

# PROJECT

## Πολυδιάστατες Δομές Δεδομένων



**ΔΗΜΗΤΡΗΣ ΚΑΤΣΑΝΟΣ ΑΜ: 1093384**

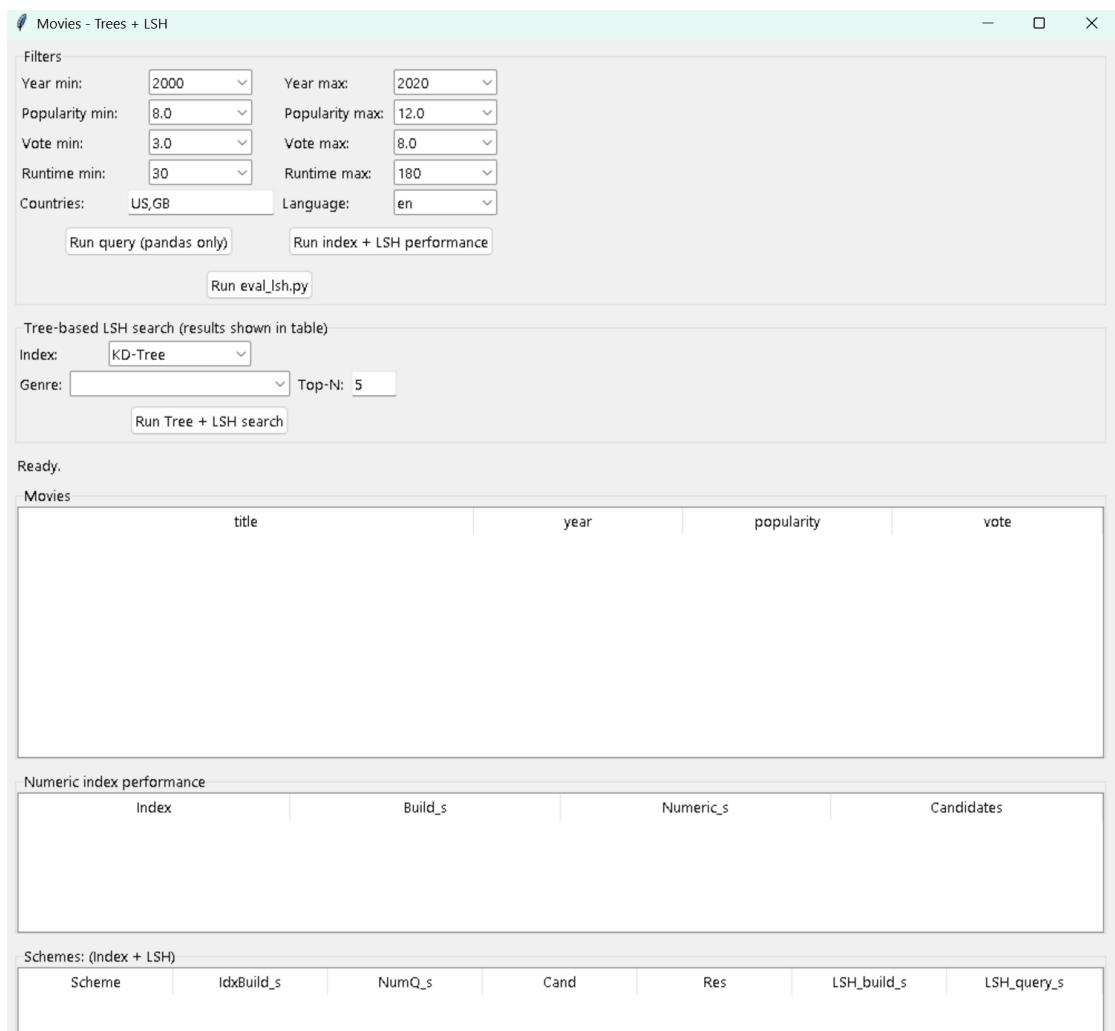
**ΑΝΔΡΕΑΣ ΖΑΦΕΙΡΟΠΟΥΛΟΣ ΑΜ: 1093361**

**ΘΑΛΗΣ-ΧΡΥΣΟΣΤΟΜΟΣ ΤΣΙΩΝΑΣ ΑΜ: 1097467**

**ΙΩΑΝΝΗΣ ΓΡΗΓΟΡΟΠΟΥΛΟΣ ΑΜ: 1097444**

Ασχοληθήκαμε με το Project 1 στο οποίο υλοποιήσαμε ένα σύστημα αποδοτικής αναζήτησης ταινιών, το οποίο υποστηρίζει πολυδιάστατα αριθμητικά ερωτήματα και αναζήτηση ομοιότητας σε κατηγορικά χαρακτηριστικά. Στόχος μας ήταν η μείωση του χρόνου αναζήτησης μέσω κατάλληλων δομών ευρετηρίασης (Indexes) και τεχνικών hashing, σε σύγκριση με απλή σειριακή αναζήτηση, η οποία θα ήταν πολύ αργή λόγο του μεγάλου όγκου δεδομένων. Επίσης δημιουργήσαμε και ένα GUI για την καλύτερη αλληλεπίδραση μας με το Project.

## To GUI



## *Dataset και Προεπεξεργασία*

Το Dataset που κατεβάσαμε από το link της αναφοράς περιλαμβάνει έναν μεγάλο όγκο από πληροφορίες για ταινίες, όπως έτος, popularity, μέσο όρο βαθμολογίας, διάρκεια, χώρες παραγωγής, γλώσσες παραγωγής, genres κ.α. Η φόρτωση και προεπεξεργασία των δεδομένων υλοποιείται στο αρχείο main.py μέσω των συναρτήσεων `load_dataset()` και `preprocess_dataset()`. Σε αυτό το στάδιο γίνεται καθαρισμός των δεδομένων, μετατροπή τύπων και εξαγωγή δομών που διευκολύνουν τις επόμενες φάσεις (π.χ. μετατροπή genres σε σύνολα). Στην συνέχεια το βασικό dataframe δημιουργείται με την `build_base_pool()` το οποίο χρησιμοποιείται τόσο από το main.py όσο και από το GUI (gui\_tk.py).

## **Δομές Ευρετηρίασης (Indexes)**

Για την επιτάχυνση των numeric range queries υλοποιήθηκαν οι τέσσερις δομές ευρετηρίασης που μας ζητήθηκαν και από την εκφώνηση. Συγκεκριμένα KD-Tree (kd\_tree.py), Quad\_tree (quad\_tree.py), Range-Tree (range\_tree.py) και R-Tree (r\_tree.py). Κάθε δομή χρησιμοποιείται για να φιλτράρει γρήγορα τα δεδομένα βάσει αριθμητικών κριτηρίων και να επιστρέψει ένα σύνολο υποψήφιων αποτελεσμάτων (candidates).

## ***Candidates και Φιλτράρισμα***

Στο GUI, το φιλτράρισμα εκτελείται αρχικά με pandas, ώστε να παραχθεί το βασικό αποτέλεσμα και να υποστηριχθεί η επιλογή genres. Στην συνέχεια, τα indexes χρησιμοποιούνται για αριθμητικό φιλτράρισμα και παραγωγή candidates, με σκοπό τη σύγκριση της απόδοσης τους.

## **Αναζήτηση ομοιότητας με LSH**

Για την αναζήτηση ομοιότητας βάση genres χρησιμοποιείται LSH, υλοποιημένο στο lsh\_text.py. Η μέθοδος βασίζεται σε MinHash signatures και επιτρέπει την αποδοτική εύρεση παρόμοιων ταινιών χωρίς πλήρη σύγκριση όλων των στοιχείων.

## **Συνδυαστικά Σχήματα και Αξιολόγηση**

Το σύστημα αξιολογείται μέσω συνδυαστικών σχημάτων της μορφής Index + LSH. Η πειραματική αξιολόγηση υλοποιείται με την συνάρτηση evaluate\_indexes() στο main.py και παρουσιάζει χρόνους build, χρόνους query, πλήθος candidates και τελικά αποτελέσματα.

## kNN

Επιπλέον υλοποιήθηκαν kNN queries στο KD-Tree στο αρχείο kd\_tree.py. Τα kNN queries αξιολογούνται μέσω του main.py, συγκρίνοντας τα αποτελέσματα και τον χρόνο εκτέλεσης με μια απλή προσέγγιση που υπολογίζει τις αποστάσεις από όλα τα σημεία του dataset (brute-force).

```
# kNN
def knn_query(
    self,
    query_point: Sequence[float],
    k: int = 5,
    return_distances: bool = False,
) -> List[int] | List[Tuple[int, float]]:
    """Επιστρέφει τους k κοντινότερους γειτονες του query_point (Euclidean).
    Αν return_distances=True επιστρέφει (index, distance), άλλως μόνο indices."""
    if self.root is None or self.k == 0 or k <= 0:
        return []
    if len(query_point) != self.k:
        raise ValueError("query_point must have length equal to k dimensions")
    k = min(k, self.n_points)
    # max-heap με (-dist_sq, index) για να κρατάμε toys k kalyterous
    best: List[Tuple[float, int]] = []
    self._knn_search(self.root, query_point, k, best)
    # taksinomisi apo pio kontino se mio makrino
    best_sorted = sorted((-d2, idx) for d2, idx in best), key=lambda x: x[0]
    if return_distances:
        return [(idx, dist ** 0.5) for dist, idx in best_sorted]
    return [idx for dist, idx in best_sorted]

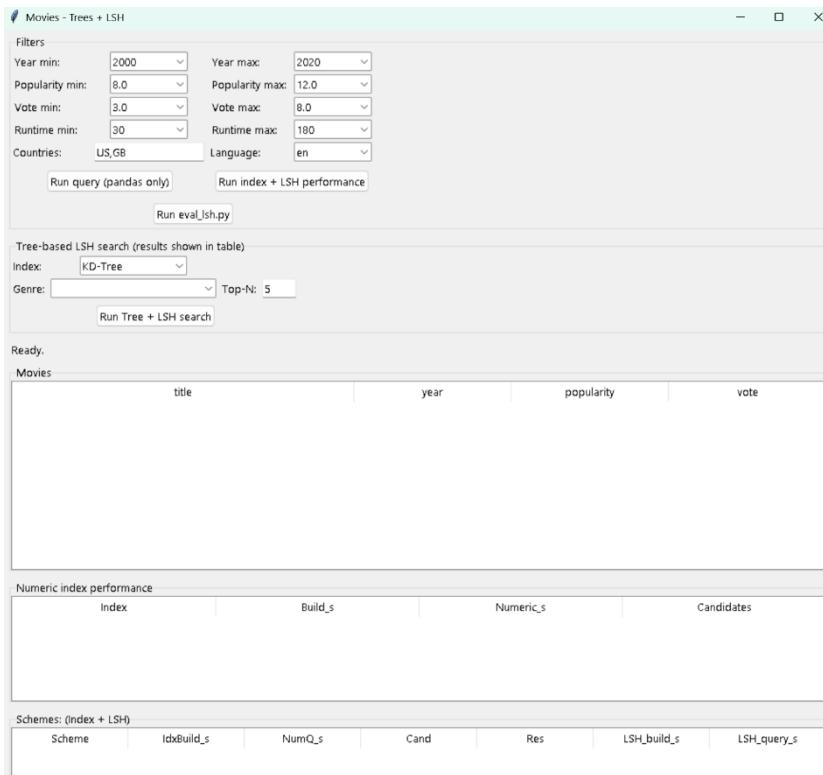
def _knn_search(
    self,
    node: Optional[KDNode],
    query_point: Sequence[float],
    k: int,
    best: List[Tuple[float, int]],
) -> None:
    """Αναδρομική αναζήτηση kNN με pruning."""
    if node is None:
        return
    # apostasi tou node.point apo query
    d2 = 0.0
    p = node.point
    for dim in range(self.k):
        diff = p[dim] - query_point[dim]
        d2 += diff * diff
    # enimerosi heap
    if len(best) < k:
        heapq.heappush(best, (-d2, node.index))
    else:
        # an briame kalytero apo ton xeirotero, antikatastasi
        worst_neg_d2, _ = best[0]
        if -d2 > worst_neg_d2:
            heapq.heapreplace(best, (-d2, node.index))
    axis = node.axis
    diff_axis = query_point[axis] - p[axis]
    # prwta pame sto pio kontino ypodentro
    near = node.left if diff_axis < 0 else node.right
    far = node.right if diff_axis < 0 else node.left
    self._knn_search(near, query_point, k, best)
    # pruning
    if len(best) < k:
        self._knn_search(far, query_point, k, best)
    else:
        worst_d2 = -best[0][0]
        if diff_axis * diff_axis <= worst_d2:
            self._knn_search(far, query_point, k, best)
```

## GUI

Το GUI επιτρέπει την εκτέλεση baseline queries, index-based queries και πειραματικής αξιολόγησης. Χρησιμοποιείται για επίδειξη και σύγκριση απόδοσης των μεθόδων, χωρίς να επηρεάζει την βασική λογική του συστήματος.

Ακολουθεί από κάτω ένα παράδειγμα με την λειτουργικότητα του GUI:

### 1. Αρχική Κατάσταση GUI



Ορίζουμε τα αριθμητικά φίλτρα του ερωτήματος (έτος, popularity, βαθμολογία, διάρκεια, χώρα και γλώσσα) πριν την εκτέλεση της αναζήτησης

## 2. Εκτέλεση pandas-only query

Movies - Trees + LSH

Filters

Year min: 2000 Year max: 2020  
Popularity min: 8.0 Popularity max: 12.0  
Vote min: 3.0 Vote max: 8.0  
Runtime min: 30 Runtime max: 180  
Countries: US,GB Language: en

Run query (pandas only) Run index + LSH performance  
Run eval\_lsh.py

Tree-based LSH search (results shown in table)

Index: KD-Tree  
Genre: Top-N: 5  
Run Tree + LSH search

Loading dataset...

Movies

title	year	popularity	vote
-------	------	------------	------

Numeric index performance

Index	Build_s	Numeric_s	Candidates
-------	---------	-----------	------------

Schemes: (Index + LSH)

Scheme	IdxBuild_s	NumQ_s	Cand	Res	LSH_build_s	LSH_query_s
--------	------------	--------	------	-----	-------------	-------------

Εκτελούμε αναζήτηση με χρήση pandas, ώστε να εφαρμοστούν όλα τα φίλτρα και να παραχθεί το βασικό αποτέλεσμα (ground truth).

### 3. Αποτελέσματα pandas-only

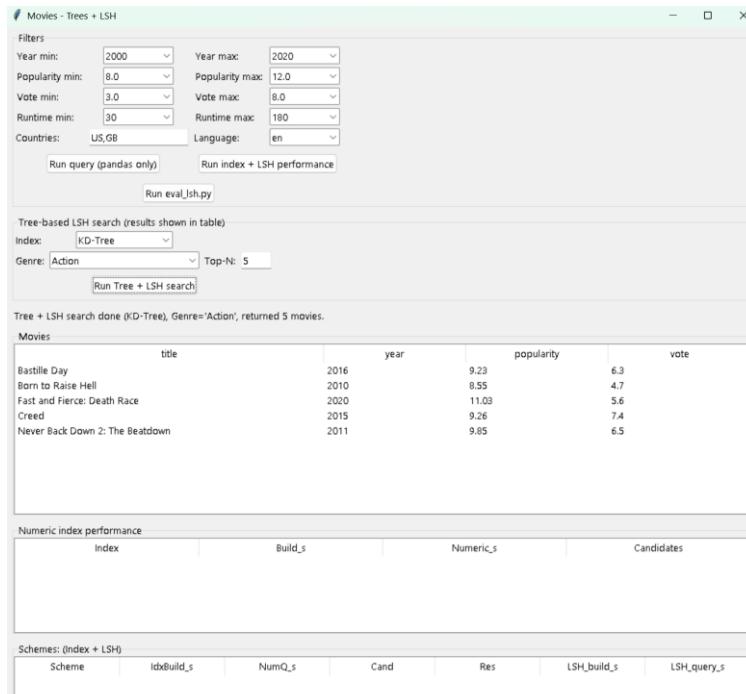
The screenshot shows a software application window titled "Movies - Trees + LSH". The interface includes the following sections:

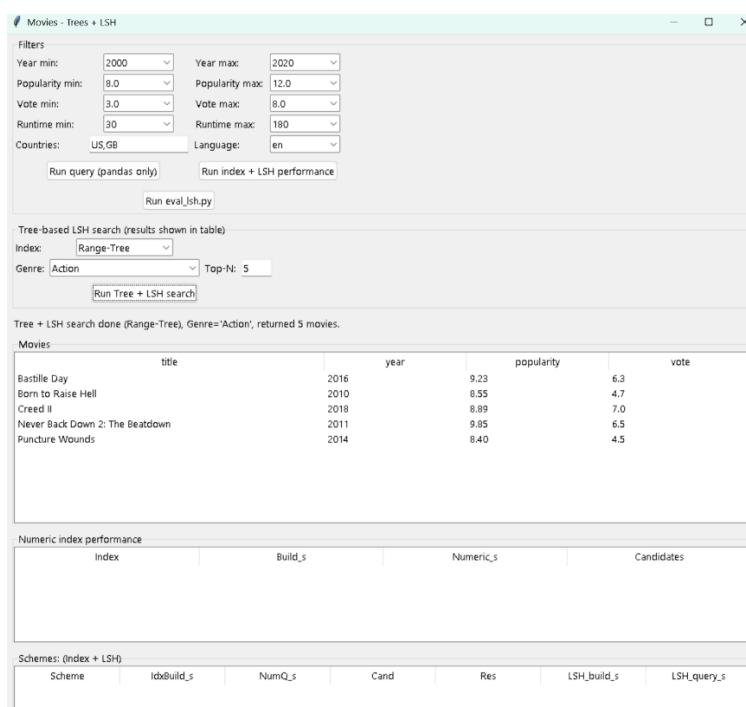
- Filters:** Includes dropdowns for Year min (2000), Year max (2020), Popularity min (8.0), Popularity max (12.0), Vote min (3.0), Vote max (8.0), Runtime min (30), Runtime max (180), Countries (US,GB), and Language (en). Buttons for "Run query (pandas only)", "Run index + LSH performance", and "Run eval\_lsh.py" are present.
- Tree-based LSH search (results shown in table):** Includes dropdowns for Index (KD-Tree), Genre (Action), and Top-N: 5. A button for "Run Tree + LSH search" is also here.
- Movies:** A table showing movie details with columns: title, year, popularity, and vote. The data includes:

title	year	popularity	vote
The Emperor's New Groove	2000	10.97	7.6
Scary Movie 2	2001	9.61	5.8
Final Destination	2000	10.99	6.6
Scary Movie	2000	10.63	6.4
The Mummy Returns	2001	8.59	6.4
Mickey's Magical Christmas: Snowed in at the House of Mouse	2001	8.98	6.9
Child Star: The Shirley Temple Story	2001	9.19	6.7
Legally Blonde	2001	8.64	6.8
American Psycho	2000	8.38	7.4
What Lies Beneath	2000	9.64	6.4
- Numeric index performance:** A table with columns: Index, Build\_s, Numeric\_s, and Candidates. The "Index" column is currently empty.
- Schemes: (Index + LSH):** A table with columns: Scheme, IdxBUILD\_s, NumQ\_s, Cand, Res, LSH\_build\_s, and LSH\_query\_s. The "Scheme" column is currently empty.

Εμφανίζουμε τα αποτελέσματα της baseline αναζήτησης χωρίς χρήση index, τα οποία χρησιμοποιούμε ως σημείο αναφοράς και για την επιλογή genre.

## 4. Tree + LSH search

The screenshot shows the 'Movies - Trees + LSH' application window. At the top, there are filters for Year min (2000), Year max (2020), Popularity min (8.0), Popularity max (12.0), Vote min (3.0), Vote max (8.0), Runtime min (30), Runtime max (180), Countries (US, GB), and Language (en). Below the filters are buttons for 'Run query (pandas only)', 'Run index + LSH performance', and 'Run eval\_lsh.py'. A section titled 'Tree-based LSH search (results shown in table)' shows an index set to 'KD-Tree', genre set to 'Action', and Top-N set to 5. A button 'Run Tree + LSH search' is present. The results table lists 5 movies: Bastille Day (2016, 9.23, 6.3), Born to Raise Hell (2010, 8.55, 4.7), Fast and Fierce: Death Race (2020, 11.03, 5.6), Creed (2015, 9.26, 7.4), and Never Back Down 2: The Beatdown (2011, 9.85, 6.5). Below the table is a section for 'Numeric index performance' with tabs for Index, Build\_s, Numeric\_s, and Candidates. A 'Schemes: (Index + LSH)' table shows columns for Scheme, IdxBuild\_s, NumQ\_s, Cand, Res, LSH\_build\_s, and LSH\_query\_s.

The screenshot shows the same application window but with a different index type. The 'Index' dropdown is now set to 'Range-Tree'. The rest of the interface, including filters, search parameters, and the results table, remains identical to the first screenshot. The results table lists 5 movies: Bastille Day (2016, 9.23, 6.3), Born to Raise Hell (2010, 8.55, 4.7), Creed II (2018, 8.89, 7.0), Never Back Down 2: The Beatdown (2011, 9.85, 6.5), and Puncture Wounds (2014, 8.40, 4.5).

Εκτελούμε αναζήτηση χρησιμοποιώντας διαφορετικά είδη Tree για το αριθμητικό φιλτράρισμα και LSH για αναζήτηση ομοιότητας με βάση το επιλεγμένο genre. Στα δύο αυτά screenshots δείχνουμε ότι με 2 διαφορετικά αλλά στο ίδιο genre, επιστρέφει διαφορετικά αποτελέσματα μεταξύ τους. Κάτι το οποίο είναι λογικό, καθώς κάθε index επιστρέφει διαφορετικό σύνολο υποψήφιων αποτελεσμάτων πάνω στο οποίο εφαρμόζεται το LSH.

## 5. Εκτέλεση Index + LSH performance

The screenshot shows a software interface for movie search and indexing. At the top, there are filters for Year (min: 2000, max: 2020), Popularity (min: 8.0, max: 12.0), Vote (min: 3.0, max: 8.0), Runtime (min: 30, max: 180), Countries (US, GB), and Language (en). Buttons include 'Run query (pandas only)', 'Run index + LSH performance' (highlighted in blue), and 'Run eval\_lsh.py'. Below this is a section for 'Tree-based LSH search (results shown in table)' with an 'Index' dropdown set to 'Range-Tree' and a 'Genre' dropdown set to 'Action'. A 'Top-N: 5' button is also present. A 'Run Tree + LSH search' button is at the bottom. The main area displays a table titled 'Movies' with columns: title, year, popularity, and vote. The data is as follows:

title	year	popularity	vote
Bastille Day	2016	9.23	6.3
Born to Raise Hell	2010	8.55	4.7
Creed II	2018	8.89	7.0
Never Back Down 2: The Beardown	2011	9.85	6.5
Puncture Wounds	2014	8.40	4.5

Below the table is a section for 'Numeric index performance' with tabs: Index, Build\_s, Numeric\_s, and Candidates. At the bottom, a 'Schemes: (Index + LSH)' section lists tabs: Scheme, IdxBUILD\_s, NumQ\_s, Cand, Res, LSH\_build\_s, and LSH\_query\_s.

Εκτελούμε την διαδικασία αξιολόγησης της απόδοσης για όλα τα indexes και τα συνδυαστικά σχήματα Index + LSH.

## 6. Numeric Index Performance Table και Schemes: (Index + LSH) Table

Movies - Trees + LSH

**Filters**

Year min:	2000	Year max:	2020
Popularity min:	8.0	Popularity max:	12.0
Vote min:	3.0	Vote max:	8.0
Runtime min:	30	Runtime max:	180
Countries:	US,GB	Language:	en

**Run query (pandas only)**    **Run index + LSH performance**

**Run eval\_lsh.py**

**Tree-based LSH search (results shown in table)**

**Index:** Range-Tree    **Genre:** Action    **Top-N:** 5

**Run Tree + LSH search**

**Index + LSH performance done.** Common LSH on GT (|GT|=388): build=0.0296s, query=0.000164s

**Movies**

title	year	popularity	vote
Bastille Day	2016	9.23	6.3
Born to Raise Hell	2010	8.55	4.7
Creed II	2018	8.89	7.0
Never Back Down 2: The Beardown	2011	9.85	6.5
Puncture Wounds	2014	8.40	4.5

**Numeric index performance**

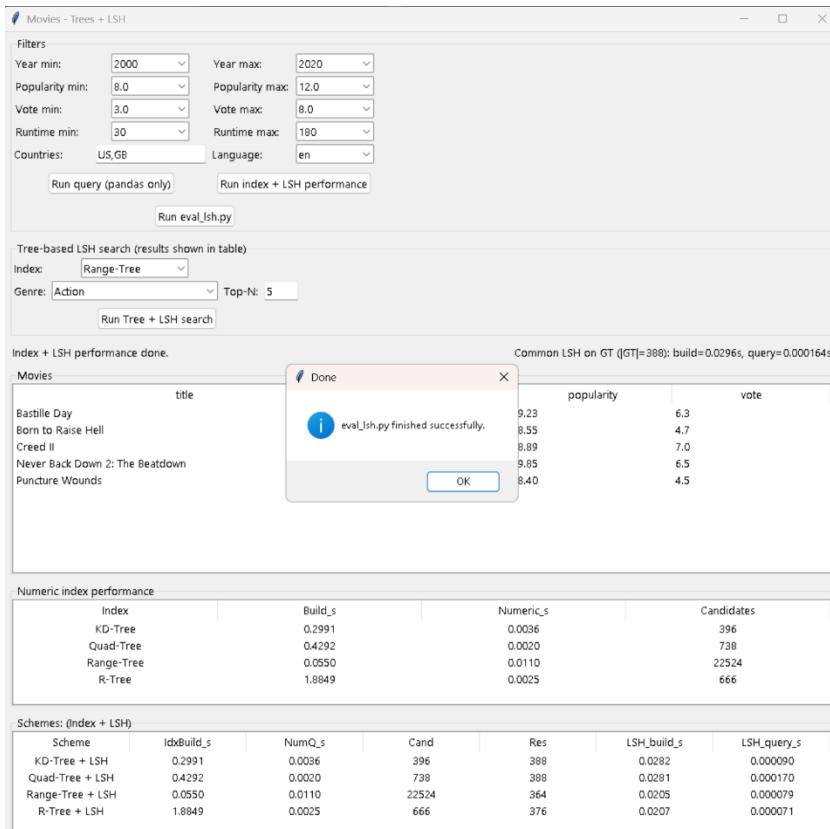
Index	Build_s	Numeric_s	Candidates
KD-Tree	0.2991	0.0036	396
Quad-Tree	0.4292	0.0020	738
Range-Tree	0.0550	0.0110	22524
R-Tree	1.8849	0.0025	666

**Schemes: (Index + LSH)**

Scheme	IdxBuild_s	NumQ_s	Card	Res	LSH_build_s	LSH_query_s
KD-Tree + LSH	0.2991	0.0036	396	388	0.0282	0.000090
Quad-Tree + LSH	0.4292	0.0020	738	388	0.0281	0.000170
Range-Tree + LSH	0.0550	0.0110	22524	364	0.0205	0.000079
R-Tree + LSH	1.8849	0.0025	666	376	0.0207	0.000071

Παρουσιάζουμε συγκριτικά την απόδοση των Indexes και των συνδυαστικών σχημάτων Index + LSH, ως προς τον χρόνο κατασκευής, τον χρόνο εκτέλεσης των αριθμητικών queries, τον αριθμό των υποψήφιων αποτελεσμάτων και τους χρόνους των LSH.

## 7. Εκτέλεση αξιολόγησης LSH (eval\_lsh.py)



Τέλος με το κουμπί "Run eval\_lsh.py" εκτελούμε το συγκεκριμένο αρχείο, το οποίο εκτελεί ανεξάρτητη αξιολόγηση της λειτουργικότητάς του LSH και επιβεβαιώνει την σωστή εκτέλεση του.