**Σύνδεση στη βάση δεδομένων**

1 def connect\_to\_mongodb(MONGO\_HOST, MONGO\_PORT, MONGO\_DB, MONGO\_USER, MONGO\_PASSWORD, colllection\_name):

2 client = MongoClient(MONGO\_HOST, MONGO\_PORT)

3 db = client[MONGO\_DB]

4 #db.authenticate(MONGO\_USER, MONGO\_PASSWORD)

5 print("Connection Succeed")

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

return db[colllection\_name]

Η παραπάνω συνάρτηση πραγματοποιεί την σύνδεση στη βάση δεδομένων. Τα ορίσματα της συνάρτησης είναι τα host, port, database name, username, password, collection name. H συνάρτηση επιστρέφει το collection της βάσης, έχοντας πραγματοποιήσει τη σύνδεση σε αυτό, έτοιμο για να χρησιμοποιηθεί από τις υπόλοιπες συναρτήσεις, οι οποίες θα παρουσιαστούν παρακάτω. To # μπροστά από το db.authenticate υποδηλώνει ότι η συγκεκριμένη γραμμή έχει μπει σε σχόλιο, καθώς στα παραδείγματα που θα παρουσιαστούν η σύνδεση στη βάση δεν απαιτεί username και password.

Ένα παράδειγμα χρήσης της παραπάνω συνάρτησης είναι το εξής:

mycol = connect\_to\_mongodb('localhost', 27017, 'trajectories', '', '','trajectory')

Στο παραπάνω παράδειγμα το host ισούται με localhost, το port είναι το 27017, το όνομα της βάσης είναι trajectories, η σύνδεση στην βάση δεν απαιτεί username και password και από τη βάση θα χρησιμοποιηθεί το collection objects.

Η παραπάνω γραμμή μπορεί να χρησιμοποιηθεί είτε μέσα σε κάποια άλλη συνάρτηση, είτε το mycol να χρησιμοποιηθεί σαν όρισμα σε κάποια άλλη συνάρτηση.

**Εύρεση k κοντινότερων γειτόνων**

H k\_neighbors συνάρτηση υπολογίζει τους k κοντινότερους γείτονες σε ενός σημείου (x,y) που ικανοποιούν ένα χρονικό περιορισμό, δηλαδή το t βρίσκεται ανάμεσα σε t\_min και t\_max.

Αρχικά η συνάρτηση λαμβάνει ως ορίσματα τα (mycol, x, y, k, t\_min, t\_max) που αντιστοιχούν στα εξής: το mycol αντιστοιχεί στη σύνδεση σε κάποιο collection της βάσης, όπως παρουσιάστηκε παραπάνω, x και y είναι οι συντεταγμένες για τις οποίες πρέπει να βρεθούν οι κοντινότεροι γείτονες. k είναι ο αριθμός των κοντινότερων γειτόνων και t\_min, t\_max το χρονικό διάστημα μέσα στο οποίο θα πραγματοποιηθεί η αναζήτηση των κοντινότερων γειτόνων.

Στη συνέχεια καλείται η παραπάνω συνάρτηση, η οποία πραγματοποιεί τη σύνδεση με τη βάση δεδομένων, όπως παρουσιάζεται παρακάτω:

def k\_neghbors(mycol,x,y,k,t\_min,t\_max):

Η λογική που ακολουθείται είναι η εξής: Πρώτα βρίσκονται όλες οι εγγραφές που ικανοποιούν τον χρονικό περιορισμό δηλαδή βρίσκονται μεταξύ t\_min και t\_max και στη συνέχεια υπολογίζονται οι k κοντινότερες εγγραφές στο x και y. Ο υπολογισμός των εγγραφών που βρίσκονται μεταξύ t\_min και t\_max γίνεται με τις εξής εντολές:

query='{"$and":[{"t":{"$gt":'+str(t\_min)+'}}, {"t":{"$lt":'+str(t\_max)+'}}]}'

k1 = json.loads(query)

cursor = mycol.find(k1)

Στη συνέχεια τα αποτελέσματα αποθηκεύονται στη λίστα documents

documents = []

counter = 0

for document in cursor:

documents.insert(counter, document)

counter += 1

Το επόμενο βήμα είναι για κάθε εγγραφή του documents να υπολογιστεί η Ευκλείδια απόσταση από τις παραμέτρους (x, y). Για να πραγματοποιηθεί η παραπάνω διαδικασία, οι παράμετροι (x, y) αποθηκεύονται σε έναν πίνακα b1 και τα στοιχεία (lon, lat) των εγγραφών του document σε έναν πίνακα b2. Με χρήση της συνάρτησης linalg.norm(b1-b2) του πακέτου numpy της Python υπολογίζεται η ευκλείδια απόσταση και αποθηκεύεται σε μια λίστα distances μαζί με τον αύξων αριθμό της εγγραφής του document.

b1 = np.array((x,y))

distances = []

counter = 0

for doc in documents:

b2 = np.array((doc['coordinates'][0],doc['coordinates'][1]))

dist = np.linalg.norm(b1-b2)

distances.append((counter,dist))

counter += 1

Τέλος, κάνοντας μια ταξινόμηση στη λίστα distances με τις αποστάσεις σε αύξουσα σειρά, μπορεί να γίνει η επιλογή των k πρώτων εγγραφών και με χρήση του αύξων αριθμού πραγματοποιείται εύρεση των εγγραφών στη λίστα documents.

Στην Python, αν το k είναι μεγαλύτερο από το πλήθος των εγγραφών στο distances, θα υπάρξει error. Επομένως, γίνεται και ο έλεγχος των δύο τιμών και αν το k είναι μεγαλύτερο, τότε επιστρέφονται όλες οι εγγραφές:

distances = sorted(distances, key=lambda distance: distance[1])

document = []

if len(distances) > k:

for j in range(k):

document.append(documents[distances[j][0]])

else:

for j in range(len(distances)):

document.append(documents[distances[j][0]])

return document

**Εύρεση k κοντινότερων γειτόνων με μοναδικό id**

Μια παραλλαγή της συνάρτησης που παρουσιάστηκε παραπάνω είναι να βρεθούν οι k κοντινότεροι γείτονες που θα έχουν διαφορετικό id. Πρακτικά αυτό σημαίνει ότι αν για ένα σημείο (x, y) βρεθούν τα 3 κοντινότερα σημεία, αυτά τα σημεία θα πρέπει να μην έχουν τα ίδια id ή διαφορετικά να μην είναι σημεία του ίδιου trajectory.

O αλγόριθμος είναι ίδιος με την παραπάνω συνάρτηση μέχρι το σημείο που γίνεται η ταξινόμηση στη λίστα distances με τη διαφορά ότι στη λίστα distances, αποθηκεύεται και το id της κάθε εγγραφής, εκτός από την απόσταση από το (x, y) και τον αύξων αριθμό.

Στη συνέχεια, έχοντας ταξινομημένη τη λίστα, η λογική που ακολουθείται είναι να βρεθούν τα πρώτα k στοιχεία με διαφορετικό id.

Επομένως για κάθε στοιχείο της λίστας distances ελέγχεται η τελική λίστα document. Αν η λίστα δεν είναι άδεια, τότε συγκρίνονται τα δύο id, αν είναι διαφορετικά τότε το στοιχείο που αντιστοιχεί στη συγκεκριμένη εγγραφή του distances εισάγεται στην τελική λίστα. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να γίνει εισαγωγή των k στοιχείων.

for j in range(len(distances)):

if len(document) < k:

if document != []:

for doc in document:

traj\_object = documents[distances[j][0]]

if traj\_object['id'] != doc['id']:

document.append(documents[distances[j][0]])

else:

document.append(documents[distances[j][0]])

else:

break

return document

**Box Range Query**

Η Box Range συνάρτηση υπολογίζει πόσες εγγραφές υπήρξαν μέσα σε ένα ορθογώνιο ένα συγκεκριμένο χρονικό διάστημα.

Τα ορίσματα της συνάρτησης είναι (x\_min, y\_min) που αντιστοιχούν στην κάτω αριστερά γωνία του ορθογωνίου, (x\_max, y\_max) που αντιστοιχούν στην πάνω δεξιά γωνία του ορθογωνίου και (t\_min, t\_max) που είναι το χρονικό διάστημα από t\_min ως t\_max.

Επομένως, το query ζητάει όλες τις εγγραφές όπου το lon είναι ανάμεσα σε x\_min και x\_max, το lat ανάμεσα σε y\_min και y\_max και παράλληλα ικανοποιείται και ο χρονικός περιορισμός.

Η mongo περιλαμβάνει τον geospatial query operator $geowithin, ο οποίος μπορεί να ακολουθείται από κάποιους άλλους τελεστές όπως $center για κύκλο ή $box για ορθογώνιο και χρησιμοποιείται για να βρίσκονται συντεταγμένες που περιέχονται μέσα σε μια γεωγραφική περιοχή.

def range\_box(mycol, x\_min, x\_max, y\_min, y\_max, t\_min, t\_max):

query = '{"$and":[{"coordinates":{"$geoWithin": {"$box":[['+str(x\_min)+','+str(y\_min)+'] , ['+str(x\_max)+','+str(y\_max)+']]}}},{"t":{"$gt":'+str(t\_min)+'}},{"t":{"$lt":'+str(t\_max)+'}}]}'

k1 = json.loads(query)

cursor = mycol.find(k1)

documents = []

counter = 0

for document in cursor:

documents.insert(counter, document)

counter += 1

return documents

**Box Range query aggregate**

Ο παραπάνω αλγόριθμος μπορεί να εκτελεστεί και ως εξής: Εκτελούνται όλα τα βήματα μέχρι να βρεθούν όλες οι εγγραφές που ικανοποιούν τους περιορισμούς, με τη διαφορά ότι η συνάρτηση περιέχει μια ακόμα παράμετρο. Στο τέλος, η συνάρτηση δεν επιστρέφει όλες τις εγγραφές, αλλά ότι ορίζει η παράμετρος agg. Παρακάτω παρουσιάζεται η περίπτωση όπου το agg είναι ίσο με count και κατά συνέπεια η συνάρτηση επιστρέφει το πλήθος των εγγραφών.

def range\_box(mycol, x\_min, x\_max, y\_min, y\_max, t\_min, t\_max):

query = '{"$and":[{"coordinates":{"$geoWithin": {"$box":[['+str(x\_min)+','+str(y\_min)+'] , ['+str(x\_max)+',' +str(y\_max)+']]}}},{"t":{"$gt":'+str(t\_min)+'}},{"t":{"$lt":'+str(t\_max)+'}}]}'

k1 = json.loads(query)

cursor = mycol.find(k1)

documents = []

counter = 0

for document in cursor:

documents.insert(counter, document)

counter += 1

if agg == "count":

return len(documents)

**Circle Range Query**

Η Cirlcle Range συνάρτηση υπολογίζει πόσες εγγραφές υπήρξαν μέσα σε έναν κύκλο μια συγκεκριμένη χρονική στιγμή.

Ομοίως με το box range query, εκτελείται πρώτα το query που βρίσκει τις εγγραφές που βρίσκονται μέσα στον κύκλο και ικανοποιούν τον χρονικό περιορισμό.

def range\_circle(mycol, x, y, r, t\_min, t\_max):

query='{"$and":[{"coordinates":{"$geoWithin":{"$center": [['+str(x)+','+str(y)+'],'+str(r)+']}}} ,{"t":{"$gt":'+str(t\_min)+'}},{"t":{"$lt":'+str(t\_max)+'}}]}'

k1 = json.loads(query)

cursor = mycol.find(k1)

documents = []

counter = 0

for document in cursor:

documents.insert(counter, document)

counter += 1

return documents

**Εισαγωγή Δεδομένων στη Βάση και Αποθήκευση Στατιστικών Στοιχείων**

Τα δεδομένα που αποθηκεύτηκαν στη βάση δεδομένων και χρησιμοποιήθηκαν στα πειράματα που θα παρουσιαστούν παρακάτω, προέρχονται από το Zenodo Repository. Συγκεκριμένα, οι εγγραφές είναι 4565 και αφορούν τροχιές πλοίων. Κάθε εγγραφή αποτελείται από τα εξής στοιχεία: sourcemmsi, altitude, speedoverground, courseoverground, lon, lat, ts. Τα στοιχεία που αποθηκεύτηκαν στη βάση δεδομένων είναι το sourcemmsi ως id, τα lon και lat που αφορούν τις συντεταγμένες και το ts που είναι το timestamp για κάθε εγγραφή.

H λογική που ακολουθήθηκε είναι η εξής: χωρίζοντας τον χώρο που βρίσκονται τα δεδομένα σε κελιά και αποθηκεύοντας τον αριθμό των εγγραφών που βρίσκονται σε κάθε κελί, κάποια ερωτήματα μπορούν να απαντηθούν με προσπέλαση λιγότερων εγγραφών από τη βάση δεδομένων (των εγγραφών που βρίσκονται στο συγκεκριμένο κελί ή και των γειτονικών κελιών).

Αρχικά ο χρήστης μπορεί να εισάγει των αριθμό των κελιών στον άξονα x και στον άξονα y με τις εντολές:

xcells = int(input("number of xcells: "))

ycells = int(input("number of ycells: "))

και να διαβάσει το csv αρχείο με την εντολή read\_csv του πακέτου pandas.

df = pd.read\_csv('Zenodo\_Datasets/AISData/nari\_dynamic\_sar.csv')

Στη συνέχεια γίνεται εύρεση των μικρότερων και μεγαλύτερων lon και lat και αποθήκευση στις κατάλληλες μεταβλητές.

min\_lon = min(df['lon'])

max\_lon = max(df['lon'])

min\_lat = min(df['lat'])

max\_lat = max(df['lat'])

Πλέον, γνωρίζοντας τo ορθογώνιο μέσα στο οποίο βρίσκονται όλες οι συντεταγμένες, καθώς και τον αριθμό των κελιών x και y, μπορεί να γίνει ο έλεγχος για κάθε εγγραφή έτσι ώστε να βρεθεί το κελί μέσα στο οποίο βρίσκεται.

Ο έλεγχος που πραγματοποιείται για κάθε εγγραφή είναι αν το lon είναι μεγαλύτερο από το min\_lon+x\*(max\_lon-min\_lon)/xcells που ουσιαστικά είναι η αριστερή πλευρά του ορθογωνίου με την αρίθμηση του x να ξεκινάει από το 0 και μικρότερο από το min\_lon+(x+1)\*(max\_lon-min\_lon)/xcells που είναι η δεξιά πλευρά. Ομοίως για το lat κάθε εγγραφής πρέπει να είναι μεγαλύτερο από το min\_lat+y\*(max\_lat-min\_lat)/ycells και μικρότερο από το min\_lat+(y+1)\*(max\_lat-min\_lat)/ycells.

Εφόσον βρεθεί το ορθογώνιο στο οποίο ανήκει η εγγραφή, εισάγεται στη βάση δεδομένων με τις εντολές:

mydict = { "id": str(df['sourcemmsi'][i]), "t": int(df['ts'][i]), "coordinates":[df['lon'][i], df['lat'][i]] }

mycol.insert\_one(mydict)

Τα min και max lon και lat του ορθογωνίου, τα x και y που αναφέρονται στο κελί καθώς και ένας μοναδικός αύξων αριθμός για κάθε κελί αποθηκεύονται σε μια λίστα.

coords\_list.append([y\*xcells+x, lon\_start, lon\_end, lat\_start, lat\_end, x, y])

Όταν ολοκληρωθεί η διαδικασία και έχει πραγματοποιηθεί η εισαγωγή όλων των εγγραφών στη βάση δεδομένων, για κάθε κελί βρίσκεται το άθροισμα όλων των μοναδικών αριθμών που αναφέρονται σε αυτό και έχουν εισαχθεί στη λίστα.

Τέλος, σε ένα αρχείο statistics.txt για κάθε κελί εισάγονται τα εξής στοιχεία: άθροισμα των εγγραφών, min longitude, max longitude, min latitude, max latitude, x, y.

**Εύρεση k κοντινότερων γειτόνων με χρήση στατιστικών στοιχείων**

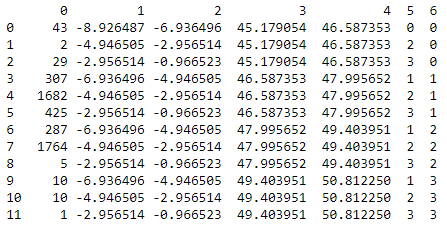
Για την εύρεση των k κοντινότερων γειτόνων θα χρησιμοποιηθεί το .txt αρχείο, στο οποίο έχει αποθηκευτεί ο αριθμός των εγγραφών που βρίσκονται σε κάθε κελί. Με αυτό τον τρόπο η αναζήτηση επικεντρώνεται σε ένα κελί ή και σε κάποια γειτονικά κελιά και ο αριθμός των εγγραφών που θα εξεταστούν είναι μικρότερος από το σύνολο όλων των εγγραφών.

Οι παράμετροι της συνάρτησης είναι το mycol με το οποίο έχει πραγματοποιηθεί η σύνδεση στη βάση δεδομένων, τα x και y που είναι οι συντεταγμένες στις οποίες θα αναζητηθούν οι κοντινότεροι γείτονες, το k που είναι ο αριθμός των κοντινότερων γειτόνων και τα tmin και tmax που είναι τα χρονικό διάστημα μέσα στα οποία θα πρέπει να βρίσκονται οι εγγραφές.

Αρχικά, διαβάζεται το αρχείο στο οποίο βρίσκονται τα δεδομένα και αποθηκεύονται σε ένα dataframe. Αυτό επιτυγχάνεται με την συνάρτηση read\_csv του πακέτου pandas.

df = pd.read\_csv('statistics.txt', header = None)

Τα δεδομένα που αποθηκεύτηκαν στο df έχουν την εξής μορφή:



Η στήλη 0 παρουσιάζει τον αριθμό των εγγραφών που βρίσκονται σε κάθε κελί, οι στήλες 1 και 2 αναφέρονται στα min και max longitude, οι στήλες 3 και 4 στα min και max latitude και οι στήλες 5 και 6 στον αύξων αριθμό του κελιού στον οριζόντιο και κάθετο άξονα αντίστοιχα.

Στη συνέχεια από το df επιλέγεται η εγγραφή για την οποία η παράμετρος x της συνάρτησης είναι μεγαλύτερη από την τιμή της στήλης 1 και μικρότερη από την τιμή της στήλης 2 και αντίστοιχα η παράμετρος y που είναι μεγαλύτερη από την τιμή της στήλης 3 και μικρότερη από την τιμή της στήλης 4. Οι τιμές των στηλών 5 και 6 της εγγραφής που επιλέχθηκε αποθηκεύονται στις μεταβλητές xcell και ycell.

df1 = df.loc[(df[1]<=x) & (df[2]>=x) & (df[3]<=y) & (df[4]>=y)]

xcell = int(df1[5])

ycell = int(df1[6])

Για τη συγκεκριμένη εγγραφή αν η τιμή της στήλης 0 είναι μεγαλύτερη από το k, δηλαδή στο συγκεκριμένο κελί βρίσκονται περισσότερες από k εγγραφές, τότε υπολογίζεται ποια γωνία του ορθογωνίου έχει τη μεγαλύτερη απόσταση από το (x, y).

if sum(df1[0]) >= k:

dist = np.array((x,y))

dist13= np.array((df1[1][df1.index[0]],df1[3][df1.index[0]]))

dist24= np.array((df1[2][df1.index[0]],df1[4][df1.index[0]]))

dist14= np.array((df1[1][df1.index[0]],df1[4][df1.index[0]]))

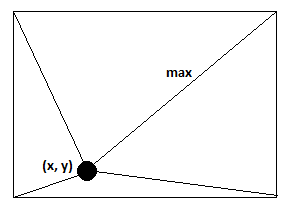
dist23= np.array((df1[2][df1.index[0]],df1[3][df1.index[0]]))

max\_distance = max(np.linalg.norm(dist-dist13),\

np.linalg.norm(dist-dist24),\

np.linalg.norm(dist-dist14),\

np.linalg.norm(dist-dist23))



Μετά την εύρεση της γωνίας με τη μεγαλύτερη απόσταση εκτελείται η range\_circle που παρουσιάστηκε παραπάνω. Η συγκεκριμένη συνάρτηση επιστρέφει όλες τις εγγραφές που βρίσκονται μέσα σε έναν κύκλο με κέντρο το (x, y) και ακτίνα ίση με την μεγαλύτερη απόσταση της γωνίας του ορθογωνίου.

documents = range\_circle(mycol, x, y, max\_distance, tmin, tmax)

Τέλος, για όλες τις εγγραφές που επέστρεψε η range\_circle υπολογίζεται η απόσταση από το (x, y),

τα αποτελέσματα ταξινομούνται από το μικρότερο στο μεγαλύτερο

b1 = np.array((x,y))

distances = []

counter = 0

for doc in documents:

b2 = np.array((doc['coordinates'][0],doc['coordinates'][1]))

dist = np.linalg.norm(b1-b2)

distances.append((counter,dist))

counter += 1

distances = sorted(distances, key=lambda distance: distance[1])

και επιλέγονται τα πρώτα k αποτελέσματα.

document = []

for j in range(k):

document.append(documents[distances[j][0]])

Στην περίπτωση που ο αριθμός των εγγραφών που βρίσκονται στο συγκεκριμένο κελί είναι μικρότερος από k, επιλέγονται όλα τα γειτονικά κελιά και η διαδικασία επαναλαμβάνεται για το ορθογώνιο που θα σχηματιστεί από όλα τα κελιά. Η διαδικασία επαναλαμβάνεται μέχρι ο αριθμός των εγγραφών που βρίσκονται στο ορθογώνιο που σχηματίζεται να είναι μεγαλύτερος από k. Στο παρακάτω παράδειγμα το i παίρνει τιμές από 1 ως 4, καθώς όταν έγινε ο διαχωρισμός του χώρου που βρίσκονται οι συντεταγμένες σε κελιά, θεωρήσαμε ότι ο μεγαλύτερος αριθμός κελιών στον x και στον y άξονα είναι 4.

else:

for i in range(1, 4):

df2 = df.loc[(df[5]>=xcell-i) & (df[5]<=xcell+i) & (df[6]>=ycell-i) & (df[6]<=ycell+i)]

if sum(df2[0]) >= k:

.

.

.

break

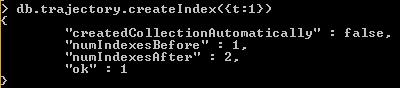
**ΠΕΙΡΑΜΑΤΙΚΗ ΜΕΛΕΤΗ**

Στα πειράματα που πραγματοποιήθηκαν, η μελέτη επικεντρώθηκε στον αριθμό των εγγραφών της βάσης δεδομένων, που πρέπει να προσπελάσει ο κάθε αλγόριθμος προκειμένου να επιστρέψει το τελικό αποτέλεσμα. Προκειμένου να βρεθεί ο αριθμός των εγγραφών που εξετάζει ο κάθε αλγόριθμος χρησιμοποιήθηκε η συνάρτηση explain. Μετά την εκτέλεση της συνάρτησης find, εκτελέστηκε η εντολή print(cursor.explain()['executionStats']). Η συνάρτηση explain μπορεί να πάρει μία από τις 3 παρακάτω παραμέτρους:

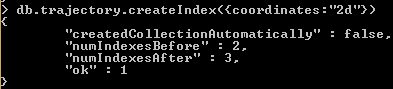
* **queryPlanner:** Η MongoDB εκτελεί ένα query optimizer προκειμένου να αποφασίσει ποιο πλάνο είναι το καλύτερο και επιστρέφει πληροφορίες σχετικά με αυτό.
* **executionStats:** Το καλύτερο πλάνο εκτελείται και επιστρέφονται στατιστικά δεδομένα που περιγράφουν την εκτέλεση του.
* **allPlansExecution:** Το καλύτερο πλάνο εκτελείται και επιστρέφονται στατιστικά δεδομένα για αυτό, όπως επίσης στατιστικά δεδομένα επιστρέφονται και για όλα τα υποψήφια πλάνα.

Όπως είναι προφανές η παράμετρος για την οποία υπάρχει ενδιαφέρον είναι η «executionStats» και συγκεκριμένα ένα πεδίο που ονομάζεται «totalDocsExamined». Η τιμή αυτού του πεδίου δείχνει πόσες εγγραφές προσπέλασε ο αλγόριθμος προκειμένου να βρεθεί το σωστό αποτέλεσμα.

Στα πειράματα πραγματοποιήθηκαν χωρίς τη χρήση index, καθώς επίσης και με τη χρήση index είτε στο πεδίο «t», δηλαδή στο χρόνο είτε στις συντεταγμένες ή και με τη χρήση και των δύο index. Οι εντολές για τη χρήση των index είναι οι εξής:



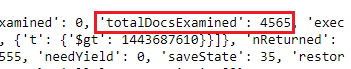
Η τιμή «1» στο t ότι οι τιμές του πεδίου θα ταξινομηθούν σε αύξουσα σειρά, αν η τιμή ήταν «-1», τότε η σειρά θα ήταν φθίνουσα.



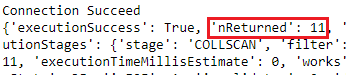
Η τιμή «2d» δείχνει ότι το συγκεκριμένο πεδίο περιέχει ζεύγη τιμών όπως είναι οι συντεταγμένες. Φυσικά, επειδή το συγκεκριμένο index έχει σχεδιαστεί για συντεταγμένες, η πρώτη τιμή, το longitude, πρέπει να βρίσκεται στο διάστημα [-180,180] και το latitude στο [-90, 90]. Σε διαφορετική περίπτωση θα υπάρξει error.

**Εύρεση k κοντινότερων γειτόνων**

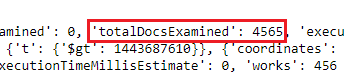
Ο πρώτος αλγόριθμος που μελετήθηκε ήταν ο k\_neighbors χωρίς την χρήση στατιστικών δεδομένων καθώς επίσης και χωρίς την χρήση indexes. Όπως ήταν αναμενόμενο, ο συγκεκριμένος αλγόριθμος προκειμένου να επιστρέψει το τελικό αποτέλεσμα θα έπρεπε να προσπελάσει όλες τις εγγραφές. Οι παράμετροι του αλγόριθμου ήταν οι εξής: (mycol, -3.4032917, 46.91752, 2, 1443687610, 1643689790), δηλαδή αναζητήθηκαν οι 2 κοντινότεροι γείτονες στο σημείο (-3.4032917, 46.91752) που βρίσκονται στο χρονικό διάστημα [1443687610, 1643689790]. Παρακάτω παρουσιάζεται ένα μέρος του αποτελέσματος όπως αυτό παρουσιάστηκε από την συνάρτηση explain.



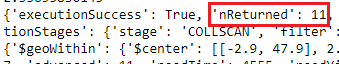
Ο αλγόριθμος εξέτασε 4565 εγγραφές και επέστρεψε μόλις 11.



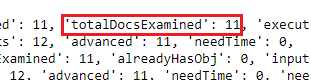
Χωρίς την χρήση indexes και ο δεύτερος αλγόριθμος k\_neighbors\_with\_statistics, που χρησιμοποιεί στατιστικά δεδομένα επέστρεψε το ίδιο αποτέλεσμα, δηλαδή χρειάστηκε να ερευνήσει όλα τα έγγραφα προκειμένου να βρει το σωστό αποτέλεσμα.



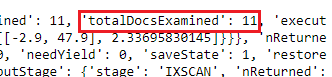
Όπως και ο πρώτος αλγόριθμος, έτσι και ο k\_neighbors\_with\_statistics επιστρέφει 11 αποτελέσματα.



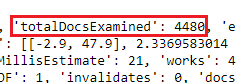
Στην περίπτωση που δημιουργηθεί ένα index στο πεδίο t, δηλαδή στο χρόνο, ο πρώτος αλγόριθμος εξετάζει 11 εγγραφές, αυτές που επιστρέφει.



Ομοίως και ο k\_neighbors\_with\_statistics.

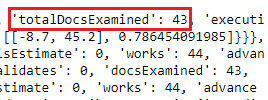


Στην περίπτωση που στον k\_neighbors\_with\_statistics υπάρχει ενδιαφέρον μόνο για το χώρο και κατ συνέπεια εισαχθεί μόνο το “2d” index δεν αλλάζει κάτι, εξετάζονται και πάλι 11 εγγραφές γιατί μόνο με το “2d” index και έχοντας χωρίσει το χώρο σε 4x4 κελιά, οι εγγραφές που εξετάζονται είναι 4480, δηλαδή υπάρχει μια μικρή βελτίωση.

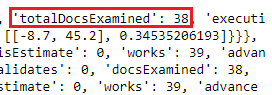


Χωρίς τη χρήση του index στο χρόνο και μόνο με τη χρήση του «2d» index το coordinates, υπάρχει βελτίωση σε σχέση με τις 4480 εγγραφές που εξετάζει ο αλγόριθμος όταν ο χώρος έχει χωριστεί σε 4x4 κελιά, αλλά το αποτέλεσμα δεν είναι καλύτερο από την περίπτωση που υπάρχει το index στο χρόνο.

Όταν ο χώρος έχει χωριστεί σε 10x10 κελιά οι εγγραφές που εξετάζονται είναι 43:



Όταν ο χώρος χωριστεί σε 20x20 κελιά εξετάζονται 38 εγγραφές:

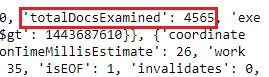


Φυσικά, η βελτίωση είναι πολύ μεγάλη, αλλά τα καλύτερα αποτελέσματα παρουσιάζονται με την εισαγωγή του index στο χρόνο.

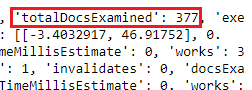
**Circle Range**

Η Circle Range συνάρτηση όπως είναι αναμενόμενο χωρίς κανένα index, θα πρέπει να ελέγξει όλες τις εγγραφές προκειμένου να επιστρέψει κάποιο αποτέλεσμα. Σημαντική βελτίωση υπάρχει με την εισαγωγή index στο χώρο ή στο χρόνο. Φυσικά, μια πρόβλεψη μπορεί να γίνει καθώς στο προηγούμενο ερώτημα, η k\_neighbors\_with\_statistics εκτελεί την circle range. Η παράμετροι της συνάρτησης είναι (mycol, -3.4032917, 46.91752, 0.5, 1443687610, 1643689790), δηλαδή το ερώτημα που κλήθηκε να απαντήσει είναι να βρεθούν όλες οι εγγραφές που βρίσκονται μέσα στον κύκλο με κέντρο (-3.4032917, 46.91752) και ακτίνα 0.5, καθώς επίσης και στο χρονικό διάστημα [1443687610, 1643689790].

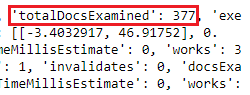
Αρχικά, όπως αναφέρθηκε χωρίς κανένα index η συνάρτηση εξετάζει όλες τις εγγραφές.



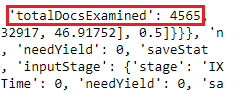
Αν πραγματοποιηθεί εισαγωγή ενός index στο χώρο, τότε η συνάρτηση εξετάζει 377 εγγραφές, δηλαδή η βελτίωση είναι πολύ μεγάλη.



Αν πραγματοποιηθεί και εισαγωγή index στο χρόνο, τότε δεν αλλάζει κάτι, οι εγγραφές που εξετάζονται είναι και πάλι 377. Αυτό σημαίνει ότι μόνο με το ευρετήριο στο χρόνο οι εγγραφές που εξετάζονται είναι περισσότερες.



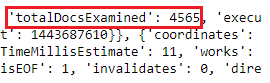
Αν χρησιμοποιούνταν μόνο ευρετήριο στο χρόνο, τότε θα εξεταζότανε είναι 4565. Αυτό συμβαίνει καθώς το χρονικό διάστημα είναι πολύ μεγάλο, επομένως με το ευρετήριο στο χρόνο δεν υπάρχει κανένα όφελος.



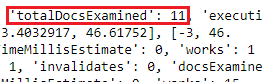
**Box Range**

H Box Range συνάρτηση χρησιμοποιήθηκε με παραμέτρους (mycol, -3.4032917, 46.61752, -3, 46.9, 1443687610, 1443689790), δηλαδή το ορθογώνιο σχηματιζότανε από τις γωνίες (-3.4032917, 46.61752) και (-3, 46.9) και το χρονικό διάστημα μέσα στο οποίο πρέπει να βρίσκονται τα δεδομένα είναι [1443687610, 1443689790].

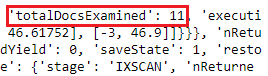
Όπως σε όλα τα παραδείγματα, χωρίς index εξετάζονται όλα τα δεδομένα.



Με χρήση index στο χώρο εξετάζονται 11 εγγραφές.



Στο συγκεκριμένο παράδειγμα τυχαίνει να εξετάζονται 11 εγγραφές αν χρησιμοποιηθεί μόνο το index στο χρόνο.



Συνεπώς, με χρήση index είτε στο χώρο, είτε στο χρόνο η αποδοτικότητα του αλγόριθμου αυξάνεται.

Στο συγκεκριμένο παράδειγμα επιστρέφεται μόνο μία εγγραφή.

