Τμήμα Μηχανικών Η/Υ και Πληροφορικής
Πανεπιστήμιο Πατρών
Πολυτεχνική Σχολή

Τομέας Λογικού των Υπολογιστών
Διδάσκοντες: Βασίλειος Μεγαλοοικονόμου, Χρήστος Μακρής
Ακαδημαϊκό Έτος: 2023 – 2024
Ημ / νία Παράδοσης: 09/ 06 / 2024

Εξόρυξη Δεδομένων και Αλγόριθμοι Μάθησης

Επιλεγόμενο Μάθημα – CEID_NE562

Εργαστηριακή Άσκηση
Εαρινό Εξάμηνο 2024

Χρυσαυγή Πατέλη | 1084513 | up1084513@ac.upatras.gr
Μηλτιάδης Μαντές | 1084661 | up1084661@ac.upatras.gr

Περιεχόμενα

1 Εισαγωγή

1.0 Σύνολο Δεδομένων

Σύνολο Δεδομένων: **HARTH**

Το σύνολο δεδομένων **Human Activity Recognition Trondheim** (**HARTH**) περιέχει καταγραφές από 22 συμμετέχοντες που φορούσαν δύο 3-αξονικά επιταχυνσιόμετρα Axivity AX3 για περίπου 2 ώρες σε ένα ελεύθερο περιβάλλον. Ένας αισθητήρας ήταν τοποθετημένος στον δεξιό μπροστινό μηρό και ο άλλος στη χαμηλή πλάτη. Ο παρεχόμενος ρυθμός δειγματοληψίας είναι 50Hz.

Οι καταγραφές κάθε συμμετέχοντα παρέχονται σε ξεχωριστό αρχείο .csv. Κάθε τέτοιο αρχείο .csv περιέχει τις ακόλουθες στήλες:

- **timestamp**: ημερομηνία και ώρα του καταγεγραμμένου δείγματος
- **back_x**: επιτάχυνση του αισθητήρα στην πλάτη προς την κατεύθυνση x (κάτω) στη μονάδα g
- **back_y**: επιτάχυνση του αισθητήρα στην πλάτη προς την κατεύθυνση y (αριστερά) στη μονάδα g
- **back_z**: επιτάχυνση του αισθητήρα στην πλάτη προς την κατεύθυνση z (μπροστά) στη μονάδα g
- **thigh_x**: επιτάχυνση του αισθητήρα στον μηρό προς την κατεύθυνση x (κάτω) στη μονάδα g
- **thigh_y**: επιτάχυνση του αισθητήρα στον μηρό προς την κατεύθυνση y (δεξιά) στη μονάδα g
- **thigh_z**: επιτάχυνση του αισθητήρα στον μηρό προς την κατεύθυνση z (πίσω) στη μονάδα g
- **label**: κωδικός της καταγεγραμμένης δραστηριότητας

Το σύνολο δεδομένων περιέχει τις ακόλουθες καταγεγραμμένες δραστηριότητες με τους αντίστοιχους κωδικούς:

1: **walking**
2: **running**
3: **shuffling**
4: **stairs(ascending)**
5: **stairs(descending)**
6: **standing**
7: **sitting**
8: **lying**
13: **cycling(sit)**
14: **cycling(stand)**
130: **cycling (sit, inactive)**
140: **cycling (stand, inactive)**

1.1 Περιβάλλον Υλοποίησης & Βιβλιοθήκες

Περιβάλλον Υλοποίησης

Ως περιβάλλον υλοποίησης έχουμε επιλέξει το **Google Colab**, το οποίο μας προσφέρει μεγάλη διευκόλυνση όσον αφορά την υλοποίηση και τη ταχύτητα εκτέλεσης κώδικα πάνω σε μεγάλο όγκο δεδομένων. Πιο συγκεκριμένα, προσφέρει δωρεάν πρόσβαση σε *T4 GPU* (Graphics Processing Unit) και *TPU* (Tensor Processing Unit), επιτρέποντας έτσι την ταχύτερη εκτέλεση αλγορίθμων μηχανικής μάθησης και deep learning, το οποίο είναι και το κύριο αντικείμενο αυτής της εργασίας. Άλλωστε σε κάθε κελί ενός Notebook σε **Google Colab** μπορούμε να γράψουμε και να εκτελέσουμε κώδικα Python ακριβώς όπως και σε ένα τοπικό **Jupyter Notebook** χωρίς καμία απολύτως διαφορά. Τέλος, ένας άλλος λόγος που επιλέξαμε το **Google Colab** είναι ότι διευκόλυνε τη συνεργατική υλοποίηση της εργασίας, καθώς μπορούσαμε να μοιραστούμε τα Notebooks μας και να τα επεξεργαστούμε σε πραγματικό χρόνο.

Παρακάτω ακολουθεί και μια σύντομη περιγραφή όλων των βιβλιοθηκών λογισμικού που εγκαταστήσαμε στο περιβάλλον μας για την υλοποίηση της εργασίας:

Γενικές Βιβλιοθήκες

1. **os:** παρέχει διασύνδεση με το λειτουργικό σύστημα, επιτρέποντας την εκτέλεση εντολών για τη διαχείριση αρχείων και καταλόγων. Έτσι, μπορούμε να αλλάξουμε τον τρέχοντα κατάλογο και να διαβάσουμε το περιεχόμενο ενός καταλόγου στο κώδικά μας.
2. **pandas:** είναι μια ισχυρή βιβλιοθήκη για την ανάλυση και τη διαχείριση δεδομένων. Παρέχει δομές δεδομένων όπως DataFrame και Series που διευκολύνουν την επεξεργασία, τον καθαρισμό και την ανάλυση δεδομένων που διαβάζουμε από τα αρχεία CSV του dataset.
3. **numpy:** είναι η βασική βιβλιοθήκη για αριθμητικούς υπολογισμούς στη Python. Παρέχει υποστήριξη για πολυδιάστατους πίνακες και μια πληθώρα μαθηματικών συναρτήσεων για τη λειτουργία πάνω σε αυτούς τους πίνακες.
4. **matplotlib.pyplot:** είναι ένα υποσύνολο της βιβλιοθήκης **matplotlib** και παρέχει ένα σύνολο εντολών που επιτρέπουν τη δημιουργία διαγραμμάτων και γραφημάτων.

Βιβλιοθήκες 1ου Ερωτήματος

1. **seaborn:** είναι μια βιβλιοθήκη για τη στατιστική απεικόνιση των δεδομένων μας που βασίζεται στη **matplotlib**. Παρέχει ευκολότερους τρόπους για τη δημιουργία σύνθετων γραφημάτων. Στο συγκεκριμένο ερώτημα εμείς τη χρησιμοποιούμε για την υλοποίηση γραφημάτων διασποράς και θερμοχάρτες (heatmaps) που προκύπτουν από τη στατιστική επεξεργασία των δεδομένων.

Βιβλιοθήκες 2ου Ερωτήματος

1. **sklearn.metrics:** είναι υποσύνολο της βιβλιοθήκης **sklearn** η οποία χρησιμοποιείται για μηχανική μάθηση και παρέχει εργαλεία για την αξιολόγηση της απόδοσης μοντέλων μηχανικής μάθησης. Περιλαμβάνει μετρικές για ταξινόμηση, παλινδρόμηση και ομαδοποίηση, όπως **accuracy_score**, **precision_score**, **recall_score**, **f1_score**, **confusion matrix** κ.ά.
2. **sklearn.ensemble:** είναι υποσύνολο της βιβλιοθήκης **sklearn** και περιλαμβάνει αλγορίθμους συνόλων (ensemble algorithms) που συνδυάζουν τις προβλέψεις πολλαπλών μοντέλων για τη βελτίωση της απόδοσης και της ακρίβειας. Το μοντέλο που θα αξιοποιήσουμε εμείς από αυτή τη βιβλιοθήκη είναι αυτό του **Random Forest**.
3. **tensorflow:** είναι μια βιβλιοθήκη ανοιχτού κώδικα για τη μηχανική μάθηση και την ανάπτυξη νευρωνικών δικτύων μέσω του *Keras API*. Από το *Keras API* εμείς αξιοποιούμε για την υλοποίηση του **Artificial Neural Network** το Sequential μοντέλο και το στρώμα Dense.
4. **pgmpy.models:** είναι μέρος της βιβλιοθήκης **pgmpy** που χρησιμοποιείται για τη δημιουργία και τη διαχείριση μοντέλων γραφικών πιθανοτήτων (probabilistic graphical models). Αυτά τα μοντέλα περιλαμβάνουν τα **Bayesian Networks** που θα αξιοποιήσουμε εμείς.
5. **pgmpy.estimators:** είναι μέρος της βιβλιοθήκης **pgmpy** και παρέχει μεθόδους για την εκτίμηση παραμέτρων και δομών για τα μοντέλα γραφικών πιθανοτήτων. Χρησιμοποιείται για την εκμάθηση της δομής

του δικτύου και των πιθανοτήτων από τα δεδομένα σε ένα **Bayesian Network**. Από αυτές τις μεθόδους εμείς αξιοποιούμε τη **MaximumLikelihoodEstimator**, τη **HillClimbSearch** και τη **BicScore**.

6. **pgmpy.inference:** είναι μέρος της βιβλιοθήκης **pgmpy** και παρέχει εργαλεία και αλγορίθμους για τον υπολογισμό πιθανοτήτων και εκτίμηση μεταβλητών σε πιθανοτικά γραφικά μοντέλα. Η ανάλυση που παρέχει βοηθά στην κατανόηση των συμπερασμάτων που μπορούν να εξαχθούν από τα πιθανοτικά μοντέλα και τις εκτιμήσεις που μπορούν να πραγματοποιηθούν. Για το Bayesian Network αξιοποιήσαμε τη κλάση **VariableElimination** για τον υπολογισμό πιθανοτήτων ή αναμενόμενων τιμών.
7. **sklearn.preprocessing:** είναι υποσύνολο της βιβλιοθήκης **sklearn** και παρέχει εργαλεία για την προεπεξεργασία δεδομένων. Κάποια από αυτά τα εργαλεία τα οποία θα αξιοποιήσουμε είναι η κανονικοποίηση (normalization), και η μετατροπή κατηγοριών σε δυαδικούς δείκτες (one-hot encoding).

Βιβλιοθήκες 3ου Ερωτήματος

1. **sklearn.preprocessing:** είναι υποσύνολο της βιβλιοθήκης **sklearn** και παρέχει εργαλεία για την προεπεξεργασία δεδομένων. Κάποια από αυτά τα εργαλεία τα οποία θα αξιοποιήσουμε είναι η κανονικοποίηση (normalization), και η μετατροπή κατηγοριών σε δυαδικούς δείκτες (one-hot encoding).
2. **sklearn.cluster:** είναι υποσύνολο της βιβλιοθήκης **sklearn** και περιλαμβάνει αλγορίθμους για την ομαδοποίηση (clustering) δεδομένων, όπως **K-means**.
3. **sklearn.mixture:** είναι υποσύνολο της βιβλιοθήκης **sklearn** και παρέχει μοντέλα μείγματος (mixture models), όπως **Gaussian Mixture Model (GMM)** το οποίο και χρησιμοποιήσαμε.
4. **mpl_toolkits.mplot3d:** είναι μια υποβιβλιοθήκη της **matplotlib** που επιτρέπει τη δημιουργία τρισδιάστατων γραφημάτων. Αξιοποιήθηκε για τη τρισδιάστατη αναπαράσταση των συστάδων στις οποίες χωρίσαμε τα δεδομένα για κάθε έναν από τους δύο αισθητήρες.
5. **minisom:** είναι μια απλή και γρήγορη βιβλιοθήκη για την εκπαίδευση και την εφαρμογή αυτοοργανωμένων χαρτών (Self-Organizing Maps, SOMs). Τα SOMs είναι ένας τύπος τεχνητών νευρωνικών δικτύων που χρησιμοποιούνται για την ομαδοποίηση και τη μείωση της διάστασης των δεδομένων. Μέσω αυτής της βιβλιοθήκης υλοποιήσαμε έτσι τη συσταδοποίηση μέσω **Kohonen Networks**.

2 Ερώτημα 1ο

2.0 Περιγραφή

Για να κατανοήσουμε περισσότερο τη δομή του συνόλου δεδομένων που θα μελετήσουμε καθώς και τη κατανομή και τις συσχετίσεις των μετρήσεων των αισθητήρων σε αυτό, πραγματοποιούμε στατιστική επεξεργασία τόσο με **αριθμητικό περιγραφικό** τρόπο όσο και με **γραφικό**.

Όσον αφορά τα **αριθμητικά περιγραφικά μέτρα**, πρόκειται για ποσοτικά μεγέθη που βοηθούν στην περιγραφή της κατανομής του συνόλου δεδομένων με όρους ποσοτικούς. Στη προκειμένη περίπτωση θα χρησιμοποιήσουμε στατιστικά **μέτρα θέσης** (μέση τιμή, διάμεσος, 25ο και 75ο εκατοστημόριο) και **μεταβλητότητας** (εύρος συνόλου δεδομένων ή αλλιώς μέγιστη και ελάχιστη τιμή, τυπική απόκλιση).

Στατιστικά Μέτρα Θέσης:

1. **Μέση Τιμή (Mean):** Υπολογίζει τον αριθμητικό μέσο όρο των τιμών κάθε γνωρίσματος.
2. **Διάμεσος (50%):** Η τιμή που χωρίζει τα δεδομένα σε δύο ίσα μέρη όταν αυτά ταξινομούνται.
3. **25ο και 75ο Εκατοστημόριο (Quartiles):** Τα σημεία που χωρίζουν το κατώτερο 25% και 75% των δεδομένων.

Στατιστικά Μέτρα Μεταβλητότητας:

1. **Εύρος (Range):** Η διαφορά μεταξύ της μέγιστης και ελάχιστης τιμής.
2. **Τυπική Απόκλιση (Standard Deviation):** Ένα μέτρο της διασποράς των τιμών γύρω από τη μέση τιμή.

Όσον αφορά την **γραφική αναπαράσταση** των δεδομένων αξιοποιούμε ένα **ιστόγραμμα κατανομής συχνοτήτων** κάθε γνωρίσματος, καθώς και ένα **θερμοχάρτη (heatmap)** για τον εντοπισμό της ετεροσυσχέτισης ανάμεσα στα 6 γνωρίσματα του συνόλου δεδομένων. Το ιστόγραμμα συχνοτήτων επιπλέον προσαρμόζεται κατάλληλα για να απεικονίζεται η πυκνότητα πιθανότητας κάθε γνωρίσματος και όχι οι απόλυτες συχνότητες εμφάνισής του. Αυτό σημαίνει ότι το άθροισμα των περιοχών των ράβδων θα είναι ίσο με 1, καθιστώντας το ιστόγραμμα κατάλληλο για τη σύγκριση με κανονικοποιημένες κατανομές. Έχουμε επιλέξει πάνω σε κάθε ιστόγραμμα να εμφανίζεται γραφικά και η αντίστοιχη **κατανομή Gauss** την οποία προσεγγίζει η κάθε συνάρτηση πυκνότητας πιθανότητας, προκειμένου η σύγκριση να είναι πιο κατανοητή. Ακόμη, για τον θερμοχάρτη επιλέγουμε να χρησιμοποιήσουμε τον **συντελεστή συσχέτισης Pearson** για να μετρήσουμε τη γραμμική συσχέτιση μεταξύ των γνωρισμάτων.

Τέλος, εφόσον τα δεδομένα μας μεταβάλλονται και με βάση το χρόνο, πέρα από στατιστική επεξεργασία κάνουμε και απεικόνιση της **χρονοσειράς** κάθε γνωρίσματος. Η χρησιμότητα της χρονοσειράς έγκειται στο ότι μπορούμε να αναγνωρίσουμε πιο εύκολα τάσεις, διακυμάνσεις, μοτίβα και ανωμαλίες στα δεδομένα με τη πάροδο του χρόνου.

Πριν προχωρήσουμε στην υλοποίηση, σημειώνουμε ότι έχουμε πραγματοποιήσει στατιστική επεξεργασία των γνωρισμάτων του συνόλου δεδομένων τόσο συγκεντρωτικά (λαμβάνοντας υπόψη τις μετρήσεις των αισθητήρων από όλους τους συμμετέχοντες ταυτόχρονα), όσο και ατομικά (λαμβάνοντας υπόψη τις μετρήσεις από κάθε συμμετέχοντα ξεχωριστά). Επιπλέον, έχουμε επεξεργαστεί στατιστικά όχι μόνο τα δεδομένα ως προς τους συμμετέχοντες αλλά και ως προς τις διακριτές φυσικές δραστηριότητες που πραγματοποιούν. Άρα συνολικά έχουμε πραγματοποιήσει 3 φάσεις επεξεργασίας: **Συνολική Στατιστική Επεξεργασία**, **Στατιστική Επεξεργασία ανά Συμμετέχοντα** και **Στατιστική Επεξεργασία ανά Δραστηριότητα (label)**.

2.1 Υλοποίηση

2.1.0 Συνολική Στατιστική Επεξεργασία

Αρχικά, ορίζουμε το μονοπάτι του φακέλου που περιέχει τα αρχεία CSV και μέσω της **os.chdir( )** αλλάζουμε τον τρέχοντα κατάλογο εργασίας με το μονοπάτι που είναι αποθηκευμένο το dataset, ώστε να μπορεί να διαβάσει το πρόγραμμα τα αρχεία CSV από αυτόν τον φάκελο. Έπειτα, για κάθε αρχείο (δηλαδή για κάθε συμμετέχοντα) αποθηκεύουμε τις στήλες του σε ένα ξεχωριστό DataFrame και έπειτα όλα τα DataFrames από όλα τα αρχεία αποθηκεύονται στη λίστα **dfs**. Μέσω της **pd.concat( )** όλα τα DataFrame ενώνονται σε ένα **merged_df**, από το οποίο μέσω της **drop( )** αφαιρούμε όλες τις αχρείαστες στήλες. Τέλος, με τη **describe( )** εμφανίζονται στην οθόνη τα βασικά στατιστικά μεγέθη που αναφέραμε πιο πάνω συγκεντρωτικά για όλο το dataset. Έπειτα, δημιουργούμε ένα ιστόγραμμα με τη χρήση της μεθόδου **ax.hist()** καθένα από τα οποία αναπαριστά τη συχνότητα εμφάνισης των διαφορετικών τιμών σε κάθε μία από τις 6 στήλες με τα δεδομένα των αισθητήρων.

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Μονοπάτι για τον φάκελο που περιέχει τα αρχεία CSV
path = '/content/drive/MyDrive/harth'
os.chdir(path)

# Πραγματοποίηση πέρασματος σε κάθε αρχείο CSV
dfs = []
for filename in os.listdir(path):
    if filename.endswith('.csv'):
        df = pd.read_csv(os.path.join(path, filename))
        dfs.append(df)

# Συγχώνευση όλων των DataFrame σε ένα
merged_df = pd.concat(dfs, ignore_index=True)
# Αφαίρεση των αχρείαστων στηλών 'index', 'label' και 'Unnamed: 0'
merged_df.drop(['index', 'label', 'Unnamed: 0'], axis=1, inplace=True)
print(merged_df.describe().T)
```

Αρχικά, ορίζουμε το μονοπάτι του φακέλου που περιέχει τα αρχεία CSV και μέσω της **os.chdir( )** αλλάζουμε τον Έπειτα, δημιουργούμε ένα ιστόγραμμα με τη χρήση της μεθόδου **ax.hist( )** καθένα από τα οποία αναπαριστά τη συχνότητα εμφάνισης των διαφορετικών τιμών σε κάθε μία από τις 6 στήλες με τα δεδομένα των αισθητήρων. Για κάθε στήλη, προσεγγίζουμε την κατανομή των δεδομένων με μια κανονική κατανομή. Υπολογίζουμε τη μέση τιμή **mu** και την τυπική απόκλιση **sigma** των δεδομένων του **merged_df** με τις συναρτήσεις **mean( )** και **std( )** και σχεδιάζουμε την κατανομή πιθανότητας της κανονικής κατανομής **p** πάνω από το αντίστοιχο ιστόγραμμα.

```
# Αναπαράσταση γραφικών της συχνότητας των τιμών για τις 6 στήλες
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
fig.suptitle('Appearance Frequency Distributions of Features', fontsize=10)

for ax, column in zip(axes.flatten(), ['back_x', 'back_y', 'back_z', 'thigh_x',
'thigh_y', 'thigh_z']):
    ax.hist(merged_df[column], bins=35, color='skyblue', edgecolor='black',
density=True)
    ax.set_title(f'{column}', fontsize=10)
    ax.set_ylabel('Density', fontsize=8)
    ax.tick_params(axis='both', which='major', labelsize=6)
    ax.grid(True)

    # Προσεγγίζουμε τη κατανομή συχνοτήτων με μια Κανονικη Κατανομή
    mu, sigma = merged_df[column].mean(), merged_df[column].std()
    xmin, xmax = merged_df[column].min(), merged_df[column].max()
    x = np.linspace(xmin, xmax, 50000)
    p = (1 / (sigma * np.sqrt(2 * np.pi))) * np.exp(-(x - mu) ** 2 / (2 * sigma ** 2))
    ax.plot(x, p, 'k', linewidth=1)
```

```
plt.tight_layout()
plt.show()
```

Με την εντολή **merged_df[['back_x', 'back_y', 'back_z', 'thigh_x', 'thigh_y', 'thigh_z']].corr( )** υπολογίζουμε τον πίνακα συσχέτισης για τις επιλεγμένες στήλες του DataFrame. Η συνάρτηση **corr( )** υπολογίζει το συντελεστή συσχέτισης Pearson για κάθε ζεύγος στηλών. Επίσης, η **sns.heatmap( )** υλοποιεί το θερμοχάρτη με βάση τις τιμές των συσχετίσεων που υπολογίσθηκαν.

```
# Δημιουργία του heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(merged_df[['back_x', 'back_y', 'back_z', 'thigh_x', 'thigh_y',
'thigh_z']].corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Heatmap of Feature Correlations')
plt.show()
```

Με βάση τον θερμοχάρτη που εμφανίστηκε εντοπίζουμε τα 4 ζεύγη γνωρισμάτων με τη μεγαλύτερη συσχέτιση (**thigh_x** και **back_x**, **thigh_x** και **thigh_z**, **back_y** και **thigh_y, back_z** και **thigh_z**) και με την συνάρτηση **scatter( )** εκτυπώνουμε για κάθε ζεύγος το scatter plot με όλα τα δεδομένα κάθε γνωρίσματος. Αυτή η απεικόνιση μπορεί να βοηθήσει στην αναγνώριση τυχόν γραμμικών σχέσεων ή άλλων μοτίβων μεταξύ των χαρακτηριστικών.

```
# Γραφική αναπαράσταση των 4 ζευγαριών από features με τις μεγαλύτερες ετεροσυσχετίσεις
fig, axs = plt.subplots(2, 2, figsize=(12, 8))

# Ζευγάρι 1: thigh_x και back_x
axs[0, 0].scatter(merged_df['thigh_x'], merged_df['back_x'], alpha=0.5)
axs[0, 0].set_xlabel('thigh_x')
axs[0, 0].set_ylabel('back_x')
axs[0, 0].set_title('Correlation between thigh_x and back_x')

# Ζευγάρι 2: thigh_x και thigh_z
axs[0, 1].scatter(merged_df['thigh_x'], merged_df['thigh_z'], alpha=0.5)
axs[0, 1].set_xlabel('thigh_x')
axs[0, 1].set_ylabel('thigh_z')
axs[0, 1].set_title('Correlation between thigh_x and thigh_z')

# Ζευγάρι 3: back_y και thigh_y
axs[1, 0].scatter(merged_df['back_y'], merged_df['thigh_y'], alpha=0.5)
axs[1, 0].set_xlabel('back_y')
axs[1, 0].set_ylabel('thigh_y')
axs[1, 0].set_title('Correlation between back_y and thigh_y')

# Ζευγάρι 4: back_z και thigh_z
axs[1, 1].scatter(merged_df['back_z'], merged_df['thigh_z'], alpha=0.5)
axs[1, 1].set_xlabel('back_z')
axs[1, 1].set_ylabel('thigh_z')
axs[1, 1].set_title('Correlation between back_z and thigh_z')

plt.tight_layout()
plt.show()
```

2.1.1 Στατιστική Επεξεργασία ανά Συμμετέχοντα

Τώρα θέλουμε να δημιουργήσουμε και εμφανίσουμε τις χρονοσειρές για κάθε αρχείο CSV που βρίσκεται στο συγκεκριμένο φάκελο με το dataset. Αρχικά, η λίστα **file_names** δημιουργείται για να αποθηκεύσει τα ονόματα των αρχείων CSV (δηλαδή των συμμετεχόντων). Αφού αποθηκεύσουμε πάλι κάθε αρχείο σε ένα DataFrame μέσω της **pd.to_datetime( )** μετατρέπουμε τη στήλη **'timestamp'** κάθε **df** από αλφαριθμητικό σε μορφή datetime. Τέλος, για κάθε χαρακτηριστικό (**back_x**, **back_y**, **back_z**, **thigh_x**, **thigh_y**, **thigh_z**), σχεδιάζεται η χρονοσειρά του μέσω της **plot( )**.

```
# Λίστα για τα ονόματα των συμμετεχόντων
file_names = []
```

```
# Σχεδιάζουμε τη χρονοσειρά για κάθε αρχείο CSV
for filename in os.listdir(path):
    file_path = os.path.join(path, filename)
    df = pd.read_csv(file_path)
    # Χωρίς κατάληξη
    base_name = os.path.splitext(filename)[0]

    # Μετατροπή του timestamp σε datetime
    df['timestamp'] = pd.to_datetime(df['timestamp'])

    # Δημιουργία γραφήματος για τα γνωρίσματα
    plt.figure(figsize=(12, 6))

    # Σχεδιάζουμε τη χρονοσειρά για κάθε γνώρισμα
    for column in ['back_x', 'back_y', 'back_z', 'thigh_x', 'thigh_y', 'thigh_z']:
        plt.plot(df['timestamp'], df[column], label=column)

    # Διαμόρφωση γραφήματος
    plt.title(f"Time Series for {base_name}")
    plt.xlabel("Time")
    plt.ylabel("Values")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()

    # Εμφάνιση γραφήματος
    plt.show()
```

Όμοια τώρα για κάθε αρχείο αποθηκεύουμε τα δεδομένα στο **df** και καλώντας πάνω σε αυτό τη **describe( )** εμφανίζονται τα ίδια στατιστικά μεγέθη για κάθε αρχείο ατομικά αντί από όλα τα αρχεία συνολικά όπως πριν.

```
# Πραγματοποίηση πέρασματος στο φάκελο για την εμφάνιση στατιστικών για κάθε αρχείο
ξεχωριστά
for filename in os.listdir(path):
    if filename.endswith('.csv'):
        df = pd.read_csv(os.path.join(path, filename))
        # Αφαίρεση της αχρείαστης στήλης 'label'
        df.drop(['label'], axis=1, inplace=True)
        # Χωρίς κατάληξη
        base_name = os.path.splitext(filename)[0]
        print(f"\n Statistic Values for Participant {base_name}")
        print(df.describe().T)
```

Αρχικά, ορίζουμε ένα λεξικό **mean_values_per_file** που θα αποθηκεύει τις μέσες τιμές των γνωρισμάτων για κάθε συμμετέχοντα. Επίσης, ορίζουμε τις λίστες **file_names** και **features** που αποθηκεύουν τα ονόματα των αρχείων (συμμετεχόντων) και των χαρακτηριστικών που θα χρησιμοποιήσουμε. Έπειτα, για κάθε χαρακτηριστικό διατρέχουμε όλα τα αρχεία CSV του φακέλου και κάνουμε εξαγωγή του ονόματος του αρχείου (χωρίς κατάληξη), το οποίο προστίθεται στη λίστα **file_names**. Στη συνέχεια, γίνεται υπολογισμός της μέσης τιμής του τρέχοντος χαρακτηριστικού για τον συμμετέχοντα μέσω της εντολής **np.mean(df[feature])** αλλά και αποθήκευση της μέσης τιμής στο λεξικό **mean_values_per_file**. Τέλος, πραγματοποιείται υπολογισμός της συνολικής μέσης τιμής του χαρακτηριστικού για όλους τους συμμετέχοντες και σχεδίαση των αντίστοιχων γραφικών παραστάσεων, αφού πρώτα τα ονόματα των συμμετεχόντων έχουν ταξινομηθεί αλφαβητικά μέσω της **sorted( )** με όρισμα τα κλειδιά του λεξικού **mean_values_per_file** που επιστρέφει η **keys( )**.

```
# Λίστα με τα ονόματα των συμμετεχόντων
file_names = []

# Λεξικό που θα διατηρεί τις μέσες τιμές των γνωρισμάτων ανά συμμετέχοντα
mean_values_per_file = {}

# Λίστα με τα features
features = ['back_x', 'back_y', 'back_z', 'thigh_x', 'thigh_y', 'thigh_z']
```

```
# Δημιουργούμε ένα γράφημα για κάθε γνώρισμα
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
axes = axes.flatten()

for i, feature in enumerate(features):
    # Πραγματοποιούμε πέρασμα σε όλα τα αρχεία CSV στο φάκελο
    for filename in os.listdir(path):
        if filename.endswith('.csv'):
            # Χωρίς κατάληξη
            base_name = os.path.splitext(filename)[0]
            file_names.append(base_name)
            df = pd.read_csv(os.path.join(path, filename))

            # Υπολογίζουμε τη μέση τιμή του γνωρίσματος για το συγκεκριμένο
συμμετέχοντα
            mean_value = np.mean(df[feature])

            # Προσθέτουμε τη μέση τιμή στο λεξικό
            if base_name not in mean_values_per_file:
                mean_values_per_file[base_name] = {}
            mean_values_per_file[base_name][feature] = mean_value

    # Ταξινομούμε τα ονόματα των συμμετεχόντων για να διατηρήσουμε την ίδια σειρά στο
γράφημα
    sorted_file_names = sorted(mean_values_per_file.keys())

    x_values = range(len(sorted_file_names))
    y_values = [mean_values_per_file[participant][feature] for participant in
sorted_file_names]

    axes[i].plot(x_values, y_values, '-o', color='blue')

    # Υπολογίζουμε τη συνολική μέση τιμή του γνωρίσματος από όλα τα αρχεία
    total_mean_value = np.mean([mean_values_per_file[participant][feature] for
participant in sorted_file_names])

    # Δημιουργούμε την οριζόντια γραμμή για την συνολική μέση τιμή
    axes[i].axhline(y=total_mean_value, color='orange', linestyle='--', label='Overall
Mean')

    # Προσθήκη τίτλου και ετικέτας
    axes[i].set_title(f'Mean Value of Feature {feature}')
    axes[i].set_xlabel('Participants')
    axes[i].set_ylabel('Mean Value')

    # Χρησιμοποιούμε τις ονομασίες των συμμετεχόντων ως ticks στον άξονα x
    axes[i].set_xticks(range(len(sorted_file_names)))
    axes[i].set_xticklabels(sorted_file_names, rotation=45)

    # Προσθέτουμε το κάθετο διαγώνιο και εμφανίζουμε το γράφημα
    axes[i].grid(True)
plt.tight_layout()
plt.show()
```

2.1.2 Στατιστική Επεξεργασία ανά Δραστηριότητα

Αρχικά, ορίζονται οι λίστες με τα χαρακτηριστικά (**features**) που περιλαμβάνουν τις μετρήσεις επιτάχυνσης από τους αισθητήρες που είναι τοποθετημένοι στην πλάτη και στον μηρό, καθώς και τα **labels** που αντιπροσωπεύουν τις διάφορες δραστηριότητες. Στη συνέχεια, δημιουργείται ένα λεξικό **dfs_per_label** όπου κάθε **label** (δραστηριότητα) αντιστοιχεί σε ένα κενό DataFrame με τις στήλες των χαρακτηριστικών. Ο κώδικας στη συνέχεια διατρέχει όλα τα αρχεία CSV και για κάθε αρχείο CSV, διαβάζει τα δεδομένα και φιλτράρει τις εγγραφές για κάθε **label** (δραστηριότητα). Αν υπάρχουν δεδομένα για μια συγκεκριμένη δραστηριότητα, αυτά προστίθενται στο αντίστοιχο

DataFrame του λεξικού μέσω της **pd.concat( )**. Τέλος, για κάθε δραστηριότητα, τυπώνονται τα στατιστικά περιγραφικά των δεδομένων μέσω της **describe( )**, όπως η μέση τιμή, η τυπική απόκλιση, το ελάχιστο και το μέγιστο.

```
# Λίστα με τα χαρακτηριστικά
features = ['back_x', 'back_y', 'back_z', 'thigh_x', 'thigh_y', 'thigh_z']

# Λίστα με τα labels
labels = [1, 2, 3, 4, 5, 6, 7, 8, 13, 14, 130, 140]

# Δημιουργία λεξικού για τα DataFrames
dfs_per_label = {label: pd.DataFrame(columns=features) for label in labels}

# Πραγματοποιούμε πέρασμα σε όλα τα αρχεία CSV στο φάκελο
for filename in os.listdir(path):
    if filename.endswith('.csv'):
        df = pd.read_csv(os.path.join(path, filename))

        for label in labels:
            label_df = df[df['label'] == label]

            # Αν υπάρχουν δεδομένα για το συγκεκριμένο label, τα προσθέτουμε στο
αντίστοιχο DataFrame
            if not label_df.empty:
                dfs_per_label[label] = pd.concat([dfs_per_label[label],
label_df[features]], ignore_index=True)

# Τυπώνουμε τα στατιστικά για κάθε label
for label, df in dfs_per_label.items():
    print(f"Statistics for Label {label}:")
    print(df.describe().T)
    print("\n")
```

Αρχικά, διατρέχουμε όλα τα ζεύγη **label** και DataFrame από το λεξικό **dfs_per_label**. Για κάθε δραστηριότητα υπολογίζονται οι μέσες τιμές για κάθε χαρακτηριστικό (**mean_values**) και η συνολική μέση τιμή όλων των χαρακτηριστικών (**overall_mean**) μέσω της συνάρτησης **mean( ).** Έπειτα, δημιουργείται ένα γράφημα γραμμών (**lineplot**) χρησιμοποιώντας το **seaborn** (**sns**), όπου στον άξονα x τοποθετούνται τα χαρακτηριστικά και στον άξονα y οι μέσες τιμές των χαρακτηριστικών. Τέλος, προστίθεται μια οριζόντια διακεκομμένη γραμμή στο επίπεδο της συνολικής μέσης τιμής (**overall_mean**) για να οπτικοποιήσουμε την απόκλιση κάθε μέσης τιμής κάθε δραστηριότητας από τη συνολική μέση τιμή.

```
# Για κάθε label δημιουργούμε τα γραφήματα
fig, axes = plt.subplots(4, 3, figsize=(20, 15))
axes = axes.flatten()

for i, (label, df) in enumerate(dfs_per_label.items()):
    # Υπολογίζουμε τις μέσες τιμές για κάθε γνώρισμα
    mean_values = df.mean()

    # Υπολογίζουμε τη συνολική μέση τιμή
    overall_mean = mean_values.mean()

    # Δημιουργία γραφήματος
    sns.lineplot(x=features, y=mean_values, color='blue', marker='o', ax=axes[i])
    axes[i].axhline(y=overall_mean, color='orange', linestyle='--', label='Overall
Mean')

    # Προσθήκη τίτλου και ετικετών
    axes[i].set_title(f'Mean Values of Features for Label {label}')
    axes[i].set_xlabel('Features')
    axes[i].set_ylabel('Mean Value')
    axes[i].grid(True)
plt.tight_layout()
plt.show()
```

2.2 Αποτελέσματα

2.2.0 Συνολική Στατιστική Επεξεργασία

Παρακάτω φαίνονται τα συνολικά στατιστικά μεγέθη για κάθε ένα από τα 6 γνωρίσματα που μελετάμε:

```
            count      mean       std       min       25%       50%  \
back_x   6461328.0 -0.884957  0.377592 -8.000000 -1.002393 -0.974900
back_y   6461328.0 -0.013261  0.231171 -4.307617 -0.083129  0.002594
back_z   6461328.0 -0.169378  0.364738 -6.574463 -0.372070 -0.137451
thigh_x  6461328.0 -0.594888  0.626347 -8.000000 -0.974211 -0.421731
thigh_y  6461328.0  0.020877  0.388451 -7.997314 -0.100087  0.032629
thigh_z  6461328.0  0.374916  0.736098 -8.000000 -0.155714  0.700439

              75%       max
back_x  -0.812303  2.291708
back_y   0.072510  6.491943
back_z   0.046473  4.909483
thigh_x -0.167876  7.999756
thigh_y  0.154951  7.999756
thigh_z  0.948675  8.406235
```

Συνολικά Συμπεράσματα:

- Όλα τα χαρακτηριστικά έχουν τον ίδιο αριθμό δειγμάτων, δηλαδή 6.461.328. Αυτό δείχνει ότι δεν υπάρχουν ελλείποντα δεδομένα για κανένα από τα χαρακτηριστικά. Οι μέσες τιμές δείχνουν ότι τα δεδομένα είναι σχετικά συγκεντρωμένα γύρω από το μηδέν, με κάποιες μικρές αποκλίσεις. Το χαρακτηριστικό **thigh_z** έχει τη μεγαλύτερη μέση τιμή (0.374916), ενώ το **back_x** έχει την πιο αρνητική μέση τιμή (-0.884957). Η τυπική απόκλιση δείχνει πόσο διασπασμένα είναι τα δεδομένα από τη μέση τιμή. Το χαρακτηριστικό **thigh_z** έχει τη μεγαλύτερη διασπορά (0.736098), ενώ το **back_y** έχει τη μικρότερη (0.231171).

- Ακόμα, τα χαρακτηριστικά **back_x**, **thigh_x**, **thigh_y** και **thigh_z** έχουν ελάχιστες τιμές κοντά στο -8, που υποδηλώνουν ακραίες αρνητικές τιμές. Το **back_x** και το **back_z** έχουν ακραίες θετικές τιμές με μέγιστα 2.291708 και 4.909483 αντίστοιχα, ενώ το **thigh_x**, **thigh_y** και **thigh_z** έχουν μέγιστες τιμές κοντά στο 8.

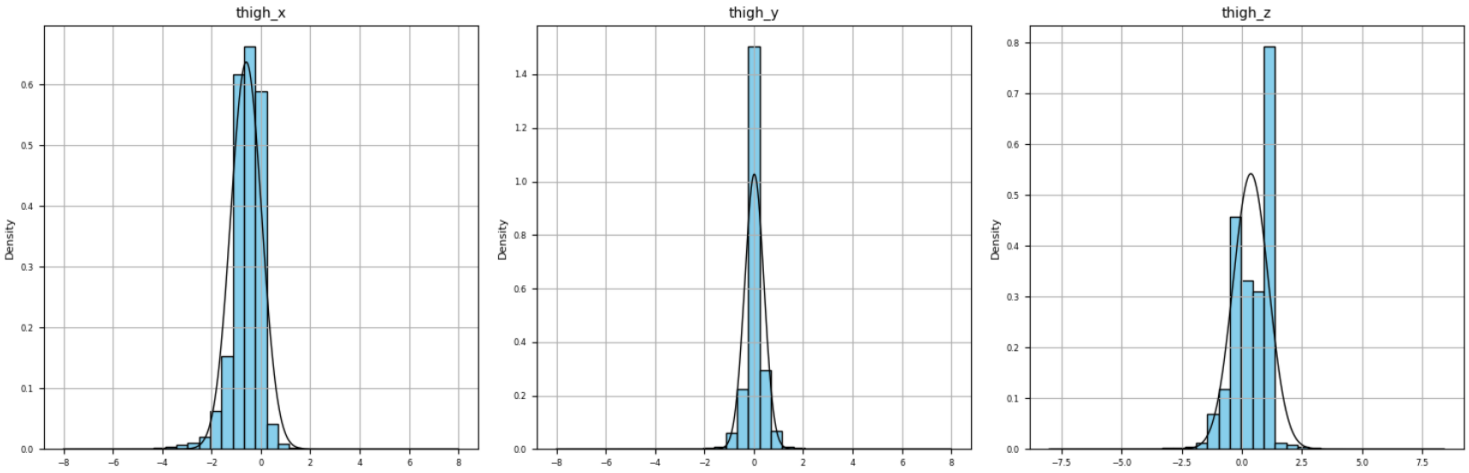
- Οι τιμές του 25% και του 75% για τα περισσότερα χαρακτηριστικά είναι σχετικά κοντά στο μηδέν, δείχνοντας ότι τα δεδομένα είναι σχετικά συγκεντρωμένα γύρω από τη διάμεσο (50%).

Σύνθετα Συμπεράσματα:

- Τα δεδομένα των χαρακτηριστικών **back_x**, **back_z** και **thigh_x** φαίνεται να έχουν πιο αρνητική συμμετρία, καθώς οι μέσες τιμές τους είναι αρνητικές και οι ελάχιστες τιμές τους είναι πολύ χαμηλές. Αντίθετα, τα χαρακτηριστικά **thigh_y** και **thigh_z** δείχνουν μια θετική κλίση, με μέσες τιμές θετικές και ελάχιστες τιμές αρνητικές.
- Το χαρακτηριστικό **thigh_z** έχει τη μεγαλύτερη διασπορά, που δείχνει μεγαλύτερη μεταβλητότητα στις μετρήσεις του. Αντίθετα, το χαρακτηριστικό **back_y** έχει τη μικρότερη διασπορά, υποδεικνύοντας σταθερότητα στις μετρήσεις του.

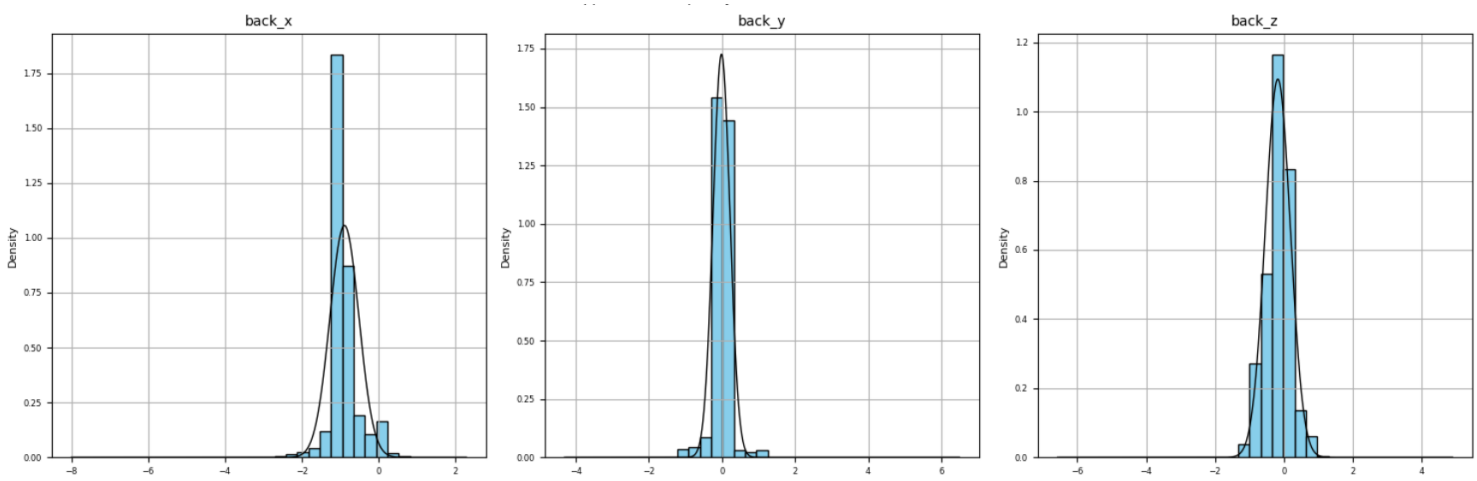
Παρακάτω φαίνονται οι γραφικές με τη πυκνότητα πιθανότητας και τη κατανομή που προσεγγίζει κάθε ένα από τα 6 γνωρίσματα που μελετάμε:

Κατανομή Τιμών Αισθητήρα "thigh":



- Όλες οι κατανομές είναι κεντραρισμένες γύρω από το μηδέν.
- Η κατανομή των τιμών στις διαστάσεις **y** και **z** είναι πιο συγκεντρωμένη γύρω από το μηδέν σε σχέση με την διάσταση **x**.

Κατανομή Τιμών Αισθητήρα "back":

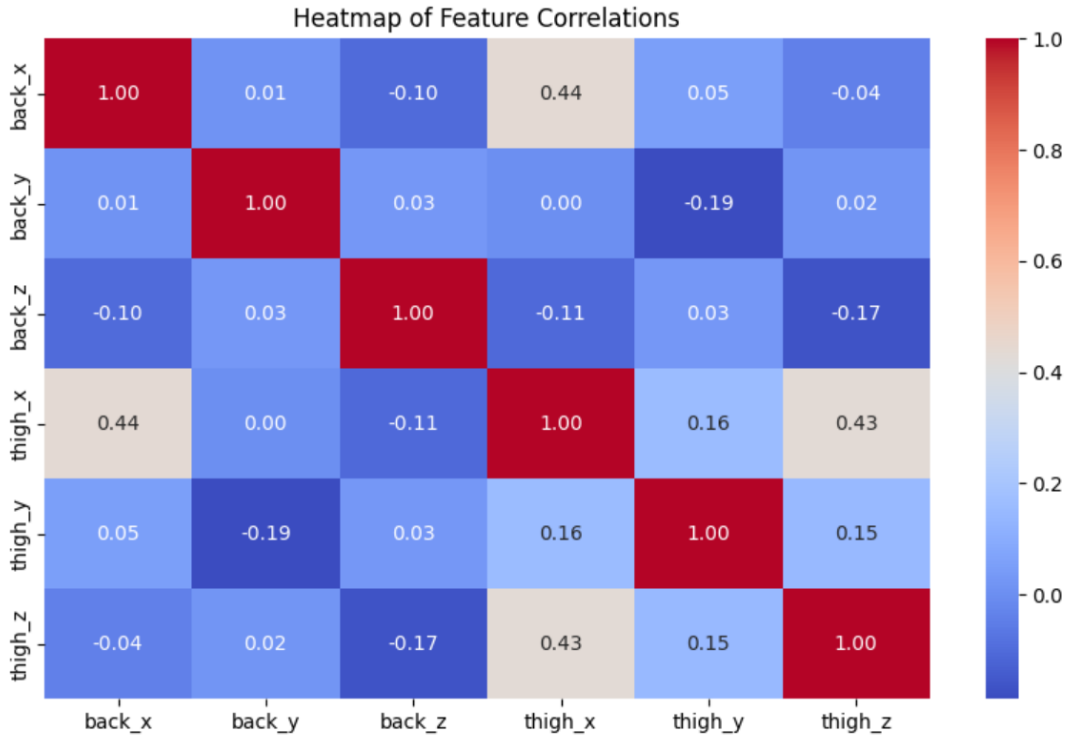


- Οι κατανομές είναι επίσης όλες κεντραρισμένες γύρω από το μηδέν.
- Η κατανομή των τιμών στις διαστάσεις **y** και **z** είναι πιο συγκεντρωμένη γύρω από το μηδέν σε σχέση με την διάσταση **x**.

Συγκριτική Ανάλυση:

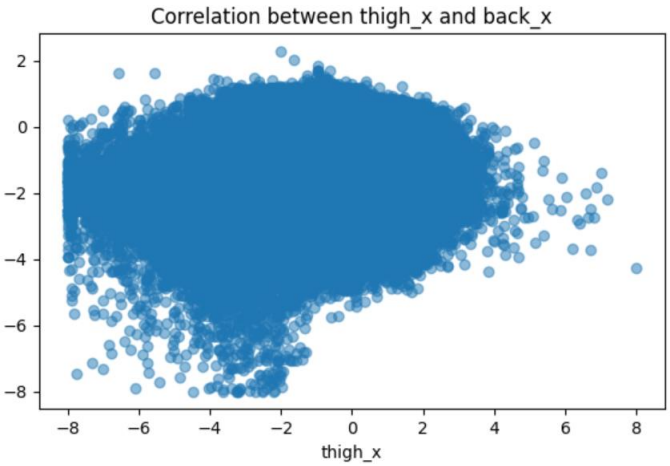
Και για τα δύο μέρη του σώματος οι κατανομές των τιμών στις διαστάσεις **y** και **z** είναι πιο συγκεντρωμένες και στενότερες γύρω από το μηδέν, υποδεικνύοντας μικρότερη διακύμανση (μεγαλύτερη σταθερότητα) σε αυτές τις διαστάσεις. Αντίθετα, οι κατανομές των τιμών στη διάσταση **x** είναι ελαφρώς πιο πλατιές, υποδεικνύοντας μεγαλύτερη διακύμανση στις μετρήσεις. Συνοψίζοντας, οι κατανομές των τιμών και για **thigh** και για **back** είναι κατά προσέγγιση κανονικές και συμμετρικές, με μικρές αποκλίσεις που μπορεί να υπάρχουν λόγω φυσιολογικής διακύμανσης των δεδομένων. Οι κατανομές δεν φαίνονται να έχουν πολλά ακραία σημεία (outliers), γεγονός που υποδηλώνει ότι τα δεδομένα είναι σχετικά ομοιογενή.

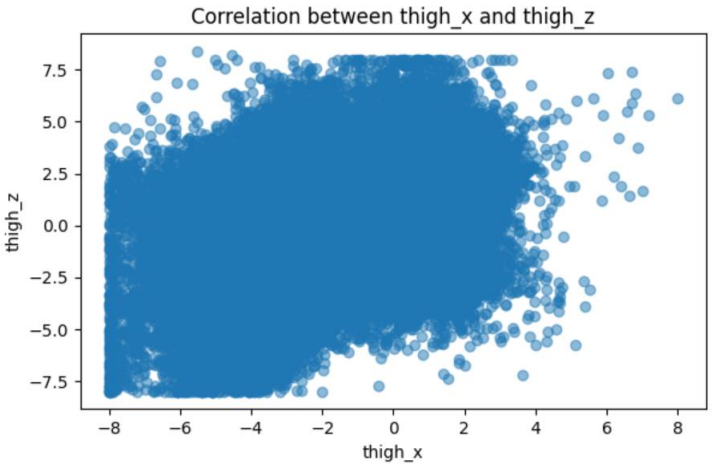
Τέλος, βλέπουμε το θερμοχάρτη και τις γραφικές παραστάσεις συσχέτισης των δεδομένων για κάθε ένα από τα 4 πιο ισχυρά συσχετιζόμενα ζεύγη γνωρισμάτων:

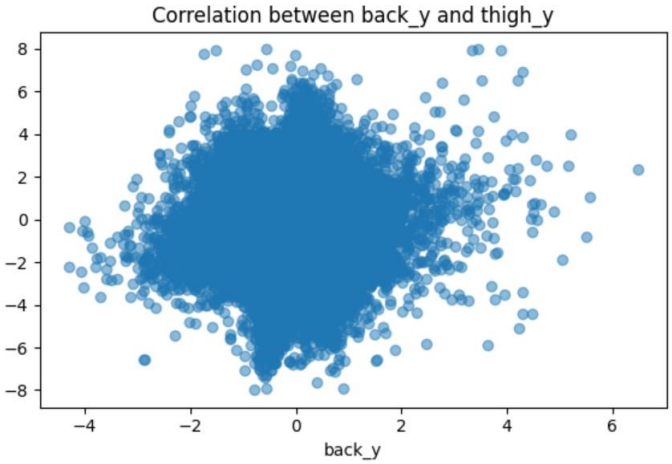


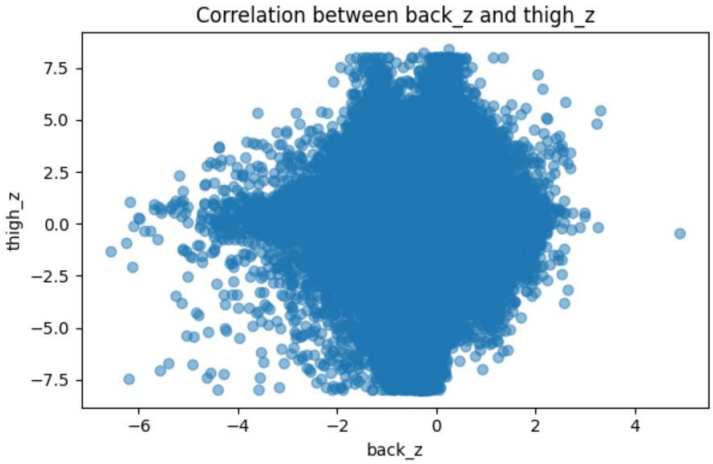
Συμπεράσματα:

- Τα γνωρίσματα **thigh_x** και **back_x** έχουν συντελεστή συσχέτισης 0.44, υποδεικνύοντας μια μέτρια θετική συσχέτιση. Αυτό σημαίνει ότι καθώς αυξάνεται η τιμή του **thigh_x**, τείνει να αυξάνεται και η τιμή του **back_x**.
- Τα γνωρίσματα **thigh_x** και **thigh_z** έχουν συντελεστή συσχέτισης 0.43, υποδεικνύοντας επίσης μια μέτρια θετική συσχέτιση.
- Τα γνωρίσματα **thigh_y** και **thigh_z** έχουν συντελεστή συσχέτισης 0.15, υποδεικνύοντας μια χαμηλή αλλά θετική συσχέτιση.
- Τα γνωρίσματα **back_y** και **thigh_y** έχουν συντελεστή συσχέτισης -0.19, υποδεικνύοντας μια χαμηλή αρνητική συσχέτιση.
- Τα γνωρίσματα **back_z** και **thigh_z** έχουν συντελεστή συσχέτισης -0.17, υποδεικνύοντας μια χαμηλή αρνητική συσχέτιση.
- Οι περισσότερες από τις υπόλοιπες συσχετίσεις είναι κοντά στο μηδέν, υποδηλώνοντας ότι δεν υπάρχει σημαντική γραμμική συσχέτιση μεταξύ των χαρακτηριστικών αυτών.









2.2.1 Στατιστική Επεξεργασία ανά Συμμετέχοντα

Παρακάτω φαίνονται ενδεικτικά τα στατιστικά μεγέθη για 3 συμμετέχοντες που μελετάμε:

```
Statistic Values for Participant S008
             count      mean       std       min       25%       50%       75%  \
back_x   418989.0 -0.920351  0.130877 -3.066853 -0.998823 -0.972054 -0.826290
back_y   418989.0  0.040018  0.107516 -1.209330 -0.013310  0.048302  0.101082
back_z   418989.0 -0.326746  0.297591 -0.960136 -0.580225 -0.297335 -0.143575
thigh_x  418989.0 -0.332798  0.463427 -5.922062 -0.882999 -0.085027 -0.005905
thigh_y  418989.0  0.050361  0.218070 -2.857320 -0.088989  0.068947  0.166770
thigh_z  418989.0  0.656598  0.523197 -4.233158  0.256630  0.950087  0.994429

              max
back_x   0.873471
back_y   1.255642
back_z   1.872940
thigh_x  2.312848
thigh_y  4.809320
thigh_z  5.324356

Statistic Values for Participant S010
             count      mean       std       min       25%       50%       75%  \
back_x   351649.0 -1.019898  0.186093 -2.365137 -1.049960 -1.015238 -0.974607
back_y   351649.0 -0.002224  0.089664 -0.891766 -0.054221 -0.000030  0.047685
back_z   351649.0 -0.014543  0.155128 -1.120693 -0.085287 -0.000642  0.071708
thigh_x  351649.0 -0.905389  0.409359 -4.449651 -1.035450 -0.989746 -0.843884
thigh_y  351649.0  0.017624  0.262037 -3.333658 -0.097479  0.020164  0.115373
thigh_z  351649.0  0.146144  0.542611 -4.198951 -0.114183  0.022649  0.313726

              max
back_x  -0.309204
back_y   0.994556
back_z   1.586732
thigh_x  1.372425
thigh_y  2.662940
thigh_z  5.062087
Statistic Values for Participant S012
             count      mean       std       min       25%       50%       75%  \
back_x   382414.0 -0.915515  0.242906 -3.810360 -1.000070 -0.987624 -0.971043
back_y   382414.0  0.005064  0.137488 -1.506640 -0.035278  0.023321  0.073108
back_z   382414.0 -0.130062  0.290484 -1.343093 -0.151946 -0.058455  0.027089
thigh_x  382414.0 -0.321236  0.350287 -6.547333 -0.451028 -0.165490 -0.103086
thigh_y  382414.0  0.011753  0.204115 -3.652872 -0.121605  0.049881  0.131408
```

```
thigh_z   382414.0   0.809352   0.395776 -2.656861   0.879231   0.978771   0.999394

              max
back_x   0.511867
back_y   1.758769
back_z   2.035369
thigh_x  2.828190
thigh_y  4.131459
thigh_z  3.386593
```

Συμπεράσματα:

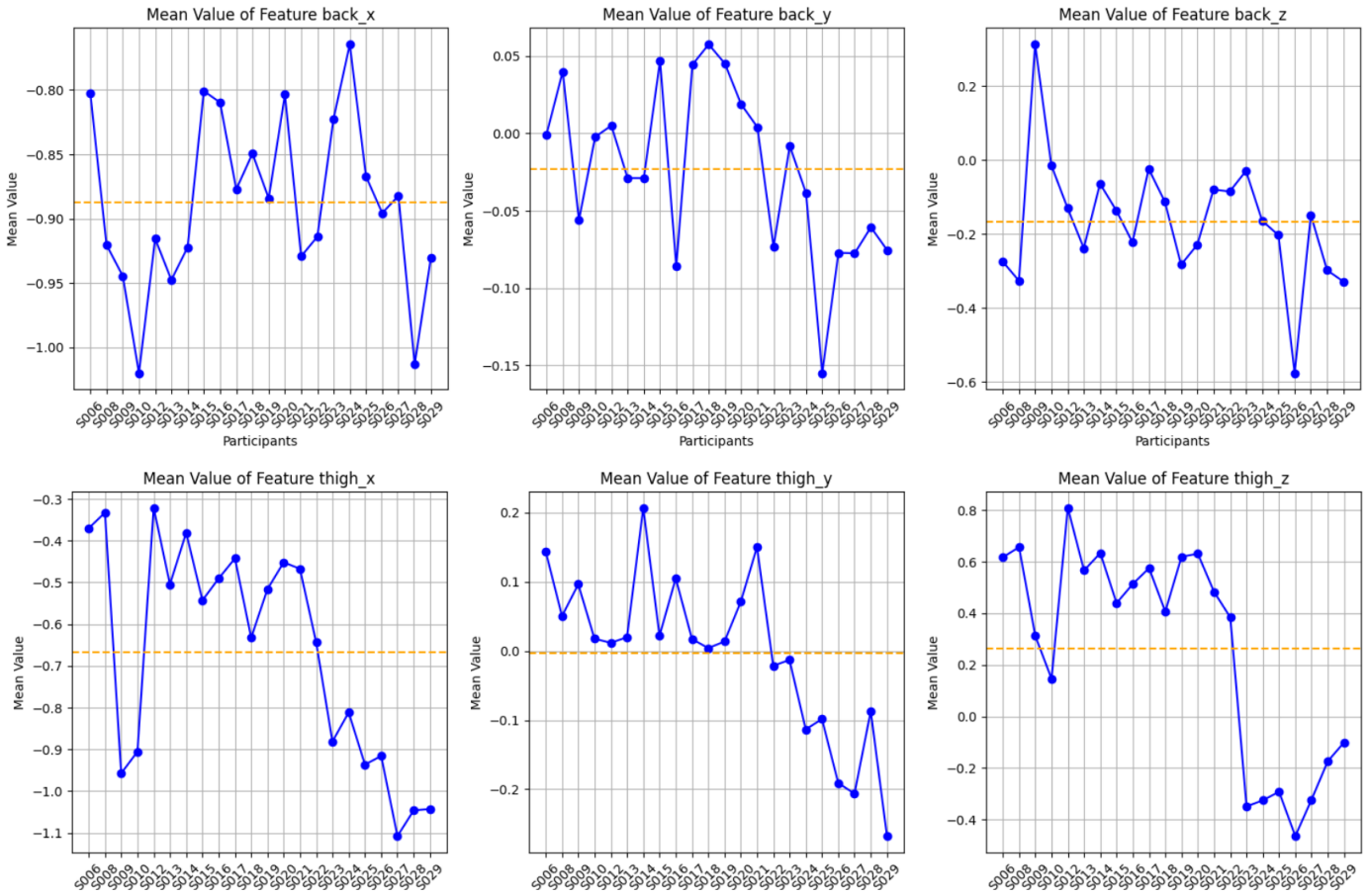
- Για τα περισσότερα χαρακτηριστικά, οι μέσες τιμές των συμμετεχόντων είναι κοντά στις συνολικές μέσες τιμές, αλλά με μικρότερες τυπικές αποκλίσεις, υποδηλώνοντας ότι τα δεδομένα των συμμετεχόντων είναι λιγότερο διασκορπισμένα σε σχέση με τα συνολικά δεδομένα.

- Υπάρχουν μερικές διαφορές στις μέσες τιμές μεταξύ των συμμετεχόντων και των συνολικών δεδομένων, οι οποίες μπορεί να υποδηλώνουν συγκεκριμένες ιδιαιτερότητες στις κινήσεις ή τις στάσεις των συμμετεχόντων.

- Οι τιμές των χαρακτηριστικών **back_x**, **back_y** και **back_z** για τους συμμετέχοντες φαίνεται να έχουν λιγότερες ακραίες τιμές σε σχέση με τα συνολικά δεδομένα.

Με βάση τα παραπάνω, μπορούμε να συμπεράνουμε ότι οι συμμετέχοντες εμφανίζουν αρκετή ομοιογένεια στις μετρήσεις τους, αλλά υπάρχουν και συγκεκριμένες διαφορές που μπορεί να χρειάζονται περαιτέρω διερεύνηση.

Επίσης, βλέπουμε συνολικά τη μεταβολή της μέσης τιμής κάθε γνωρίσματος αναλογικά με κάθε συμμετέχοντα και την απόκλιση της από τη συνολική μέση τιμή που υπολογίσαμε πριν:



Συμπεράσματα:

- **back_x**: Οι μέσες τιμές κυμαίνονται μεταξύ περίπου -1.0 και -0.8. Υπάρχει μια σαφής διακύμανση μεταξύ των συμμετεχόντων, υποδεικνύοντας ότι η κίνηση της πλάτης στον άξονα x ποικίλλει ανάλογα με τον συμμετέχοντα.

- **back_y**: Οι μέσες τιμές κυμαίνονται μεταξύ περίπου -0.15 και 0.05. Παρόμοια με το **back_x**, υπάρχει διακύμανση μεταξύ των συμμετεχόντων, αν και η διακύμανση φαίνεται να είναι μικρότερη από αυτή του back_x.

- **back_z**: Οι μέσες τιμές κυμαίνονται μεταξύ περίπου -0.6 και 0.2. Υπάρχουν αρκετές κορυφές και κοιλάδες, δείχνοντας ότι η κίνηση της πλάτης στον άξονα z είναι αρκετά μεταβλητή μεταξύ των συμμετεχόντων.

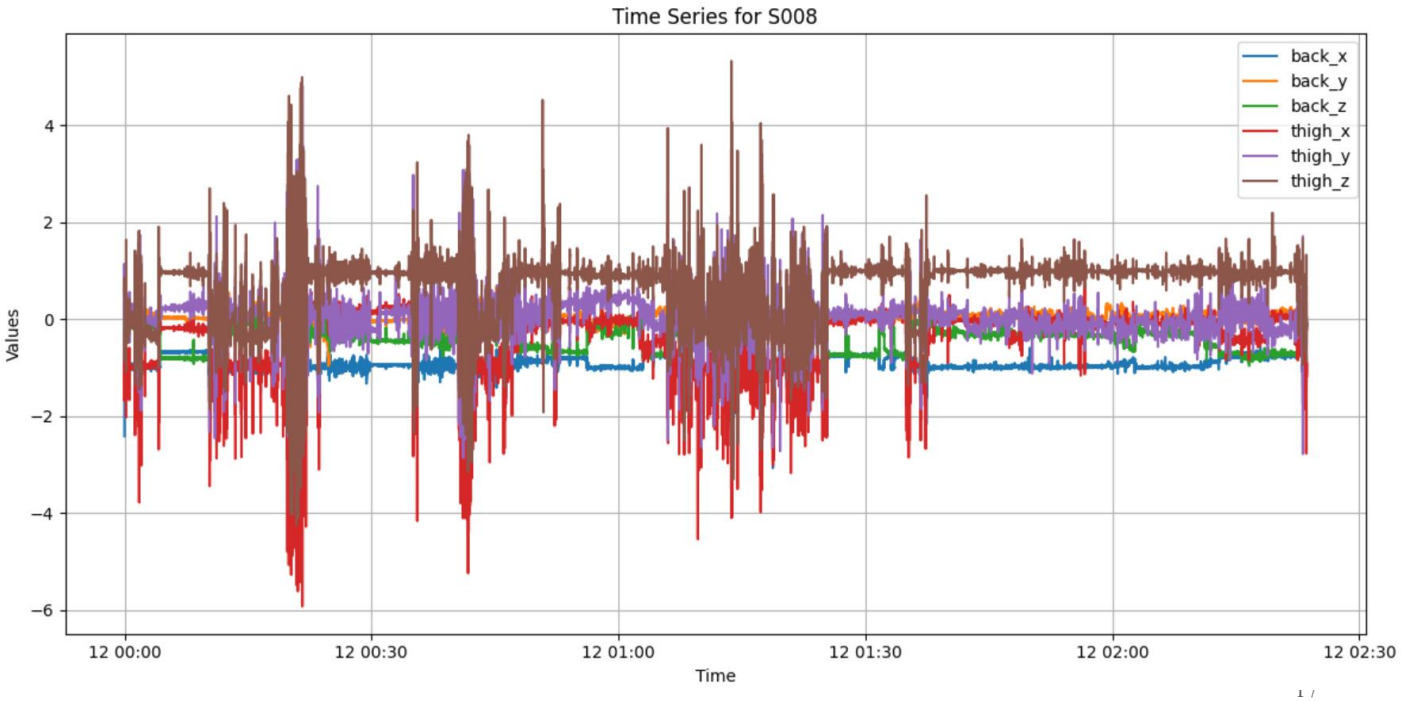
- **thigh_x**: Οι μέσες τιμές κυμαίνονται μεταξύ περίπου -1.1 και -0.3. Υπάρχει μια σαφής πτωτική τάση για ορισμένους συμμετέχοντες, δείχνοντας ότι η κίνηση του μηρού στον άξονα x ποικίλλει σημαντικά.

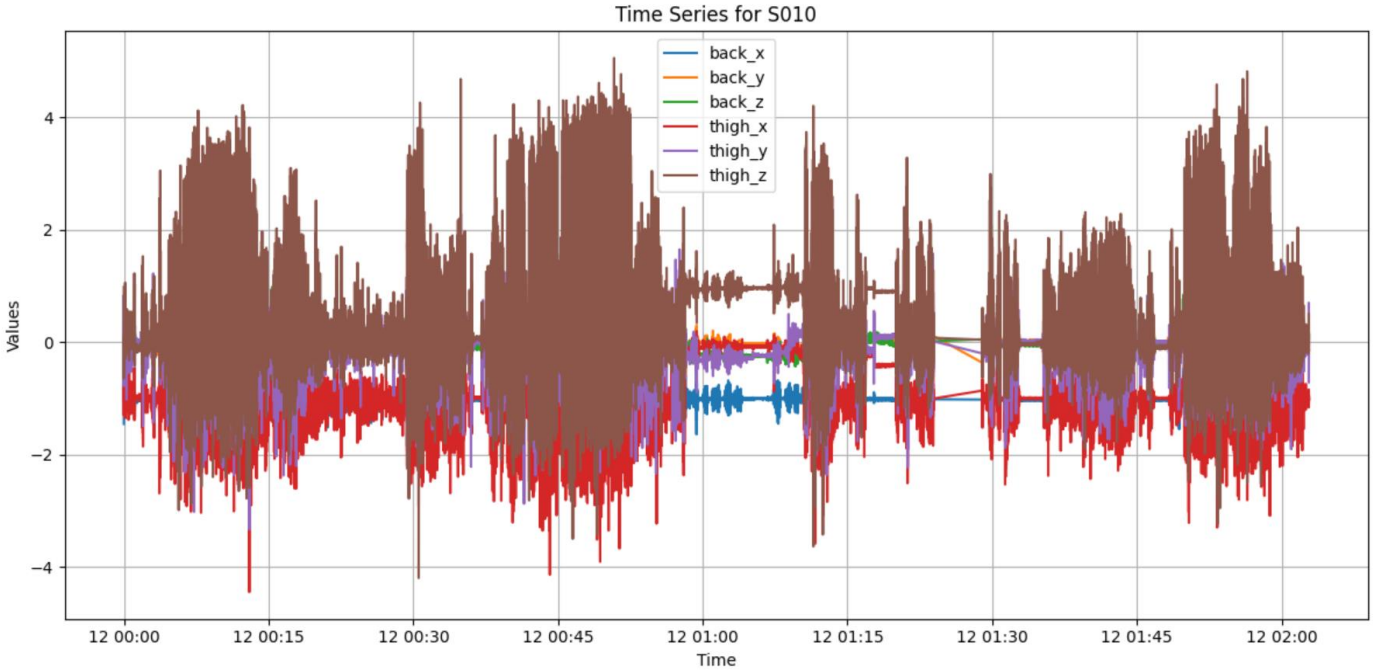
- **thigh_y**: Οι μέσες τιμές κυμαίνονται μεταξύ περίπου -0.2 και 0.2. Η διακύμανση μεταξύ των συμμετεχόντων είναι εμφανής, με αρκετές κορυφές και κοιλάδες, υποδεικνύοντας ότι η κίνηση του μηρού στον άξονα y είναι αρκετά μεταβλητή.

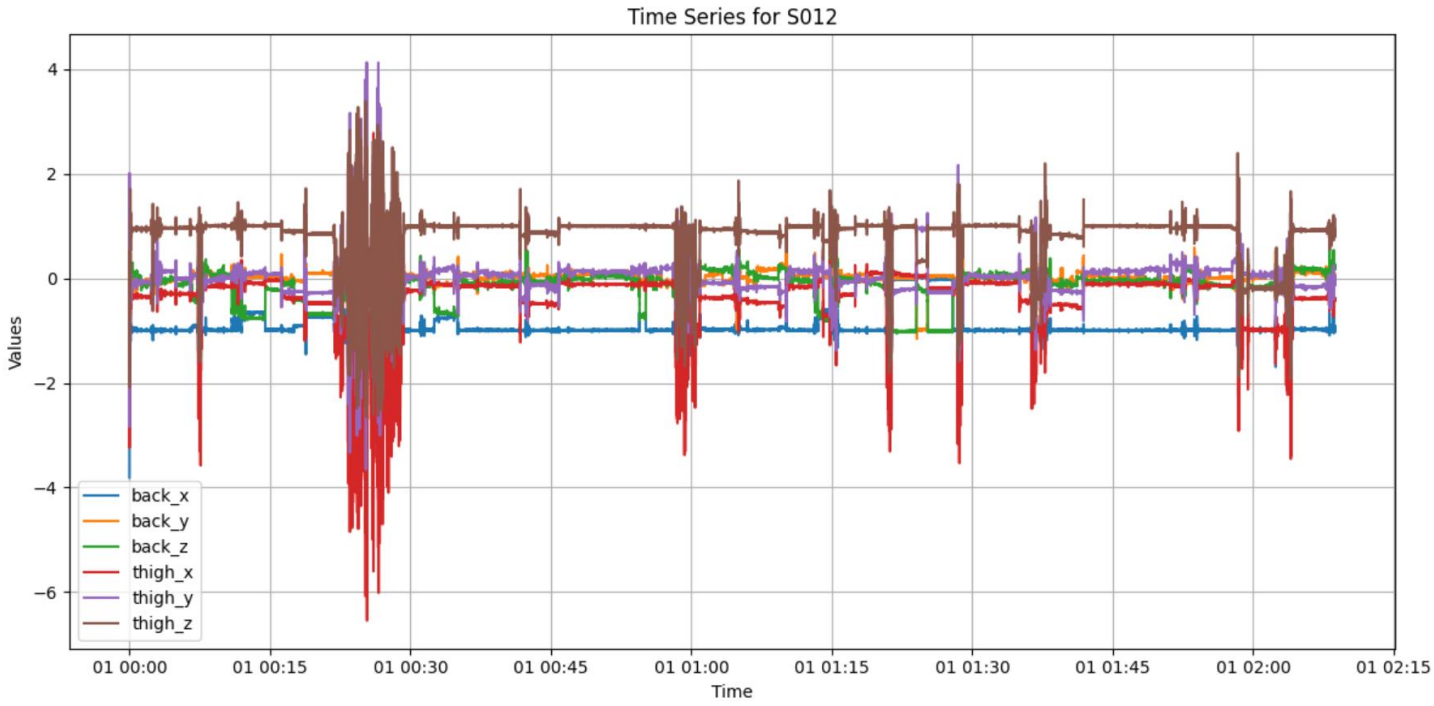
- **thigh_z**: Οι μέσες τιμές κυμαίνονται μεταξύ περίπου -0.4 και 0.8. Το χαρακτηριστικό αυτό δείχνει τη μεγαλύτερη διακύμανση μεταξύ των συμμετεχόντων, υποδεικνύοντας ότι η κίνηση του μηρού στον άξονα z ποικίλλει σημαντικά.

Γενικά, παρατηρούμε ότι υπάρχει σημαντική μεταβλητότητα στις μέσες τιμές των χαρακτηριστικών μεταξύ των συμμετεχόντων. Αυτό υποδηλώνει ότι οι κινήσεις των συμμετεχόντων ποικίλλουν σημαντικά. Ορισμένα χαρακτηριστικά, όπως το **thigh_z**, δείχνουν μεγαλύτερη μεταβλητότητα και μπορεί να είναι πιο κρίσιμα για την ανάλυση ή την ταξινόμηση. Αντίθετα, χαρακτηριστικά όπως το **back_y** δείχνουν λιγότερη μεταβλητότητα και μπορεί να είναι λιγότερο σημαντικά. Τέλος, ορισμένοι συμμετέχοντες δείχνουν συγκεκριμένες τάσεις στα δεδομένα τους, όπως η πτωτική τάση στο **thigh_x**. Αυτές οι τάσεις μπορεί να υποδηλώνουν συγκεκριμένες συμπεριφορές ή μοτίβα κίνησης που αξίζουν περαιτέρω διερεύνηση.

Τέλος, βλέπουμε ενδεικτικά πάλι τις χρονοσειρές για τους ίδιους 3 συμμετέχοντες και τη μεταβολή κάθε γνωρίσματος τους σε σχέση με το χρόνο μετρήσεων:







Συμπεράσματα:

- **S010**: Παρατηρείται μεγάλη ποικιλία στις τιμές των δεδομένων, ειδικά στον άξονα z του χαρακτηριστικού **thigh**. Υπάρχουν συχνές και απότομες αλλαγές στις τιμές, με πολλαπλές κορυφές και κοιλάδες, που πιθανόν να υποδηλώνουν έντονες κινήσεις.

- **S012**: Υπάρχει ομαλότερη μεταβολή σε σύγκριση με τον συμμετέχοντα **S010**. Οι τιμές παραμένουν πιο σταθερές, λιγότερο έντονες και φαίνεται να είναι πιο συγκεντρωμένες γύρω από ένα συγκεκριμένο εύρος, χωρίς πολλές απότομες μεταβολές.

- **S008**: Η δραστηριότητα είναι κάπου μεταξύ των **S010** και **S012**, με αρκετές μεταβολές αλλά όχι τόσο έντονες όσο στον **S010**. Υπάρχουν επαναλαμβανόμενα μοτίβα, που υποδηλώνουν ότι ο συμμετέχων έχει συγκεκριμένο ρυθμό κίνησης.

Γενικά, κάθε συμμετέχων παρουσιάζει διαφορετικό μοτίβο κίνησης, όπως φαίνεται και από τις γραφικές παραστάσεις.

2.2.2 Στατιστική Επεξεργασία ανά Δραστηριότητα

Παρακάτω φαίνονται ενδεικτικά τα στατιστικά μεγέθη για τις πρώτες 3 δραστηριότητες που μελετάμε:

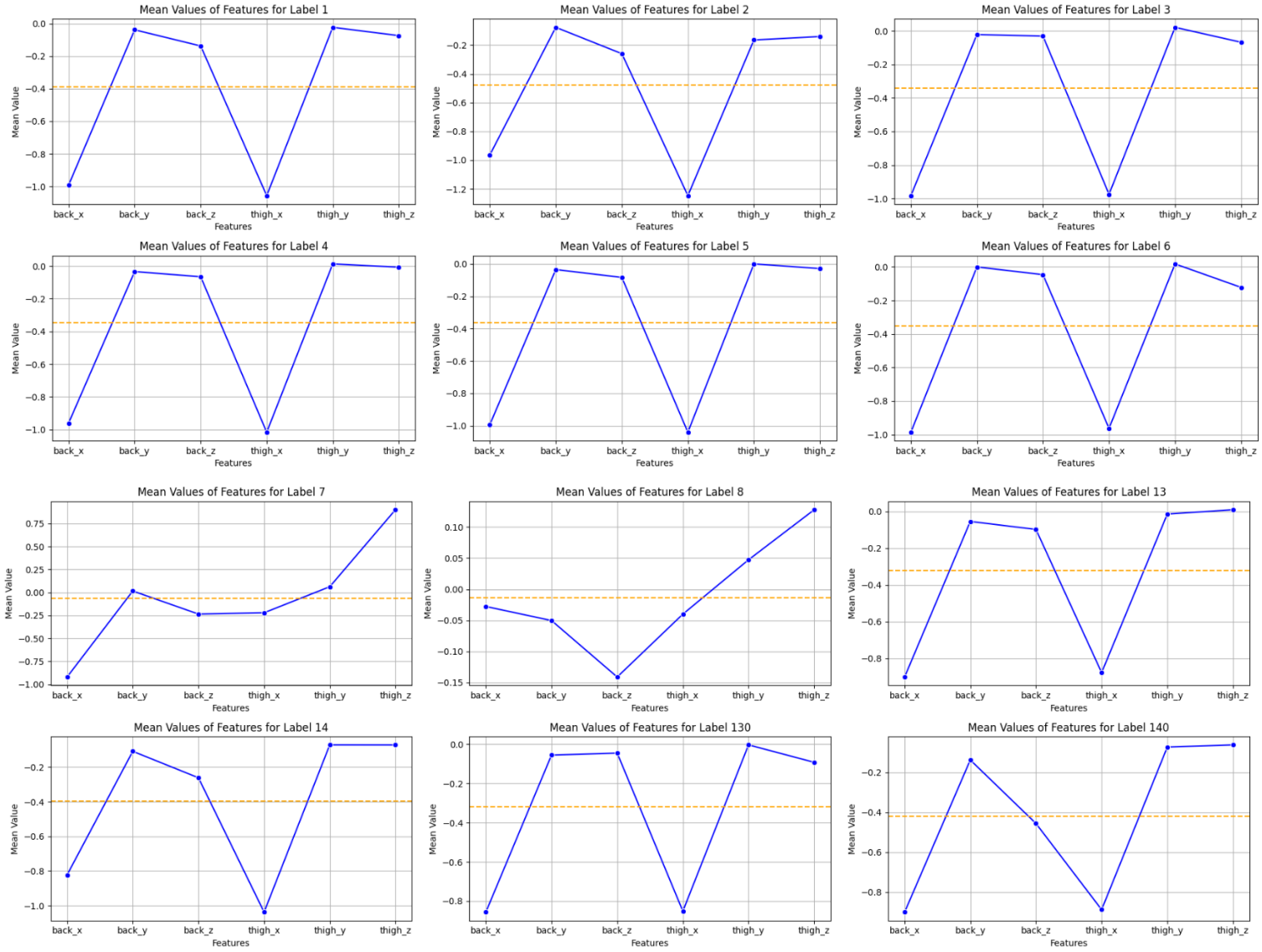
```
Statistics for Label 1:
             count      mean       std       min       25%       50%  \
back_x   1197155.0 -0.992566  0.311378 -7.974365 -1.158848 -0.975721
back_y   1197155.0 -0.038755  0.190476 -3.016498 -0.141057 -0.030273
back_z   1197155.0 -0.137808  0.287737 -3.827393 -0.278809 -0.127185
thigh_x  1197155.0 -1.056683  0.639900 -8.000000 -1.314697 -0.994395
thigh_y  1197155.0 -0.023477  0.536738 -7.997314 -0.200704 -0.000487
thigh_z  1197155.0 -0.074345  0.721997 -8.000000 -0.372559 -0.095456

              75%       max
back_x  -0.803997  2.291708
back_y   0.070179  3.256592
back_z   0.044074  2.896858
thigh_x -0.808350  4.272705
thigh_y  0.172181  5.979248
thigh_z  0.216064  6.897688


Statistics for Label 2:
             count      mean       std       min       25%       50%       75%  \
back_x   291356.0 -0.965280  1.113858 -8.000000 -1.871094 -0.858643  0.040771
back_y   291356.0 -0.076626  0.407701 -4.307617 -0.278076 -0.070312  0.121338
back_z   291356.0 -0.259829  0.451772 -6.574463 -0.430176 -0.225586 -0.022217
thigh_x  291356.0 -1.246811  1.438550 -8.000000 -2.219727 -1.141688 -0.265869
thigh_y  291356.0 -0.164790  0.898353 -7.929199 -0.680930 -0.142822  0.338391
thigh_z  291356.0 -0.140530  1.381836 -8.000000 -0.871582 -0.169678  0.560547

              max
back_x   1.698069
back_y   6.491943
back_z   3.306308
thigh_x  7.999756
thigh_y  7.999756
thigh_z  8.406235


Statistics for Label 3:
             count      mean       std       min       25%       50%       75%  \
back_x   254839.0 -0.982356  0.106649 -4.255035 -1.015621 -0.989241 -0.954638
back_y   254839.0 -0.022316  0.158300 -1.694547 -0.122845 -0.012600  0.079404
back_z   254839.0 -0.031349  0.208112 -2.093994 -0.159320 -0.027011  0.096443
thigh_x  254839.0 -0.974374  0.192159 -4.853268 -1.009913 -0.976833 -0.935715
thigh_y  254839.0  0.020759  0.226300 -5.268984 -0.085332  0.031040  0.136107
thigh_z  254839.0 -0.068220  0.294548 -5.178711 -0.244184 -0.101230  0.063019

              max
back_x   1.849398
back_y   1.974854
back_z   1.754575
thigh_x  1.664554
thigh_y  3.755050
thigh_z  4.701908
```

Συμπεράσματα:

- Η δραστηριότητα 2 (**running**) δείχνει να έχει μεγαλύτερη διακύμανση στα περισσότερα χαρακτηριστικά, όπως υποδεικνύεται από τις υψηλότερες τιμές τυπικής απόκλισης. Επίσης, έχει χαμηλότερες μέσες τιμές σε πολλά χαρακτηριστικά σε σύγκριση με τις δραστηριότητες 1 (**walking**) και 3 (**shuffling**), υποδεικνύοντας πιθανές διαφορές στην κλίση ή τη στάση του σώματος των συμμετεχόντων που ανήκουν σε αυτήν την κατηγορία.
- Οι μέσες τιμές των χαρακτηριστικών για τις δραστηριότητες 1 και 3 (**walking** και **shuffling**) είναι πιο κοντά στη συνολική μέση τιμή, υποδεικνύοντας ότι οι μετρήσεις τους είναι πιο αντιπροσωπευτικές του συνολικού συνόλου.

Αυτές οι διαφορές είναι αναμενόμενες καθώς κάθε φυσική δραστηριότητα απαιτεί διαφορετικές κινήσεις και στάσεις του σώματος.

Τέλος, βλέπουμε συνολικά τη μεταβολή της μέσης τιμής κάθε δραστηριότητας αναλογικά με κάθε χαρακτηριστικό και την απόκλιση της από τη συνολική μέση τιμή της δραστηριότητας από όλα τα χαρακτηριστικά:



Συμπεράσματα:

- Οι μέσες τιμές των χαρακτηριστικών φαίνεται ότι παρουσιάζουν σταθερότητα σε κάθε δραστηριότητα, δηλαδή οι καμπύλες έχουν σχεδόν παρόμοιο σχήμα. Αυτό δείχνει ότι τα χαρακτηριστικά έχουν σταθερές τάσεις ανάμεσα στις διαφορετικές δραστηριότητες.
- Οι γραφικές παραστάσεις φαίνεται να έχουν ένα επαναλαμβανόμενο και συμμετρικό μοτίβο που επιβεβαιώνει πως τα χαρακτηριστικά **back_x**, **back_y** και **thigh_x**, **thigh_y** έχουν υψηλότερες μέσες τιμές, ενώ τα **back_z** και **thigh_z** έχουν χαμηλότερες μέσες τιμές σε όλες τις δραστηριότητες των συμμετεχόντων. Αυτό μπορεί να υποδηλώνει ότι οι μετρήσεις των αισθητήρων είναι συνδεδεμένες ή ότι επηρεάζονται παρόμοια από τις δραστηριότητες που καταγράφονται.
- Οι γραφικές για τις διαφορετικές δραστηριότητες δεν επικαλύπτονται πολύ, κάτι που υποδηλώνει ότι τα χαρακτηριστικά μπορούν να διαχωρίσουν αποτελεσματικά τις διάφορες δραστηριότητες.

Γενικά, τα γραφήματα μπορούν να βοηθήσουν στην κατανόηση των χαρακτηριστικών που είναι σημαντικά για τον διαχωρισμό των δραστηριοτήτων και πώς αυτά συμπεριφέρονται για κάθε δραστηριότητα.

3 Ερώτημα 2ο

3.0 Περιγραφή

3.0.0 Γενικά

Lag Features

Στην υλοποίηση μας έχουμε επιλέξει να χρησιμοποιήσουμε τα **lag features**, είναι χαρακτηριστικά τα οποία χρησιμοποιούνται στην ανάλυση χρονοσειρών, καθώς προσθέτουν πληροφορία σχετικά με τις παρελθοντικές τιμές και έτσι μας επιτρέπουν να δούμε πως οι τιμές αυτές επηρεάζουν τις μελλοντικές. Επιπλέον στα μοντέλα μηχανικής μάθησης που χρησιμοποιούν χρονοσειρές είναι ζωτικής σημασίας, καθώς χρησιμοποιούνται σαν είσοδος, αυξάνοντας τα δεδομένα που χρησιμοποιούνται για πρόβλεψη και έτσι βοηθούν στην κατασκευή ενός μοντέλου που λαμβάνει υπόψη τη συμπεριφορά του παρελθόντος για την πρόβλεψη μελλοντικών γεγονότων.

Διαχωρισμός Training Set και Testing Set

Για τη διάσπαση του αρχικού συνόλου δεδομένων σε Σύνολο Εκπαίδευσης (**Training Set**) και Σύνολο Ελέγχου (**Testing Set**) μια κοινή αναλογία είναι 80-20 ή 70-30, όπου το 80% ή 70% των δεδομένων χρησιμοποιείται για εκπαίδευση και το υπόλοιπο για έλεγχο. Στη δικιά μας υλοποίηση επιλέγουμε να χρησιμοποιήσουμε την αναλογία 80-20.

3.0.1 Random Forest

Αρχικά, μέσω **Bootstrapping** δημιουργούνται πολλαπλά τυχαία υποσύνολα από το σύνολο εκπαίδευσης (**Training Set**) με αντικατάσταση. Έπειτα, Κάθε υποσύνολο δεδομένων χρησιμοποιείται για την εκπαίδευση ενός δέντρου απόφασης. Στον κάθε κόμβο του δέντρου, επιλέγεται τυχαία ένα υποσύνολο χαρακτηριστικών (features) για να αποφασιστεί η καλύτερη διάσπαση. Αυτό μειώνει τη συσχέτιση μεταξύ των δέντρων και βελτιώνει την γενίκευση του μοντέλου. Έπειτα, για κάθε κόμβο το αντίστοιχο δέντρο αποφασίζει ποιο χαρακτηριστικό και ποια τιμή αυτού του χαρακτηριστικού θα χρησιμοποιήσει για να διαχωρίσει τα δεδομένα. Συνηθισμένα κριτήρια για την επιλογή του διαχωρισμού είναι η μείωση της αβεβαιότητας με βάση το **Gini index** ή την **εντροπία**. Κατά την πρόβλεψη της κλάσης μιας νέας παρατήρησης, κάθε δέντρο στο δάσος δίνει την δική του πρόβλεψη και η τελική απόφαση λαμβάνεται με πλειοψηφική ψηφοφορία. Για προβλήματα παλινδρόμησης, η τελική πρόβλεψη είναι ο μέσος όρος των προβλέψεων όλων των δέντρων.

Σημαντικές παράμετροι που μας απασχολούν στην κατασκευή ενός Random Forest Classifier είναι:

- **n_estimators:** Ο αριθμός των δέντρων στο δάσος. Περισσότερα δέντρα συνήθως βελτιώνουν την απόδοση αλλά αυξάνουν τον υπολογιστικό χρόνο και την πολυπλοκότητα.
- **max_features:** Ο μέγιστος αριθμός χαρακτηριστικών που εξετάζονται για κάθε διαχωρισμό κόμβου. Μικρότερος αριθμός χαρακτηριστικών μειώνει την συσχέτιση μεταξύ των δέντρων αλλά μπορεί να μειώσει και την απόδοση κάθε μεμονωμένου δέντρου.
- **max_depth:** Το μέγιστο βάθος κάθε δέντρου. Περιορίζοντας το βάθος, μπορούμε να αποτρέψουμε την υπερεκπαίδευση (overfitting).
- **random_state:** Τυχαίος seed για αναπαραγωγιμότητα των αποτελεσμάτων.

3.0.2 Artificial Neural Network

Για την υλοποίηση του νευρωνικού δικτύου επιλέξαμε να χρησιμοποιήσουμε το **Artificial Neural Network**. Το artificial neural network περιέχει τεχνητούς νευρώνες που ονομάζονται **units**. Τα **units** είναι τοποθετημένα σε μια σειρά από επίπεδα που όλα μαζί αποτελούν το νευρωνικό δίκτυο.

Αποτελείται από 3 επίπεδα:

- **Είσοδο (input layer):** το στρώμα εισόδου λαμβάνει δεδομένα από τον εξωτερικό κόσμο τα οποία το νευρωνικό δίκτυο πρέπει να αναλύσει ή να μάθει.
- **Κρυφά επίπεδα (hidden layers):** ένα ή περισσότερα επίπεδα νευρώνων τα οποία επεξεργάζονται την είσοδο και την μετατρέπουν σε δεδομένα που είναι σημαντικά για την έξοδο.
- **Έξοδο (outout layer):** παράγει την τελική πρόβλεψη.

Τα **units** συνδέονται μεταξύ τους από το ένα επίπεδο στο άλλο. Κάθε μια από αυτές τις συνδέσεις έχει βάρη που καθορίζουν την επιρροή του ενός **unit** σε άλλο **unit**. Καθώς τα δεδομένα μεταφέρονται από το ένα **unit** στο άλλο, το νευρωνικό δίκτυο μαθαίνει όλο και περισσότερα για τα δεδομένα, τα οποία τελικά καταλήγουν σε μια έξοδο από το επίπεδο εξόδου.

Κωδικοποίηση Δεδομένων

Επιλέγουμε να εφαρμόσουμε κωδικοποίηση **one-hot encoding** στη στήλη labels που είναι και η στήλη εξόδου που θέλουμε να προβλέψει το μοντέλο, πριν αρχίσουμε την εκπαίδευσή του. Η **one-hot encoding** κωδικοποιεί τα δεδομένα σε δυαδικά διανύσματα, για κάθε δραστηριότητα δημιουργείται ένα διάνυσμα που έχει μήκος ίσο με τον αριθμό των δραστηριοτήτων και το διάνυσμα έχει την τιμή 1 στην θέση που αντιστοιχεί στην δραστηριότητα και 0 σε όλες τις άλλες.

Δημιουργία Κρυφών Επιπέδων και Εξόδου

Για την δημιουργία των κρυφών επιπέδων θα χρησιμοποιήσουμε την συνάρτηση ενεργοποίησης **ReLU,** η οποία ελέγχει αν η είσοδος x είναι θετική αν είναι τότε επιστρέφει την ίδια είσοδο, ενώ αν είναι αρνητική επιστρέφει το μηδέν. Για την δημιουργία της εξόδου θα επιλέξουμε την συνάρτηση **softmax** καθώς χρησιμοποιούμε **one hot encoding** και η **softmax** δέχεται σαν είσοδο ένα διάνυσμα εισόδου και επιστρέφει ένα διάνυσμα εξόδου που αντιπροσωπεύει μια κατανομή πιθανοτήτων πάνω στις διάφορες κατηγορίες.

Compiling και Fitting Μοντέλου

Κατά την εκπαίδευση του μοντέλου μας θα επιλέξουμε να χρησιμοποιήσουμε και την συνάρτηση απώλειας **categorical crossentropy**, η οποία μετρά πόσο καλά αποδίδει το μοντέλο μας στο να προβλέπει τις δραστηριότητες για τα δεδομένα εισόδου. Επιλέγουμε την συγκεκριμένη συνάρτηση, καθώς τα δεδομένα μας έχουν κωδικοποιηθεί με **one-hot encoding**, οπότε η συνάρτηση αυτή θα υπολογίζει την απόσταση μεταξύ των προβλεπόμενων πιθανοτήτων και των πραγματικών τιμών και αποσκοπεί στην μείωση αυτής της απόστασης. Ουσιαστικά υπολογίζει τη διαφορά μεταξύ του πραγματικού one-hot encoded διανύσματος και του διανύσματος πιθανοτήτων του μοντέλου, αυτό γίνεται για κάθε δείγμα στο σύνολο των δεδομένων. Επίσης, θα χρησιμοποιήσουμε τον αλγόριθμο βελτιστοποίησης **Adam**, ο οποίος προσφέρει ταχύτητα και αποδοτικότητα σε μεγάλα σύνολα δεδομένων. Επιπλέον για την εκπαίδευση του νευρωνικού δικτύου θα χρησιμοποιήσουμε την μέθοδο **batch training**. Η μέθοδος αυτή ορίζει **batch_size** αυτό υποδηλώνει ότι τα δεδομένα εκπαίδευσης θα χωριστούν σε μικρότερα υποσύνολα **batches** και το καθένα **batch_size** δείγματα. Επιπλέον, θα πρέπει να ορίσουμε τον αριθμό των **epochs** τα οποία σηματοδοτούν πόσες φορές το μοντέλο θα διατρέξει όλο το σετ δεδομένων. Το μοντέλο προσαρμόζει τα βάρη του μετά από κάθε batch, αντί να περιμένει να δει όλα τα δείγματα σε μια εποχή. Τέλος, το **verbose** καθορίζει τα επίπεδα εξόδου που θα εμφανίζονται κατά τη διάρκεια της εκπαίδευσης και εμείς θα το θέσουμε με 0, οπότε δεν θα εμφανίζεται τίποτα.

3.0.3 Bayesian Network

ΣΗΜΕΙΩΣΗ Παρατηρήσαμε ότι η υλοποίηση ενός κανονικού **Bayesian Network** έκανε εξαιρετικά χρονοβόρο το τρέξιμο του κώδικα λόγω των πολλών συσχετίσεων που χρησιμοποιεί το μοντέλο για να κάνει την πρόβλεψη. Έτσι αποφασίσαμε να κάνουμε την παραδοχή ότι η υλοποίησή μας θα βασιστεί στον αλγόριθμο **Naive Bayes**, ο οποίος είναι στην ουσία μια πιο απλοϊκή εκδοχή ενός **Bayesian Network**. Πιο συγκεκριμένα, βασίζεται στην υπόθεση ότι όλες οι χαρακτηριστικές μεταβλητές είναι ανεξάρτητες μεταξύ τους, δεδομένης της κλάσης. Συνεπώς, πρόκειται για ένα πολύ απλό **Bayesian Network** με έναν κόμβο για την κλάση και πολλούς κόμβους για τα χαρακτηριστικά, όπου κάθε χαρακτηριστικό θεωρείται ανεξάρτητο από τα άλλα. Αντίθετα, σε ένα κανονικό **Bayesian Network** οι εξαρτήσεις μεταξύ των μεταβλητών μπορούν να είναι σύνθετες και ρητά ορισμένες από τη δομή του δικτύου.

Για την κατασκευή του **Bayesian Network** επιλέξαμε να χρησιμοποιήσουμε τον κατηγοριοποιητή **Gaussian Naive Bayes** που αποτελεί μια παραλλαγή του αλγορίθμου **Naive Bayes**. Ο συγκεκριμένος κατηγοριοποιητής βασίζεται στο θεώρημα του *Bayes* (η πιθανότητα να συμβεί ένα γεγονός Α δεδομένου ότι προηγουμένως έχει συμβεί ένα άλλο γεγονός Β) και οι προβλέψεις που πραγματοποιεί βασίζονται σε πιθανότητες. Η εκτίμηση των πιθανοτήτων γίνεται χρησιμοποιώντας κανονική κατανομή, άρα αναμένει ότι τα δεδομένα θα ακολουθούν κανονική κατανομή. Επίσης, υποθέτει ότι τα χαρακτηριστικά είναι ανεξάρτητα μεταξύ τους, παρόλο που μπορεί να μην ισχύει. Η υπόθεση αυτή παράγει ένα ικανοποιητικό αποτέλεσμα στην πλειονότητα των περιπτώσεων.

3.1 Υλοποίηση

3.1.0 Γενικά

Lag Features

Αρχικά, δημιουργήθηκε η συνάρτηση **create_lags( ),** η οποία θα χρησιμοποιηθεί και για τις 3 υλοποιήσεις (Neural Network, Bayesian Network, Random Forest). Δημιουργούμε μια λίστα **cols_to_shift** στην οποία αποθηκεύουμε τα ονόματα των στηλών για τις οποίες θέλουμε να δημιουργήσουμε τα lag, δηλαδή όλες εκτός από την στήλη **timestamp** και **label**. Στην συνέχεια δημιουργούμε ένα λεξικό **lagged_data** για κάθε στήλη που χρησιμοποιούμε, το οποίο περιέχει ως κλειδιά τα ονόματα των νέων στηλών που θα δημιουργηθούν, και σαν τιμές έχει τα δεδομένα των στηλών μετατοπισμένα κατά lag χρονικά βήματα. Από το παραπάνω λεξικό δημιουργούμε ένα dataframe **lagged_df.** Επιπλέον, με την χρήση της **pd.concat** ενώνουμε το αρχικό dataframe με το **lagged_df** στο **df.** Η ένωση πραγματοποιείται κατά μήκος των στηλών. Επίσης, η μετατόπιση των δεδομένω στις πρώτες γραμμές του dataframe θα δημιουργεί NaN τιμές, καθώς δεν υπάρχουν τιμές για ναμετατοπιστούν. Οπότε αντικαθιστούμε τις NaN τιμές με 0 διασφαλίζοντας έτσι ότι το dataframe δεν θα περιέχει κενά δεδομένα. Η συνάρτηση επιστρέφει το τελικό dataframe **df.**

```
import os
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

def create_lags(df, lag):
    # Επιλογή των στηλών που θα καθυστερήσουν (αφαιρούμε τις άχρηστες στήλες
'timestamp' και 'label')
    cols_to_shift = [col for col in df.columns if (col != 'timestamp' and col !=
'label')]
    # Δημιουργία ενός λεξικού που περιέχει τις καθυστερημένες στήλες
    lagged_data = {f'{col}_lag_{i}': df[col].shift(i) for col in cols_to_shift for i in
range(1, lag + 1)}
    # Δημιουργία ενός νέου DataFrame από το λεξικό με τα καθυστερημένα δεδομένα
    lagged_df = pd.DataFrame(lagged_data)
    # Συνένωση του αρχικού DataFrame με το DataFrame που περιέχει τα καθυστερημένα
δεδομένα
    df = pd.concat([df, lagged_df], axis=1)
    # Αντικατάσταση των τιμών NaN που δημιουργήθηκαν από τις καθυστερήσεις με 0
    df.fillna(0, inplace=True)
    return df

# Μονοπάτι για τον φάκελο που περιέχει τα αρχεία CSV
path = '/content/drive/MyDrive/harth'
```

3.1.1 Random Forest

Δημιουργούμε 4 λίστες, οι οποίες θα χρησιμοποιηθούν για να αποθηκεύουν τις μετρικές απόδοσης. **rf_accuracies = [ ], rf_precisions = [ ], rf_recalls = [ ], rf_f1_scores = [ ].** Χρησιμοποιώντας την συνάρτηση **RndomForestClassifier( )**, δημιουργούμε έναν ταξινομητή random forest με 30 δέντρα και μια σταθερή τιμή για το **random_state.** Στη συνέχεια, διατρέχουμε όλα τα αρχεία που περιέχει ο φάκελος harth, κάθε αρχείο το αποθηκεύουμε σε ένα dataframe **df** και ύστερα πάνω σε αυτό καλούμε την συνάρτηση **create_lags(df,50),** οπότε για κάθε χρονική στιγμή θα κοιτάμε και τις 50 προηγούμενες. Στην συνέχεια, θα χωρίσουμε το dataframe σε **train** και **test**, το **train** θα αποτελείται από το 80% των δεδομένων και θα είναι τα αρχικά δεδομένα του **df** και το **test** θα αποτελείται από το υπόλοιπο 20%. Επιπλέον, θα διαχωρίσουμε τις στήλες που θα χρησιμοποιούμε (x: ανεξάρτητες μεταβλητές, y εξαρτημένες μεταβλητές). Τα χαρακτηριστικά **x_train, x_test** θα περιλαμβάνουν όλες τις στήλες εκτός από την **timestamp** και την **label** και τα

χαρακτηριστικά **y_train, y_test** θα αποτελούνται μόνο από την στήλη **label**. Ύστερα, καλώντας την συνάρτηση **fit()** πάνω στον ταξινομητή που δημιουργήσαμε θα τον εκπαιδεύσουμε χρησιμοποιώντας τα δεδομένα εκπαίδευσης (**X_test**, **Y_test**). Επίσης, με την **predict( )** θα πραγματοποιήσουμε τις προβλέψεις πάνω στα δεδομένα δοκιμής και θα υπολογίσουμε και τις μετρικές. Αποθηκεύουμε, εκτυπώνουμε τις μετρικές και υπολογίζουμε confusion matrix με την συνάρτηση **confusion_matrix( )** και το εμφανίζουμε ως heatmap. Τέλος, υπολογίζουμε και εκτυπώνουμε τις μέσες τιμές για όλες τις μετρικές για όλους τους συμμετέχοντες.

```
from sklearn.ensemble import RandomForestClassifier

# Λίστες για την αποθήκευση των μετρικών από κάθε αρχείο
rf_accuracies = []  # Λίστα για την αποθήκευση των ακριβειών από κάθε αρχείο
rf_precisions = []  # Λίστα για την αποθήκευση των precisions από κάθε αρχείο
rf_recalls = []     # Λίστα για την αποθήκευση των recalls από κάθε αρχείο
rf_f1_scores = []   # Λίστα για την αποθήκευση των f1-scores από κάθε αρχείο

# Δημιουργία και εκπαίδευση του ταξινομητή Random Forest με 30 δέντρα και καθορισμένο
τυχαίο seed για αναπαραγωγιμότητα
rf_classifier = RandomForestClassifier(n_estimators=30, random_state=42)

# Διατρέχουμε τα αρχεία στον φάκελο
for filename in os.listdir(path):
    if filename.endswith(".csv"):
        # Φορτώνουμε το CSV αρχείο
        df = pd.read_csv(os.path.join(path, filename))

        # Δημιουργούμε τα lags (καθυστέρηση) για τη δημιουργία χαρακτηριστικών από
προηγούμενες τιμές
        df = create_lags(df, 50)

        # Διαχωρίζουμε τα δεδομένα σε train και test sets (80% για εκπαίδευση και 20%
για έλεγχο)
        train_size = int(0.8 * len(df))
        train_df = df.iloc[:train_size]
        test_df = df.iloc[train_size:]

        # Διαχωρισμός features (X) και labels (y) για το train και test set
        X_train = train_df.drop(['timestamp', 'label'], axis=1)
        y_train = train_df['label']
        X_test = test_df.drop(['timestamp', 'label'], axis=1)
        y_test = test_df['label']

        # Εκπαίδευση του ταξινομητή Random Forest με τα δεδομένα του συνόλου
εκπαίδευσης
        rf_classifier.fit(X_train, y_train)

        # Προβλέπουμε τις ετικέτες του test set
        y_pred = rf_classifier.predict(X_test)

        # Υπολογισμός των μετρικών απόδοσης του μοντέλου
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, average='weighted')
        recall = recall_score(y_test, y_pred, average='weighted')
        f1 = f1_score(y_test, y_pred, average='weighted')

        # Προσθήκη των μετρικών στις λίστες
        rf_accuracies.append(accuracy)
        rf_precisions.append(precision)
        rf_recalls.append(recall)
        rf_f1_scores.append(f1)

        # Εκτύπωση των μετρικών για κάθε συμμετέχοντα
        base_name = os.path.splitext(filename)[0]  # Αφαίρεση της κατάληξης από το
```

```
όνομα του αρχείου
        print(f"Metrics for Participant {base_name}:")
        print(f"  Accuracy: {accuracy}")
        print(f"  Precision: {precision}")
        print(f"  Recall: {recall}")
        print(f"  F1-Score: {f1}")

        # Υπολογισμός και εμφάνιση του confusion matrix
        unique_labels = sorted(y_test.unique())  # Ταξινόμηση των μοναδικών ετικετών
        cm = confusion_matrix(y_test, y_pred, labels=unique_labels)
        plt.figure(figsize=(10, 7))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=unique_labels,
yticklabels=unique_labels)
        plt.title(f'Confusion Matrix for Participant {base_name}')
        plt.xlabel('Predicted Labels')
        plt.ylabel('True Labels')
        plt.show()

# Υπολογισμός της μέσης τιμής για κάθε μετρική από όλους τους συμμετέχοντες
rf_mean_accuracy = sum(rf_accuracies) / len(rf_accuracies)
rf_mean_precision = sum(rf_precisions) / len(rf_precisions)
rf_mean_recall = sum(rf_recalls) / len(rf_recalls)
rf_mean_f1 = sum(rf_f1_scores) / len(rf_f1_scores)

# Εκτύπωση των μέσων τιμών των μετρικών για όλους τους συμμετέχοντες
print(f"Mean Metrics of All Participants:")
print(f"  Mean Accuracy: {rf_mean_accuracy}")
print(f"  Mean Precision: {rf_mean_precision}")
print(f"  Mean Recall: {rf_mean_recall}")
print(f"  Mean F1-Score: {rf_mean_f1}")
```

3.1.2 Artificial Neural Network

Αρχικά, δημιουργούμε 4 λίστες, οι οποίες θα χρησιμοποιηθούν για να αποθηκεύουν τις μετρικές απόδοσης. **ann_accuracies = [ ], ann_precisions = [ ], ann_recalls = [ ], ann_f1_scores = [ ].** Θέλουμε να υλοποιήσουμε το νευρωνικό σε όλα τα αρχεία που υπάρχουν μέσα στον φάκελος harth, οπότε διατρέχουμε τα αρχεία ένα προς ένα και για κάθε αρχείο εκτελούμε την ίδια διαδικασία. Το αποθηκεύουμε σε ένα dataframe **df** και ύστερα πάνω σε αυτό καλούμε την συνάρτηση **create_lags(df,50),** οπότε για κάθε χρονική στιγμή θα κοιτάμε και τις 50 προηγούμενες. Στην συνέχεια, θα χωρίσουμε το dataframe σε **train** και **test**, το **train** θα αποτελείται από το 80% των δεδομένων και θα είναι τα αρχικά δεδομένα του **df** και το **test** θα αποτελείται από το υπόλοιπο 20%. Επιπλέον, θα διαχωρίσουμε τις στήλες που θα χρησιμοποιούμε (x: ανεξάρτητες μεταβλητές, y εξαρτημένες μεταβλητές). Τα χαρακτηριστικά **x_train, x_test** θα περιλαμβάνουν όλες τις στήλες εκτός από την **timestamp** και την **label** και τα χαρακτηριστικά **y_train, y_test** θα αποτελούνται μόνο από την στήλη **label**. Χρησιμοποιώντας την συνάρτηση **tf.keras.utils.to_categorical( )** κωδικοποιούμε την στήλη **label** με **one-hot encoding**. Ύστερα με την συνάρτηση **tf.keras.models.Sequential( )** αρχικοποιούμε το νευρωνικό μας δίκτυο. Με την συνάρτηση **add( )** κατασκευάζουμε δύο κρυφά επίπεδα με 50 units το καθένα και ενεργοποίηση **ReLU** καθώς και το επίπεδο εξόδου με ενεργοποίηση **softmax**. Για την σύνθεση του νευρωνικού χρησιμοποιούμε την συνάρτηση **compile( ),** η οποία δέχεται σαν όρισμα τον αλγόριθμο βελτιστοποίησης **Adam optimizer**, την συνάρτηση απώλειας **categorical cgossentropy** και την μετρική **accuracy**. Στην συνέχεια κάνουμε fit το μοντέλο με τη συνάρτηση **fit( )**, η οποία δέχεται σαν όρισμα τα δεδομένα εκπαίδευσης, το **batch_size**, τα **epochs** εκπαίδευσης και το **verbose**. Επιπλέον, πραγματοποιούμε τις προβλέψεις με την **predict( )** και υπολογίζουμε τις μετρικές. Αποθηκεύουμε, εκτυπώνουμε τις μετρικές και υπολογίζουμε confusion matrix με την συνάρτηση **confusion_matrix( )** και το εμφανίζουμε ως heatmap. Τέλος, υπολογίζουμε και εκτυπώνουμε τις μέσες τιμές για όλες τις μετρικές για όλους του συμμετέχοντες.

```
import tensorflow as tf

# Λίστες για την αποθήκευση των μετρικών από κάθε αρχείο
ann_accuracies = []
ann_precisions = []
ann_recalls = []
```

```
ann_f1_scores = []

# Διατρέχουμε τα αρχεία στον φάκελο
for filename in os.listdir(path):
    if filename.endswith(".csv"):
        # Φορτώνουμε το CSV αρχείο
        df = pd.read_csv(os.path.join(path, filename))

        # Δημιουργούμε τα lags
        df = create_lags(df, 50)

        # Διαχωρίζουμε σε train και test sets
        train_size = int(0.8 * len(df))
        train_df = df.iloc[:train_size]
        test_df = df.iloc[train_size:]

        # Διαχωρισμός features (X) και labels (y) για το train και test set
        X_train = train_df.drop(['timestamp', 'label'], axis=1)
        y_train = train_df['label']
        X_test = test_df.drop(['timestamp', 'label'], axis=1)
        y_test = test_df['label']

        # Κωδικοποιούμε τα labels σε κατηγορίες one-hot encoding
        y_train = tf.keras.utils.to_categorical(y_train)
        y_test = tf.keras.utils.to_categorical(y_test)


        # Αρχικοποιούμε το Artificial Neural Network
        ann = tf.keras.models.Sequential()

        # Προσθέτουμε το πρώτο κρυφό layer. Χρησιμοποιούμε 50 νευρώνες και την
ενεργοποίηση ReLU. Το input_shape ορίζει την είσοδο, που είναι το training set
        ann.add(tf.keras.layers.Dense(units=50, activation="relu",
input_shape=(X_train.shape[1],)))

        # Προσθέτουμε το δεύτερο κρυφό layer. Προσθέτουμε ακόμα 50 νευρώνες και την
ενεργοποίηση ReLU
        ann.add(tf.keras.layers.Dense(units=50, activation="relu"))

        # Προσθέτουμε το layer εξόδου που έχει τόσους νευρώνες όσες και οι κατηγορίες
που έχουμε στο y_train. Η ενεργοποίηση softmax χρησιμοποιείται για να διασφαλίσουμε ότι
οι τιμές εξόδου είναι πιθανότητες που αθροίζουν σε 1
        ann.add(tf.keras.layers.Dense(units=y_train.shape[1], activation="softmax"))

        # Εφαρμόζουμε το ANN με optimizer "adam" και loss function
"categorical_crossentropy" καθώς έχουμε κατηγορίες που είναι σε μορφή one-hot encoding
        ann.compile(optimizer="adam", loss="categorical_crossentropy",
metrics=['accuracy'])

        # Εκπαιδεύουμε το μοντέλο με τα training data έχοντας batch size 32 και 50
εποχές εκπαίδευσης
        ann.fit(X_train, y_train, batch_size=32, epochs=50, verbose=0)


        # Προβλέπουμε τις δραστηριότητες του test set
        y_pred = ann.predict(X_test)
        y_pred_classes = np.argmax(y_pred, axis=1)
        y_test_classes = np.argmax(y_test, axis=1)

        # Υπολογίζουμε τις μετρικές
        accuracy = accuracy_score(y_test_classes, y_pred_classes)
        precision = precision_score(y_test_classes, y_pred_classes, average='weighted')
        recall = recall_score(y_test_classes, y_pred_classes, average='weighted')
        f1 = f1_score(y_test_classes, y_pred_classes, average='weighted')
```

```
        # Προσθήκη των μετρικών στις λίστες
        ann_accuracies.append(accuracy)
        ann_precisions.append(precision)
        ann_recalls.append(recall)
        ann_f1_scores.append(f1)

        # Εκτύπωση των μετρικών για κάθε συμμετέχοντα
        base_name = os.path.splitext(filename)[0]  # Χωρίς κατάληξη
        print(f"Metrics for Participant {base_name}:")
        print(f"  Accuracy: {accuracy}")
        print(f"  Precision: {precision}")
        print(f"  Recall: {recall}")
        print(f"  F1-Score: {f1}")

        # Υπολογισμός και εμφάνιση του confusion matrix
        unique_labels = sorted(df['label'].unique())
        cm = confusion_matrix(y_test_classes, y_pred_classes)
        plt.figure(figsize=(10, 7))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=unique_labels,
yticklabels=unique_labels)
        plt.title(f'Confusion Matrix for Participant {base_name}')
        plt.xlabel('Predicted Labels')
        plt.ylabel('True Labels')
        plt.show()

# Υπολογισμός της μέσης τιμής για κάθε μετρική από όλους τους συμμετέχοντες
ann_mean_accuracy = np.mean(ann_accuracies)
ann_mean_precision = np.mean(ann_precisions)
ann_mean_recall = np.mean(ann_recalls)
ann_mean_f1 = np.mean(ann_f1_scores)

print("Mean Metrics of All Participants:")
print(f"  Mean Accuracy: {ann_mean_accuracy}")
print(f"  Mean Precision: {ann_mean_precision}")
print(f"  Mean Recall: {ann_mean_recall}")
print(f"  Mean F1-Score: {ann_mean_f1}")
```

3.1.3 Bayesian Network

Αρχικά, δημιουργούμε 4 λίστες, οι οποίες θα χρησιμοποιηθούν για να αποθηκεύουν τις μετρικές απόδοσης. **nbn_accuracies = [ ], nbn _precisions = [ ], nbn _recalls = [ ], nbn _f1_scores = [ ].** Όπως και στο νευρωνικό θέλουμε να υλοποιήσουμε το Bayesian σε όλα τα αρχεία του φακέλου harth, οπότε εκτελούμε την ίδια διαδικασία, διατρέχουμε τα αρχεία ένα προς ένα και κάθε αρχείο το αποθηκεύουμε σε ένα dataframe **df** και ύστερα πάνω σε αυτό καλούμε την συνάρτηση **create_lags(df,50),** οπότε για κάθε χρονική στιγμή θα κοιτάμε και τις 50 προηγούμενες. Στην συνέχεια, χωρίζουμε το dataframe σε **train** και **test**, το **train** θα αποτελείται από το 80% των δεδομένων και θα είναι τα αρχικά δεδομένα του **df** και το **test** θα αποτελείται από το υπόλοιπο 20%. Επιπλέον, διαχωρίζουμε τις στήλες που θα χρησιμοποιούμε (x: ανεξάρτητες μεταβλητές, y εξαρτημένες μεταβλητές). Τα χαρακτηριστικά **x_train, x_test** θα περιλαμβάνουν όλες τις στήλες εκτός από την **timestamp** και την **label** και τα χαρακτηριστικά **y_train, y_test** θα αποτελούνται μόνο από την στήλη **label**. Ύστερα, θα κανονικοποιήσουμε τα δεδομένα χρησιμοποιώντας τον **StandarScaler**. Με την εντολή **scaler.fit_transform(X_train)** ο **StandarScaler** υπολογίζει την μέση τιμή και την τυπική απόκλιση των χαρακτηριστικών στο σύνολο εκπαίδευσης **X_train** και χρησιμοποιώντας αυτές τις πληροφορίες θα κανονικοποιήσει τα δεδομένα. Επίσης, με την εντολή **scaler.transform(X_test)** ο **StandarScaler** μετασχηματίζει τα δεδομένα χρησιμοποιώντας τα αποτελέσματα που υπολογίστηκαν κατά την διαδικασία του fit , με αυτόν τον τρόπο τα δεδομένα εκπαίδευση και δοκιμής κανονικοποιούνται χρησιμοποιώντας τις ίδιες μέσες τιμές και τυπικές αποκλίσεις. Έπειτα, ξεκινάει η κατασκευή του Bayesian, δημιουργούμε ένα αντικείμενο του Gaussian Naive Bayes Classifier καλώντας την συνάρτηση **GaussianNB( )** και καλώντας την συνάρτηση **fit( )**, με ορίσματα τα κανονικοποιημένα δεδομένα της εκπαίδευσης και τα δεδομένα του **y_train**, θα εκπαιδεύσουμε το μοντέλο μας. Χρησιμοποιώντας την συνάρτηση **predict( )** στο εκπαιδευμένο μοντέλο θα προβλέψουμε τα αποτελέσματα (τις δραστηριότητες) των κανονικοποιημένων δεδομένων δοκιμής. Αποθηκεύουμε, εκτυπώνουμε τις μετρικές και

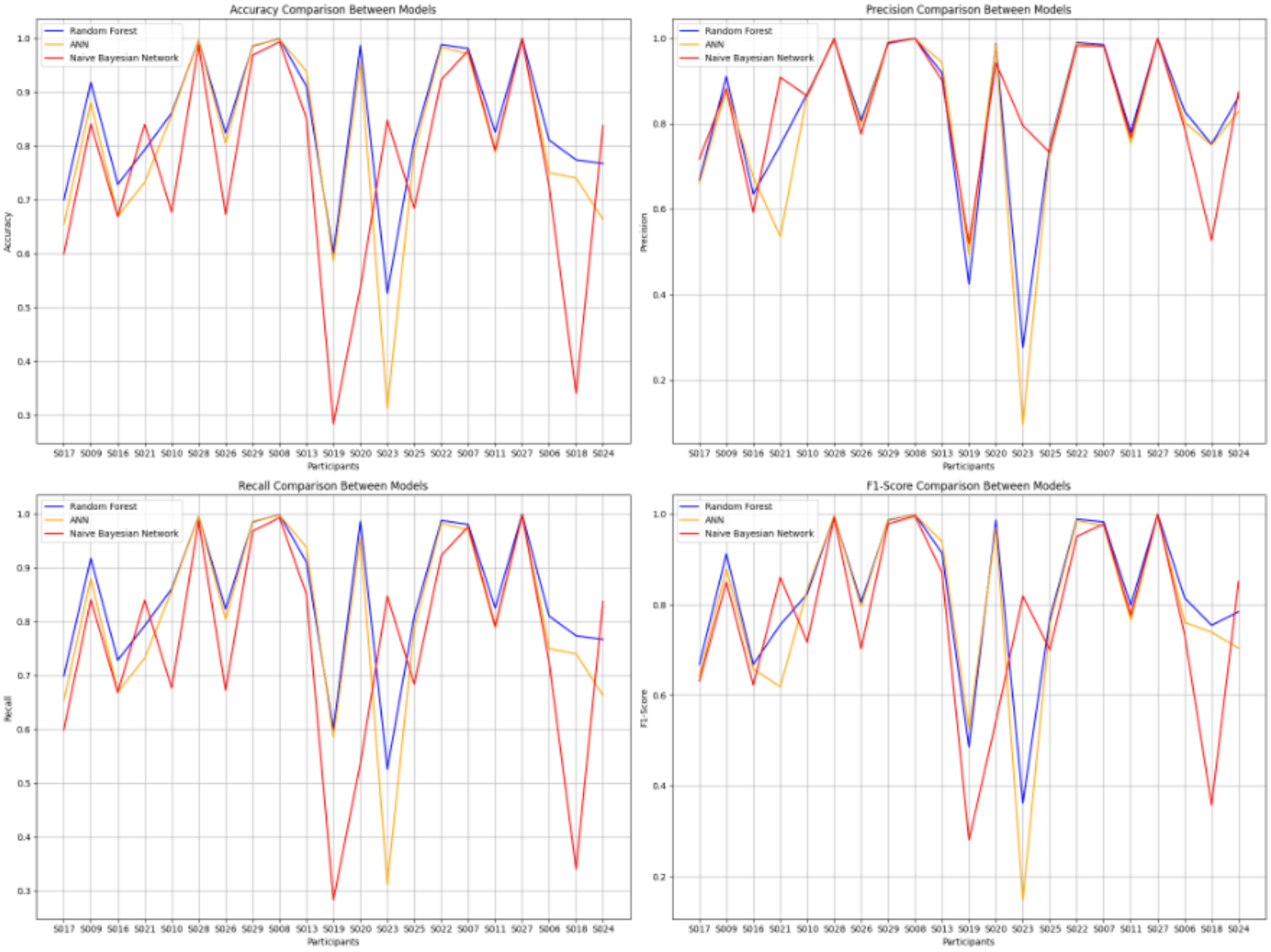
υπολογίζουμε confusion matrix με την συνάρτηση **confusion_matrix( )** και το εμφανίζουμε ως heatmap. Τέλος, υπολογίζουμε και εκτυπώνουμε τις μέσες τιμές για όλες τις μετρικές για όλους του συμμετέχοντες.

```
# Λίστες για την αποθήκευση των μετρικών από κάθε συμμετέχοντα
nbn_accuracies = []
nbn_precisions = []
nbn_recalls = []
nbn_f1_scores = []

# Διάβασμα των αρχείων στο φάκελο
for filename in os.listdir(path):
    if filename.endswith(".csv"):
        # Διάβασμα του CSV αρχείου
        df = pd.read_csv(os.path.join(path, filename))

        # Δημιουργία καθυστερημένων χαρακτηριστικών
        df = create_lags(df, 50)

        # Διαχωρισμός σε σύνολα εκπαίδευσης και δοκιμών
        train_size = int(0.8 * len(df))
        train_df = df.iloc[:train_size]
        test_df = df.iloc[train_size:]

        X_train = train_df.drop(['timestamp', 'label'], axis=1)
        y_train = train_df['label']
        X_test = test_df.drop(['timestamp', 'label'], axis=1)
        y_test = test_df['label']

        # Κανονικοποίηση των δεδομένων
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Εκπαίδευση του μοντέλου Gaussian Naive Bayes
        gnb = GaussianNB()
        gnb.fit(X_train_scaled, y_train)

        # Πρόβλεψη των δραστηριοτήτων για το σύνολο δοκιμών
        y_pred = gnb.predict(X_test_scaled)

        # Υπολογίζουμε τις μετρικές
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, average='weighted')
        recall = recall_score(y_test, y_pred, average='weighted')
        f1 = f1_score(y_test, y_pred, average='weighted')

        # Προσθήκη των μετρικών στις λίστες
        nbn_accuracies.append(accuracy)
        nbn_precisions.append(precision)
        nbn_recalls.append(recall)
        nbn_f1_scores.append(f1)

        # Εκτύπωση των μετρικών για κάθε συμμετέχοντα
        base_name = os.path.splitext(filename)[0]  # Χωρίς κατάληξη
        print(f"Metrics for Participant {base_name}:")
        print(f"  Accuracy: {accuracy}")
        print(f"  Precision: {precision}")
        print(f"  Recall: {recall}")
        print(f"  F1-Score: {f1}")

        # Υπολογισμός και εμφάνιση του confusion matrix
        unique_labels = sorted(df['label'].unique())
        cm = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(10, 7))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=unique_labels,
yticklabels=unique_labels)
```

```
        plt.title(f'Confusion Matrix for Participant {base_name}')
        plt.xlabel('Predicted Labels')
        plt.ylabel('True Labels')
        plt.show()

# Υπολογισμός της μέσης τιμής για κάθε μετρική από όλους τους συμμετέχοντες
nbn_mean_accuracy = np.mean(nbn_accuracies)
nbn_mean_precision = np.mean(nbn_precisions)
nbn_mean_recall = np.mean(nbn_recalls)
nbn_mean_f1 = np.mean(nbn_f1_scores)

print("Mean Metrics of All Participants:")
print(f"  Mean Accuracy: {nbn_mean_accuracy}")
print(f"  Mean Precision: {nbn_mean_precision}")
print(f"  Mean Recall: {nbn_mean_recall}")
print(f"  Mean F1-Score: {nbn_mean_f1}")
```

3.2 Αποτελέσματα

Παρακάτω βλέπουμε συγκριτικά τις τέσσερις μετρικές που χρησιμοποιήσαμε για την αξιολόγηση όλων των μοντέλων:



Συμπεράσματα:

- **Accuracy**: Ο **Random Forest** και ο **ANN** παρουσιάζουν παρόμοια απόδοση και είναι συνήθως ανώτεροι από το **Bayesian Network** σε πολλές περιπτώσεις. Υπάρχουν ωστόσο κάποιες περιπτώσεις όπου οι τιμές είναι χαμηλές, αλλά αυτές φαίνεται να είναι σποραδικές και πιθανόν να οφείλονται σε συγκεκριμένα χαρακτηριστικά των δεδομένων για αυτούς τους συμμετέχοντες.
- **Precision**: Και εδώ ο **Random Forest** και ο **ANN** παρουσιάζουν παρόμοιες αποδόσεις και είναι πιο σταθεροί σε σχέση με το **Bayesian Network**.
- **Recall**: Η ανάκληση δείχνει παρόμοια τάση με την ακρίβεια, με τον **Random Forest** και τον **ANN** να αποδίδουν γενικά καλύτερα από το **Bayesian Network**.
- **F1-Score**: Ο δείκτης F1 συνδυάζει την ακρίβεια και την ανάκληση, και οι τάσεις είναι για ακόμη μια φορά παρόμοιες με αυτές που παρατηρούνται στις άλλες μετρήσεις.

Γενικά, και οι δύο ταξινομητές (**Random Forest** και **ANN**) παρουσιάζουν παρόμοια επίπεδα απόδοσης, με ελαφρώς διαφορετικές τάσεις ανάλογα με τον συμμετέχοντα, αλλά γενικά υπερέχουν του **Bayesian Network** σε όλες τις περιπτώσεις.

Τα συγκεκριμένα συμπεράσματα επιβεβαιώνονται και αν παρατηρήσουμε τη μέση τιμή κάθε μετρικής από όλες τις προβλέψεις συνολικά σε κάθε μοντέλο:

Random Forest

```
Mean Metrics of All Participants:
  Mean Accuracy: 0.8441635604659896
  Mean Precision: 0.8139383465902178
  Mean Recall: 0.8441635604659896
  Mean F1-Score: 0.8181594825177491
```

Artificial Neural Network

```
Mean Metrics of All Participants:
  Mean Accuracy: 0.7750586364670002
  Mean Precision: 0.7621695764832146
  Mean Recall: 0.7750586364670002
  Mean F1-Score: 0.7580208056065826
```

Bayesian Network

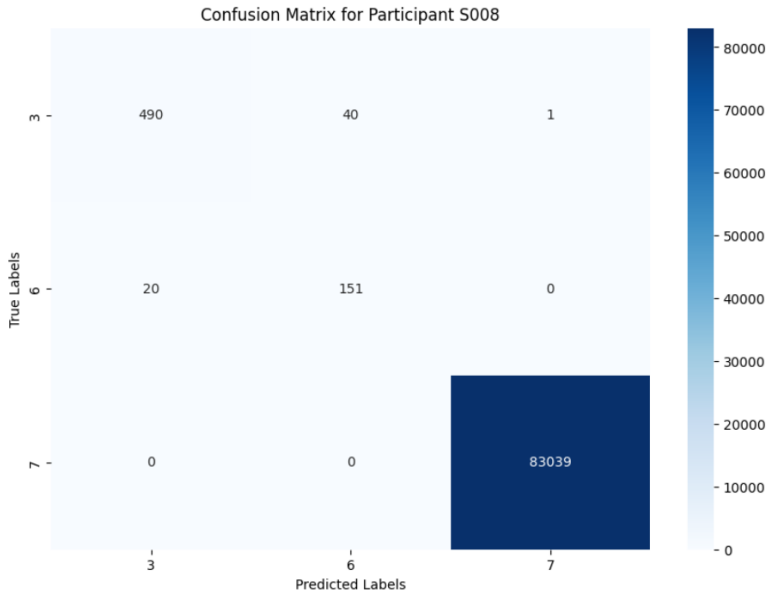
```
Mean Metrics of All Participants:
  Mean Accuracy: 0.7593278928511523
  Mean Precision: 0.8318097395238057
  Mean Recall: 0.7593278928511523
  Mean F1-Score: 0.7686956836502273
```

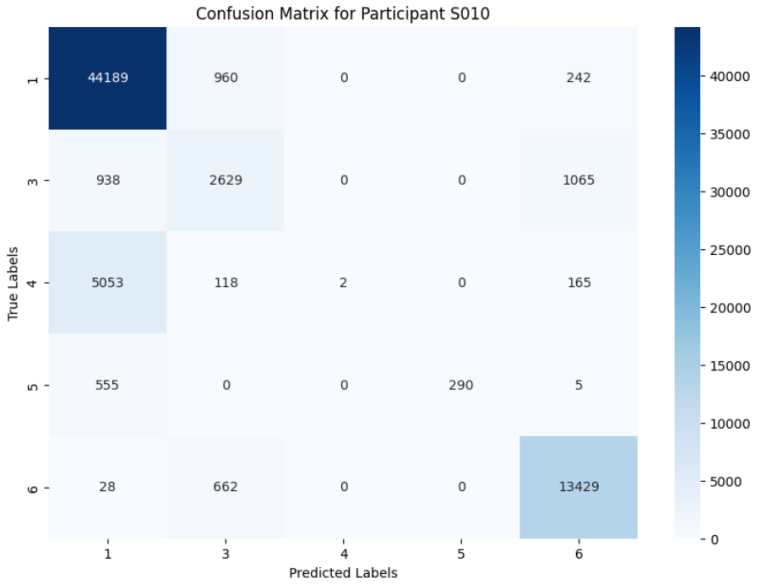
ΣΗΜΕΙΩΣΗ Οι μέσες τιμές των μετρικών για τον συμμετέχοντα **S015** αποκλειστικά στο **Artificial Neural Network** παρουσίασαν μια απροσδόκητη συμπεριφορά με το **Accuracy** να είναι πολύ κοντά στο 0 και συνεπώς επηρεάστηκε και η συνολική απόδοση του μοντέλου. Αν δεν συνέβαινε το παραπάνω αναπάντεχο υπολογίζουμε πως η συνολική μέση τιμή του **Accuracy** θα ήταν της τάξης του $0.80 – 0.81$ αντί για 0.77 και γι' αυτό αποφασίσαμε να μη συμπεριλάβουμε τον συμμετέχοντα **S015** σε καμία γραφική για να έχουμε μια πιο αντικειμενική σύγκριση.

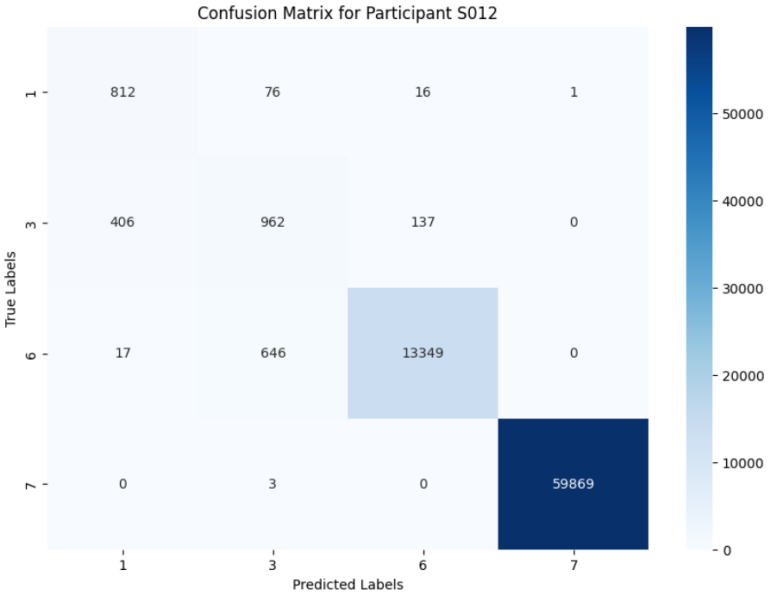
Επιπρόσθετα, βλέπουμε πάλι ενδεικτικά τα confusion matrices και τις μετρικές που προκύπτουν από τη πρόβλεψη κάθε ταξινομητή για τους ίδιους τρεις συμμετέχοντες:

Random Forest

```
Metrics for Participant S008:
  Accuracy: 0.9985918518341727
  Precision: 0.999283508696996
  Recall: 0.9985918518341727
  F1-Score: 0.9989020050958209

Metrics for Participant S010:
  Accuracy: 0.8607848713209156
  Precision: 0.870296227997705
  Recall: 0.8607848713209156
  F1-Score: 0.8236508957484419

Metrics for Participant S012:
  Accuracy: 0.9805054718041918
  Precision: 0.9844702111924505
  Recall: 0.9805054718041918
  F1-Score: 0.9822888811083434
```





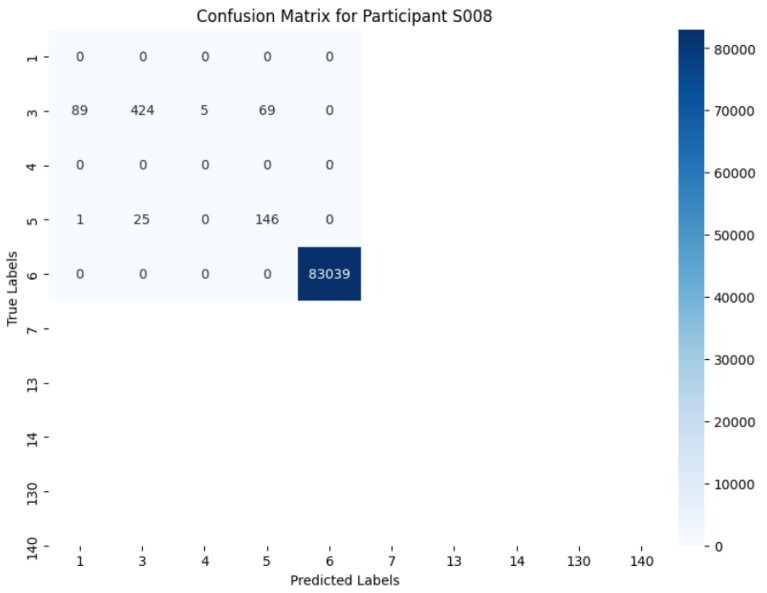


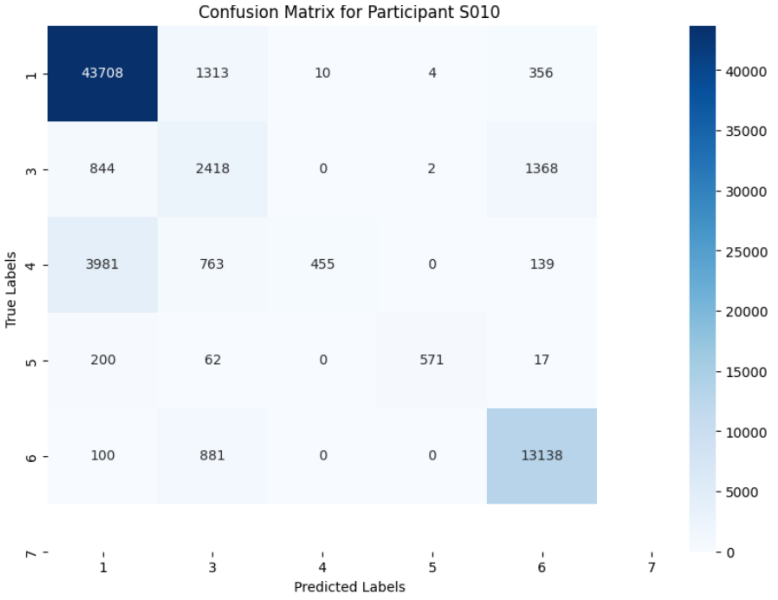
Συμπεράσματα:

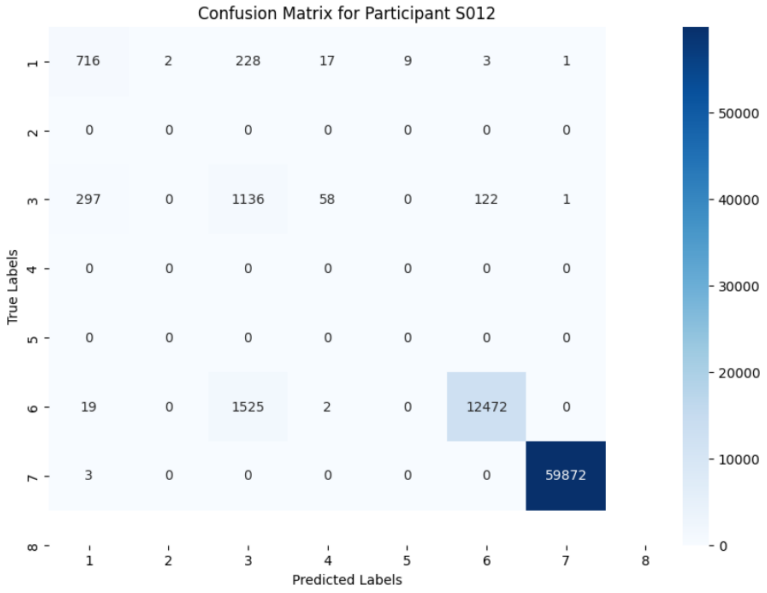
- Βλέπουμε ότι η δραστηριότητα 7 έχει πολύ υψηλό αριθμό σωστών προβλέψεων (π.χ. 83039, 83039, 82763). Αυτό δείχνει ότι το μοντέλο είναι πιθανώς πολύ καλά εκπαιδευμένο στην αναγνώριση της δραστηριότητας 7 για τον συγκεκριμένο συμμετέχοντα.
- Άλλες δραστηριότητες έχουν λίγες ή καθόλου παρατηρήσεις, όπως φαίνεται από τους μικρούς αριθμούς στην κύρια διαγώνιο των πινάκων.

Artificial Neural Network

```
Metrics for Participant S008:
  Accuracy: 0.9977445762428697
  Precision: 0.9989512429694252
  Recall: 0.9977445762428697
  F1-Score: 0.9982249760406756

Metrics for Participant S010:
  Accuracy: 0.8572444191667852
  Precision: 0.8688061035146128
  Recall: 0.8572444191667852
  F1-Score: 0.8330156258954037

Metrics for Participant S012:
  Accuracy: 0.9700979302590118
  Precision: 0.9814172297121889
  Recall: 0.9700979302590118
  F1-Score: 0.974330676729749
```





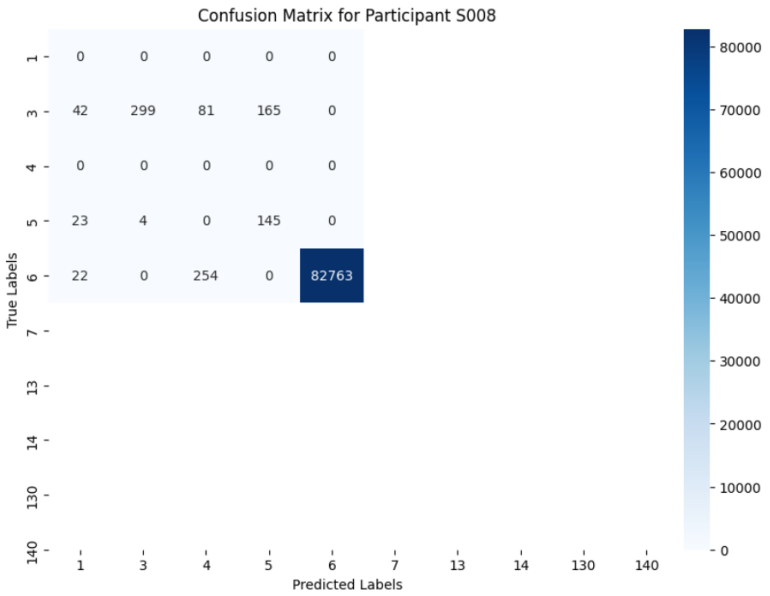


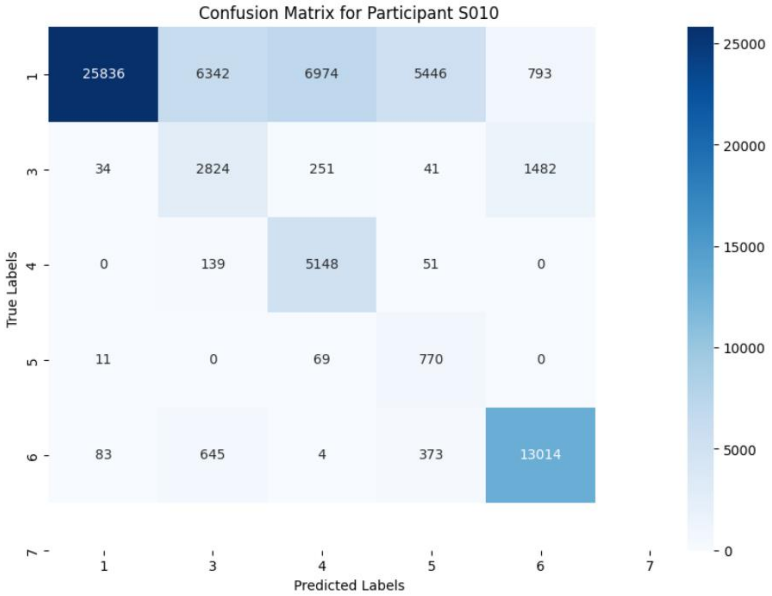
Συμπεράσματα:

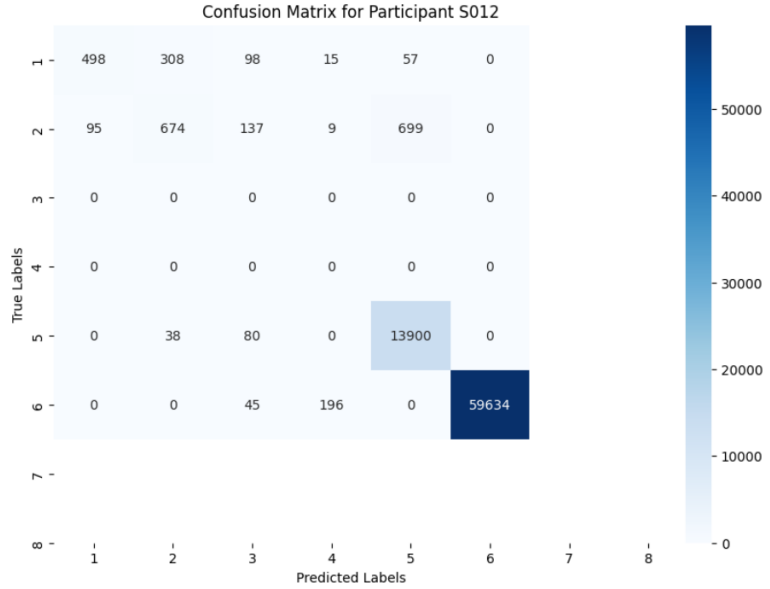
- Η δραστηριότητα 1 έχει υψηλό αριθμό σωστών προβλέψεων (π.χ. 44189, 43708, 25836), αλλά υπάρχουν και πολλές εσφαλμένες προβλέψεις σε άλλες δραστηριότητες.
- Σημαντικός αριθμός παρατηρήσεων της δραστηριότητας 6 έχει ταξινομηθεί λανθασμένα σε άλλες δραστηριότητες (π.χ. 662 προβλέψεις σε δραστηριότητα 4 στον συμμετέχοντα **S008**).
- Γενικά, το μοντέλο παρουσιάζει δυσκολία στην ακριβή ταξινόμηση των παρατηρήσεων για τον συμμετέχοντα **S010** σε όλες τις δραστηριότητες.

Bayesian Network

```
Metrics for Participant S008:
  Accuracy: 0.9929473257118309
  Precision: 0.998815036611682
  Recall: 0.9929473257118309
  F1-Score: 0.99523457592481

Metrics for Participant S010:
  Accuracy: 0.6766955779894782
  Precision: 0.8645795134260345
  Recall: 0.6766955779894782
  F1-Score: 0.7169719804503892

Metrics for Participant S012:
  Accuracy: 0.9767660787364513
  Precision: 0.9813430214901993
  Recall: 0.9767660787364513
  F1-Score: 0.9778714198270776
```







Συμπεράσματα:

- Υψηλός αριθμός σωστών προβλέψεων για την δραστηριότητα 7 (π.χ. 59869, 59872, 59634).
- Υπάρχουν εσφαλμένες προβλέψεις στις δραστηριότητες 1, 3, και 4, κάτι που δείχνει ότι το μοντέλο δυσκολεύεται να διαχωρίσει αυτές τις δραστηριότητες μεταξύ τους.
- Η δραστηριότητα 6 παρουσιάζει επίσης υψηλό αριθμό σωστών προβλέψεων μαζί με την 7 (π.χ. 13349, 12472, 13900).

Γενικό Συμπέρασμα:

Έστω και από αυτό το μικρό δείγμα συμμετεχόντων μπορούμε να καταλήξουμε ότι ο ταξινομητής **Random Forest** φαίνεται να είναι ο πιο καλός συνολικά και πολύ κοντά στο **Artificial Neural Network**. Έχει την καλύτερη απόδοση για τις κρίσιμες δραστηριότητες, με λιγότερες εσφαλμένες προβλέψεις, και διατηρεί υψηλή ακρίβεια ειδικά για την δραστηριότητα 7, σε σύγκριση με τα άλλα δύο μοντέλα. Άρα παρατηρώντας και τα confusion matrices πάλι καταλήγουμε στο ίδιο συμπέρασμα με παραπάνω.

4 Ερώτημα 3ο

4.0 Περιγραφή

4.0.0 Γενικά

Κανονικοποίηση

Επιλέξαμε να εφαρμόσουμε κανονικοποίηση για τα δεδομένα μας, καθώς βοηθά στην σταθεροποίηση της διαδικασίας εκπαίδευσης και τα χαρακτηριστικά κυμαίνονται στην ίδια κλίμακα τιμών, οπότε αποτρέπει ένα χαρακτηριστικό να κυριαρχεί από τα υπόλοιπα λόγω διαφορετικής κλίμακας. Ο μετασχηματιστής που εφαρμόζουμε είναι ο **StandarScaler** της βιβλιοθήκης **transformer**, ο οποίος θα μετασχηματίζει τα δεδομένα ώστε να έχουν μέση τιμή 0 και τυπική απόκλιση 1.

Elbow Method

Η **Elbow Method** είναι μια μέθοδος που χρησιμοποιείται για την εύρεση του βέλτιστου αριθμού συστάδων. Πιο συγκεκριμένα υπολογίζει το σύνολο των τετραγωνικών αποστάσεων των σημείων από τα κοντινότερα κέντρα συστάδων για ένα πλήθος τιμών. Η γραφική παράσταση απεικονίζει της διακυμάνσεις για τα διάφορα k και ο βέλτιστος αριθμός συστάδων θα είναι το σημείο «αγκώνας», όπου ο ρυθμός μείωσης της διακύμανσης εξισώνεται απότομα.

Διαμοιρασμός Συμμετεχόντων σε Συστάδες

Όσον αφορά τους αλγορίθμους **K-Means** και **Gaussian Mixture**, το κριτήριο με βάση το οποίο τοποθετήσαμε τους συμμετέχοντες στις συστάδες που παράχθηκαν από όλα τα δεδομένα των μετρήσεων ήταν η απόσταση της μέσης τιμής κάθε γνωρίσματος του συμμετέχοντα από το κεντροειδές της συστάδας. Πιο συγκεκριμένα, αποφασίσαμε να αναπαραστήσουμε κάθε συμμετέχοντα ως ένα εξαδιάστατο διάνυσμα στο χώρο το οποίο έχει ως συντεταγμένες τη μέση τιμή κάθε γνωρίσματός του και έπειτα υπολογίσαμε την Ευκλείδεια Απόστασή του από το εξαδιάστατο διάνυσμα του κεντροειδούς κάθε συστάδας. Έτσι, η συστάδα στην οποία θα άνηκε ο κάθε συμμετέχοντας εν τέλει θα ήταν αυτή η οποία είχε την ελάχιστη Ευκλείδεια Απόσταση από το διάνυσμα του συμμετέχοντα. Από την άλλη, όσον αφορά το **Kohonen Network** η Ευκλείδεια Απόσταση ανάμεσα στα δείγματα κάθε συμμετέχοντα με τους νευρώνες του δικτύου που αντιπροσωπεύουν τις συστάδες υπολογίζεται αυτόματα από τον αλγόριθμο.

4.0.1 K-Means

Ο **K-Means** είναι διαμεριστικός αλγόριθμος και στόχος του είναι να βρει την κατάλληλη συσταδοποίηση που θα ελαχιστοποιεί το αποτέλεσμα της συνάρτησης του αθροίσματος των τετραγώνων των σφαλμάτων (**SSE**) από τα πλησιέστερα κέντρα των συστάδων χρησιμοποιώντας μια άπληστη τεχνική. Επίσης, ο συγκεκριμένος αλγόριθμος παράγει συστάδες με κυρτό σχήμα και ο συνολικός χρόνος εκτέλεσης που απαιτεί είναι O(tnkd), όπου **t**: οι επαναλήψεις που εκτελούνται, **n**: τα σημεία που έχουμε, **k**: οι συστάδες, **d**: οι d πράξεις που απαιτούνται στις **d** διαστάσεις. Ο αλγόριθμος λειτουργεί επαναληπτικά αρχικά ορίζει με τυχαίο τρόπο **k** σημεία στο χώρο των δεδομένων και στην συνέχεια αντιστοιχίζει τα δεδομένα στις συστάδες (τα τοποθετεί στην συστάδα που απέχει την μικρότερη απόσταση από το κέντρο βάρους της) και ενημερώνει τα κέντρα βάρους, . Στη συνέχεια υπολογίζονται οι νέες μέσες τιμές για κάθε συστάδα και η διαδικασία επαναλαμβάνεται μέχρι να βρούμε τα τοπικά ελάχιστα.

4.0.2 Kohonen Network (Self-Organizing Map)

Τα δίκτυα **Kohonen** ανήκουν στη κατηγορία της μη επιβλεπόμενης μηχανικής μάθησης, η οποία επιτρέπει στο νευρωνικό δίκτυο να εκπαιδευτεί χωρίς τη παρουσία δασκάλου και να αποφασίσει από μόνο του ποιες είναι οι συστάδες που προκύπτουν αλλά και να τοποθετήσει τα δεδομένα εισόδου σε αυτές. Πιο συγκεκριμένα, το νευρωνικό δίκτυο δέχεται ως εισόδους τις τιμές από τα γνωρίσματα όλων των συμμετεχόντων και παράγει ως εξόδους τις συστάδες που

προκύπτουν από αυτές έχοντας προσθέσει στις ακμές και τα κατάλληλα βάρη. Επίσης, το δίκτυο στηρίζεται σημαντικά και στην έννοια της ανταγωνιστικής μάθησης, καθώς στόχος του είναι η εύρεση ενός νευρώνα ο οποίος προσεγγίζει περισσότερο το πρότυπο εισόδου. Το δίκτυο στη συνέχεια τροποποιεί αυτό τον νευρώνα και τους γειτονικούς του έτσι ώστε να μοιάζουν περισσότερο με το πρότυπο αυτό. Τέλος, ο λόγος που επιλέξαμε το συγκεκριμένο δίκτυο είναι επειδή διευκολύνει την απεικόνιση υψηλής διάστασης δεδομένων σε χαμηλότερη διάσταση.

Αρχικοποίηση

Το **SOM** αποτελείται από ένα πλέγμα νευρώνων μεγέθους **som_size**. Κάθε νευρώνας έχει έναν διάνυσμα βαρών που έχει την ίδια διάσταση με τα δεδομένα εισόδου. Τα βάρη των νευρώνων αρχικοποιούνται τυχαία ή μέσω μιας μεθόδου που καλύπτει το εύρος των δεδομένων εισόδου.

Εκπαίδευση

Η εκπαίδευση του **SOM** περιλαμβάνει τα εξής βήματα για κάθε δεδομένο εκπαίδευσης:

- Για κάθε δεδομένο εισόδου, υπολογίζεται η απόσταση (συνήθως η ευκλείδεια απόσταση) μεταξύ του διανύσματος εισόδου και του διανύσματος βαρών κάθε νευρώνα.
- Ο νευρώνας με τη μικρότερη απόσταση από το δεδομένο εισόδου ονομάζεται **"Best Matching Unit"** (**BMU**).
- Τα βάρη του **BMU** και των γειτονικών νευρώνων ενημερώνονται για να πλησιάσουν το δεδομένο εισόδου.

Επίσης, έχουμε τις εξής παραμέτρους:

- **len(columns)**: Ο αριθμός των χαρακτηριστικών των δεδομένων.
- **sigma**: Η διασπορά της συνάρτησης γειτονιάς.
- **learning_rate**: Ο αρχικός ρυθμός εκμάθησης.
- **random_seed**: Ορίζει ένα seed για αναπαραγωγιμότητα.

Τέλος, το νευρωνικό δίκτυο εκπαιδεύεται με συνεχείς επαναλήψεις του αλγορίθμου με τυχαία επιλογή των δεδομένων. Μετά την εκπαίδευση για να γίνει η συσταδοποίηση το δίκτυο εντοπίζει το νευρώνα που είναι ο "νικητής" για το εκάστοτε σημείο δεδομένων **x** του συνόλου δεδομένων, υπολογίζοντας την απόσταση μεταξύ του διανύσματος χαρακτηριστικών **x** και των βαρών **w***i* κάθε νευρώνα **i**.

4.0.3 Gaussian Mixture Model

Το **Gaussian Mixture Model** είναι ένα πιθανοτικό μοντέλο που χρησιμοποιείται για την ομαδοποίηση και την εκτίμηση πυκνότητας. Το **GMM** αποδίδει πιθανότητες στα σημεία δεδομένων, επιτρέποντάς τους να ανήκουν σε πολλές συστάδες ταυτόχρονα και υποθέτει ότι τα δεδομένα παράγονται από πολλές γκαουσιανές συνιστώσες καθεμία από τις οποίες αντιπροσωπεύει μια συστάδα. Επίσης, το μοντέλο υποθέτει ότι υπάρχει ένας ορισμένος αριθμός συνιστωσών, όπου κάθε συνιστώσα είναι μια κατανομή Gauss. Άρα, ομαδοποιεί τα δεδομένα που ανήκουν σε μια μόνο γκαουσιανή συνιστώσα.

4.1 Υλοποίηση

4.1.1 K-Means

Για την υλοποίηση του k-means ξεκινάμε ορίζοντας το path που βρίσκονται τα αρχεία και δημιουργώντας 3 λίστες η **all_data[ ]** που θα χρησιμοποιηθεί για την αποθήκευση των δεδομένων από όλα τα αρχεία, η **file_names[ ]** θα χρησιμοποιηθεί για την αποθήκευση των ονομάτων των αρχείων και η **mean_value_list[ ]** που θα χρησιμοποιηθεί για την αποθήκευση των μέσων τιμών κάθε αρχείου. Διατρέχουμε όλα τα αρχεία στο φάκελο και για καθένα από αυτά αποθηκεύουμε τα δεδομένα σε ένα dataframe **df** και προσθέτουμε στην λίστα **file_names** το όνομα του αρχείου. Στην συνέχεια, εξάγουμε τα δεδομένα από τις στήλες **'back_x'**, **'back_y'**, **'back_z'**, **'thigh_x'**, **'thigh_y'**, **'thigh_z'**, τα προσθέτουμε σε ένα dataframe **X** και στην συνέχεια στην λίστα **all_data_append**. Για κάθε στήλη που επιλέξαμε υπολογίζουμε τις μέσες τιμές με την συνάρτηση **mean()** και τις αποθηκεύουμε στη λίστα **mean_values_list**. Όταν το πρόγραμμα διατρέξει όλα τα αρχεία στον φάκελο ενώνει όλα τα δεδομένα από τα αρχεία κάθετα, με την συνάρτηση **concatenate( ),** σε έναν ενιαίο πίνακα **all_data_comdined**. Ύστερα, με την μέθοδο **fit_transform( )** θα υπολογίσουμε την μέση τιμή και την τυπική απόκλιση των δεδομένων και στην συνέχεια θα κανονικοποιήσουμε τα δεδομένα μας, χρησιμοποιώντας τον μετασχηματιστή **StandarScaler** πάνω στα υπολογισμένα στατιστικά μεγέθη και θα τα αποθηκεύσουμε στην λίστα **all_data_scaled**. Πριν εφαρμόσουμε τον k-means επιλέξαμε να χρησιμοποιήσουμε την elbow method, ώστε να βρούμε τον κατάλληλο αριθμό clusters για τα δεδομένα που έχουμε. Οπότε υπολογίζουμε το SSE για διαφορετικό πλήθος συστάδων(από 1 έως 9). Για κάθε διαφορετικό αριθμό συστάδων εκπαιδεύει το μοντέλο **KMeans** και αποθηκεύει το **insertia** (μετρά πόσο καλά τα δεδομένα ομαδοποιούνται γύρω από τα κέντρα των συστάδων τους) στη λίστα **sse**.Τέλος, σχεδιάζουμε την καμπύλη **SSE**.

```
import os
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Ορισμός του φακέλου που περιέχει τα αρχεία CSV
path = '/content/drive/MyDrive/harth'

# Λίστες για την αποθήκευση των δεδομένων από όλα τα αρχεία
all_data = []
# Λίστα για τα ονόματα των αρχείων
file_names = []
# Λίστες για τις μέσες τιμές από κάθε αρχείο
mean_values_list = []

# Διάβασμα των αρχείων στο φάκελο
for filename in os.listdir(path):
    if filename.endswith(".csv"):
        # Διάβασμα του CSV αρχείου
        df = pd.read_csv(os.path.join(path, filename))
        file_names.append(filename)  # Κρατάμε το όνομα του αρχείου για κάθε δείγμα

        # Επιλογή των συγκεκριμένων χαρακτηριστικών
        columns = ['back_x', 'back_y', 'back_z', 'thigh_x', 'thigh_y', 'thigh_z']
        X = df[columns]

        # Αποθήκευση των δεδομένων στη λίστα
        all_data.append(X)

        # Υπολογισμός μέσης τιμής κάθε στήλης
        mean_values = X.mean().values

        # Προσθήκη των μέσων τιμών στη λίστα
        mean_values_list.append(mean_values)

# Συνδυασμός όλων των δεδομένων σε ένα συνολικό πίνακα
all_data_combined = np.concatenate(all_data, axis=0)
```

```
# Κανονικοποίηση των πλήρων δεδομένων
scaler = StandardScaler()
all_data_scaled = scaler.fit_transform(all_data_combined)

# Υπολογισμός του SSE για διαφορετικούς αριθμούς συστάδων
sse = []
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(all_data_scaled)
    sse.append(kmeans.inertia_)

# Σχεδίαση της καμπύλης SSE
plt.plot(range(1, 10), sse, marker='o', color='blue')
plt.xlabel('Number of clusters')
plt.ylabel('SSE')
plt.title('Elbow Method for Optimal k')
plt.show()
```

Από την παραπάνω γραφική παρατηρήσαμε ότι το κατάλληλο πλήθος συστάδων είναι το 3, οπότε το αποθηκεύουμε στην μεταβλητή **optimal**. Ξεκινάμε την εφαρμογή του αλγορίθμου **K-Means**, αρχικοποιούμε τον αλγόριθμο **kmeans** με 3 συστάδες και ορίζουμε και μια σταθερή τιμή για το **random_state.** Στην συνέχεια εκπαιδεύουμε τον αλγόριθμο με τα κανονικοποιημένα δεδομένα χρησιμοποιώντας την συνάρτηση **fit( )**. Όταν τελειώσει η εκπαίδευση εκτυπώνουμε τα κέντρα των συστάδων (είναι οι μέσες τιμές των δεδομένων που ανήκουν σε κάθε συστάδα). Ύστερα, δημιουργούμε ένα dataframe **mean_values_df** το οποίο θα περιέχει τα ονόματα των αρχείων ως δείκτες και τις μέσες τιμές των μετρήσεων που είχαμε υπολογίσει παραπάνω ως δεδομένα. Θα εφαρμόσουμε κανονικοποίηση και στις μέσες τιμές και με βάση αυτές τις τιμές θα προβλέψουμε με την συνάρτηση **predict( )** σε ποια συστάδα ανήκει κάθε αρχείο. Θα κατασεκυάσουμε το **clusters_df,** το οποίο θα περιέχει τα όνομα του αρχείου και την συστάδα που έχει τοποθετηθεί. Τέλος, ομαδοποιούμε με την συνάρτηση **groupby( )** τα αρχεία ανά συστάδα και εκτυπώνουμε τα ονόματα των αρχείων κάτω από την συστάδα στην οποία ανήκουν.

```
# Επιλογή κατάλληλου πληθους συστάδων
optimal = 3

# Εφαρμογή του αλγορίθμου k-means
kmeans = KMeans(n_clusters=optimal, random_state=42)  # Ορισμός αριθμού συστάδων και
τυχαίας κατάστασης για επαναληψιμότητα
kmeans.fit(all_data_scaled)

# Εκτύπωση των κέντρων των συστάδων (clusters)
print("Cluster Centers:")
for center in kmeans.cluster_centers_:
    print(center)

# Δημιουργία DataFrame με τα ονόματα των αρχείων ως δείκτες και τις μέσες τιμές ως
δεδομένα
mean_values_df = pd.DataFrame(mean_values_list, columns=columns, index=file_names)

# Κανονικοποίηση των μέσων τιμών
scaler = StandardScaler()
mean_values_scaled = scaler.fit_transform(mean_values_df)

# Προβλέψεις για τις συστάδες των μέσων τιμών
cluster_assignments = kmeans.predict(mean_values_scaled)

# Δημιουργία DataFrame με τα αρχεία και τις αντίστοιχες συστάδες
clusters_df = pd.DataFrame({'filename': file_names, 'cluster': cluster_assignments})

# Ομαδοποίηση και εκτύπωση των αρχείων ανά συστάδα
grouped = clusters_df.groupby('cluster')['filename'].apply(list)

print("\nPlacement of Participants in Clusters")
for cluster, files in grouped.items():
    print(f"Cluster {cluster}:")
```

```
    for file in files:
        # Χωρίς κατάληξη
        base_name = os.path.splitext(file)[0]
        print(f"  {base_name}")
```

4.1.2 Kohonen Network

Αρχικά, ορίζουμε τις παραμέτρους του δικτύου, δηλαδή το μέγεθος του πλέγματος, τη διασπορά **sigma** της συνάρτησης γειτνίασης, το ρυθμό μάθησης **learning_rate** και ένα τυχαίο **seed**. Έπειτα, τα περνάμε ως ορίσματα στη συνάρτηση **MiniSom** και δημιουργούμε ένα αντικείμενο **som.** Στο αντικείμενο αυτό καλούμε τη συνάρτηση **train_random( )**, η οποία πραγματοποιεί 100 επαναλήψεις του αλγορίθμου εκπαίδευσης λαμβάνοντας τυχαία δείγματα κάθε φορά από τη λίστα **all_data_scaled** που περιέχει τα κανονικοποιημένα δεδομένα. Έπειτα, δημιουργούμε ένα λεξικό **cluster_assignments** το οποίο αποθηκεύει τις αναθέσεις κάθε συμμετέχοντα σε συστάδες. Στη συνέχεια, χρησιμοποιούμε τον **scaler** που έχει εκπαιδευτεί στα δεδομένα για να κλιμακώσει τις μέσες τιμές των χαρακτηριστικών (μέσω της **np.mean( )**). Αφού κλιμακώσουμε τις μέσες τιμές, διατρέχουμε κάθε μία από αυτές και βρίσκουμε τη θέση του νικητή νευρώνα στο εκπαιδευμένο **SOM** πλέγμα που αντιστοιχεί σε αυτήν μέσω της συνάρτησης **winner( )**. Για να αντιστοιχίσουμε σε ποια συστάδα ανήκει κάθε συμμετέχοντας αποθηκεύουμε το όνομα του αρχείου και τη θέση του νικητή νευρώνα στη λίστα **cluster_assignments**. Τέλος, δημιουργούμε ένα λεξικό με κλειδιά από 0 έως **som_size** * **som_size** - 1), όπου κάθε κλειδί αντιπροσωπεύει μια συστάδα και η τιμή είναι μια κενή λίστα που θα περιέχει τα ονόματα των αρχείων που ανήκουν σε αυτή τη συστάδα. Στο λεξικό διατρέχουμε κάθε ανάθεση συστάδας από τη λίστα **cluster_assignments**, η οποία περιέχει τα ονόματα των αρχείων και τις θέσεις των νικητών νευρώνων. Με την εντολή **cluster_index = cluster[0] * som_size + cluster[1]** υπολογίζουμε τον μοναδικό αριθμό συστάδας μετατρέποντας τη θέση του νικητή νευρώνα (cluster) σε μονοδιάστατο δείκτη και προσθέτουμε το όνομα του αρχείου στη λίστα της αντίστοιχης συστάδας στο λεξικό **cluster_dict**. Η εκτύπωση των τελικών συστάδων γίνεται με παρόμοιο τρόπο με πριν. Για την οπτικοποίηση του **SOM** δημιουργούμε ένα γράφημα και τοποθετούμε μέσα στα περιγράμματα των κελιών τα ονόματα των αρχείων στις θέσεις των νικητών νευρώνων.

```
from minisom import MiniSom

# Ορισμός παραμέτρων για το SOM
som_size = 2
som = MiniSom(som_size, som_size, len(columns), sigma=0.2, learning_rate=0.2,
random_seed=42) # som_size: μέγεθος του πλέγματος SOM (10x10), sigma: διασπορά της
συνάρτησης γειτονιάς (0.5), learning_rate: αρχικός ρυθμός εκμάθησης (0.5)

# Εκπαίδευση του SOM
som.train_random(all_data_scaled, 100)  # 100 επαναλήψεις του αλγορίθμου

# Ανάθεση κάθε συμμετέχοντα σε μια συστάδα
cluster_assignments = []
mean_values_scaled = scaler.transform([np.mean(X, axis=0) for X in all_data])
for i, x in enumerate(mean_values_scaled):
    # x: κλιμακωμένη μέση τιμή του τρέχοντος στοιχείου
    # win_position: θέση του νικητή στο SOM
    win_position = som.winner(x)
    cluster_assignments.append((file_names[i], win_position))

# Δημιουργία λεξικού για τις συστάδες
cluster_dict = {i: [] for i in range(som_size * som_size)}
for file_name, cluster in cluster_assignments:
    cluster_index = cluster[0] * som_size + cluster[1]
    cluster_dict[cluster_index].append(file_name)

# Εκτύπωση των τελικών συστάδων
print("\nPlacement of Participants in Clusters")
for cluster, files in cluster_dict.items():
    print(f"Cluster {cluster}:")
    for file in files:
        base_name = os.path.splitext(file)[0]
        print(f"  {base_name}")
```

```
# Οπτικοποίηση του SOM
plt.figure(figsize=(10, 10))
for file_name, (x, y) in cluster_assignments:
    offset_x = np.random.uniform(-0.4, 0.4)
    offset_y = np.random.uniform(-0.4, 0.4)
    base_name = os.path.splitext(file_name)[0]
    plt.text(x + 0.5 + offset_x, y + 0.5 + offset_y, base_name, fontsize=8,
ha='center', va='center',
             bbox=dict(facecolor='white', alpha=0.5, lw=0))

# Προσθέτουμε τα περιγράμματα για τα κελιά του SOM
plt.xlim([0, som_size])
plt.ylim([0, som_size])
plt.xticks(np.arange(som_size+1))
plt.yticks(np.arange(som_size+1))
plt.grid()
plt.title('Clustering of Participants Based on Activities (Kohonen Network)')
plt.gca().invert_yaxis()  # Αντιστρέφουμε τον άξονα y για καλύτερη απεικόνιση
plt.show()

# Μετατροπή των συστάδων από θέσεις στο SOM σε μοναδικούς αριθμούς συστάδων
labels = [cluster[0] * som_size + cluster[1] for _, cluster in cluster_assignments]

# Υπολογισμός του Silhouette Score
silhouette_avg = silhouette_score(mean_values_scaled, labels)
print(f"Silhouette Score: {silhouette_avg}")
```

4.1.3 Gaussian Mixture

Όπως και για τον **K-Means** έτσι και για το **Gaussian Mixture** πριν την υλοποίησή του επιλέξαμε να χρησιμοποιήσουμε την elbow method, ώστε να βρούμε τον κατάλληλο αριθμό clusters για τα δεδομένα που έχουμε. Ξεκινάμε ορίζοντας μια λίστα **insertia[ ]**, η οποία θα χρησιμοποιηθεί για την αποθήκευση των τιμών του **BIC** (μέτρο αξιολόγησης) για κάθε αριθμό συστάδων που δοκιμάζονται. Στη συνέχεια, για διαφορετικό πλήθος συστάδων θα εφαρμόσουμε την συνάρτηση **GaussianMixture( )** και θα εκπαιδεύσουμε το μοντέλο πάνω στα δεδομένα που υπάρχουν στην λίστα **all_data_scaled**. Επιπλέον, θα ποθηκεύσουμε στη λίστα **insertia** τις τιμές που θα προκύψουν χρησιμοποιώντας την συνάρτηση **BIC**. Τέλος, θα απεικονίσουμε γραφικά τις τιμές του **BIC** για κάθε πλήθος συστάδων.

```
from sklearn.mixture import GaussianMixture
from collections import Counter

inertia = [] # Λίστα για την αποθήκευση των inertia για κάθε αριθμό συστάδων

# Δοκιμάζουμε διαφορετικούς αριθμούς συστάδων
for k in range(1, 10):
    gmm = GaussianMixture(n_components=k, random_state=42)
    gmm.fit(all_data_scaled)
    inertia.append(gmm.bic(all_data_scaled))  # Χρησιμοποιούμε το Bayesian Information
Criterion (BIC) ως μέτρο αξιολόγησης

# Οπτικοποίηση του Elbow Method
plt.plot(range(1, 10), inertia, marker='o', color='blue')
plt.xlabel('Number of Clusters')
plt.ylabel('BIC')
plt.title('Elbow Method for Optimal k')
plt.xticks(range(1, 10))
plt.grid(True)
plt.show()
```

Από την παραπάνω γραφική παρατηρήσαμε ότι το κατάλληλο πλήθος συστάδων είναι το 3, οπότε το αποθηκεύουμε στην μεταβλητή **optimal**. Ξεκινάμε την εφαρμογή του αλγορίθμου **Gaussian Mixture** καλώντας την συνάρτηση **GaussianMixture( )** και δίνοντας σαν όρισμα 3 συστάδες και μια σταθερή τιμή για το **random_state**. Στην συνέχεια, εκπαιδεύουμε τον αλγόριθμο με τα κανονικοποιημένα δεδομένα χρησιμοποιώντας την συνάρτηση **fit( )** και εκτυπώνουμε τα κέντρα των συστάδων (είναι οι μέσες τιμές των δεδομένων που ανήκουν σε κάθε συστάδα). Επίσης, δημιουργούμε μια λίστα **mean_values_list** με τις μέσες τιμές για κάθε συμμετέχοντα και την αποθηκεύουμε και στο dataframe **mean_values_df** μαζί με τα ονόματα των αρχείων ως δείκτες. Ύστερα, εφαρμόζουμε κανονικοποίηση στις μέσες τιμές και με βάση αυτές θα υπολογίσουμε την απόσταση κάθε συμμετέχοντα από τα κεντροειδή χρησιμοποιώντας την συνάρτηση **np.linalg.norm( ).** Επιπλέον, με την συνάρτηση **np.argmin( )** θα αντιστοιχίσουμε κάθε συμμετέχοντα στην συστάδα με την μικρότερη απόσταση και τα αποτελέσματα θα τα αποθηκεύσουμε στη μεταβλητή **cluster_assignments**. Ακόμη, θα κατασκευάσουμε το **clusters_df**, το οποίο θα περιέχει το όνομα του αρχείου και την συστάδα που έχει τοποθετηθεί. Τέλος, ομαδοποιούμε με την συνάρτηση **groupby( )** τα αρχεία ανά συστάδα, εκτυπώνουμε τα ονόματα των αρχείων κάτω από την συστάδα στην οποία ανήκουν και απεικονίζουμε γραφικά την θέση των συμμετεχόντων ανά συστάδα.

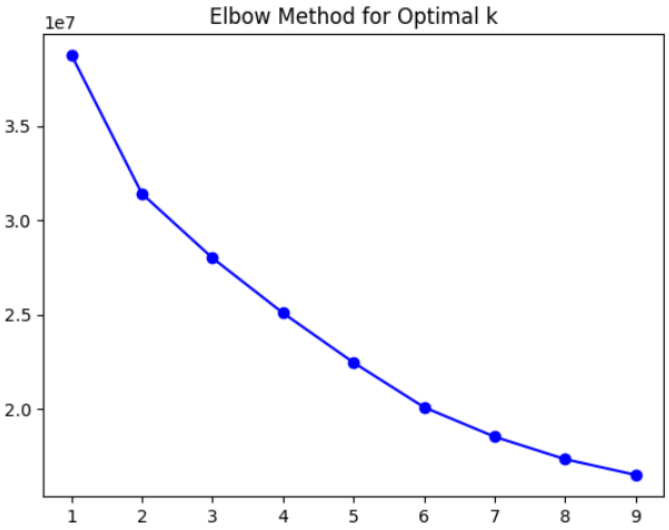
```
# Επιλογή κατάλληλου πληθους συστάδων
optimal = 3

# Εφαρμογή του Gaussian Mixture Model
gmm = GaussianMixture(n_components=optimal, random_state=42)
gmm.fit(all_data_scaled)

# Εκτύπωση των κεντροειδών των συστάδων
print("Cluster Centers (GMM):")
for i, center in enumerate(gmm.means_):
    print(f"Cluster {i + 1}: {center}")

# Δημιουργία DataFrame με τις μέσες τιμές για κάθε συμμετέχοντα
mean_values_list = [np.mean(data, axis=0) for data in all_data]
mean_values_df = pd.DataFrame(mean_values_list, columns=columns,
index=[os.path.splitext(f)[0] for f in file_names])

# Κανονικοποίηση των μέσων τιμών
scaler = StandardScaler()
mean_values_scaled = scaler.fit_transform(mean_values_df)

# Υπολογισμός αποστάσεων από τα κεντροειδή
distances = np.linalg.norm(mean_values_scaled[:, np.newaxis] - gmm.means_, axis=2)

# Αντιστοίχιση κάθε συμμετέχοντα στη συστάδα με την ελάχιστη απόσταση
cluster_assignments = np.argmin(distances, axis=1)

# Δημιουργία DataFrame με τα αρχεία και τις αντίστοιχες συστάδες
clusters_df = pd.DataFrame({'filename': mean_values_df.index, 'cluster':
cluster_assignments})

# Ομαδοποίηση και εκτύπωση των αρχείων ανά συστάδα
grouped = clusters_df.groupby('cluster')['filename'].apply(list)

print("\nPlacement of Participants in Clusters (GMM):")
for cluster, files in grouped.items():
    print(f"Cluster {cluster}:")
    for file in files:
        print(f"  {file}")

# Οπτικοποίηση των συστάδων και των σημείων των συμμετεχόντων
plt.figure(figsize=(12, 8))

# Δημιουργία χρωματικής παλέτας για τις συστάδες
unique_labels = set(cluster_assignments)
colors = plt.cm.get_cmap('tab10', len(unique_labels)).colors

# Προσθήκη των σημείων στις συντεταγμένες ανά συστάδα
```

```
for i, assignment in enumerate(cluster_assignments):
    cluster = assignment
    x = cluster + np.random.rand() * 0.1 - 0.05  # Μικρή τυχαία μετατόπιση
    y = np.random.rand() * 0.1 - 0.05  # Μικρή τυχαία μετατόπιση
    plt.scatter(x, y, s=100, color=colors[cluster % len(colors)], label=f'Cluster
{cluster + 1}' if i == 0 else "")
    plt.text(x + 0.02, y, clusters_df['filename'][i], fontsize=9, ha='left')  # Σταθερή
μετατόπιση για την ετικέτα

# Προσθήκη υπομνήματος
for cluster in unique_labels:
    plt.scatter([], [], color=colors[cluster % len(colors)], label=f'Cluster {cluster +
1}')

plt.title('Clustering of Participants Based on Activities (GMM)')
plt.axis('off')
plt.show()

# Υπολογισμός του Silhouette Score
silhouette_avg = silhouette_score(mean_values_scaled, cluster_assignments)
print(f"\nSilhouette Score for GMM: {silhouette_avg})”
```

4.2 Αποτελέσματα

4.2.0 K-Means

Παρακάτω βλέπουμε το γράφημα που προέκυψε εφαρμόζοντας το **Elbow Method w/ SSE** προκειμένου να καθορίσουμε το ιδανικό πλήθος συστάδων:

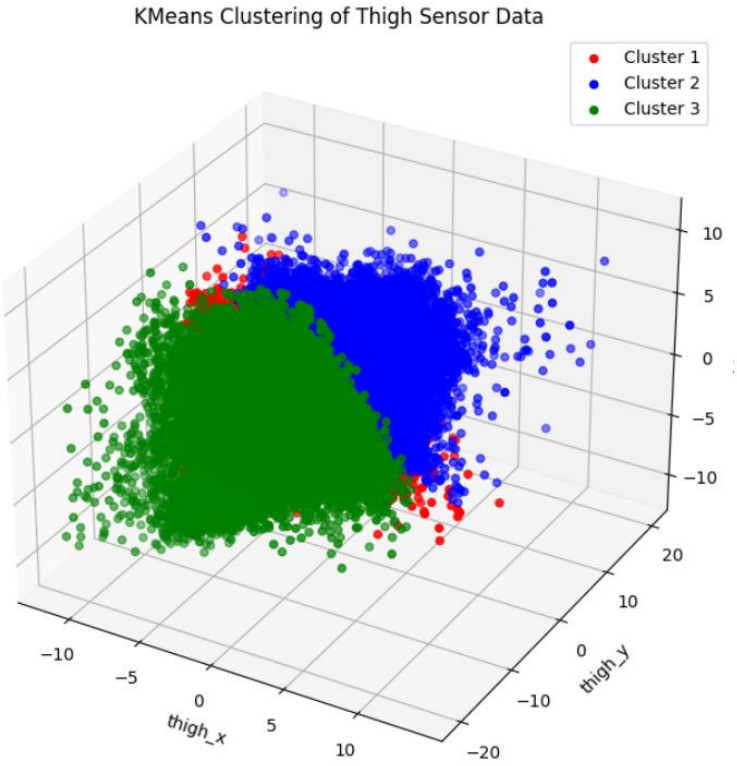


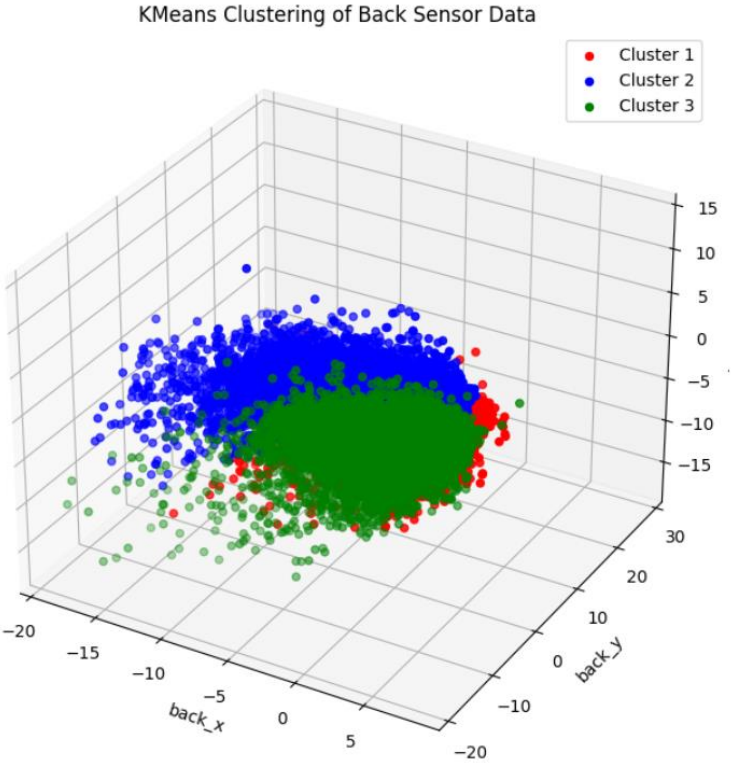
Παρατηρούμε ότι δεν είναι τόσο ξεκάθαρο το σημείο όπου η πτώση του **SSE** γίνεται αισθητά λιγότερο απότομη, ωστόσο παρατηρούμε ότι η πτώση αρχίζει και συμβαίνει με μικρότερο ρυθμό όταν k = 3, 4. Εμείς επιλέξαμε να χρησιμοποιήσουμε k = 3.

Επίσης, βλέπουμε τα κεντροειδή κάθε συστάδας που δημιουργήθηκε καθώς και την διανομή των συμμετεχόντων σε αυτές:

```
Cluster Centers:
[-0.38195496 -0.07802592  0.1923045  -0.84522871 -0.12633215 -0.71296328]
[ 1.90404647  1.2372765   1.07283398  0.62173226 -0.82318124 -1.21469077]
[ 0.08235243 -0.08319561 -0.28097775  0.6057119   0.19854412  0.71597493]

Placement of Participants in Clusters
Cluster 0:
  S009
  S010
  S028
  S026
  S029
  S023
  S025
  S027
Cluster 1:
  S024
Cluster 2:
  S017
  S016
  S021
  S015
  S008
  S013
  S019
  S020
  S022
  S012
  S014
  S006
  S018
```

Επιπρόσθετα, βλέπουμε σε τρισδιάστατη απεικόνιση τις συστάδες των δεδομένων που δημιουργούνται ξεχωριστά από τα δεδομένα του κάθε αισθητήρα και τη διανομή των συμμετεχόντων πάλι σε αυτές τις συστάδες. Σκοπός μας ήταν να μελετήσουμε αν τα δεδομένα μας έχουν μεγάλη επικάλυψη για να αποφανθούμε αν χρειαζόμαστε εκτός από σφαιρικές συστάδες και ελλειπτικές.






```
Cluster Centers:
[-0.6217236   0.07216152 -0.81567132]
[0.59188653 0.21971581 0.73072849]
[-0.94900892 -2.30729703 -0.86118157]

Placement of Participants in Clusters
Cluster 0:
  S009
  S010
  S028
  S023
  S025
  S024
Cluster 1:
  S017
  S016
  S021
  S015
  S008
  S013
  S019
  S020
  S022
  S012
  S014
  S006
  S018
Cluster 2:
  S026
  S029
  S027
```

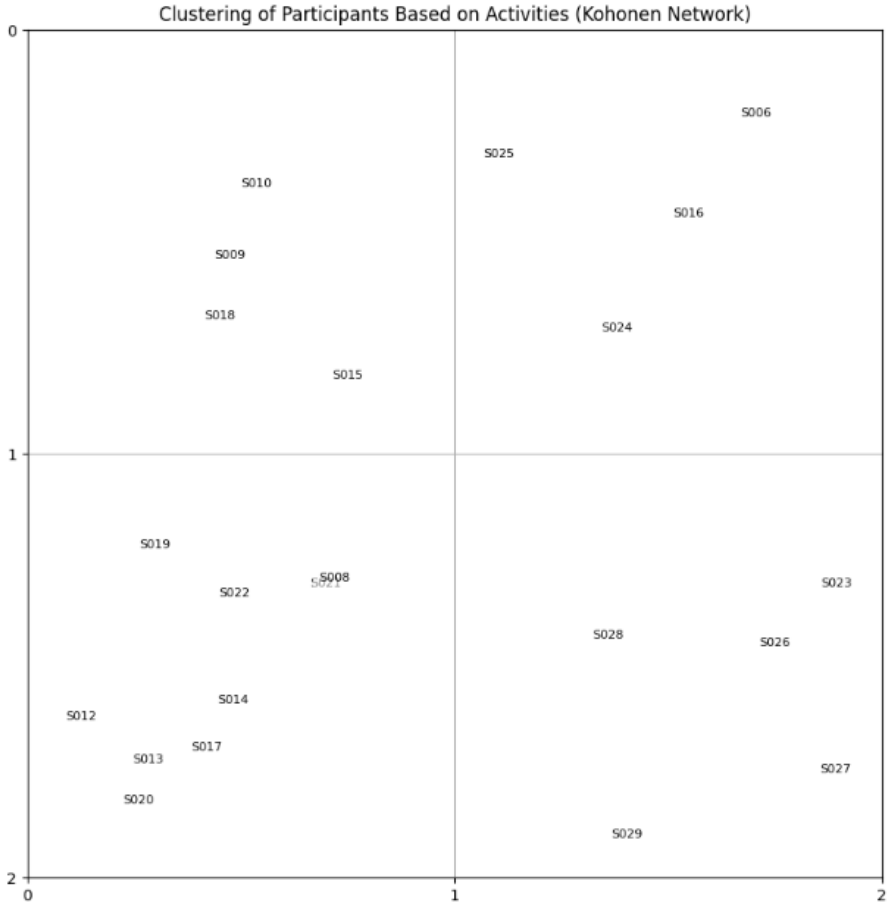
```
Cluster Centers:
[ 0.63199244  0.3980111  -1.16813884]
[-0.28774754 -0.00358566  0.41384503]
[ 1.950978   -3.46336172 -0.48884879]

Placement of Participants in Clusters
Cluster 0:
  S016
  S015
  S026
  S008
  S019
  S020
  S006
  S024
Cluster 1:
  S017
  S009
  S021
  S010
  S028
  S029
  S013
  S023
  S022
  S012
  S014
  S027
  S018
Cluster 2:
  S025
```

Συμπεράσματα:

- Όσον αφορά τις συστάδες παρατηρούμε ότι στον μηρό φαίνονται να είναι πιο διακριτές, με τα δεδομένα να έχουν μια πιο καθαρή γεωμετρική διάταξη. Στη πλάτη από την άλλη, οι συστάδες είναι λίγο πιο αλληλοεπικαλυπτόμενες, αλλά ακόμα διακρίνονται οι τρεις ομάδες με τα δεδομένα να παρουσιάζουν μια μεγαλύτερη διασπορά των σημείων. Υπάρχουν συμμετέχοντες που τοποθετούνται στα ίδια clusters και στις δύο προσεγγίσεις συσταδοποίησης, κάτι που δείχνει κάποια συνέπεια. Για παράδειγμα, οι συμμετέχοντες **S009**, **S010**, **S028** είναι στο ίδιο cluster και στις δύο προσεγγίσεις.
- Όσον αφορά την επικάλυψη των δεδομένων παρατηρούμε ότι υπάρχει έντονη επικάλυψη και άρα είναι συνετό να αξιοποιήσουμε και ελλειπτική συσταδοποίηση. Κάτι τέτοιο επιτυγχάνει ο αλγόριθμος **Gaussian Mixture**.

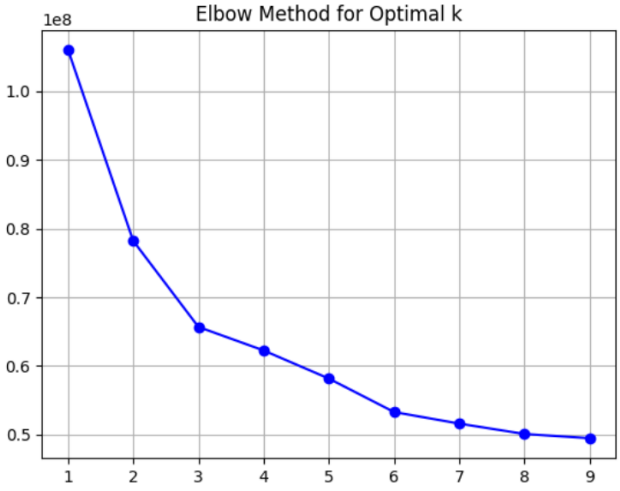
4.2.1 Kohonen Network (Self-Organizing Map)

Πιο κάτω βλέπουμε στον χάρτη την διανομή των συμμετεχόντων στις 4 συστάδες που δημιουργήθηκαν:




```
Placement of Participants in
Clusters
Cluster 0:
  S009
  S010
  S015
  S018
Cluster 1:
  S017
  S021
  S008
  S013
  S019
  S020
  S022
  S012
  S014
Cluster 2:
  S016
  S025
  S006
  S024
Cluster 3:
  S028
  S026
  S029
  S023
  S027
```

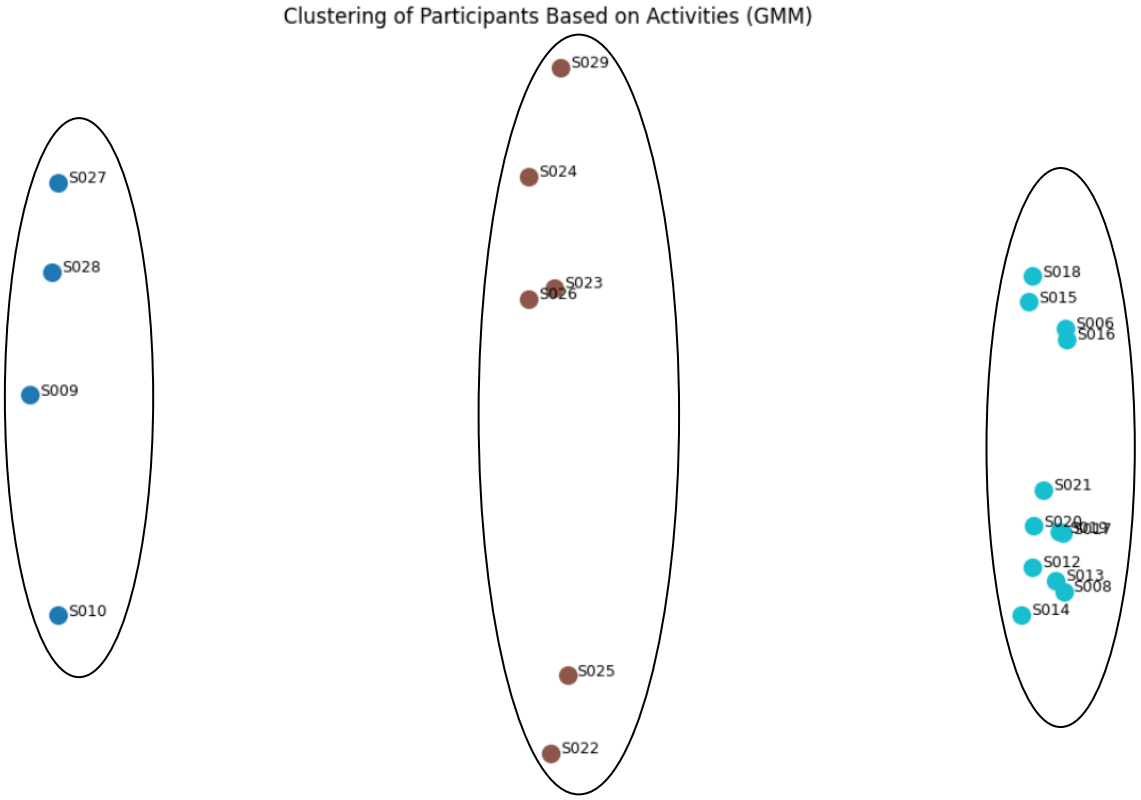
4.2.2 Gaussian Mixture



Πιο δίπλα βλέπουμε το γράφημα που προέκυψε εφαρμόζοντας το **Elbow Method w/ BIC** προκειμένου να καθορίσουμε το ιδανικό πλήθος συστάδων:

Αυτή τη φορά παρατηρούμε ότι είναι πιο εμφανής η μείωση του ρυθμού πτώσης όταν $k = 3$.

Επίσης, βλέπουμε τα κεντροειδή κάθε συστάδας που δημιουργήθηκε καθώς και την διανομή των συμμετεχόντων σε αυτές:




```
Cluster Centers (GMM):
Cluster 1: [-0.19555186 -0.01402299  0.30042484 -0.60045127 -0.01583482 -0.65839864]
Cluster 2: [ 0.30136117 -0.15948463  0.06424919 -0.33767086 -0.14190394 -0.53312885]
Cluster 3: [-0.10534744  0.12388885 -0.22083051  0.59325248  0.11217892  0.76872057]

Placement of Participants in Clusters (GMM):
Cluster 0:
  S009
  S010
  S028
  S027
Cluster 1:
  S026
  S029
  S023
  S025
  S022
  S024
Cluster 2:
  S017
  S016
  S021
  S015
  S008
  S013
  S019
  S020
  S012
  S014
  S006
  S018
```

Συμπεράσματα:

Παρατηρώντας τη διανομή των συμμετεχόντων στις παραχθείσες συστάδες μπορούμε να συμπεράνουμε τα εξής:

- Κοινοί συμμετέχοντες μεταξύ των τριών αλγορίθμων στο **Cluster 0**: **S009**, **S010**.
- Κοινοί συμμετέχοντες μεταξύ των τριών αλγορίθμων στο **Cluster 1**: **S024**.
- Κοινοί συμμετέχοντες μεταξύ των τριών αλγορίθμων στο **Cluster 2**: **S016**, **S006**.

Συνεπώς, οι αλγόριθμοι **K-Means** και **GMM** τοποθετούν τους περισσότερους κοινούς συμμετέχοντες σε κάθε συστάδα, άρα μπορούμε να καταλήξουμε στο συμπέρασμα ότι λειτουργούν πιο αποδοτικά.

K-Means

```
Silhouette Score: 0.2219287327166749
```

Kohonen Network

```
Silhouette Score: 0.11829160548476043
```

Gaussian Mixture

```
Silhouette Score for GMM: 0.2837100937215892
```

Συμπεράσματα:

Παραπάνω βλέπουμε το **Silhouette Score** το οποίο μας δίνει μια καλή εικόνα του πόσο αποτελεσματική ήταν η συσταδοποίηση που πραγματοποιήσαμε. Όσο πιο κοντά στο 0 είναι η τιμή του, τόσο περισσότερο βρίσκονται σε επικάλυψη οι παρατηρήσεις μέσα στα clusters, ενώ όσο πιο κοντά είναι στο 1 τόσο καλύτερη είναι η συσταδοποίηση. Με βάση αυτό αυτές συμπεραίνουμε τα εξής:

- Το υψηλότερο **Silhouette Score** (0.2837) δείχνει ότι ο **GMM** έχει τη καλύτερη απόδοση όσον αφορά τη διακριτότητα και την ομοιογένεια των clusters. Αυτό σημαίνει ότι οι παρατηρήσεις μέσα σε κάθε cluster είναι πιο ομοιογενείς και τα clusters είναι καλύτερα διαχωρισμένα.
- Το **Silhouette Score** του **K-Means** (0.2219) είναι χαμηλότερο από το **GMM** αλλά υψηλότερο από το **Kohonen Network**. Αυτό υποδηλώνει ότι τα clusters του **K-Means** είναι λιγότερο διακριτά και ομοιογενή από αυτά του **GMM**, αλλά καλύτερα από αυτά του **Kohonen Network**.
- Το χαμηλότερο Silhouette Score (0.1183) δείχνει ότι το **Kohonen Network** έχει τη χειρότερη απόδοση μεταξύ των τριών αλγορίθμων. Οι παρατηρήσεις είναι λιγότερο ομοιογενείς μέσα σε κάθε cluster και άρα τα clusters είναι λιγότερο διακριτά.

Συνολικά, το μέτριο **Silhouette Score** οφείλεται στο ότι οι παρατηρήσεις στα clusters είναι παρόμοιες, με αποτέλεσμα να υπάρχει επικάλυψη όπως αναφέραμε και πιο πάνω. Αυτό υποδηλώνει ότι τα χαρακτηριστικά που χρησιμοποιούνται για την συσταδοποίηση ίσως δεν διαχωρίζουν καλά τις παρατηρήσεις. Βέβαια ο αλγόριθμος **Gaussian Mixture** διαχειρίζεται καλύτερα τα δεδομένα συγκριτικά με τους υπόλοιπους αλγορίθμους, καθώς με τη παραγωγή ελλειπτικών συστάδων επιτυγχάνει να μετριάσει κάπως την υπάρχουσα επικάλυψη.

Σε κάθε περίπτωση ωστόσο, καταλήγουμε στο συμπέρασμα ότι οι αλγόριθμοι **K-Means** και **GMM** λειτουργούν αποδοτικότερα με κοντινή απόδοση ο ένας με τον άλλο.

5 Παράρτημα

Παρακάτω παρατίθεται ο σύνδεσμος με το Github Repository όπου περιέχεται ο πλήρης κώδικας για κάθε ερώτημα σε Jupyter Notebook μαζί με το απαραίτητο Documentation:

https://github.com/miltiadiss/Data-Mining