

# Γεωγραφικά και Πληροφοριακά Συστήματα



**Πανεπιστήμιο Πειραιώς**  

---

**University of Piraeus**

Απαλλακτική Εργασία  
7ου εξαμήνου

Σκάρος Γεώργιος – Π15128  
Χαρμπής Θεμιστοκλής – Π15151

## Περιεχόμενα

- Εισαγωγή
- Φόρτωση Δεδομένων - Preprocessing
- Υπολογισμός Στατιστικών
- Επεξεργασία και Αναλυτική των Δεδομένων
- Επεξεργασία Μεγάλων Δεδομένων

# Εισαγωγή

Το dataset μας αποτελούταν από στίγματα πλοίων στον Αργοσαρωνικό τη περίοδο Φεβρουαρίου-Μαρτίου 2018.

## Φόρτωση Δεδομένων - Preprocessing

( 3.1 - 3.2 α )

Χρησιμοποιήθηκε το Σύστημα Διαχείρισης Βάσεων Δεδομένων PostgreSQL με το extension PostGIS που μας επιτρέπει να επεξεργαστούμε χωρικά δεδομένα (spatial data).

```
1 CREATE EXTENSION postgis
```

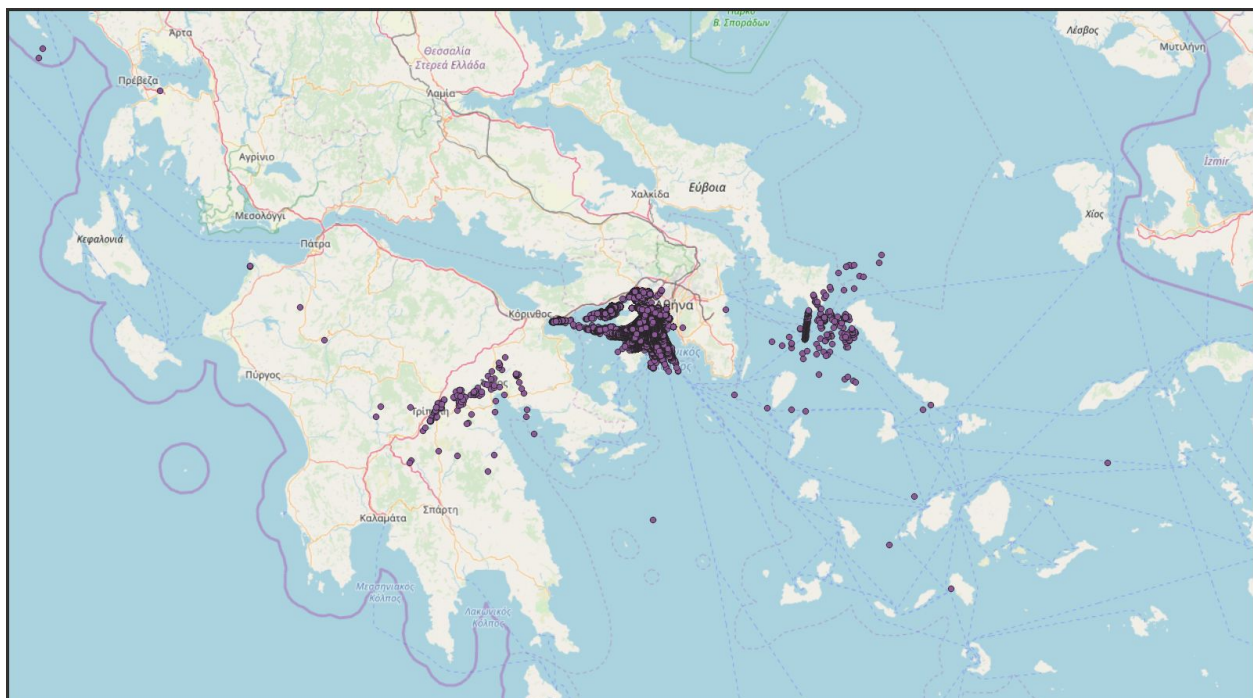
Επιλέξαμε τη POINT 2D απεικόνιση των δεδομένων έτσι ώστε να μπορέσουμε να κάνουμε ευκολότερα το preprocessing.

Αφού περάσαμε τα data στη βάση, φτιάξαμε ένα column geometry type POINT 4326 και το γεμίσαμε με τις τιμές των latitude και longitude.

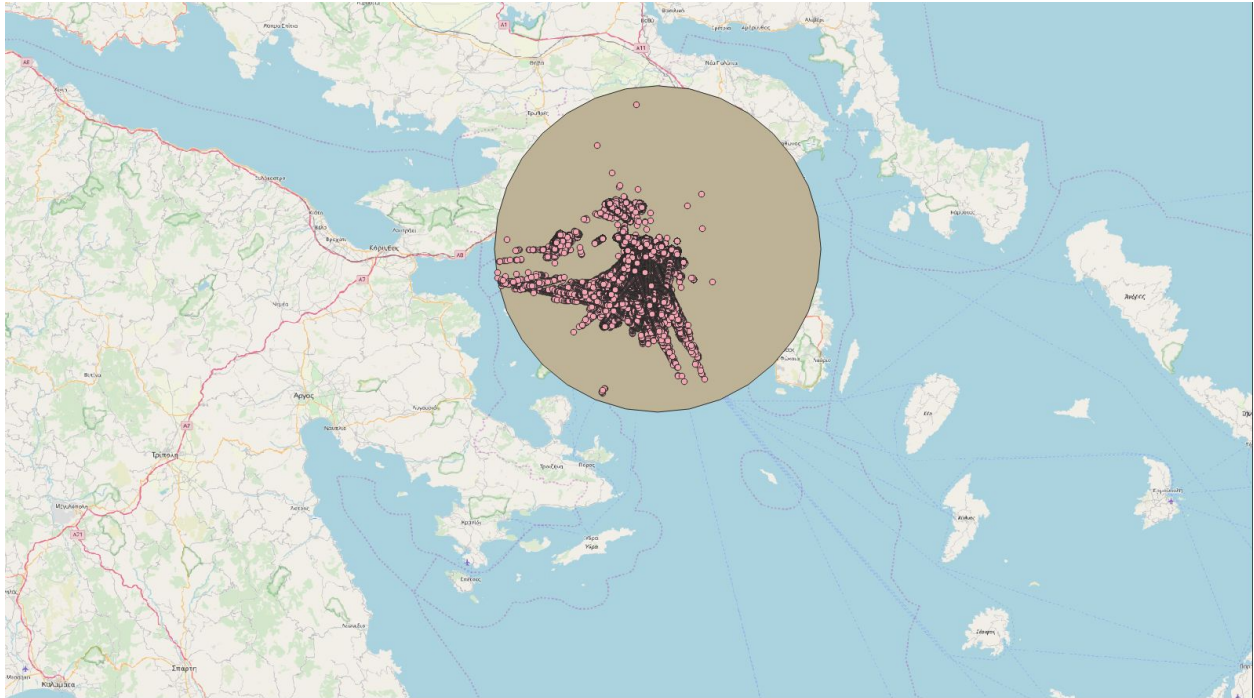
```
ALTER TABLE gis_all ADD COLUMN geom geometry(Point, 4326);
```

```
UPDATE gis_all SET geom = ST_SetSRID(ST_MakePoint(lon, lat), 4326);
```

Έπειτα τα εμφανίσαμε στο QGIS.



Παρατηρήσαμε πως υπάρχουν δεδομένα πολύ μακριά από τη κεραία του Πανεπιστημίου Πειραιώς ( [Marine Traffic #2782](#), [VesselFinder #5299](#) ), οπότε φτιάξαμε ένα buffer (ST\_Buffer) με ακτίνα 37 χιλιομέτρων, ίση δηλαδή με τη μέση ακτίνα της κεραίας. Με την εντολή ST\_Intersects είδαμε ποιά στοιχεία έρχονται σε επαφή με το buffer και σβήσαμε τα υπόλοιπα.



Στη συνέχεια, παρατηρήσαμε πως μερικά δεδομένα εμφανίζονται πάνω στη στεριά, στη περιφέρεια της Αττικής.

Πήραμε από τα έγγραφα του 2ου Εργαστηρίου το αρχείο με τις περιφέρειες της Ελλάδας. Βρήκαμε τη περιφέρεια της Αττικής (id=13) και φτιάξαμε ένα query που έπαιρνε σαν είσοδο τα δεδομένα από τον προηγούμενο καθαρισμό. Με τη χρήση πάλι της εντολής ST\_Intersect κρατήσαμε αυτά τα οποία δεν έκαναν intersect με το MULTIPLE POLYGON μας (αφού η περιφέρεια της Αττικής αποτελείται από πολλά πολύγωνα - νησιά).





# Υπολογισμός Στατιστικών

( 3.2 β,γ )

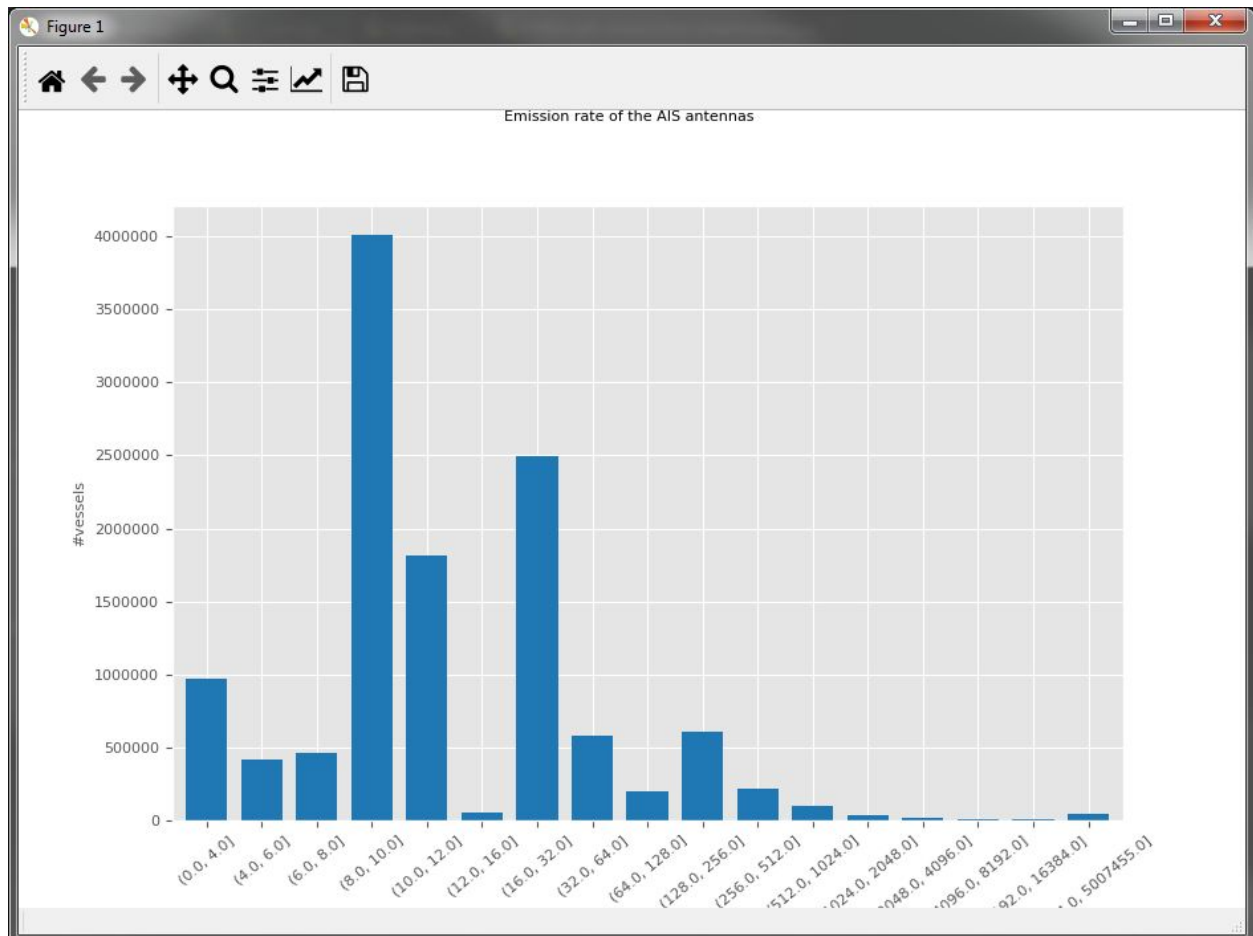
Για τον υπολογισμό των στατιστικών χρησιμοποίησαμε τη Python 3.7 όπως επίσης και τις βιβλιοθήκες :

- Pandas και GeoPandas, για τη διαχείριση των δεδομένων
- Psycopg2.extras, για τη σύνδεση με τη βάση PostgreSQL
- Matplotlib, για την εμφάνιση γραφημάτων

Αφού φορτώσαμε τα δεδομένα, αρχικά διαιρέσαμε τη στήλη χρόνου (ts) με το 1000 για να τη μετατρέψουμε από millisecond σε δευτερόλεπτα.

Έπειτα, βρήκαμε το ρυθμό δειγματοληψίας (sampling rate) βρίσκοντας τις διαφορές μεταξύ χρονικών στιγμών χρησιμοποιώντας την εντολή diff(-1).

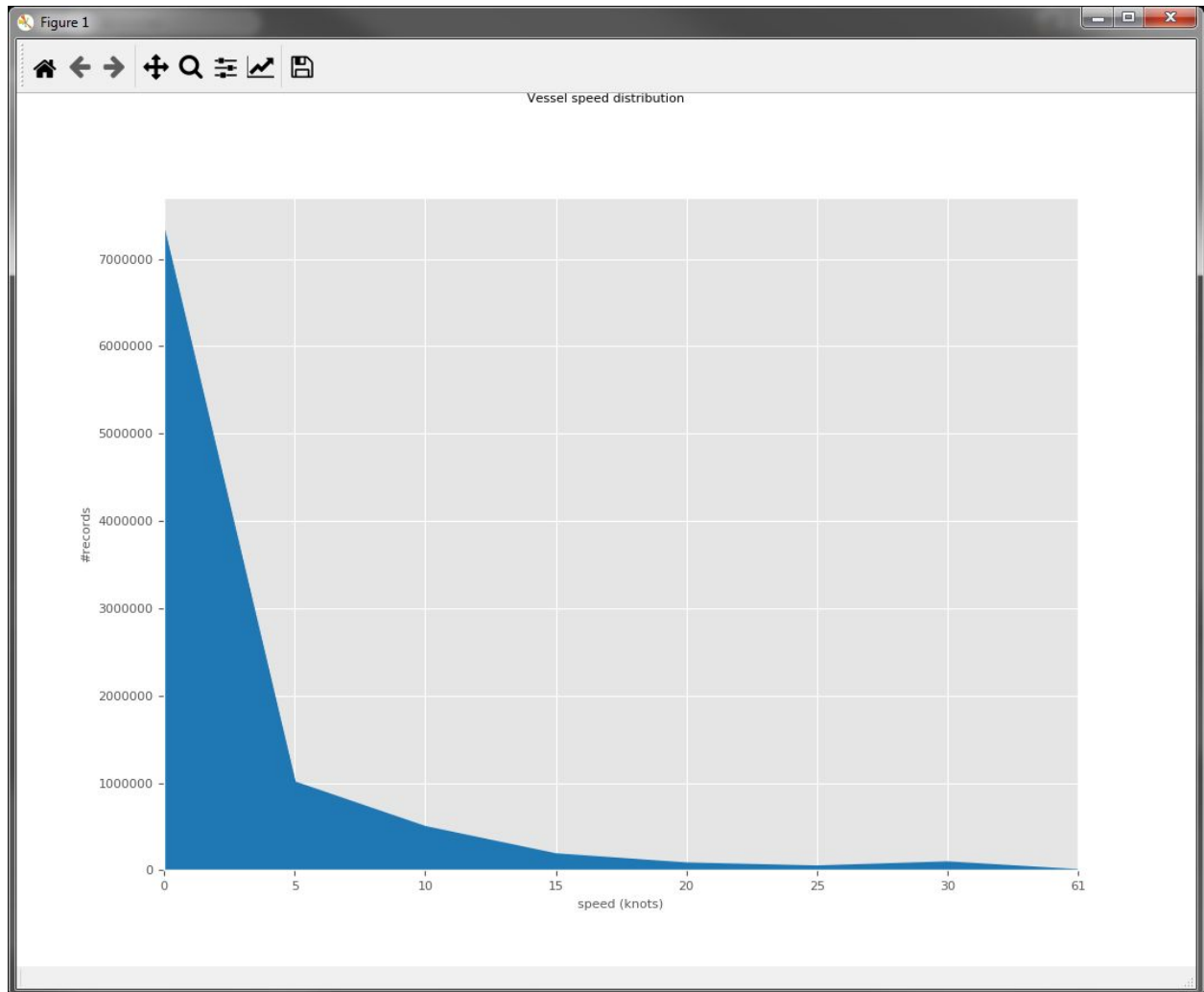
Με το Matplotlib δημιουργήσαμε κάποια bins στα οποία μοιράσαμε τα δεδομένα μας και τα εμφανίσαμε σε ένα ιστόγραμμα (histogram).

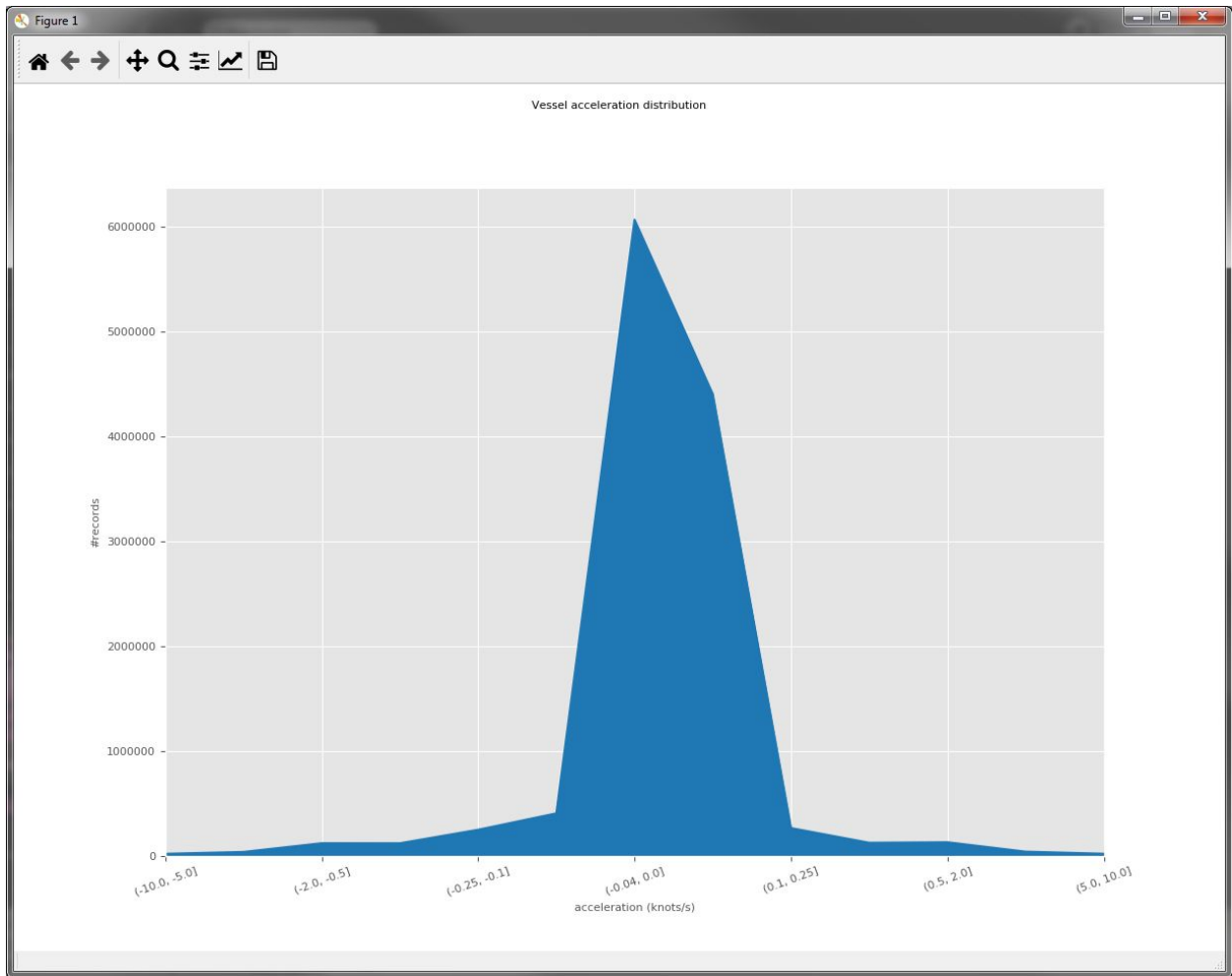


Κάποια από τα δεδομένα δεν ήταν αξιόπιστα όποτε χρειάστηκε να τα επανυπολογίσουμε.

Για τη ταχύτητα ανά σημείο χρησιμοποιήσαμε τη συνάρτηση από το αρχείο preprocessing.py που μας είχε δοθεί στο 4ο Εργαστήριο. Το ίδιο έγινε για τον υπολογισμό της επιτάχυνσης (acceleration) και του bearing. Τα παραπάνω ενσωματώθηκαν στο αρχικό πίνακα.

Στη συνέχεια σβήσαμε τις εγγραφές οι οποίες είχαν ακραίες τιμές ( $velocity > 60$ ,  $-10 < acceleration < 10$ ) και απεικονήσαμε (area plot) τις τιμές που έμειναν.





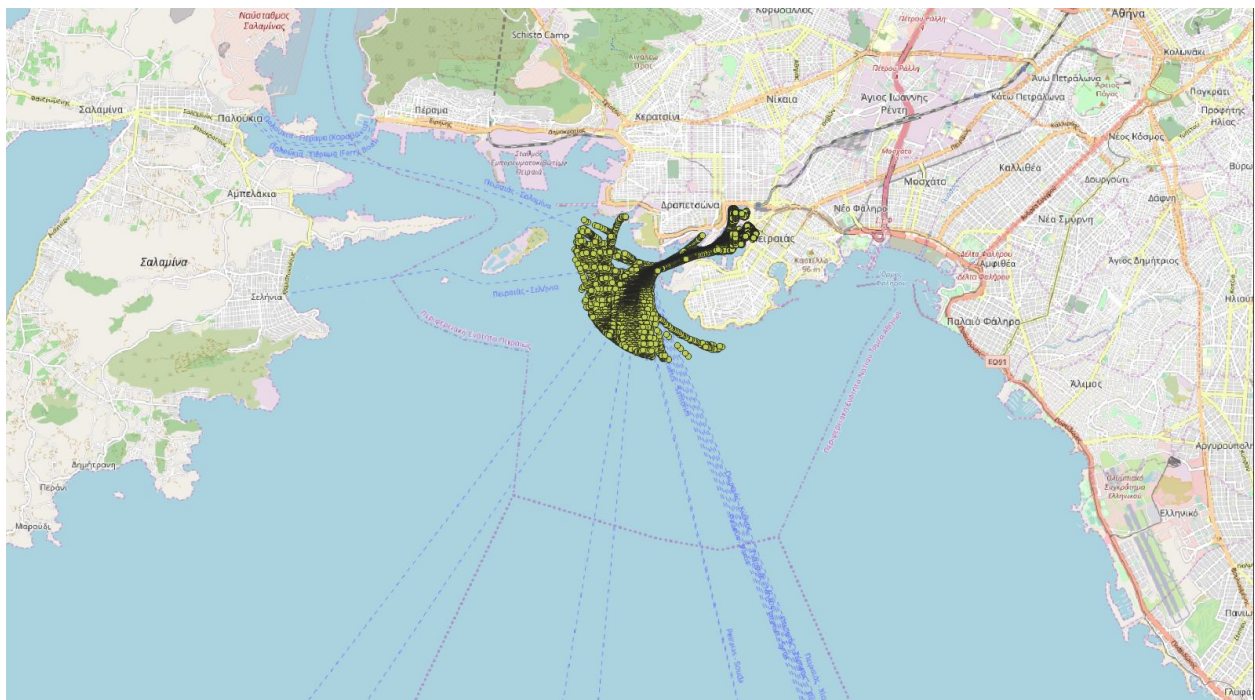


## Επεξεργασία και Αναλυτική των Δεδομένων

Αφού φορτώσαμε τα δεδομένα από τη βάση, με την εντολή `.sindex` δημιουργήσαμε ένα ευρετήριο τύπου R-Tree στη στήλη της Γεωμετρίας του πίνακα μας. Κάνουμε `intersect` το ευρετήριο με τα MSR των πολυγώνων. Αυτό μας δίνει τις τιμές οι οποίες είναι πιθανόν να βρίσκονται μέσα στο πολύγωνο.

Αυτές οι τιμές δεν έχουν `false negatives`. Είναι όμως πιθανό να έχουν `false positives`, δηλαδή κάποιες από τις τιμές να βρίσκονται μέσα στο MSR χωρίς να βρίσκονται στο πολύγωνο.

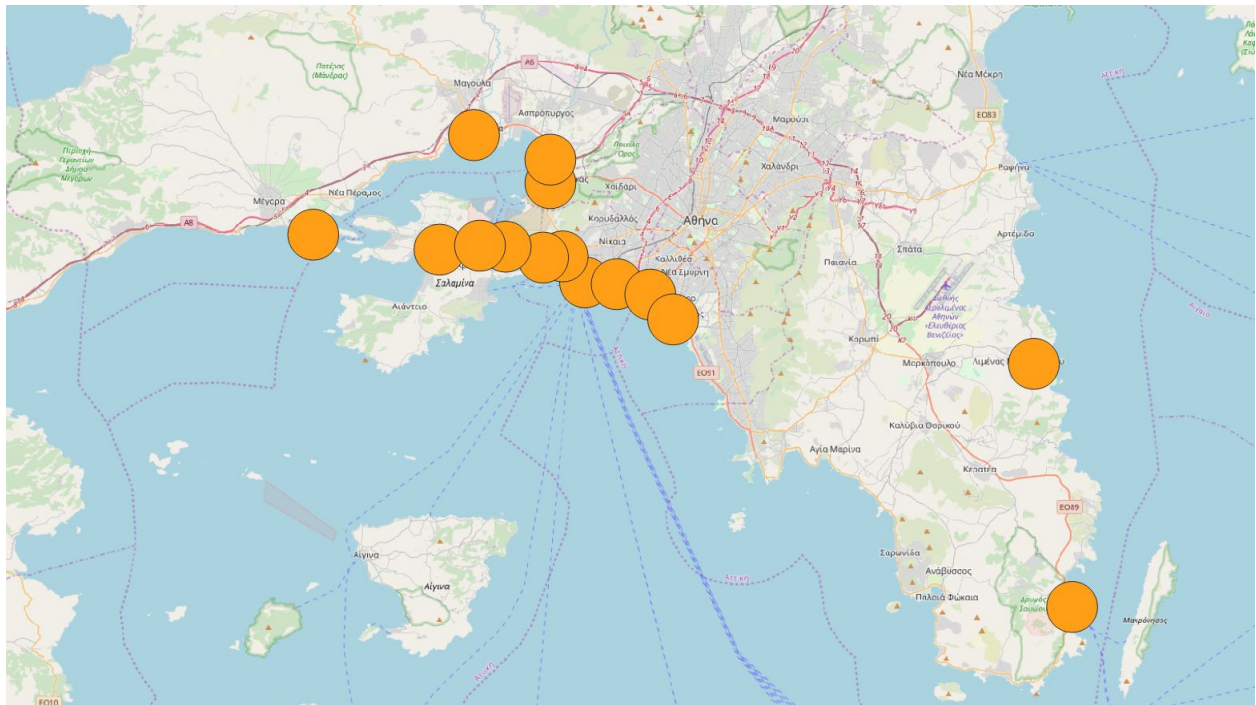
Τέλος, ελέγχουμε τις παραπάνω τιμές για να δούμε ποιές βρίσκονται όντως εντός πολυγώνου, να βρούμε, δηλαδή τα σημεία τα οποία βρίσκονταν εντός του λιμανιού του Πειραιά και σε απόσταση 2 χιλιομέτρων από τη μέση χρονική στιγμή της βάσης μας και μετά.



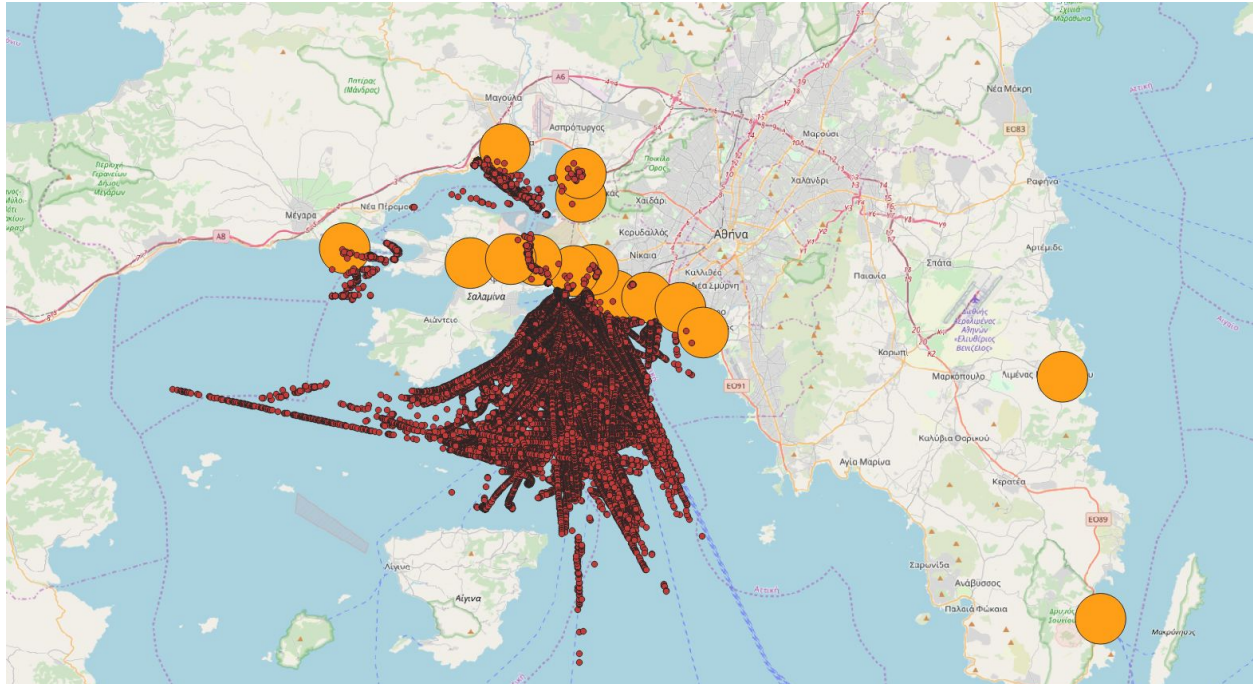
```
# R-tree search
# find the points that are inside the port of Piraeus after the given time
poss_matches_index = list(sindex.intersection(temp.bounds))
poss_matches = traj_gdf.iloc[poss_matches_index]
precise_matches = poss_matches[poss_matches.intersects(temp)]
points_within_geom = precise_matches.drop_duplicates(subset=['mmsi', 'ts'])
points_outside_geom = traj_gdf[~traj_gdf.isin(points_within_geom)].dropna(how='all')
points_in_piraeus = points_within_geom[points_within_geom['ts'] > 1519000000]
```

Για να ορίσουμε τη τροχιά θεωρήσαμε ότι τα πλοία τα οποία βρίσκονται εντός των λιμανιών - μαρινών του Αργοσαρωνικού δε μας δίνουν κάποια πληροφορία για τη τροχιά ή το ταξίδι τους. Οπότε δημιουργήσαμε ένα πίνακα στη βάση (Geometry Type POINT 4326) με τα ονόματα και τις συντεταγμένες όλων των λιμανιών που περικλείονται από το buffer που δημιουργήσαμε στην

ενότητα 2 (ακτίνας 37km από το λιμάνι του Πειραιά). Έπειτα, καθώς τα λιμάνια ήταν τύπου POINT δημιουργήσαμε ένα buffer γύρω από αυτά ακτίνας 2 χιλιομέτρων ώστε να κάνουμε τα λιμάνια πολύγωνα και να έχει νόημα η σύγκριση.



Με τη χρήση του R-Tree του προηγούμενου ερωτήματος και ένα for-loop ελέγξαμε, βρήκαμε και μαρκάραμε τις τιμές που βρίσκονται μέσα στις περιοχές των λιμένων-μαρινών ακολουθώντας την ίδια διαδικασία με πριν. Αφού κατηγοριοποιήσαμε τα δεδομένα χρειάστηκε να φτιάξουμε segments. Για αυτά χρειαστήκαμε τα σημεία εισόδου και εξόδου κάθε πλοίου τα οποία τα μαρκάραμε όπως φαίνεται στον κωδικά. Τέλος θέσαμε ένα κατώφλι 12 ωρών για να χωρίσουμε τα πλοία τα οποία δεν είχαν μετακινηθεί σε αυτό το διάστημα. Στο ακόλουθο screenshot φαίνονται όλα τα trips που δημιουργήθηκαν από το pipeline μας



number_of_entries_before_removing_ports bigint	number_of_entries_after_removing_ports bigint
11966339	2209745

Για την αναζήτηση ομοιότητας χρησιμοποιήθηκε ο αλγόριθμος dynamic time warping(DTW) απο την tslearn.metrix.dtw. Ο αλγόριθμος αυτός πραγματοποιήθηκε ως εξής:

- Έγινε μία ταξινόμηση στα δεδομένα ως προς το trip και τον χρόνο
- Έγινε αρχικοποίηση ενός ταξιδιού που θα γίνει με αυτό η σύγκριση
- Για κάθε trip στο dataset μας:
  - Βρέθηκε ξεχωριστά η απόσταση του lat,lon από το κάθε trip στο trip στόχο
  - Αν αυτή ήταν μικρότερη της προηγούμενης καταγεγραμμένης τιμής τότε αυτή ενημερώνεται
  - Για την χωροχρονική ομοιότητα προστέθηκε ένα τρίτο κόστος στο άθροισμα αυτο της απόστασης του χρόνου



```

# initialising the trip that we are going to compare to

gdf1 = traj_gdf[traj_gdf['trip_id'] == reference_id]
lat1 = gdf1['lat']
lon1 = gdf1['lon']
ts1 = gdf1['ts']

# starting the loop, we are going to compare to each trip of the data set in order to find the one with the smallest score
for trip in traj_gdf.trip_id.unique():
    if trip == reference_id:
        continue
    gdf = traj_gdf[traj_gdf['trip_id'] == trip]
    temp_lat = gdf['lat']
    temp_lon = gdf['lon']
    temp_ts = gdf['ts']
    score_lat = dtw(lat1, temp_lat)
    score_lon = dtw(lon1, temp_lon)
    score_ts = dtw(ts1, temp_ts)
    latlon_score = score_lat + score_lon
    ts_score = latlon_score + score_ts
    # if core is less than the previous recorded sve it
    if latlon_score < dtw_score_latlon:
        dtw_score_latlon = latlon_score
        id_of_best_latlon = trip
    if ts_score < dtw_score_ts:
        dtw_score_ts = ts_score
        id_of_best_ts = trip

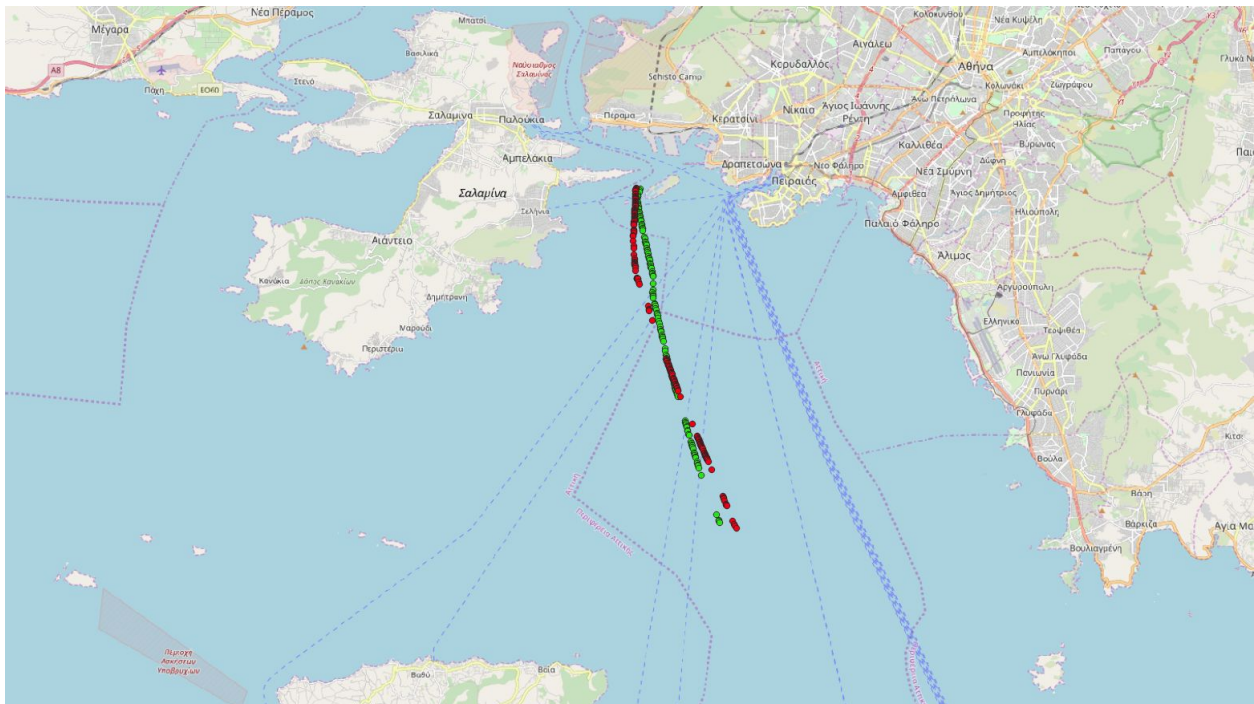
```

```

E:\Users\George\Anaconda\python.exe C:/Users/George/Desktop/unipi/GIS/similarity_search.py
Connecting with DB...
Connected
Fetching main table...
Table fetched
The best matching trip for just lat,lon is 4047
With a score of 0.045568555120173576
The best matching trip for lat,lon and time is 5964
With a score of 8642.252988498734

Process finished with exit code 0

```



Στην αναδειγματοληψία, με την χρήση της pandas και αφού φορτώσαμε τα δεδομένα σε ένα geodataframe κρατήσαμε μόνο τις στήλες που χρειαζόμασταν. Κάναμε index της βάσης την στήλη της ημερομηνίας(ts) για να μπορέσει να γίνει το resampling. Στην συνέχεια κάναμε resample τα δεδομένα με αναγωγή στα 2 δευτερόλεπτα. Με την χρήση της interpolate() που συμπληρώνει τα κενά που δημιουργούνται με ενδιάμεσες τιμές. Τέλος προσθέτει το καινούριο dataset μετά την αναδειγματοληψία σε ένα csv αρχείο.

```

# getting table from DB
print('Fetching main table...')
traj_sql = "SELECT ts,mmsi,lat,lon, trip_id,geom FROM curated ORDER BY trip_id,ts ;"
traj_gdf = gpd.GeoDataFrame.from_postgis(traj_sql, con, geom_col="geom")
print('Table fetched')
con.close()

# creating a DF to save the data after resampling
resampled_data = pd.DataFrame()
# start the resampling
print('Resampling is initiated')
for trip in traj_gdf.trip_id.unique():
    # take each trip
    gdf = traj_gdf[traj_gdf['trip_id'] == trip]
    # remove geom
    tmp = gdf.drop(['geom'], axis=1)
    tmp.drop_duplicates(subset='ts', keep=False, inplace=True)
    tmp['ts'] = pd.to_datetime(tmp['ts'], unit='s')
    tmp.set_index(['ts'],inplace=True)
    # resample to a new index then interpolate each column separately
    rs = tmp.resample("2S").asfreq()
    rs['mmsi'] = rs['mmsi'].interpolate()
    rs['lat'] = rs['lat'].interpolate()
    rs['lon'] = rs['lon'].interpolate()
    rs['trip_id'] = rs['trip_id'].interpolate()
    resampled_data = resampled_data.append(rs)

print('Resampling finished')
print('Creating CSV file')
# saving to csv file
resampled_data.dropna(subset=['trip_id','lat','lon']).to_csv(r'E:\unipi\GIS\resampled2s.csv', header=True)

```

2018-03-30 09:31:10	24116000.0	37.940032	23.576882	0.0
2018-03-30 09:31:12	24116000.0	37.939940	23.576866	0.0
2018-03-30 09:31:14	24116000.0	37.939849	23.576850	0.0
2018-03-30 09:31:16	24116000.0	37.939758	23.576834	0.0
2018-03-30 09:31:18	24116000.0	37.939666	23.576818	0.0
2018-03-30 09:31:20	24116000.0	37.939575	23.576802	0.0

Με την χρήση της βιβλιοθήκης sklearn φορτώσαμε τους αλγόριθμους DBSCAN ,OPTICS για να βρούμε τα hot spots μέσα στο data set μας , αυτοί οι αλγόριθμοι έδωσαν σαν αποτέλεσμα ένα σύνολο δεδομένων με labels και από αυτά τα labels βρήκαμε τα κεντροειδή της κάθε κλάσης.



```

X = np.radians(traj_gdf[['lat', 'lon']])
print("Starting dbscan")
db = DBSCAN(eps=1/6371, min_samples=len(traj_gdf)//50, algorithm='ball_tree', metric='haversine').fit(X)
print('Finding labels of clusters')
set(db.labels_)
print(f'We have{len(np.unique(db.labels_))} : {np.unique(db.labels_)} clusters')
cent = helper.get_clusters_centers(X, db.labels_)
print(cent)

print("Starting OPTICS")
optics = OPTICS(max_eps=1/6371, min_samples=len(traj_gdf)//50, metric='haversine').fit(X)
print('Finding labels of clusters')
set(optics.labels_)
print(f'We have{len(np.unique(optics.labels_))} : {np.unique(optics.labels_)} clusters')
opt = helper.get_clusters_centers(X, optics.labels_)
print(opt)

```

Για την εύρεση των ομάδων πλοίων χρησιμοποιήθηκαν τα evolving clusters. Φορτώθηκε το dataset από ένα csv και το μετατρέψαμε με τέτοιο τρόπο ώστε να μπορεί να το επεξεργαστεί ο αλγόριθμος. Λόγω έλλειψης υπολογιστικών πόρων και χρόνου, δεν ήταν δυνατή η εκτέλεση του αλγορίθμου σε όλο το dataset, οπότε παρότι ο αλγόριθμος έτρεξε σωστά δεν βρήκε κάποιο cluster.

```

df1 = pd.read_csv(r'C:\Users\George\Desktop\unipi\GIS\csv\resampled2s.csv')
print(df1.head(10))
df1.columns = ['datetime', 'lat', 'lon', 'trip', 'mmsi']
print(df1.head(10))
res = evolving_clusters(df1, 'convoys', min_cardinality=5, disable_progress_bar=False)
print(res.head())

```

## Επεξεργασία Μεγάλων Δεδομένων (Spark)

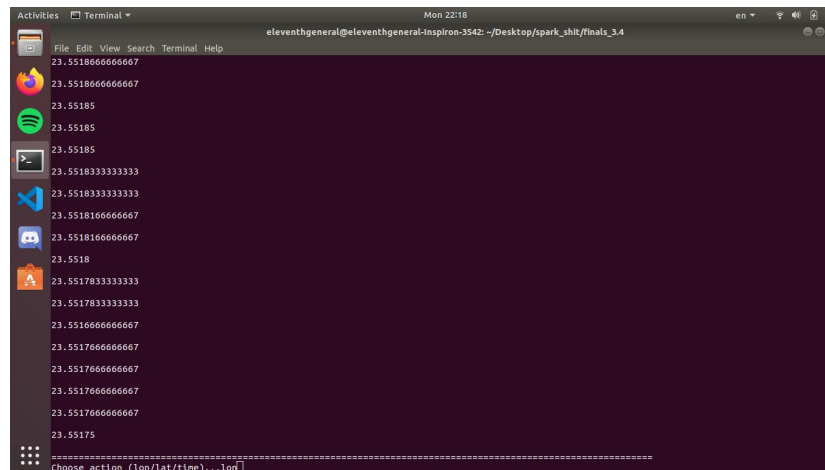
Σε ότι αφορά τα big spatial data και το Spark, χρησιμοποιήσαμε τη PySpark βιβλιοθήκη της Python και πιο συγκεκριμένα τα :

- pyspark.sql, SparkSession
- pyspark, SparkContext
- pyspark.sql, SQLContext

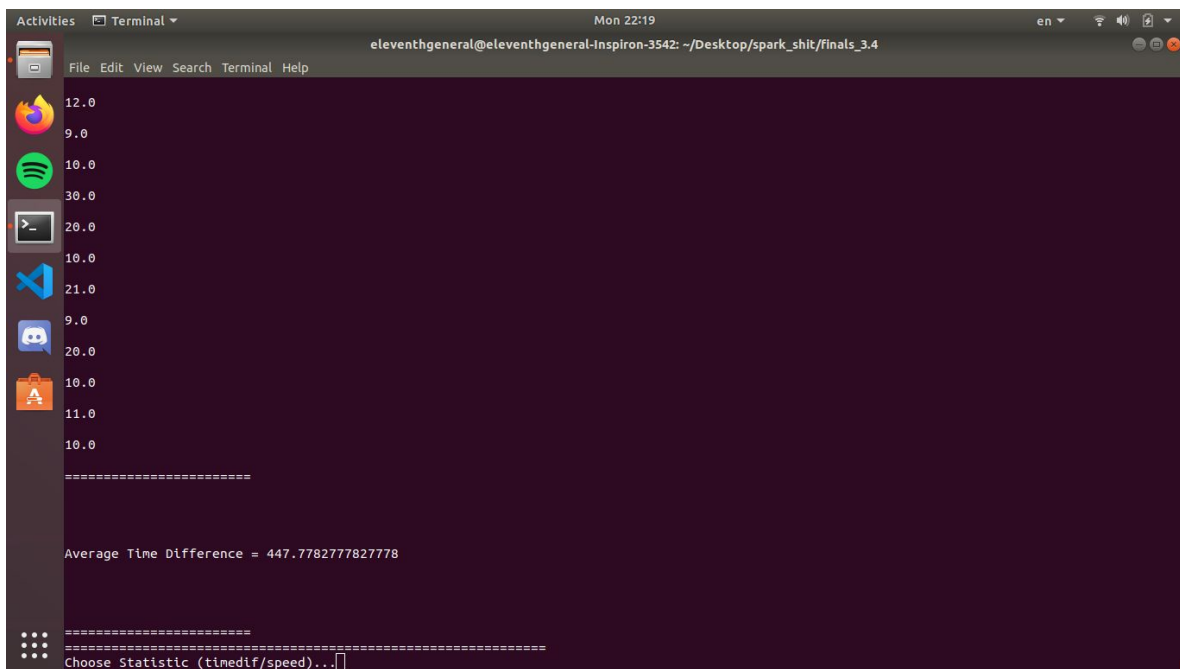
Ανοίγουμε το spark session και φορτώνουμε τη βάση.

```
sc= SparkContext()  
data = sc.textFile('curated_final.csv')  
prettySummary(data)
```

Ελέγξαμε αν το πρόγραμμα μπορεί να εμφανίσει ξεχωριστά τα δεδομένα που θέλουμε.



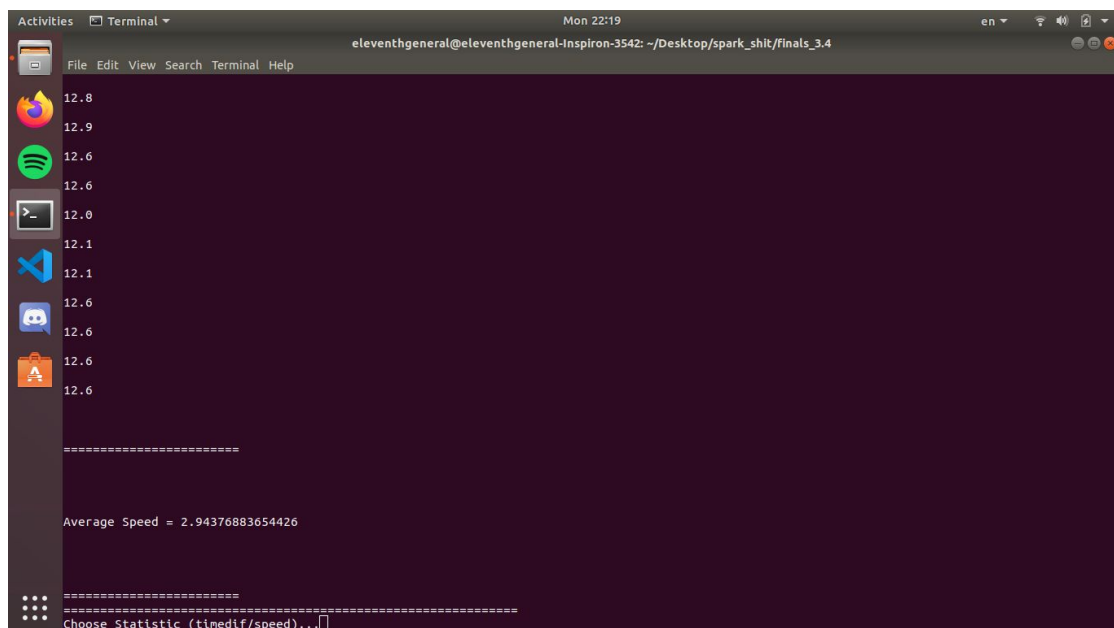
Τέλος για τα στατιστικά της ενότητας 3.2 ψάξαμε να εμφανίσουμε μέσω Spark το ρυθμό δειγματοληψίας (Time Difference)



A terminal window titled 'eleventhgeneral@eleventhgeneral-Inspiron-3542: ~/Desktop/spark\_shit/finals\_3.4' displays the following output:

```
12.0
9.0
10.0
30.0
20.0
10.0
21.0
9.0
20.0
10.0
11.0
10.0
=====
Average Time Difference = 447.7782777827778
=====
Choose Statistic (timedif/speed)...
```

και τη μέση ταχύτητα των σημείων της βάσης.



A terminal window titled 'eleventhgeneral@eleventhgeneral-Inspiron-3542: ~/Desktop/spark\_shit/finals\_3.4' displays the following output:

```
12.8
12.9
12.6
12.6
12.0
12.1
12.1
12.6
12.6
12.6
12.6
=====
Average Speed = 2.94376883654426
=====
Choose Statistic (timedif/speed)...
```

Παρατηρούμε ότι τα αποτελέσματα που προκύπτουν από το Spark είναι ίδια με τα αποτελέσματα που βρήκαμε στην ενότητα 3.2.