ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΠΜΣ «ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ ΚΑΙΥΠΗΡΕΣΙΕΣ»

ΚΑΤΕΥΘΥΝΣΗ «ΜΕΓΑΛΑ ΔΕΔΟΜΕΝΑ ΚΑΙ ΑΝΑΛΥΤΙΚΗ»

Διαχείριση Δεδομένων για Σχεσιακές και μη Σχεσιακές Βάσεις Δεδομένων

ΕΥΑΓΓΕΛΙΔΑΚΗΣ ΛΕΑΝΔΡΟΣ

Περιεχόμενα

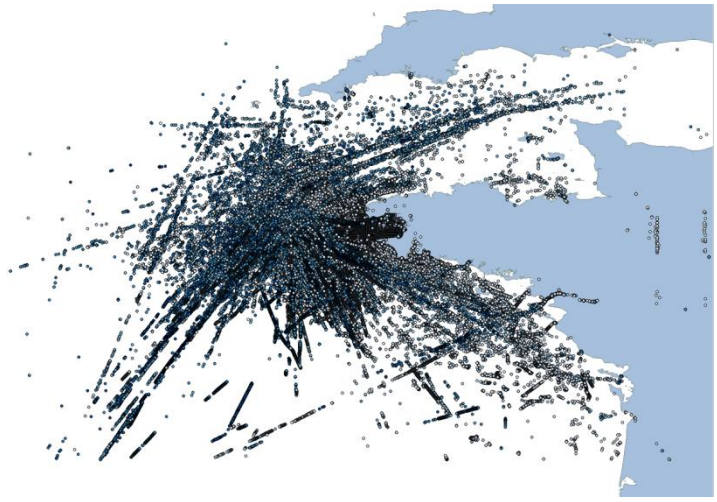
Περιγραφή του προβλήματος

Στην παρούσα εργασία εξετάζουμε τα διαθέσιμα δεδομένα από AIS (Automatic Identification System), για το εξάμηνο από 1η Οκτωβρίου 2015 έως τον Μάρτιο του 2016, που αναφέρονται στην περιοχή της Κελτικής Θάλασσας και του Βισκαϊκού κόλπου. Το επίκεντρο της μελέτης μας είναι η αλιεία. Εξετάζουμε τα αλιευτικά πλοία και κάνουμε συσταδοποίηση ανάλογα με την τροχιά που κινούνται (trajectory clustering) καθώς και σημειακή συσταδοποίηση για να βρούμε περιοχές ψαρέματος (fishing spots clustering). Επίσης εξετάζουμε την παράνομη αλιεία (fishing violations), σε κάποιες περιοχές όπου απαγορεύεται.

Περιγραφή των δεδομένων

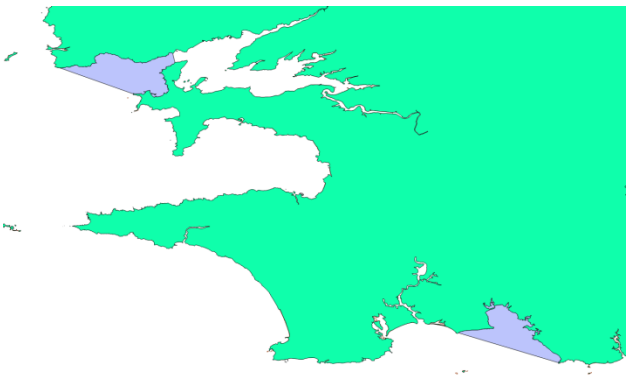
Το σύνολο των δεδομένων περιέχει δεδομένα πλοήγησης, δεδομένα σχετικά με τα πλοία, γεωγραφικά δεδομένα αλλά και περιβαντολλογικά. Τα δεδομένα που εξετάζουμε κυρίως, είναι το σύνολο [P1] AIS Data, που περιέχει μεταξύ άλλων το αρχείο nari_dynamic.csv και το nari_static.csv. Εκεί περιέχονται οι περισσότερες καταχωρήσεις των σημάτων μέσω τουAIS , όπως λαμβάνονται από τους σταθμούς που τα αποθηκεύουν. Το σύστημα AIS χρησιμοποιεί 27 είδη μηνυμάτων (ITU 1,ITU 2 κ.λπ.)

Στο αρχείονari_dynamic.csv υπάρχουν πληροφορίες που στέλνουν αυτόματα τα πλοία κατά τη διάρκεια της δραστηριότητάς τους, και αναφέρονται στην τοποθεσία τους, την κατεύθυνση που κινούνται, την ταχύτητα κλπ. Αυτά αντιστοιχούν στα μηνύματα τύπου ITU 1,2,3,18 και 19 . Ο χρόνος (timestamp) που αναφέρεται σε κάθε μήνυμα αντιστοιχεί στην στιγμή που ο σταθμός τα λαμβάνει και όχι τη στιγμή που τα εκπέμπει το κάθε πλοίο. Επίσης κάθε πλοίο αναφέρει με έναν ακέραιο (navigationalstatus) την κατάστασή του τη στιγμή εκείνη (π.χ. εν κινήσει, αγκυροβολημένο κ.λπ.). Σε αυτή τη μεταβλητή, το 7 αντιστοιχεί στην αλιευτική δραστηριότητα και έτσι μπορούμε και εξάγουμε εύκολα τα δεδομένα που αφορούν την αλιεία.



Εικόνα 1. Τα δεδομένα του nari_dynamic.csv

Στο αρχείο Fishing constraints, περιέχονται δύο περιοχές που απαγορεύεται η αλιεία, στην περιοχή της Βρετάνης. Η πρώτη περιοχή (Camaret, βόρεια), αναφέρεται στο διάστημα από 05/02/2015 έως 21/01/2016, ενώ η δεύτερη (Concarneau, νότια) από 03/09/2015 έως 15/10/2015.



Εικόνα 2. Περιοχές όπου απαγορεύεται η αλιεία

Μέθοδοι και τεχνικές που χρησιμοποιήθηκαν

Η αποθήκευση των δεδομένων γίνεται χρησιμοποιώντας τη μη-σχεσιακή βάση δεδομένων MongoDB, σε συνδυασμό με την Python για την εισαγωγή, εξαγωγή και ανάλυσή τους.

Για την αποθήκευση, επεξεργαζόμαστε ελάχιστα τα δεδομένα του nari_dynamic.csv και τα εισάγουμε στη MongoDB. Αυτό γίνεται χρησιμοποιώντας κυρίως τις βιβλιοθήκες pandas και pymongo της Python. Την ίδια διαδικασία ακολουθούμε και για τα δεδομένα για τις περιοχές με απαγόρευση αλιείας. Και στις δύο περιπτώσεις, κάνουμε χρήση του γεγονότος ότι η Mongo δέχεται έγγραφα τύπου GeoJSON, δηλαδή γεωγραφικά δεδομένα όπως συντεταγμένες (Points), πολύγονα (Polygons) κ.λπ. και γι'αυτό εισάγουμε τα δεδομένα αφότου τα προσαρμόσουμε στη μορφή αυτή.

Καθότι μελετάμε την αλιευτική δραστηριότητα και τον έλεγχο αυτής σε συγκεκριμένες περιοχές (χρονικά εξαρτώμενες), δημιουργούμε ευρετήρια με σκοπό να βελτιστοποιήσουμε την ταχύτητα των διαδικασιών αυτών. Χρησιμοποιούμε απλά ευρετήρια (single indexes) αλλά και "compound" ευρετήρια που δέχεται η MongoDB και πρόκειται για ευρετήρια με πολλαπλές παραμέτρους. Συγκεκριμένα, φτιάχνουμε compound ευρετήριο ως προς το χρόνο και ως προς την τοποθεσία (2dsphere).

Για την ανάλυση, γίνεται προσπάθεια η ανάκτηση των απαιτούμενων δεδομένων να προέρχεται κυρίως από queries στη βάση, καθώς ο αρχικός σκοπός της εργασίας είναι η ενασχόληση με μη-σχεσιακές βάσεις δεδομένων. Έτσι με την Python αντλούμε τα δεδομένα από τη Mongo, με τρόπο τέτοιο ώστε να είναι έτοιμα για την μετέπειτα ανάλυσής τους. Η ανάλυση περιλαμβάνει την παρουσίαση κάποιων βασικών πληροφοριών που εξάγουμε σχετικά με την αλιευτική δραστηριότητα , την συσταδοποίηση των τροχιών των αλιευτικών πλοίων (fishing vessels trajectory clustering), την συσταδοποίηση των μεμονωμένων σημείων αλιείας (fishing spots clustering) και τέλος τον εντοπισμό παράνομης αλιευτικής δραστηριότητας (fishing violations).

Η συσταδοποίηση γίνεται χρησιμοποιώντας αλγορίθμους βελτιστοποιημένους για Big Data. Συγκεκριμένα, χρησιμοποιούμε τους αλγορίθμους KMeans, DBSCAN,OPTICS και BIRCH. Οι DBSCAN και OPTICS είναι μέθοδοι με βάση την πυκνότητα (density based) και χρησιμοποιούνται ευρέως κατά την ανάλυση γεωχωρικών δεδομένων. Ο BIRCH χρησιμοποιεί δενδρικές δομές για τη συσταδοποίηση και εφαρμόζεται γιατί παρέχει εντυπωσιακές ταχύτητες σε μεγάλα δεδομένα (πρόκειται για βελτίωση του MiniBatch-KMeans).

Για την συσταδοποίηση σημείων, οι διαδικασίες είναι απλές καθώς πρόκειται για κλασσικού τύπου συσταδοποίηση.

Η συσταδοποίηση τροχιών διαφέρει αρκετά από την προηγούμενη διαδικασία. Η τροχιά που σχηματίζει ένα πλοίο αποτελείται από τις προηγούμενες τοποθεσίες του σε χρονολογική σειρά. Τα σχήματα που δημιουργούν οι τροχιές είναι αρκετά πολύπλοκα ιδιαίτερα για τα αλιευτικά πλοία. Επίσης, όπως είναι και λογικό, οι τροχιές των αλιευτικών πλοίων διαφέρουν κατά πολύ συγκριτικά με τα υπόλοιπα πλοία. Για να συγκρίνουμε δύο τροχιές, πρέπει να βρούμε έναν τρόπο μέτρησης της ομοιότητας δύο τροχιών, δηλαδή δύο σχημάτων. Ένας από τους τρόπους να το κάνουμε αυτό είναι η απόσταση του Hausdorff που μετράει την απόσταση μεταξύ δύο υποσυνόλων ενός μετρικού χώρου εξετάζοντας τις αποστάσεις μεταξύ των επιμέρους στοιχείων τους. Έτσι χρησιμοποιούμε αυτή την απόσταση ως είσοδο στους αλγορίθμους συσταδοποίησης , κατασκευάζοντας χειροκίνητα τον πίνακα των αποστάσεων.

Για τον εντοπισμό παράνομης αλιευτικής δραστηριότητας, κάνουμε γεωχωρικά queries για συγκεκριμένα χρονικά διαστήματα, κάνοντας χρήση της δυνατότητας της MongoDB να εντοπίζει σημεία που βρίσκονται σε κάποιο γεωμετρικό σχήμα (π.χ. Polygons). Με την υποστήριξη και από το compoundευρετήριο που δημιουργούμε, τέτοιου είδους queries εκτελούνται σε ικανοποιητικό χρόνο.

Εισαγωγή δεδομένων στη MongoDB

Αρχικά δημιουργούμε μια βάση δεδομένων ‘AISdata’ στη MongoDB. Η εισαγωγή των επιμέρους collections γίνεται από την Python μέσω της βιβλιοθήκης pymongo. Για την εισαγωγή των δεδομένων, διαβάζουμε τα δεδομένα του nari_dynamic ως dataframe μέσω της βιβλιοθήκης pandas. Παρακάτω φαίνεται ο αντίστοιχος κώδικας και οι πέντε πρώτες γραμμές των δεδομένων

```
import pymongo
from pymongo import MongoClient
import pandas as pd
import seaborn as sns
import geopandas as geo
import pprint
```

```
client = MongoClient()
client = MongoClient('localhost', 27017)
db = client.AISdata
nari_dynamic = db.nari_dynamic
```

```
data = pd.read_csv("../data/[P1] AIS data/nari_dynamic.csv")
data.head()
```

	sourcemmsi	navigationalstatus	rateofturn	speedoverground	courseoverground	trueheading	lon	lat	t
0	245257000	0.0	0.0	0.1	13.1	36	-4.465718	48.382490	1443650402
1	227705102	15.0	-127.0	0.0	262.7	511	-4.496571	48.382420	1443650403
2	228131600	15.0	-127.0	8.5	263.7	511	-4.644325	48.092247	1443650404
3	228051000	0.0	-127.0	0.0	295.0	511	-4.485108	48.381320	1443650405
4	227574020	15.0	-127.0	0.1	248.6	511	-4.495441	48.383660	1443650406

Όπως βλέπουμε ο χρόνος είναι σε μορφή ακεραίου (UNIX epochs). Καθώς είναι δύσκολη η ανθρώπινη κατανόηση σε αυτή τη μορφή τον μετατρέπουμε στη συνήθη μορφή. Επίσης, θα ενώσουμε το γεωγραφικό μήκος και πλάτος ώστε να το φέρουμε στη μορφή του GeoJSON που αναγνωρίζει η MongoDB. Η μορφή αυτή είναι τύπου: <field>: {type: <GeoJSON type>, coordinates: <coordinates>}. Για παράδειγμα: { type: "Point", coordinates: [40, 5] }. Έπειτα διαβάζουμε γραμμή προς γραμμή το αρχείο και το εισάγουμε στη βάση.

```
data['t']=pd.to_datetime(data.t,unit='s')
```

```

def add_to_db():

    for index, row in data.iterrows():
        point=[row.lon,row.lat]
        doc = {'sourcemmsi':row.sourcemmsi,
               'navigationalstatus':row.navigationalstatus,
               'rateofturn':row.rateofturn,
               'speedoverground':row.speedoverground,
               'courseoverground':row.courseoverground,
               'trueheading':row.trueheading,
               't':row.t,
               'location':{'type':'Point',
                           'coordinates':point}}
        nari_dynamic.insert_one(doc)

    return 'Done'


```

```
add_to_db()
```

```
'Done'
```

Για την εισαγωγή των απαγορευμένων περιοχών εξετάζουμε τα δεδομένα που βρίσκονται στο αρχείο fishing_interdiction.shp. Για να διαβάσουμε το αρχείο χρησιμοποιούμε τη βιβλιοθήκη geopandas. Σε αυτό βρίσκονται δύο καταχωρήσεις, για τις δύο περιοχές αντίστοιχα. Η πρώτη (Camaret) είναι ένα απλό πολύγωνο (Polygon) ενώ η δεύτερη (Concarneau) είναι πολύ-πολύγωνο (MultiPolygon). Για την εισαγωγή ενός πολυγώνου ως GeoJSON γίνεται με τον ίδιο τρόπο που είδαμε προηγουμένως. Η μόνη διαφορά είναι ότι εδώ πρέπει να αναπροσαρμόσουμε τον τρόπο που διαβάζει η geopandas τα δεδομένα, ώστε αντί για tuples να χρησιμοποιεί λίστες, ώστε να μπορούν να εισαχθούν απευθείας στη βάση. Για την περίπτωση του MultiPolygon, παίρνουμε ένα-ένα τα πολύγωνα που περιέχει και επαναλαμβάνουμε την προηγούμενη διαδικασία.

```
forbidden_fishing_areas=geo.read_file("../data/[C5] Fishing Constraints/fishing_interdiction.shp")

#Camaret (Polygon)
x0,y0=forbidden_fishing_areas.geometry[0].exterior.coords.xy
polygon0=list(list(t) for t in zip(x0,y0))

#Concarneau (MultiPolygon)
polygons1=[]
multipolygon=forbidden_fishing_areas.geometry[1]
for i in range(len(multipolygon)):
    x1,y1=multipolygon[i].exterior.coords.xy
    polygon1=list(list(t) for t in zip(x1,y1))
    polygons1.append(polygon1)

```

```
#Insert forbidden areas to mongodb :

forbidden_fishing_areas = db.forbidden_fishing_areas

#Camaret
forbidden_area_0 = {'id':'Camaret',
                    'type':'Polygon',
                    'coordinates':[polygon0]}
forbidden_fishing_areas.insert_one(forbidden_area_0)

#Concarneau
i=0
for poly in polygons1:

    forbidden_area_1 = {'id':'Concarneau'+str(i),
                        'type':'Polygon',
                        'coordinates':[poly]}
    forbidden_fishing_areas.insert_one(forbidden_area_1)
    i+=1

```

Ευρετήρια

Όπως αναφέραμε, θα χρησιμοποιήσουμε δύο ευρετήρια. Ένα compound index που περιλαμβάνει πρώτα το χρόνο (“t”) και έπειτα το γεωγραφικό σημείο (“location”), και ένα απλό ευρετήριο ως προς την κατάσταση που δηλώνει κάθε πλοίο (“navigationalstatus”).

Για την δημιουργία τους εκτελούμε απευθείας στη βάση της ακόλουθες δύο εντολές.

db.nari_dynamic.createIndex({"t":1,"location" : "2dsphere"})

db.nari_dynamic.createIndex({"navigationalstatus":1})

Ανάλυση των δεδομένων

Αλιευτική δραστηριότητα

Αρχικά θα εξάγουμε κάποια βασικά στοιχεία σχετικά με την αλιευτική δραστηριότητα από τα διαθέσιμα δεδομένα του αρχείου nari_dynamic.csv. Τα queries που χρησιμοποιούμε γίνονται εφαρμόζοντας το **aggregation pipeline** της MongoDB, που αποτελείται από διάφορα στάδια.

Πλήθος καταχωρήσεων για αλιευτική δραστηριότητα για κάθε πλοίο

```
#Find distinct fishing vessels and count their fishing activity instances

query = ([
    { '$match':{'navigationalstatus':{'$eq':7}}},
    { '$group': {'_id':'$sourcemmsi', 'num_appearances':{'$sum':1}}},
    { '$sort':{'num_appearances':-1}}
    ])

document = nari_dynamic.aggregate(query)

results=[]
for i in document:
    results.append(i)


res=pd.DataFrame(results).copy()
res['_id']=res['_id'].astype(str)
```

```
fishing_vessels=list(res['_id'].astype(int))
print('There are',len(fishing_vessels),'vessels that were fishing')
```

```
There are 208 vessels that were fishing
```

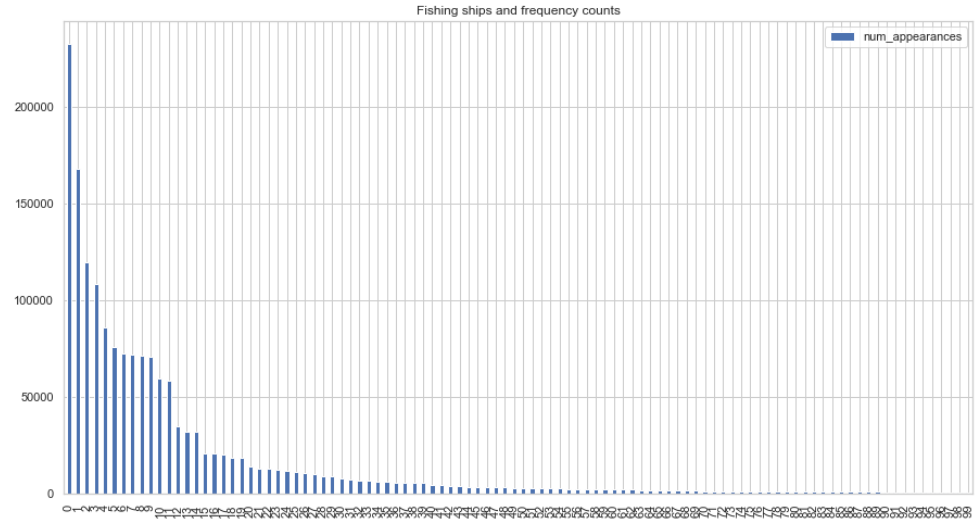
Όπως φαίνεται, στέλνουμε το query στη βάση και αποθηκεύουμε σε dataframe τα δεδομένα που επιστρέφονται. Επίσης εμφανίζουμε το πλήθος των διακριτών πλοίων που δήλωσαν ότι ψαρεύανε, το οποίο είναι 208 πλοία. Παρακάτω βλέπουμε τις πέντε πρώτες και πέντε τελευταίες γραμμές των αποτελεσμάτων αλλά και ένα ραβδόγραμμα με τα 100 πρώτα αποτελέσματα.

```
res
```

	_id	num_appearances
0	227941000	232591
1	227300000	167941
2	228394000	119457
3	227391000	108516
4	228144000	85724
...	...	...
203	235103397	2
204	250002464	2
205	225361000	1
206	228143600	1
207	227573960	1

```
1 res[:100].plot(kind='bar',figsize=(15,8),title='Fishing ships and frequency counts')
2
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x242849a2668>
```



Παρατηρούμε ότι τα πρώτα δέκα πλοία έχουν περισσότερες από 50.000 καταχωρήσεις. Αντίθετα υπάρχουν πλοία με ελάχιστες (έως και 1). Στη συνέχεια παραθέτουμε κάποια στατιστικά σχετικά με το πλήθος των καταχωρήσεων.

```
1 res.describe()
```

	num_appearances
count	208.0000
mean	8071.0577
std	26218.5440
min	1.0000
25%	33.0000
50%	453.5000
75%	2735.7500
max	232591.0000

Όπως φαίνεται, κατά μέσο όρο τα αλιευτικά πλοία έχουν περίπου 80.000 καταχωρήσεις με τυπική απόκλιση περίπου 26.000. Η ελάχιστη τιμή είναι 1 καταχώρηση ενώ η μέγιστη πάνω από 230.000. Επίσης τα μισά πλοία έχουν το πολύ 453 καταχωρήσεις ενώ το 75% αυτών το πολύ 2.736

Χώρες προέλευσης

Το αναγνωριστικό (sourcemmsi) κάθε πλοίου αντιστοιχεί σε έναν 9ψήφιο ακέραιο αριθμό. Τα πρώτα τρία στοιχεία αυτού του αριθμού αντιστοιχούν στη χώρα προέλευσης. Έτσι μπορούμε να υπολογίσουμε πόσα αλιευτικά αντιστοιχούν σε κάθε κωδικό, δηλαδή σε κάθε χώρα.

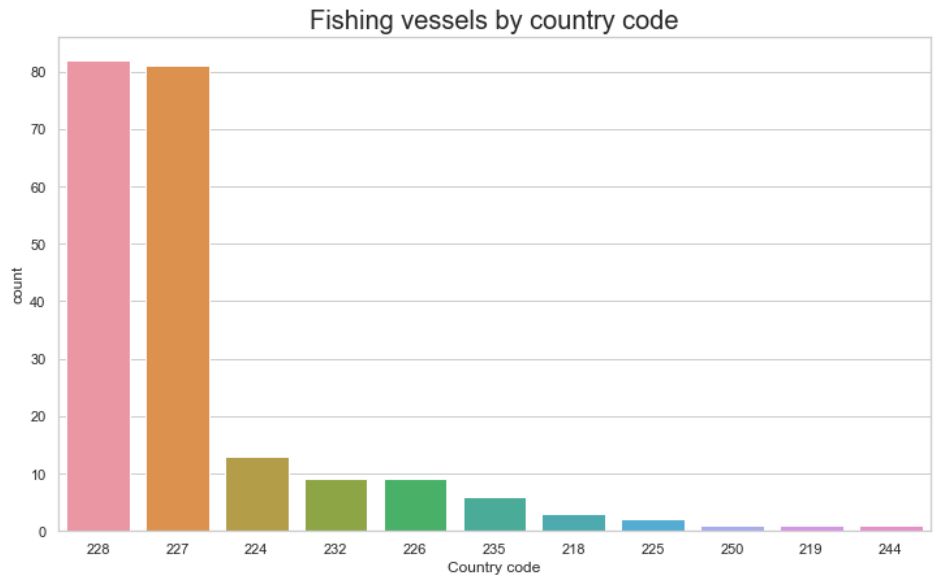
```
:  1 mmsi=res['_id'].apply( lambda x : ''.join(list(x)[0:3]) )
   2 print('There are vessels from',len(mmsi.unique()),'countries that were fishing')
   3 print('Country codes: ',mmsi.unique())
```

```
There are vessels from 11 countries that were fishing
Country codes:  ['227' '228' '226' '224' '219' '232' '235' '244' '218' '225' '250']
```

Οι χώρες που αντιστοιχούν στους παραπάνω κωδικούς είναι :

227, 228, 226: Γαλλία
224,225 : Ισπανία
218: Γερμανία
219: Δανία
232, 235 : Ηνωμένο Βασίλειο
244: Ολλανδία
250: Ιρλανδία

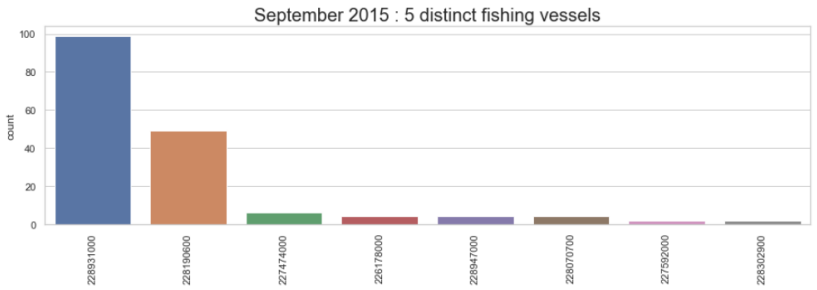
```
1 plt.figure(figsize=(12,7))
2 plt.title('Fishing vessels by country code',fontsize=20)
3 sns.countplot(mmsi,order = mmsi.value_counts().index)
4 plt.xlabel('Country code')
5 plt.show()
```

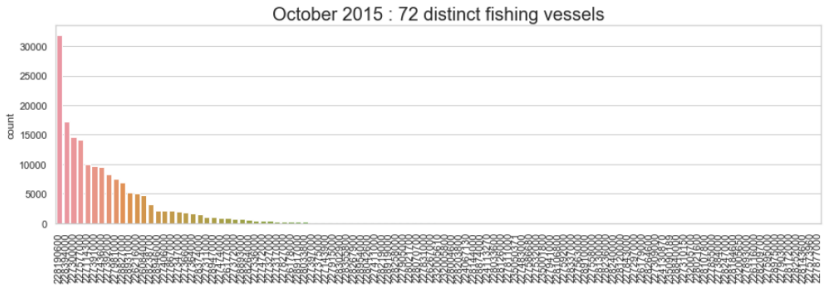


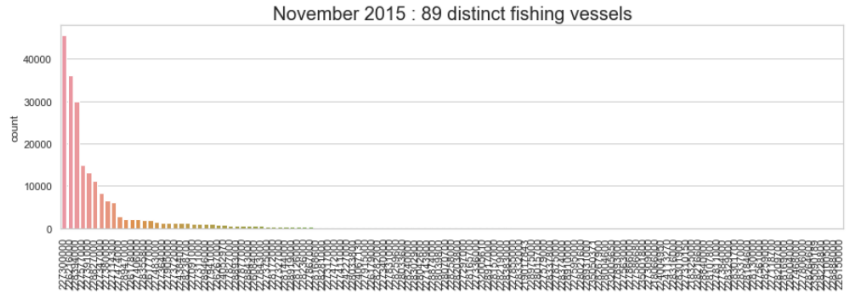
Στοιχεία ανά μήνα

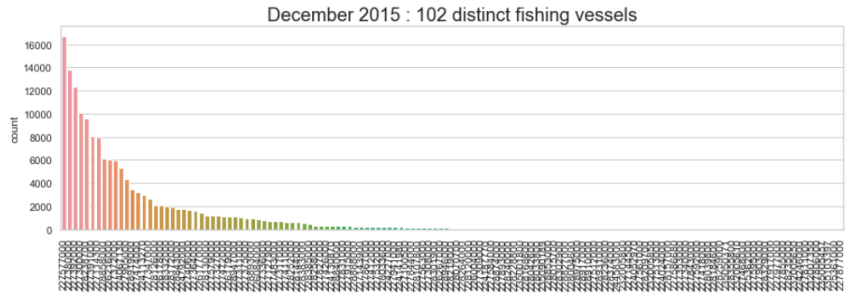
Το χρονικό εύρος των δεδομένων είναι από τον Σεπτέμβρη του 2015 έως τον Μάρτιο του 2016. Στα επόμενα, υπολογίζουμε πόσα πλοία ενεπλάκησαν σε αλιευτική δραστηριότητα για κάθε μήνα, και φτιάχνουμε ένα διάγραμμα για κάθε έναν. Κάνοντας χρήση της μεθόδου **$month** της MongoDB, μπορούμε εύκολα να ομαδοποιήσουμε τα έγγραφα με βάση τους μήνες και με τη μέθοδο **$push**, αποθηκεύουμε σε ένα διάνυσμα όλα τα αναγνωριστικά των πλοίων που εμφανίστηκαν τον κάθε μήνα.

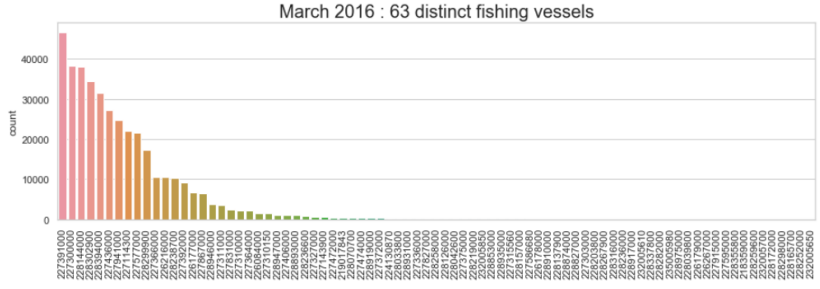
```
#Get all coordinates from the fishing vessels,group by months and  count

query = ([
    { '$match':{'navigationalstatus':{'$eq':7}}
    },
    { '$group': {'_id':{'month':{'$month':'$t'}},
                 'count':{'$sum':1},
                 'sourcemmsi':{'$push':'$sourcemmsi'}}
    },
    { '$sort':{'_id':1}}
    ])

document = nari_dynamic.aggregate(query,allowDiskUse=True)

results=[]
for i in document:
    results.append(i)

res_monthly=pd.DataFrame(results)

months={
        1:'January 2016',
        2:'February 2016',
        3:'March 2016',
        9:'September 2015',
        10:'October 2015',
        11:'November 2015',
        12:'December 2015',
}
i=0

for month in res_monthly['_id']:

    montly_counts=pd.Series(res_monthly.sourcemmsi[i]).value_counts()

    plt.figure(figsize=(12,5))
    plt.title(months[month['month']]+' : '+str(len(np.unique(montly_counts)))+' distinct fishing vessels',fontsize=
    plot=sns.countplot(res_monthly.sourcemmsi[i],order=montly_counts.index)
    plot.set_xticklabels(plot.get_xticklabels(),rotation=90)
    plt.show()
    i+=1
```

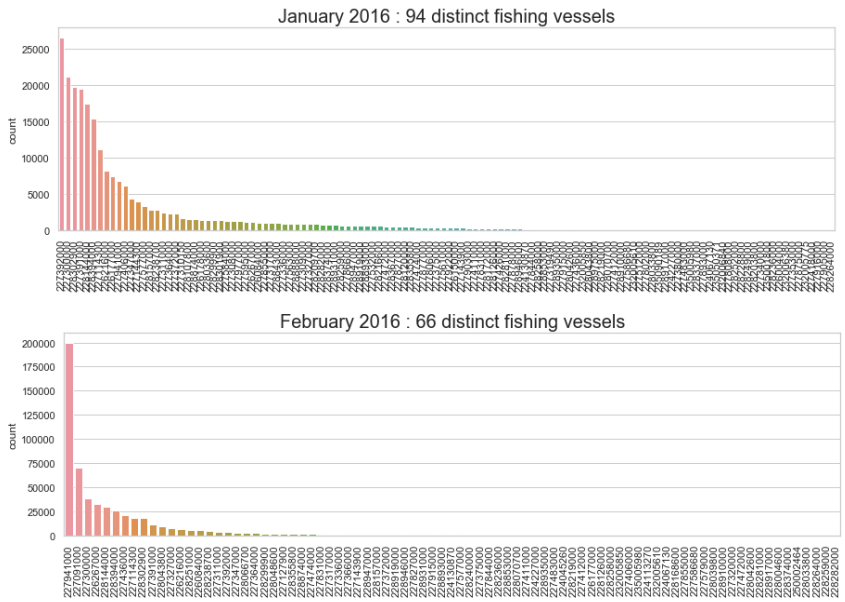

September 2015 : 5 distinct fishing vessels
count
100
80
60
40
20
0
228931000
228190600
227474000
226178000
228947000
228070700
227592000
228302900


October 2015 : 72 distinct fishing vessels
count
30000
25000
20000
15000
10000
5000
0


November 2015 : 89 distinct fishing vessels
count
40000
30000
20000
10000
0


December 2015 : 102 distinct fishing vessels
count
16000
14000
12000
10000
8000
6000
4000
2000
0


March 2016 : 63 distinct fishing vessels
count
40000
30000
20000
10000
0



Συσταδοποίηση τροχιών αλιευτικών πλοίων

Αρχικά, πρέπει να φέρουμε τα δεδομένα στη μορφή που χρειάζεται για να μπορούμε να τα αναλύσουμε. Αυτή είναι τύπου: #πλοίο : [[ΘΕΣΗ 1],[ΘΕΣΗ2],…].

Δηλαδή πρέπει να αντιστοιχήσουμε σε κάθε διακριτό πλοίο μια λίστα με τις προηγούμενες συντεταγμένες του. Τα πλοία που θα χρησιμοποιήσουμε τα έχουμε βρει στα προηγούμενα. Από αυτά, επιλέγουμε να αφαιρέσουμε τα πλοία με λίγες εγγραφές (<20) καθώς και αυτά με υπερβολικά πολλές (>50.000). Εν τέλει, θα αναλύσουμε τις τροχιές για 154 πλοία.

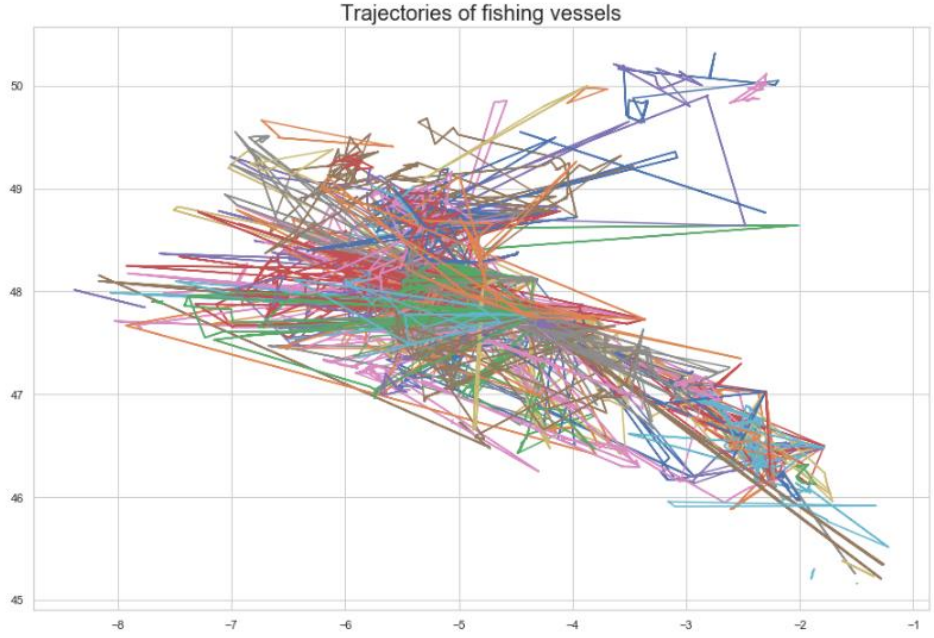
Το query που χρησιμοποιούμε χρησιμοποιεί και εδώ το aggregation pipeline. Με τη χρήση του τελεστή **$in** επιλέγουμε τα πλοία που βρίσκονται στη λίστα με το σύνολο των πλοίων και με τη χρήση του **$push**, εισάγουμε σε μια λίστα όλες τις τοποθεσίες για κάθε πλοίο.

```
#Get all coordinates from the fishing vessels

query = ([
    { '$match':{'sourcemmsi':{'$in':fishing_vessels[-164:-10:1]}}},
    { '$group': {'_id':'$sourcemmsi', 'locations':{'$push':'$location.coordinates'}}},
    { '$sort':{'t':-1}}
    ])

document2 = nari_dynamic.aggregate(query,allowDiskUse=True)

results2=[]
for i in document2:
    results2.append(i)


res2=pd.DataFrame(results2)
print(res2)
res2['_id']=res2['_id'].astype(str)
```

```
          _id                                          locations
0    226150000  [[-2.8469417, 46.808846], [-2.8375932999999995...
1    228120000  [[-2.610005, 45.885033], [-2.6056633, 45.88513...
2    227303000  [[-5.3681116, 48.111694], [-5.3650199999999995...
3    228240000  [[-4.841038, 47.854034000000006], [-4.81325, 4...
4    228203800  [[-5.135842, 48.717915000000005], [-5.13125, 4...
..         ...                                                ...
149  228236000  [[-5.0786614000000005, 47.633907], [-5.0735765...
150  228021600  [[-4.9238214000000005, 48.291733], [-4.918665,...
151  228106800  [[-4.6752234, 47.962368], [-4.676148400000001,...
152  226004000  [[-2.0297666000000003, 46.298626], [-1.923525,...
153  232005800  [[-5.9387784, 49.2931], [-5.9356465, 49.29647]...

[154 rows x 2 columns]
```

Παρακάτω φαίνονται οι επιμέρους τροχιές που σχηματίζονται από τα δεδομένα.

```
plt.figure(figsize=(15,10))
plt.title('Trajectories of fishing vessels',fontsize=20)
for trajectory in res2.locations:
    plt.plot(trajectory[:, 0], trajectory[:, 1])
```



Τώρα θα χρησιμοποιήσουμε την απόσταση Hausdorff για να υπολογίσουμε τον πίνακα αποστάσεων, στον οποίο μετά θα εφαρμόσουμε τους αλγορίθμους συσταδοποίησης. Η απόσταση αυτή υπάρχει διαθέσιμη από τη βιβλιοθήκη SciPy και έτσι η διαδικασία γίνεται αρκετά εύκολα. Παρακάτω φαίνεται ο κώδικας για τη δημιουργία του πίνακα αποστάσεων και ο κώδικας για μια συνάρτηση όπου θα οπτικοποιεί την ανάθεση των συστάδων. Επειδή κάποιοι αλγόριθμοι (DBSCAN), σχηματίζουν μία συστάδα με τα outliers (ή noise), στην οποία μάλιστα ανατίθεται ο αριθμός -1 , επιλέγουμε να τη χρωματίσουμε με μαύρο χρώμα ώστε να ξεχωρίζει.

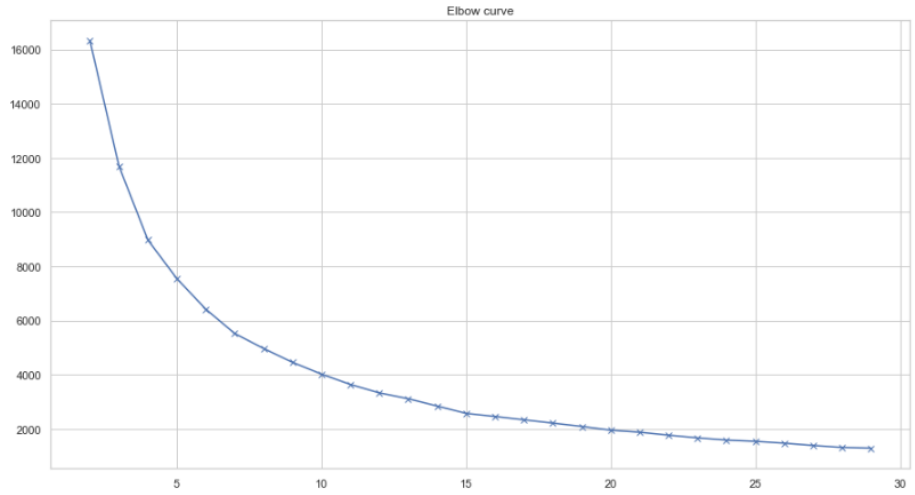
```
#Creating hausdorff distance matrix

def hausdorff( u, v):
    d = max(directed_hausdorff(u, v)[0], directed_hausdorff(v, u)[0])
    return d

traj_count = len(res2.locations)
D = np.zeros((traj_count, traj_count))


for i in range(traj_count):
    for j in range(i + 1, traj_count):
        distance = hausdorff(res2.locations[i], res2.locations[j])
        D[i, j] = distance
        D[j, i] = distance
```

```

#Function for plottting clustered trajectories

colors = plt.rcParams['axes.prop_cycle'].by_key()['color']
colors.extend(['firebrick', 'olive', 'indigo', 'khaki', 'teal', 'saddlebrown',
               'skyblue', 'coral', 'darkorange', 'lime', 'darkorchid', 'dimgray'])

def plot_cluster(trajs, clusters):

    cluster_count = np.max(clusters) + 1

    for traj, cluster in zip(trajs, clusters):

        if cluster == -1:
            plt.plot(traj[:, 0], traj[:, 1], c ='k', linestyle='dashed')

        else:
            plt.plot(traj[:, 0], traj[:, 1], c = colors[cluster % len(colors)])
    plt.show()
```

Για να πάρουμε μια πρώτη γνώμη σχετικά με το βέλτιστο πλήθος συστάδων, χρησιμοποιούμε τη μέθοδο του αγκώνα (elbow method) με τον αλγόριθμο KMeans, για τον πίνακα αποστάσεων που υπολογίσαμε προηγουμένως.

```python
from sklearn.cluster import KMeans

def elbow_method(data,max_clusters=10):
    inertias = []
    for k in range(2, max_clusters):
        kmeans = KMeans(n_clusters=k,random_state=7,n_jobs=-1)
        kmeans.fit(data)
        inertias.append(kmeans.inertia_)

    fig = plt.figure(figsize=(15, 8))
    plt.plot(range(2, max_clusters), inertias, 'bx-')
    plt.grid(True)
    plt.title('Elbow curve')
```

```python
elbow_method(D,30)
```

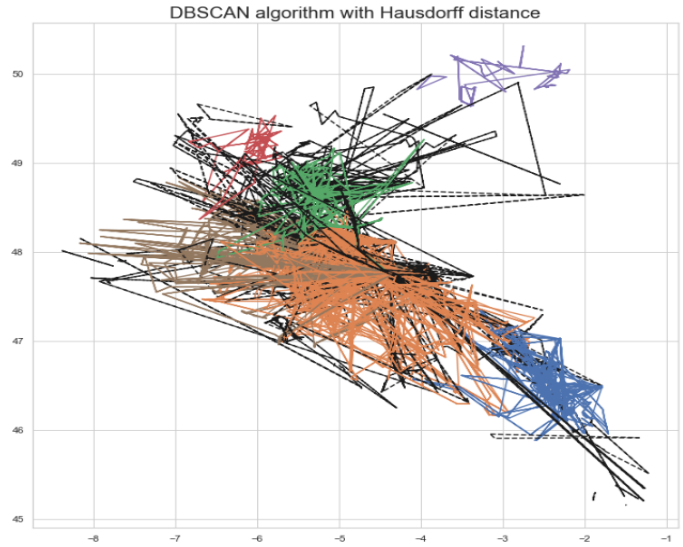


Όπως φαίνεται, το βέλτιστο πλήθος σύμφωνα με το παραπάνω διάγραμμα φαίνεται να είναι από 6 έως 10 συστάδες. Στη συνέχεια εφαρμόζουμε τους αλγορίθμους συσταδοποίησης. Οι αλγόριθμοι DBSCAN και OPTICS δεν απαιτούν την εισαγωγή του πλήθους των συστάδων. Παρόλ'αυτά, ο πρώτος καταλήγει σε 7 συστάδες ενώ ο δεύτερος σε 10. Κάτι που φαίνεται να έρχεται σε συμφωνία με τα αποτελέσματα της μεθόδου του αγκώνα. Στον αλγόριθμο BIRCH, επιλέγουμε 7 συστάδες.

DBSCAN

```
dbscan = DBSCAN(eps=4, min_samples=4,n_jobs=-1)
clusters = dbscan.fit_predict(D)
print('Nof clusters:',len(np.unique(clusters)))
plt.figure(figsize=(12,12))
plt.title('DBSCAN algorithm with Hausdorff distance',fontsize=20)
plot_cluster(res2.locations, clusters)

```

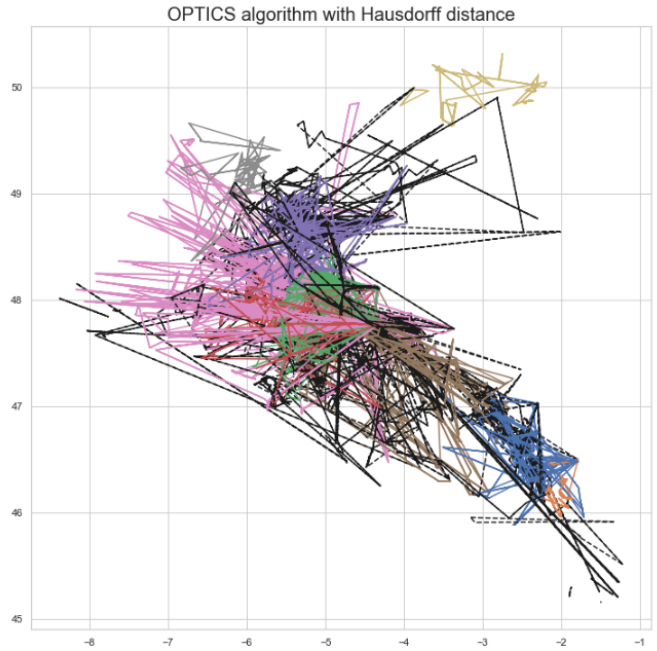
```
Nof clusters: 7
```



OPTICS

```
optics = OPTICS(eps=5, min_samples=6,leaf_size=30,n_jobs=-1)
clusters = optics.fit_predict(D)
print('Nof clusters:',len(np.unique(clusters)))
plt.figure(figsize=(12,12))
plt.title('OPTICS algorithm with Hausdorff distance',fontsize=20)
plot_cluster(res2.locations, clusters)

```

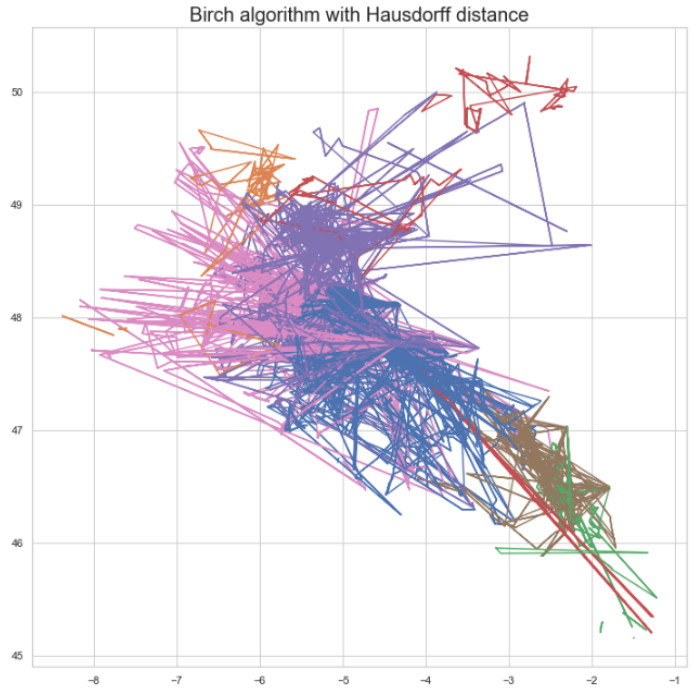
```
Nof clusters: 10
```



BIRCH

```
brc = Birch(n_clusters=7)
clusters = brc.fit_predict(D)
print('Nof clusters:',len(np.unique(clusters)))
plt.figure(figsize=(12,12))
plt.title('Birch algorithm with Hausdorff distance',fontsize=20)
plot_cluster(res2.locations, clusters)
```

```
Nof clusters: 7
```



Συσταδοποίηση σημείων αλιευτική δραστηριότητας

Στη συνέχεια εξετάζουμε τη συσταδοποίηση των σημείων που εκδηλώθηκε αλιευτική δραστηριότητα. Εδώ δεν θα υπολογίσουμε για τα αλιευτικά πλοία όλες τις προηγούμενες γεωγραφικές τους θέσεις, αλλά μόνο τις θέσεις στις οποίες ενεπλάκησαν σε αλιευτικές διαδικασίες.

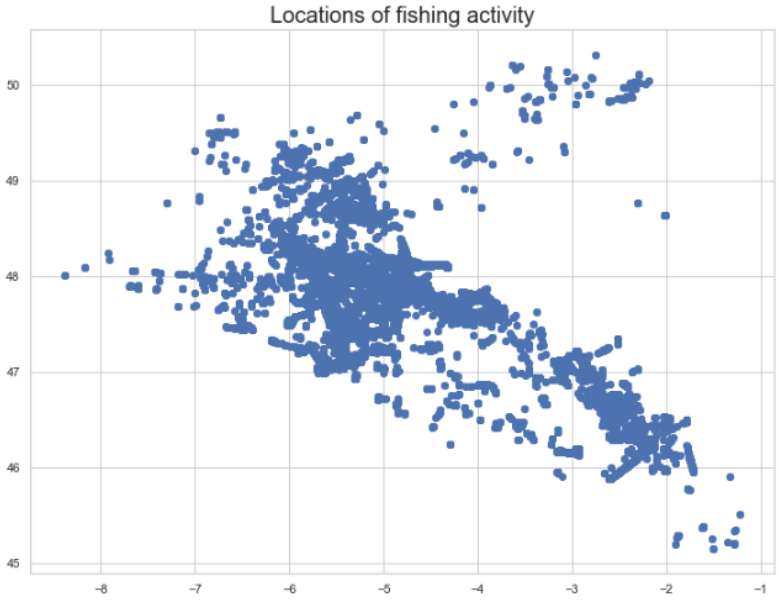
Και πάλι χρησιμοποιούμε το aggregation pipeline όπως κάναμε και στην προηγούμενη ενότητα, για να πάρουμε τις τοποθεσίες από 154 πλοία. Παρακάτω φαίνεται ο αντίστοιχος κώδικας καθώς και ένα διάγραμμα με τις τοποθεσίες. Εδώ δεν μας ενδιαφέρει το κάθε πλοίο, αλλά συνολικά οι συντεταγμένες, γι'αυτό τις ενώνουμε σε μία ενιαία λίστα ώστε να τις δούμε όλες μαζί.

```
#Get all coordinates from the fishing vessels

query = ([
    { '$match':{'navigationalstatus':{'$eq':7},
                'sourcemmsi':{'$in':fishing_vessels[-164:-10:1]}}},
    { '$group': {'_id':'$sourcemmsi',
                 'locations':{'$push':'$location.coordinates'}}},
    ])

document = nari_dynamic.aggregate(query,allowDiskUse=True)

results=[]
for i in document:
    results.append(i)


res3=pd.DataFrame(results)
print(res3)
res3['_id']=res3['_id'].astype(str)
```

```
          _id                                          locations
0    227472000  [[-5.22182, 48.238859999999995], [-5.0097933, ...
1    228840000  [[-2.2973516, 47.02714], [-2.2978283999999998,...
2    227317000  [[-4.831978299999999, 47.862957], [-4.85794, 4...
3    228282000  [[-5.6109519999999999, 47.398586], [-5.517648, ...
4    228209700  [[-2.3717099999999998, 46.34022], [-2.3722167,...
..         ...                                                ...
149  228259000  [[-3.0895984, 47.11665], [-3.0897317, 47.11651...
150  228917000  [[-1.2608683, 45.346214], [-1.2799083, 45.3450...
151  232007350       [[-2.94427, 49.90136], [-2.94427, 49.90136]]
152  228252000  [[-4.1755949999999995, 47.835873], [-4.175598,...
153  227315560  [[-5.183143, 48.78235], [-5.354912000000001, 4...

[154 rows x 2 columns]
```

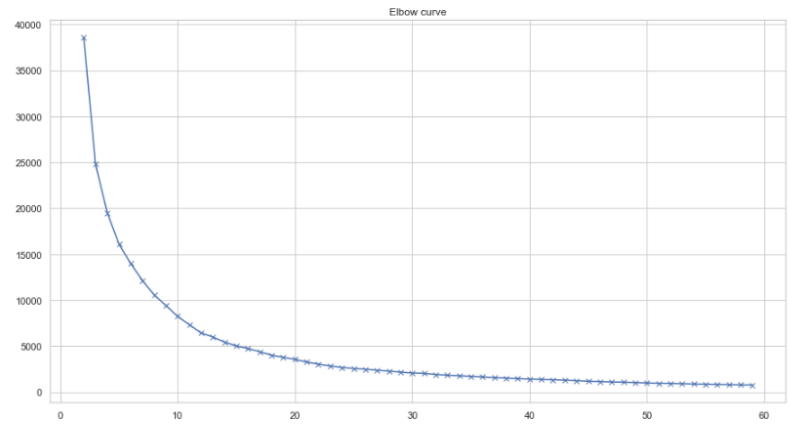
```
fishing_coords = []
for sublist in res3.locations:
    for item in sublist:
        fishing_coords.append([item[0],item[1]])

fishing_coords = pd.DataFrame.from_records(fishing_coords,columns=['lon','lat'])
```

```
plt.figure(figsize=(12,9))
plt.title('Locations of fishing activity',fontsize=20)
plt.scatter(fishing_coords.lon,fishing_coords.lat)
plt.show()
```



Στη συνέχεια φαίνονται τα αποτελέσματα του elbow method για τα δεδομένα. Και πάλι το βέλτιστο πλήθος συστάδων φαίνεται να είναι κοντά στο 10.

```
elbow_method(fishing_coords,60)
```

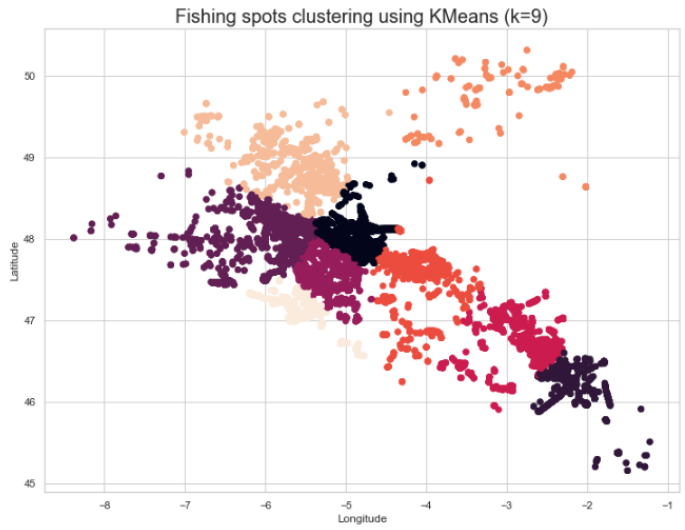


Έπειτα χρησιμοποιούμε τους αλγορίθμους KMeans και Birch για τη συσταδοποίηση. Οι άλλοι αλγόριθμοι DBSCAN και OPTICS δεν μας έδωσαν ικανοποιητικά αποτελέσματα και οι χρόνοι εκτέλεσής τους ήταν ιδιαίτερα μεγάλοι. Γι'αυτό δεν τους χρησιμοποιούμε σε αυτή την περίπτωση.

KMEANS

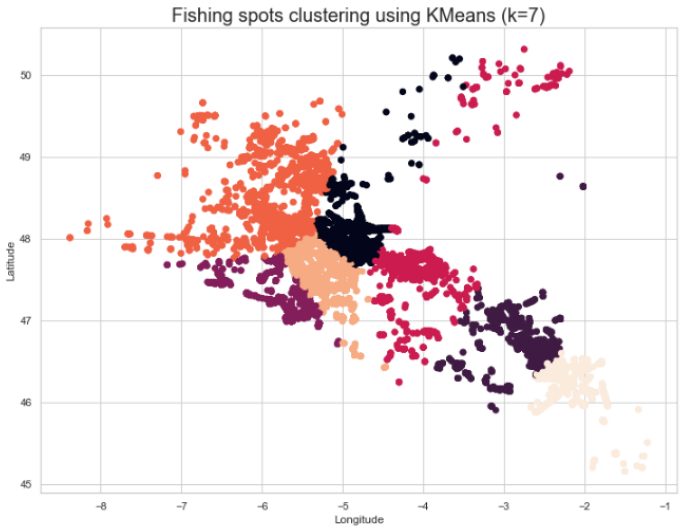
```
kmeans = KMeans(9,n_jobs=-1)
clusters = kmeans.fit_predict(fishing_coords)
print('Nof clusters:',len(np.unique(clusters)))

clusterd_fishing_coords=fishing_coords.copy()
clusterd_fishing_coords['clusters']=clusters

fig, ax = plt.subplots(figsize=[12, 9])
plt.scatter(fishing_coords.lon,fishing_coords.lat,c=clusters)
plt.title('Fishing spots clustering using KMeans (k=9)',fontsize=20)
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
plt.show()
```

```
Nof clusters: 9
```



```
kmeans = KMeans(7,n_jobs=-1)
clusters = kmeans.fit_predict(fishing_coords)
print('Nof clusters:',len(np.unique(clusters)))

clusterd_fishing_coords=fishing_coords.copy()
clusterd_fishing_coords['clusters']=clusters

fig, ax = plt.subplots(figsize=[12, 9])
plt.scatter(fishing_coords.lon,fishing_coords.lat,c=clusters)
plt.title('Fishing spots clustering using KMeans (k=7)',fontsize=20)
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
plt.show()
```

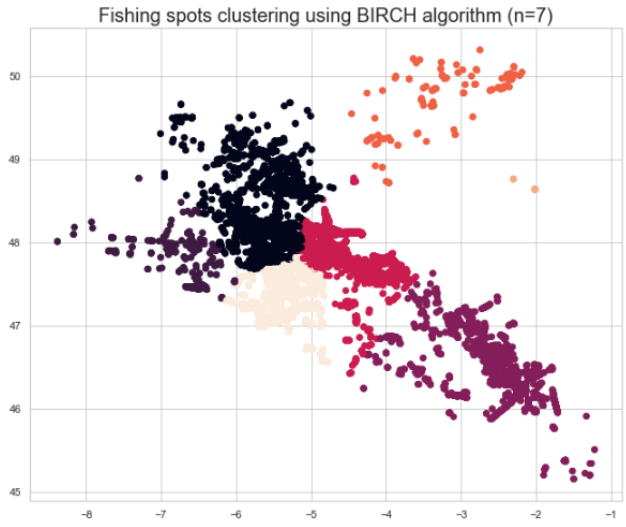
```
Nof clusters: 7
```



BIRCH

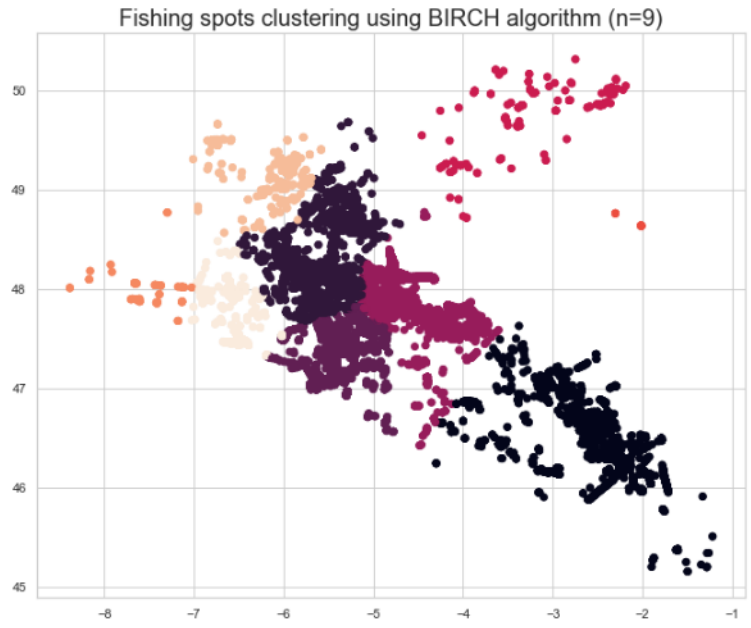
```
#Using BIRCH

brc = Birch(n_clusters=7)
clusters = brc.fit_predict(fishing_coords)
print('Nof clusters:',len(np.unique(clusters)))
clusterd_fishing_coords=fishing_coords.copy()
clusterd_fishing_coords['clusters']=clusters
plt.figure(figsize=(11,9))
plt.title('Fishing spots clustering using BIRCH algorithm (n=7)',fontsize=20)
plt.scatter(clusterd_fishing_coords.lon,clusterd_fishing_coords.lat,c=clusterd_fishing_coords['clusters'])
plt.show()
```

```
Nof clusters: 7
```



```
#Using BIRCH

brc = Birch(n_clusters=9)
clusters = brc.fit_predict(fishing_coords)
print('Nof clusters:',len(np.unique(clusters)))
clusterd_fishing_coords=fishing_coords.copy()
clusterd_fishing_coords['clusters']=clusters
plt.figure(figsize=(11,9))
plt.title('Fishing spots clustering using BIRCH algorithm (n=9)',fontsize=20)
plt.scatter(clusterd_fishing_coords.lon,clusterd_fishing_coords.lat,c=clusterd_fishing_coords['clusters'])
plt.show()
```

```
Nof clusters: 9
```



Παράνομη αλιεία

Ο εντοπισμός περιπτώσεων όπου υπάρχει αλιευτική δραστηριότητα σε απαγορευμένες περιοχές ανάγεται σε γεωχωρικά, χρονικώς εξαρτώμενα, queries. Εδώ γίνεται και η μεγαλύτερη αξιοποίηση των ευρετηρίων καθώς το compound ευρετήριο που δημιουργήσαμε είναι συνδυασμός χρόνου και τοποθεσίας. Τις περιοχές όπου υπάρχει απαγόρευση τις έχουμε ήδη εισάγει στη βάση σε μορφή GeoJSON με πολύγωνα. Έτσι, κάνουμε χρήση της λειτουργίας **$geoWithin** της MongoDB, με την οποία εξετάζουμε εάν ένα σημείο βρίσκεται μέσα στα πολύγωνα που μας ενδιαφέρουν, δηλαδή στις απαγορευμένες περιοχές. Παρακάτω φαίνεται ο κώδικας και τα αντίστοιχα πλοία που εντοπίστηκαν.

Camaret area, north ¶

```
import datetime

#Camaret area, north
start=datetime.datetime(2015,2,5,0,0,0)
end=datetime.datetime(2016,1,21,0,0,0)
```

```
camaret=db.forbidden_fishing_areas.find({"id" : 'Camaret'})
```

```
camaret_violations=db.nari_dynamic.find({"navigationalstatus":{"$eq":7},
                                         "t":{"$gte":start,"$lte":end},
                                         "location":{"$geoWithin":{"$geometry":camaret[0]}}})
```

```
results3=[]
for i in camaret_violations:
    results3.append(i)

res3=pd.DataFrame(results3)
res3['sourcemmsi']=res3['sourcemmsi'].astype(str)
```

```
print('There are',len(res3['sourcemmsi'].unique()),'vessels that were fishing illegaly at Camaret area:')
print(res3['sourcemmsi'].unique())
```

```
There are 2 vessels that were fishing illegaly at Camaret area:
['228355800' '228853000']
```

Concarneau area, south ¶

```
start=datetime.datetime(2015,9,3,0,0,0)
end=datetime.datetime(2015,10,15,0,0,0)

import re
regx = re.compile("^Concarneau", re.IGNORECASE)
```

```
concarneau=db.forbidden_fishing_areas.find({"id" : regx})
```

```
concarneau_areas=list(concarneau)
```

```
concarenau_violations=[]

for area in concarneau_areas:
    violation = db.nari_dynamic.find({"navigationalstatus":{"$eq":7},
                                      "t":{"$gte":start,"$lte":end},
                                      "location":{"$geoWithin":{"$geometry":area}}})
    concarenau_violations.append(violation)
```

```
results4=[]
for violations in concarenau_violations:
    for j in violations:
        results4.append(j)

res4=pd.DataFrame(results4)
res4['sourcemmsi']=res4['sourcemmsi'].astype(str)
```

```
print('There are',len(res4['sourcemmsi'].unique()),'vessels that were fishing illegaly at Concarneau area:')
print(res4['sourcemmsi'].unique())
```

```
There are 2 vessels that were fishing illegaly at Concarneau area:
['228827000' '227577000']
```

Συμπεράσματα

Η χρήση της μη-σχεσιακή βάσης δεδομένων MongoDB ήταν ιδιαίτερα χρήσιμη για την αποθήκευση των δεδομένων και την ανάκτησή τους. Με τη χρήση ευρετηρίων τα queries τρέχουν άμεσα και η δυνατότητα για επεξεργασία γεωχωρικών δεδομένων (GeoJSON) από τη MongoDB είναι εξαιρετικά χρήσιμη για παρόμοια προβλήματα με το δικό μας.

Για τη συσταδοποίηση τροχιών οι αλγόριθμοι που χρησιμοποιήσαμε (κυρίως density-based) προσφέρουν ικανοποιητικά αποτελέσματα και τρέχουν σε μικρό χρόνο. Επίσης η επιλογή της μετρικής του Hausdorff προσφέρει επιπλέον ταχύτητα στους υπολογισμούς.

Για τη συσταδοποίηση των σημείων, χρησιμοποιήσαμε πιο απλούς αλγορίθμους (KMeans,Birch) καθώς οι υπόλοιποι απαιτούσαν ιδιαίτερα μεγάλο χρόνο εκτέλεσης, ενώ ταυτόχρονα αναγνώριζαν πολύ μεγάλο πλήθος συστάδων.

Για την εξέταση περιπτώσεων παράνομης αλιευτικής δραστηριότητας, το γεγονός ότι έχουμε compoundευρετήριο ως προς χρόνο και τόπο ταυτόχρονα, μας προσφέρει ταχύτατα αποτελέσματα Εντοπίσαμε παράνομη δραστηριότητα και στις δύο περιοχές όπου υπήρξε απαγόρευση, για διαφορετικά χρονικά διαστήματα.