Εργασία Εξαμήνου:
«Ανάλυση και Διαχείριση Μεγάλων και Πολυδιάστατων Δεδομένων 2023-2024»

Ιωάννης Τσάμπρας ΣΜΗΝ 1066584

Δομή Εργασίας:

1. Προεπεξεργασία Clinical
    a. Αριθμιτικοποίηση
    b. Καθαρισμός
    c. Μέθοδοι Συμπλήρωσης Κενών
2. Κατηγοριοποίηση Clinical
    a. Μέθδοι
    b. Αποτελέσματα
3. Προεπεξεργασία Beacons
    a. Κανονικοποίηση
    b. Εξαγωγή Χαρακτηριστηκών
4. Συνδιασμός Clinical και Beacons Features
5. Συσταδοποίηση τελικού Dataset
    a. Συσταδοποίηση
    b. PCA
    c. Αποτελέσματα

1. Προεπεξεργασία Clinical

a. Αριθμιτικοποίηση

Αρχικά φορτώνουμε το .csv αρχείο στη μνήμη ως δισιδιάστατό array. Παρατήρησα πως το αρχείο που μας δώθηκε χρησιμοποιούσε encoding “utf-8-sig». Ύστερα πραγματοποιούμε έλεγχο δομής (αν όλες οι γραμμές έχουν ίδιο αριθμό στηλών) και ξεχωρίζουμε header από body.

```
def load_to_memory(original_file_path):
    # Initialize an empty list to store the data
    data_2d_list = []

    # Open the CSV file
    with open(original_file_path, 'r', encoding="utf-8-sig", newline='') as csvfile:
        # Create a CSV reader object
        csv_reader = csv.reader(csvfile,delimiter=";")

        # Iterate over each row in the CSV file
        for row in csv_reader:
            # Append each row to the 2D list
            data_2d_list.append(row)


    return data_2d_list # Return the resulting 2D list representing the dataset


### Init
original_file_path = 'Project_Desc/clinical_dataset.csv'
#original_file_path = 'tasks/Part_A/task_1/test.csv'
dataset=load_to_memory(original_file_path=original_file_path)
#print(dataset[1][0])
header=dataset[0]
entries=dataset[1:]
###

### Diagnostics
structure_test_result=check_csv_structure(header=header,entries=entries)
if (structure_test_result!=True):
    print(structure_test_result)
    exit(1)
###
```

Η πρώτη μας σημαντική δοκιμασία είναι η αριθμιτικοποίηση των δεδομένων. Για αυτό τον σκοπό δημιουργούμε μια δομή πίνακα από «set» , ένα για κάθε στήλη, και ελέγχοντας κάθε γραμμή προσθέτουμε τις τιμές του κάθε κελιού στο set της αντίστοιχης στήλης.

Αφού προσπελάσουμε όλο το dataset έχουμε για κάθε πεδίο το σχετικό σύνολο τιμών.

```
def gather_field_ranges(head,entries):
    sets=[set() for _ in range(len(head))]
    for entry in entries:
        for i,value in enumerate(entry):
            sets[i].add(value)
    return sets #list of sets containing possible values for each field
```

Έχοντας τα σύνολα τιμών διερευνούμε ποιά πεδία είναι αριθμητικά και ποιά όχι καθώς απαιτούν διαφορετική προεπεξεργασία. Εδώ είναι σημαντικό να πούμε πως αγνοούμε κενούς χαρακτήρες, οπότε αριθμητικές στήλες με κενά δεν θεωρούνται μη αριθμητικές.

```
#find which fields are non numerical
non_num_fields=[]
for i,range_of_field in enumerate(sets):
    if has_non_digit_elements(range_of_field):
        #print(header[i],range_of_field)
        #print(i,header[i])
        non_num_fields.append(i) #list of indexes corresponding to columns
```

```
def has_non_digit_elements(str_list):
    for element in str_list:
        if not is_number(element) and element.strip()!="":
            return True
    return False
```

Αφού εκτυπώσουμε τα μη αριθμητικά πεδία και τα σύνολα τιμών τους κατασκευάζουμε με το χέρι το παρακάτω λεξικό που αντιστοιχεί κάθε πιθανή σωστή είσοδο στην αντίστοιχη αριθμιτική τιμή της.

```
nom_dicts=[ #list indexes correspond to columns
    {"Frail":2,"Non frail":0,"Pre-frail":1},
    {"M":-1,"F":+1},
    {"No":0,"Yes":1},
    {'Sees well':0, 'Sees moderately':1, 'Sees poorly':2},
    {'Hears poorly':2, 'Hears well':0, 'Hears moderately':1},
    {"No":0,"Yes":1},
    {'<5 sec':0,'>5 sec':1},
    {'FALSE':0, 'TRUE':1},
    {"No":0,"Yes":1},
    {"No":0,"Yes":1},
    {"No":0,"Yes":1},
    {"No":0,"Yes":1},
    {'Permanent sleep problem':2, 'Occasional sleep problem':1, 'No sleep problem':0},
    {"No":0,"Yes":1},
    {"No":0,"Yes":1},
    {"No":0,"Yes":1},
    {"No":0,"Yes":1},
    {'2 - Bad':1, '4 - Good':3, '5 - Excellent':4, '3 - Medium':2, '1 - Very bad':0},
    {'2 - A little worse':1, '3 - About the same':2, '4 - A little better':3, '5 - A lot better':4, '1 - A lot worse':0},
    {'No':0, '< 2 h per week':1, '> 5 h per week':3, '> 2 h and < 5 h per week':2},
    {'Never smoked':0, 'Current smoker':2, 'Past smoker (stopped at least 6 months)':1},
]
```

Ύστερα προσπελνούμε το dataset εφαρμόζοντας το λεξικό για την αριθμητικοποίηση, όσες τιμές δεν συναύδουν με το πεδίο ορισμού του λεξικού (π.χ. κενά κελιά) αντικαθίστονται με “None”.

```
#replace values
new_entries=[]
for entry in entries:
    new_entries.append([])
    for i,field in enumerate(entry):
        if i in non_num_fields:
            try:
                new_entries[-1].append(nom_dicts[non_num_fields.index(i)][field]) #if existing value in dict then translate it according to dict
            except:
                new_entries[-1].append(None) #if existing value not in dict replace with None
        else:
            new_entries[-1].append(field) #if no nominalisation needed just add old value

###new dataset nominalised
```

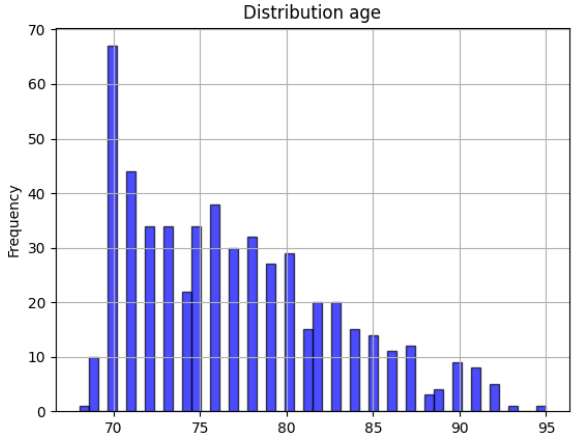
b. Καθαρισμός

Για τις αριθμητικές στήλες πρέπει να προχωρήσουμε σε καθαρισμό λανθασμένων τιμών, τις σκέτες κενές εισόδους τις αντικαθιστούμε με None. Κατά τη γνώμη μου είναι λάθος να διαγράψουμε εντελώς όλες τις «999» εγγραφές καθώς σε πολλά πεδία το «999» μπορεί να χρησημοποιείται από τους ερευνητές για να δηλώσουν «άπειρο» χρόνο.

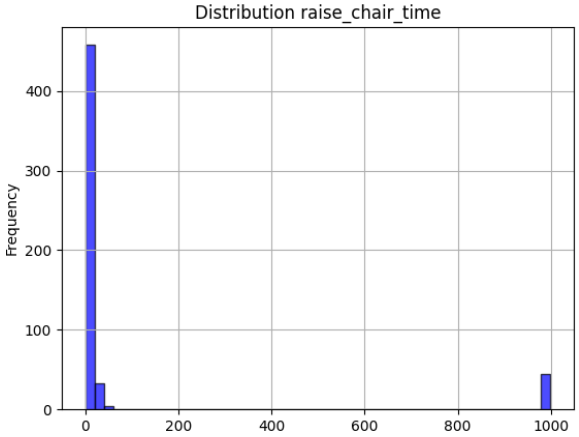
Π.χ. στην μέτρηση χρόνου σηκώματος, το 999 χρησημοποιείται αρκετά συχνά για να είναι απλό λάθος και μάλλον υποδεικνύει αδυναμία ολοκλήρωσης του τέστ (ο συμμετέχον μπορεί να μην μπορεί να σηκωθεί από το κρεββάτι). Αυτή η πληροφορία δεν πρέπει να χαθεί οπότε αντί της πλήρης διαγραφής της τιμής την αντικαθιστώ με ένα ανώτατο όριο που προκύπτει από τις υπόλοιπες μη «ακραίες» εγγραφές.

Για να αποκτήσουμε intuition για τα δεδομένα μας πριν προχωρήσουμε σε αλλαγές εκτελούμε γραφικές αναπαραστάσεις των κατανομών κάθε αριθμητικού πεδίου και ελέγχουμε για ακραίες τιμές (όχι μόνο τις «999») ενώ ταυτόχρονα με βάση τον αριθμό εμφάνισης εγγραφών «999» και την κατανομή των δεδομένων αποφασίζουμε τη διαγραφή ή εφαρμογή ανώτατου ορίου.

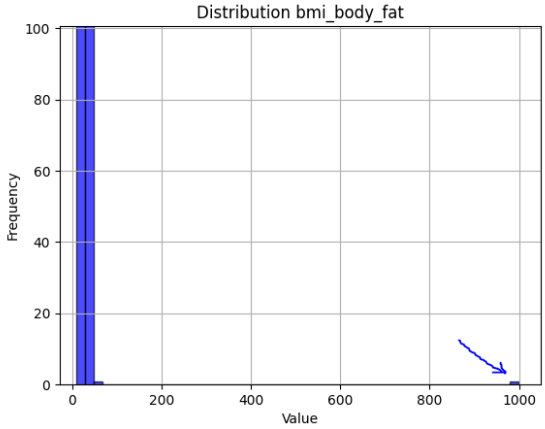
Μερικές από τις κατανομές φαίνονται παρακάτω:



Παράδειγμα Καλής Κατανομής



Παράδειγμα Προβληματικής Κατανομής με Σημασιολογικής Βαρύτητας Λάθη



Παράδειγμα Προβληματικής Κατανομής με Καθαρά Λάθη

Για τον καθαρισμό ή την εφαρμογή ορίου στις αριθμητικές κατανομές κατασκευάστηκε η παρακάτω λίστα κανόνων όπου :

- None -> Καμία Αλλαγή
- ‘<T’ -> Διαγραφή όσων κελιών είναι μεγαλύτερα του T
- ‘>T’ -> Διαγραφή όσων κελιών είναι μικρότερα του T
- ‘x<T’ -> Αλλαγή όσων κελιών είναι μεγαλύτερα του T σε T
- ‘x>T’ -> Αλλαγή όσων κελιών είναι μικρότερα του T σε T

```
# from visualy inspecting the distributions plotted we establish the following thresholds and commen
# the added "x" in front indicates that we will be capping the values because although outliers they
thresholds=[
    None,
    None,
    None, ###but there are propably real but extreme outliers, maybe need to add log scale
    "x<15", ### maybe 999 represents living in hospital, 2 entries
    None,
    "x<55", ###maybe the 999 represents inability to lift, 40+ entries are not mistakes
    "x<65", ###maybe the 999 represents inability to get up, ~9 entries
    "x<50",  ###maybe the 999 represents inability to run, ~6 entries
    "x<7",  ###the 999 is propably just a mistake or inability to move in general, ~3 entries
    "<5",  ###the 999 is propably just wrong/ clearly eronius
    "<45",  ###the ~900 entry is physically impossible so clearly eronius
    "<65", ###the 999 entry is clearly eronius
    None,   ###there is one outlier but it is propably real
    ">10",   ###the ~ -400 entry is clearly eronius
    None,
    None,
    None,
    None,
    None,
    None,
    "<35",  ###don't know how to interpet those, propably eronius but 4 entries
    "<200", ###don't know how to interpet those, propably eronius 2 entries
    None,
    "x<200", ###maybe not eronius , it may signify something 10+ entries. there are also outliers
    "x<200", ###maybe not eronius , it may signify something 10+ entries. there are also outliers
    None,
    None,
    None,
    "x<65", ###maybe grandad is alcoholic
    None,
    None,
    None,
    None,
    None
]
```

Ύστερα προχωρούμε σε εφαρμογή των κανόνων μας.

```python
#cap or remove weird values and keep the rest as they are
def clean_eronius_values(thresholds,non_num_fields,old_entries):
    new_entries=[]
    #print(len(old_entries[0]))
    for entry in old_entries:
        new_entries.append([])
        thres_index=-1
        for i,field in enumerate(entry):
            if i not in non_num_fields:
                thres_index+=1
                threshold=thresholds[thres_index]
                try:
                    test=float(field)
                except:
                    new_entries[-1].append(None) #if non numerical value then set it to None and pass
                    continue


                if threshold!= None:
                    if "x" in threshold:
                        threshold=threshold.replace("x","")
                        #mode="cap"
                        if "<" in threshold:
                            threshold=float(threshold.replace("<",""))
                            new_entries[-1].append(min(float(field),threshold))
                        else:
                            threshold=float(threshold.replace(">",""))
                            new_entries[-1].append(max(float(field),threshold))
                    else:
                        #mode="remove"
                        if "<" in threshold:
                            threshold=float(threshold.replace("<",""))
                            new_entries[-1].append(float(field) if float(field)<threshold else threshold)
                        else:
                            threshold=float(threshold.replace(">",""))
                            new_entries[-1].append(float(field) if float(field)>threshold else threshold)
                else:
                    new_entries[-1].append(field) #if clearing is not needed just add old value


            else:
                new_entries[-1].append(field) #if clearing is not needed just add old value
    return new_entries

new_entries=clean_eronius_values(thresholds,non_num_fields,new_entries)
### entries cleaned and nominalised, filling empty spots is all it is left
```

Και τέλος σώζουμε το καθαρισμένο και αριθμητικοποιημένο μας dataset σε νέο csv.

```python
#save cleaned dataset to csv
def save_csv(header, data, filename):
    with open(filename, 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)

        # Write the header
        csv_writer.writerow(header)

        # Write the data, replacing None with an empty string
        for row in data:
            csv_writer.writerow(['' if cell is None else cell for cell in row])


save_csv(header=header,data=new_entries,filename="tasks/Part_A/task_1/cleaned.csv")
```

c. Μέθοδοι Συμπλήρωσης Κενών

Στα πλαίσια της εργασίας αξιοποιήθηκαν **δύο μέθοδοι για την συμπλήρωση** των κενών οι οποίοι συγκρίνονται και κατά την κατηγοριοποίηση.

Ο πρώτος είναι η πιο naive προσσέγγιση όπου γεμίζουμε κάθε κελί με την **μέση τιμή** της στήλης. Τα αποτελέσματα αυτής της μεθόδου βρίσκονται στο “cleanedANDfilled2.csv”.

Ο δεύτερος τρόπος **είναι η εκπαίδευση Ν feedforward νευρωνικών δικτύων**, όπου Ν το πλήθως των στηλών με έστω και μια κενή εγγραφή, για την **πρόβλεψη των κενών τιμών** με βάση τα περιεχόμενα της γραμμής.

Η εκπαίεδευση των νευρωνικών δικτύων γίνεται πάνω στις γραμμές που δεν έχουν κανένα κενό εξαρχής (περίπου οι μισές εγγραφές) και αν μια εγγραφή έχει πάνω από μία κενή τιμή τότε οι κενές τιμές της συμπληρώνονται πρώτα με βάση την μέθοδο μέσων τιμών και μετά εφαρμόζονται τα νευρωνικά για την διόρθωση των τιμών της.

Η εκπαίδευση για κάθε νευρωνικό γίνεται μία φορά και αποθηκεύεται στη μνήμη σε περίπτωση που προκύψει πάλι κενό ίδιας στήλης.

```python
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

def train_neural_network(dataset, target_column_index, epochs=2000, batch_size=64):
    # Extract features and target variable
    features = np.array([row[:target_column_index] + row[target_column_index+1:] for row in dataset])
    target = np.array([row[target_column_index] for row in dataset])

    # Split the dataset into training and testing sets
    features_train, features_test, target_train, target_test = train_test_split(features, target, test_size=0.2, random_state=42)

    # Standardize the features
    scaler = StandardScaler()
    features_train_scaled = scaler.fit_transform(features_train)
    features_test_scaled = scaler.transform(features_test)

    # Build a simple neural network model
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(8192, activation='relu', input_shape=(features_train_scaled.shape[1],)),
        tf.keras.layers.Dense(1)
    ])

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Train the model
    model.fit(features_train_scaled, target_train, epochs=epochs, batch_size=batch_size, validation_split=0.2, verbose=0)

    # Evaluate the model on the test set
    loss = model.evaluate(features_test_scaled, target_test, verbose=0)
    print(f'Test Loss: {loss}')

    return model, scaler
```

```python
def predict_cell_value(model, scaler, row, target_column_index):
    # Extract features for prediction
    features = np.array(row[:target_column_index] + row[target_column_index+1:])

    # Standardize the features using the same scaler used during training
    features_scaled = scaler.transform([features])

    # Make the prediction
    prediction = model.predict(features_scaled)[0][0]
    if target_column_index in non_num_fields:
        return int(prediction)
    return prediction


def fix_rows(rows):
    fixed_rows=[]
    from collections import defaultdict

    models = defaultdict(lambda: None)

    for i,row in enumerate(rows):
        print("Fixing row ",i)
        #my_row=[1008,2,1,74,10,11.0,0,1,1,1,1,55.0,None,46.0,15.0,0,1,1,1,3.0,0.0,33.95555556,32.7,103,51.4172,11,17,0,0,25,5,3.6,0,
        none_indices = [index for index, value in enumerate(row) if value is None] #get indexes where cell is empty
        semi_fixed_row=row.copy()
        for index_i in none_indices: #we fill empty values with avg
            semi_fixed_row[index_i]=avg_values[index_i]

        for index_i in none_indices:
            prediction_column_index=index_i
            if models[index_i]==None:
                print(f"model for column {index_i} is being created")
                trained_model, scaler = train_neural_network(full_entries, target_column_index=prediction_column_index)
                models[index_i]=[trained_model,scaler]
            else:
                print(f"model for column {index_i} is cached")



            # Now, you can use the trained model and scaler to make predictions:
            trained_model,scaler=models[index_i]
            predicted_value = predict_cell_value(trained_model, scaler, semi_fixed_row, target_column_index=prediction_column_index)
            semi_fixed_row[index_i]=predicted_value
        fixed_row=semi_fixed_row.copy()
        fixed_rows.append(fixed_row)
            #print(f'Predicted value: {predicted_value}, Ground Truth: {my_row[prediction_column_index]}')
    return fixed_rows
fixed=fix_rows(problematic_entries)
#for i in fixed:
    #print(i)

final_dataset=fixed+full_entries
final_dataset.sort(key=lambda x: float(x[0]))
```

Οι υπερπαράμετροι του νευρωνικού δικτύου και της εκπαίδευσής του αποκτήθηκαν με βάση πειραμματισμό. Οι στήλες με ακέραιες τιμές στρογγυλοπούνται στην κοντινότερη, της πρόβλεψης, ακέραια τιμή.

Όπως φαίνεται στην παρακάτω φωτογραφία το πρόγραμμα κάνει cache τα μοντέλα για επαναχρησιμοποίηση σε μετεγενέστερες εγγραφές.

```
Test Loss: 0.23286639153957367
1/1 [==============================] - 0s 60ms/step
Fixing row  1
model for column 12 is cached
1/1 [==============================] - 0s 17ms/step
model for column 16 is being created
Test Loss: 0.08290605247020721
1/1 [==============================] - 0s 36ms/step
model for column 22 is being created
Test Loss: 33.13172912597656
1/1 [==============================] - 0s 39ms/step
model for column 24 is being created
```

Μέρος της εκτέλσης του complete.py

Τέλος το «γεμισμένο» dataset σώζεται στο αρχείο “cleanedANDfilled.csv”.

2. Κατηγοριοποίση Clinical

a. Μέθοδοι

Για την κατηγοριοποίηση των ασθενών αξιοποιήθηκαν δύο μέθοδοι:

- KNN
- Feed Forward NN

Για το validation accuracy του νευρωνικού δικτύου εκτελέστηκαν 16 προπονήσεις και κρατήσαμε τον μέσο όρο. Επιπλέον τα αποτελέσματα συγκρίνονται και με τα δύο dataset “cleanedANDfilled.csv” και “cleanedANDfilled2.csv” που παρήχθησαν από τις δύο μεθόδους συμπλήρωσης κενών.

Λόγω της αδυναμίας μου να αξιοποιήσω GPU κατά την εκπαίδευση εφάρμοσα παράλληλη εκτέλση των πολλαπλών προπονήσεων (για την απόκτηση μέσου όρου για τα validation accuracy) σε CPU.

```python
def run_parallel(times): #run the model training multiple times to evaluate accuracy
    # Number of times to run the function
    num_runs = times

    # Create a ThreadPoolExecutor with the desired number of workers
    with concurrent.futures.ThreadPoolExecutor(max_workers=num_runs) as executor:
        # Submit the function for execution in parallel
        futures = [executor.submit(model_eval) for _ in range(num_runs)]

        # Collect the results as they become available
        results = [future.result() for future in concurrent.futures.as_completed(futures)]

    return results

if __name__ == "__main__":
    parallel_results = run_parallel(16)
    print(parallel_results)
    print(sum(parallel_results)/16)
```

Parallel Training

```python
#now lets use a simple feed forward neural network
def model_eval():
    import numpy as np
    import tensorflow as tf
    #tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import LabelEncoder
    from tensorflow.keras.utils import to_categorical
    import matplotlib.pyplot as plt

    # Assuming 'dataset' is your 2D list and 'desired_outputs' is your 1D list
    # X is your feature matrix, and y is your target variable
    features = classifier_input
    labels = classifier_output


    # Convert your data to NumPy arrays
    features = np.array(features)
    labels = np.array(labels)

    # Encode labels to integers
    label_encoder = LabelEncoder()
    encoded_labels = label_encoder.fit_transform(labels)

    # Convert integers to one-hot encoding
    one_hot_labels = to_categorical(encoded_labels, num_classes=3)

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(features, one_hot_labels, test_size=0.1, random_state=42)

    # Build a simple neural network model using TensorFlow for multi-class classification
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(1024, activation='relu', input_shape=(X_train.shape[1],)),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(3, activation='softmax')  # Output layer with softmax for multi-class classification
    ])

    # Compile the model
    custom_optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)


    model.compile(optimizer=custom_optimizer, loss='categorical_crossentropy', metrics=['accuracy'])


    # Store the training history
    history = model.fit(X_train, y_train, epochs=2000, batch_size=64, validation_data=(X_test, y_test),verbose=0)

    # Evaluate the model on the test set
    loss, accuracy = model.evaluate(X_test, y_test)
    print(f'Test Loss: {loss}')
    print(f'Test Accuracy: {accuracy}')
    return accuracy

import concurrent.futures
```

Neural Network

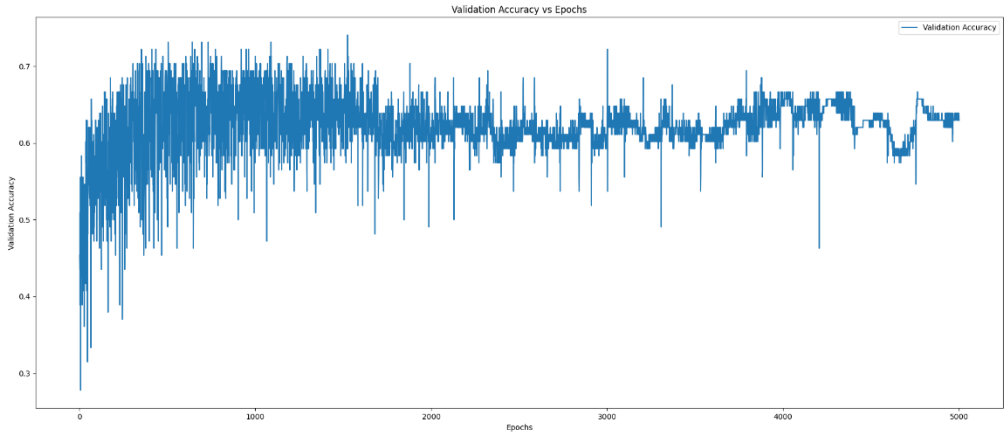
Για την εκπαίδευση αφαιρέθηκαν οι στήλες «0,1, 9, 10, 16, 17, 18» του dataset.

```
excluded_columns = [0,1, 9, 10, 16, 17, 18]


classifier_input=[]
classifier_output=[]
for entry in dataset[1:]:#exclude the header
    #print(entry)
    classifier_input.append([])
    for i,column in enumerate(entry):
        if i not in excluded_columns:
            classifier_input[-1].append(float(column))
    classifier_output.append(float(entry[1]))


from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Assuming 'dataset' is your 2D list and 'desired_outputs' is your 1D list
# X is your feature matrix, and y is your target variable
X = classifier_input
y = classifier_output

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)


acc=[]
for i in range(1,40):

    # Initialize the KNN classifier
    knn_classifier = KNeighborsClassifier(n_neighbors=i)  # You can adjust the number of neighbors (k) as needed

    # Train the classifier on the training data
    knn_classifier.fit(X_train, y_train)

    # Make predictions on the test set
    predictions = knn_classifier.predict(X_test)

    # Evaluate the accuracy of the classifier
    accuracy = accuracy_score(y_test, predictions)
    acc.append(accuracy)
    #print(f'Accuracy: {accuracy * 100:.2f}% for {i} nearest neighboors')
print(sum(acc)/len(acc))
print(max(acc))
```

KNN

Για τους παράμετρους του νευρωνικού έγιναν πολλαπλές δοκιμές και κατέληξα σε :

- 3 layers (1024,128,3)
- Learning rate 0.0001
- 2000 epochs
- Batchsize 64



Validation Accuracy through Epochs

b. Αποτελέσματα

Για τον KNN τα αποτελέσμτα δεν αλλάζουν, με βάση το dataset, με μέγιστο accuracy 0.66 και μέσο 0.59 για τους πρώτους 40 γείτονες.

Για το νευρωνικό δίκτυο πραγματοποιήθηκαν αρχικά 8 και ύστερα επιπλέον 16 trainings με κάθε dataset και από αυτά προέκυψαν οι παρακάτω validation accuracies:

```
#in the 8 runs
#with predictor dataset 0.7337962910532951 [0.7
#with simple averages   0.6296296268701553 [0.6

#in the 16 runs
#with predictor dataset 0.7303240746259689 [0.7
#with simple averages   0.6238425895571709 [0.5
```

Όπου φαίνεται η **αύξηση της ακρίβειας (κατά 11%) με τη χρήση του dataset όπου οι τιμές συμπληρώθηκαν με τη χρήση νευρωνικού δικτύου** έναντι απλών μέσω τιμών.

3. Προεπεξεργασία Beacons

a. Κανονικοποίηση

Αρχικά καθαρίζουμε τις εγγραφές από μη προβλεπόμενα IDs και ύστερα συλλέγουμε όλες τις διαφορετικές ονομασίες δωματίων.

```
### removing faulty lines
clean_entries=[]
for row in entries:
    if is_4digit_number(row[0]) and row[-1]!='':
        clean_entries.append(row)
    else:
        pass
        #print(row[0])
#print(len(clean_entries))
#print(len(entries))


participants={}
room_names=set()
for entry in clean_entries:
    try:
        participants[entry[0]].append(entry.copy())
    except:
        participants[entry[0]]=[]
        participants[entry[0]].append(entry.copy())
    room_names.add(entry[-1])
```

Ύστερα χειροκίνητα κατασκευάζουμε ένα λεξικό αντιστοίχησης όλων των διαφορετικών ονομασιών με τις πραγματικές ονομασίες που επιλέξαμε οι οποίες είναι 8:

- Bathroom
- Bedroom
- Diningroom
- Hall
- Kitchen
- Livingroom
- Office
- Outdoor

Το λεξικό φαίνεται στην παρακάτω εικόνα:

```
fix_dict={
    'Outdoor':'Outdoor',
'Bathroim':'Bathroom',
'Livingroon':'Livingroom',
'Livingroom1':'Livingroom',
'TV':'Livingroom',
'Garage':'Outdoor',
'Bathroom1':'Bathroom',
'Office1':'Office',
'DinnerRoom':'Diningroom',
'Bathroom-1':'Bathroom',
'Sitingroom':'Livingroom',
'Hall':'Hall',
'Washroom':'Bathroom',
'Livingroom':'Livingroom',
'Garden':'Outdoor',
'Baghroom':'Bathroom',
'Kitchen':'Kitchen',
'Bathroom':'Bathroom',
'Liningroom':'Livingroom',
'Guard':'Hall',
'One':'Livingroom',
'DiningRoom':'Diningroom',
'Bedroom':'Bedroom',
'SittingOver':'Livingroom',
'Kichen':'Kitchen',
'Bsthroom':'Bathroom',
'LivingRoom2':'Livingroom',
'Bedroom2':'Bedroom',
'Kithen':'Kitchen',
'three':'Bathroom',
'bedroom':'Bedroom',
'2ndRoom':'Bedroom',
'LuvingRoom':'Livingroom',
'LivibgRoom':'Livingroom',
'Pantry':'Kitchen',
'Office1st':'Office',
'LeavingRoom':'Livingroom',
'Office-2':'Office',
'livingroom':'Livingroom',
'SittingRoom':'Livingroom',
'Four':'Outdoor',
'Kitcheb':'Kitchen',
'Veranda':'Outdoor',
'DinningRoom':'Diningroom',
'Office2':'Office',
'ExitHall':'Hall',
'Two':'Bedroom',
'Chambre':'Hall',
'Sittingroom':'Livingroom',
'Kitvhen':'Kitchen',
'Leavingroom':'Livingroom',
'Bathroon':'Bathroom',
'Livingroom2':'Livingroom',
'Kitcen':'Kitchen',
'Workroom':'Office',
'Entrance':'Hall',
'Laundry':'Bathroom',
'Office':'Office',
'Sittinroom':'Livingroom',
'Box-1':'Livingroom',
'Living':'Livingroom',
'Storage':'Outdoor',
'K':'Kitchen',
'Bedroom1st':'Bedroom',
'Library':'Office',
'DinerRoom':'Diningroom',
'Leavivinroom':'Livingroom',
'LivingRoom':'Livingroom',
'Desk':'Office',
'Kiychen':'Kitchen',
'Box':'Livingroom',
'SeatingRoom':'Livingroom',
'Sittigroom':'Livingroom',
'T':'Diningroom',
'Bqthroom':'Bathroom',
'Kitchen2':'Kitchen',
'Luvingroom1':'Livingroom',
'Bedroom1':'Bedroom',
'Bedroom-1':'Bedroom',
'Dinerroom':'Diningroom',
'LaundryRoom':'Bathroom',
'kitchen':'Kitchen'
}
```

Ύστερα αλλάζουμε με τη χρήση του λεξικού τις εγγραφές με τα σωστά ονόματα δωματίων ενώ ταυτόχρονα συμπτύσουμε τις στήλες ημερομηνίας και ώρας και τις μετατρέπουμε σε UNIX time για να σιευκολύνουμε την επεξεργασία τους αργότερα.

```
#### fix
participants={}
room_names=set()
for entry_unfixed in clean_entries:
    entry=[0,0,0]
    entry[0]=entry_unfixed[0] ### leave id as is



    datetime_string=entry_unfixed[1]+entry_unfixed[2].replace(":","")  ### replace date and time fields to one unix time
    # Convert the string to a datetime object
    dt_object = datetime.strptime(datetime_string, "%Y%m%d%H%M%S")

    # Get the Unix time (timestamp)
    unix_time = int(dt_object.timestamp())
    entry[1]=unix_time


    entry[-1]=fix_dict[entry_unfixed[-1]]   ### fix room names
    try:
        participants[entry[0]].append(entry.copy())
    except:
        participants[entry[0]]=[]
        participants[entry[0]].append(entry.copy())
    room_names.add(entry[-1])


print(room_names) ### ceck new room name set
```

Έπειτα κάνουμε sort με βάση το datetime και το participant ID και αποθηκεύουμε το αποτέλεσμα.

```
for participant in sorted(list(participants.keys())):

    participants[participant].sort(key=lambda x: x[1]) ### sort entries for each partiucipant based on time



### save to csv
data=[]
header=["part_id","unix_timestamp","room"]
for participant in sorted(list(participants.keys())):
    for entry in participants[participant]:
        data.append(entry)
def save_csv(header, data, filename):
    with open(filename, 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)

        # Write the header
        csv_writer.writerow(header)

        # Write the data, replacing None with an empty string
        for row in data:
            csv_writer.writerow(['' if cell is None else cell for cell in row])

save_csv(header=header,data=data,filename="tasks/Part_B/fixed.csv")
```

b. Εξαγωγή Χαρακτηριστηκών

Για τον υπολογισμό των χρόνων διαμονής σε κάθε δωμάτιο ορίζουμε τις 3 παρακάτω συναρτήσεις γαι να μας διευκολύνουν στις περιπτώσης αλλαγής ημέρας μεταξύ δύο εγγραφών σύμφωνα με τις οδηγίες που μας δώθηκαν.

```
#func to check if it's a new day
def has_day_changed(timestamp1, timestamp2):  #
    # Convert Unix timestamps to datetime objects
    date1 = datetime.utcfromtimestamp(timestamp1).date()
    date2 = datetime.utcfromtimestamp(timestamp2).date()

    # Compare the day components
    return date1 != date2


def seconds_until_midnight(unix_timestamp):
    # Convert Unix timestamp to datetime object
    dt = datetime.utcfromtimestamp(unix_timestamp)

    # Calculate midnight of the next day
    midnight = datetime(dt.year, dt.month, dt.day) + timedelta(days=1)

    # Calculate the time difference in seconds
    seconds_remaining = (midnight - dt).total_seconds()

    return int(seconds_remaining)

def seconds_from_start_of_day(unix_timestamp):
    # Convert Unix timestamp to datetime object
    dt = datetime.utcfromtimestamp(unix_timestamp)

    # Calculate midnight of the same day
    start_of_day = datetime(dt.year, dt.month, dt.day)

    # Calculate the time difference in seconds
    seconds_elapsed = (dt - start_of_day).total_seconds()

    return int(seconds_elapsed)
```

Έπειτα βρίσκουμε τις ξεχωριστές διαμονές των συμμετεχόντων

```
participants_stays={}
for participant in sorted(list(participants.keys())):

    moves=participants[participant]
    stays= {key: 0 for key in room_names}
    for i,move in enumerate(moves):
        if i==0:
            continue
        if has_day_changed(moves[i][1],moves[i-1][1]):
            stays[moves[i-1][-1]]+=seconds_until_midnight(moves[i-1][1]) #add to last days room the seconds until end of day
            stays[moves[i][-1]]+=seconds_from_start_of_day(moves[i][1])  #add to this days first room the seconds from the start of the day
        else:
            stays[moves[i-1][-1]]+=int(moves[i][1])-int(moves[i-1][1])
    participants_stays[participant]=stays.copy()

print(len(list(participants_stays.keys())))
```

Και τέλος υπολογίζουμε τους επιμέρους χρόνους και ποσοστά διαμονής πριν τα αποθηκεύσουμε στο νέο dataset.

```
#calc percentages of time spent
new_dataset=[]
new_header=["part_id"]+sorted(room_names)

for participant in sorted(list(participants_stays.keys())):
    total=0
    for room in sorted(list(participants_stays[participant].keys())):
        total+=participants_stays[participant][room]
        #print(total)
    if total==0:
        continue

    new_dataset.append([])
    new_dataset[-1].append(participant)


    for room in sorted(list(participants_stays[participant].keys())):

        new_dataset[-1].append(int(1000*participants_stays[participant][room]/total)/10)


print(new_dataset[0])
save_csv(header=new_header,data=new_dataset,filename="tasks/Part_B/new_dataset.csv")
```

```
part_id,Bathroom,Bedroom,Diningroom,Hall,Kitchen,Livingroom,Office,Outdoor
3061,10.9,11.7,25.3,0.0,4.4,11.7,0.0,35.7
3062,0.0,0.0,21.7,26.9,24.0,14.3,0.0,12.9
3064,2.1,25.0,0.0,0.0,18.9,53.7,0.0,0.0
3065,0.7,18.3,44.5,0.0,7.0,9.4,0.0,19.7
3066,5.3,7.0,0.0,0.0,7.7,59.8,0.6,19.2
3067,0.0,0.0,2.2,61.3,27.5,0.1,0.3,8.2
3068,0.0,18.2,12.2,0.0,35.2,11.2,19.9,3.0
3070,3.6,0.0,32.5,0.0,29.2,10.0,0.5,23.9
```

New Dataset Format

4. Συνδιασμός Clinical και Beacons Feature

Ακόμη συνδιάζουμε το Clinical με το New_Dataset (που προέκυψε από την εξαγωγή χαρακτηριστηκών από το Beacons) σε ένα νέο.

```
new_header=header_beacons+header_clinical[1:]
#print(new_header)

merged_datasets=[]

def find_entry(element_to_find,my_list):
    index_i = next((i for i, sublist in enumerate(my_list) if sublist[0] == element_to_find), None)
    return index_i

#print(find_entry('3546',dataset_beacons))


for row in entries_beacons:
    beacons_index=row[0]
    #print(beacons_index)
    clinical_index=find_entry(beacons_index,entries_clinical)
    if clinical_index!=None:
        #print(row,entries_clinical[clinical_index])
        merged_datasets.append(row+entries_clinical[clinical_index][1:])

def save_csv(header, data, filename):
    with open(filename, 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)

        # Write the header
        csv_writer.writerow(header)

        # Write the data, replacing None with an empty string
        for row in data:
            csv_writer.writerow(['' if cell is None else cell for cell in row])

save_csv(header=new_header,data=merged_datasets,filename="tasks/Part_B/merged.csv")
```

tasks > Part_B > merged.csv > data

```
part_id,Bathroom,Bedroom,Diningroom,Hall,Kitchen,Livingroom,Office,Outdoor,fried,gender,age,hospitalization_one_year,hospitalization_three_years,ortho_hypot
1088,0.0,0.1,0.0,0.1,2.0,25.1,0.0,72.5,2,1,81,0,1.0,0,1,0,0,2,10.0,0,15.4,8.1,0,1,1,0,3.0,0.0,44.0625,36.6,120,71.5152,11,27,0,1,26,4,9.2,1,7,0,3.0,7.0,360,
1090,0.0,0.0,0.0,99.8,0.1,0.0,0.0,0.0,1,1,73,0,3.0,0,1,0,0,1,55.0,0,7.9,11.9,0,1,0,1,0.0,0.0,27.890625,30.2,92,49.8372,13,24,0,1,25,4,3.7,1,3,0,2.0,7.0,40,0
1104,19.8,16.2,0.0,0.0,16.2,0.0,0.0,47.5,2,1,79,0,0.0,0,1,0,0,2,55.0,0,65.0,22.8,1,1,1,0,0.0,0.0,38.73577952,33.7,118,64.1121,14,25,0,1,28,2,3.4,1,7,1,4.0,3
1109,0.0,9.0,0.0,0.0,82.0,8.9,0.0,0.0,1,-1,84,0,0.0,0,0,2,0,1,8.7,0,11.2,10.7,0,1,0,0,0.0,0.0,33.56544078,28.2,117,71.2974,14,26,0,1,30,7,7.8,0,7,0,1.0,3.0,
1509,1.0,9.4,0.0,19.5,41.0,11.5,0.0,17.3,2,1,70,1,1.0,0,1,0,0,1,55.0,0,65.0,50.0,1,1,1,1,0.0,0.0,37.15134888,33.4,125,64.935,14,27,0,0,29,1,5.3,1,1,1,2.0,25
1515,0.0,0.0,0.0,0.2,33.1,44.2,0.0,22.3,0,1,71,1,1.0,0,1,0,0,1,12.58,0,7.14,4.57,0,0,0,0,0.0,1.0,28.96193772,29.8,109,58.7574,14,29,0,0,29,2,5.4,1,7,0,3.0,4
2005,0.0,0.0,0.0,0.0,33.3,66.6,0.0,0.0,1,1,77,0,0.0,1,0,0,0,1,14.0,1,6.0,7.0,0,1,1,0,0.0,1.0,26.34649403,14.204269,87,34.321518,13,26,1,0,26,3,3,0,4,0,5.0,7
2006,0.0,57.4,0.0,0.0,0.2,0.2,0.0,42.0,1,-1,76,2,8.0,0,0,0,0,2,15.0,0,8.0,6.0,0,0,1,0,1.0,0.0,26.14268848,17.974712,93,52.251286,14,25,1,0,28,4,1,1,7,1,7.0,
2012,0.0,0.0,0.0,0.0,68.6,13.2,0.0,18.1,2,1,87,0,0.0,1,0,1,0,1,21.0,1,22.0,7.0,0,1,1,1,0.0,1.0,23.24459454,15.522682,82,38.28713,13,15.576326,0,0,25,2,2,0,1
```

Merged Dataset Format

5. Συσταδοποίηση τελικού Dataset

a. Συσταδοποίηση

Τα δεδομένα μας αρχικά διαιρούνται κατά στήλες με τον μέσο της στήλης για να εξισώσουμε τη συνεισφορά κάθε στείλεις στην γεωμτερική απεικόνιση των εγγραφών.

Από τα δεδομένα μας αφαιρούμε τις στήλες των IDs και το Fried-Index.

Για την συσταδοποίηση των δεδομένων αξιοποιούμε k-means και δοκιμάζουμε πολλαπλά πλήθη κεντροειδών. Αξιοποιούμε σε κάθε πιθανό πλήθος την μετρική της «αδράνειας» και το sil index και τα τοποθετούμε σε γραφική παράσταση για να βρούμε τον αγκώνα του γραφήματος και το σχετικό sil index.

```
# Normalize the dataset
normalized_data = (data)/mean  #this way we equalie the weight of each field to the output
X=normalized_data
#X=reduced_data


error_list=[]
sil_list=[]
for i in range(2,20):
    # Define the number of clusters (k)
    k = i

    # Apply k-means clustering
    kmeans = KMeans(n_clusters=i, random_state=42,n_init=20,max_iter=500)
    labels = kmeans.fit_predict(X)
    wcss = kmeans.inertia_
    error_list.append(wcss/10000)

    # Calculate silhouette score
    silhouette_avg = silhouette_score(X, labels)
    sil_list.append(silhouette_avg)
    #print(f"Silhouette Score for {i} clusters: {silhouette_avg}")

import matplotlib.pyplot as plt

# Create a plot with the first list
plt.plot(range(2, len(error_list) + 2),error_list, label='WCSS/10000')

# Add the second list to the same plot
plt.plot(range(2, len(error_list) + 2),sil_list, label='Silhouette Index')

# Add labels and a legend
plt.title('Plot of Two Lists')
plt.xlabel('X-axis Cluster Number')
plt.ylabel('Y-axis (Values)')
plt.legend()

# Show the plot
plt.show()
```

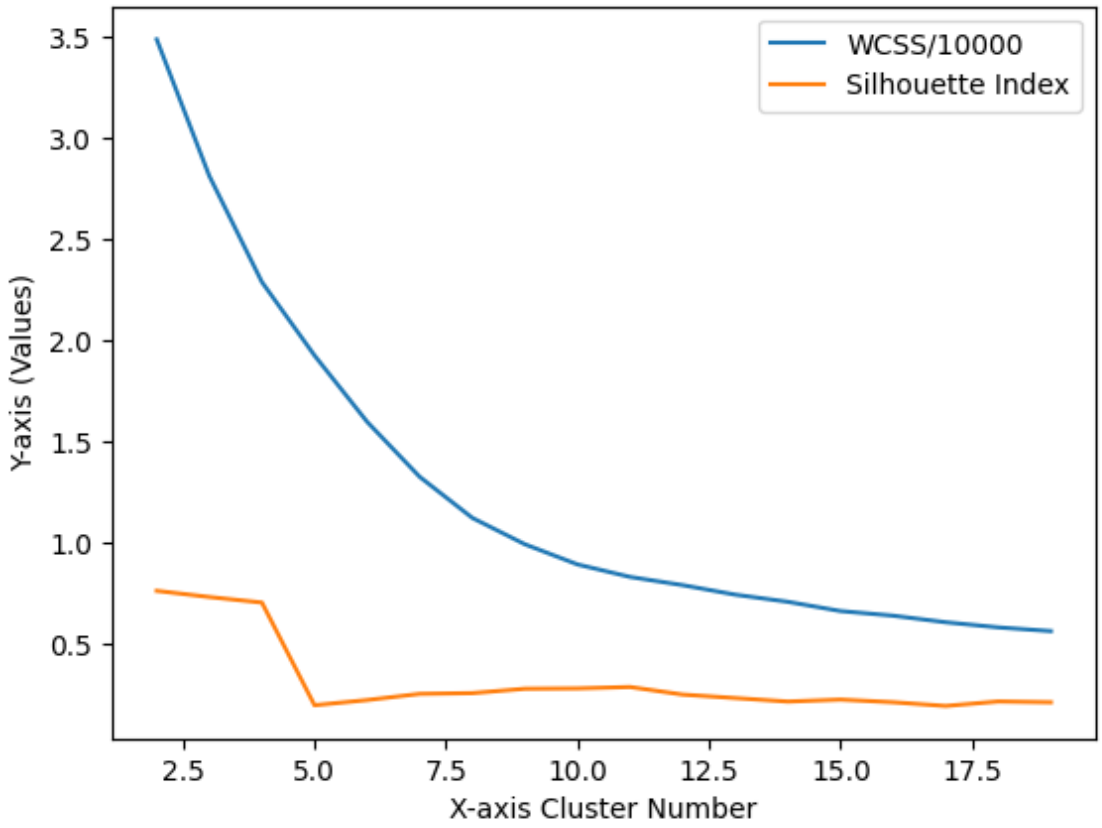
b. PCA

Επιπλέον παράγουμε ένα δεύτερο set δεδομένων με χρήση PCA για να μικρύνουμε τη διαστατικότητα. Για το πλήθως των components “k” δοκιμάστηκαν διάφορες επιλογές και τα αποτελέσματα συζητούνται παρακάτω.

```
mean = np.mean(data, axis=0)




# Step 3: Compute the covariance matrix
cov_matrix = np.cov(data/mean, rowvar=False)

# Step 4: Calculate eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Step 5: Sort eigenvalues and corresponding eigenvectors in descending order
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]

# Step 6: Choose the top k eigenvectors
k = 8  # Choose the desired dimensionality
top_eigenvectors = eigenvectors[:, :k]

# Step 7: Form a projection matrix
projection_matrix = top_eigenvectors

# Step 8: Project the original dataset onto the new subspace
reduced_data = np.dot(data/mean, projection_matrix)
```

c. Αποτελέσματα

Για το απλό clustering παίρνουμε το παρακάτω διάγραμμα.

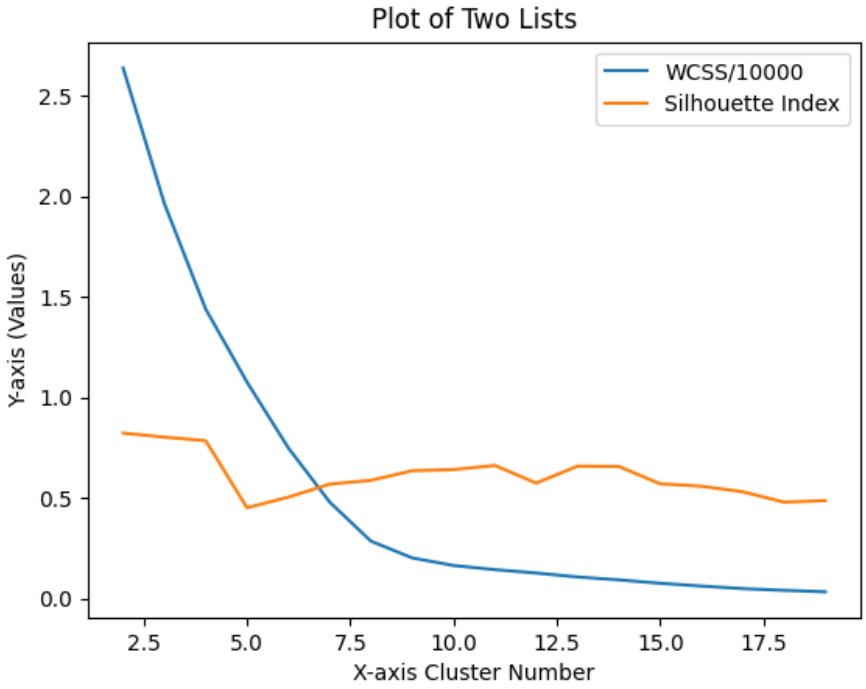


Οπτικά εντοπίζουμε τον αγκώνα του γραφήματος στα 9 κεντροειδή και το αντίστοιχο sil index είναι 0.291, σχετικά ασθενές.

Με PCA αρχικά δοκιμάζουμε πιθανά πλήθη για τα components και συναντάμε τα 9 κεντροειδή στα 8 components (όσα περισσότερα components και αν προσθέσουμε τα κεντροειδή θα παραμείνουν 9 αφού τόσα βρήκαμε και πριν το dimensionality reduction).

```
#by experiementing with pca we gain the following:
#components  #elbow  #silhouette value
#2               #2        0.965
#3               #4        0.775
#4               #5        0.778
#5               #6        0.771
#6               #6        0.771
#7               #7        0.697
#8               #9        0.673
#9               #9        0.641
#10              #9        0.564
#11              #9        0.510
#12              #9        0.472
```

Για 8 components παίρνουμε το παρακάτω γράφημα:



Με 9 κεντροειδή και sil index 0.673, καλό.