Ανάκτηση Πληροφορίας

**Project 2022-2023**



|  |  |  |
| --- | --- | --- |
| **Στυλιανάκης Στυλιανός** | 1059713 | steliostylian@gmail.com |
| **Κονταράκης Ιωάννης Γεώργιος** | 1067375 | kogi\_20010@hotmail.com |

**[Icon

Description automatically generated](https://github.com/Steliostyl/Projects-kDDs-2022)**[**Link to GitHub**](https://github.com/Steliostyl/Data-Retrieval-Project-2022)

Πίνακας περιεχομένων

[Περιβάλλον υλοποίησης 3](#_Toc125839536)

[Βιβλιοθήκες που χρησιμοποιήθηκαν 3](#_Toc125839537)

[Ερώτημα 1 – ElasticSearch 3](#_Toc125839538)

[Σύνδεση με την Elastic, δημιουργία δείκτη & εισαγωγή βιβλίων 3](#_Toc125839539)

[Πραγματοποίηση ερωτημάτων 5](#_Toc125839540)

[Υπολογισμός συνδυαστικών βαθμολογιών 6](#_Toc125839541)

# Περιβάλλον υλοποίησης

Ως γλώσσα υλοποίησης επιλέχθηκε η **python**, καθώς υπάρχουν πολλές χρήσιμες βιβλιοθήκες για την εργασία.

### Βιβλιοθήκες που χρησιμοποιήθηκαν

#### pandas

Βασική βιβλιοθήκη διαχείρισης δεδομένων. Τα περισσότερα δεδομένα μας φορτώνονται, επεξεργάζονται και ανακτούνται από **DataFrames**, τη βασική δομή δεδομένων των pandas.

#### elasticsearch

Χρησιμοποιήθηκε για τη σύνδεση με την Elasticsearch, καθώς και την πραγματοποίηση των queries, μέσω REST API calls.

#### sklearn

Βιβλιοθήκη με πολλές χρήσιμα εργαλεία για υλοποίηση clustering, καθώς και νευρωνικών δικτύων. Χρησιμοποιήθηκε κυρίως στο ερώτημα 3.

#### kmodes

Βιβλιοθήκη για clustering, η οποία χρησιμοποιεί και την **sklearn** (σε συνδυασμό με άλλες βιβλιοθήκες). Συγκεκριμένα, από αυτήν τη βιβλιοθήκη χρησιμοποιήσαμε την κλάση **KPrototypes** για να υλοποιήσουμε το clustering.

#### matplotlib, pyplot, seaborn

Βιβλιοθήκες για σχεδιασμό και εμφάνιση γραφημάτων.

#### re

Βιβλιοθήκη αναγνώρισης κανονικών εκφράσεων, χρησιμοποιήθηκε στον καθαρισμό των datasets.

# Ερώτημα 1 – ElasticSearch

### Σύνδεση με την Elastic, δημιουργία δείκτη & εισαγωγή βιβλίων

Ξεκινώντας, αρχικοποιούμε ένα instance της κλάσης Elasticsearch με τα απαραίτητα στοιχεία για τη σύνδεση στο cluster μας. Την πρώτη φορά που θα τρέξουμε την Elasticsearch, πρέπει να δημιουργήσουμε ένα index στο οποίο θα προσθέσουμε τα βιβλία, όπως φαίνεται παρακάτω.

def main():

    ############################ ELASTICSEARCH #############################

    # Connect to the ElasticSearch cluster

    es = Elasticsearch(

        "http://localhost:9200",

        ssl\_assert\_fingerprint=CERT\_FINGERPRINT,

        basic\_auth=("elastic", ES\_PASSWORD)

        )

    # Create a new index in ElasticSearch called books

    print("Creating books index...")

    es\_functions.createIndex(es, idx\_name="books")

    # Insert a sample of data from dataset into ElasticSearch

    book\_count = es\_functions.insertData(es)[0]["count"]

    print(f"Inserted {book\_count} books into index.")

Με τη μέθοδο createIndex δημιουργούμε το books index, ορίζοντας τις ιδιότητες των δεδομένων που θα εισάγουμε. Σε σχόλιο υπάρχει η μέθοδος με την οποία μπορούμε να διαγράψουμε το index που έχουμε δημιουργήσει για να το ξαναφτιάξουμε από την αρχή.

def createIndex(es: Elasticsearch, idx\_name: str = "books") -> None:

  """Creates a new book index, deleting

  any pre-existing with the same name."""

  # Define the mappings of the books index.

  mappings = {

    "properties": {

      "isbn": {"type": "text", "analyzer": "keyword"},

      "book\_title": {"type": "text", "analyzer": "english"},

      "book\_author": {"type": "text", "analyzer": "standard"},

      "year\_of\_publication": {"type": "integer"},

      "publisher": {"type": "text", "analyzer": "standard"},

      "summary": {"type": "text", "analyzer": "english"},

      "category": {"type": "text", "analyzer": "standard"}

    }

  }

  # Delete pre-existing index with the same name

  #es.indices.delete(index=idx\_name)

  # Create a new index named idx\_name with the defined mappings

  es.indices.create(index=idx\_name, mappings=mappings)

Με τη μέθοδο insertData εισάγουμε τα βιβλία στη βάση. Επιλέξαμε να τα εισάγουμε όλα μαζί (in bulk) με μία κλήση στην Elastic, η οποία περιέχει λίστα με όλα τα βιβλία. Η μέθοδος επιστρέφει την απάντηση που πήραμε από την Elastic μετά την εισαγωγή των βιβλίων, όπου φαίνεται πόσα βιβλία υπάρχουν.

def insertData(es: Elasticsearch) -> str:

  """Parses the data of a specified csv file and

  inserts the data into Elasticsearch. Returns

  the number of entries after insertion."""

  # Parse the CSV dataset

  dataframe = pd.read\_csv(BOOKS).dropna().reset\_index()

  # Create a list containing the parsed rows from the CSV file

  bulk\_data = []

  for i,row in dataframe.iterrows():

      bulk\_data.append(

          {

              "\_index": "books",

              "\_id": i,

              "\_source": {

                "isbn": row["isbn"],

                "book\_title": row["book\_title"],

                "book\_author": row["book\_author"],

                "year\_of\_publication": row["year\_of\_publication"],

                "publisher": row["publisher"],

                "summary": row["summary"],

                "category": row["category"]

              }

          }

      )

  # Bulk insert the rows into ElasticSearch

  bulk(es, bulk\_data)

  # Refresh the books index and return the number of items in it.

  es.indices.refresh(index="books")

  resp = es.cat.count(index="books", format="json")

  return resp

### Πραγματοποίηση ερωτημάτων

Για την πραγματοποίηση ερωτημάτων, ο χρήστης πρέπει να εισάγει έναν αλφαριθμητικό όρο, τον οποίο θα αναζητήσει η Elastic στα βιβλία που της έχουμε φορτώσει στη βάση, καθώς και το αναγνωριστικό του.

    # Input query parameters and make query to ES

    search\_string = input("Enter search string:")

    user\_id = \_input("Enter your user ID (must be an integer): ", int)

    es\_reply = es\_functions.makeQuery(es, search\_string)

Τα ερωτήματα γίνονται μέσω της μεθόδου makeQuery. Για κάθε ερώτημα, ζητάμε στην Elastic να αναζητήσει τους όρους σε πολλαπλά πεδία των εγγραφών, πιο συγκεκριμένα στους τίτλους των βιβλίων, καθώς και στην περιγραφή τους. Όπως φαίνεται στο σώμα του ερωτήματος στη γραμμή των πεδίων, αποφασίσαμε να δώσουμε πιο μεγάλο βάρος στον τίτλο του βιβλίου (1.5 αντί για 1 συγκεκριμένα), παρά στην περίληψη. Αυτήν την απόφαση την πήραμε βασισμένοι στην υπόθεση ότι, αν οι όροι αναζήτησης υπάρχουν στον τίτλο, ουσιαστικά είναι εγγυημένο ότι περιλαμβάνονται στο **βασικό** θέμα του βιβλίου, σε αντίθεση με την περίληψη.

def makeQuery(es: Elasticsearch, search\_string: str) -> tuple:

  """Creates a query and returns Elasticsearch's answer."""

  # Create query body using inputs. Using multi\_match we check multiple fields

  # of an entry for the given search string and the final score is calculated

  # by adding the (weighted) score of the fields.

  query\_body = {

    "query": {

        "multi\_match": {

            "query": search\_string,

            "type": "most\_fields",

            "fields": ["book\_title^1.5", "summary"]

        }

    }

  }

  # Make the query to ElasticSearch

  es\_reply = es.search(

    index = "books",

    body = query\_body,

    size = 10\_000

  )

  return es\_reply["hits"]

### Υπολογισμός συνδυαστικών βαθμολογιών

Μόλις λάβουμε την απάντηση της Elastic, υπολογίζουμε τη συνδυαστική βαθμολογία των βιβλίων συνυπολογίζοντας τις πιθανές βαθμολογίες που έχουν κάνει οι χρήστες στα σχετικά βιβλία.

    # Get the combined scores for all replies

    print("\nCalculating combined scores...")

    combined\_scores\_df = functions.calculateCombinedScores(es\_reply, user\_id)

    combined\_scores\_df.to\_csv(SCORES\_NO\_CLUST, index=False)

Η μέθοδος calculateCombinedScores χρησιμοποιείται τόσο για το 1ο, όσο και για το 2ο ερώτημα, μετά το clustering. Για αυτόν το λόγο, έχει κάποια προαιρετικά ορίσματα τα οποία μας χρησιμεύουν μόνο στο 2ο ερώτημα. Τα βασικά ορίσματα που χρειάζονται και για τα 2 ερωτήματα, είναι η απάντηση της Elastic και το αναγνωριστικό του χρήστη. Τελικά, η μέθοδος επιστρέφει ένα DataFrame με την ανανεωμένη συνδυαστική βαθμολογία.

def calculateCombinedScores(es\_reply: dict, user\_id: int, use\_cluster\_ratings:\

    bool = False, avg\_clust\_ratings: pd.DataFrame = None,\

        cluster\_assigned\_users: pd.DataFrame = None) -> pd.DataFrame:

Αρχικά, η μέθοδος αποθηκεύει το max\_score, όλα τα βιβλία που επέστρεψε η Elastic σε μία λίστα από λεξικά και φορτώνει όλες τις αξιολογήσεις του χρήστη σε ένα DataFrame, χρησιμοποιώντας τη μέθοδο getUserRatings.

    books = es\_reply["hits"]

    max\_score = es\_reply["max\_score"]

    user\_ratings = getUserRatings(user\_id)

    print(f"User {user\_id} has rated {len(user\_ratings)} book(s).")

Η μέθοδος getUserRatings απλώς διαβάζει το CSV με όλα τα Book Ratings και επιστρέφει μόνο τις γραμμές που περιέχουν το user\_id που της περνάμε.

def getUserRatings(user\_id: int, filename: str = RATINGS) -> pd.DataFrame:

    """Read ratings CSV and return specified user's ratings."""

    users\_ratings = pd.read\_csv(filename)

    return  pd.read\_csv(filename).loc[users\_ratings["uid"] == user\_id]

Στη συνέχεια, αρχικοποιεί μία άδεια λίστα όπου θα μπουν τα βιβλία αφού υπολογιστεί η συνδυαστική βαθμολογία τους. Έπειτα, εάν η μέθοδος έχει κληθεί με το use\_cluster\_ratings=True (όπως καλείται στο 2ο ερώτημα δηλαδή), ανακτάται από το αντίστοιχο dataframe η συστάδα στην οποία ανήκει ο χρήστης, εάν αυτός υπάρχει στο αρχείο BX-Users.csv. Σε περίπτωση που δεν υπάρχει, εκτυπώνεται αντίστοιχο μήνυμα.

    # List to be filled with book entries

    # and their combined scores

    books\_list = []

    users\_cluster = -1

    # Get user's cluster

    if use\_cluster\_ratings:

        try:

            users\_cluster = cluster\_assigned\_users["Cluster"].\

                loc[cluster\_assigned\_users["User\_ID"] == user\_id].iloc[0]

            print(f"User {user\_id} belongs in cluster {users\_cluster}")

        except:

            print(f"User {user\_id} doesn't exist in database.")

            print("Scores with clustering will be the same as previous scores.")

Το κύριο κομμάτι της μεθόδου αυτής είναι η βασική της επανάληψη, στην οποία υπολογίζεται η συνδυαστική βαθμολογία κάθε βιβλίου και δημιουργούνται εγγραφές βιβλίων. Οι εγγραφές αυτές περιέχουν όλες τις πληροφορίες κάθε βιβλίου, καθώς και την υπολογισμένη βαθμολογία τους. Οι εγγραφές αυτές προστίθενται στην books\_list.

Αρχικά, προσπαθούμε να ανακτήσουμε τη βαθμολογία του χρήστη για το συγκεκριμένο βιβλίο. Σε περίπτωση που δεν το έχει βαθμολογήσει και δε χρησιμοποιούμε clustering, πριν υπολογιστεί η συνδυαστική βαθμολογία, θέτουμε τη βαθμολογία του χρήστη ίση με DEFAULT\_RATING, σταθερά η οποία έχει οριστεί στην αρχή του αρχείου.

Ως τιμή για αυτή τη σταθερά επιλέξαμε να είναι το 5, η μέση τιμή των πιθανών βαθμολογιών δηλαδή. Επιλέξαμε την τιμή αυτή επειδή θέλουμε τα βιβλία που έχει βαθμολογήσει αρνητικά ο χρήστης (δηλαδή με βαθμολογία κάτω από 5) να εμφανίζονται πιο κάτω στην κατάταξη από ότι αυτά που δεν έχει βαθμολογήσει ακόμα. Αντίστοιχα, τα βιβλία στα οποία έχει βάλει θετική βαθμολογία, προωθούνται πιο ψηλά στην κατάταξη.

    # Iterate through the documents of the ES reply

    for book in books:

        isbn = book["\_source"]["isbn"]

        norm\_es\_score = 10\*book["\_score"]/max\_score

        # User has rated book

        try:

            # Get user's book rating and typecast it to float (from string)

            user\_rating = float(user\_ratings\_df.\

                loc[user\_ratings ["isbn"] == isbn]["rating"].iloc[0])

        # User has not rated this book

        except:

            # User doesn't exist in the database or function has

            # been called with arg use\_cluster\_ratings = False

            if users\_cluster == -1:

                user\_rating = DEFAULT\_RATING

            else:

                user\_rating = getAvgClusterRating(

                    users\_cluster, isbn, avg\_clust\_ratings)

        # Combined score will be calculated using combinedScoreFunc

        score = combinedScoreFunc(norm\_es\_score, user\_rating)

        # Create a new book entry as a list

        new\_book = [

            score, user\_rating, norm\_es\_score, isbn,

            book['\_source']["book\_title"],

            book['\_source']["book\_author"],

            book['\_source']["year\_of\_publication"],

            book['\_source']["publisher"],

            book['\_source']["summary"],

            book['\_source']["category"]

            ]

        books\_list.append(new\_book)

Για τον υπολογισμό της συνδυαστικής βαθμολογίας, χρησιμοποιείται η μέθοδος combinedScoreFunc, η οποία παίρνει σαν είσοδο το score της Elasticsearch (κλιμακωμένο στο διάστημα [0, 10]), καθώς και τη βαθμολογία του χρήστη.

Στη βαθμολογία του χρήστη έχουμε προσθέσει ένα βάρος, ώστε να έχει μεγαλύτερη σημασία από ότι το σκορ της Elastic. Ο λόγος που δε σκεφτήκαμε την αντίθετη περίπτωση είναι επειδή η Elastic επιστρέφει μόνο σχετικά βιβλία στα οποία έχει βρει τους όρους αναζήτησης. Πειραματιστήκαμε λίγο με τη σταθερά αυτή και καταλήξαμε στο βάρος 1.25 για τη βαθμολογία του χρήστη.

Η συνδυαστική βαθμολογία προκύπτει από το σταθμισμένο άθροισμα των 2 εισόδων, κλιμακωμένο στο διάστημα [0, 10] και επιστρέφεται από τη μέθοδο.

def combinedScoreFunc(norm\_es\_score: float, user\_rating: float) -> float:

    """Function that accepts as inputs the elastic search score of a book,

    as well as the user's rating of the book and returns a combined score."""

    # Add the normalized and weighted scores and scale them in the range [0, 10]

    return (USER\_R\_WEIGHT \* user\_rating + norm\_es\_score) / (1 + USER\_R\_WEIGHT)

Αφού τελειώσει η επανάληψη και έχουμε προσθέσει όλες τις εγγραφές των σχετικών βιβλίων στην books\_list, δημιουργούμε ένα DataFrame με τις εγγραφές αυτές, προσθέτοντας και τα ονόματα των αντίστοιχων στηλών τους. Τέλος, τα ταξινομούμε και επιστρέφουμε μόνο τις 10% καλύτερες αντιστοιχίσεις, σύμφωνα με τις συνδυαστικές τους βαθμολογίες.

    # Create a new dataframe from books\_list and sort it by score

    best\_matches = pd.DataFrame(data=books\_list, columns=[

        "score", "user\_rating", "es\_scaled\_score", "isbn", "book\_title",

        "book\_author", "year\_of\_publication", "publisher", "summary",

        "category"]).sort\_values(by="score", ascending=False)

    # Only keep the best 10% documents

    return best\_matches.head(len(best\_matches.index)//10)

# Ερώτημα 2 – Clustering

### Προεπεξεργασία του users dataset

Πριν ξεκινήσουμε το clustering, πρέπει να γίνει μία προεπεξεργασία του Users dataset. Για αυτόν το σκοπό χρησιμοποιούμε τη μέθοδο processUsersCSV.

    # Create a CSV containing users sorted by their country

    # Entries that have a high chance of being fake are ignored

    print("Creating processed users CSV...")

    processed\_users = functions.processUsersCSV()

    processed\_users.to\_csv(PROC\_USERS, index=False)

Αρχικά, η μέθοδος διαβάζει το αρχείο BX-Users.csv, το φορτώνει σε ένα dataframe και διατρέχει τις γραμμές του, εξάγοντας το uid του χρήστη, την ηλικία του, τη χώρα του και το μοναδικό id της. Για τον κάθε χρήστη, αυτές οι τιμές αποθηκεύονται σε μία λίστα η οποία με τη σειρά της προστίθεται στη λίστα χρηστών users. Τελικά, δημιουργείται ένα dataframe με τους χρήστες, ορίζοντας τα κατάλληλα ονόματα για την κάθε στήλη.

Σε περίπτωση που ο χρήστης δεν έχει συμπληρώσει ηλικία ή ηλικία που έχει συμπληρώσει δεν ανήκει σε ένα φυσιολογικό διάστημα [0, 120], του αναθέτουμε μία καινούρια τιμή. Επιλέξαμε αυτή η τιμή να είναι η μέση τιμή του διαστήματος, δηλαδή 60. Άλλη ανάθεση που θα μπορούσε να γινόταν θα ήταν να διαλέγαμε τυχαία έναν αριθμό στο διάστημα αυτό. Σε κάθε περίπτωση, δεν μπορούμε να αφήσουμε κενή την ηλικία, καθώς ο αλγόριθμος του clustering δε δέχεται μη αριθμητικές τιμές.

Η χώρα του εξάγεται με τη χρήση μίας κανονικής έκφρασης η οποία ουσιαστικά διαβάζει όλους τους χαρακτήρες μετά το τελευταίο κόμμα του αλφαριθμητικού location.

Το μοναδικό id της κάθε χώρας είναι ουσιαστικά η σειρά εμφάνισής της στο dataset μας, με εξαίρεση της κενές εγγραφές. Για κάθε εγγραφή που έχει κενή χώρα, το id της χώρας υπολογίζεται ως συνάρτηση του uid, αυξημένο κατά 1000. Το 1000 επιλέχθηκε επειδή, τουλάχιστον για το dataset μας, εγγυάται ότι δε θα υπάρξουν συγκρούσεις με τα ids άλλων χωρών.

Η στήλη uid αποθηκεύεται όπως είναι, αφού γίνει μετατροπή του αλφαρηθμιτικού σε ακέραιο.

def processUsersCSV() -> pd.DataFrame:

    """Extract vital information from BX-Users csv and save it to a new CSV."""

    # Create empty users dictionary

    users = []

    countries = []

    # Iterate BX-Users.csv rows

    for entry in pd.read\_csv(USERS).iterrows():

        # Cast age to int

        uid = int(entry[1][0])

        # Get location

        location = entry[1][1]

        # Try to cast age to int. If it's empty or

        # out of bounds, set it to the default age

        try:

            age = int(entry[1][2])

            if age > MAX\_AGE or age < 0:

                age = DEF\_AGE

        except:

            age = DEF\_AGE

        # Extract country from location string

        country = re.findall(r"[\s\w+]+$", location)

        # If country is empty, set country\_id to 1000+uid, so that users with

        # incomplete countries do not belong in the same "country". Essential

        # for clustering! Also, 500 is arbitrarily chosen butis empirically

        # higher than the count of normal countries, so there are no conflicts.

        if not country:

            country = ""

            country\_id = 1000 + uid

        else:

            country = country[0][1:]

        # If country isn't already in the countries list,

        # its index will be equal to the length of countries

        if country not in countries:

            country\_id = len(countries)

            countries.append(country)

        # Otherwise, its index is equal to the index of the country in countries

        else:

            country\_id = countries.index(country)

        users.append([uid, age, country, country\_id])

    users\_df = pd.DataFrame(

        users, columns=["User\_ID", "Age", "Country", "Country\_ID"]

    )

    return users\_df

### Αλγόριθμος clustering

Στην εργασία, ζητείται ως αλγόριθμος ομαδοποίησης να χρησιμοποιηθεί ο Kmeans. Παρόλα αυτά, ο απλός Kmeans δέχεται **μόνο** αριθμητικές ιδιότητες (features). Πιο συγκεκριμένα, ο αλγόριθμος …………..\*περιγραφή Kmeans\*

Όμως, στην πραγματικότητα, η ιδιότητα της χώρας δεν αποτελεί αριθμητική ιδιότητα, αλλά κατηγορηματική. Αυτό σημαίνει πως δεν έχει νόημα να χρησιμοποιήσουμε τις αποστάσεις των ids των χωρών ως είσοδο για τον αλγόριθμο ομαδοποίησης. Για παράδειγμα, δεν έχει νόημα να πούμε ότι η χώρα με id 50 έχει μεγαλύτερη απόσταση με τη χώρα με id 1 από ότι έχει με τη χώρα 49. Για το λόγο αυτό, τη χρησιμοποιούμε ως κατηγορηματική ιδιότητα, το οποίο σημαίνει ότι, κατά την ομαδοποίηση, μας ενδιαφέρει μόνο η ισότητα και όχι η απόσταση! Δηλαδή, από τη στιγμή που δεν έχουν το ίδιο id, ο αλγοριθμός θεωρεί ότι όλες οι χώρες «ισαπέχουν».

Έτσι, αποφασίσαμε να χρησιμοποιήσουμε τον αλγόριθμο **Kprototypes**, ο οποίος είναι ουσιαστικά μία παραλλαγή του kMeans που δέχεται τόσο αριθμητικές, όσο και κατηγορηματικές ιδιότητες.

Η είσοδος του αλγορίθμου δημιουργείται με τη μέθοδο getClusteringInput, η οποία διαβάζει το dataset των επεξεργασμένων χρηστών, κρατάει τις χρήσιμες για εκείνη στήλες, δηλαδή το Country\_ID και το Age. Συγκεκριμένα, το age περνάει από τυποποίηση (standardization), μεταφέρονται δηλαδή οι τιμές του στο διάστημα [-1, 1].

Για να αποφασίσουμε τον αριθμό των συστάδων χρησιμοποιήσαμε τη μέθοδο του αγκώνα, την οποία σχεδιάζομε γραφικά με τη μέθοδο plot\_elbow\_curve.

    # Plot elbow curve to help determine optimal number of clusters

    print("\nPlotting elbow curve...")

    clustering.plot\_elbow\_curve(2, 12, 10\_000, processed\_users)

    k = int(input("Choose number of clusters to use: "))

Η μέθοδος τρέχει τον αλγόριθμο σε ένα μικρό υποσύνολο, με μεταβαλλόμενο αριθμό συστάδων k, από 2 μέχρι 12, αποθηκεύοντας το κόστος κάθε τιμή του k. Στη συνέχεια εμφανίζουμε γραφικά τις τιμές αυτές και αποφασίζουμε την τιμή του k για την τελική εκτέλεση του αλγορίθμου.

def plot\_elbow\_curve(start: int, end: int, sample\_size: int,

proc\_users: pd.DataFrame) -> None:

    """Plots elbow curve. Used for optimizing K."""

    data = getClusteringInput(proc\_users)

    if sample\_size > 0 and sample\_size < len(data):

        data = data[:sample\_size]

    categorical\_index = [1]

    no\_of\_clusters = list(range(start, end+1))

    cost\_values = []

    threads = min(PC, cpu\_count())

    print(f"Number of available threads: {cpu\_count()}")

    print(f"Starting testing using {threads} threads...")

    for k in no\_of\_clusters:

        print(f"Testing with {k} clusters...")

        test\_model = KPrototypes(n\_clusters=k, init='Huang', n\_init=N\_INIT, n\_jobs=threads)

        test\_model.fit\_predict(data, categorical=categorical\_index)

        cost\_values.append(test\_model.cost\_)

    seaborn.set\_theme(style="whitegrid", palette="bright", font\_scale=1.1)

    plt.figure(figsize=(15, 7))

    ax = seaborn.lineplot(

        x=no\_of\_clusters, y=cost\_values, marker="o", dashes=False

    )

    ax.set\_title('Elbow curve', fontsize=18)

    ax.set\_xlabel('No of clusters', fontsize=14)

    ax.set\_ylabel('Cost', fontsize=14)

    ax.set(xlim=(start-0.1, end+0.1))

    plt.plot()

    plt.show()