

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Τμήμα Πληροφορικής



Εργασία Μαθήματος

«Αναλυτική Δεδομένων (6^ο εξ.)»

Όνομα φοιτητή – Αρ. Μητρώου	Μίμογλου Γιώργος – Π17073 Παπαθεοχάρους Ιωάννης – Π17002
Ημερομηνία παράδοσης	24/6/2020



Εισαγωγή

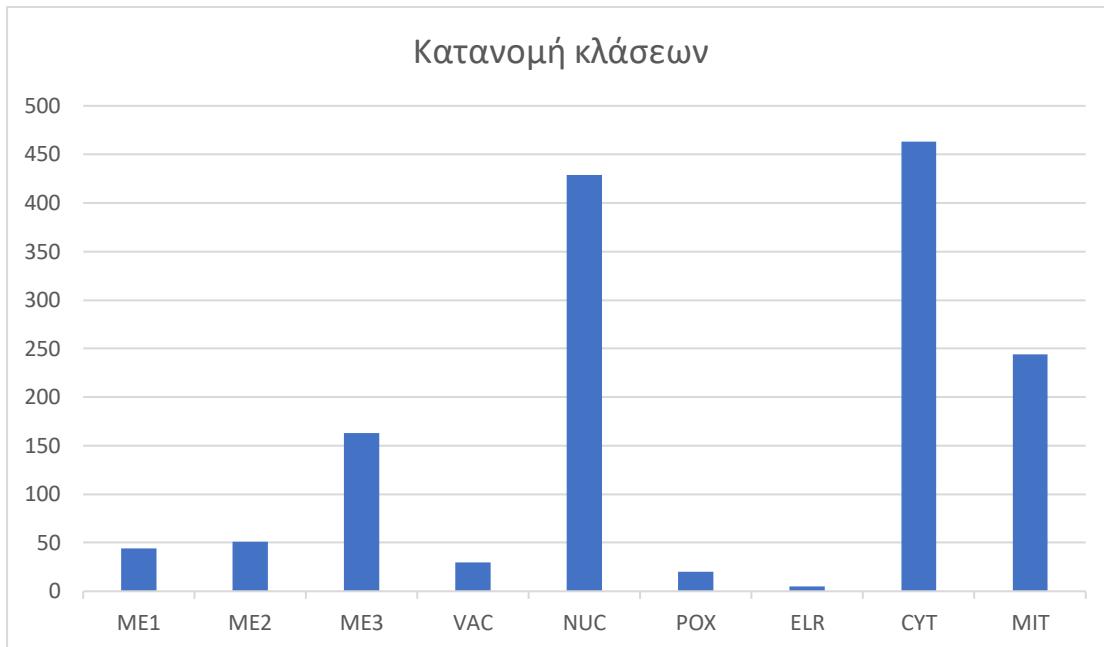
Η Αναλυτική Δεδομένων έχει ως σκοπός τη χρήση και την ανάλυση των δεδομένων, ώστε να προβούμε σε ορισμένα συμπεράσματα και να εξάγουμε γνώση από αυτά. Τα βασικά στάδια της διαδικασίας αυτής είναι η συλλογή των δεδομένων, ο καθαρισμός και ο μετασχηματισμός τους, η ανάλυση και η εξαγωγή συμπεράσματος.

Σκοπός της εργασίας αυτής, είναι να έρθει ο φοιτητής σε επαφή με την εν λόγω διαδικασία, να αναπτύξει της γνώσεις του στο κύριο εργαλείο που χρησιμοποιείται, τη μηχανική μάθηση και εξοικειωθεί με τις τεχνολογίες που τον βοηθούν στα στάδια της.

Υλοποίηση

Μέρος Α

Το πρόβλημα αυτό αποτελεί πρόβλημα ταξινόμησης 10 κυτταρικών τοποθεσιών. Το σύνολο αποτελείται από 1484 εγγραφές με 8 χαρακτηριστικά. Η κατανομή των κλάσεων είναι η εξής:



Από αυτό και μόνο το γράφημα καταλαβαίνουμε πως η διαδικασία προπαρασκευής των δεδομένων πρέπει να γίνει προσεκτικά. Θα αναφερθούμε σε αυτό αργότερα.

Ερώτημα 1: Προπαρασκευή δεδομένων

Αρχικά γίνεται ο μετασχηματισμός του αρχείου δεδομένων σε .csv (Εικόνα 1) διότι η αρχική μορφή προκαλούσε προβλήματα κατά τη φόρτωση των δεδομένων στο DataFrame. Στη συνέχεια γίνεται φόρτωση των δεδομένων σε DataFrame με τη βοήθεια του πακέτου pandas (Εικόνα 2).



```
# TRANSFORM FILE

# Changing separators to "," , because double space caused problems with pandas.
# I renamed the file yeast.data to data.txt.
# Uncomment and execute, if newdata.csv does not exist.

file = open("data.txt", "r")
lines = file.readlines()
file.close()
new_lines = []
for l in lines:
    # To replace double space.
    nl = l.replace("  ", ",")
    # To replace double comma.
    nl = nl.replace(",,", ",")
    # To eliminate any space left.
    nl = nl.replace(" ", "")

    new_lines.append(nl)

file = open("newdata.csv", "w")
file.writelines(new_lines)
file.close()
```

Εικόνα 1 File transformation

```
# Loading data.
names = ["name", "MCG", "GVH", "ALM", "MIT", "ERL", "POX", "VAC", "NUC", "class"]
data = pd.read_csv("newdata.csv", sep=",", names=names, header=None)
print(f"Dataset size: {len(data)}")
print(data.head(5))

Dataset size: 1484
      name    MCG   GVH   ALM   MIT   ERL   POX   VAC   NUC class
0  ADT1_YEAST  0.58  0.61  0.47  0.13  0.5  0.0  0.48  0.22   MIT
1  ADT2_YEAST  0.43  0.67  0.48  0.27  0.5  0.0  0.53  0.22   MIT
2  ADT3_YEAST  0.64  0.62  0.49  0.15  0.5  0.0  0.53  0.22   MIT
3  AAR2_YEAST  0.58  0.44  0.57  0.13  0.5  0.0  0.54  0.22   NUC
4  AATM_YEAST  0.42  0.44  0.48  0.54  0.5  0.0  0.48  0.22   MIT
```

Εικόνα 2 Load data from csv into DataFrame

Σύμφωνα με το συνοδευτικό αρχείο πληροφοριών των δεδομένων, το dataset δεν περιέχει τιμές NaN. Επιπλέον, παρατηρήθηκε πως δεν υπήρχαν λοιπά λάθη, όπως τυπογραφικά στα ονόματα των κλάσεων. Παρόλα αυτά, έγιναν οι αντίστοιχες διαδικασίες καθαρισμού (Εικόνα 4), καθώς και η διαγραφή της στήλης name (Εικόνα 3).



```
# 1.1) CLEANING

# Dropping name column, because we don't need the Accession number for the SWISS-PROT database.
data.drop("name", axis=1, inplace=True)
print(f"Dataset size: {len(data)}")
print(data.head(5))

Dataset size: 1484
   MCG    GVH    ALM    MIT    ERL    POX    VAC    NUC  class
0  0.58  0.61  0.47  0.13  0.5   0.0   0.48  0.22  MIT
1  0.43  0.67  0.48  0.27  0.5   0.0   0.53  0.22  MIT
2  0.64  0.62  0.49  0.15  0.5   0.0   0.53  0.22  MIT
3  0.58  0.44  0.57  0.13  0.5   0.0   0.54  0.22  NUC
4  0.42  0.44  0.48  0.54  0.5   0.0   0.48  0.22  MIT
```

Εικόνα 3 Dropping name column

```
# Dropping rows with NA value.
print(f"Dataset size before: {len(data)}")
data.dropna(inplace=True)
data.head(5)
print(f"Dataset size after: {len(data)}")
print(data.head(5))

Dataset size before: 1484
Dataset size after: 1484
   MCG    GVH    ALM    MIT    ERL    POX    VAC    NUC  class
0  0.58  0.61  0.47  0.13  0.5   0.0   0.48  0.22  MIT
1  0.43  0.67  0.48  0.27  0.5   0.0   0.53  0.22  MIT
2  0.64  0.62  0.49  0.15  0.5   0.0   0.53  0.22  MIT
3  0.58  0.44  0.57  0.13  0.5   0.0   0.54  0.22  NUC
4  0.42  0.44  0.48  0.54  0.5   0.0   0.48  0.22  MIT

# Dropping records with wrong class. Maybe, the dataset contains records with unacceptable classes.
# Accepted classes.
CLASSES = ["CYT", "NUC", "MIT", "ME3", "ME2", "ME1", "EXC", "VAC", "POX", "ERL"]

print(f"Dataset size before: {len(data)}")
data = data[data['class'].isin(CLASSES)]
print(f"Dataset size after: {len(data)}")
print(data.head(5))

Dataset size before: 1484
Dataset size after: 1484
   MCG    GVH    ALM    MIT    ERL    POX    VAC    NUC  class
0  0.58  0.61  0.47  0.13  0.5   0.0   0.48  0.22  MIT
1  0.43  0.67  0.48  0.27  0.5   0.0   0.53  0.22  MIT
2  0.64  0.62  0.49  0.15  0.5   0.0   0.53  0.22  MIT
3  0.58  0.44  0.57  0.13  0.5   0.0   0.54  0.22  NUC
4  0.42  0.44  0.48  0.54  0.5   0.0   0.48  0.22  MIT
```

Εικόνα 4 Dropping NA values and wrong classes

Όπως φαίνεται για στα παραπάνω screenshots, το μέγεθος του συνόλου δεδομένων διατηρείται το ίδιο μετά από κάθε διαδικασία καθαρισμού.

Έπειτα, ακολουθεί η διαδικασία της κανονικοποίησης (Εικόνα 5).

Το τελευταίο στάδιο της προπαρασκευής δεδομένων που έγινε είναι η μείωση του όγκου δεδομένων. Το dataset είναι εξαρχής μικρό, οπότε τα παραγόμενα σύνολα δεν χρησιμοποιούνται στην διαδικασία της ανάλυσης. Σε αυτό το στάδιο πειραματιστήκαμε με διάφορες τεχνικές. Η πρώτη απόπειρα έγινε με τη χρήση istogramματος για κάθε ένα attribute. Έγιναν δοκιμές με διάφορες τιμές των bins και παρατηρήσαμε διαφορές.

Ενδεικτικά, για $bins = 10$ η πληροφορία που δεχτήκαμε ήταν σχετικά μικρή (Εικόνα 6). Για $bins = 80$, λάβαμε πιο συγκεκριμένες πληροφορίες (Εικόνα 7) όπως ποιες τιμές του χαρακτηριστικού έχουν χαμηλή και υψηλή συχνότητα, με μεγαλύτερη ακρίβεια σε σχέση το προηγούμενο istogramμα. Για $bins = 100$ εμφανίστηκαν μερικά κενά ενδιάμεσα της κατανομής (Εικόνα 8). Μας κίνησε την πειράργεια για το πως αυτή η πληροφορία θα μας



βοηθούσε ή αν σήμαινε κάτι όμως δεν καταλήξαμε πουθενά. Ίσως, ευθύνεται η φύση των δεδομένων και κάποιος expert της μοριακής βιολογίας να μας βοηθούσε.

```
# Data to np arrays
x = np.array(data.iloc[:, :8])
print(x[:5])

[[0.58 0.61 0.47 0.13 0.5 0. 0.48 0.22]
 [0.43 0.67 0.48 0.27 0.5 0. 0.53 0.22]
 [0.64 0.62 0.49 0.15 0.5 0. 0.53 0.22]
 [0.58 0.44 0.57 0.13 0.5 0. 0.54 0.22]
 [0.42 0.44 0.48 0.54 0.5 0. 0.48 0.22]]

▶ MI 8/8

# Standardize data
std = StandardScaler().fit_transform(x)
print(std[:5])

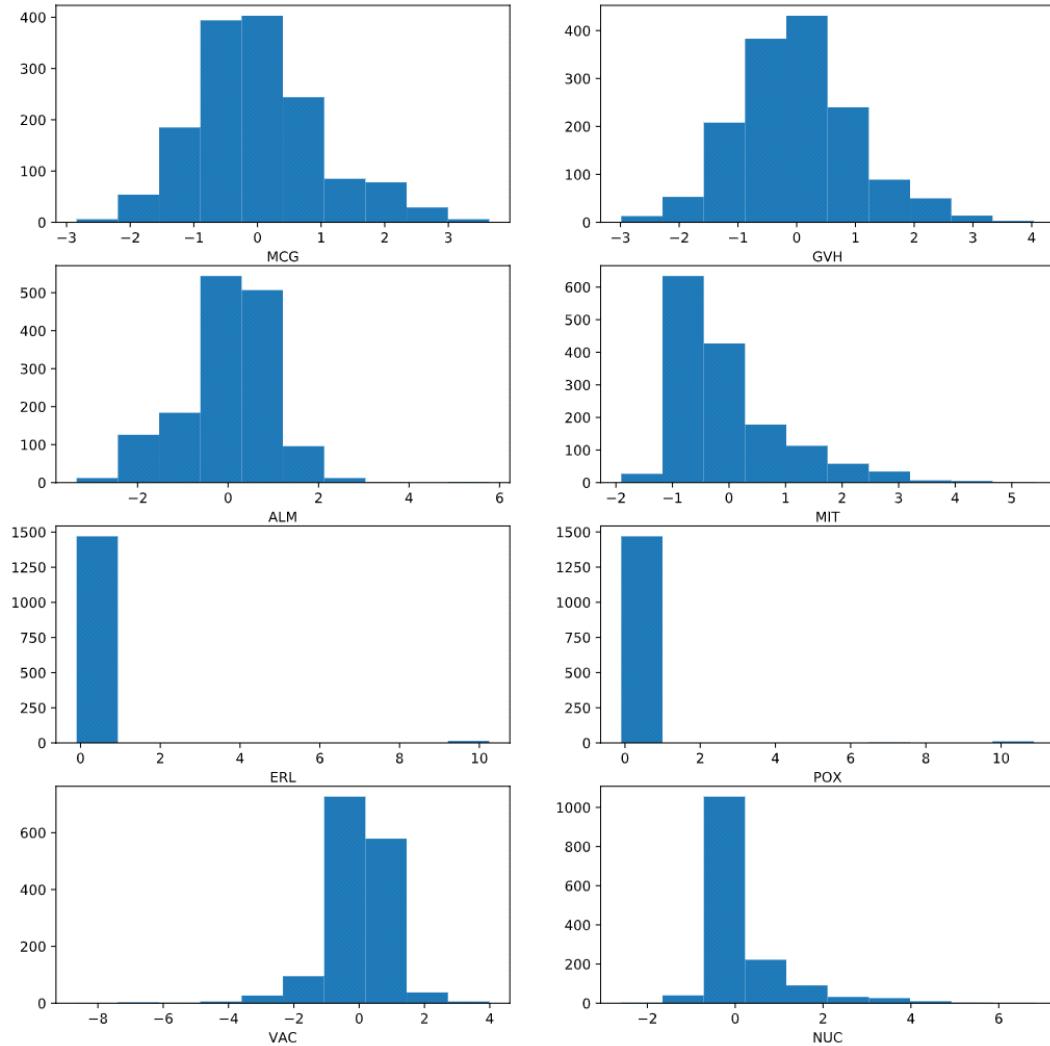
[[ 0.58198136  0.88848148 -0.3466451  -0.95720258 -0.09759001 -0.0991314
 -0.34417514 -0.5279193 ]
 [-0.51089067  1.37281104 -0.23122636  0.06431174 -0.09759001 -0.0991314
  0.52121948 -0.5279193 ]
 [ 1.01913017  0.96920307 -0.11580762 -0.81127196 -0.09759001 -0.0991314
  0.52121948 -0.5279193 ]
 [ 0.58198136 -0.48378562  0.80754232 -0.95720258 -0.09759001 -0.0991314
  0.69429841 -0.5279193 ]
 [-0.5837488 -0.48378562 -0.23122636  2.03437508 -0.09759001 -0.0991314
 -0.34417514 -0.5279193 ]]

▶ MI 8/8

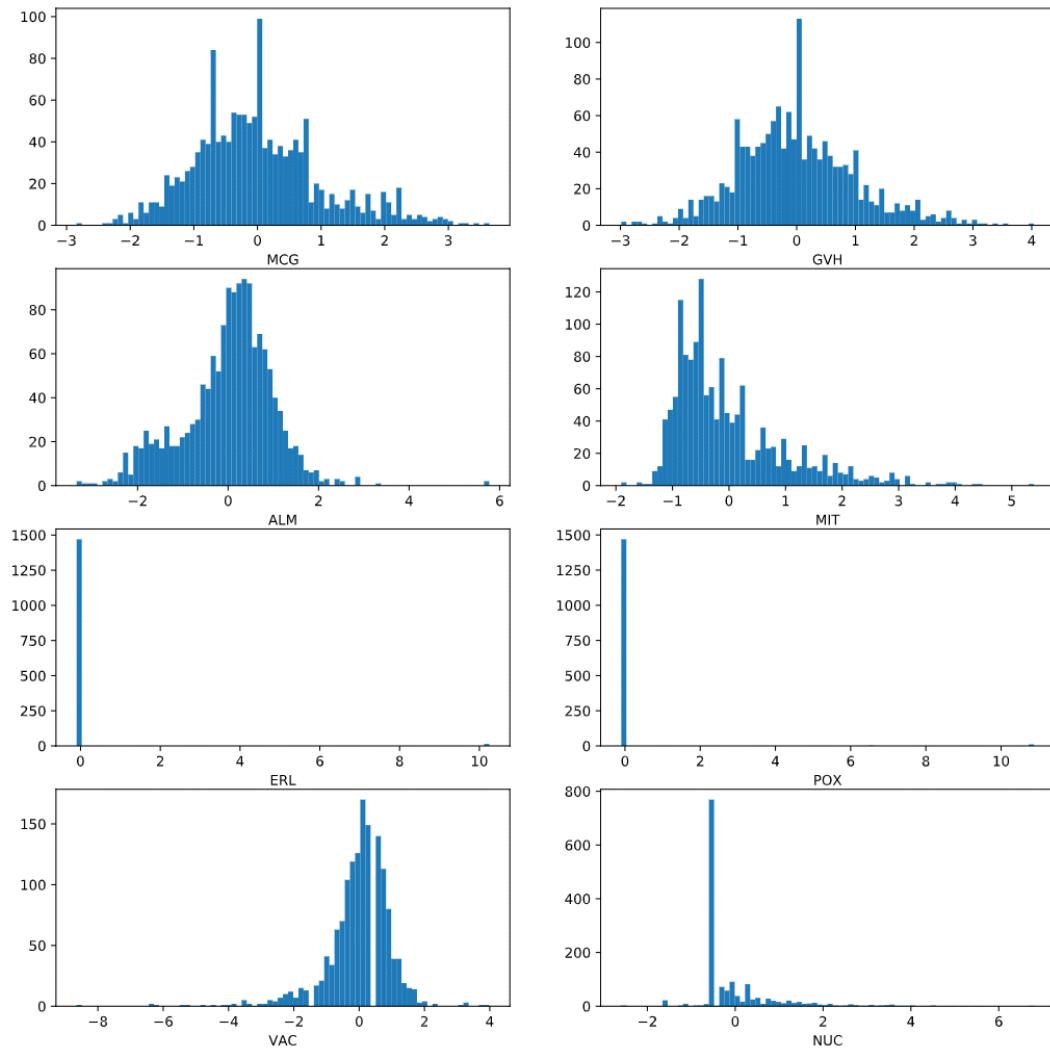
# Standardized data to DataFrame
standardized_data = data.copy()
standardized_data.iloc[:, :-1] = std[:, :]
standardized_data.head(5)
```

	MCG	GVH	ALM	MIT	ERL	POX	VAC	NUC	class
0	0.581981	0.888481	-0.346645	-0.957203	-0.09759	-0.099131	-0.344175	-0.527919	MIT
1	-0.510891	1.372811	-0.231226	0.064312	-0.09759	-0.099131	0.521219	-0.527919	MIT
2	1.019130	0.969203	-0.115808	-0.811272	-0.09759	-0.099131	0.521219	-0.527919	MIT
3	0.581981	-0.483786	0.807542	-0.957203	-0.09759	-0.099131	0.694298	-0.527919	NUC
4	-0.583749	-0.483786	-0.231226	2.034375	-0.09759	-0.099131	-0.344175	-0.527919	MIT

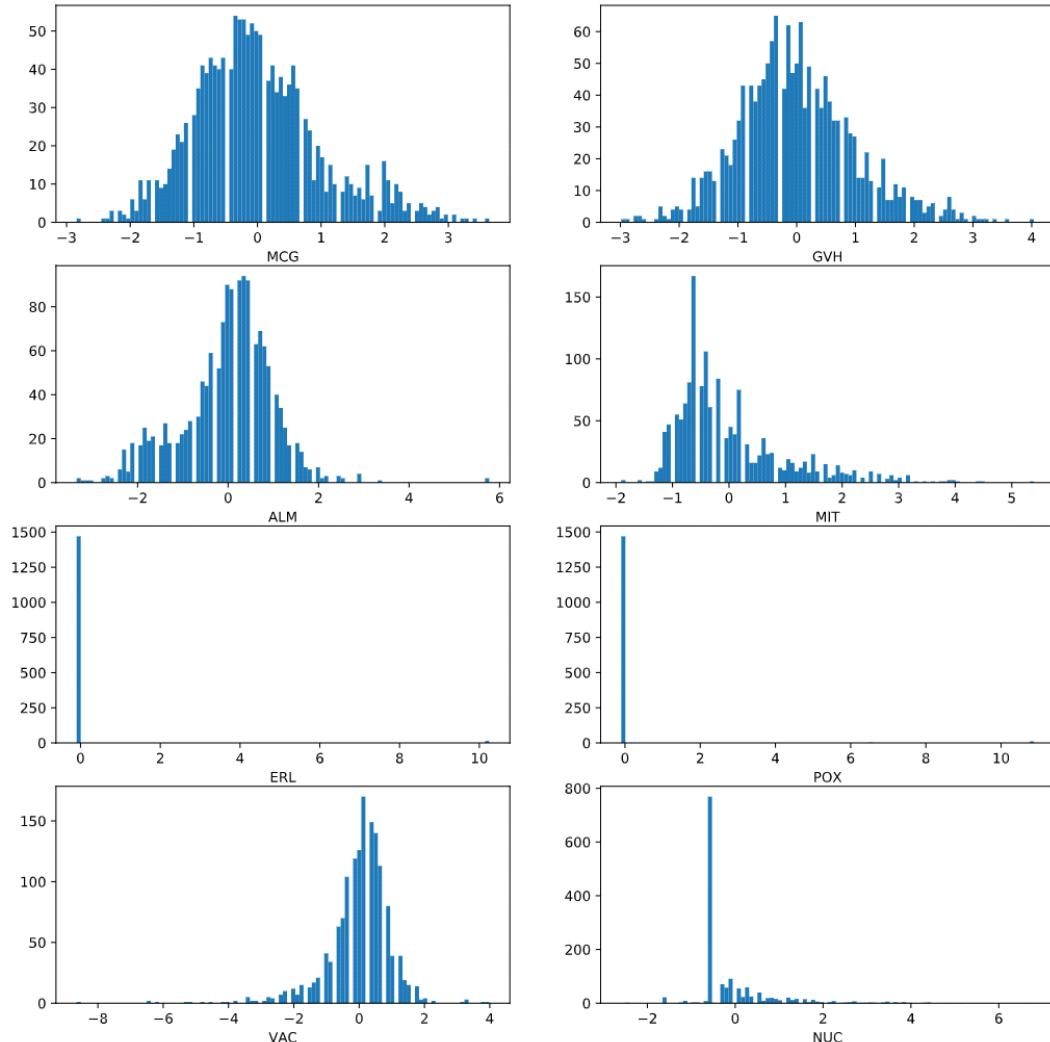
Εικόνα 5 Data standardization



Εικόνα 6 Bins = 10



Εικόνα 7 Bins = 80



Εικόνα 8 Bins = 100

Εδώ αναρωτιόμαστε πως ένα ιστόγραμμα θα μπορούσε να συμβάλλει στην μείωση των δεδομένων. Η απάντηση είναι απλή. Παρατηρώντας τα ιστογράμματα, θέτουμε ένα κατώτατο όριο συχνότητας και έτσι στη συνέχεια, βγάζουμε τις εγγραφές με τις αντίστοιχες τιμές, εκτός του τελικού συνόλου δεδομένων. Υλοποιήσαμε ένα παράδειγμα. Θεωρήσαμε πως για το χαρακτηριστικό MCG, η συχνότητα των τιμών μικρότερων του -2 και μεγαλύτερων του 3. Οπότε, διαγράψαμε τις αντίστοιχες εγγραφές (Εικόνα 9).

```
# Dropping records with low frequency attributes
# Example: Exclude all row with mcg > 3 adn mcg < -2
reduced_data = standardized_data.drop(standardized_data[((standardized_data["MCG"] > 3.0) | (standardized_data["MCG"] < -2.0)).index]
print(f"Dataset size before: {len(data)}")
print(f"Dataset size after: {len(reduced_data)}")
print(reduced_data.head(5))

Dataset size before: 1484
Dataset size after: 1466
      MCG      GVH      ALM      MIT      ERL      POX      VAC \
0  0.581981  0.888481 -0.346645 -0.957203 -0.09759 -0.099131 -0.344175 \
1 -0.510891  1.372811 -0.231226  0.064312 -0.09759 -0.099131  0.521219
2  1.019130  0.969203 -0.115808 -0.811272 -0.09759 -0.099131  0.521219
3  0.581981 -0.483786  0.807542 -0.957203 -0.09759 -0.099131  0.694298
4 -0.583749 -0.483786 -0.231226  2.034375 -0.09759 -0.099131 -0.344175

      NUC  class
0 -0.527919   MIT
1 -0.527919   MIT
2 -0.527919   MIT
3 -0.527919   NUC
4 -0.527919   MIT
```

Εικόνα 9 Dropping records with low frequency

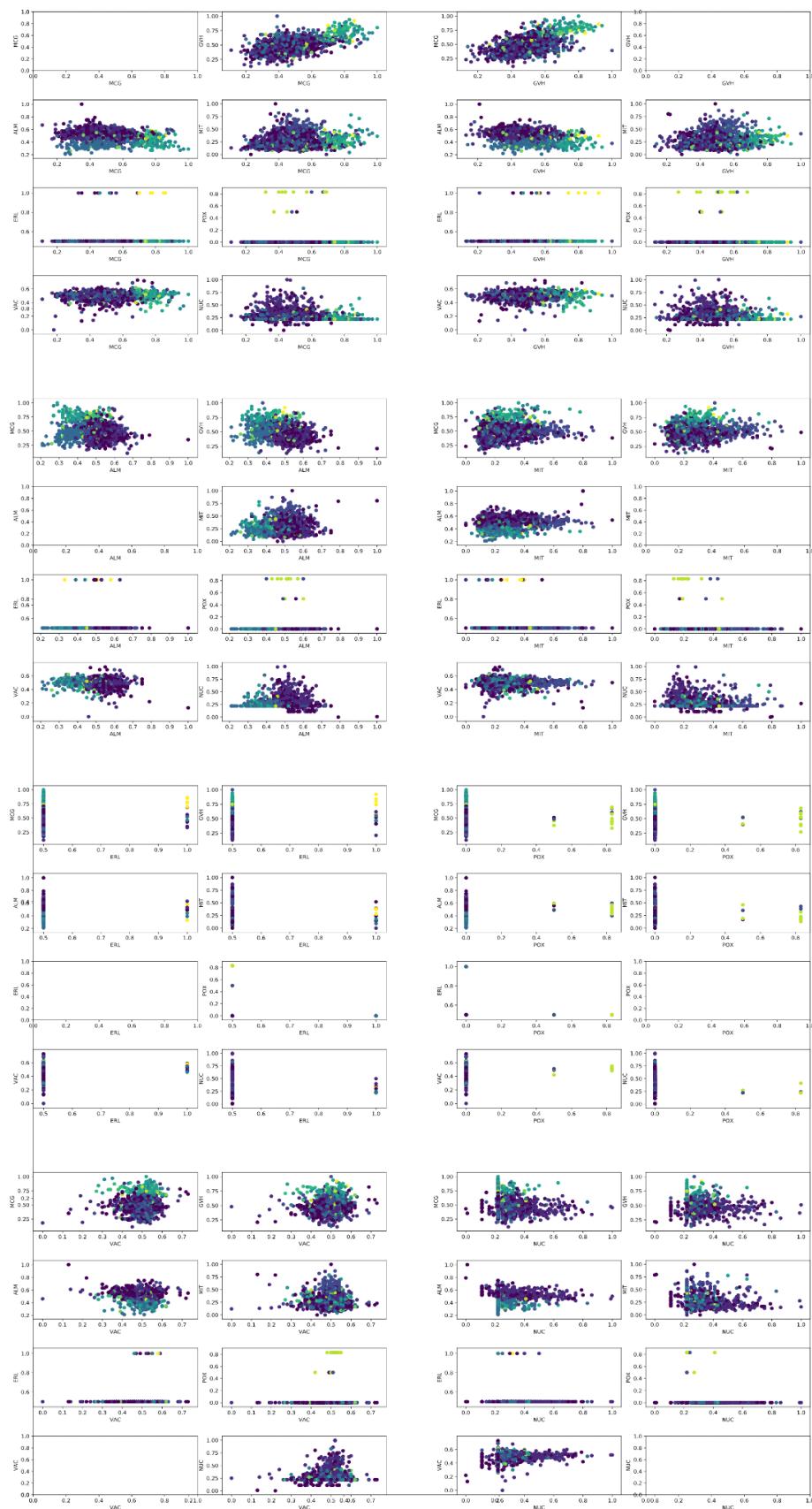


Έτσι, το μέγεθος του dataset μειώθηκε από 1484 σε 1466. Μια τέτοια μείωση σε ένα τόσο μικρό σύνολο δεδομένων παρα μόνο κακό προκαλεί για την ανάλυση που ακολουθεί, αφού αυτές οι τιμές με τις μικρές συχνότητες θα μπορούσαν να ήταν αυτές που θα καθόριζαν την κλάση ERL που βρίσκεται σε μόνο 5 εγγραφές σε όλο το αρχικό dataset. Τετοιές ενέργειες καλό θα ήταν να γίνονται με την καθοδήγηση ειδικού, διότι τα μειονεκτήματα της μεθόδου αυτής είναι αρκετά και να επηρεαστεί η αποδοτικότητα του ταξινομητή.

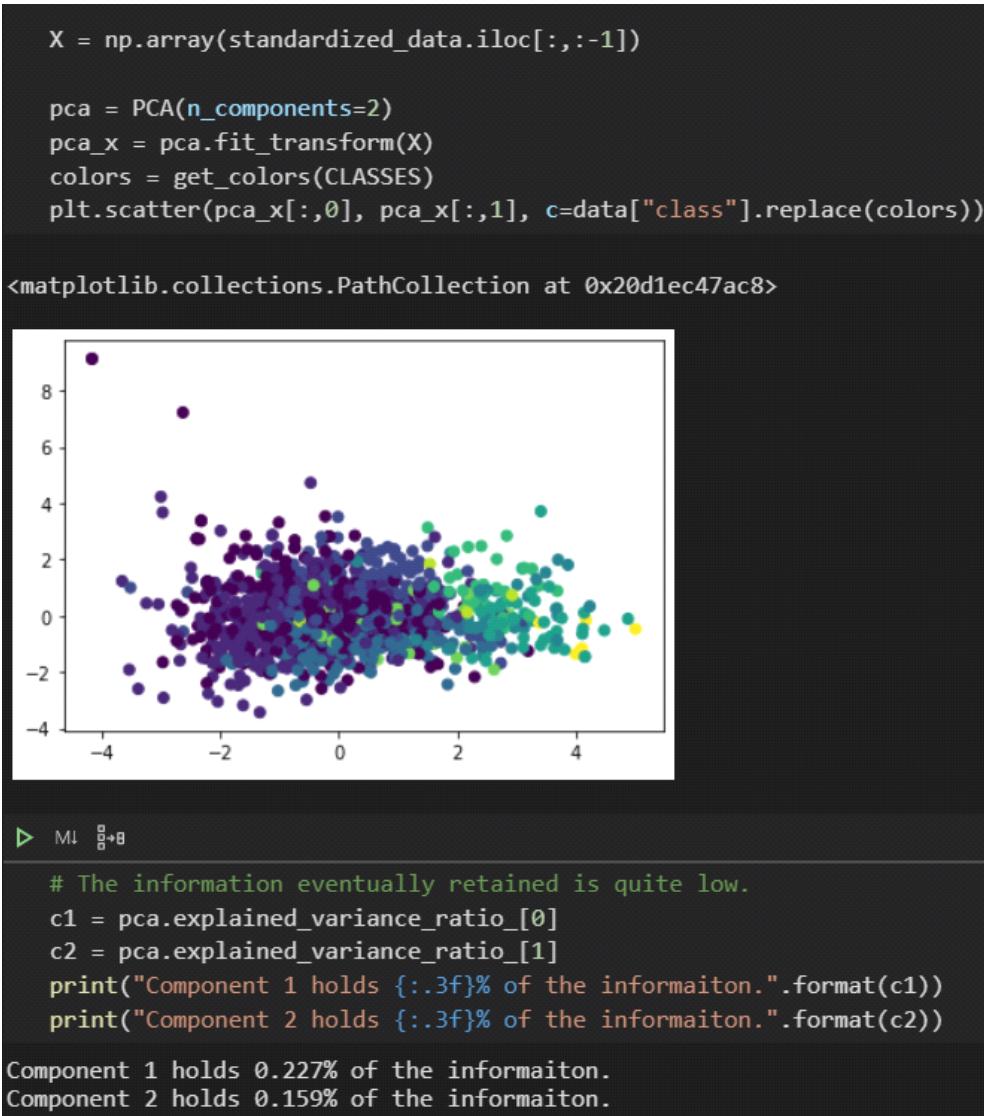
Οι δυο ακόλουθες μέθοδοι έχουν σκοπό τη μείωση των διαστάσεων των δεδομένων, δηλαδή την αγνόηση ορισμένων στηλών. Στην πρώτη μέθοδο, προσπαθούμε να βρούμε κάποια συσχέτιση μεταξύ των attributes μέσω διαγραμμάτων, ενώ στη δεύτερη γίνεται χρήση Principal Component Analysis (PCA).

Για τη πρώτη μέθοδο παράγονται 56 scatter plots (*Εικόνα 10*) δηλαδή ένα για κάθε ένα attribute ως προς το άλλο. Παρατηρήσαμε ορισμένες συσχετίσεις, όμως οι γνώσεις μας είναι περιορισμένες και δεν καταφέραμε να καταλήξουμε σε κάποιο συμπέρσμα.

Στη δεύτερη μέθοδο, έγινε χρήση της PCA. Με λίγα λόγια, η PCA είναι μια διαδικασία που προσπαθεί, μέσω γραμμικών πράξεων, να μειώσει τις διαστάσεις των δεδομένων, «συμπιέζοντας» κατά κάποιον τρόπο την πληροφορία κάθε διάστασης. Είναι προφανές πως υπάρχει σημαντική μείωση της πληροφορίας με αποτέλεσμα να επηρεάζεται αρνητικά η απόδοση του ταξινομητή που θα εκπαιδευτεί σε αυτό το νέο σύλονο δεδομένων. Στην υλοποίηση μας, μειώνουμε τις διαστάσεις των δεδομένων από 8 σε 2 και παρατηρούμε τη μείωση της πληροφορίας (*Εικόνα 11*).



Εικόνα 10 Plotting each attribute as to the others



Εικόνα 11 Dimensionality Reduction with PCA

Ερώτημα 2: Clustering

Ένα σημαντικό πρόβλημα στην συσταδοποίηση είναι η τελική αντιστοίχιση των συστάδων με τις ζητούμενες κλάσεις. Ένας τρόπος για να γίνει η αντιστοίχιση είναι να βρούμε την πλειοψηφία κλάσης σε κάθε συστάδα και να την «ταιριάξουμε» με την συστάδα αυτή. Αυτό, όμως, δεν μας εγγυάται πως θα βρεθούν συστάδες που θα αντιστοιχηθούν σε όλες τις ζητούμενες κλάσεις. Ο αλγόριθμος αντιστοίχισης είναι ο εξής:



```
def match_classes(standardized_data, pred_labels, classes):
    clusters = {k: [] for k in range(len(set(pred_labels)))}
    for i in range(len(pred_labels)):
        clusters[pred_labels[i]].append(standardized_data.iloc[i])
    frq = {k: {clss: 0 for clss in CLASSES} for k in range(len(set(pred_labels)))}
    matched_classes = {}
    print("Matched Classes:")
    for k,v in clusters.items():
        for series in v:
            frq[k][series['class']] +=1
        matched_classes[k] = max(frq[k].items(), key=operator.itemgetter(1))[0]
        print(f"{k} -> {matched_classes[k]}")
    return matched_classes
```

Εικόνα 12 Matching predicted labels with the true labels

KMeans

Ο αλγόριθμος KMeans δημιουργεί *n* συστάδες ίσης διακύμανσης, ελαχιστοποιώντας την αδράνεια ή *inertia*. Inertia είναι το μέτρο εσωτερικής συνεκτικότητας των συστάδων και υπολογίζεται από τον τύπο:

$$\sum_{i=0}^n \min_{\mu_j \in C} (\|x_i - \mu_j\|^2)$$

Αυτό το μέτρο περιέχει μερικά σημαντικά μειονεκτήματα.

- Θεωρούμε ότι οι συστάδες είναι κυρτές και ισοτοπικές, πράγμα το οποίο δεν ισχύει πάντα.
- Δεν είναι κανονικοποιημένο μέτρο, ξέρουμε μόνο ότι οι μικρότερες τιμές αποδίδουν καλύτερα με βέλτιστη τιμή το μηδέν. Όμως, σε πολλές διαστάσεις, η ευκλείδεια απόσταση που υπολογίζεται, τείνει να διογκώνεται. Αυτό το φαινόμενο το ονομάζουμε [“Curse of dimensionality”](#) και προκαλεί αρκετά προβλήματα στη μηχανική μάθηση.

Το τελευταίο μειονέκτημα επηρεάζει αρκετά τις διάφορες τεχνικές της μηχανικής μάθησης, και το clustering δεν αποτελεί εξαίρεση. Οπότε, γνωρίζουμε ήδη πως η συσταδοποίηση που θα υλοποιήσουμε δεν θα είναι αρκετά ακριβής.

Για την υλοποίηση του clustering χρησιμοποιούμε το πακέτο *sklearn*. Ορίζουμε ως *n_clusters* το πλήθος των δοσμένων κλάσεων, δηλαδή 10 και ως *init* τη μέθοδο *k-means++*. Η μέθοδος αρχικοποίησης *k-means++* χρησιμοποιεί έναν έξυπνο τρόπο αρχικοποίησης των μέσων, ώστε να συγκλίνει ο αλγόριθμος γρηγορότερα.



```
# 2.1) KMeans

# k-means++ initializes the centroids in such way that the results are more accurate.
kmeans = KMeans(n_clusters=len(CLASSES), init='k-means++').fit(X)
predicted_class_frequency = pd.Series(kmeans.labels_).value_counts()
print(f"Predicted Frequencies: \n{predicted_class_frequency}\n")

matched_classes = match_classes(standardized_data['class'].values, kmeans.labels_, CLASSES)

print('=' * 12) Report for KMeans {=' * 12}\n")
get_cluster_report(standardized_data, kmeans.labels_, matched_classes)
```

Εικόνα 12 KMeans clustering

Έπειτα, υπολογίζεται η κατανομή των παραγόμενων συστάδων στο σύνολο δεδομένων, αντιστοιχίζονται οι συστάδες με τις δοσμένες κλάσεις, με το τον τρόπο που αναφέρθηκε παραπάνω και τέλος εμφανίζονται τα confusion matrix και classification report με τα precisions, recalls, f1 scores, accuracy avg και macro avg.

Predicted Frequencies:		Matched Classes:	
4	289	0	-> CYT
9	262	1	-> ME1
0	189	2	-> CYT
8	188	3	-> ERL
1	165	4	-> CYT
6	163	5	-> POX
7	121	6	-> ME3
2	78	7	-> NUC
5	15	8	-> MIT
3	14	9	-> NUC
dtype: int64			

Εικόνα 13 Predicted frequencies and matched classes for kmeans



Αν $FRQ_{original}$ το διάνυσμα συχνοτήτων των γνωστών κλάσεων και $FRQ_{predicted}$ το διάνυσμα συχνοτήτων των παραγόμενων συστάδων. Προσπαθούμε να βρούμε τέτοιο `random_state*` τέτοιο ώστε να ελαχιστοποιείται η τιμή:

$$\|FRQ_{original} - FRQ_{predicted}\|^2$$

Τέλος, υπολογίζουμε ξανά τις συστάδες που παράγονται από το `random_state*` και τις αντιστοιχούμε με τις γνωστές κλάσεις όπως παραπάνω.

```
min_dist = np.iinfo(np.int32).max
seed(1)
for i in range(300):
    rand_state = randint(0,500)
    kmeans = KMeans(n_clusters=len(CLASSES), init='k-means++', random_state=rand_state).fit(X)
    predicted_class_frequency = pd.Series(kmeans.labels_).value_counts()
    # Calculate the distance and keep the minimum.
    dist = np.linalg.norm(np.array(original_class_frequency) - np.array(predicted_class_frequency), 2)
    if dist < min_dist:
        min_dist = dist
        min_labels = kmeans.labels_

print(f"Minimum distance: {min_dist}\n")
min_predicted_class_frequency = pd.Series(min_labels).value_counts()
print(f"Predicted Frequencies: \n{min_predicted_class_frequency}\n")

matched_classes = match_classes(stanardized_data['class'].values, min_labels, CLASSES)
print('=' * 12) Report for KMeans {'=' * 12}\n")
get_cluster_report(stanardized_data['class'].values, min_labels, matched_classes)
```

Εικόνα 15 Experimental procedure to find best random_state

Σημαντικό!

Ο παραπάνω τρόπος εννοείται πως περιέχει λάθη και εξακολουθεί να αστοχεί στην σωστή συσταδοποίηση. Ήταν μια ιδέα εκείνης της στιγμής και θελήσαμε να την υλοποιήσουμε.

<pre>Minimum distance: 186.47251808242416 Predicted Frequencies: 0 394 3 315 5 195 7 160 6 122 9 119 1 88 8 62 4 15 2 14 dtype: int64</pre>	<pre>Matched Classes: 0 -> CYT 1 -> CYT 2 -> ERL 3 -> NUC 4 -> POX 5 -> MIT 6 -> NUC 7 -> ME3 8 -> EXC 9 -> ME1</pre>
---	---

Εικόνα 16 Predicted frequencies and matched classes for our implementation



- $-2.8 < GVH < 3.5$
- $-3.0 < ALM < 4$
- $-1.8 < MIT < 4.5$
- $-6.0 < VAC < 3.5$
- $-2.0 < NUC < 6$

```
def data_prep_noise(standardized_data):
    standardized_data_with_noise = standardized_data.copy()
    for i in range(len(standardized_data_with_noise)):
        record = standardized_data_with_noise.iloc[i,:]

        cnt = 0
        if not -2.5 < record["MCG"] < 3.5 or not -2.8 < record["GVH"] < 3.5
        or not -3 < record["ALM"] < 4 or not -1.8 < record["MIT"] < 4.5
        or not -6 < record["VAC"] < 3.5 or not -2 < record["NUC"] < 6:

            standardized_data_with_noise.at[i, 'class'] = "NOISE"

    original_class_with_noise_frequency = standardized_data_with_noise.iloc[:, -1].value_counts()
    return (standardized_data_with_noise, original_class_with_noise_frequency)
```

Εικόνα 18 Getting data with noise

Επόμενος στόχος μας, ο καθορισμός των παραμέτρων min_samples και eps. Για την πρώτη, παρατηρούμε στο dataset πως η κλάση με τις μικρότερες εμφανίσεις είναι η ERL, με 5 εμφανίσεις. Οπότε, το 5 αποτελεί ανώτατο όριο για την τιμή του min_samples, ώστε ο αλγόριθμος να εντοπίσει και την εν λόγω κλάση. Για την δεύτερη, τα πράγματα είναι πιο δύσκολα, αφού το eps αποτελεί μέγιστη απόσταση μεταξύ γειτόνων και όπως αναφέραμε, το “Curse of Dimensionality” κυριαρχεί στο σύνολο των δεδομένων. Οι δοκιμές που έπρεπε να γίνουν ήταν αρκετές. Για αυτό το λόγο, δημιουργήσαμε μια επαναληπτική διαδικασία, δοκιμάζοντας κάθε φορά διαφορετικές τιμές του eps. Σε αντίθεση με την επαναληπτική διαδικασία του KMeans, αυτή ψάχνει τη την τιμή που παράγει την καλύτερη ακρίβεια (Εικόνα 19). Δυστυχώς, τα αποτελέσματα δεν ήταν τόσο ενθαρρυντικά, καθώς βρέθηκε ως μέγιστο accuracy μόνο 19% (Εικόνα 20 & 21). Οπότε, σαν τελικό eps θέσαμε 0,66 και πήραμε, εννοείται, τα ίδια αποτελέσματα με την δοκιμαστική επαναληπτική διαδικασία.



```

max_acc = np.iinfo(np.int32).min
seed(1)
classes_with_noise = CLASSES.copy()
classes_with_noise.append('NOISE')
standardized_data_with_noise, original_class_with_noise_frequency = data_prep_noise(standardized_data)
X_noise = np.array(standardized_data_with_noise.iloc[:, :-1])
for eps in np.arange(0.01, 20.00, 0.01):
    dbSCAN = DBSCAN(eps=eps, min_samples=5).fit(X_noise)
    predicted_class_frequency = pd.Series(dbSCAN.labels_).value_counts()
    # Get only those that have lenght 10 and does not contain noise
    if len(set(dbSCAN.labels_)) != 11:
        continue

    matched_classes = match_classes(standardized_data_with_noise['class'].values, dbSCAN.labels_, classes_with_noise)
    trans = lambda x: matched_classes[x]
    decoder = np.vectorize(trans)
    decoded_labels = decoder(dbSCAN.labels_)

    acc = accuracy_score(standardized_data_with_noise['class'].values, decoded_labels)

    if acc > max_acc:
        max_acc = acc
        max_eps = eps
        max_labels = dbSCAN.labels_

print(f"Max accuracy: {max_acc}\n")
print(f"Eps: {max_eps}\n")

max_predicted_class_frequency = pd.Series(max_labels).value_counts()
print(f"Predicted Frequencies: \n{max_predicted_class_frequency}\n")

matched_classes = match_classes(standardized_data_with_noise['class'].values, max_labels, classes_with_noise)
print(f"{'=' * 12} Report for DBSCAN {'=' * 12}\n")
get_cluster_report(standardized_data_with_noise['class'].values, max_labels, matched_classes)

```

Εικόνα 19 Experimental procedure to find best eps.

Predicted Frequencies:	
-1	1004
0	407
2	23
5	10
7	7
3	7
1	7
8	5
6	5
4	5
9	4

Matched Classes:	
-1	-> NOISE
0	-> CYT
1	-> MIT
2	-> MIT
3	-> MIT
4	-> NUC
5	-> NUC
6	-> EXC
7	-> CYT
8	-> ME3
9	-> NUC

Εικόνα 20 Predicted frequencies and matched classes for experimental dbSCAN.



Ερώτημα 3: Classification

Η προπαρασκευή των δεδομένων για την ταξινόμηση είναι σχετικά απλή. Ανακατεύουμε τα δείγματα, τα οποία έχουν ήδη κανονικοποιημένα τα χαρακτηριστικά τους. Μετατρέπουμε τις ετικέτες τους σε one-hot encoding και χωρίζουμε το dataset σε train set, validation set και test set (*Εικόνα 25*).

```
standardized_data = shuffle(standardized_data)

X = np.array(standardized_data.iloc[:, :8])
Y = np.array(standardized_data.iloc[:, -1])

# Labels to one-hot encoding
encoder = LabelEncoder().fit(Y)
y_bool = encoder.transform(Y)
y = np_utils.to_categorical(y_bool)

# Setting sizes
len_data = X.shape[0]
print("Data size: %d" % len_data)
train_size = int(len_data * .6)
valid_size = int(len_data * .1)

print ("Train size: %d" % train_size)
print ("Validation size: %d" % valid_size)
print ("Test size: %d" % (len_data - (train_size + valid_size)))

xtr = X[:train_size,:]
ytr = y[:train_size,:]
ytr_bool = y_bool[:train_size]

xva = X[train_size:train_size+valid_size,:]
yva = y[train_size:train_size+valid_size,:]
yva_bool = y_bool[train_size:train_size+valid_size]

xte = X[train_size+valid_size:,:]
yte = y[train_size+valid_size:,:]
yte_bool = y_bool[train_size+valid_size:]
```

Εικόνα 25 Data preparation for classification.

Έπειτα, ακολουθεί η κατασκευή του μοντέλου. Το μοντέλο περιέχει συνολικά 4 στοιβάδες, μια εισόδου, δυο κρυφές και μια εξόδου. Η στοιβάδα εισόδου έχει τόσους νευρώνες όσο και το πλήθος των χαρακτηριστικών, δηλαδή 8. Η στοιβάδα εξόδου έχει τόσους νευρώνες όσος και το πλήθος των ζητούμενων κλάσεων, δηλαδή 10. Το πρόβλημα ανάγεται στον καθορισμό του πλήθους των νευρώνων στις κρυφές στοιβάδες. Ενδεικτικά το μοντέλο είναι:



```
# model definition
model = Sequential()

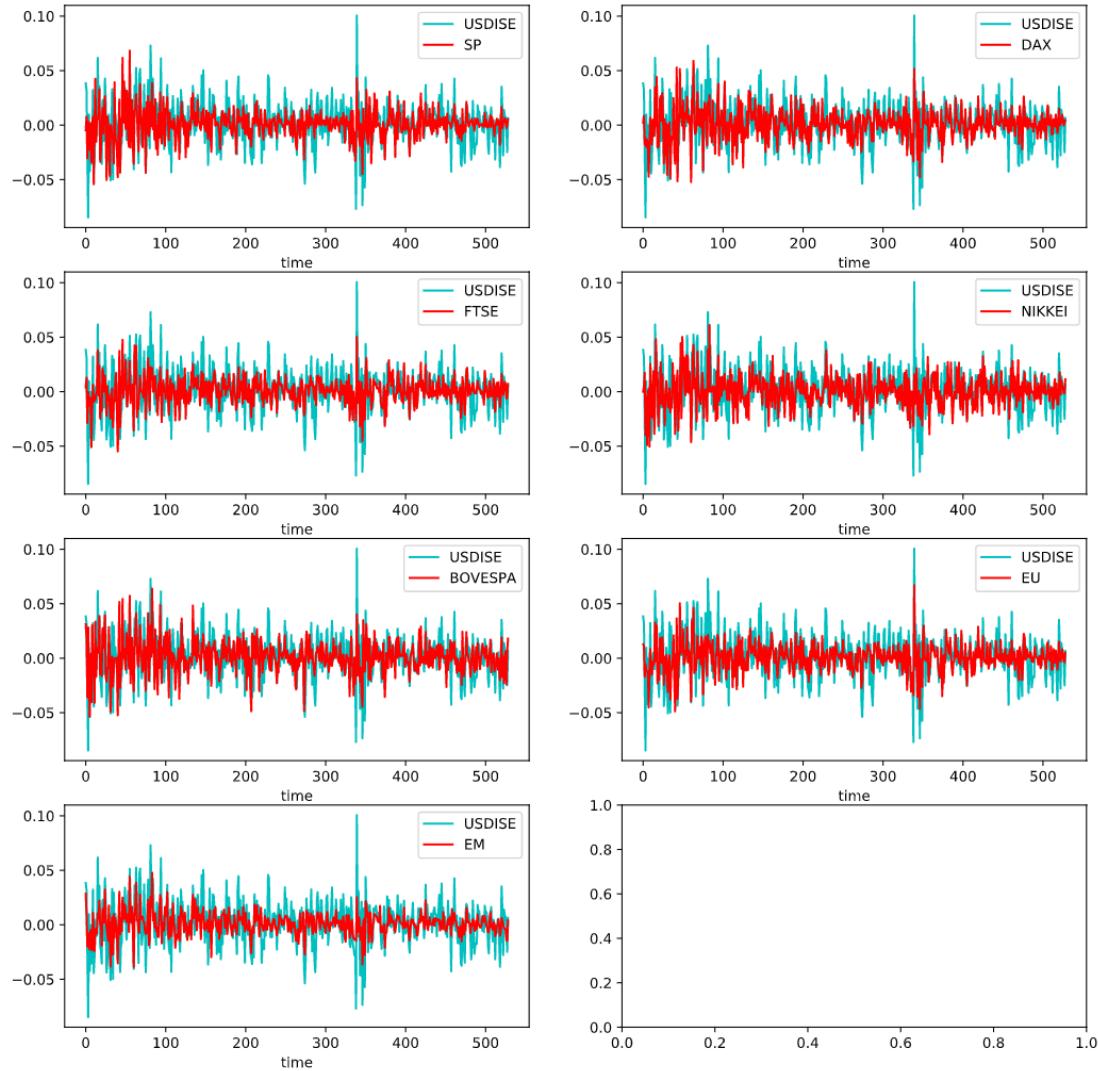
# Input Layer-----
model.add(Dense(10, input_dim=8, activation='relu')) # 1st Hidden layer
model.add(Dense(13, activation='relu')) # 2nd Hidden layer
model.add(Dense(len(CLASSES), activation='softmax')) # Output layer

model.summary()
Model: "sequential_1"
Layer (type)          Output Shape         Param #
=====
dense_1 (Dense)      (None, 10)           90
dense_2 (Dense)      (None, 13)           143
dense_3 (Dense)      (None, 10)           140
=====
Total params: 373
Trainable params: 373
Non-trainable params: 0

▶ MI 8+8
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['categorical_accuracy'])

▶ MI 8+8
epochs = 10
batch_size = 30
history = model.fit(xtr, ytr, validation_data=(xva, yva), epochs=epochs, batch_size=batch_size, verbose=0)
```

Αρχικά, έγιναν διάφορες δοκιμές, παραμετροποιώντας κάθε φορά το πλήθος των νευρώνων, το batch size και τα epochs. Σε αυτό το σημείο να αναφερθεί πως έγινε χρήση δύο στατιστικών δεικτών για την αξιολόγηση των μοντέλων, Micro f1-score και Matthews Correlation Coefficient (MCC). Παρακάτω παρουσιάζονται ενδεικτικά 10 δοκιμές με τα γραφήματά και τα αποτελέσματα τους.



Εικόνα 28 Plotting indices

Επόμενο βήμα είναι η κανονικοποίηση των δεδομένων. Χρησιμοποιήσαμε 2 scalers, ένα για τους 7 δείκτες και ένα για τον δείκτη στόχο, ώστε να μπορέσουμε στο τέλος να αντιστρέψουμε την τιμή του δείκτη στόχου ατομικά.



```
# Data Standardization.
# Scaler only for USDIKE.
usd_scaler = StandardScaler()
usdX = usd_scaler.fit_transform(np.reshape(data["USDIKE"].values , (data.shape[0], 1)))
# Scaler for the other attributes.
scaler = StandardScaler()
dataX = scaler.fit_transform(np.array(data.iloc[:,1:]))

# To DataFrame
standardized_data = data.copy()
standardized_data.iloc[:,0] = usdX[:]
standardized_data.iloc[:,1:] = dataX[:, :]
standardized_data.head(5)

USDIKE      SP      DAX      FTSE      NIKKEI      BOVESPA      EU      EM
0   1.736028 -0.378457  0.099118  0.264087 -0.021632  1.916382  0.935166  2.622395
1   1.426363  0.503267  0.527318  0.969910  0.257529  1.139329  0.831114  0.745067
2  -1.317912 -2.202572 -1.270325 -2.303040  1.138142 -2.332096 -1.346950 -1.991118
3  -4.071492  0.192382 -0.852733 -0.078971 -2.708397  1.732288 -0.464498 -1.934886
4   0.381101 -1.570534 -1.409802 -1.042262 -0.321653 -0.677085 -0.880562 -0.830304
```

Εικόνα 29 Standardization

Τέλος, μετατρέπουμε τη χρονοσειρά σε πρόβλημα supervised με μορφή [samples, steps, features].

```
# Function to transform the initial data into timeseries data.
def timeseries_to_supervised(df, n_in, n_out):
    agg = pd.DataFrame()

    for i in range(n_in, 0, -1):
        df_shifted = df.shift(i).copy()
        df_shifted.rename(columns=lambda x: ('%s(t-%d)' % (x, i)), inplace=True)
        agg = pd.concat([agg, df_shifted], axis=1)

    for i in range(0, n_out):
        df_shifted = df.shift(-i).copy()
        if i == 0:
            df_shifted.rename(columns=lambda x: ('%s(t)' % (x)), inplace=True)
        else:
            df_shifted.rename(columns=lambda x: ('%s(t+%d)' % (x, i)), inplace=True)
        agg = pd.concat([agg, df_shifted], axis=1)

    agg.dropna(inplace=True)
    return agg
```

Εικόνα 30 timeseries_to_supervised function

Στο παρακάτω screenshot βλέπουμε, ενδεικτικά, ότι προκύπτει steps = 3 |[(t-3), (t-2), (t-1)]| και features = 8 |[USDIKE, SP, DAX, FTSE, NIKKEI, BOVESPA, EU, EM]|, γιατί συμπεριλαμβάνεται και η προηγούμενη τιμή του στόχου.



Ερώτημα 2: Time series prediction

Σε αυτό το σημείο ήρθε η ώρα να φτιάξουμε τα μοντέλα μας. Δημιουργούνται δύο ειδών νευρωνικών δικτύων, ένα MLP και ένα RNN. Για την αξιολόγηση των μοντέλων χρησιμοποιούνται ο στατιστικός δείκτης R Squared. Ως loss function γίνεται χρήση του MSE, ενώ για activation function χρησιμοποιείται η tanh.

A) Time series prediction using MLP

Η κατασκευή του μοντέλου είναι απλή, ορίσαμε 2 hidden layer και δοκιμάσαμε διάφορες τιμές για τα units. Όπως αναφέρθηκε, το MLP δέχεται διανύσματα ως είσοδο, οπότε το μέγεθος εισόδου για steps = 3 είναι $8 \times 4 = 24$. Ορίζουμε batch size ίσο με 1 και δοκιμάζουμε διάφορες τιμές για τα epochs. Έτσι, ενδεικτικά το μοντέλο είναι:

```
batch_size = 1
model = Sequential()

model.add(Dense(50, input_dim=xtr.shape[1], activation="tanh"))
model.add(Dense(45, activation="tanh"))
model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
model.summary()

del: "sequential_1"

layer (type)          Output Shape         Param #
===== =====
dense_1 (Dense)      (None, 50)           1250
dense_2 (Dense)      (None, 45)            2295
dense_3 (Dense)      (None, 20)            920
dense_4 (Dense)      (None, 1)             21
=====
total params: 4,486
trainable params: 4,486
non-trainable params: 0

Ml  =>
history = model.fit(xtr,ytr, epochs=100, batch_size=batch_size, verbose=0)
```

Εικόνα 33 MLP



```
# Predicting
trainPredict = model.predict(xtr, batch_size=batch_size)
testPredict = model.predict(xte, batch_size=batch_size)

# Invert predictions
ytr2d = np.reshape(ytr, (ytr.shape[0], 1))
yte2d = np.reshape(yte, (yte.shape[0], 1))
trainPredict = usd_scaler.inverse_transform(trainPredict)
trainY = usd_scaler.inverse_transform(ytr2d)
testPredict = usd_scaler.inverse_transform(testPredict)
testY = usd_scaler.inverse_transform(yte2d)

# Calculate error
print("Train MSE: ", mean_squared_error(trainY, trainPredict))
print("Test MSE: ", mean_squared_error(testY, testPredict), '\n')

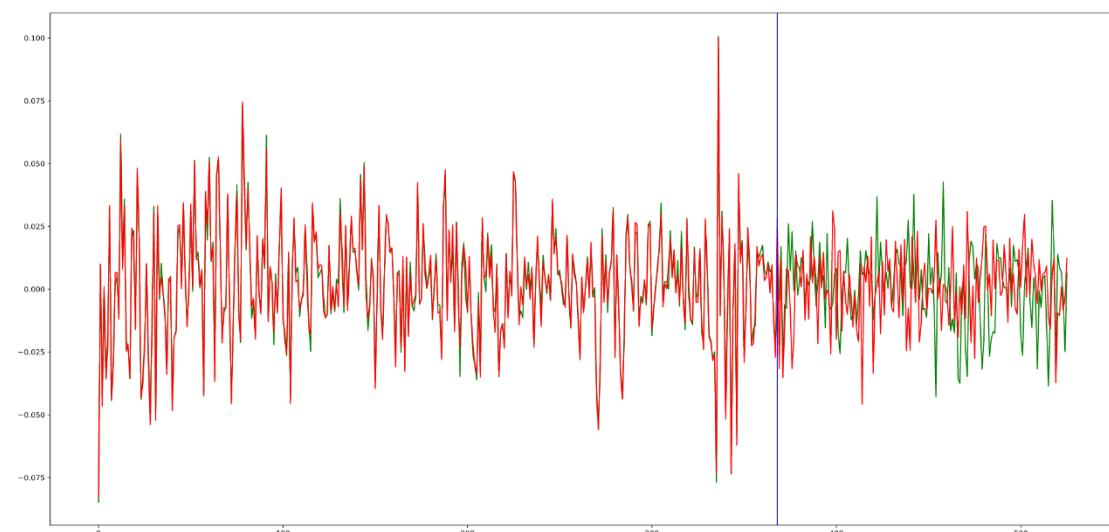
# Calculate R2
print("Train R2: ", r2_score(trainY, trainPredict))
print("Test R2: ", r2_score(testY, testPredict), '\n')

Train MSE:  9.373851777774744e-06
Test MSE:  0.00043381391430089575

Train R2:  0.9824891575951669
Test R2:  -0.8283317732757023
```

Εικόνα 34 Calculating scores

Σε αυτό το σημείο να αναφερθεί πως οι τιμές του R Squared που είναι πιο κοντά στο 1 είναι καλύτερες. Αυτό σημαίνει ότι το προβλεπόμενο σήμα τείνει να ταυτιστεί στο πραγματικό. Αν το μοντέλο προβλέπει συνέχεια τη μέση τιμή των πραγματικών τιμών, η τιμή του R Squared θα είναι 0. Οτιδήποτε χειρότερο από αυτό, έχει ως αποτέλεσμα αρνητικές τιμές.



Εικόνα 35 Plotting the predicted signal (red) on top of the real signal (green)



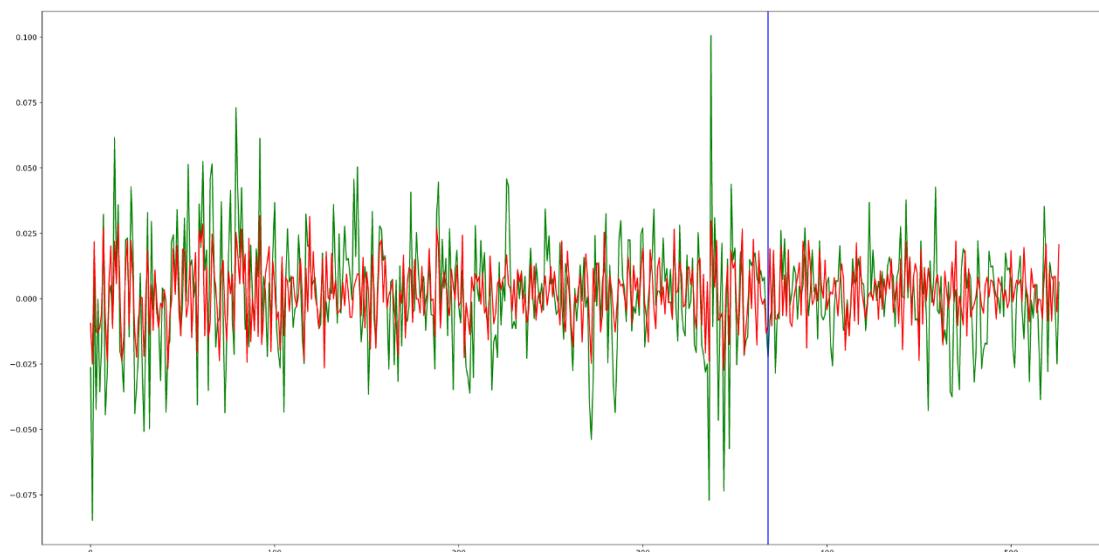
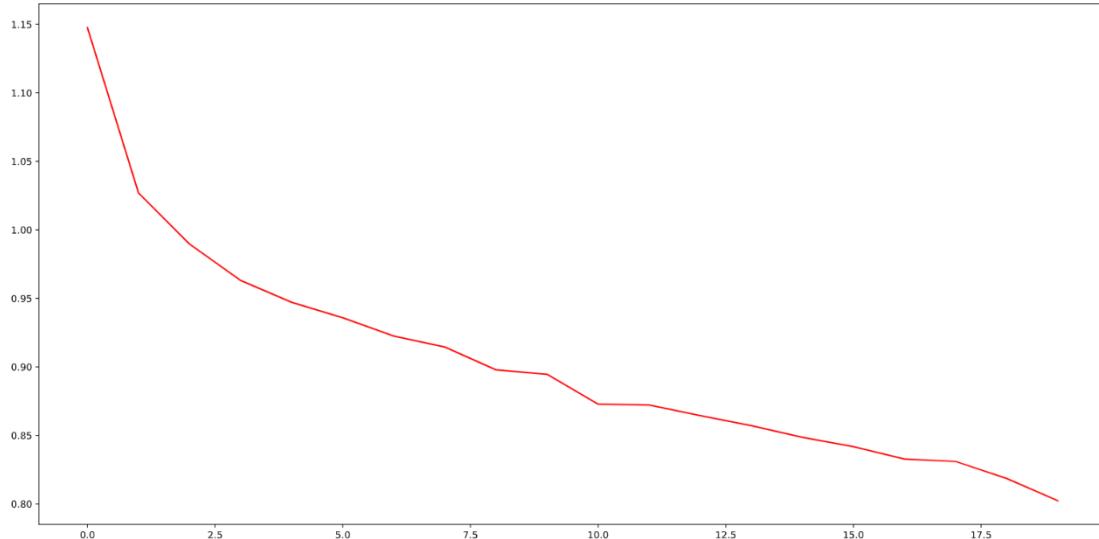
Όπως παρατηρούμε το παραπάνω διάγραμμα, το προβλεπόμενο σήμα -αριστερά της μπλε γραμμής- «ταιριάζει» σε μεγάλο βαθμό με το πραγματικό. Για αυτό, η τιμή του R Squared στο train set είναι 0,98 δηλαδή κοντά στο 1. Αντιθέτως, το προβλεπόμενο σήμα -δεξιά της μπλε γραμμής- απέχει αρκετά από το πραγματικό. Αυτό έχει ως αποτέλεσμα μια αρνητική τιμή για το R Squared.

Παρακάτω ακολουθούν 10 ενδεικτικές δοκιμές μοντέλων με διαφορετικές υπερπαραμέτρους:



• Δοκιμή #1

Layer 1	Layer 2	Epochs	Batch Size	Samples	Steps	Features
13	10	20	1	368	2	7



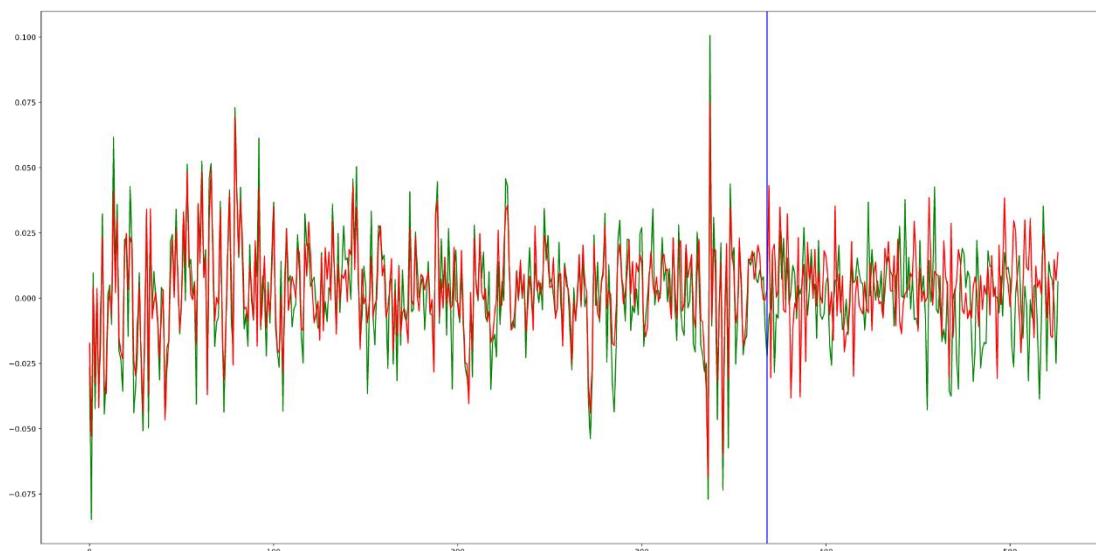
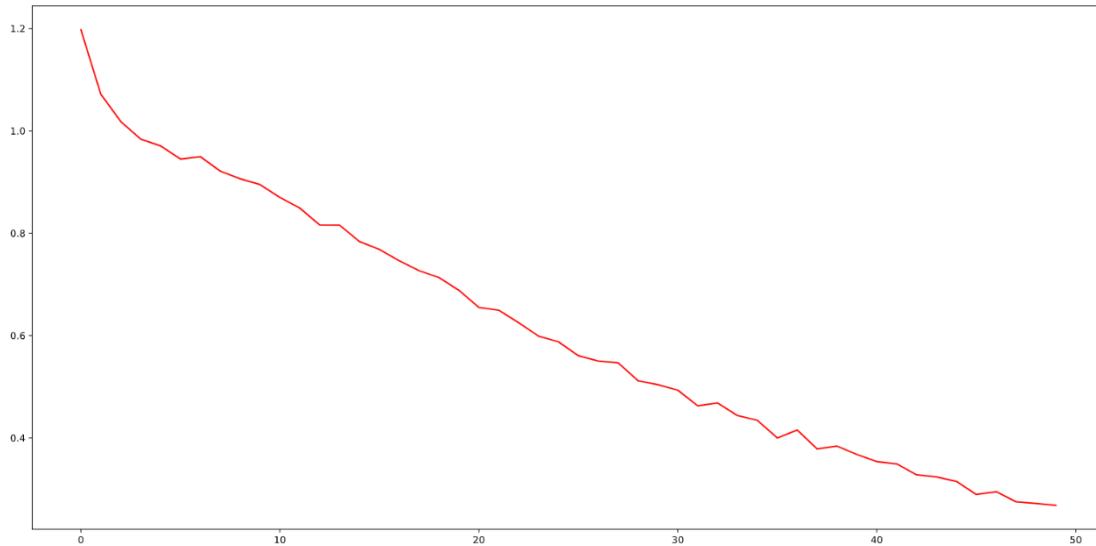
Train MSE: 0.0003426876594998246
Test MSE: 0.0002567626310855533

Train R2: 0.36055545091611163
Test R2: -0.07401332177585118



- Δοκιμή #2

Layer 1	Layer 2	Epochs	Batch Size	Samples	Steps	Features
26	20	50	1	368	2	7



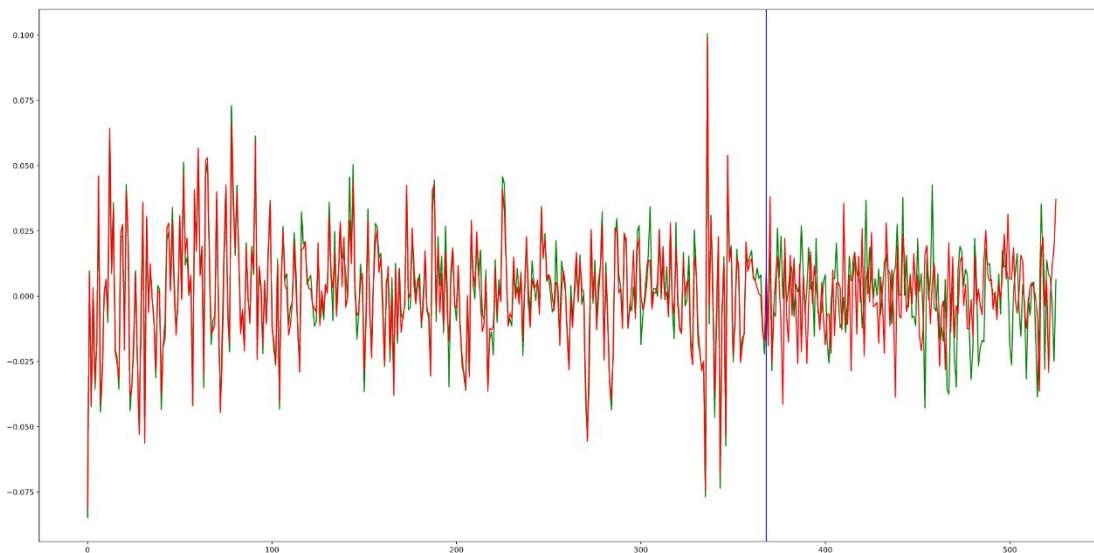
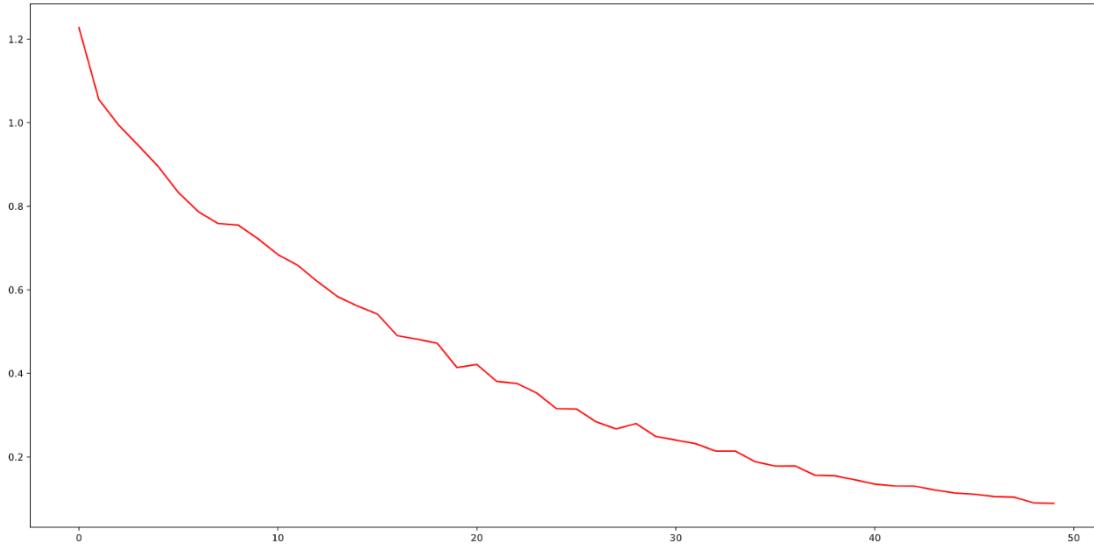
Train MSE: 9.744802079877926e-05
Test MSE: 0.00044353426663317706

Train R2: 0.8181650141422012
Test R2: -0.8552610596570454



- Δοκιμή #3

Layer 1	Layer 2	Epochs	Batch Size	Samples	Steps	Features
26	28	50	1	368	3	7



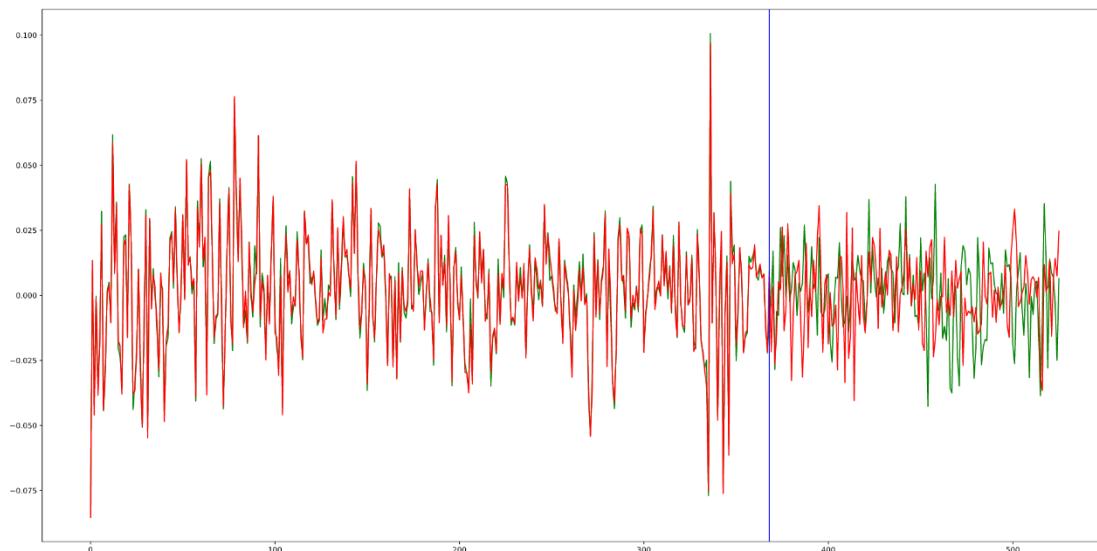
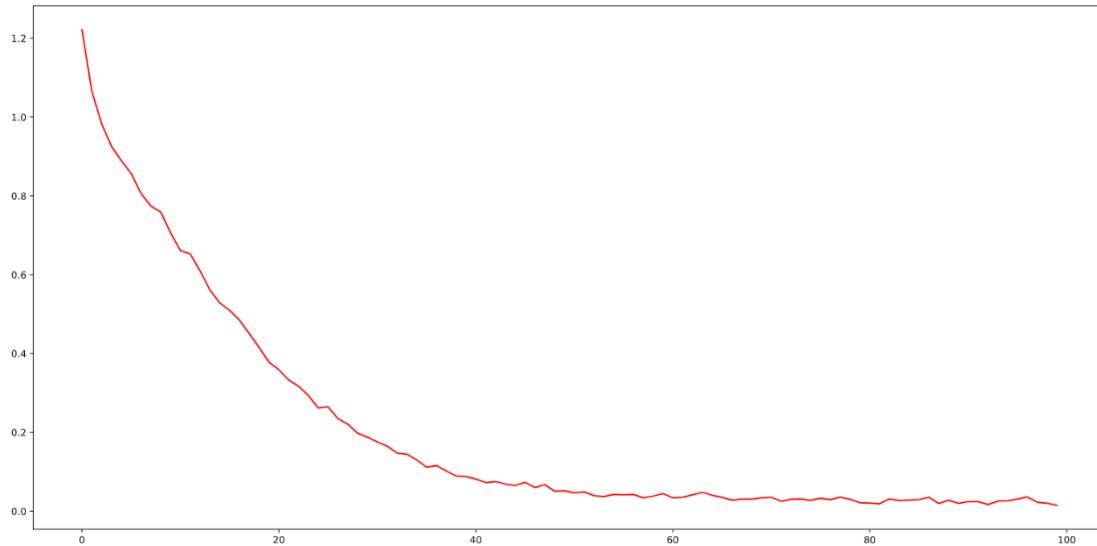
Train MSE: 2.6200413758609016e-05
Test MSE: 0.0003828804251432508

Train R2: 0.9510562651143887
Test R2: -0.6136698791297137



• Δοκιμή #4

Layer 1	Layer 2	Epochs	Batch Size	Samples	Steps	Features
35	26	100	1	368	3	7



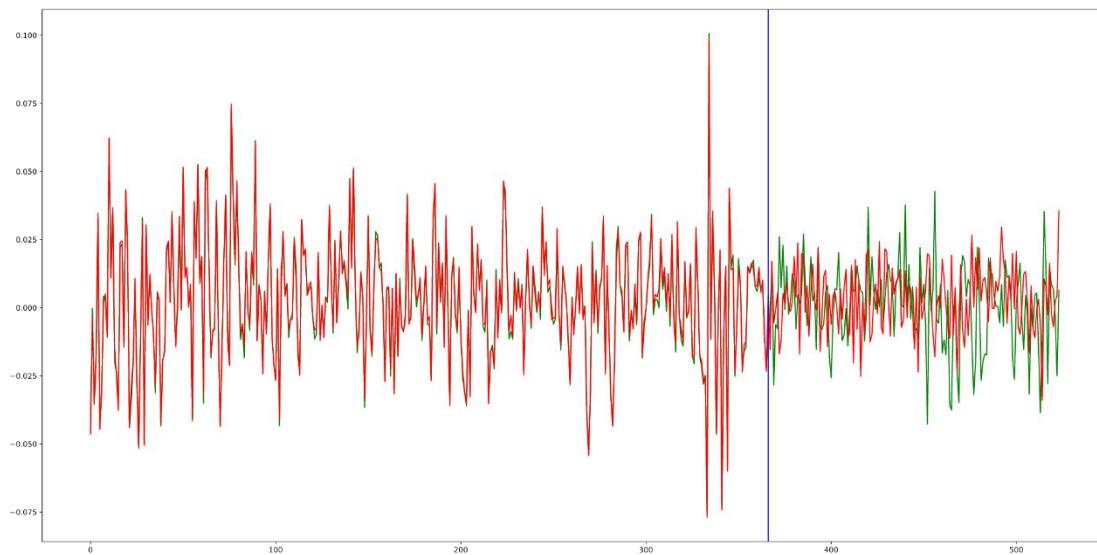
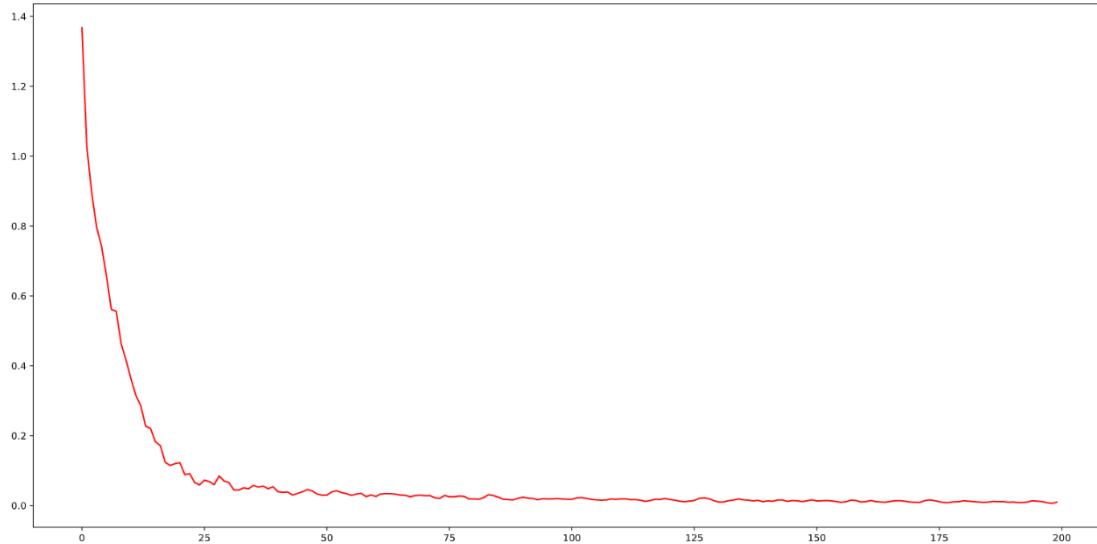
Train MSE: 7.898441546763162e-06
Test MSE: 0.00041124856696156766

Train R2: 0.9852453006034209
Test R2: -0.7332289189055683



• Δοκιμή #5

Layer 1	Layer 2	Epochs	Batch Size	Samples	Steps	Features
50	46	200	1	366	5	7



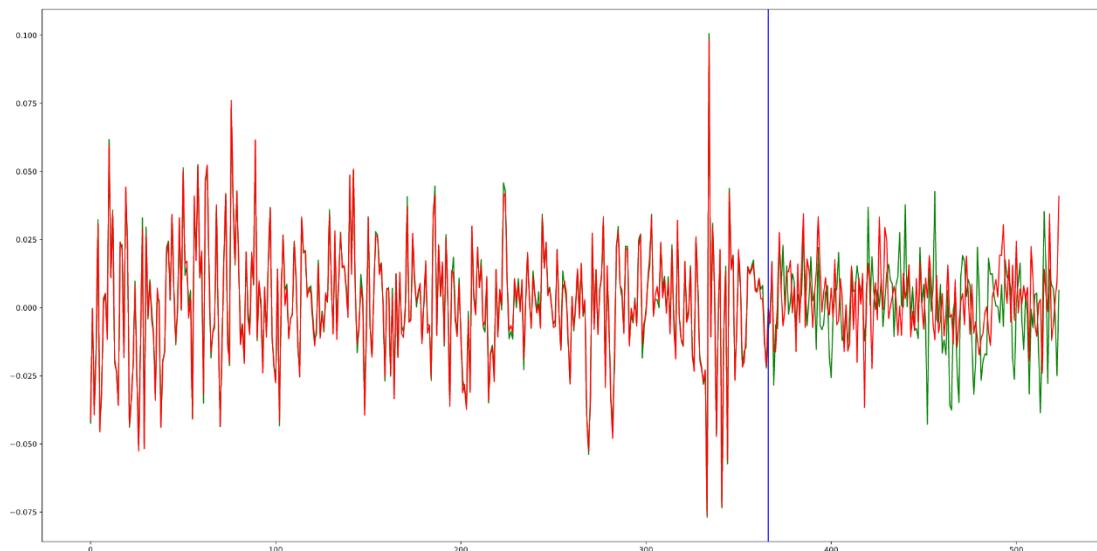
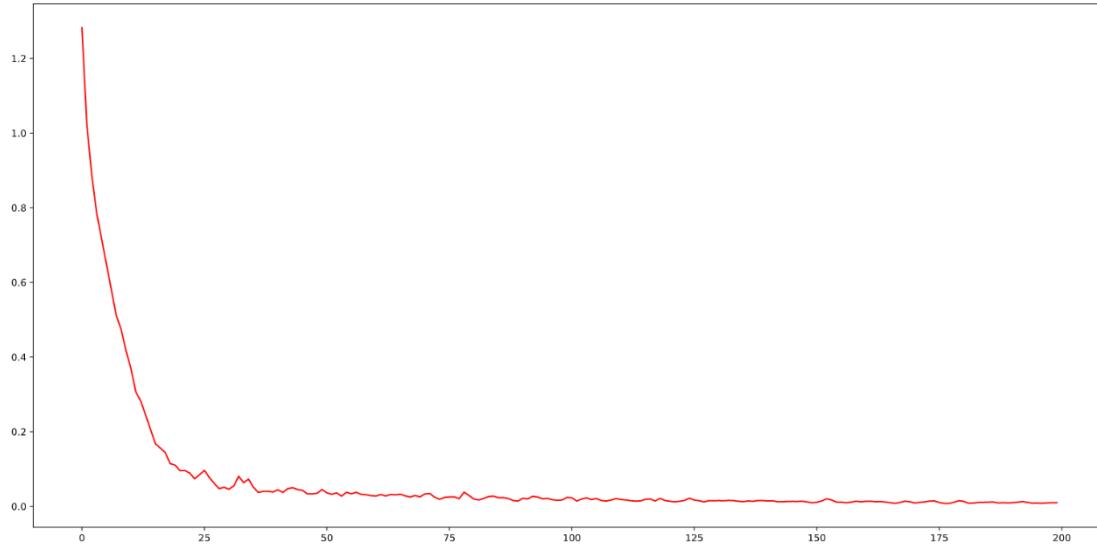
Train MSE: 4.504097239128958e-06
Test MSE: 0.0003531612266217924

Train R2: 0.9912976618747623
Test R2: -0.48841673915001627



• Δοκιμή #6

Layer 1	Layer 2	Epochs	Batch Size	Samples	Steps	Features
52	57	200	1	366	5	7



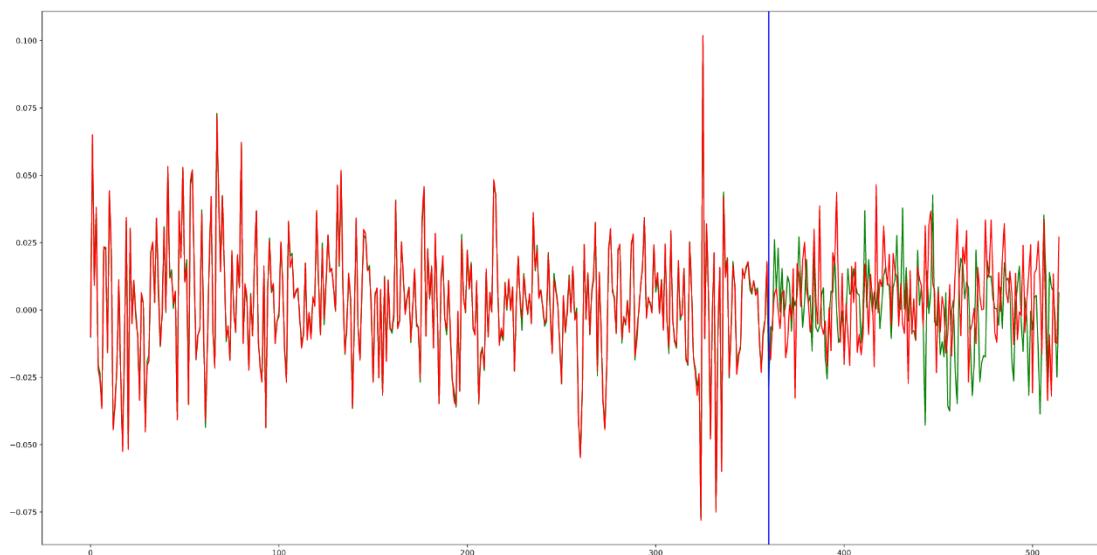
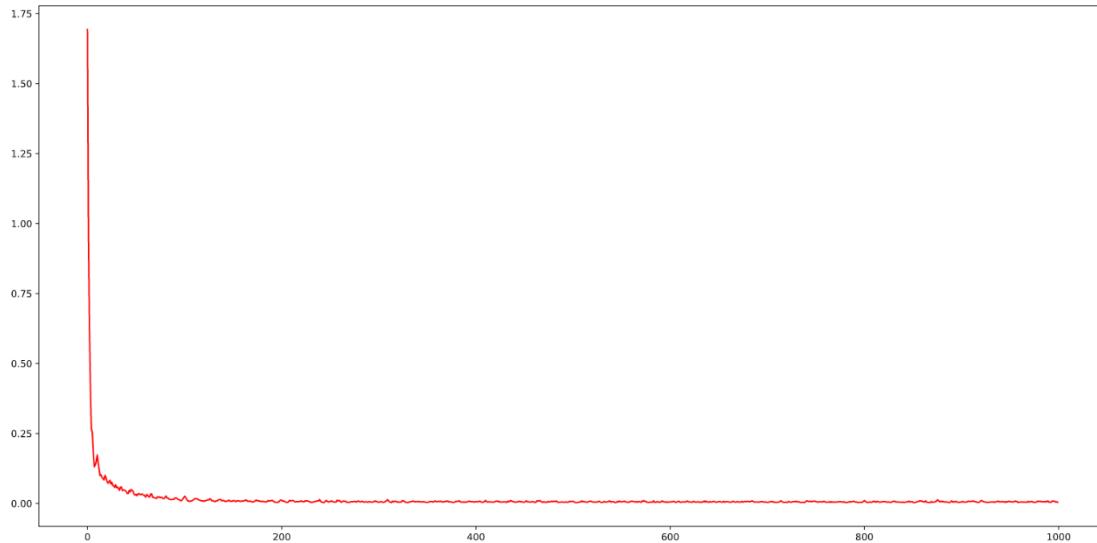
```
Train MSE: 4.20970988955495e-06
Test MSE: 0.00031532388398123755

Train R2: 0.9918664458329615
Test R2: -0.3289492497829902
```



• Δοκιμή #7

Layer 1	Layer 2	Epochs	Batch Size	Samples	Steps	Features
142	113	1000	1	360	14	7



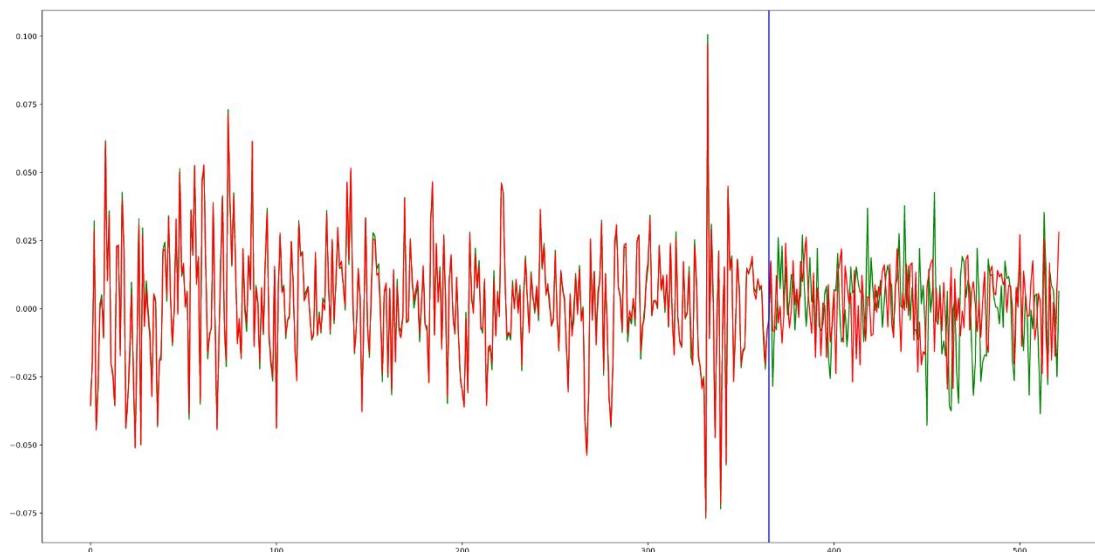
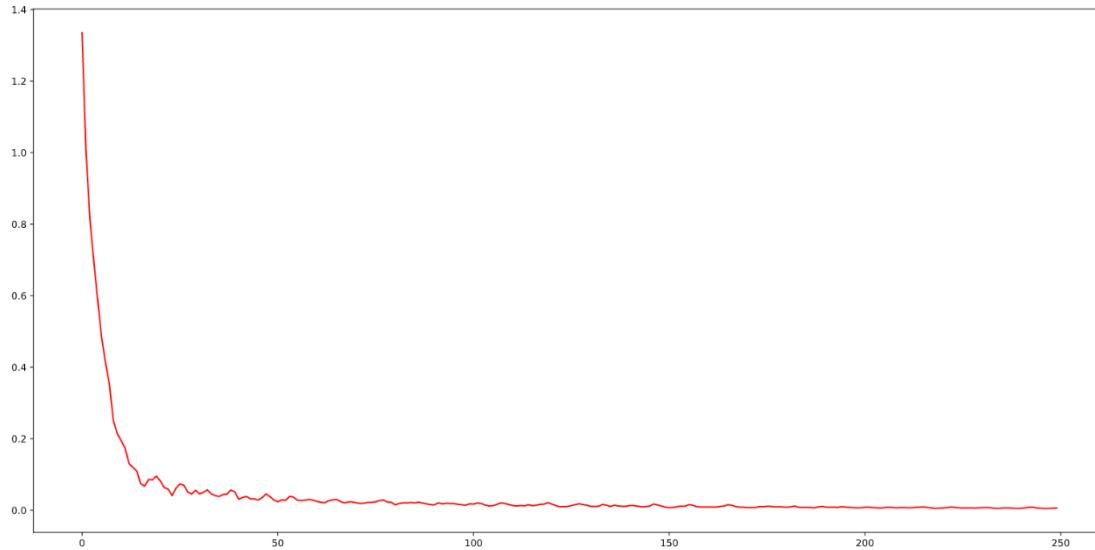
Train MSE: 1.67242149241591e-06
Test MSE: 0.00044147057179157276

Train R2: 0.9966898780544563
Test R2: -0.842873395944467



• Δοκιμή #8

Layer 1	Layer 2	Epochs	Batch Size	Samples	Steps	Features
55	63	250	1	365	7	7



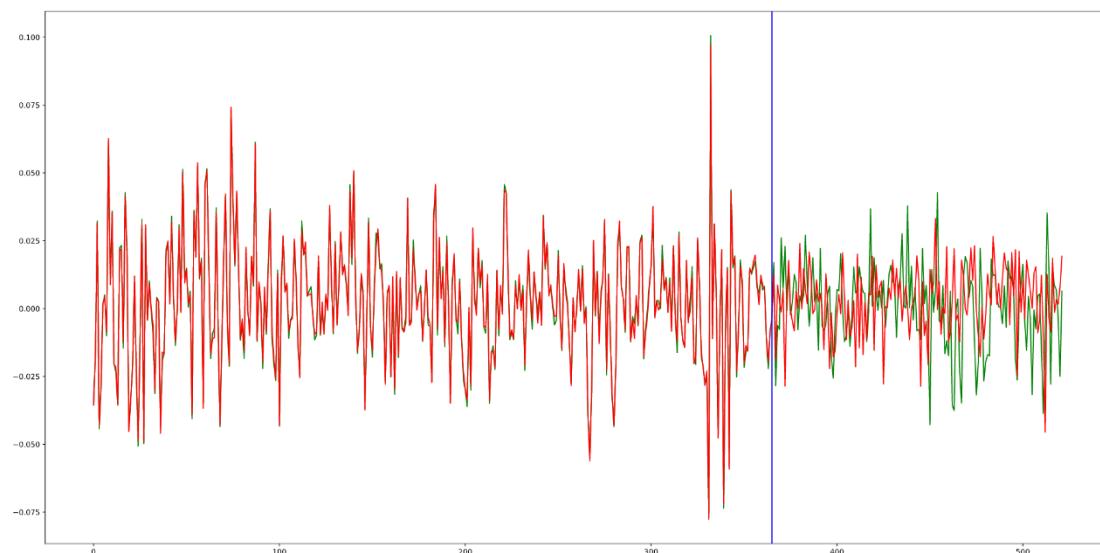
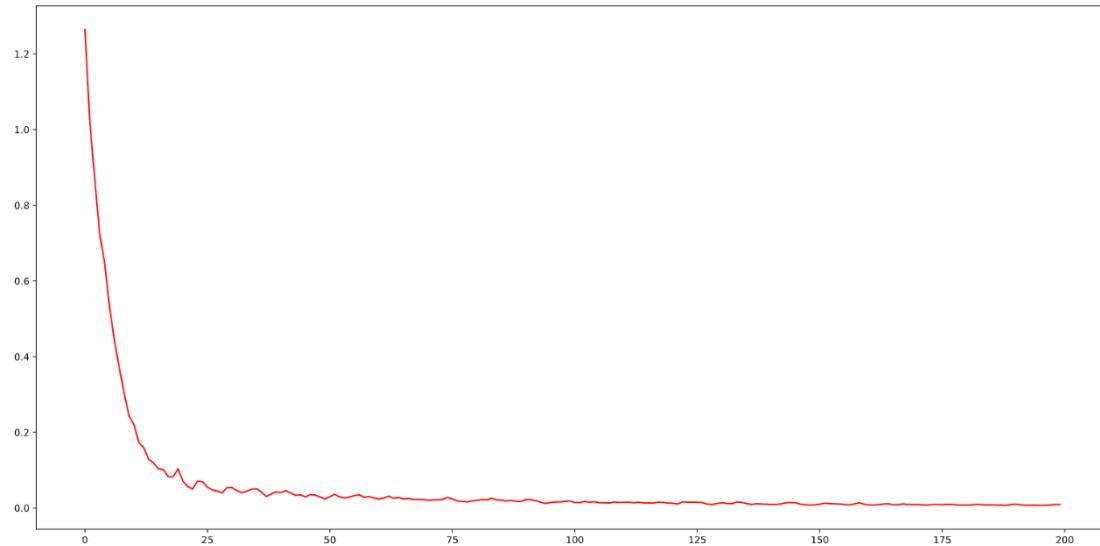
Train MSE: 2.5354295643501706e-06
Test MSE: 0.0003678330251214281

Train R2: 0.9950657817556741
Test R2: -0.5436699495164878



• Δοκιμή #9

Layer 1	Layer 2	Epochs	Batch Size	Samples	Steps	Features
53	59	200	1	365	7	7



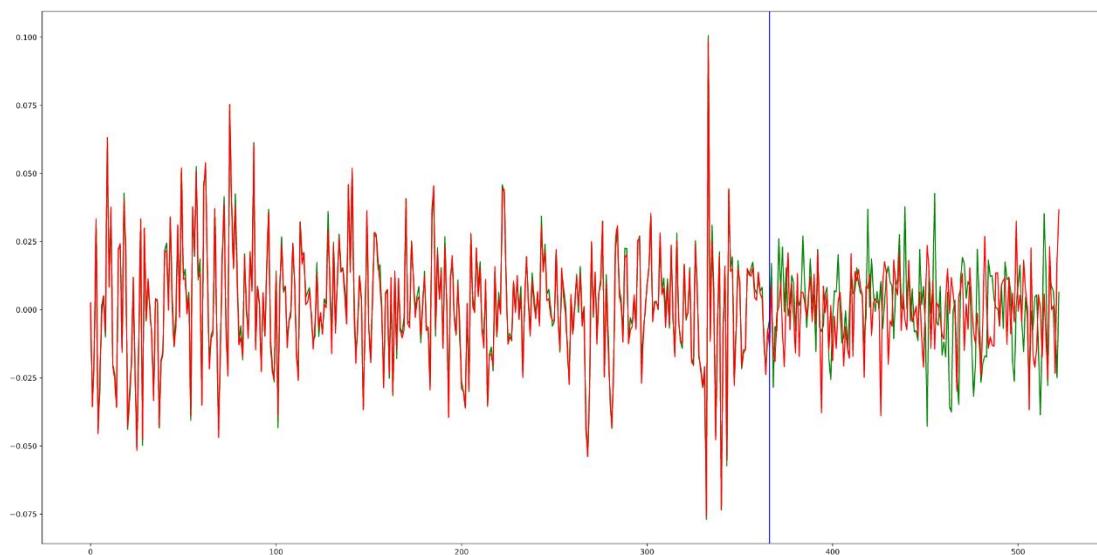
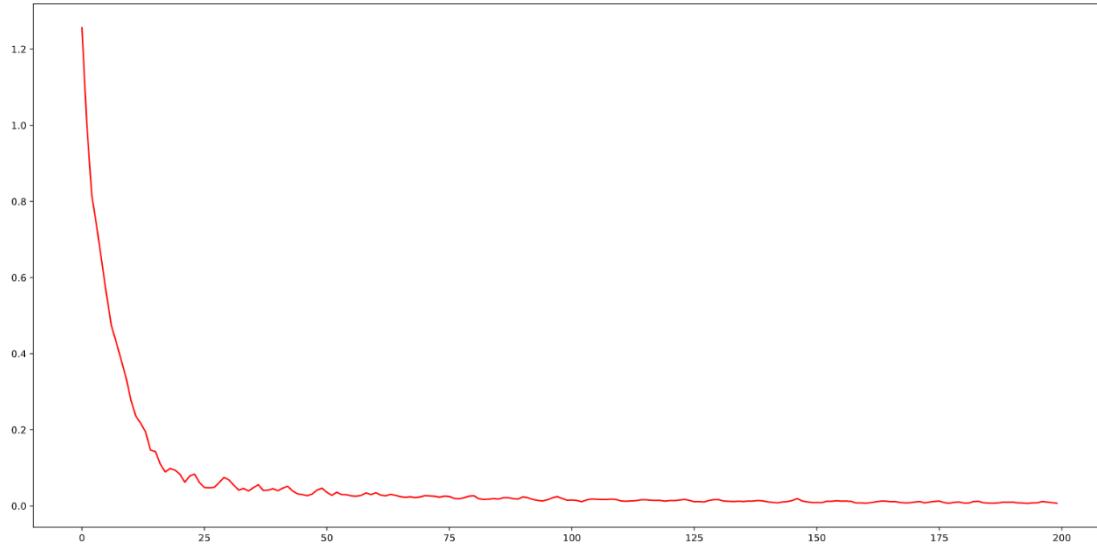
Train MSE: 3.008942907747695e-06
Test MSE: 0.0003736947141928438

Train R2: 0.9941442739327886
Test R2: -0.5682694624883493



- Δοκιμή #10

Layer 1	Layer 2	Epochs	Batch Size	Samples	Steps	Features
54	56	200	1	366	5	7



Train MSE: 5.315116746375294e-06
Test MSE: 0.0003535603552419879

Train R2: 0.9896281891711609
Test R2: -0.4837724142557873



Η δομή του μοντέλου ακολουθεί τη δομή της προηγούμενης υλοποίησης, χρησιμοποιώντας 2 κρυφές στοιβάδες.

```
# Building a model with SimpleRNN
batch_size = 1
model = Sequential()

model.add(SimpleRNN(units=150, batch_input_shape=(batch_size, xtr.shape[1], xtr.shape[2]), activation="relu", return_sequences=True))
model.add(Dense(120, activation="relu"))
model.add(Dense(1))

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

model.summary()

Model: "sequential_116"
Layer (type)                 Output Shape              Param #
simple_rnn_97 (SimpleRNN)    (1, 33, 150)           23850
dense_243 (Dense)            (1, 33, 120)            18120
dense_244 (Dense)            (1, 33, 1)             121
Total params: 42,091
Trainable params: 42,091
Non-trainable params: 0
```

RNN model

Το RNN δέχεται είσοδο τριών διαστάσεων, της μορφής [samples, steps, features]. Αυτό προϋποθέτει να μετασχηματισθούν τα δεδομένα, πρώτα με τη χρήση της συνάρτησης timeseries_to_supervised με `n_in = 1` και στη συνέχεια στην προαναφερθείσα μορφή.

```
# RNN Implementation

n_in = 1
n_out = 1
ts = timeseries_to_supervised(standardized_data, n_in, n_out)
print(ts.head(5))

JSDISE(t-1)    SP(t-1)   DAX(t-1)   FTSE(t-1)  NIKKEI(t-1)  BOVESPA(t-1) \
1     1.736028   -0.378457   0.099118   0.264087   -0.021632    1.916382
2     1.426363   0.503267   0.527318   0.969910   0.257529    1.139329
3    -1.317912   -2.202572  -1.270325  -2.303040   1.138142   -2.332096
4    -4.071492   0.192382  -0.852733  -0.078971   -2.708397    1.732288
5     0.381101   -1.570534  -1.409802  -1.042262   -0.321653   -0.677085

EU(t-1)    EM(t-1)   USDISE(t)    SP(t)      DAX(t)   FTSE(t)  NIKKEI(t) \
1     0.935166   2.622395   1.426363   0.503267   0.527318   0.969910   0.257529
2     0.831114   0.745067  -1.317912  -2.202572  -1.270325  -2.303040   1.138142
3    -1.346950  -1.991118  -4.071492   0.192382  -0.852733  -0.078971  -2.708397
4    -0.464498  -1.934886   0.381101  -1.570534  -1.409802  -1.042262  -0.321653
5     -0.880562  -0.830304  -2.073183  -1.661734  -0.975783  -0.437699  -3.310467

BOVESPA(t)   EU(t)    EM(t)
1     1.139329  0.831114  0.745067
2    -2.332096 -1.346950 -1.991118
3     1.732288 -0.464498 -1.934886
4    -0.677085 -0.880562 -0.830304
5     -3.468858 -0.992681 -2.239599
```

Using `timeseries_to_supervised` with `n_in=1`



```
# Exclude all (t) values from x
X = ts.loc[:, list(map(lambda x: '(t)' not in x, ts.columns))].values
y = ts['USDISE(t)'].values
print(X.shape, y.shape)

(528, 8)

# Split data into train and test
len_data = X.shape[0]
train_size = int(len_data * .5)
test_size = len_data - train_size

xtr, ytr = X[:train_size, :], y[:train_size]
xte, yte = X[train_size:, :], y[train_size:]
print(xtr.shape, ytr.shape)
print(xte.shape, yte.shape)

(264, 8) (264,)
(264, 8) (264,)
```

Train and Test sets.

```
steps = 3
train_samples = int(train_size/steps)
test_samples = int(test_size/steps)
print(samples)
features_in = 8
features_out = 1
xtr = np.reshape(xtr, (train_samples, steps, features_in))
ytr = np.reshape(ytr, (train_samples, steps, features_out))
print(xtr.shape, ytr.shape)
xte = np.reshape(xte, (test_samples, steps, features_in))
yte = np.reshape(yte, (test_samples, steps, features_out))
print(xte.shape, yte.shape)

153
(88, 3, 8) (88, 3, 1)
(88, 3, 8) (88, 3, 1)
```

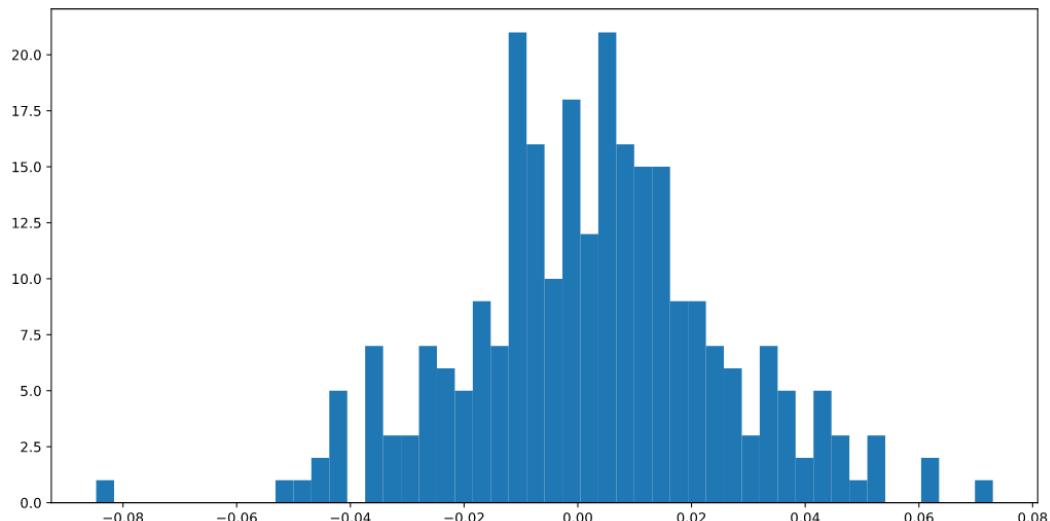
Data transformation to [samples, steps, features].

Ο αριθμός των samples εξαρτάται κάθε φορά από την επιλογή των steps, ενώ ο αριθμός των features παραμένει σταθερός και ίσος με 8. Επιπλέον, η επιλογή των steps εξαρτάται από τον αριθμό των train και test sizes. Πρέπει δηλαδή η τιμή του steps να είναι κοινός διαιρέτης των μεγεθών αυτών. Έτσι, για ευκολία το train και test set θεωρήθηκε ίδιο και ίσο με το πρώτο και το δεύτερο μισό, αντίστοιχα, των δεδομένων. Έχοντας, λοιπόν, sizes ίσα με 264, οι πιθανές τιμές των steps είναι 1, 2, 3, 4, 6, 8, 11, 12, 22, 24, 33, 44, 66, 88, 132, 264.

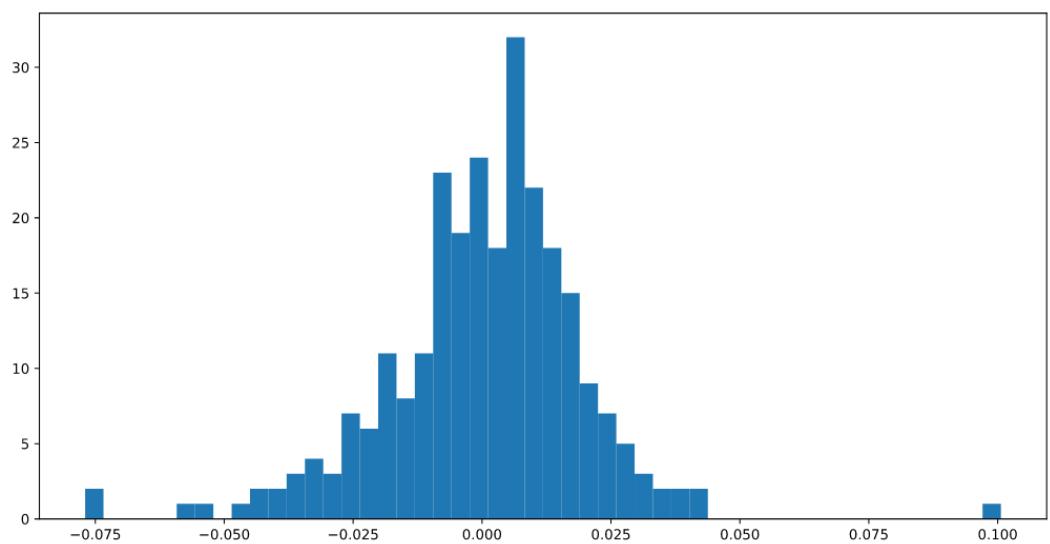


```
train_var = np.var(trainY)
test_var = np.var(testY)
print(f"train variance: {train_var}")
print(f"test variance: {test_var}")
print(f"train variance / test variance: {train_var/test_var}")

train variance: 0.0005210918860559889
test variance: 0.000371530007757081
train variance / test variance: 1.4025566580794102
```



Train set distribution



Test set distribution

Παρατηρούμε ότι χωρίζοντας τα μισά δεδομένα σε train set και τα άλλα μισά σε test set, προκύπτει ότι η διακύμανση του πρώτου είναι σχεδόν μιάμιση φορά μεγαλύτερη από αυτή της δεύτερης. Σε γενικές γραμμές, θέλουμε οι διακυμάνσεις να είναι σχεδόν ίδιες. Οπότε, μια καταλληλότερη επιλογή των train και test set να βοηθούσε. Οι δοκιμές και τα αποτελέσματα της υλοποίησης με MLP προήλθαν από 70% του dataset ως train set, ενώ το



υπόλοιπο 30% ως test set. Με αυτόν τον χωρισμό, παρατηρούμε ακόμα μεγαλύτερη διαφορά στις διακυμάνσεις.

```
train_var = np.var(trainY)
test_var = np.var(testY)
print(f"train variance: {train_var}")
print(f"test variance: {test_var}")
print(f"train variance / test variance: {train_var/test_var}")

train variance: 0.0004943914413937571
test variance: 0.0002367288107179419
train variance / test variance: 2.0884295405125637
```

Ένας ακόμα αίτιο της χαμηλής αποδοτικότητας είναι το μικρό μέγεθος του συνόλου δεδομένων για ένα τέτοιο πρόβλημα. Δίνοντας περισσότερα δείγματα, θα μπορούσαμε να δοκιμάσουμε και μεγαλύτερες τιμές στο steps, ώστε το δίκτυο να έχει long-term memory. Όμως, θα πρέπει να αντιμετωπιστεί άλλο ένα μεγάλο και συχνό πρόβλημα όταν “κοιτάμε” πολύ βαθιά στο παρελθόν. Ένα πρόβλημα που είναι γνωστό ως vanishing and exploding gradient. Αυτό εμφανίζεται όταν πολύ παλίες τιμές επηρεάζουν πολύ μελλοντικές τιμές. Μια λύση είναι η χρήση LSTM.

Τέλος, δεν υπάρχει γνώση στον τομέα του χρηματιστηρίου, άρα και πως παράγονται οι τιμές των δεικτών. Στην περίπτωση αυτής της εργασίας, δεν ξέρουμε αν οι 7 δείκτες που δίνονται επηρεάζουν κατά κάποιον τρόπο τον δείκτη στόχο. Θεωρητικά, αυτό συμβαίνει, αλλά δεν γνωρίζουμε περισσότερες λεπτομέρειες. Εκεί που θέλουμε να καταλήξουμε είναι πως οι παράγοντες που επηρεάζουν την τιμή του USD ISE είναι πολύ περισσότεροι από τους 7 δείκτες που έχουμε στην διάθεση μας. Έτσι, απαιτούνται παραπάνω πληροφορίες για μια καλύτερη πρόβλεψη, αυξάνοντας όμως τις διαστάσεις των δεδομένων, άρα και την υπολογιστική εργασία.