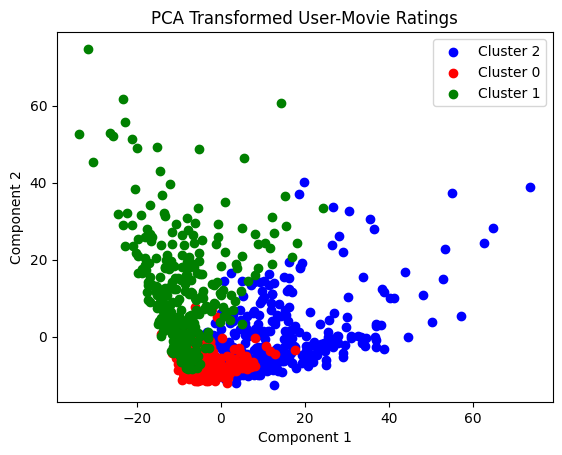
Ενώ για τη μετρική της απόστασης συνημίτονου:



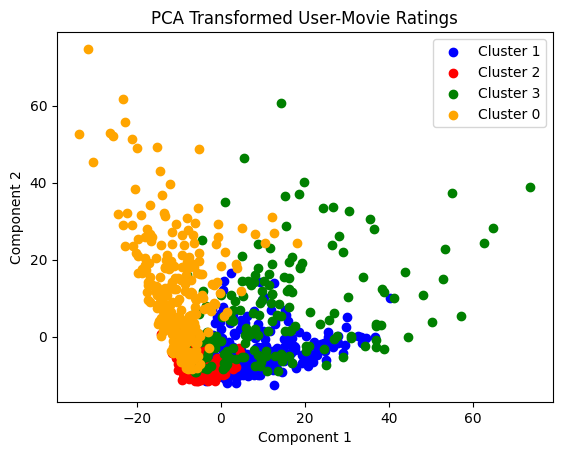


Σε κάθε περίπτωση, εκτιμούμε ότι περισσότερο ενδιαφέρον θα έχουν οι συστάδες χρηστών που θα παραχθούν για 3 < L < 5.

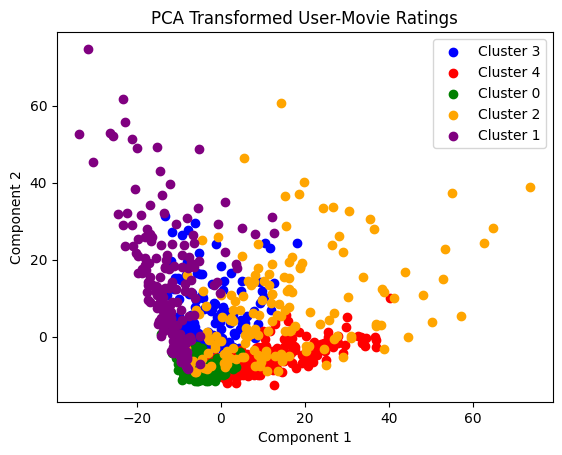
Για L = 3:



Για L = 4:

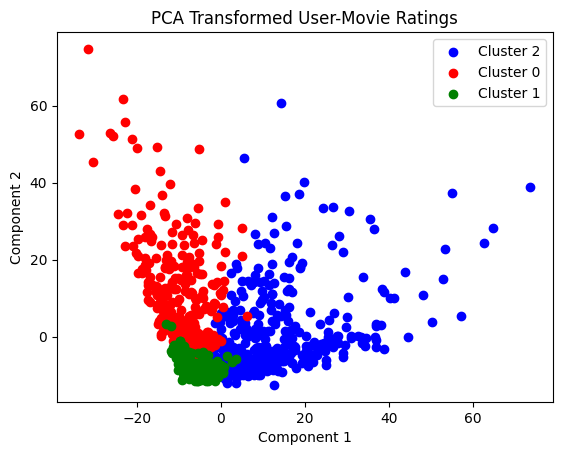


Για L = 5:

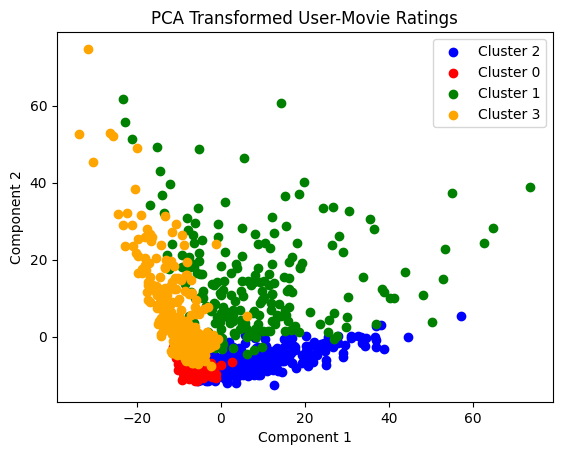


Με χρήση της απόστασης των συνημίτονων έχουμε τα εξής αποτελέσματα:

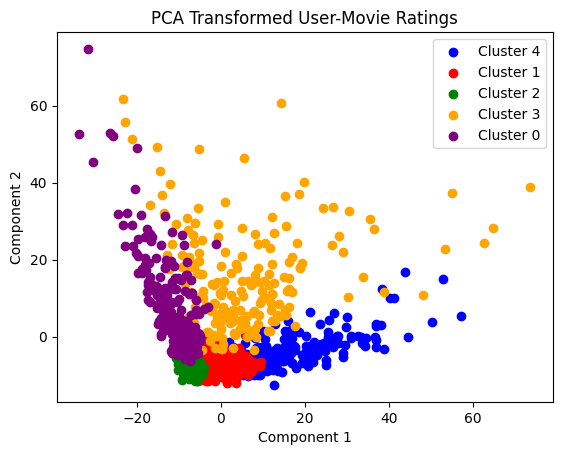
Για L = 3:



Για L = 4:



Για L = 5:



Η μεθοδολογία που ακολουθήσαμε για να απεικονίσουμε τις συστάδες των χρηστών για τις διάφορες τιμές του L ήταν ως εξής:

* Υπολογίσαμε τις συστάδες των χρηστών που παράγονται από τους πίνακες των αποστάσεων για τις δύο διαφορετικές μετρικές ( Ευκλείδεια απόσταση και απόσταση συνημίτονου ).
* Πήραμε τον πίνακα Χρηστών – Ταινιών και εφαρμόσαμε τη μέθοδο PCA ώστε να μετασχηματίσουμε τον πίνακα από [1079 rows × 68084 columns] σε [1079 rows x 2 columns] ώστε να μπορέσουμε να τους απεικονίσουμε στον δυσδιάστατο χώρο.
* Στον πίνακα που έχουμε δημιουργήσει, δημιουργούμε τη στήλη ‘Clusters’ έτσι ώστε να αναθέσουμε διαφορετικό χρώμα σε κάθε χρήστη αναλόγως τη συστάδα στην οποία ανήκει.

* 1. Να σχολιάσετε την αποτελεσματικότητα των συγκεκριμένων μετρικών στην αποτίμηση της ομοιότητας μεταξύ ενός ζεύγους διανυσμάτων προτιμήσεων χρηστών και .

Και στις δύο περιπτώσεις, η αποτελεσματικότητα της ομοιότητας των χρηστών βασίζεται σε μεγάλο βαθμό στην ύπαρξη ή όχι κοινά αξιολογημένων ταινιών από τους δύο χρήστες καθώς έχουμε τροποποιήσει τις δύο μετρικές ώστε να λαμβάνουν των πίνακα «βαρών» λ.

Για τη μετρική της Ευκλείδειας απόστασης:

* Η ομοιότητα μεταξύ των χρηστών είναι αντιστρόφως ανάλογη της απόστασης μεταξύ τους. Τα αριθμητικά στοιχεία των αξιολογήσεων δηλαδή παίζουν ρόλο κατά την επιλογή της συστάδας για τον χρήστη.

Για τη μετρική της απόστασης συνημίτονου:

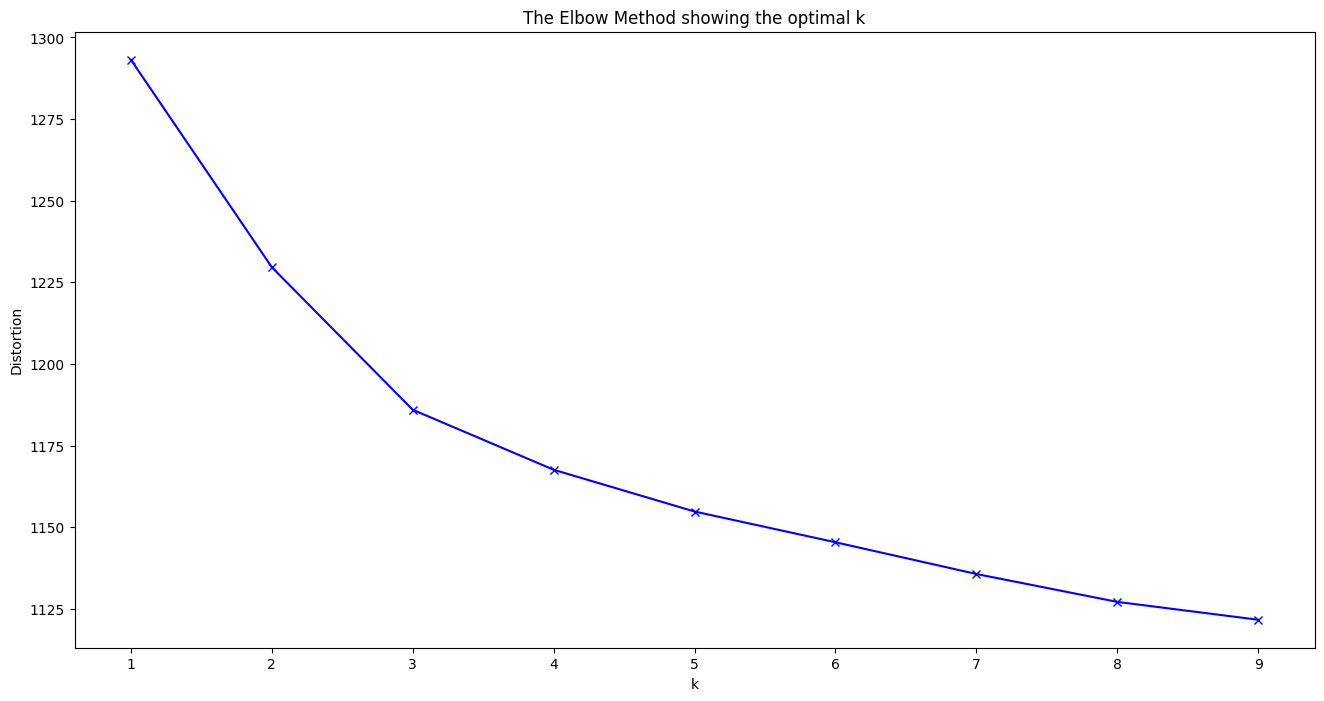
* Είναι πιο χρήσιμη όταν δε μας ενδιαφέρει τόσο η τιμή αυτή καθεαυτή των αξιολογήσεων του χρήστη, καθώς εξαρτάται από τη γωνία που σχηματίζουν τα διανύσματα τους.

# Αλγόριθμοι Παραγωγής Συστάσεων με Χρήση Τεχνητών Νευρωνικών Δικτύων

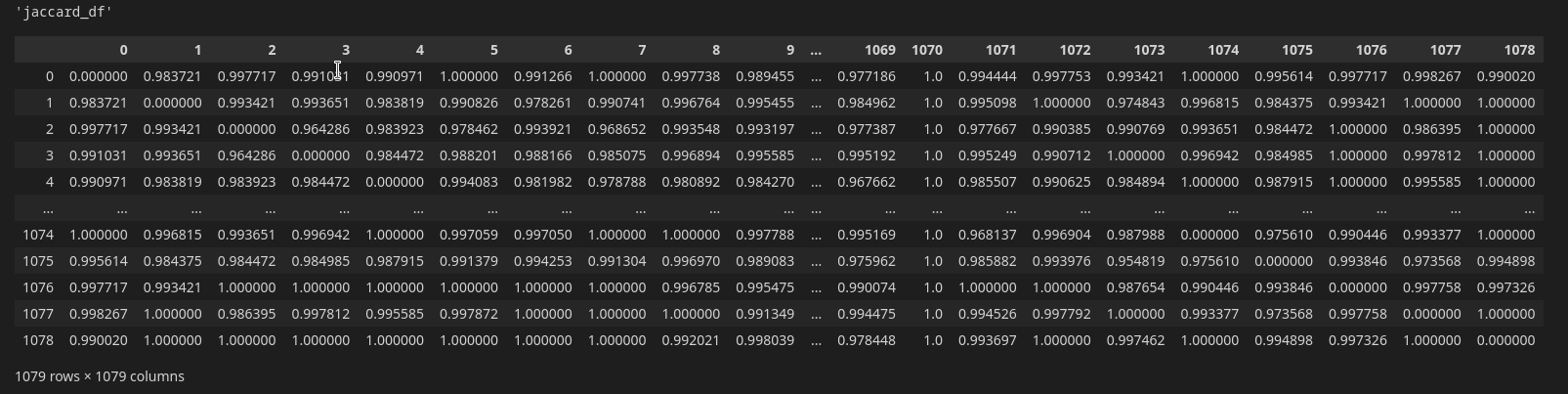
**2. Να δημιουργήσετε μια εναλλακτική οργάνωση του περιορισμένου συνόλου των χρηστών σε συστάδες έτσι ώστε με**

**κάνοντας χρήση της παρακάτω μετρικής (\*\*):**

Έχουμε βρει μέσω της **elbow method** στον πίνακα των jaccard distances ότι ο καλύτερος αριθμός clusters είναι ο 5. Έτσι ορίζουμε την global μεταβλητή **L\_CLUSTERS\_NUM = 5** για να δημιουργήσουμε 5 clusters.



Έχουμε υπολογίσει στην μεταβλητή **jaccard\_dist** (np array) και αντίστοιχα στην **jaccard\_df** τον ίδιο πίνακα σε μορφή DataFrame, τις Jaccard Distances όλων των χρηστών με όλους τους χρήστες (δηλ. **jaccard\_dist.shape** == (#Users, #Users)).



Με τον παρακάτω κώδικα βρίσκουμε τα jaccard\_labels με τα οποία κατηγοριοποιούμε τους χρήστες σε L\_CLUSTERS\_NUM clusters.

def kmeans\_clustering(jaccard\_dist, L):

# Initialize k-means object

kmeans = KMeans(n\_clusters=L)

# Fit the k-means object to the Jaccard distance matrix

kmeans.fit(jaccard\_dist)

return kmeans.labels\_

def create\_jaccard\_labels\_cached(jaccard\_dist, L: int):

npy\_file = os.path.join(DATAFOLDER\_PATH, "L\_K\_DEPEND\_jaccard\_labels.npy")

if os.path.exists(npy\_file):

jaccard\_labels: np\_typing.NDArray = np.load(npy\_file, allow\_pickle=True)

return jaccard\_labels

else:

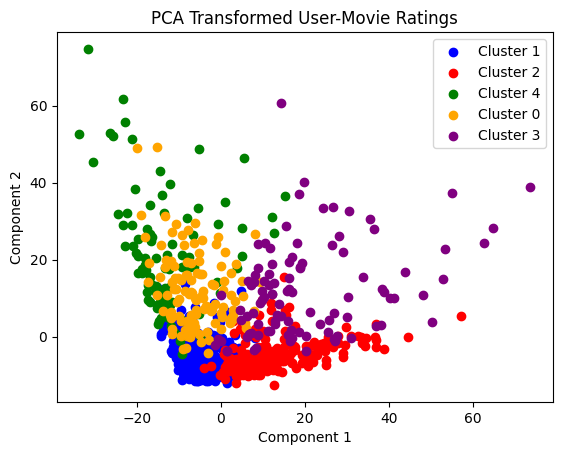
jaccard\_labels = kmeans\_clustering(jaccard\_dist, L)

np.save(npy\_file, jaccard\_labels, allow\_pickle=True, fix\_imports=True)

return jaccard\_labels

jaccard\_labels = create\_jaccard\_labels\_cached(jaccard\_dist, L\_CLUSTERS\_NUM)

Με την μέθοδο PCA απεικονίζουμε τους χρήστες χρωματίζοντάς τους αναλόγως της συστάδας (cluster) στην οποία ανήκουν:



1. **Να εξηγήσετε τι εκφράζει η συγκεκριμένη μετρική και να προσδιορίσετε τα μειονεκτήματά της σε σχέση με τις μετρικές που περιγράφονται στις σχέσεις (2) και (4) υπό το πρίσμα της ιδιαίτερης οργάνωσης των χρηστών που μπορεί να επιφέρει.**

Η μετρική της σχέσης 5 περιγράφει την Jaccard distance μεταξύ των χρηστών u και v. Η σχέση ορίζει την Jaccard index ή αλλιώς jaccard similarity coefficient μεταξύ των χρηστών u και v. Αυτός ο συντελεστής ομοιότητας στην περίπτωση μας δείχνει κατά πόσο 2 χρήστες έχουν βαθμολογήσει τις ίδιες ταινίες και είναι μια ποσότητα μεταξύ 0 και 1 όπου οι λιγότερο όμοιοι χρήστες θα έχουν συντελεστή ομοιότητας κοντά στο 1. Η Jaccard απόσταση που χρησιμοποιούμε θα είναι πάλι μεταξύ 0 και 1, αλλά οι πιο κοντινοί χρήστες θα έχουν απόσταση κοντά στο 0.

Εμείς την Jaccard απόσταση την υπολογίζουμε στον χώρο των ταινιών, αλλά με μέτρο την φ(u) η οποία θα έχει τιμή 0 ή 1 και θα δείχνει μόνο αν ένας χρήστης έχει βαθμολογήσει ή όχι την συγκεκριμένη ταινία, αγνοώντας την αριθμητική τιμή της αξιολόγησης.

**Μειονεκτήματα:**

1. Μειονέκτημα αυτής της μετρικής είναι ότι δεν λαμβάνει καθόλου υπόψιν την βαθμολογία του rating ενός χρήστη παρά μόνο τις κοινές βαθμολογήσεις και το πλήθος της ένωσης των βαθμολογήσεων και των 2 χρηστών. Μπορούν δηλαδή 2 χρήστες να έχουν βαθμολογήσει σχεδόν τις ίδιες ταινίες, ο πρώτος με 10.0 και ο δεύτερος με 1.0 αλλά η Jaccard απόσταση τους θα είναι κοντά στο 0.
2. Διαφορετικά sets με τον ίδιο αριθμό κοινών βαθμολογήσεων θα έχουν διαφορετική Jaccard similarity και Jaccard distance.

Άλλα γενικά μειονεκτήματα της Jaccard distance (όπως ότι δεν λαμβάνει υπόψιν την συχνότητα που εμφανίζονται κάποιοι όροι παρά μόνο το πλήθος των κοινών όρων) δεν μας αφορούν καθώς όπως έχουμε πει στην περίπτωσή μας έχουμε δυαδικές τιμές (0 ή 1) λόγω της συνάρτησης φ(u).

1. **Η μετρική που περιγράφεται μέσω της σχέσης (5) μπορεί να χρησιμοποιηθεί προκειμένου να προσδιοριστεί το σύνολο των πλησιέστερων γειτόνων ενός χρήστη . Επομένως, για τον εκάστοτε χρήστη της εκάστοτε συστάδας μπορούμε να το διάνυσμα τον προσωπικών του προτιμήσεων καθώς και τα διανύσματα των πλησιέστερων γειτόνων του , ,…, εντός της συστάδας . Στόχος του συγκεκριμένου ερωτήματος είναι να αναπτύξετε ένα πολυστρωματικό νευρωνικό δίκτυο για κάθε συστάδα χρηστών το οποίο θα προσεγγίζει τις αξιολογήσεις του κάθε χρήστη εντός αυτής μέσω των αξιολογήσεων των πλησιέστερων γειτόνων του μέσω μιας συνάρτησης της μορφής:**

Έπειτα από πολλές δοκιμές καταλήξαμε ότι ο βέλτιστος αριθμό κοντινότερων γειτόνων είναι >=4 και <=10. Διαλέγουμε και θέτουμε την μεταβλητή **K\_NEIGHBORS\_NUM = 6** ώστε να επιλέξουμε τους 6 κοντινότερους (κατά απόσταση Jaccard) χρήστες.

Χρησιμοποιώντας την συνάρτηση **calculate\_clusters\_ratings\_jaccard\_distances\_users\_indexes()** φτιάχνουμε για κάθε cluster τους πίνακες **cluster\_ratings** (**cluster\_ratings.shape == (#cluster users, #total movies)**) με τα ratings κάθε χρήστη του cluster και **cluster\_jaccard\_distances** (**cluster\_jaccard\_distances.shape == (#cluster users, #total users)**) με τα Jaccard distances κάθε χρήστη του cluster ως προς όλους τους άλλους χρήστες. Χρησιμποποιούμε όλους τους άλλους χρήστες και όχι μόνο αυτούς του cluster για λόγους performance γιατί τις έχουμε ήδη υπολογίσει στα προηγούμενα ερωτήματα.

def calculate\_clusters\_ratings\_jaccard\_distances\_users\_indexes(L\_CLUSTERS\_NUM: int, jaccard\_labels: np\_typing.NDArray, ratings\_matrix\_array: np\_typing.NDArray, jaccard\_dist: np\_typing.NDArray):

clusters\_ratings\_dict = { i: [] for i in range(L\_CLUSTERS\_NUM)}

clusters\_jaccard\_distances\_dict = { i: [] for i in range(L\_CLUSTERS\_NUM)}

clusters\_users\_indexes\_dict: dict[int, set[int]] = { i: set() for i in range(L\_CLUSTERS\_NUM)}

for i in range(jaccard\_labels.shape[0]):

label = jaccard\_labels[i]

cluster\_ratings = clusters\_ratings\_dict[label]

cluster\_jaccard\_distances = clusters\_jaccard\_distances\_dict[label]

cluster\_users\_indexes = clusters\_users\_indexes\_dict[label]

cluster\_ratings.append(ratings\_matrix\_array[i])

cluster\_jaccard\_distances.append(jaccard\_dist[i])

cluster\_users\_indexes.add(i)

clusters\_ratings\_list: list[np\_typing.NDArray] = []

for key in clusters\_ratings\_dict:

cluster\_ratings = np.array(clusters\_ratings\_dict[key])

clusters\_ratings\_list.append(cluster\_ratings)

clusters\_jaccard\_distances\_list: list[np\_typing.NDArray] = []

for key in clusters\_jaccard\_distances\_dict:

cluster\_jaccard\_distances = np.array(clusters\_jaccard\_distances\_dict[key])

clusters\_jaccard\_distances\_list.append(cluster\_jaccard\_distances)

clusters\_users\_indexes\_list: list[set[int]] = []

for key in clusters\_users\_indexes\_dict:

clusters\_users\_indexes\_list.append(clusters\_users\_indexes\_dict[key])

Με την συνάρτηση **find\_nearest\_neighbors\_using\_jaccard()** βρίσκουμε τους **K\_NEIGHBORS\_NUM** κοντινότερους γείτονες κάθε user για κάθε cluster, εξαιρώντας τον εαυτό του καθώς και τους γείτονες που δεν ανήκουν στο cluster. Σημείωση όπως αναφέραμε πριν τα indexes των γειτόνων είναι ως προς τον πίνακα με τους συνολικούς χρήστες (όχι μόνο του cluster) για λόγους performance, οπότε αργότερα θα πάρουμε τα ratings των γειτόνων από τον αρχικό πίνακα των ratings και όχι από το cluster\_ratings.

def find\_nearest\_neighbors\_using\_jaccard(K\_NEIGHBORS\_NUM: int, cluster\_jaccard\_distances: np\_typing.NDArray, cluster\_users\_indexes: set[int]):

nearest\_neighbors\_list: list[np\_typing.NDArray] = []

for row\_index in range(cluster\_jaccard\_distances.shape[0]):

cluster\_jaccard\_row = cluster\_jaccard\_distances[row\_index]

## set the distance of the users that are the same a index to a value higher thatn 1

for j in range(cluster\_jaccard\_row.shape[0]):

if j == row\_index or not(j in cluster\_users\_indexes):

cluster\_jaccard\_row[j] = 2

## find the k smallest indexes

k\_nearest\_indexes = np.argpartition(cluster\_jaccard\_row, K\_NEIGHBORS\_NUM)

# k\_nearest\_indexes = k\_nearest\_indexes[k\_nearest\_indexes != row\_index]

nearest\_neighbors\_list.append(k\_nearest\_indexes[:K\_NEIGHBORS\_NUM])

return np.array(nearest\_neighbors\_list)

Για κάθε cluster θα χρησιμοποιήσουμε την συνάρτηση **create\_nn\_original\_df()** για να φτιάξουμε το dataframe **nn\_origin\_df**. Αυτό θα περιέχει στη στήλη **USER\_RATINGS** τα ratings όλων των χρηστών του cluster για κάθε ταινία διαμορφωμένα στην σειρά σε μόνο μια στήλη (διάνυσμα). Στις στήλες **NEIGHBOR\_RATINGS\_**{**0**,**1**,.., **K\_NEIGHBORS\_NUM**} θα περιέχει τα ratings των K πλησιέστερων γειτόνων του κάθε χρήστη για κάθε ταινία. Αποθηκεύουμε αυτά τα dataframe σε pickle files και δεν τα κρατάμε στην μνήμη ώστε να μην γεμίζει η μνήμη και να φορτώνουμε κάθε φορά μόνο εκείνο που χρειαζόμαστε.

def create\_nn\_original\_df(ratings\_matrix\_array: np\_typing.NDArray, cluster\_ratings: np\_typing.NDArray, nearest\_neighbors: np\_typing.NDArray, NEIGHBOURS\_COLUMNS: list[str]):

# create user\_ratings\_list

user\_index\_list: list[int] = []

movie\_index\_list: list[int] = []

user\_ratings\_list: list[int] = []

for i in range(nearest\_neighbors.shape[0]):

user\_index\_list.extend([i for ratings in cluster\_ratings[i]])

movie\_index\_list.extend([j for j in range(cluster\_ratings[i].shape[0])])

user\_ratings\_list.extend(cluster\_ratings[i])

# create neighbors list

neighbors\_list: list[list[int]] = []

for i in range(nearest\_neighbors.shape[1]):

neighbor\_ratings\_list: list[int] = []

for j in range(nearest\_neighbors.shape[0]):

neighbor = nearest\_neighbors[j][i]

# nearest neighbors have indexs for the ratings\_matrix\_array up to total users

# so use ratings\_matrix\_array instead of cluster\_ratings.

# We have previously ensured that all neighbors belong to this clusters

neighbor\_ratings\_list.extend(ratings\_matrix\_array[neighbor])

neighbors\_list.append(neighbor\_ratings\_list)

nn\_origin\_df = pd.DataFrame()

nn\_origin\_df['USER\_INDEX'] = user\_index\_list

nn\_origin\_df['MOVIE\_INDEX'] = movie\_index\_list

nn\_origin\_df['USER\_RATINGS'] = user\_ratings\_list

for i in range(len(neighbors\_list)):

neighbor\_ratings\_list: list[int] = neighbors\_list[i]

nn\_origin\_df[NEIGHBOURS\_COLUMNS[i]] = neighbor\_ratings\_list

return nn\_origin\_df

Από αυτά τα δεδομένα θα χρησιμοποιήσουμε την στήλη **USER\_RATINGS** ως τα labels μας για το regression πρόβλημα που θα προσπαθήσουμε να λύσουμε με το νευρωνικό μας δίκτυο και τις στήλες **NEIGHBOR\_RATINGS\_**{**0**,**1**,.., **K\_NEIGHBORS\_NUM**} ως τα features μας. Ωστόσο για να εκπαιδευτεί σωστά το νευρωνικό δίκτυο δεν μπορούμε να έχουμε 0 στα features μας καθώς αυτές είναι βαθμολογίες όπου δεν υπάρχουν (θυμίζουμε οι υπάρχουσες βαθμολογίες είναι από 1 εώς 10 ενώ τα μηδενικά είναι στην ουσία οι NaN τιμές για τα ratings που δεν ξέρουμε καθόλου). Επίσης δεν μπορούμε να έχουμε γραμμές που να έχουν μόνο μηδενικά στα features μας καθώς αυτό θα σημαίνει ότι κανείς κοντινότερος γείτονας δεν θα έχει βαθμολογήσει την συγκεκριμένη ταινία και ως αποτέλεσμα να μην μπορεί να προβλεφτεί κάποια βαθμολογία που να βγάζει νόημα. Αυτό το επιτυγχάνουμε με την συνάρτηση **create\_nn\_filtered\_normalized\_df()**. Διαιρούμε τα labels και τα features μας με κάποιον παράγοντα για normalization, ωστόσο είδαμε χειρότερα αποτελέσματα έτσι και για αυτό έχουμε θέσει τον παράγοντα αυτόν ίσον με 1 αναιρώντας οποιαδήποτε επίδραση που θα μπορούσε να έχει.

def create\_nn\_filtered\_normalized\_df(nn\_origin\_df: pd.DataFrame, NEIGHBOURS\_COLUMNS: list[str], ratings\_normalize\_factor: float):

# filter the rows that have USER\_RATINGS == 0

nn\_filtered\_df = nn\_origin\_df.copy()[nn\_origin\_df['USER\_RATINGS'] != 0]

# filter the rows that have all NEIGHBOR\_RATINGS == 0

filter\_neighbors\_query: str = f'{NEIGHBOURS\_COLUMNS[0]} != 0'

for neihbor\_column in NEIGHBOURS\_COLUMNS[1:]:

filter\_neighbors\_query += f' | {neihbor\_column} != 0'

nn\_filtered\_df = nn\_filtered\_df.query(filter\_neighbors\_query)

# create filtered normalized df

nn\_filtered\_normalized\_df = nn\_filtered\_df.copy()

columns\_to\_normalize = ['USER\_RATINGS']

columns\_to\_normalize.extend(NEIGHBOURS\_COLUMNS)

nn\_filtered\_normalized\_df[columns\_to\_normalize] = nn\_filtered\_normalized\_df[columns\_to\_normalize] / ratings\_normalize\_factor

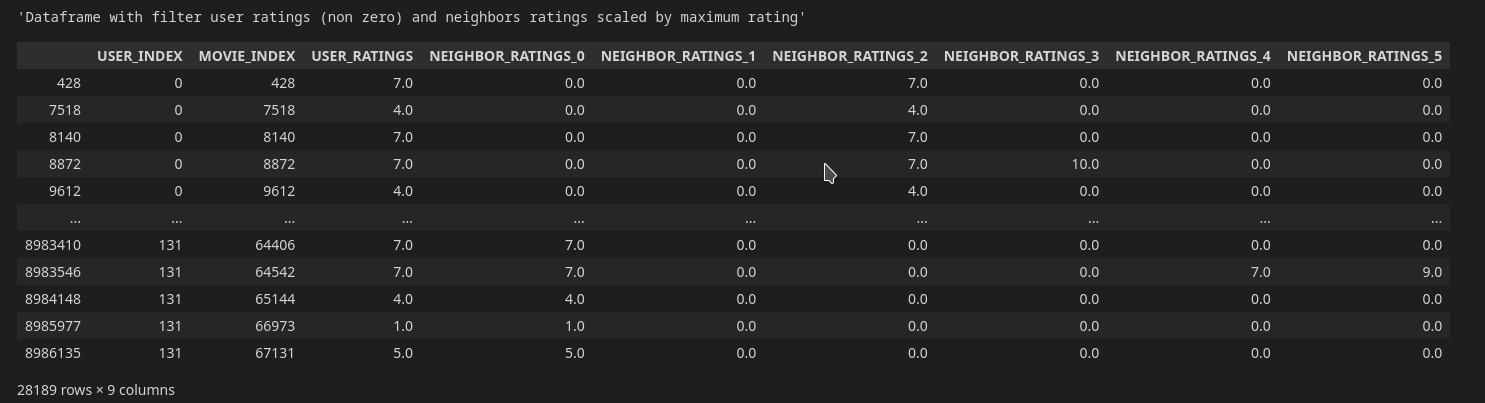
display('Dataframe with filter user ratings (non zero) and neighbors ratings scaled by maximum rating')

display(nn\_filtered\_normalized\_df)

display(nn\_filtered\_normalized\_df.describe())

return nn\_filtered\_df, nn\_filtered\_normalized\_df

Π.χ. για το Cluster0 το τελικό dataframe θα είναι:



Με την συνάρτηση **create\_model()** δημιουργούμε το νευρωνικό μας δίκτυο μέσω της κλάσης Sequential του tensorflow. Αυτό θα περιέχει ένα Input Layer, 3 κρυφά Dense layers με διπλασιαζόμενα units το καθένα από το προηγούμενο και ένα τελικό Output layer με ένα unit όπου θα προβλέπει την τελικό rating για μια ταινία ενός χρήστη από τα ratings των γειτόνων του.

def create\_model(my\_learning\_rate: float, input\_shape: tuple):

"""Create and compile a simple linear regression model."""

# Most simple tf.keras models are sequential.

model = tf.keras.models.Sequential()

# Add the layer containing the feature columns to the model.

model.add(tf.keras.layers.InputLayer(input\_shape=input\_shape))

model.add(tf.keras.layers.Masking(

mask\_value=0

))

# Implement L2 regularization in the first hidden layer.

model.add(tf.keras.layers.Dense(units=12,

activation='relu',

kernel\_regularizer=tf.keras.regularizers.l2(0.01),

name='Hidden1'))

# Implement L2 regularization in the second hidden layer.

model.add(tf.keras.layers.Dense(units=24,

activation='relu',

kernel\_regularizer=tf.keras.regularizers.l2(0.01),

name='Hidden2'))

# Implement L2 regularization in the third hidden layer.

model.add(tf.keras.layers.Dense(units=48,

activation='relu',

kernel\_regularizer=tf.keras.regularizers.l2(0.01),

name='Hidden3'))

# Define the output layer.

model.add(tf.keras.layers.Dense(units=1,

name='Output'))

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=my\_learning\_rate),

loss="mean\_squared\_error",

metrics=[tf.keras.metrics.MeanSquaredError(), tf.keras.metrics.MeanAbsoluteError()])

return model

my\_model = create\_model(learning\_rate, input\_shape)

1. **Το σύνολο των χρηστών της κάθε συστάδας μπορεί να διαμεριστεί περεταίρω σε ένα υποσύνολο χρηστών για εκπαίδευση του νευρωνικού δικτύου και σε ένα υποσύνολο χρηστών για τον έλεγχο της επίδοσης του νευρωνικού δικτύου έτσι ώστε με . Η καταλληλότερη διαμόρφωση των δεδομένων για την εκπαίδευση των νευρωνικών δικτύων αυτού του ερωτήματος θα μπορούσε να αναπαρασταθεί ως εξής:  
     
   Έστω ότι το σύνολο των χρηστών που μετέχουν στην συστάδα δίνεται από την σχέση: με . Τότε το σύνολο των διανυσμάτων χαρακτηριστικών και το αντίστοιχο σύνολο των ετικετών θα μπορούσε να οργανωθεί σε έναν πίνακα της μορφής:**

**και**

Όπως βλέπουμε στον προηγούμενο κώδικα έχουμε επιλέξει ως loss function της εκπαίδευσης το mean\_squared\_error αντί για του mean\_absolute\_error καθώς είδαμε ότι παρουσιάζει μεγαλύτερη ακρίβεια και καλύτερα αποτελέσματα.

Από το φιλτραρισμένο **filtered\_normalized\_df** παίρνουμε το 80% των δεδομένων ως train dataset και το υπόλοιπο 20% ως test dataset.

train\_df, test\_df = train\_test\_split(nn\_filtered\_normalized\_df, test\_size=0.2, random\_state=42)

Από το train\_df παίρνουμε τα features X\_train και τα labels Y\_train ως numpy arrays κι εκπαιδεύουμε το νευρωνικό δίκτυο με την **train\_model()**. Κρατάμε τα διανύσματα των epochs, καθώς και των mean\_squared\_error και mean\_absolute\_error και τα αναπαριστάμε με την **plot\_the\_loss\_curve()**. Διαλέγουμε μικρό **learning\_rate = 0.001** και λίγα **epochs = 64** επειδή είδαμε ότι πέρα από αυτά δεν αλλάζουν ουσιαστικά οι γραφικές των losses, για να αποφύγουμε το overfitting.

def plot\_the\_loss\_curve(epochs, mse\_or\_mae, is\_mse: bool, filename: str):

"""Plot a curve of loss vs. epoch."""

plt.figure()

plt.xlabel("Epoch")

ylabel = 'Train Mean Squared Error' if is\_mse else 'Train Mean Absolute Error'

plt.ylabel(ylabel)

plt.plot(epochs, mse\_or\_mae, label="Loss")

plt.legend()

plt.ylim([mse\_or\_mae.min()\*0.95, mse\_or\_mae.max() \* 0.6])

plt.savefig(os.path.join(RESULTS\_PATH, filename))

plt.show()

def train\_model(model: tf.keras.models.Sequential, X\_train: np\_typing.NDArray, Y\_train: np\_typing.NDArray, epochs: int, batch\_size: int=1):

"""Train the model by feeding it X\_train."""

# Features as a numpy array

history = model.fit(x=X\_train, y=Y\_train, batch\_size=batch\_size, epochs=epochs, shuffle=True)

# The list of epochs is stored separately from the rest of history.

epochs = history.epoch

# To track the progression of training, gather a snapshot

# of the model's mean squared error at each epoch.

hist = pd.DataFrame(history.history)

mse = hist["mean\_squared\_error"]

mae = hist["mean\_absolute\_error"]

return epochs, mse, mae

learning\_rate = 0.001

epochs = 64

batch\_size = 128

X\_train = train\_df[NEIGHBOURS\_COLUMNS].to\_numpy()

Y\_train = train\_df['USER\_RATINGS'].to\_numpy()

epochs, train\_mse\_series, train\_mae\_series = train\_model(my\_model, X\_train, Y\_train, epochs, batch\_size)

train\_mse = train\_mse\_series.iloc[-1]

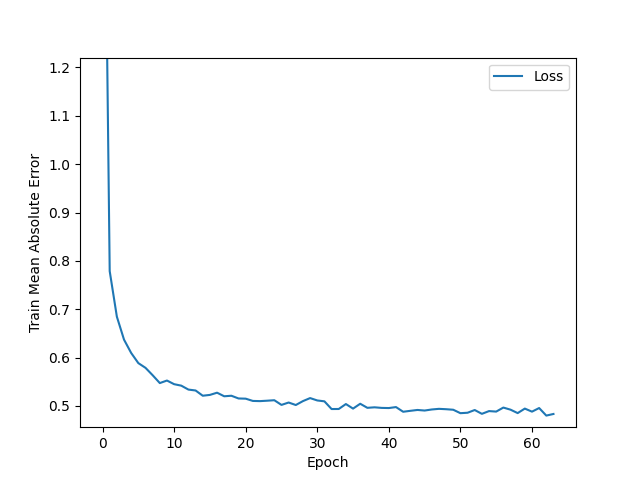
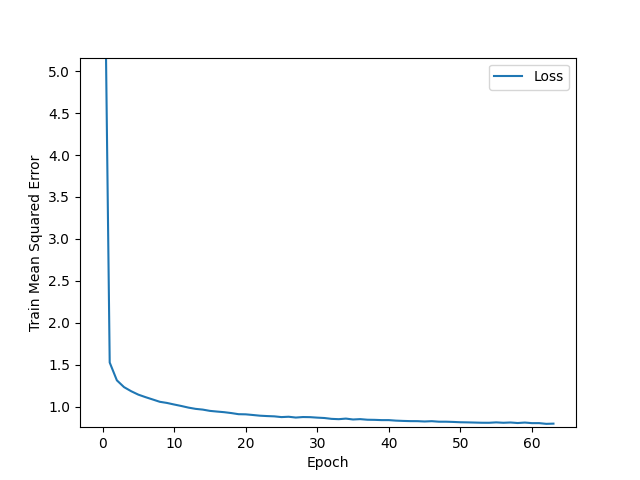
train\_mae = train\_mae\_series.iloc[-1]

plot\_the\_loss\_curve(epochs, train\_mse\_series, True, f'cluster{cluster\_index}\_mse.png')

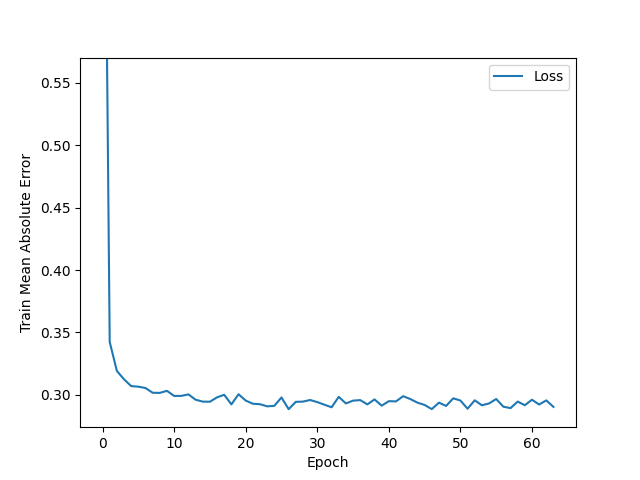
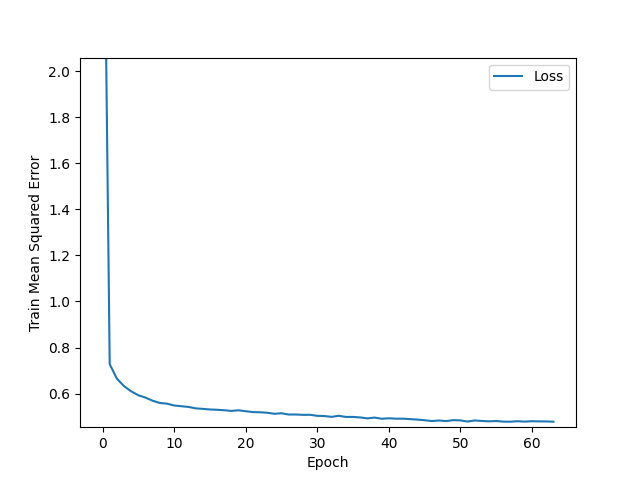
plot\_the\_loss\_curve(epochs, train\_mae\_series, False, f'cluster{cluster\_index}\_mae.png')

Παρουσιάζουμε τα train plots των mean\_squared\_error και mean\_absolute\_error σε σχέση με τα epochs για κάθε cluster:

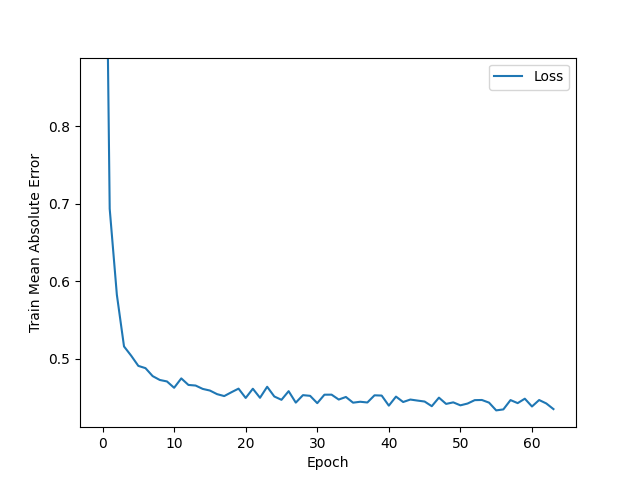
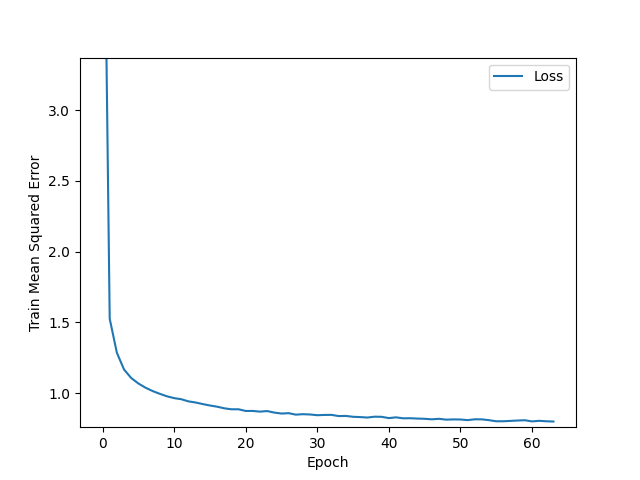
**Cluster0 train plots:**



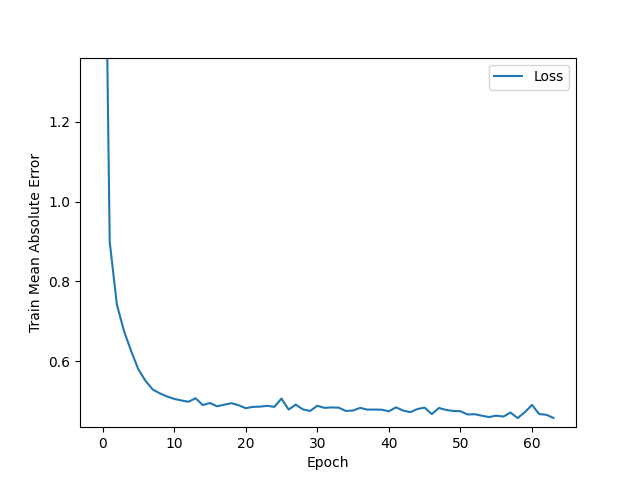
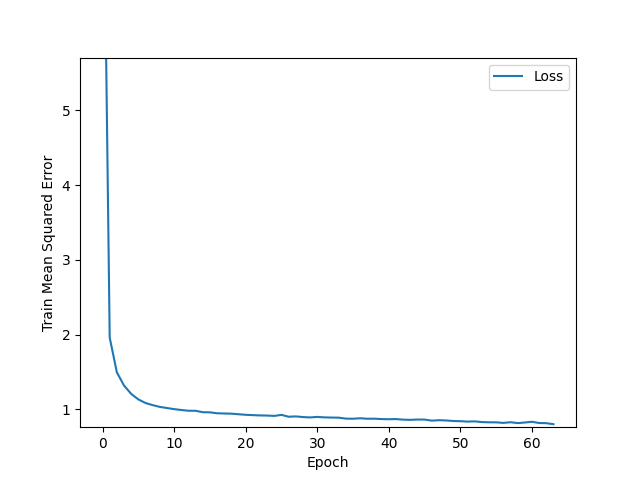
**Cluster1 train plots:**



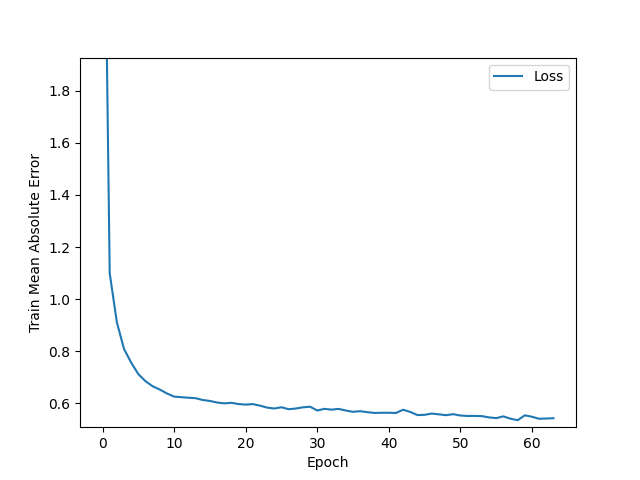
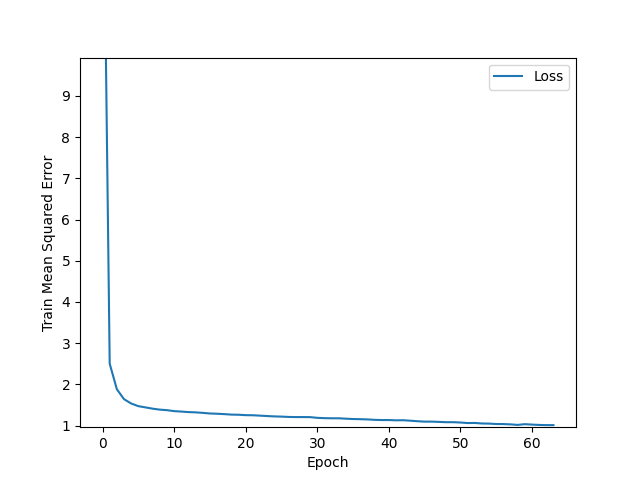
**Cluster2 train plots:**



**Cluster3 train plots:**



**Cluster4 train plots:**



1. **Η προσεγγιστική ακρίβεια των παραπάνω νευρωνικών δικτύων μπορεί να μετρηθεί μέσω της μετρικής του μέσου απόλυτου σφάλματος ανάμεσα στις πραγματικές και τις εκτιμώμενες αξιολογήσεις των χρηστών. Να παρουσιάσετε πίνακες των αποτελεσμάτων σας τόσο για την ακρίβεια εκπαίδευσης όσο και για την ακρίβεια ελέγχου για κάθε συστάδα χρηστών.**

Από το test\_df παίρνουμε τα features X\_test και τα labels Y\_test ως numpy arrays και κάνουμε evaluate το νευρωνικό δίκτυο με την **evaluate\_model()**.

def evaluate\_model(model: tf.keras.models.Sequential, X\_test: np\_typing.NDArray, Y\_test: np\_typing.NDArray, batch\_size: int=1):

"""Evaluate the model against the X\_test"""

# Features as a numpy array

return model.evaluate(x = X\_test, y = Y\_test, batch\_size=batch\_size)

X\_test = test\_df[NEIGHBOURS\_COLUMNS].to\_numpy()

Y\_test = test\_df['USER\_RATINGS'].to\_numpy()

test\_loss, test\_mse, test\_mae = evaluate\_model(my\_model, X\_test, Y\_test, batch\_size)

Παίρνουμε ως features X\_origin τα ratings από το αρχικό μη φιλτραρισμένο **nn\_origin\_df** και τα χρησιμοποιούμε για να προβλέψουμε τα USER\_RATINGS από τους γείτονες μέσω της **predict\_model()**. Υπολογίζουμε τα πραγματικά mean\_squared\_error και mean\_absolute\_error από τα δεδομένα μας που δεν είναι 0 σε σύγκριση με τα predictions μέσω της **calculate\_real\_mse\_mae()**.

def predict\_model(model: tf.keras.models.Sequential, X\_origin: np\_typing.NDArray, batch\_size: int=1):

"""Predict the model with the X\_origin"""

# Features as a numpy array

return model.predict(x = X\_origin, batch\_size=batch\_size)

def calculate\_real\_mse\_mae(origin\_ratings: np\_typing.NDArray, predictions: np\_typing.NDArray):

"""Calculate the real mse and mae comparing real ratings and predictions."""

n = 0

absolute\_sum = 0

squared\_sum = 0

for i in range(origin\_ratings.shape[0]):

# take into consideration only non 0 ratings

if origin\_ratings[i] != 0.0:

n += 1

abs\_value = abs(origin\_ratings[i] - predictions[i])

absolute\_sum += abs\_value

squared\_sum += math.sqrt(abs\_value)

mse = absolute\_sum / n

mae = squared\_sum / n

return mse, mae

X\_origin = nn\_origin\_df[NEIGHBOURS\_COLUMNS].to\_numpy()

predictions\_normalized = predict\_model(my\_model, X\_origin, batch\_size)

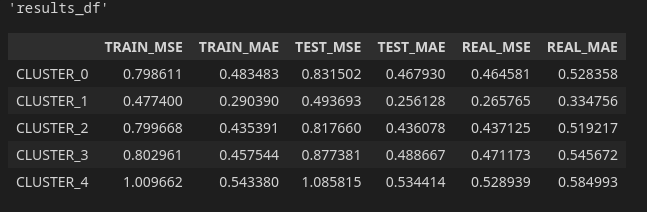
# turn list of sinle item lists to a sigle list with floats

predictions\_normalized = np.array([prediction\_normalized[0] for prediction\_normalized in predictions\_normalized])

predictions = predictions\_normalized \* ratings\_normalize\_factor

real\_mse, real\_mae = calculate\_real\_mse\_mae(real\_ratings, predictions\_normalized)

Συγκεντρώνουμε όλα τα αποτελέσματα από όλα τα clusters στο DataFrame **results\_df**:



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | TRAIN\_MSE | TRAIN\_MAE | TEST\_MSE | TEST\_MAE | REAL\_MSE | REAL\_MAE |
| CLUSTER\_0 | 0.8 | 0.48 | 0.83 | 0.47 | 0.46 | 0.53 |
| CLUSTER\_1 | 0.48 | 0.29 | 0.49 | 0.26 | 0.27 | 0.33 |
| CLUSTER\_2 | 0.8 | 0.44 | 0.82 | 0.44 | 0.44 | 0.52 |
| CLUSTER\_3 | 0.8 | 0.46 | 0.88 | 0.49 | 0.47 | 0.55 |
| CLUSTER\_4 | 1.01 | 0.54 | 1.09 | 0.53 | 0.53 | 0.58 |

Για καλύτερη ερμηνεία των προβλέψεων δημιουργούμε το **real\_ratings\_predictions\_df** ως αντίγραφο του **nn\_origin\_df** και προσθέτουμε το predictions ως στήλη “PREDICTIONS”. Ταξινομούμε ως προς το USER\_INDEX ascending και τα PREDICTIONS descending και τα αποθηκεύουμε ως csv.

real\_ratings\_predictions\_df = nn\_origin\_df.copy()

real\_ratings\_predictions\_df['USER\_RATINGS'] = real\_ratings

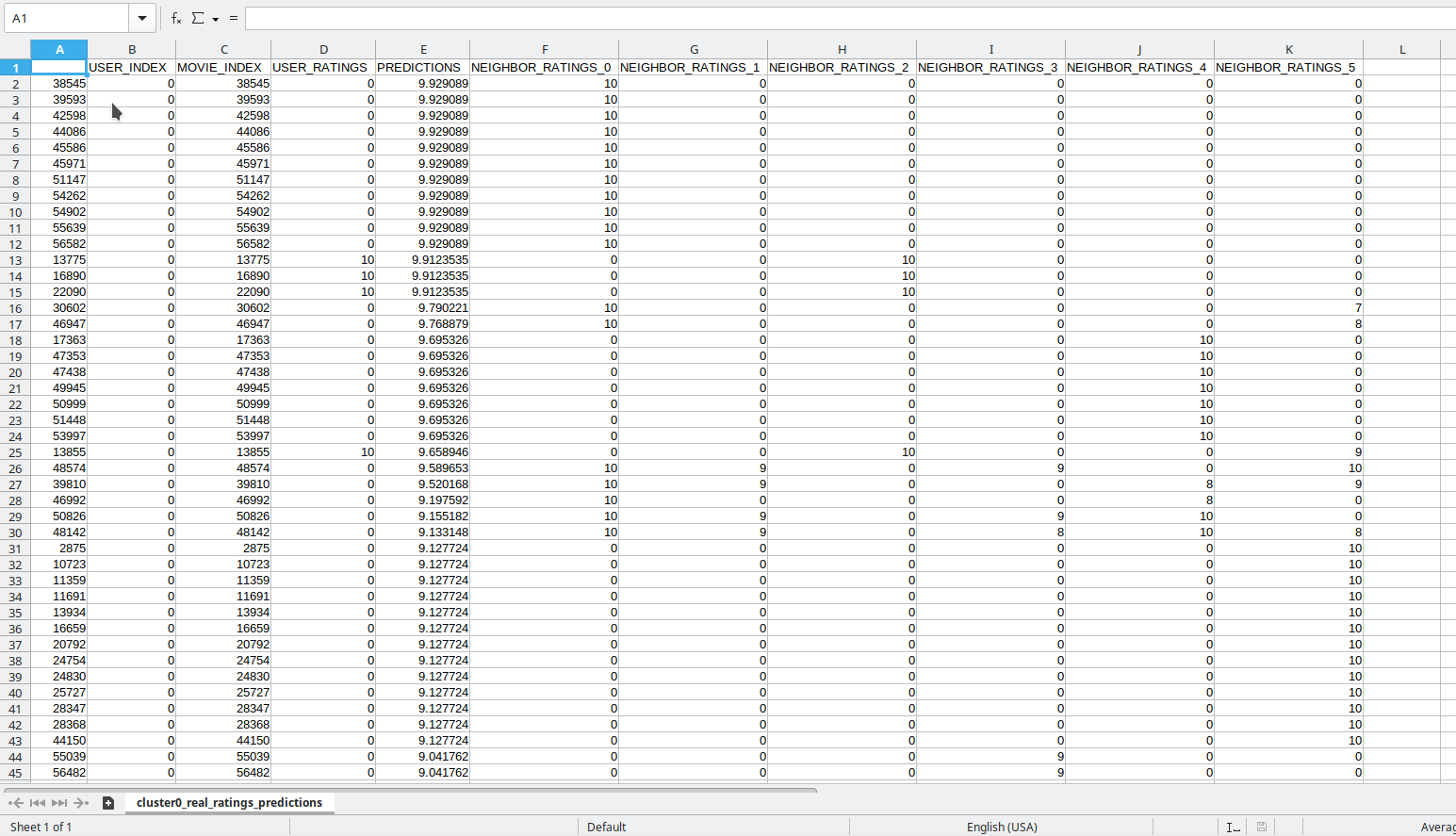
real\_ratings\_predictions\_df.insert(3, 'PREDICTIONS', predictions)

real\_ratings\_predictions\_df = real\_ratings\_predictions\_df.sort\_values(by=['USER\_INDEX', 'PREDICTIONS'], ascending=[True, False])

real\_ratings\_predictions\_df\_csv = os.path.join(RESULTS\_PATH, F"cluster{cluster\_index}\_real\_ratings\_predictions.csv")

real\_ratings\_predictions\_df.to\_csv(real\_ratings\_predictions\_df\_csv)

Έτσι βλέπουμε για παράδειγμα τα πρώτα rows του **cluster0\_real\_ratings\_predictions.csv**



Από αυτά τα αποτελέσματα μπορούμε π.χ. να φτιάξουμε ένα σύστημα προτάσεων ταινιών σε χρήστες βάση των κοντινότερων γειτόνων του. Για παράδειγμα βλέπουμε στο row 2 ότι μπορούμε να προτείνουμε την ταινία με index 38545 (index από τις συνολικές ταινίες) στο χρήστη με cluster index 0 (index από το cluster) καθώς η πρόβλεψη είναι υψηλή (9.93) λόγων των γειτόνων του ενώ ο ίδιος ο χρήστης δεν την έχει βαθμολογήσει.

Παρατηρούμε ότι οι προβλέψεις του νευρωνικού μας δικτύου δεν είναι καλές όταν όλοι οι γείτονες έχουν 0 ratings για μια ταινία (δηλαδή δεν την έχουν βαθμολογήσει), όπως και είναι αναμενόμενο. Σε ένα πραγματικό σύστημα προβλέψεων θα μπορούσαμε να φιλτράρουμε αυτές τις γραμμές όπου όλοι οι γείτονες έχουν 0 ratings ξέροντας ότι δεν μπορούμε να αποφανθούμε σωστό rating σε αυτές τις περιπτώσεις. Μπορούμε να αυξήσουμε τον αριθμό των πλησιέστερων γειτόνων για να έχουμε περισσότερες προβλέψεις όπου τουλάχιστον ένας γείτονας θα έχει κάποιο rating μη μηδενικό, αλλά από τις μετρήσεις μας είδαμε ότι χάνουμε σε ακρίβεια καθώς τα losses αυξάνονται.