

### Εργασία 3

Αλεξία Τοπαλίδου: 1115201600286

Στέφανος Μπακλαβάς: 1115201700093

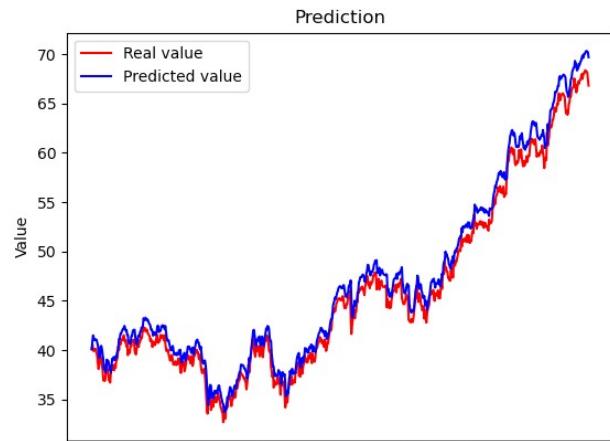
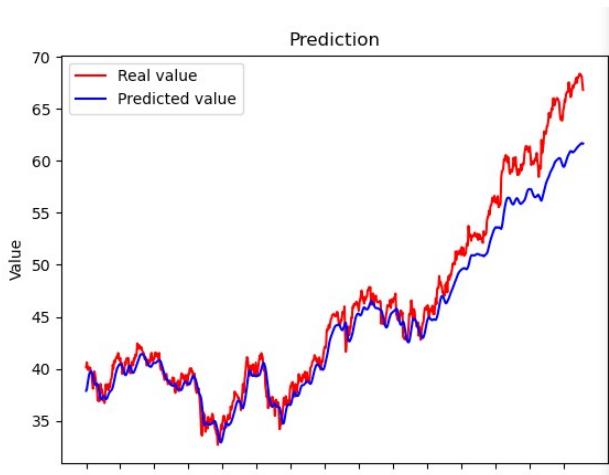
#### Ερώτημα A

Ο Αλγόριθμος κάνει εκπαίδευση με μία χρονοσειρά και με σύνολο χρονοσειρών. Αν για παράδειγμα δοθεί  $n=2$  τότε ο αλγόριθμος θα κάνει εκπαίδευση με την πρώτη χρονοσειρά και πρόβλεψη στην πρώτη, στη συνέχεια εκπαίδευση στη δεύτερη και πρόβλεψη στη δεύτερη και τέλος εκπαίδευση και με τις δύο χρονοσειρές μαζί και πρόβλεψη στην πρώτη και στη συνέχεια πρόβλεψη στη δεύτερη. Για την εκπαίδευση χρησιμοποιείται το 80% των τιμών της κάθε χρονοσειράς.

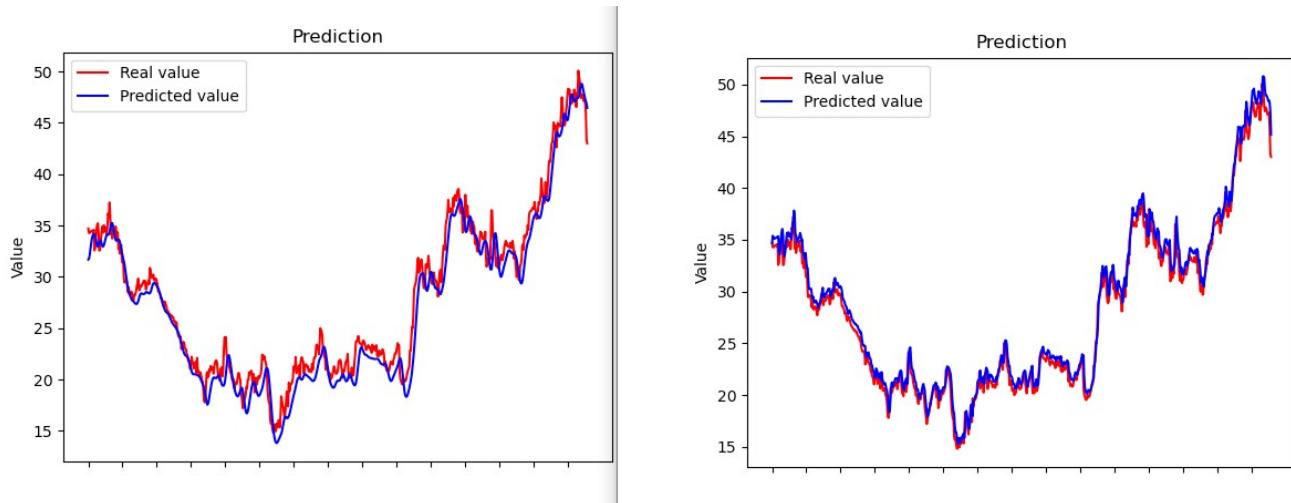
Παρακάτω παρουσιάζονται τα αποτελέσματα για το βέλτιστο μοντέλο του ερωτήματος A καθώς και αποτελέσματα για κάποια από τα μη βέλτιστα μοντέλα που δοκιμάστηκαν.

- Μετά από πειραματισμό καταλήξαμε πως το βέλτιστο μοντέλο ανάμεσα σε όσα δοκιμάστηκαν έχει 4 LSTM layers, 4 dropout layers και 1 output layer. Επίσης έχει epochs = 30, batch\_size = 32, units = 50, loss = mean\_squared\_error και feature Scaling τον MinMaxScaler. Σε αυτό το παράδειγμα δώσαμε  $n=10$ .

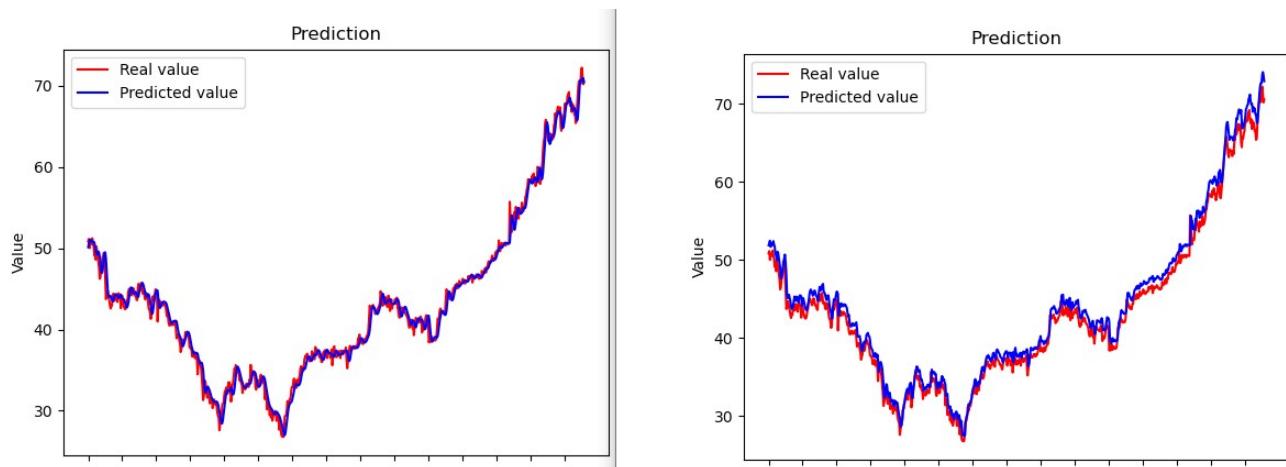
Εκπαίδευση με την πρώτη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά. Πρόβλεψη για την πρώτη χρονοσειρά.



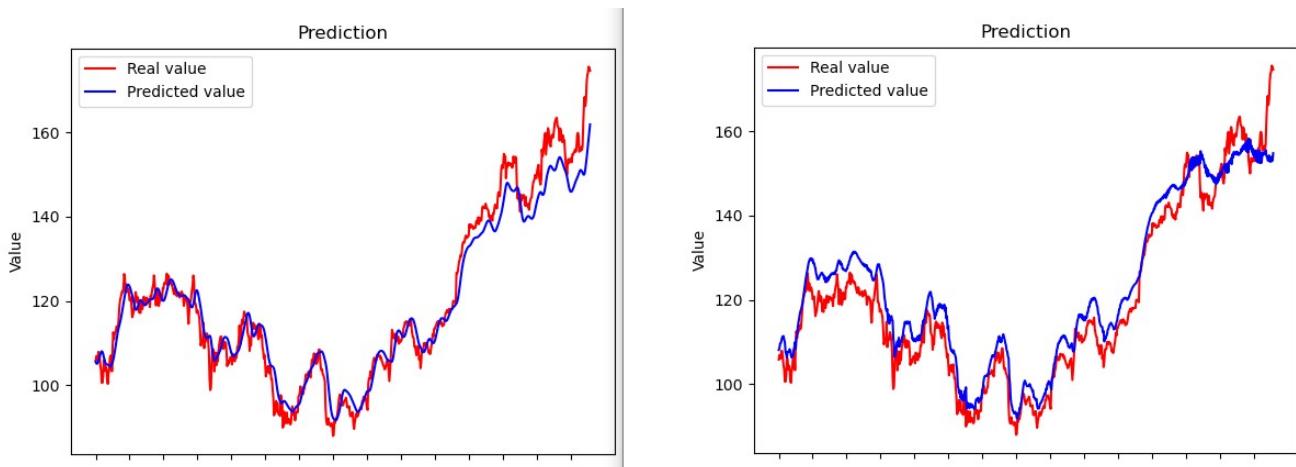
Εκπαίδευση με την δεύτερη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά. Πρόβλεψη για την δεύτερη χρονοσειρά.



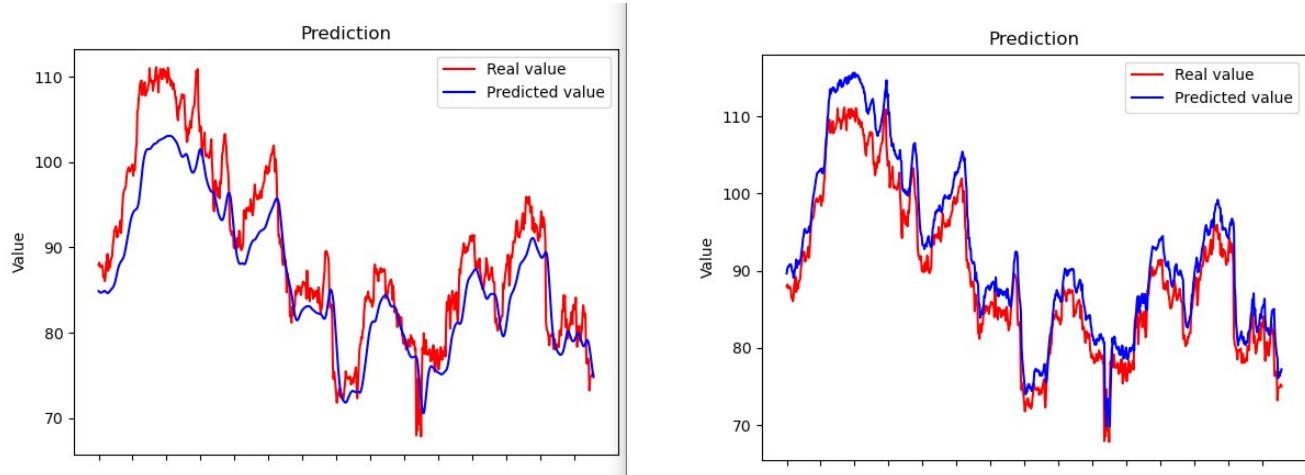
Εκπαίδευση με την τρίτη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά.  
Πρόβλεψη για την τρίτη χρονοσειρά.



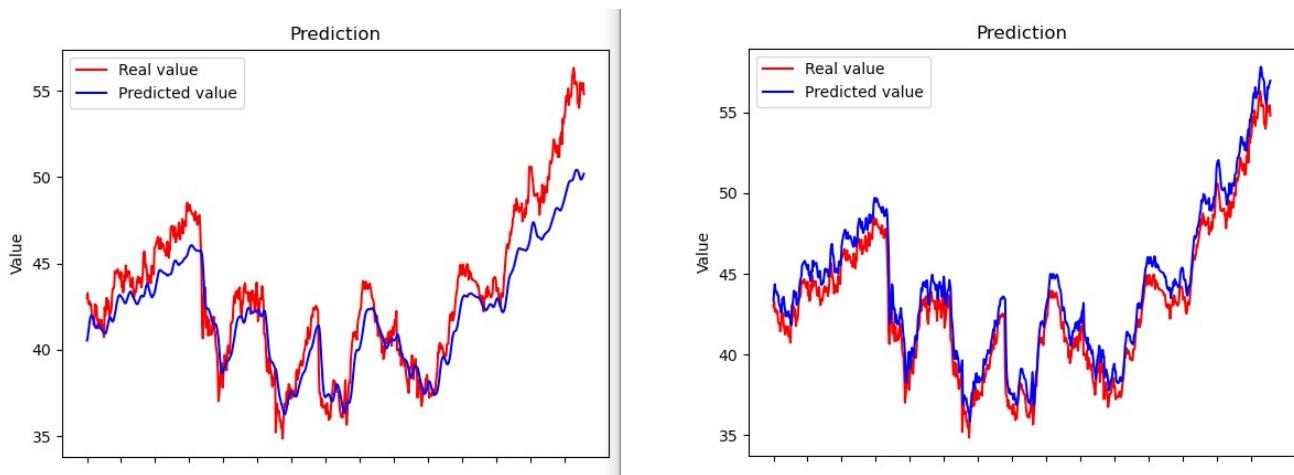
Εκπαίδευση με την τέταρτη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά.  
Πρόβλεψη για την τέταρτη χρονοσειρά.



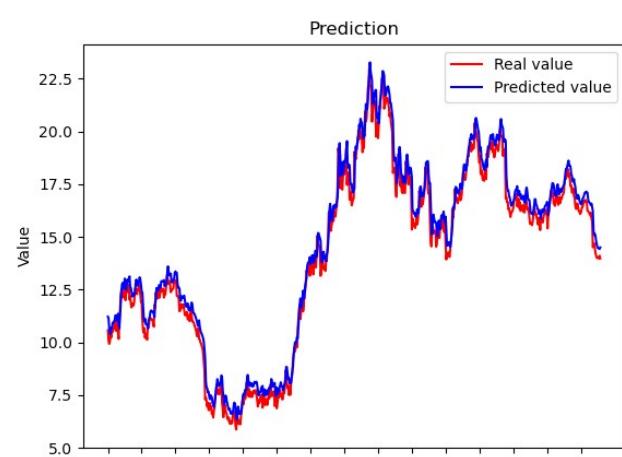
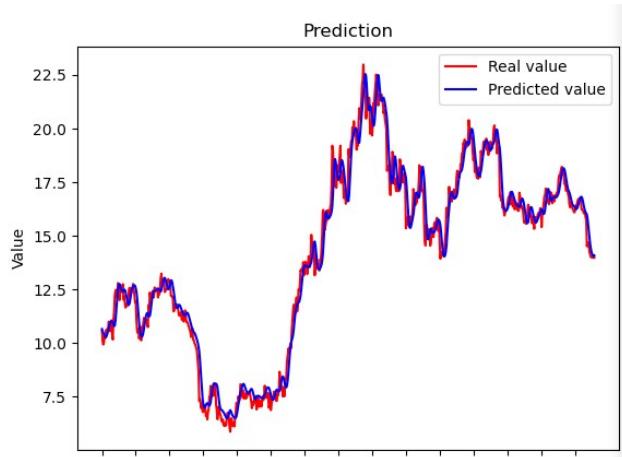
Εκπαίδευση με την πέμπτη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά. Πρόβλεψη για την πέμπτη χρονοσειρά.



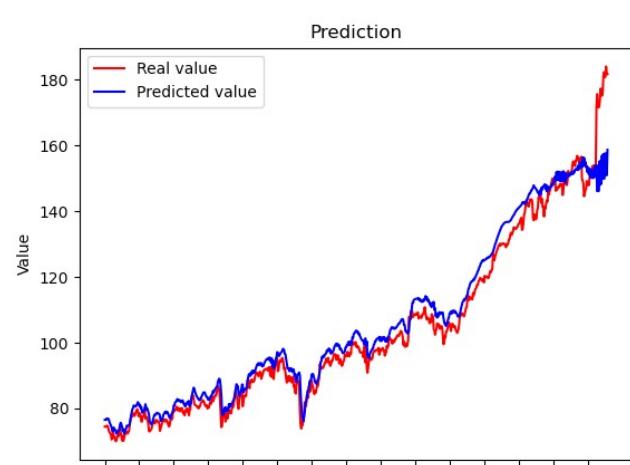
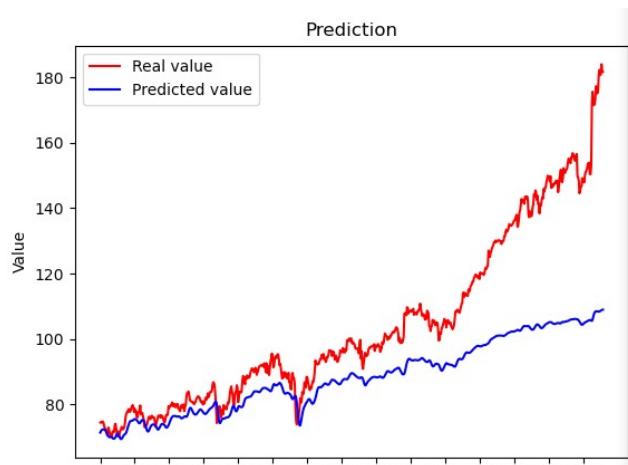
Εκπαίδευση με την έκτη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά. Πρόβλεψη για την έκτη χρονοσειρά.



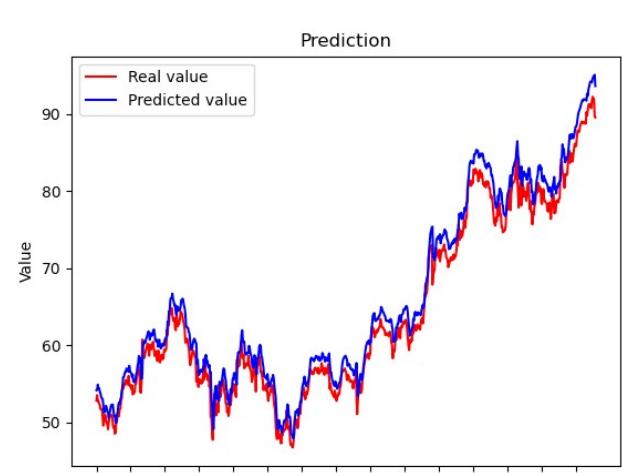
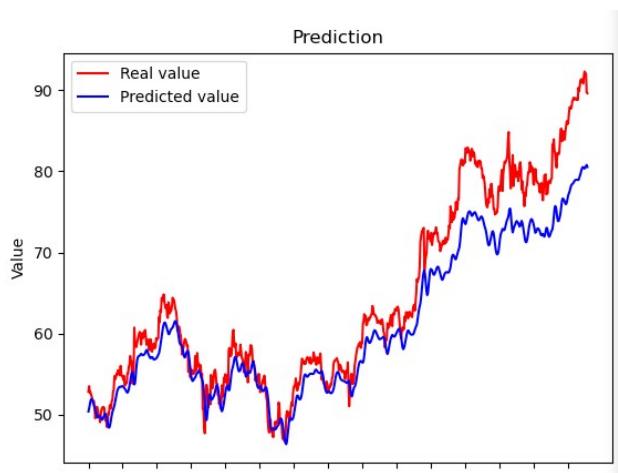
Εκπαίδευση με την έβδομη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά. Πρόβλεψη για την έβδομη χρονοσειρά.



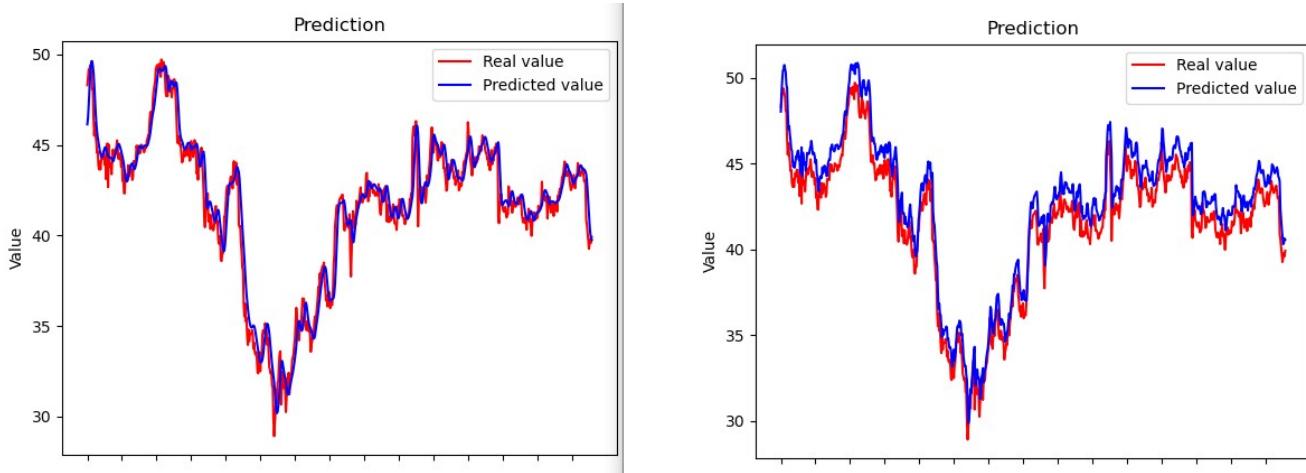
Εκπαίδευση με την όγδοη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά.  
Πρόβλεψη για την όγδοη χρονοσειρά.



Εκπαίδευση με την ένατη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά.  
Πρόβλεψη για την ένατη χρονοσειρά.



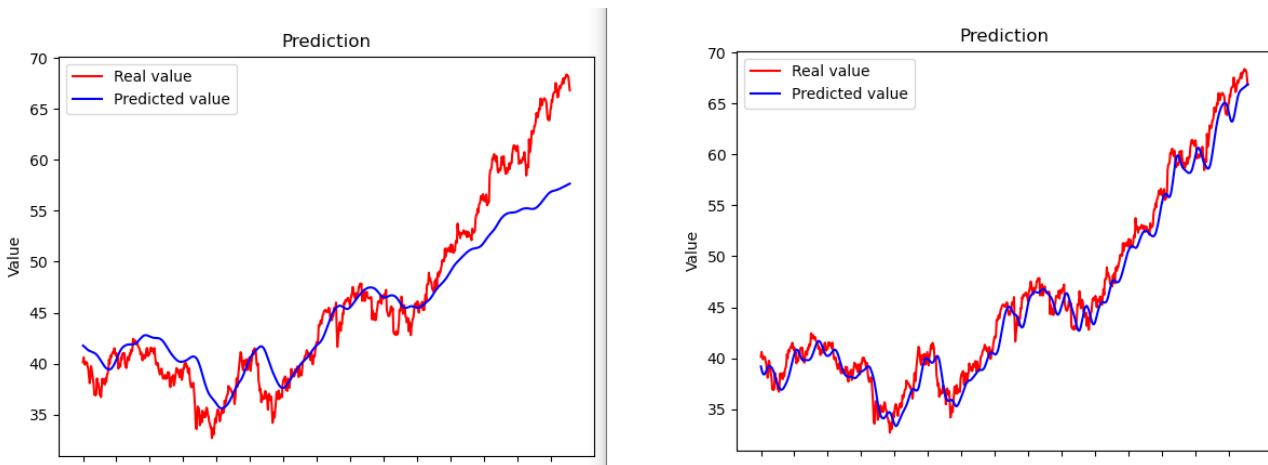
Εκπαίδευση με την δέκατη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά. Πρόβλεψη για την δέκατη χρονοσειρά.



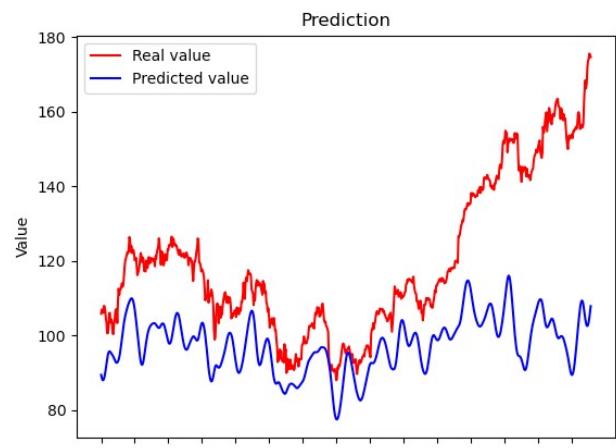
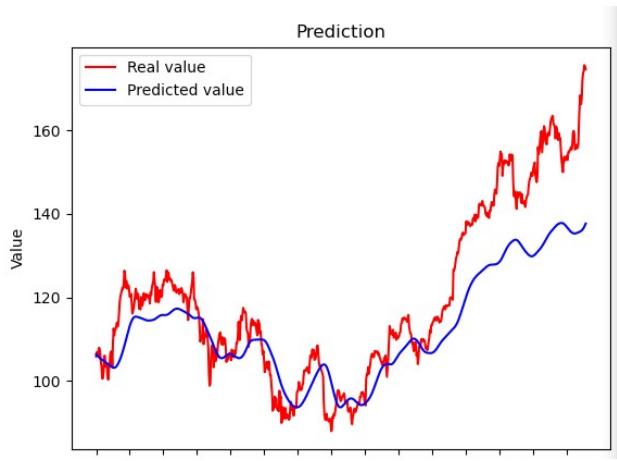
Σε γενικές γραμμές η πρόβλεψη είναι ακόμα καλύτερη με το μοντέλο που εκπαιδεύτηκε με το σύνολο των χρονοσειρών σε σχέση με την εκπαίδευση με μία χρονοσειρά καθώς ο αριθμός των δεδομένων εκπαίδευσης είναι μεγαλύτερος. Το loss είναι μία τάξη μεγέθους μικρότερο.

- Στο παραπάνω βέλτιστο μοντέλο αν **αυξηθεί ο αριθμός batch\_size** από 32 σε 2048 τότε η πρόβλεψη χαλάει ακόμα και αν όλες οι άλλες υπερπαράμετροι παραμείνουν ίδιες. Παρακάτω φαίνονται τα διαγράμματα που προέκυψαν τα οποία αν συγκριθούν με τα αντίστοιχα παραπάνω φαίνεται η διαφορά στην πρόβλεψη. Το θετικό είναι το μοντέλο ολοκληρώνει πιο γρήγορα την εκπαίδευση σε αυτή την περίπτωση.

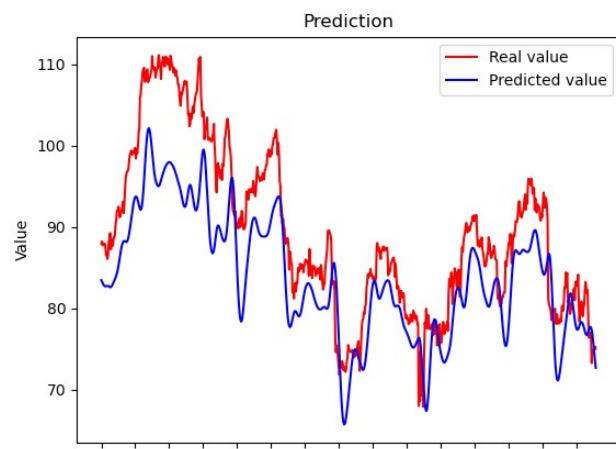
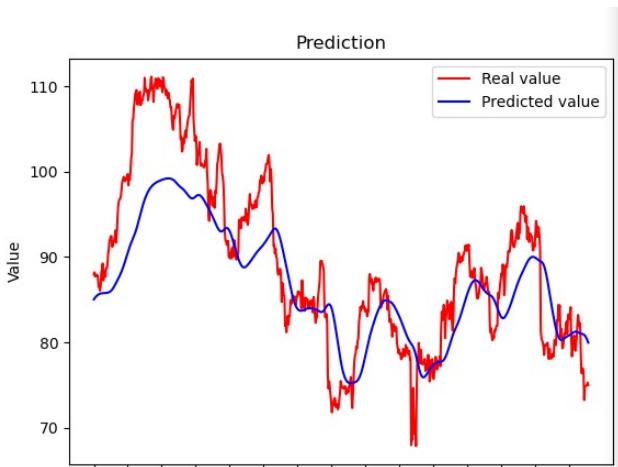
Εκπαίδευση με την πρώτη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά. Πρόβλεψη για την πρώτη χρονοσειρά.



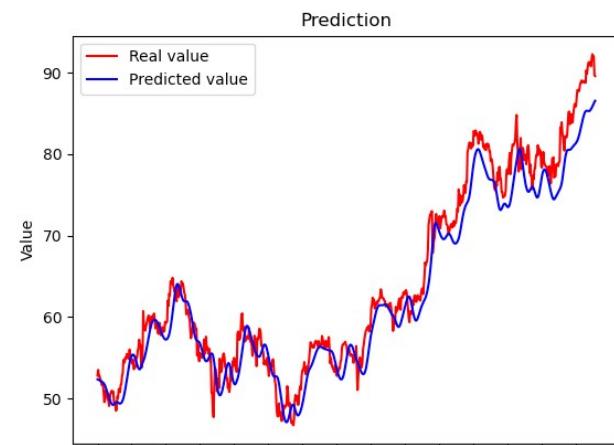
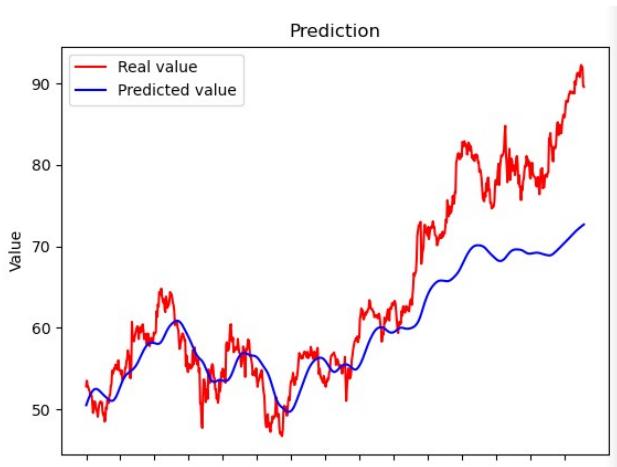
Εκπαίδευση με την τέταρτη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά. Πρόβλεψη για την τέταρτη χρονοσειρά.



Εκπαίδευση με την πέμπτη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά. Πρόβλεψη για την πέμπτη χρονοσειρά.



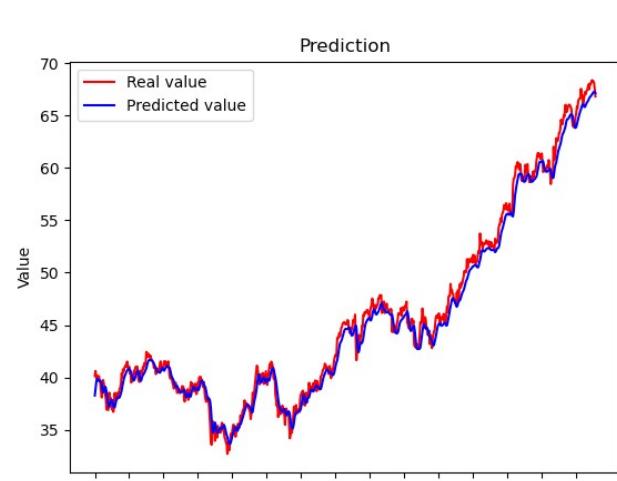
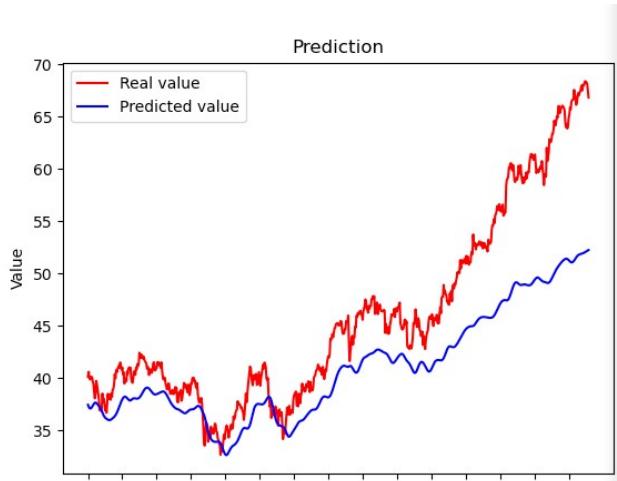
Εκπαίδευση με την ένατη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά. Πρόβλεψη για την ένατη χρονοσειρά.



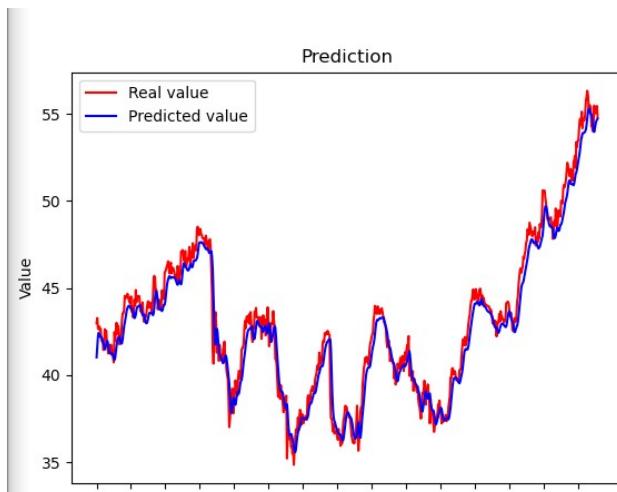
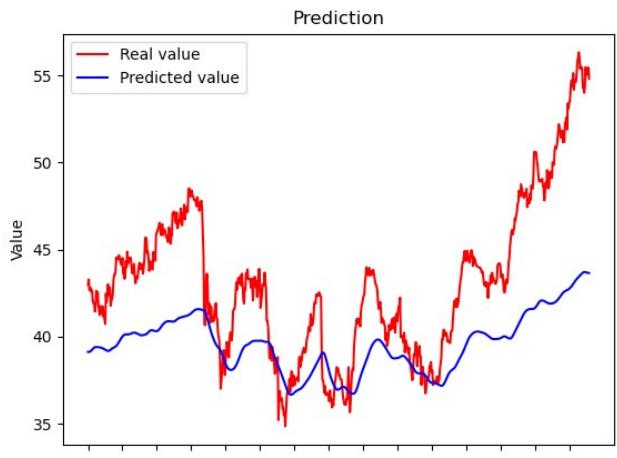
Και εδώ σε γενικές γραμμές η πρόβλεψη είναι καλύτερη με το μοντέλο που εκπαιδεύτηκε με το σύνολο των χρονοσειρών σε σχέση με την εκπαίδευση με μία χρονοσειρά.

- Αν στο βέλτιστο μοντέλο **αλλάξουμε το dropout** από 0.2 σε 0.8 τότε η πρόβλεψη χαλάει ακόμα και αν όλες οι άλλες υπερπαράμετροι παραμείνουν ίδιες. Παρακάτω φαίνονται τα διαγράμματα που προέκυψαν τα οποία αν συγκριθούν με τα αντίστοιχα του βέλτιστου μοντέλου φαίνεται η διαφορά στην πρόβλεψη.

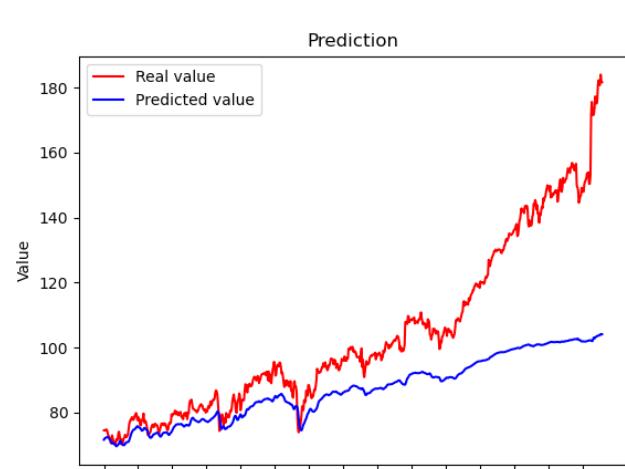
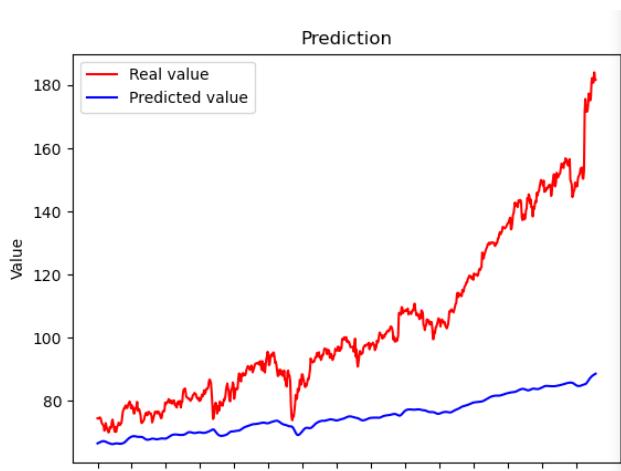
Εκπαίδευση με την πρώτη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά. Πρόβλεψη για την πρώτη χρονοσειρά.



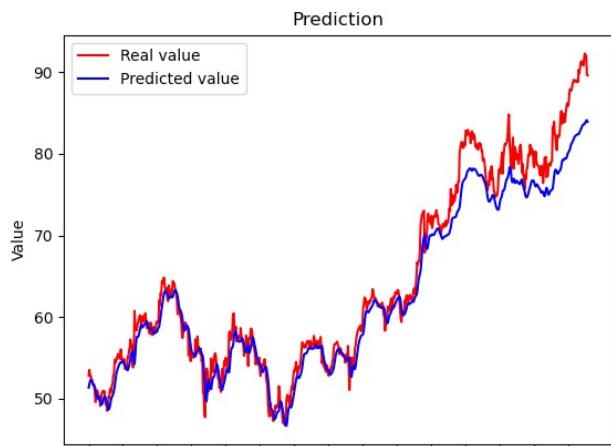
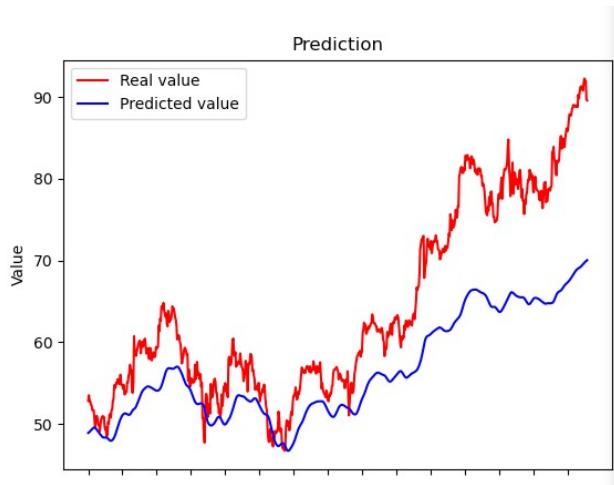
Εκπαίδευση με την έκτη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά. Πρόβλεψη για την έκτη χρονοσειρά.



Εκπαίδευση με την όγδοη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά. Πρόβλεψη για την όγδοη χρονοσειρά.



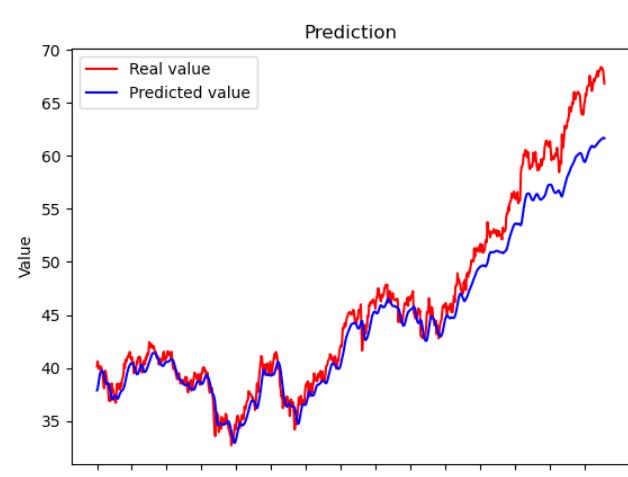
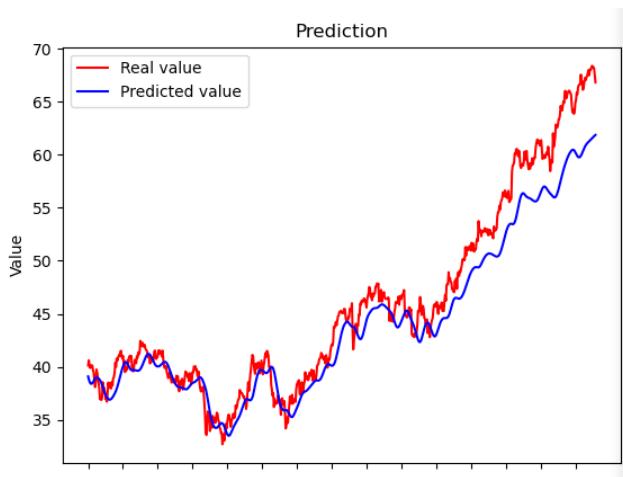
Εκπαίδευση με την ένατη χρονοσειρά αριστερά και εκπαίδευση με όλες ( $n = 10$ ) τις χρονοσειρές δεξιά. Πρόβλεψη για την ένατη χρονοσειρά.



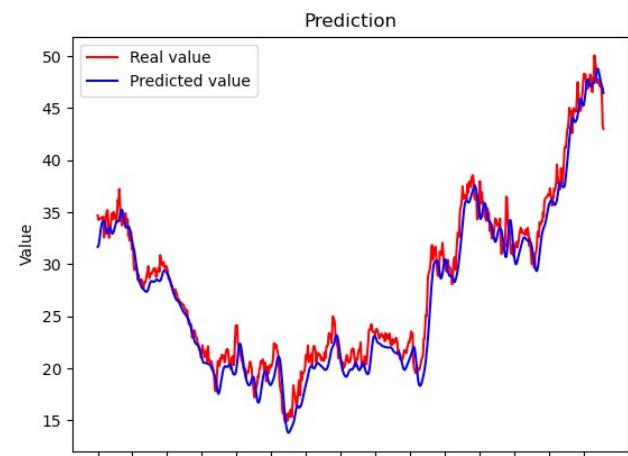
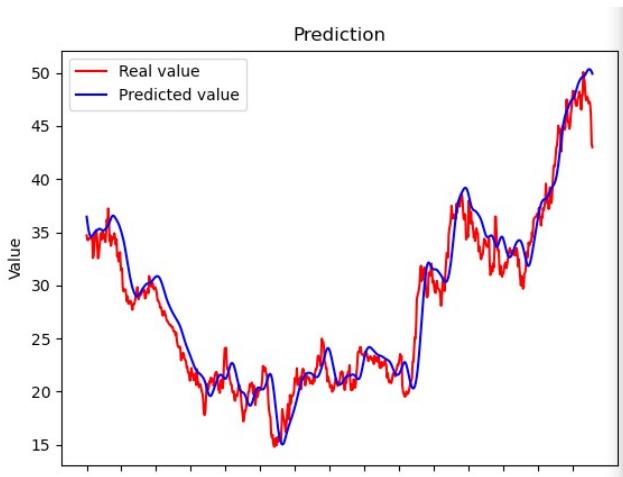
Και εδώ σε γενικές γραμμές η πρόβλεψη είναι καλύτερη με το μοντέλο που εκπαιδεύτηκε με το σύνολο των χρονοσειρών σε σχέση με την εκπαίδευση με μία χρονοσειρά.

- Αν στο βέλτιστο μοντέλο **αλλάξουμε το epochs** σε 8 τότε η πρόβλεψη χαλάει ακόμα και αν όλες οι άλλες υπερπαράμετροι παραμείνουν ίδιες. Παρακάτω φαίνονται τα διαγράμματα που προέκυψαν για τέσσερις χρονοσειρές τα οποία αν συγκριθούν με τα αντίστοιχα του βέλτιστου μοντέλου φαίνεται η διαφορά στην πρόβλεψη.

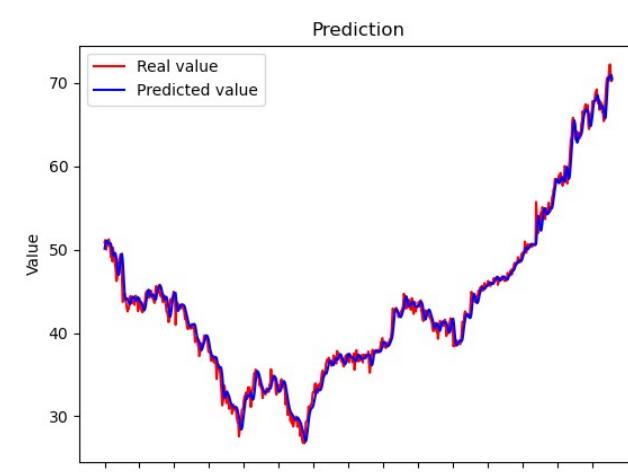
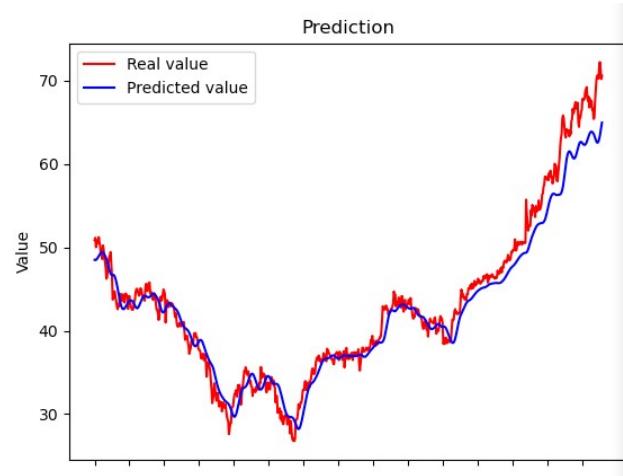
Εκπαίδευση με την πρώτη χρονοσειρά. Αριστερά φαίνεται το αποτέλεσμα με epochs = 8 και δεξιά το αποτέλεσμα του βέλτιστου μοντέλου για σύγκριση.



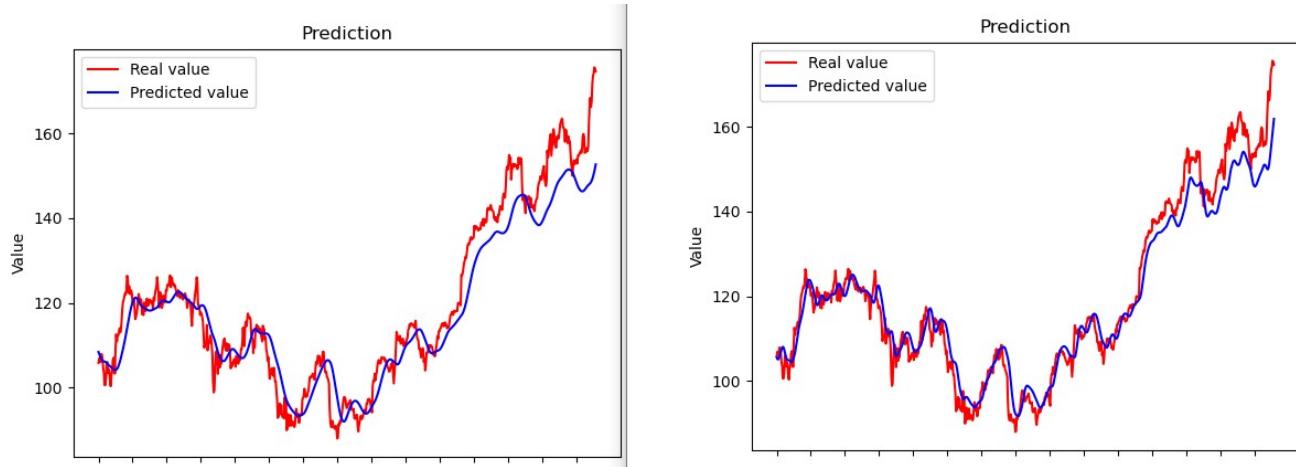
Εκπαίδευση με την δεύτερη χρονοσειρά. Αριστερά φαίνεται το αποτέλεσμα με epochs = 8 και δεξιά το αποτέλεσμα του βέλτιστου μοντέλου για σύγκριση.



Εκπαίδευση με την τρίτη χρονοσειρά. Αριστερά φαίνεται το αποτέλεσμα με epochs = 8 και δεξιά το αποτέλεσμα του βέλτιστου μοντέλου για σύγκριση.

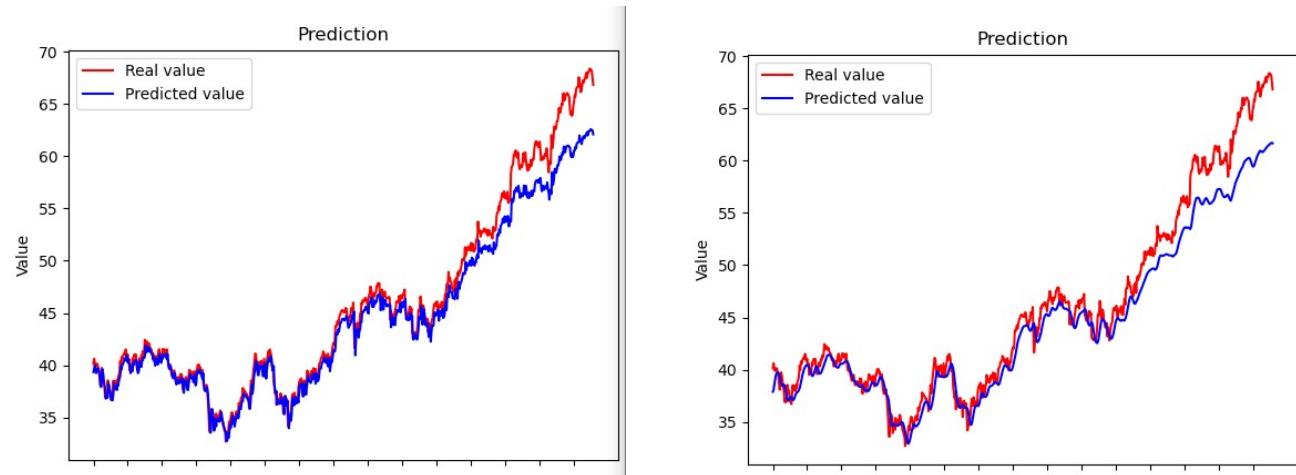


Εκπαίδευση με την τέταρτη χρονοσειρά. Αριστερά φαίνεται το αποτέλεσμα με epochs = 8 και δεξιά το αποτέλεσμα του βέλτιστου μοντέλου για σύγκριση.

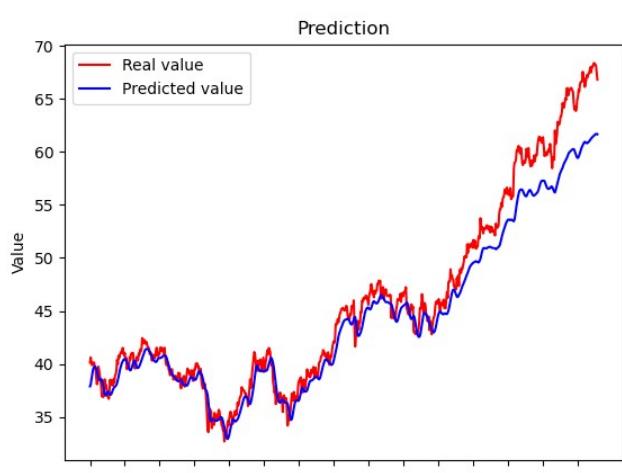
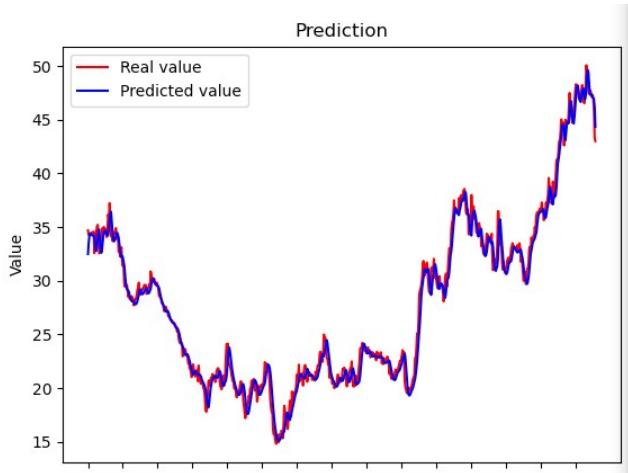


- Αν στο βέλτιστο μοντέλο **αφαιρέσουμε το dropout** τότε η πρόβλεψη εμφανίζεται καλύτερη αλλά δεν αποφεύγουμε το overfitting. Παρακάτω φαίνονται τα διαγράμματα που προέκυψαν για τέσσερις χρονοσειρές τα οποία συγκρίνονται με τα αντίστοιχα του βέλτιστου μοντέλου.

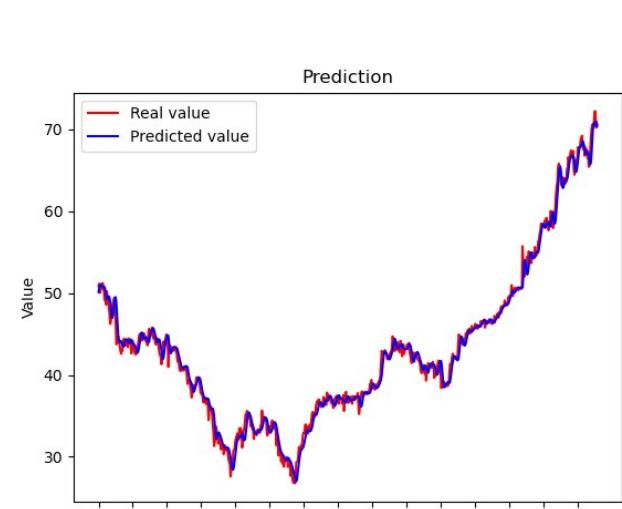
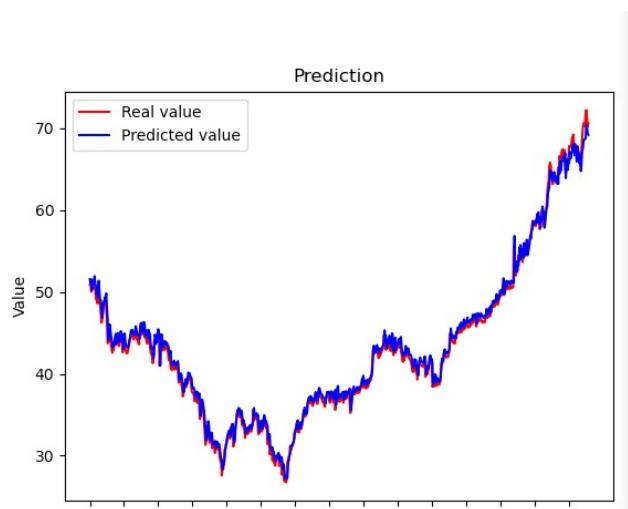
Εκπαίδευση με την πρώτη χρονοσειρά. Αριστερά φαίνεται το αποτέλεσμα χωρίς dropout και δεξιά το αποτέλεσμα του βέλτιστου μοντέλου για σύγκριση.



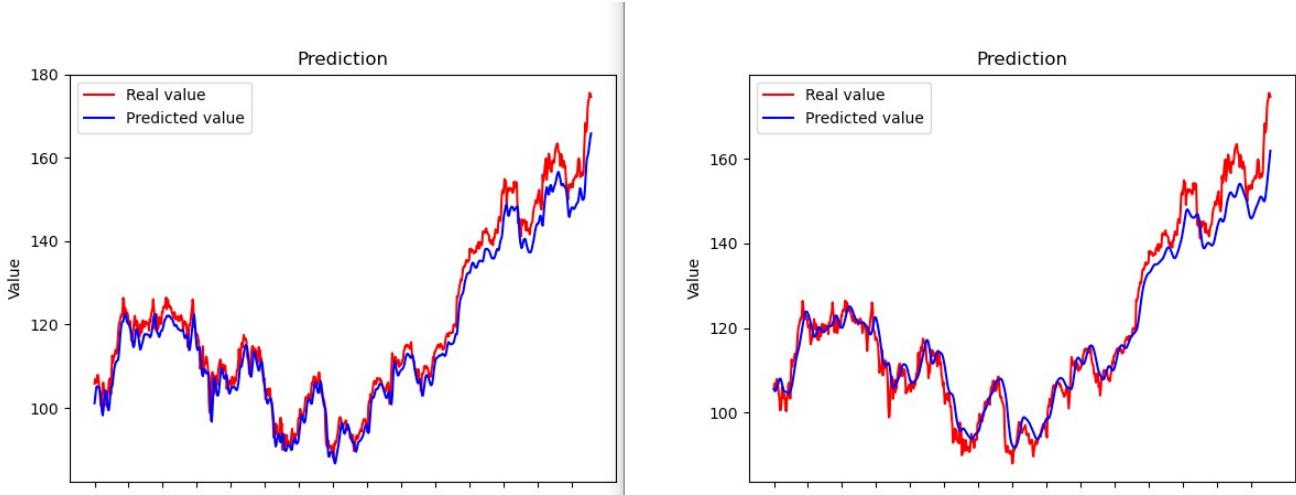
Εκπαίδευση με την δεύτερη χρονοσειρά. Αριστερά φαίνεται το αποτέλεσμα χωρίς dropout και δεξιά το αποτέλεσμα του βέλτιστου μοντέλου για σύγκριση.



Εκπαίδευση με την τρίτη χρονοσειρά. Αριστερά φαίνεται το αποτέλεσμα χωρίς dropout και δεξιά το αποτέλεσμα του βέλτιστου μοντέλου για σύγκριση.

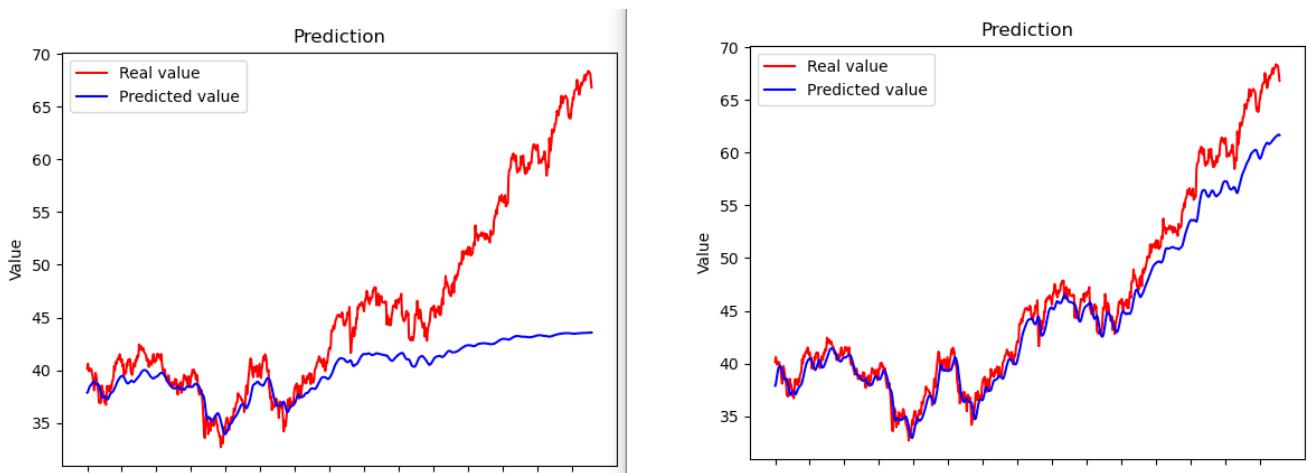


Εκπαίδευση με την τέταρτη χρονοσειρά. Αριστερά φαίνεται το αποτέλεσμα χωρίς dropout και δεξιά το αποτέλεσμα του βέλτιστου μοντέλου για σύγκριση.

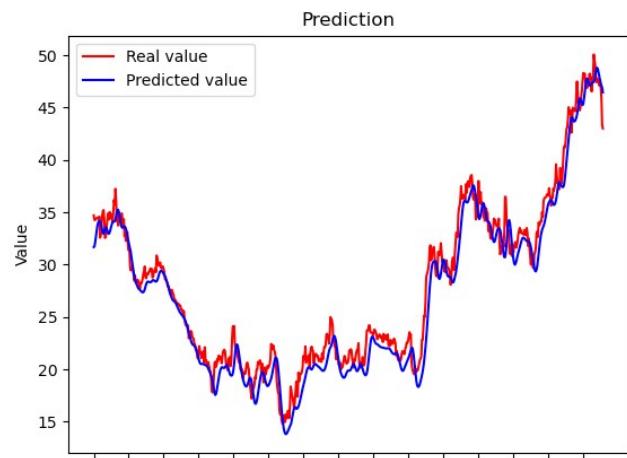
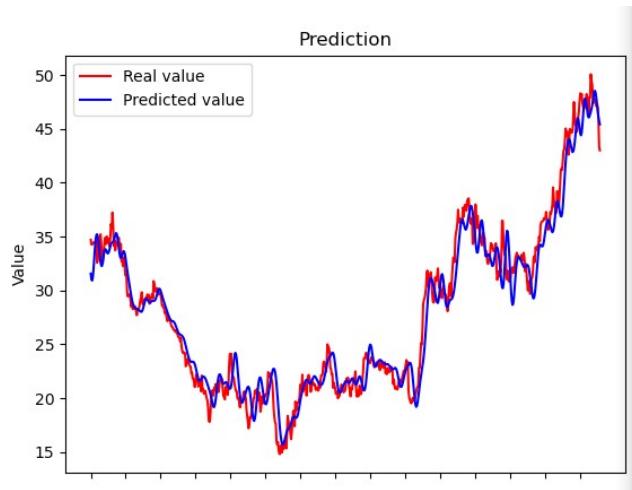


- Αν στο βέλτιστο μοντέλο **προσθέσουμε τέσσερα layers** τότε η πρόβλεψη χαλάει ακόμα και αν ολες οι άλλες υπερπαράμετροι παραμείνουν ίδιες. Παρακάτω φαίνονται τα διαγράμματα που προέκυψαν για τέσσερις χρονοσειρές τα οποία συγκρίνονται με τα αντίστοιχα του βέλτιστου μοντέλου.

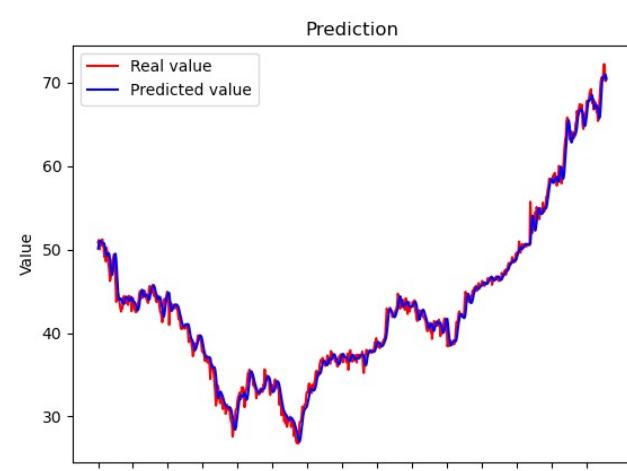
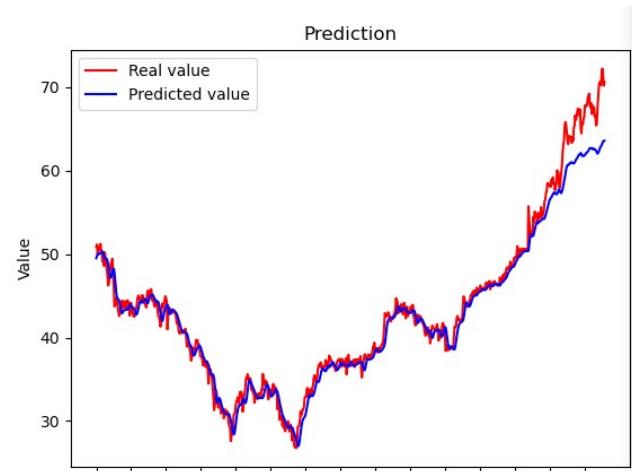
Εκπαίδευση με την πρώτη χρονοσειρά. Αριστερά φαίνεται το αποτέλεσμα με τέσσερα επιπλέον layers και δεξιά το αποτέλεσμα του βέλτιστου μοντέλου για σύγκριση.



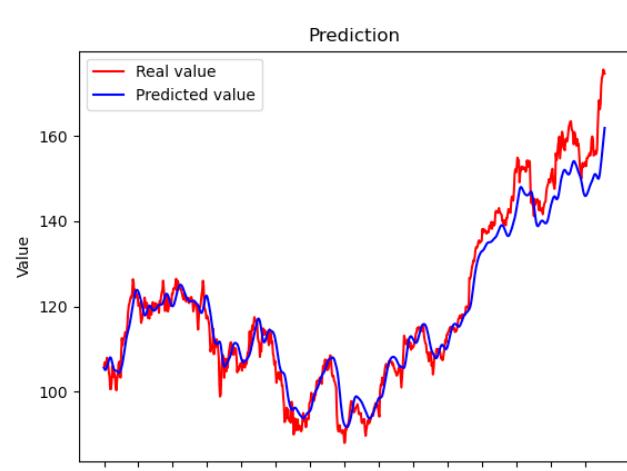
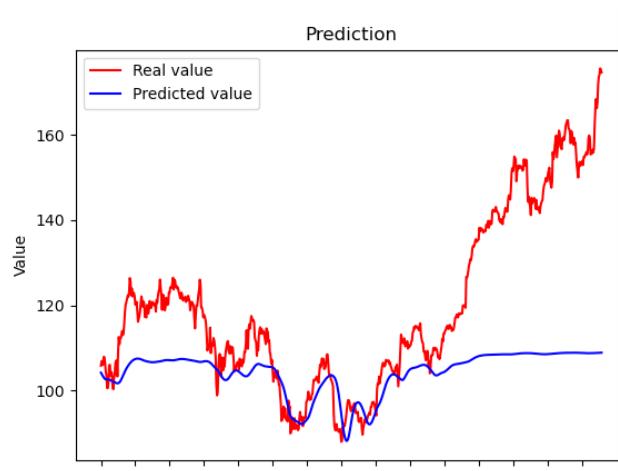
Εκπαίδευση με την δεύτερη χρονοσειρά. Αριστερά φαίνεται το αποτέλεσμα με τέσσερα επιπλέον layers και δεξιά το αποτέλεσμα του βέλτιστου μοντέλου για σύγκριση.



Εκπαίδευση με την τρίτη χρονοσειρά. Αριστερά φαίνεται το αποτέλεσμα με τέσσερα επιπλέον layers και δεξιά το αποτέλεσμα του βέλτιστου μοντέλου για σύγκριση.



Εκπαίδευση με την τέταρτη χρονοσειρά. Αριστερά φαίνεται το αποτέλεσμα με τέσσερα επιπλέον layers και δεξιά το αποτέλεσμα του βέλτιστου μοντέλου για σύγκριση.



## Ερώτημα Β

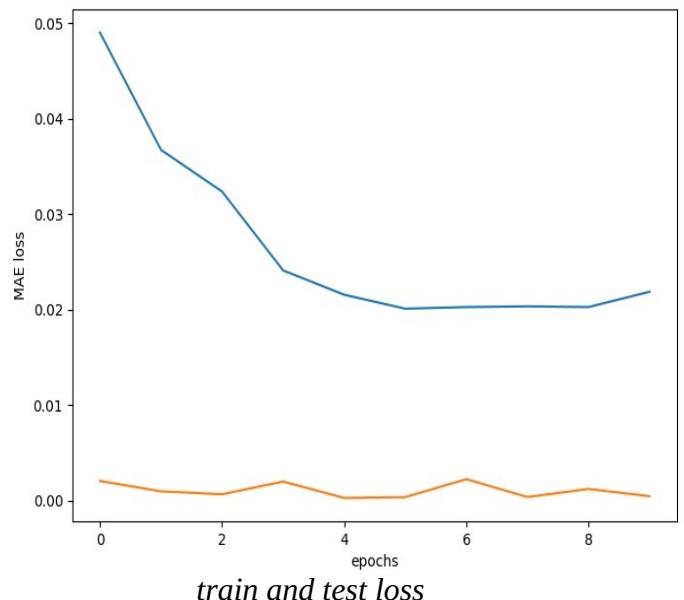
Ο Αλγόριθμος κάνει εκπαίδευση το σύνολο χρονοσειρών. Πιο συγκεκριμένα οι χρονοσειρές του dataset ενώνονται σε μία η οποία γίνεται και scaled. Για την εκπαίδευση χρησιμοποιείται το 95% των τιμών της κάθε χρονοσειράς, καθώς τα δεδομένα είναι πολλά και αρκεί ένα 5% για validation.

Παρακάτω παρουσιάζονται τα αποτελέσματα για το βέλτιστο μοντέλο του ερωτήματος Α καθώς και αποτελέσματα για κάποια από τα μη βέλτιστα μοντέλα που δοκιμάστηκαν. Το βέλτιστα μοντέλο είναι το εξής και καταλήξαμε σε αυτό μετά από σχεδόν 20 δοκιμές ενώ παρακάτω παρουσιάζονται μερικές από αυτές:

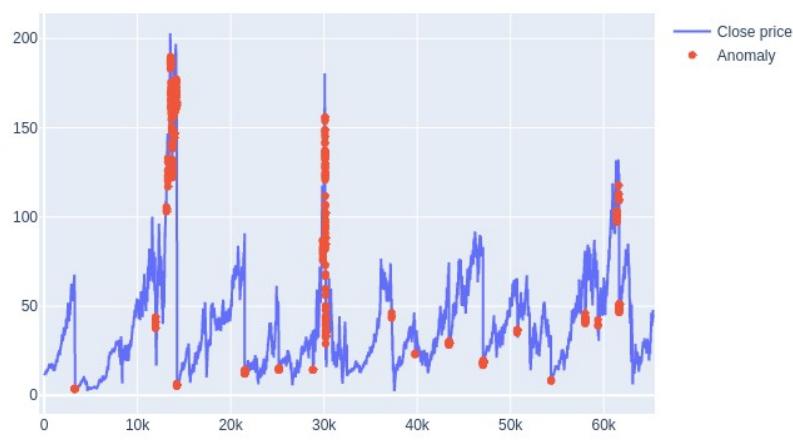
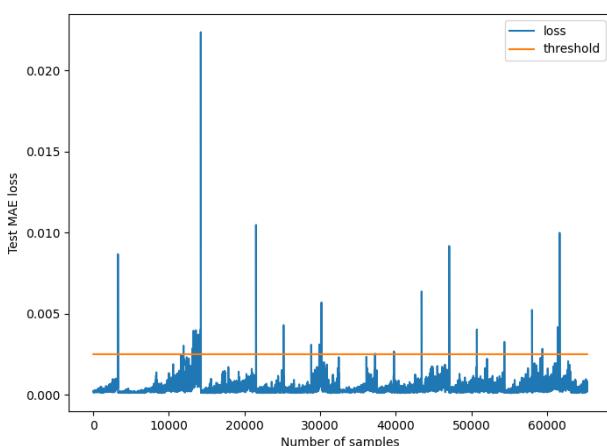
```
model = Sequential()
model.add(LSTM(
    units=64,
    input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(rate=0.5))
model.add(RepeatVector(n=X_train.shape[1]))
model.add(LSTM(units=64, return_sequences=True))
model.add(Dropout(rate=0.5))
model.add(
    TimeDistributed(
        Dense(units=1)))
model.compile(loss='mae', optimizer='adam')

history = model.fit(
    X_train, y_train,
    epochs=10,
    batch_size=1024,
    validation_split=0.1,
    shuffle=False)
```

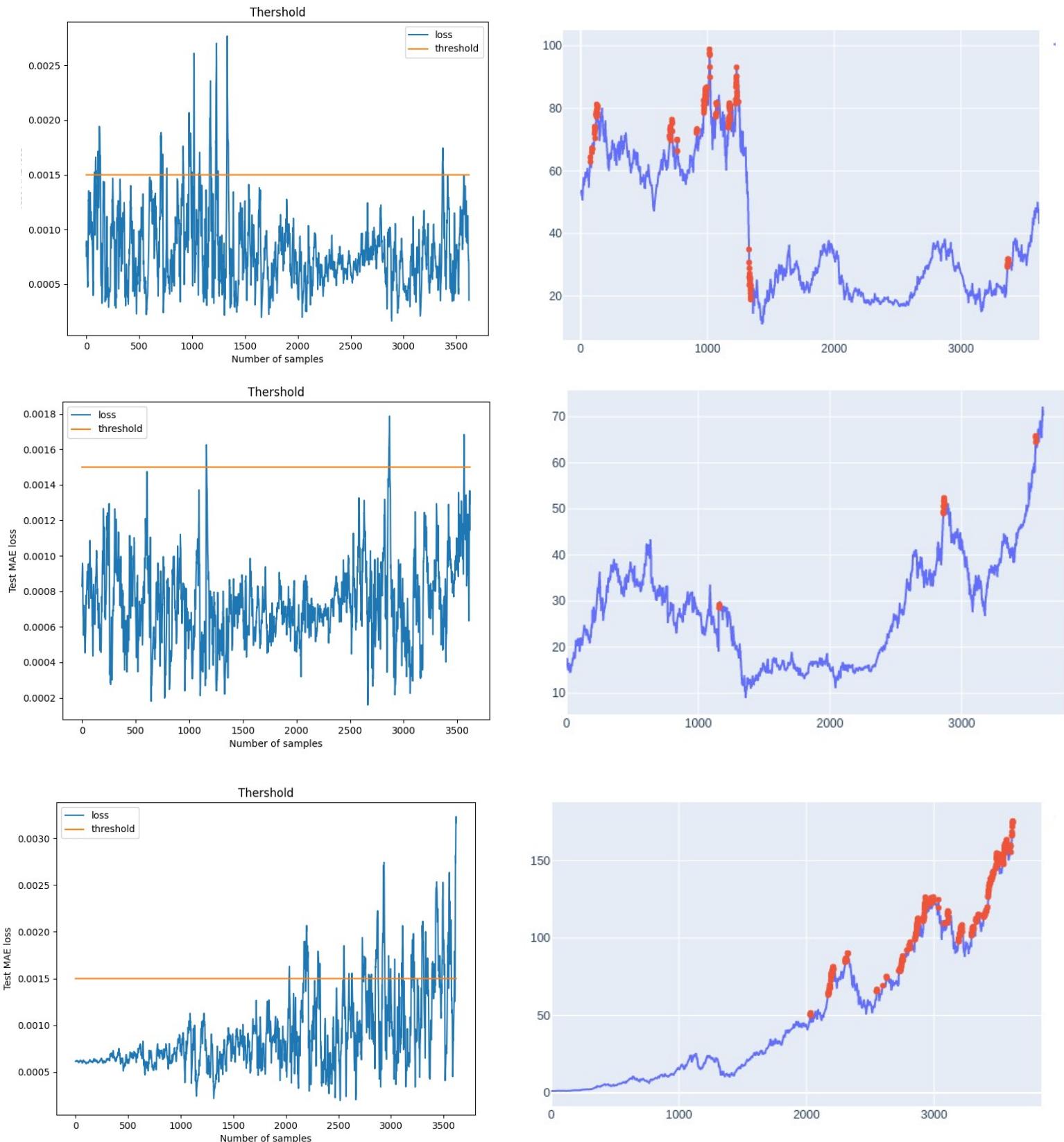
Εδώ έχουμε θέσει threshold = 0.0025 και έχουμε κάνει plot τις ανωμαλίες για την χρονοσειρά του testing



Detected anomalies

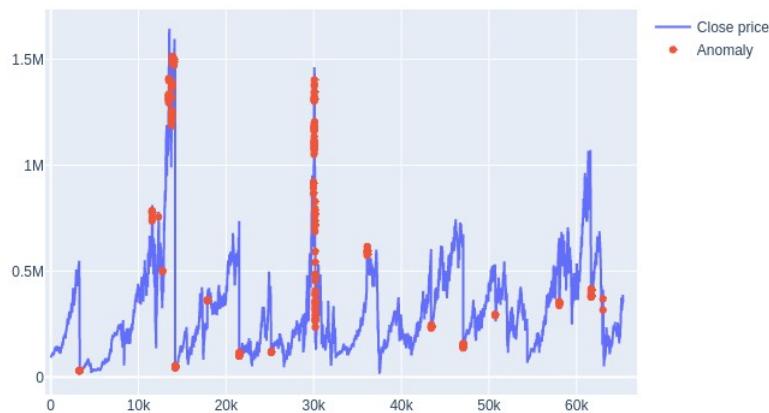
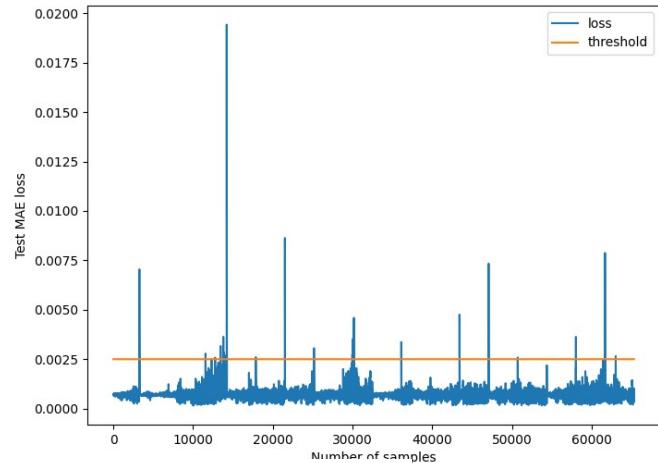
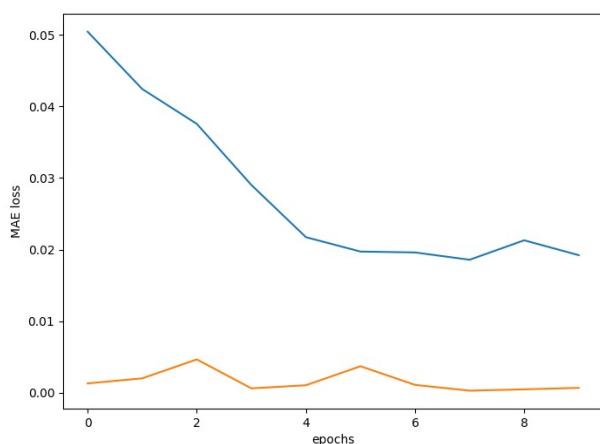


Εδώ παρουσιάζονται τα αποτελέσματα όταν με το παραπάνω εκπαίδευμένο μοντέλο κάνουμε plot τις ανωμαλίες για τις πρώτες 3 χρονοσειρές του dataset για threshold = 0.0015. Οι χρονοσειρές είναι οι aa, aaba, aapl .

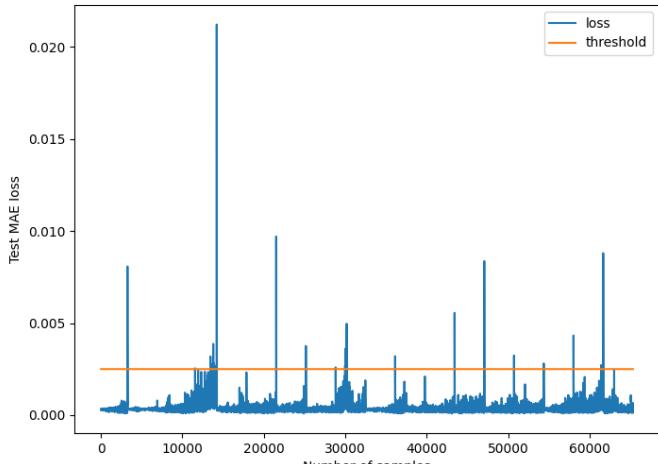
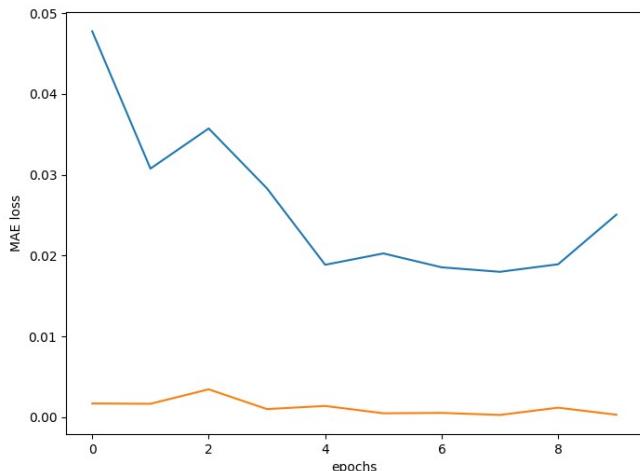


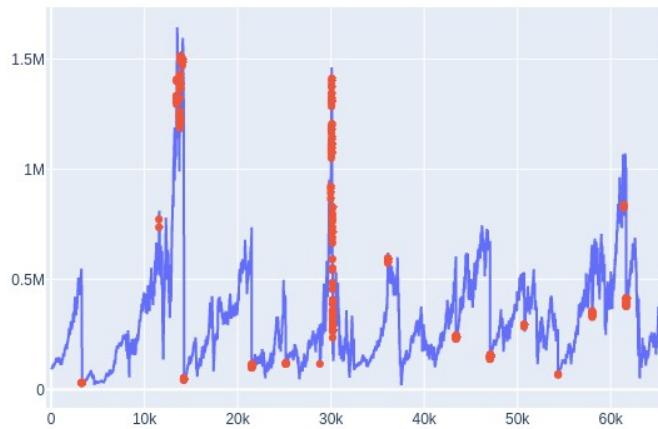
Παρακάτω παρουσιάζονται γραφήματα για δοκιμές διαφορετικών μοντέλων.

1. Ιδιο με το βέλτιστο μοντέλο μόνο με dropout = 0.2 και batch = 2024 .Ο λόγος που επιλέχθηκε τελικά το 0.5 για dropout είναι επειδή είχαμε ελάχιστα καλυτερο loss στο validation.



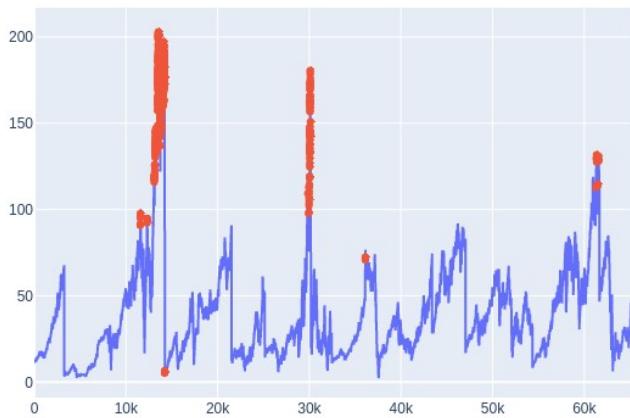
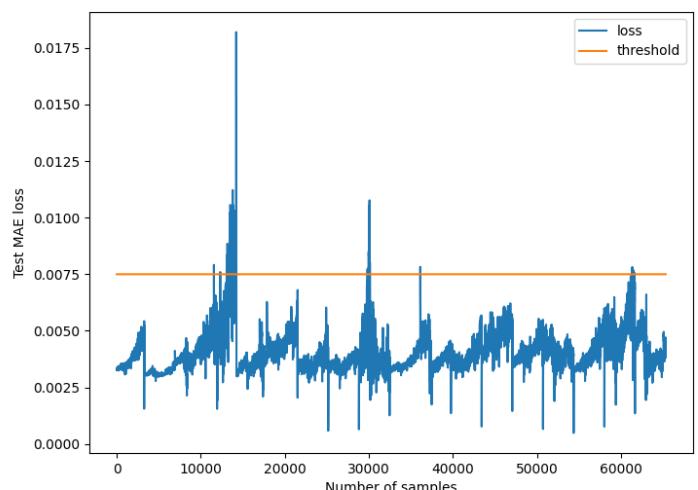
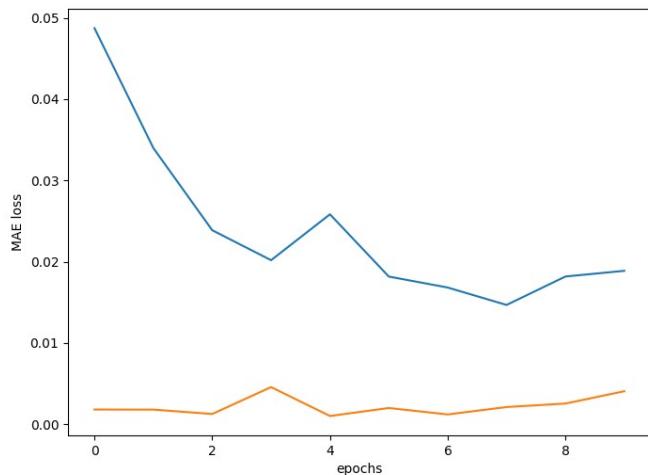
2. Ιδιο μοντέλο με το 1 αλλά με batch = 1024



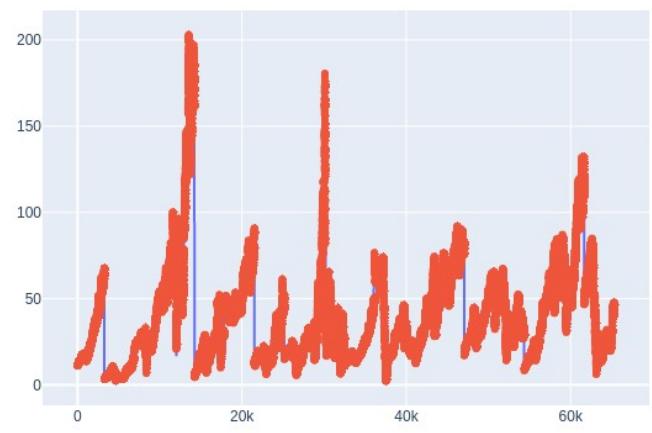


Βλέπουμε ότι με την μείωση του batch από το μοντέλο 1 στο 2 δεν έχουμε κάποια ουσιαστική διαφορά.

3.Στην συγκεκριμένη δοκιμή έχουμε το μοντέλο 1 αλλα χωρίς τα dropout layers:



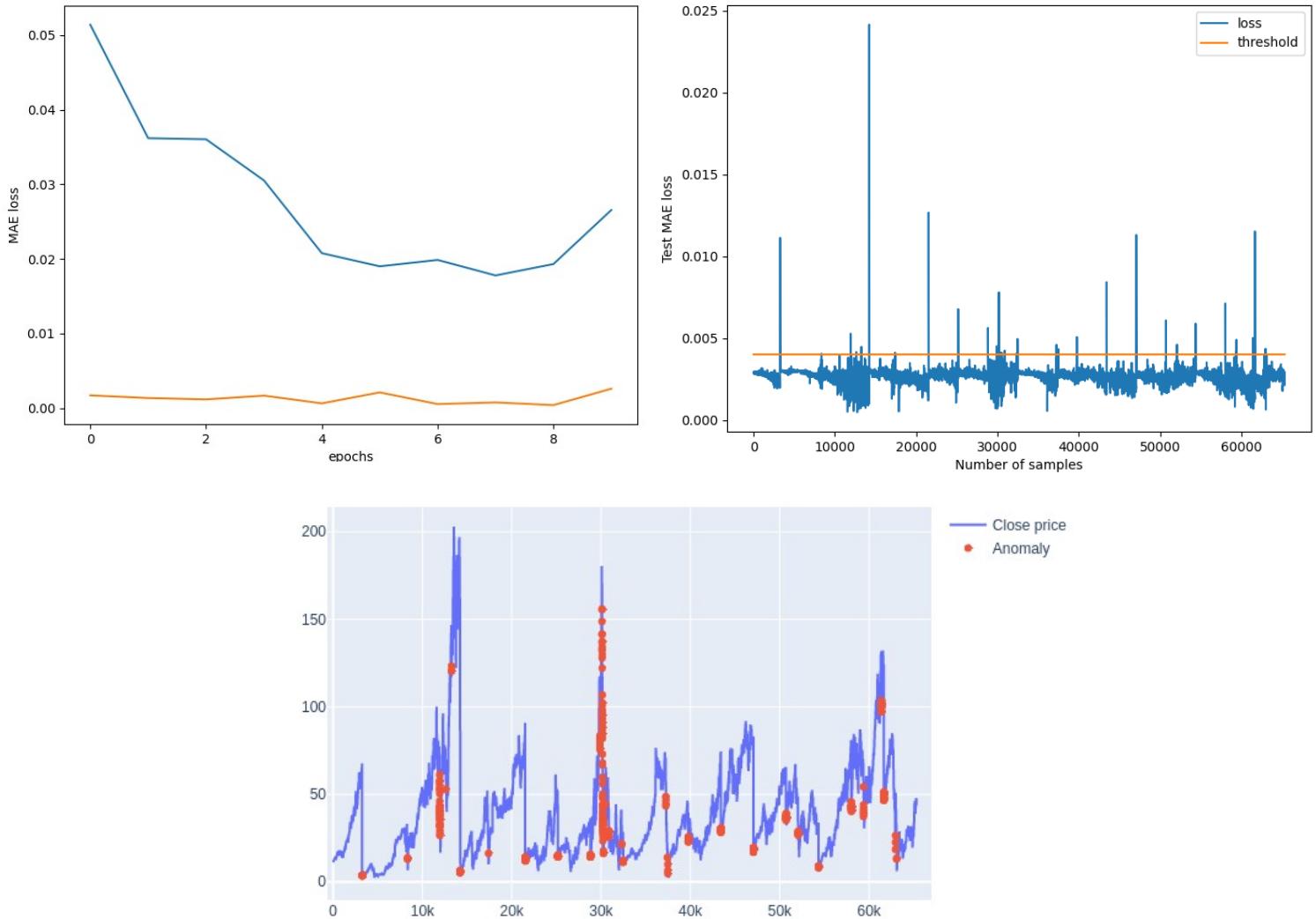
*threshold = 0.0075*



*threshold = 0.0025*

Παρατηρούμε ότι με την αφαίρεση των dropout layers αρχίζουμε να έχουμε overfit ακόμα και με τόσα λίγα epochs ενώ παράλληλα αυξάνεται και το test\_loss. Επίσης οι τιμές στις οποίες κειμένεται το test\_loss είναι μεγαλύτερες από τις προηγούμενες δοκιμές, κάτι το οποίο φαίνεται και στα διαγράμματα των ανωμαλιών του μοντέλου. Για το συγκεκριμένο μοντέλο κατάλληλο είναι το threshold = 0.075 οι ανωμαλίες του οποίου φαίνονται στο κάτω δεξιά διάγραμμα. Βλέπουμε επίσης πως αν θέσουμε threshold = 0.0025 όπως στα προηγούμενα όλο σχεδόν το διάγραμμα λαμβάνεται ως ανωμαλία.

4. Αφού είδαμε ότι είχαμε χειρότερο αποτέλεσμα με την αφαίρεση των dropout τα ξανατοποθετήσαμε και θέσαμε dropout = 0.4 με batch = 1024.

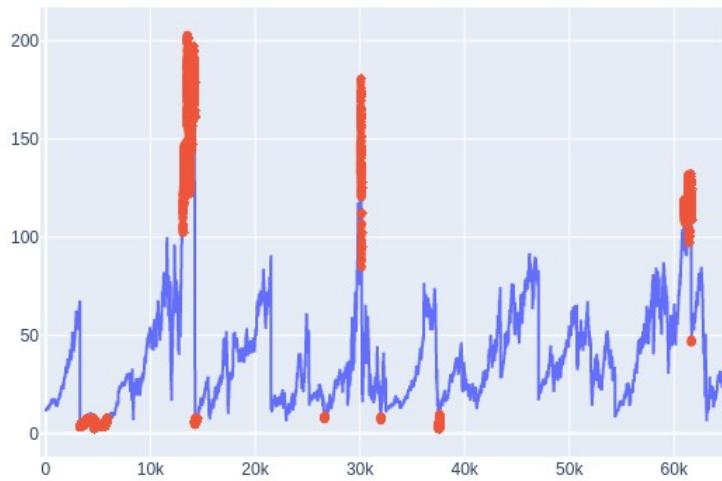
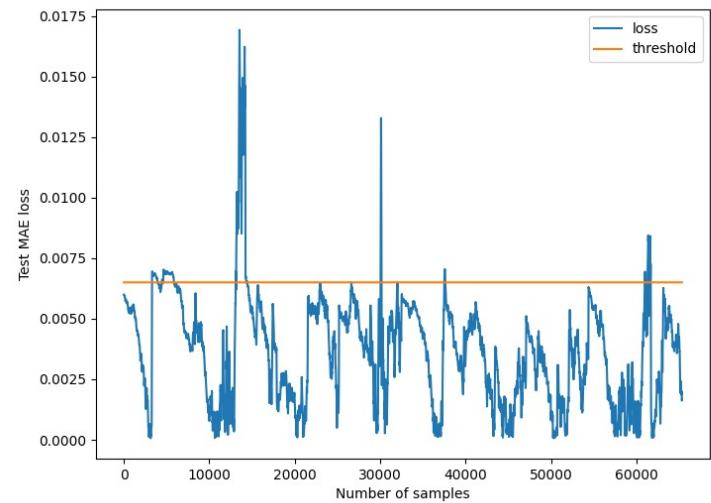
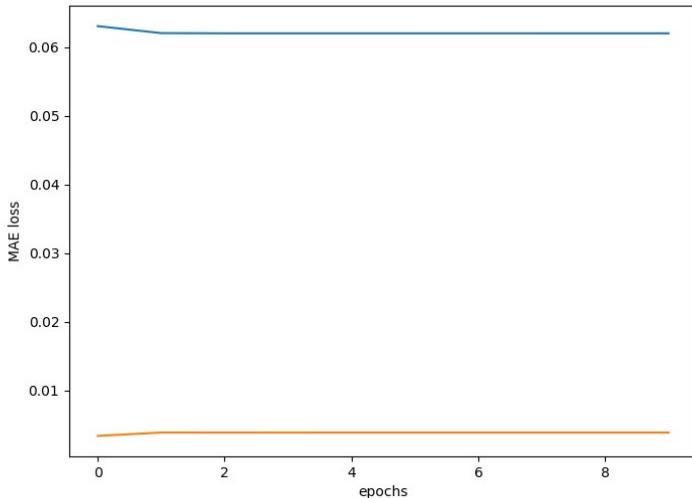


Βλέπουμε ότι υπάρχει ακόμα μια μικρή τάση για overfit, ενώ και το κατάλληλο threshold είναι κοντά στο 0.4. Για τον λόγο αυτό στο βέλτιστα μοντέλο είχε αυξηθεί το dropout σε 0.5 καθώς θέλαμε να μειώσουμε αυτήν την τάση για overfit.

5. Δοκιμάστηκε το εξής μοντέλο:

```
model = Sequential()
model.add(LSTM(X_train.shape[1], activation='relu',
               input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=True))
model.add(LSTM(6, activation='relu', return_sequences=True))
model.add(LSTM(1, activation='relu'))
model.add(RepeatVector(n=X_train.shape[1]))
model.add(LSTM(X_train.shape[1], activation='relu', return_sequences=True))
model.add(LSTM(6, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(1)))
```

Οι διαφορές του με το βέλτιστο είναι ότι δεν έχει dropout layers, ενώ τα units των lstm έχουν μειωθεί πάρα πολύ.

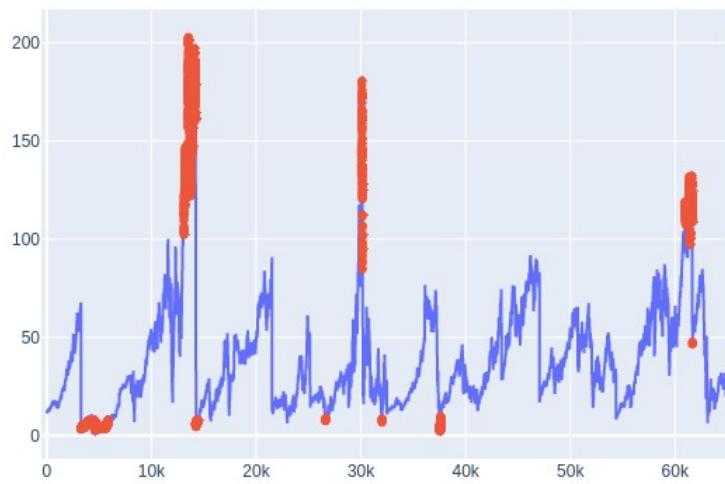
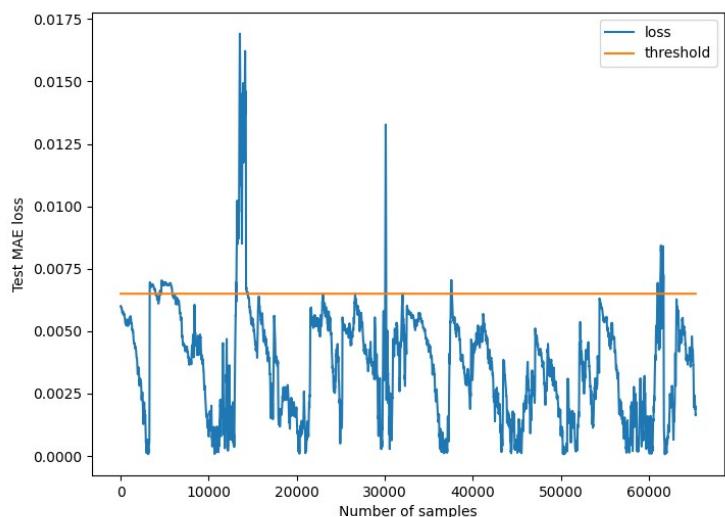
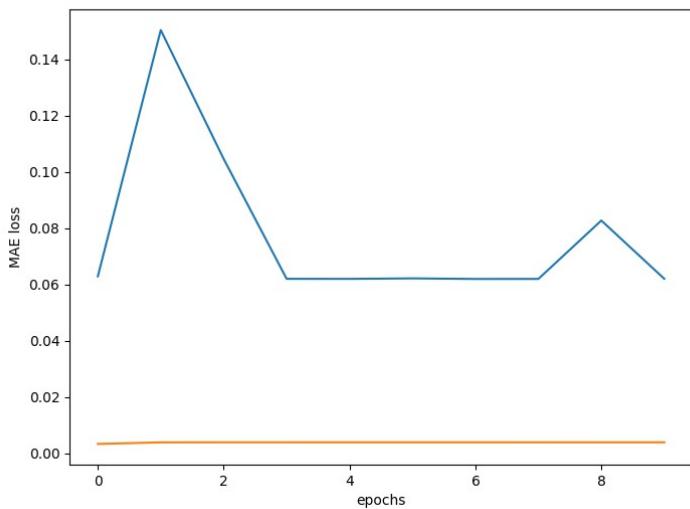


Βλλεπούμε ότι η μεγάλη μείωση στις διαστάσεις των lstm layers έχει ως αποτέλεσμα να μην εκπαιδεύεται παραπάνω το μοντέλο όσο περνάν τα epochs. Το test\_loss έχει φτάσει σε ένα σχετικά καλό threshold = 0.0065 αλλα δεν φτάνει αυτό του βέλτιστου μοντέλου που ήταν 0.0025.

## 6. Δοκιμάστηκε το εξήσις μοντέλο:

```
model = Sequential()
model.add(LSTM(X_train.shape[1], activation='relu',
input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=True))
model.add(LSTM(15, activation='relu', return_sequences=True))
model.add(LSTM(7, activation='relu'))
model.add(RepeatVector(n=X_train.shape[1]))
model.add(LSTM(X_train.shape[1], activation='relu', return_sequences=True))
model.add(LSTM(15, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(1)))
```

Οι διαφορές σε σχέση με το 5 είναι στα units των lstm layers. Σκεφτήκαμε ότι αυξάνοντας τα units ίσως καταφέρουμε να μειώσουμε το loss χωρίς να έχουμε overfit αφού και πάλι τα units δεν είναι πάαρ πολλά.

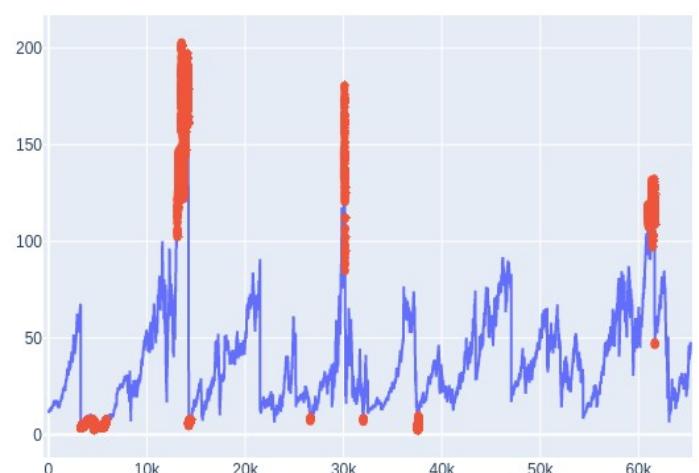
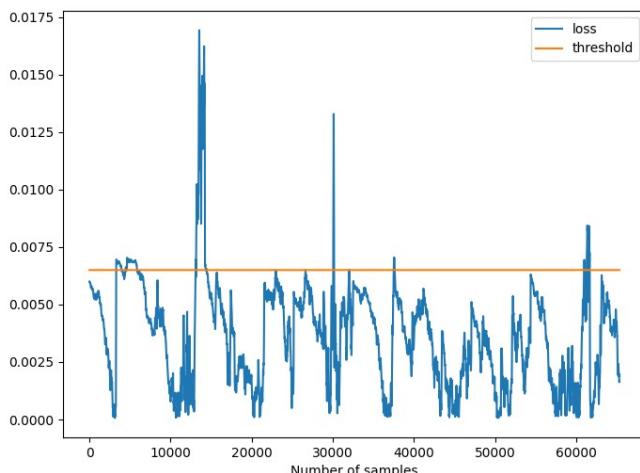
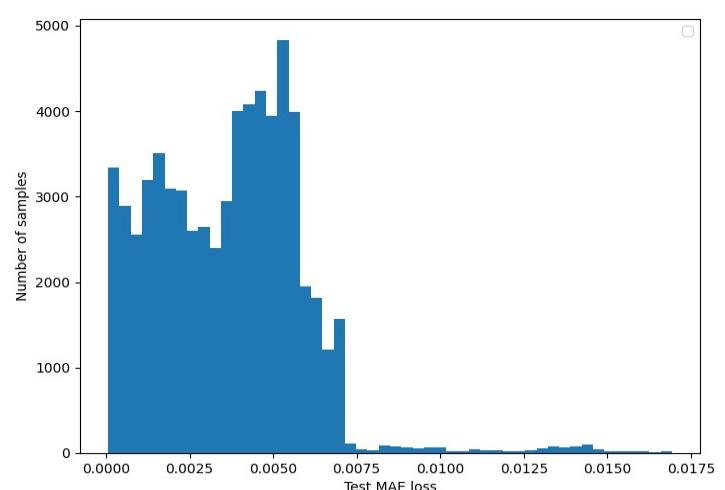
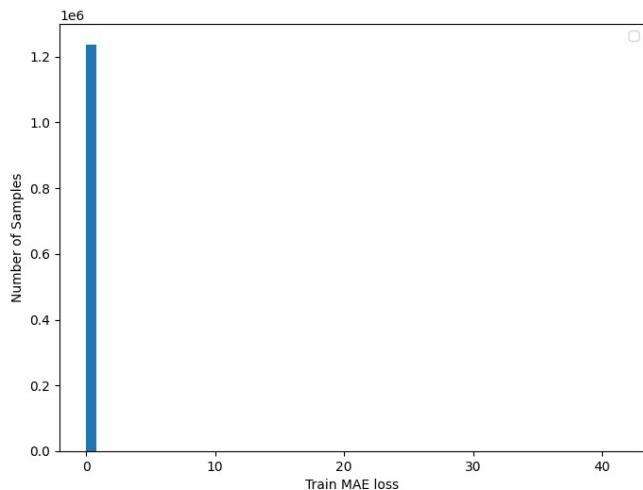


Παρατηρούμε ακριβώς ίδια εικόνα με το μοντέλο 5 τόσο στο test\_loss όσο και στα υπόλοιπα διαγράμματα. Επομένως δεν αρκεί αυτή η αύξηση στα units των lstm layers.

7. Στο στιγμιότυπο αυτό τρεις δοκιμές με διαφορετικές παραμέτρους είχαν ακριβώς ίδιο αποτέλεσμα. Οι διαστάσεις των lstm αυξήθηκαν αρκετά, ενώ προστέθηκαν και στρώματα dropout = 0.2 για να φτιαχτεί το παρακάτω μοντέλο.

```
model = Sequential()
model.add(LSTM(64, activation='relu', input_shape=(X_train.shape[1],
X_train.shape[2]), return_sequences=True))
model.add(Dropout(rate=0.2))
model.add(LSTM(32, activation='relu'))
model.add(Dropout(rate=0.2))
model.add(RepeatVector(n=X_train.shape[1]))
model.add(LSTM(64, activation='relu', return_sequences=True))
model.add(Dropout(rate=0.2))
model.add(LSTM(32, activation='relu', return_sequences=True))
model.add(Dropout(rate=0.2))
model.add(TimeDistributed(Dense(1)))
```

και σε αυτήν την περίπτωση αλλά και όταν αυξήσαμε το dropout σε 0.5 και όταν το αφαιρέσαμε εντελώς σε επόμενη δοκιμή τα διαγράμματα των test και train\_loss και το threshold με τις ανωμαλίες ήταν ακριβώς ίδια. Αυτό συμβαίνει λογικά λόγω των πολλών units των lstm layers όπου εξαιτίας αυτού του γεγονότος το dropout δεν επηρεάζει πολύ την απόδοση του μοντέλου.



## Ερώτημα Γ

Ο Αλγόριθμος κάνει εκπαίδευση στο σύνολο χρονοσειρών. Πιο συγκεκριμένα οι χρονοσειρές του dataset ενώνονται σε μία η οποία γίνεται και scaled. Για την εκπαίδευση χρησιμοποιείται το 95% των τιμών της κάθε χρονοσειράς ,καθώς τα δεδομένα είναι πολλά και αρκεί ένα 5% για validation . Οι χρονοσειρές για είσοδο στο μοντέλο αντί να χωριστούν σε shifted timesteps όπως στα προηγούμενα ερωτήματα διαιρέθηκαν σε μη επικαλυπτόμενα παράθυρα.

Είναι σημαντικό να αναφερθεί πως στην ουσία όσες δοκιμές κι αν έγιναν δεν μπόρεσε να βρεθεί κάποιο αρκετά ικανοποιητικό μοντέλο , ενώ δεν καταφέραμε να πάρουμε ντετερμινιστικά αποτελέσματα για το συγκεκριμένο ερώτημα ακόμα και όταν θέσαμε seed στον κώδικα, καθώς η βιβλιοθήκη του keras δεν είναι καθαρά ντετερμινιστική ως προς το output ειδικά όταν γίνεται train με χρήση GPU.

Συνεπώς και καθώς είναι δύσκολο να βρεθεί το ‘βέλτιστο’ μοντέλο για το συγκεκριμένο ερώτημα παρακάτω παρατίθεται ένα από τα καλύτερα που βρέθηκαν ,το οποίο καταφέραμε να αποθηκεύσουμε κατά τις δοκιμές. Λόγω των κακών αποτελεσμάτων των μοντέλων με ότι παραμέτρους και αν χρησιμοποιήσαμε έγιναν δοκιμές με πάνω από 50 διαφορετικά μοντέλα και παραμέτρους πολλές φορές και επαναληπτικά για κάθε μοντέλο αφού δεν υπήρχε ντετερμινισμός ,ενώ εδώ παρουσιάζονται κάποιες από αυτές .

Το μοντέλο το οποίο χρησιμοποιήθηκε εν τέλη για παραγωγή συμπιεσμένων χρονοσειρών είναι το εξής:

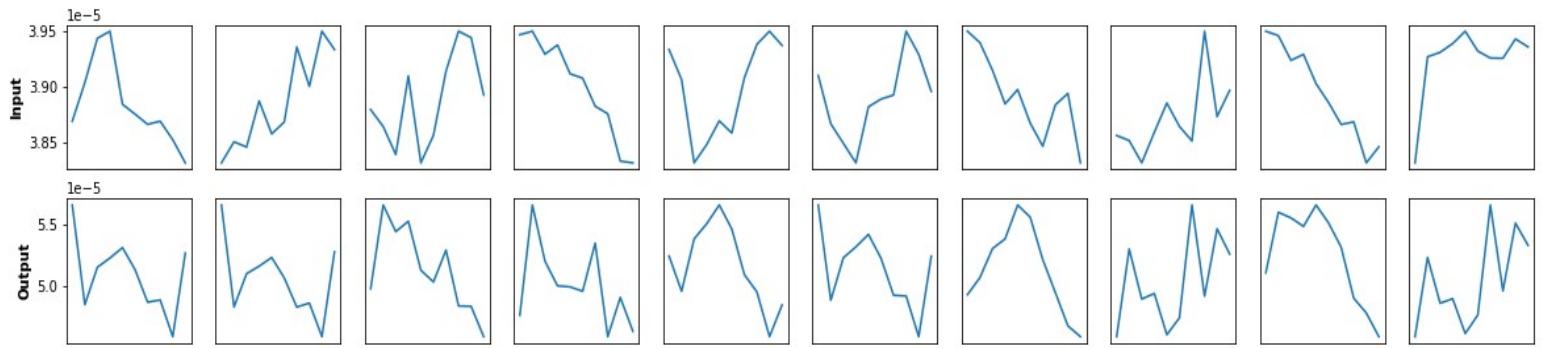
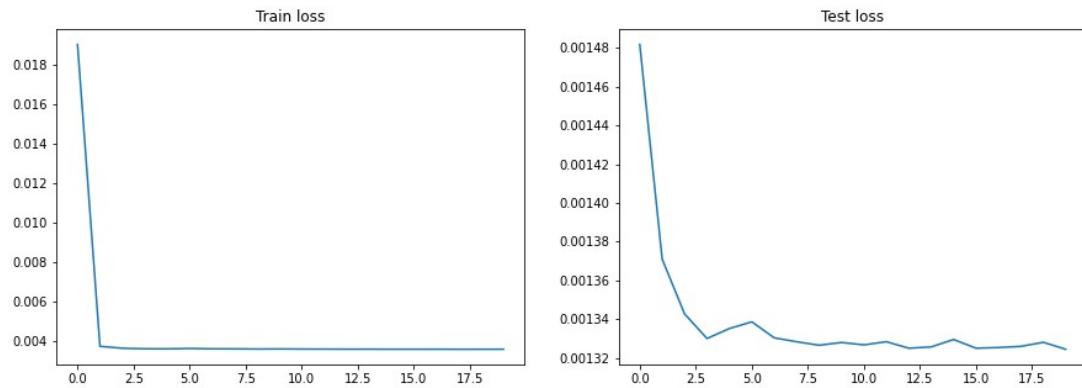
```
input_window = Input(shape=(window_length,1))
x = Conv1D(64, 3, activation="relu", padding="same")(input_window)
x = MaxPooling1D(2, padding="same")(x)
x = Conv1D(32, 3, activation="relu", padding="same")(x)
x = Conv1D(1, 3, activation="relu", padding="same")(x)
encoded = MaxPooling1D(2, padding="same")(x)

encoder = Model(input_window, encoded)

x = Conv1D(1, 3, activation="relu", padding="same")(encoded)
x = Conv1D(32, 3, activation="relu", padding="same")(x)
x = UpSampling1D(2)(x)
x = Conv1D(64, 2, activation='relu')(x)
x = UpSampling1D(2)(x)
decoded = Conv1D(1, 3, activation='sigmoid', padding='same')(x)

decoder = Model(encoded, decoded)
autoencoder = Model(input_window, decoder(encoder(input_window)))

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
history = autoencoder.fit(X_train, X_train,
epochs=20,
batch_size=64,
shuffle=True,
validation_data=(X_test, X_test))
```

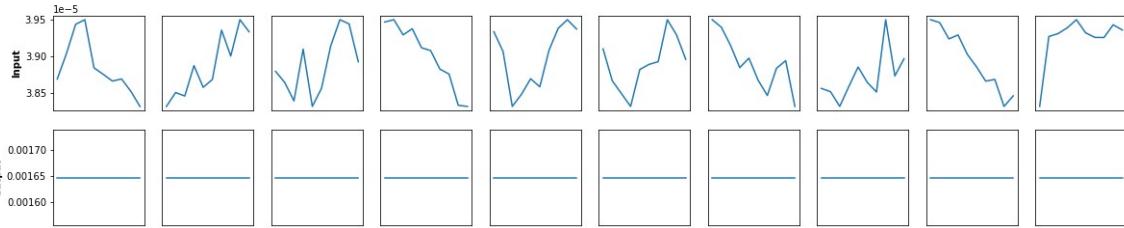


Στο πρώτο διάγραμμα παρατηρούμε το train και test loss του μοντέλου, ενώ στο επόμενο συγκρίνεται η αρχική χρονοσειρά με την decoded σε παράθυρα των 10 δειγμάτων.

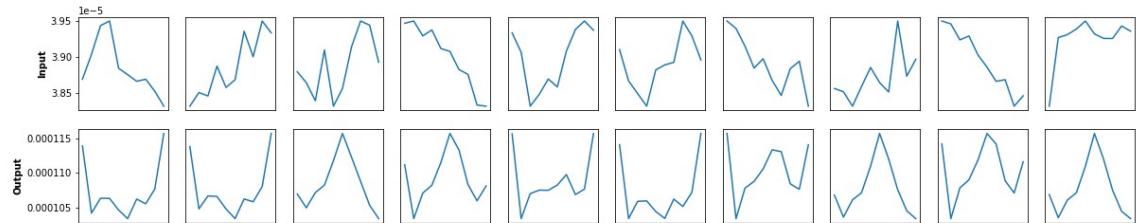
Παρακάτω παρουσιάζονται διάφορα πειράματα με διαφορετικά μοντέλα:

1. Παρακάτω παρουσιάζονται ανακατασκευές του ίδιου μοντέλου με διαφορετικό batch.

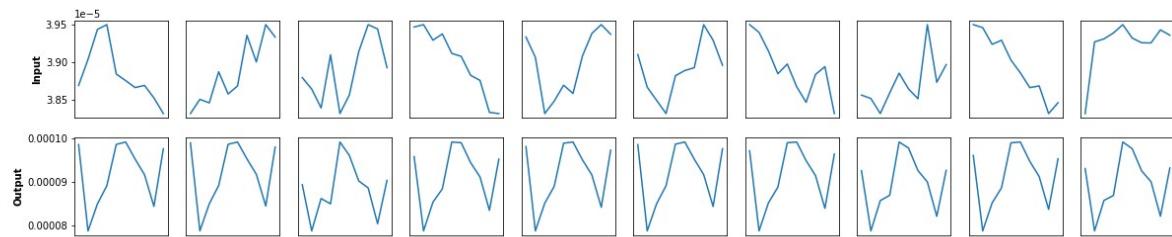
$\alpha.\text{batch} = 32$



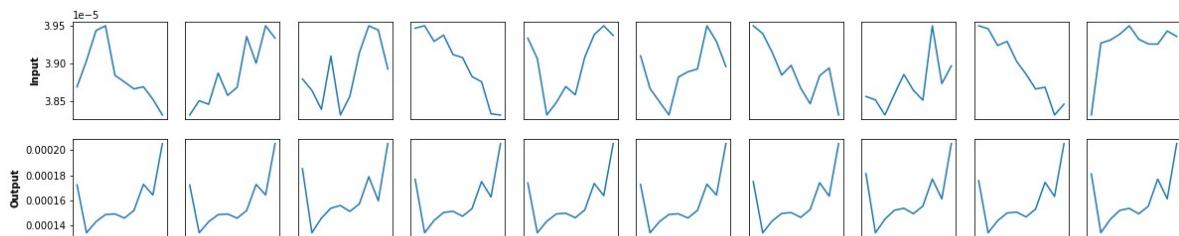
$\beta.\text{batch} = 128$



$\gamma.\text{batch} = 512$

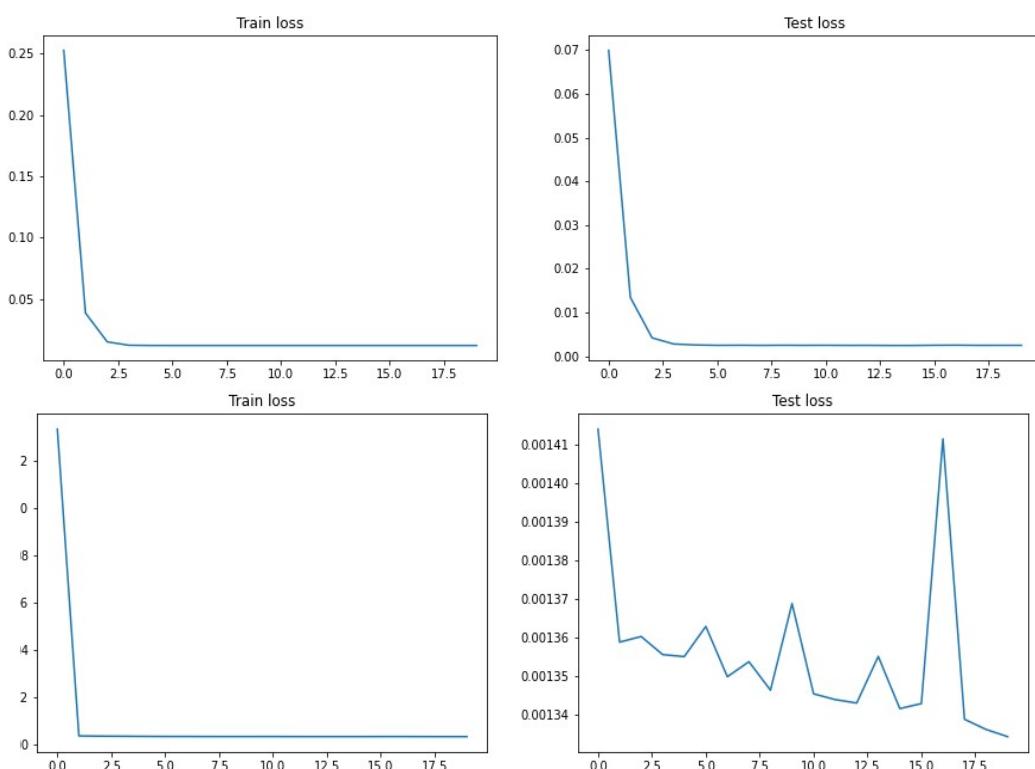


$\delta.\text{batch} = 1024$



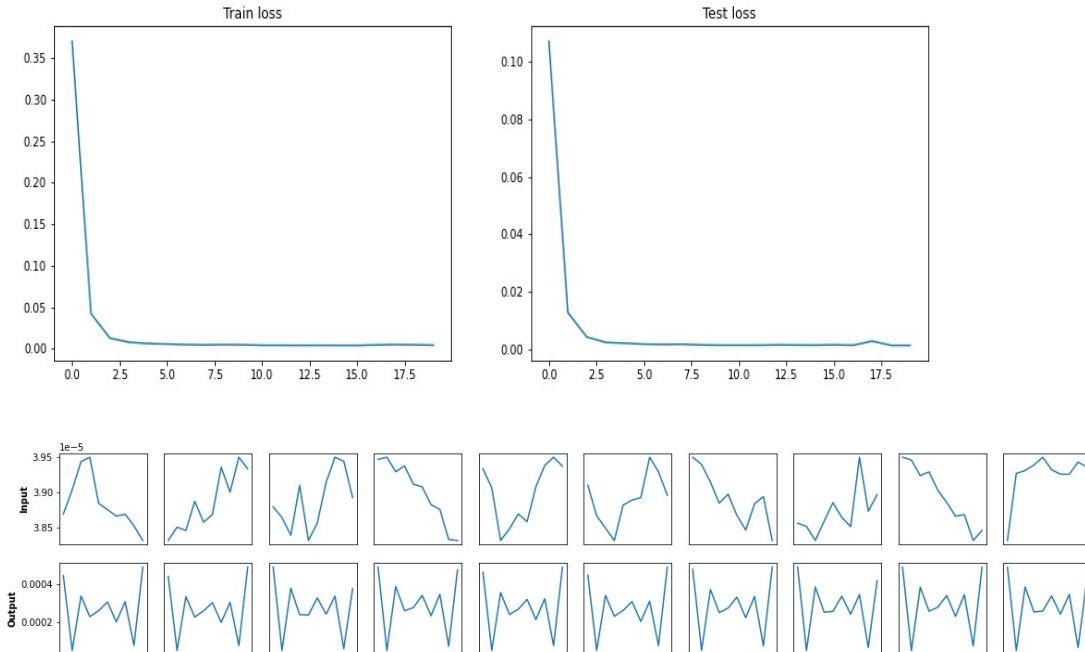
Παρατηρούμε ότι όσο μεγαλύτερο είναι το batch τόσο λιγότερες λεπτομέρειες μπορρεί και ανακατασκευάζει το μοντέλο με βάσει αυτά τα μεγέθη των layers, ενώ καταλήγει με batch = 1024 να κάνει την ίδια ανακατασκευή για οποιοδήποτε παράθυρο input. Επίσης το ίδιο συμβαίνει και με το μικρό batch = 32 στο οποίο είναι σαν να μη γίνεται καμία ανακατασκευή.

Εδώ θα πρέπει να σημειωθεί ότι δεν βρέθηκε κάποια καθαρή σχέση ανάμεσα στο loss\_plot των μοντέλων και στο διάγραμμα ανακατασκευής. Ενδεικτικά παρατίθεται το loss του α πάνω και του γ κάτω.



Βλέπουμε ότι με βάση το loss το batch = 32 θα έπρεπε να κάνει την καλύτερη ανακατασκευή.

2. Ιδιο με το βέλτιστο μοντέλο με batch = 1024 και batchNormalization layers ανάμεσα στα conv layers.



3. Έχουμε ίδια layers με το βέλτιστο όμως μειώνουμε τις διαστάσεις.

```
input_window = Input(shape=(window_length, 1))
x = Conv1D(32, 3, activation="relu", padding="same")(input_window)
x = MaxPooling1D(2, padding="same")(x)
x = Conv1D(16, 3, activation="relu", padding="same")(x)
x = Conv1D(1, 3, activation="relu", padding="same")(x)
encoded = MaxPooling1D(2, padding="same")(x)

encoder = Model(input_window, encoded)

x = Conv1D(1, 3, activation="relu", padding="same")(encoded)

x = Conv1D(16, 3, activation="relu", padding="same")(x)
x = UpSampling1D(2)(x)
x = Conv1D(32, 2, activation='relu')(x)
x = UpSampling1D(2)(x)
decoded = Conv1D(1, 3, activation='sigmoid', padding='same')(x)

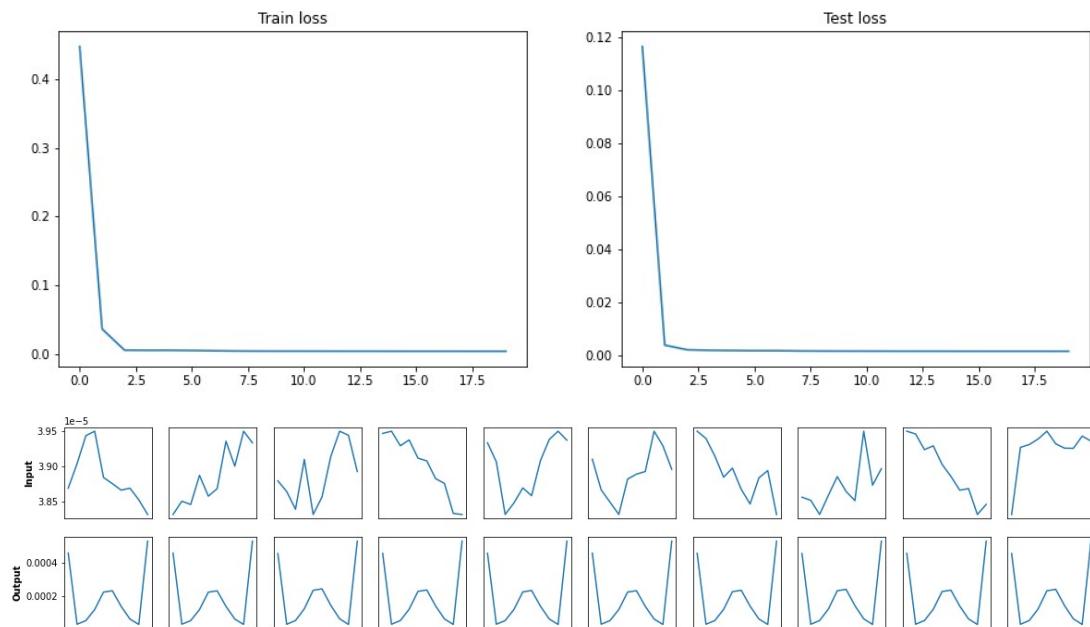
decoder = Model(encoded, decoded)
```

```

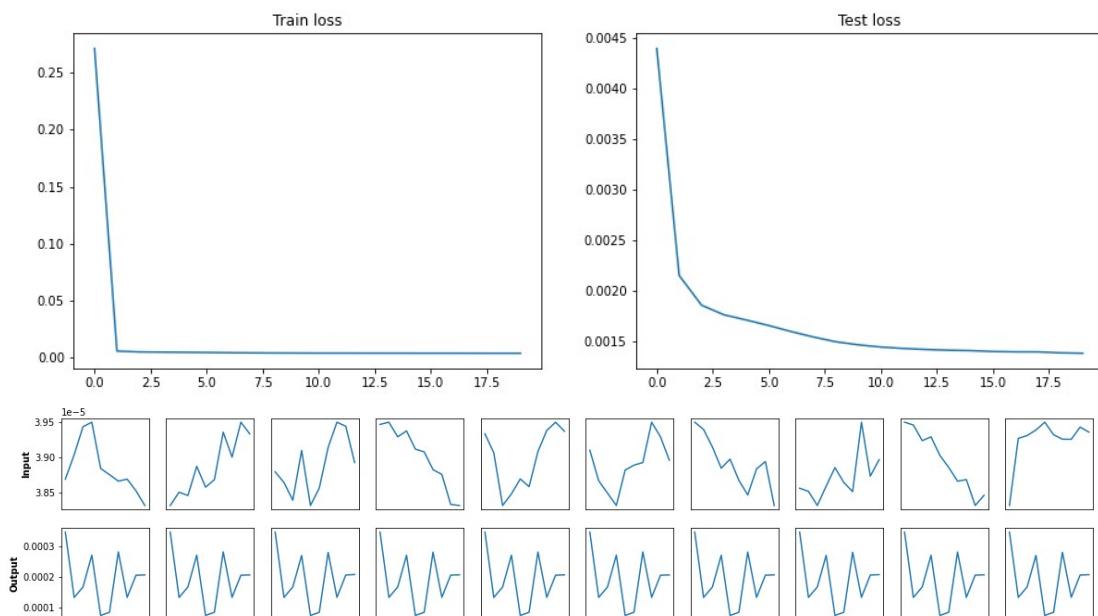
autoencoder = Model(input_window, decoder(encoder(input_window)))

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
history = autoencoder.fit(X_train, X_train,
epochs=20,
batch_size=1024,
shuffle=True,
validation_data=(X_test, X_test))

```



4. Ιδιο μοντέλο με το 3 όμως μειώνουμε τις διαστάσεις στο μισό για να δούμε τι επίδραση έχουν.



Βλέπουμε ότι αυτή η δραστική μείωση είχε μείωση στο loss δεν είχε όμως και καλύτερα αποτελέσματα.

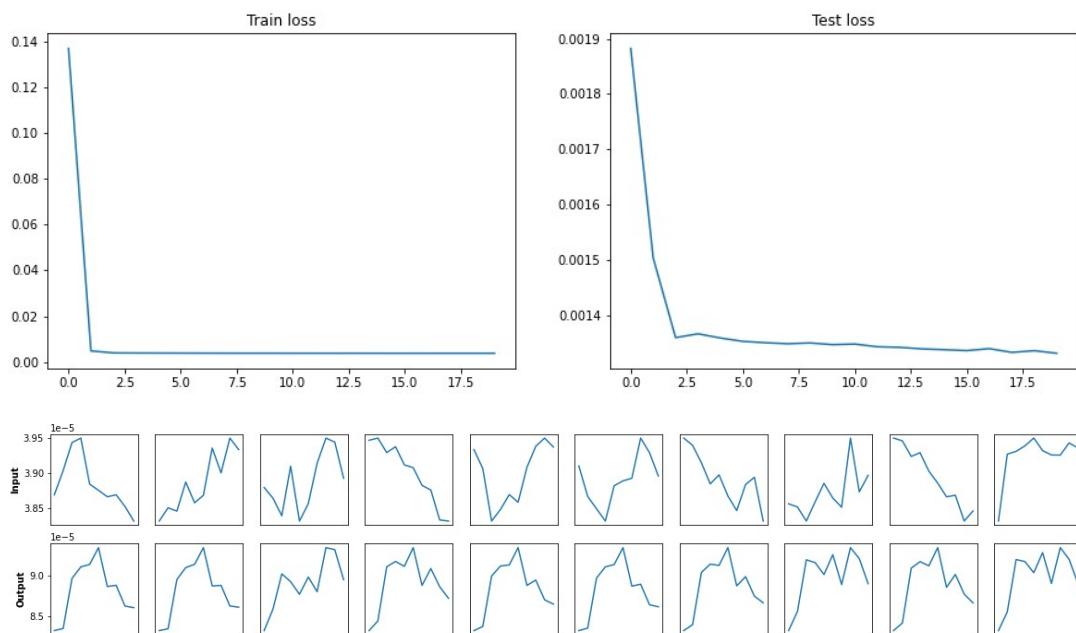
5. Στην συγκεκριμένη δοκιμή προσθέτουμε από ένα conv layer στον encoder και στον decoder ενώ είχαμε και batch = 1024

```
input_window = Input(shape=(window_length,1))
x = Conv1D(64, 3, activation="relu", padding="same")(input_window)
x = Conv1D(32, 3, activation="relu", padding="same")(x)
x = MaxPooling1D(2, padding="same")(x)
x = Conv1D(16, 3, activation="relu", padding="same")(x)
x = Conv1D(1, 3, activation="relu", padding="same")(x)
encoded = MaxPooling1D(2, padding="same")(x)

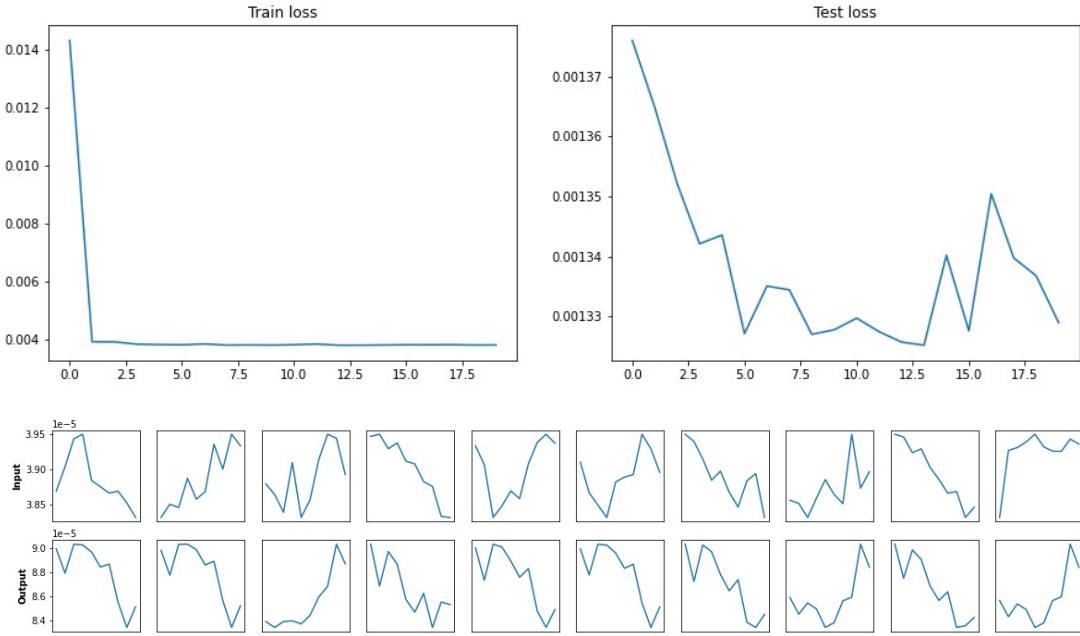
encoder = Model(input_window, encoded)

x = Conv1D(1, 3, activation="relu", padding="same")(encoded)
x = Conv1D(16, 3, activation="relu", padding="same")(x)
x = UpSampling1D(2)(x)
x = Conv1D(32, 3, activation="relu", padding="same")(x)
x = Conv1D(64, 2, activation='relu')(x)
x = UpSampling1D(2)(x)
decoded = Conv1D(1, 3, activation='sigmoid', padding='same')(x)
decoder = Model(encoded,decoded)

autoencoder = Model(input_window, decoder(encoder(input_window)))
```



Στην συνέχεια γίνανε πειράματα με το batch size με καλύτερα αποτελεσματα να έχουμε για batch = 64.



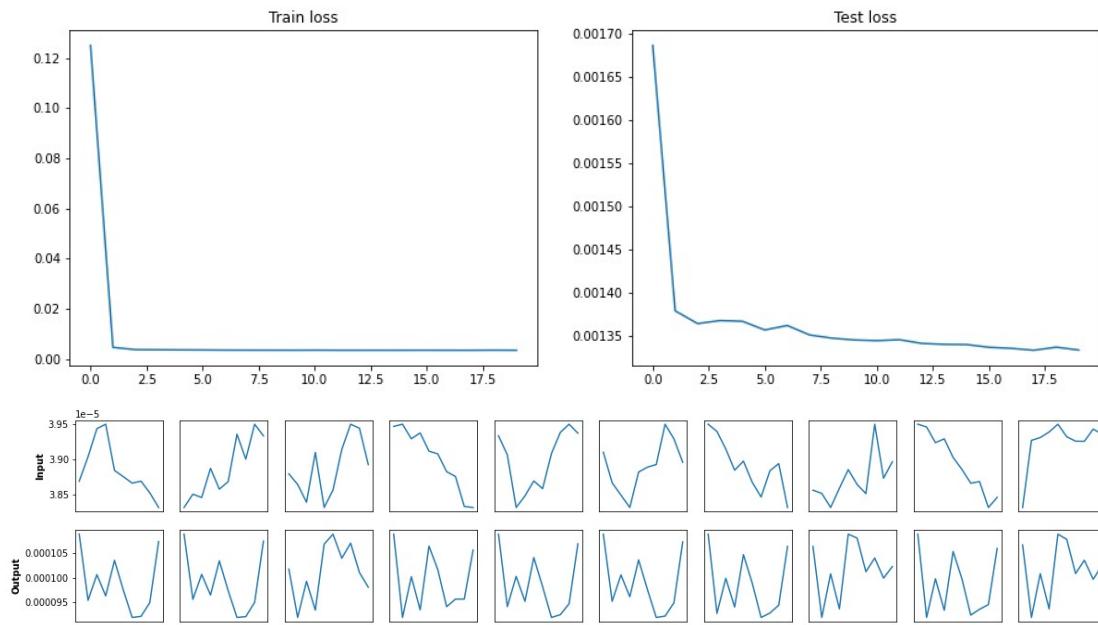
Το συγκεκριμένο ήταν από τα καλύτερα πειράματα που έγιναν ,όμως δεν καταφέραμε να αποθηκεύσουμε το μοντέλο καθώς όταν ξανατρέξαμε το πρόγραμμα πήραμε διαφορετικά αποτελέσματα με απογοητευτική ανακατασκευή λόγω μη ντετερμινισμού.

6. Το συγκεκριμένο μοντέλο είναι ίδιο σε αριθμό layers με το προηγούμενο και έχει αλλαγμένες διαστάσεις. Το batch = 1024 ήταν αυτό που δόνησε καλύτερα για το συγκεκριμένο μοντέλο..

```
input_window = Input(shape=(window_length,1))
x = Conv1D(64, 3, activation="relu", padding="same")(input_window)
x = Conv1D(48, 3, activation="relu", padding="same")(x)
x = MaxPooling1D(2, padding="same")(x)
x = Conv1D(32, 3, activation="relu", padding="same")(x)
x = Conv1D(1, 3, activation="relu", padding="same")(x)
encoded = MaxPooling1D(2, padding="same")(x)
encoder = Model(input_window, encoded)

x = Conv1D(1, 3, activation="relu", padding="same")(encoded)
x = Conv1D(32, 3, activation="relu", padding="same")(x)
x = UpSampling1D(2)(x)
x = Conv1D(48, 3, activation="relu", padding="same")(x)
x = Conv1D(64, 2, activation='relu')(x)
x = UpSampling1D(2)(x)
decoded = Conv1D(1, 3, activation='sigmoid', padding='same')(x)
decoder = Model(encoded, decoded)
```

```
autoencoder = Model(input_window, decoder(encoder(input_window)))
```

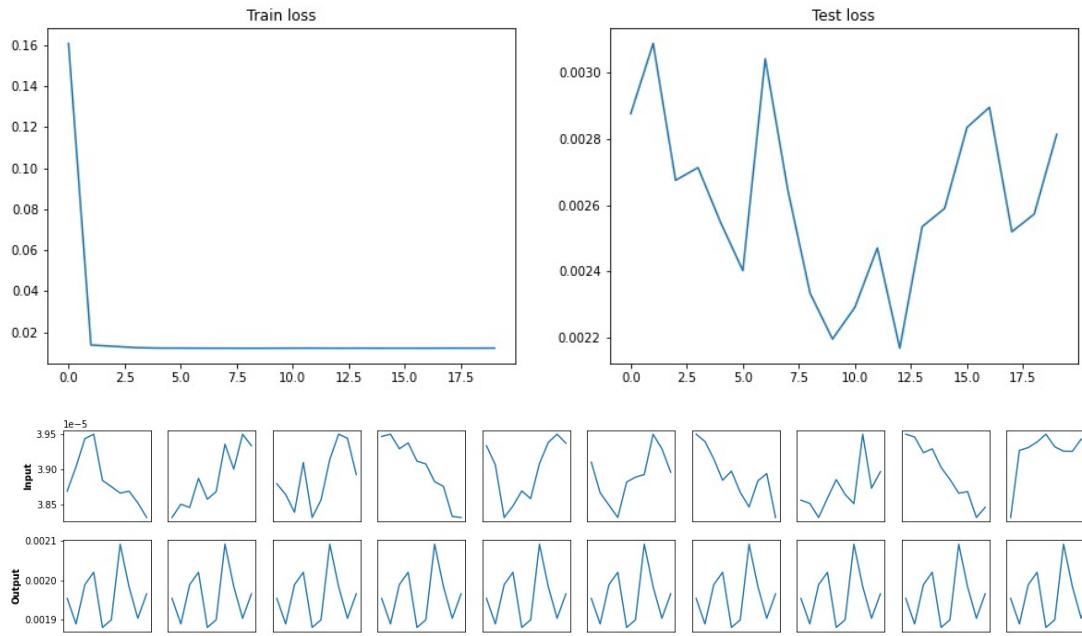


7. Εδώ έχουμε την προσθήκη ενός conv layer ακόμα στον encoder και στον decoder με batch = 1024.

```
input_window = Input(shape=(window_length,1))
x = Conv1D(64, 3, activation="relu", padding="same")(input_window)
x = Conv1D(48, 3, activation="relu", padding="same")(x)
x = MaxPooling1D(2, padding="same")(x)
x = Conv1D(32, 3, activation="relu", padding="same")(x)
x = Conv1D(16, 3, activation="relu", padding="same")(x)
x = Conv1D(1, 3, activation="relu", padding="same")(x)
encoded = MaxPooling1D(2, padding="same")(x)

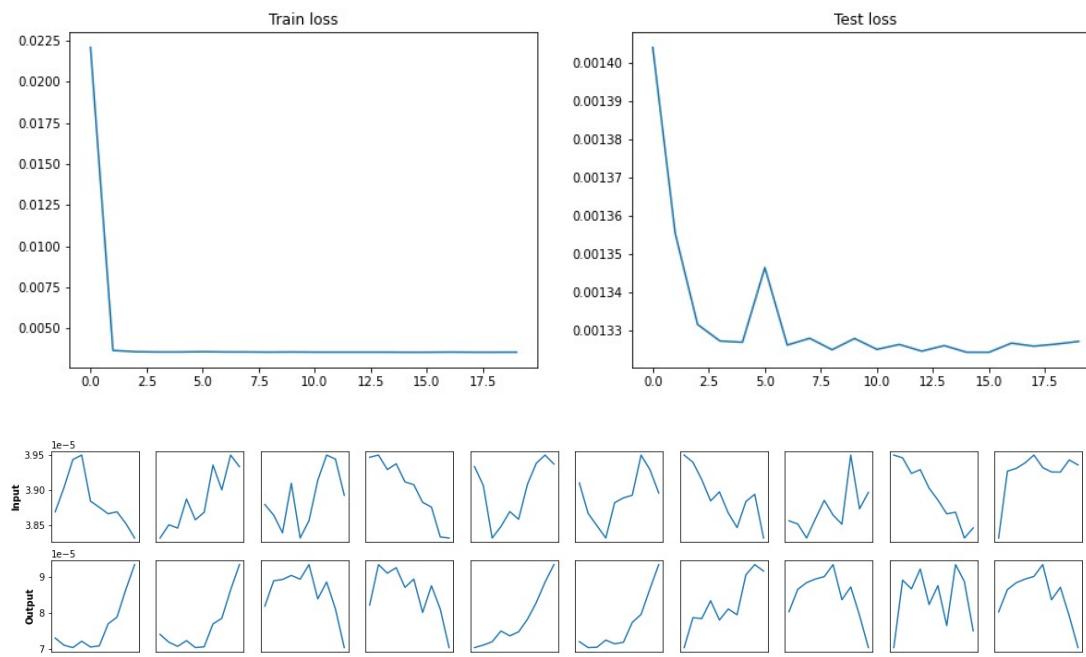
encoder = Model(input_window, encoded)

x = Conv1D(1, 3, activation="relu", padding="same")(encoded)
x = Conv1D(16, 3, activation="relu", padding="same")(x)
x = Conv1D(32, 3, activation="relu", padding="same")(x)
x = UpSampling1D(2)(x)
x = Conv1D(48, 3, activation="relu", padding="same")(x)
x = Conv1D(64, 2, activation='relu')(x)
x = UpSampling1D(2)(x)
decoded = Conv1D(1, 3, activation='sigmoid', padding='same')(x)
```



Παρατηρούμε ότι η αύξηση στις διαστάσεις των layers αλλά και η αύξηση του πλήθους τους δεν βοήθησε στο να ερμηνευτούν με τον σωστό τρόπο τα διάφορα μοτίβα στα windows του input.

8. Στο συγκεκριμένο παράδειγμα δοκιμάζουμε και την επίδραση που θα έχει ένα διαφορετικό window. Ήτοι αυξάνουμε το window σε 12, ενώ μετά από δοκιμές με batch = 1024, 512, 128 και 64 παρουσιάζουμε αυτό με batch = 64 καθώς είχε καλύτερη ανακατασκευή.



9. Στην συνέχεια δοκιμάζουμε να αυξήσουμε και άλλο το window στην τιμή 24 ώστε να μπορέσουμε να προσθέσουμε κι άλλο MaxPooling layer και καταλήγουμε στο εξής μοντέλο.

```

input_window = Input(shape=(window_length, 1))
x = Conv1D(64, 3, activation="relu", padding="same")(input_window)
x = Conv1D(32, 3, activation="relu", padding="same")(x)
x = MaxPooling1D(2, padding="same")(x)
x = Conv1D(16, 3, activation="relu", padding="same")(x)
x = MaxPooling1D(2, padding="same")(x)
x = Conv1D(8, 3, activation="relu", padding="same")(x)
encoded = MaxPooling1D(2, padding="same")(x)

encoder = Model(input_window, encoded)

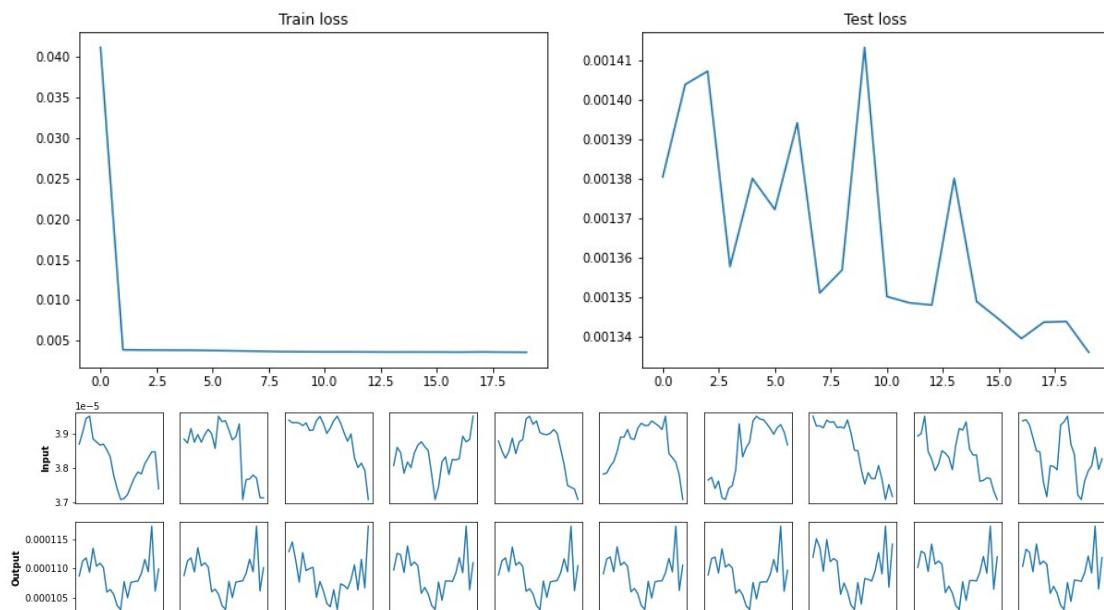
x = Conv1D(8, 3, activation="relu", padding="same")(encoded)
x = UpSampling1D(2)(x)
x = Conv1D(16, 3, activation="relu", padding="same")(x)
x = UpSampling1D(2)(x)
x = Conv1D(32, 3, activation="relu", padding="same")(x)
x = Conv1D(64, 2, activation='relu', padding = "same")(x)
x = UpSampling1D(2)(x)
decoded = Conv1D(1, 3, activation='sigmoid', padding='same')(x)

decoder = Model(encoded, decoded)

autoencoder = Model(input_window, decoder(encoder(input_window)))

```

Για το συγκεκριμένο μοντέλο το batch που δούλεψε καλύτερα ήταν το batch = 64



Παρατηρούμε και σε αυτήν την περίπτωση ότι έχουμε ίδια ανακατασκευή σε κάθε στιγμιότυπο. Ενώ έχουμε και ένα περίεργο διάγραμμα στο test\_loss.

10. Δοκιμάστηκε και το μοντέλο το οποίο υπήρχε στο tutorial που μας είχατε υποδείξει για το συγκεκριμένο ερώτημα το οποίο είναι το εξής.

```
input_window = Input(shape=(window_length, 1))

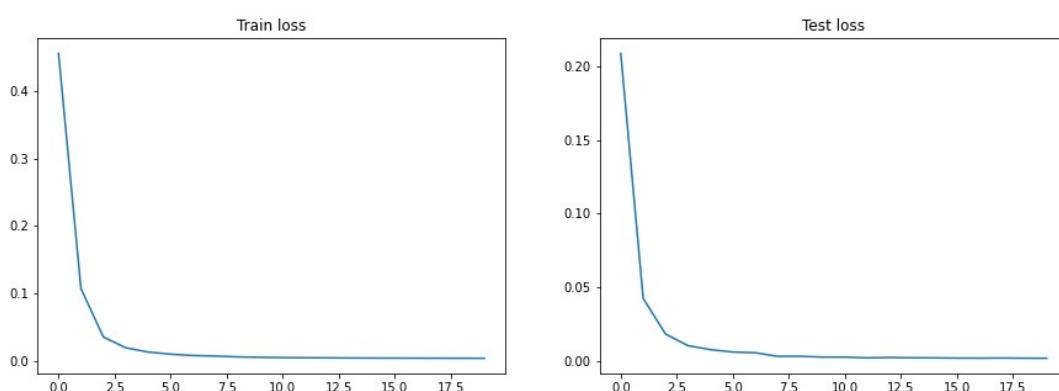
x = Conv1D(16, 3, activation="relu", padding="same")(input_window)
x = BatchNormalization()(x)
x = MaxPooling1D(2, padding="same")(x)
x = Conv1D(1, 3, activation="relu", padding="same")(x)
x = BatchNormalization()(x)
encoded = MaxPooling1D(2, padding="same")(x)

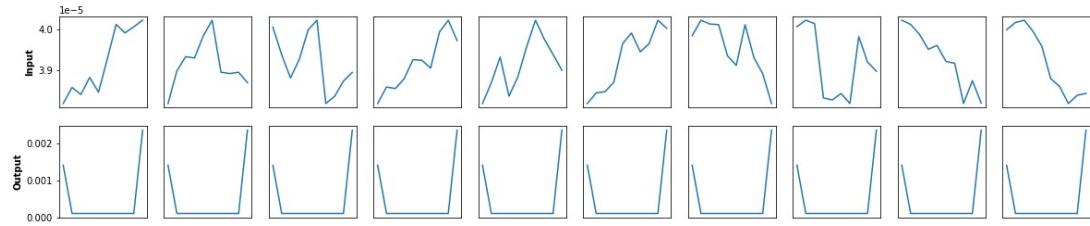
encoder = Model(input_window, encoded)

x = Conv1D(1, 3, activation="relu", padding="same")(encoded)
x = BatchNormalization()(x)
x = UpSampling1D(2)(x)
x = Conv1D(16, 2, activation='relu')(x)
x = BatchNormalization()(x)
x = UpSampling1D(2)(x)
decoded = Conv1D(1, 3, activation='sigmoid', padding='same')(x)

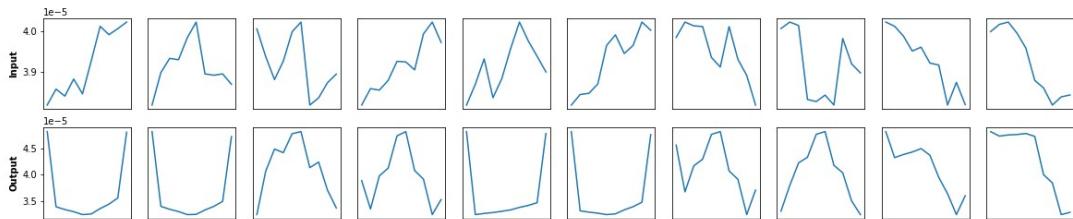
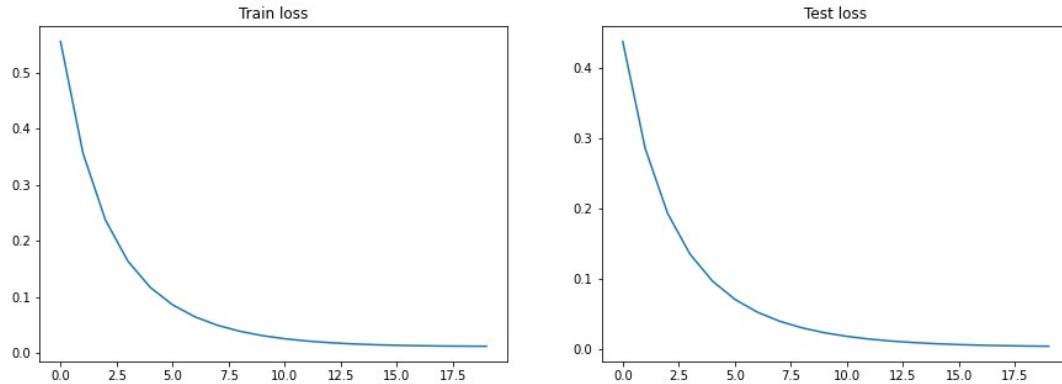
autoencoder = Model(input_window, decoded)
```

Το καλύτερο batch για το συγκεκριμένο μοντέλο ήταν το batch = 64.





Τέλος αφαιρέθηκαν από το συγκεκριμένο μοντέλο τα batchNormalization layers και είχαμε λίγο καλύτερο αποτέλεσμα.



## Ερώτημα Δ

Στο συγκεκριμένο ερώτημα θα γίνει σύγκριση των συμπιεσμένων χρονοσειρών του ερωτήματος Γ με αυτά των αρχικών χρονοσειρών της 2ης εργασίας. Να σημειωθεί ότι για οικονομία χρόνου χρησιμοποιήθηκαν οι πρώτες 50 χρονοσειρές ως dataset από το σετ που είχε δοθεί, ενώ χρησημοποιήθηκαν οι 51-53 για queryset . Στο αριστερό μέρος είναι τα αποτελέσματα του αρχικού dataset ενώ δεξιά του συμπιεσμένου.

1.search με LSH και euclidean distance:

```
1
2 Query: bcr
3 Algorithm: LSH_VECTOR
4 Approximate Nearest neighbor-0: antm
5 True Nearest neighbor-0: antm
6 distanceApproximate: 212.461
7 distanceTrue: 212.461
8
9 Query: bdx
10 Algorithm: LSH_VECTOR
11 Approximate Nearest neighbor-0: apd
12 True Nearest neighbor-0: apd
13 distanceApproximate: 117.768
14 distanceTrue: 117.768
15
16 Query: ben
17 Algorithm: LSH_VECTOR
18 Approximate Nearest neighbor-0: ash
19 True Nearest neighbor-0: ash
20 distanceApproximate: 137.241
21 distanceTrue: 137.241
22
23
24 tApproximateAverage: 0
25 tTrueAverage: 2.31579e-310
26 MAF: 1 [Maximum Approximation Factor]
```

```
1
2 Query: bcr
3 Algorithm: LSH_VECTOR
4 Approximate Nearest neighbor-0: Not found
5 True Nearest neighbor-0: ba
6 distanceApproximate: --
7 distanceTrue: 104490
8
9 Query: bdx
10 Algorithm: LSH_VECTOR
11 Approximate Nearest neighbor-0: Not found
12 True Nearest neighbor-0: apd
13 distanceApproximate: --
14 distanceTrue: 37665.3
15
16 Query: ben
17 Algorithm: LSH_VECTOR
18 Approximate Nearest neighbor-0: Not found
19 True Nearest neighbor-0: adm
20 distanceApproximate: --
21 distanceTrue: 19348.8
22
23
24
25 tApproximateAverage: 2.30384e-310
26 tTrueAverage: 2.30384e-310
27 MAF: 0 [Maximum Approximation Factor]
```

Βλέπουμε ότι στο τρίτο ερώτημα βρέθηκε ο ίδιος True Nearest Neighbor όμως λόγω κακής συμπίεσης δεν βρέθηκε κανένας Approximate από το συμπιεσμένο σετ. Σε άλλες εκτελέσεις που κάναμε όμως δυστυχώς δεν αποθηκεύτηκαν είχαμε καταφέρει με το συμπιεσμένο dataset να βρούμε και Approximate σε δύο από τα τρια ερωτήματα ο οποίος να ταυτίζεται με τον True.

2.search με Hypercube και euclidean distance και probes = 2:

```
1 Query: bcr
2 Algorithm: Hypercube
3 Approximate Nearest neighbor-0: azo
4 True Nearest neighbor-0: antm
5 distanceApproximate: 1166.38
6 distanceTrue: 212.461
7
8 Query: bdx
9 Algorithm: Hypercube
10 Approximate Nearest neighbor-0: all
11 True Nearest neighbor-0: apd
12 distanceApproximate: 217.092
13 distanceTrue: 117.768
14
15 Query: ben
16 Algorithm: Hypercube
17 Approximate Nearest neighbor-0: ash
18 True Nearest neighbor-0: ash
19 distanceApproximate: 137.241
20 distanceTrue: 137.241
21
22
23 tApproximateAverage: 5.43472e-323
24 tTrueAverage: 2.30003e-310
25 MAF: 5.48986 [Maximum Approximation Factor]
```

```
1 Query: bcr
2 Algorithm: Hypercube
3 Approximate Nearest neighbor-0: ba
4 True Nearest neighbor-0: ba
5 distanceApproximate: 104490
6 distanceTrue: 104490
7
8 Query: bdx
9 Algorithm: Hypercube
10 Approximate Nearest neighbor-0: apd
11 True Nearest neighbor-0: apd
12 distanceApproximate: 37665.3
13 distanceTrue: 37665.3
14
15 Query: ben
16 Algorithm: Hypercube
17 Approximate Nearest neighbor-0: aee
18 True Nearest neighbor-0: adm
19 distanceApproximate: 33539
20 distanceTrue: 19348.8
21
22
23
24 tApproximateAverage: 5.43472e-323
25 tTrueAverage: 2.31692e-310
26 MAF: 1.73339 [Maximum Approximation Factor]
```

Εδώ μόνο στο δεύτερο ερώτημα bdx βρέθηκε ο ίδιος True Nearest Neighbor , δεν είχαμε όμως συμφωνία και στον Approximate. Παρ' όλα αυτά ο Hypercube κατάφερε να βρει Approximate NN σε όλα τα ερωτήματα.

### 3.search με LSH και discrete Frechet:

```
1
2 Query: bcr
3 Algorithm: LSH_Frechet_Discrete
4 Approximate Nearest neighbor-0: aa
5 True Nearest neighbor-0: antm
6 distanceApproximate: 40.974
7 distanceTrue: 15.423
8
9 Query: bdx
10 Algorithm: LSH_Frechet_Discrete
11 Approximate Nearest neighbor-0: apd
12 True Nearest neighbor-0: apd
13 distanceApproximate: 8.47818
14 distanceTrue: 8.47818
15
16 Query: ben
17 Algorithm: LSH_Frechet_Discrete
18 Approximate Nearest neighbor-0: aep
19 True Nearest neighbor-0: aep
20 distanceApproximate: 12.172
21 distanceTrue: 12.172
22
23
24
25 tApproximateAverage: 17
26 tTrueAverage: 70
27 MAF: 2.65668 [Maximum Approximation Factor]
```

```
1
2 Query: bcr
3 Algorithm: LSH_Frechet_Continuous
4 Approximate Nearest neighbor-0: Not found
5 True Nearest neighbor-0: ba
6 distanceApproximate: --
7 distanceTrue: 17996.2
8
9 Query: bdx
10 Algorithm: LSH_Frechet_Continuous
11 Approximate Nearest neighbor-0: Not found
12 True Nearest neighbor-0: amgn
13 distanceApproximate: --
14 distanceTrue: 5609.18
15
16 Query: ben
17 Algorithm: LSH_Frechet_Continuous
18 Approximate Nearest neighbor-0: Not found
19 True Nearest neighbor-0: abt
20 distanceApproximate: --
21 distanceTrue: 2598.72
22
23
24
25 tApproximateAverage: 2.303e-310
26 tTrueAverage: 62
27 MAF: 0 [Maximum Approximation Factor]
```

Εδώ βλέπουμε ότι ο αλγόριθμος έχει δουλέψει καλά με το αρχικό αρχείο , καθόλου καλά όμως με το συμπιεσμένο αφού δεν ταυτίζονται ούτε οι True Nearest Neighbors των δύο αρχείων . Τα αποτελέσματα περιμέναμε ότι θα είναι χειρότερα λόγω και του filtering.

#### 4.search με LSH και continuous Frechet:

```
1
2 Query: bcr
3 Algorithm: LSH_Frechet_Continuous
4 Approximate Nearest neighbor-0: bbt
5 True Nearest neighbor-0: apa
6 distanceApproximate: 46.77
7 distanceTrue: 6.305
8
9 Query: bdx
10 Algorithm: LSH_Frechet_Continuous
11 Approximate Nearest neighbor-0: Not found
12 True Nearest neighbor-0: apd
13 distanceApproximate: --
14 distanceTrue: 4.881
15
16 Query: ben
17 Algorithm: LSH_Frechet_Continuous
18 Approximate Nearest neighbor-0: abc
19 True Nearest neighbor-0: aem
20 distanceApproximate: 14.448
21 distanceTrue: 6.387
22
23
24
25 tApproximateAverage: 1.33333
26 tTrueAverage: 49
27 MAF: 7.41792 [Maximum Approximation Factor]
```

```
1
2 Query: bcr
3 Algorithm: LSH_Frechet_Continuous
4 Approximate Nearest neighbor-0: Not found
5 True Nearest neighbor-0: ba
6 distanceApproximate: --
7 distanceTrue: 17996.2
8
9 Query: bdx
10 Algorithm: LSH_Frechet_Continuous
11 Approximate Nearest neighbor-0: Not found
12 True Nearest neighbor-0: amgn
13 distanceApproximate: --
14 distanceTrue: 5609.18
15
16 Query: ben
17 Algorithm: LSH_Frechet_Continuous
18 Approximate Nearest neighbor-0: Not found
19 True Nearest neighbor-0: abt
20 distanceApproximate: --
21 distanceTrue: 2598.72
22
23
24
25 tApproximateAverage: 2.303e-310
26 tTrueAverage: 62
27 MAF: 0 [Maximum Approximation Factor]
```

Και με continuous frechet έχουμε παρόμοια αποτελέσματα με του discrete.

## 5.clustering Classic με Mean Vector:

To clustering με το αρχικό dataset:

Algorithm Lloyds

CLUSTER-0 {centroid,bb, admp, amzn, apa, apd, atro, avy, axgn, all, au, axp, ba, bac, bbb, aaba, adi, adp, aee, afl, agn, apc, avp, bbt, bby, bc, adbe, adsk, aet, aiv, a, abx, aep, amat, amd, ash, ati, bb, abc, abt, adm, aem, an, bax, aapl, aes, azo, aa, amgn, antm, }

CLUSTER-1 {centroid,aphb, }

CLUSTER-2 {centroid,admp, }

Silhouette: [0.967207,0,0,0.322402]

To clustering με το συμπιεσμένο dataset:

Algorithm Lloyds

CLUSTER-0 {centroid,bby, aa, all, amgn, bac, bb, abt, adp, aem, atro, ba, bax, adm, aet, agn, bbb, aiv, avp, bbt, bc, aee, amat, ati, axp, aapl, afl, amzn, au, a, abx, adsk, an, apc, apd, axgn, aaba, adi, antm, avy, azo, bby, admp, adbe, aes, amd, apa, abc, aep, ash, }

CLUSTER-1 {centroid,bax, }

CLUSTER-2 {centroid,aa, aphb, aig, }

Silhouette: [0.185801,0,0.00195141,0.0625842]

Βλέπουμε ότι και στις δύο περιπτώσεις τα σημεία έχουν συγκεντρωθεί στα ίδια clusters με καλύτερη τιμή silhouette για το clustering με το αρχικό dataset.

6.clustering Classic Mean Frechet:

To clustering με το αρχικό dataset:

Algorithm Lloyds

CLUSTER-0 {centroid,adbe, abx, adm, amzn, a, abc, abt, amd, an, ash, bax, bb, aa, amat, aem, amgn, ati, au, aiv, bbb, aee, aes, agn, antm, apa, aaba, adi, admp, adp, aet, afl, apc, avp, avy, azo, bbt, bby, bc, aapl, adbe, adsk, aep, aig, atro, axgn, ba, all, apd, axp, bac, }

CLUSTER-1 {centroid,aphb, }

CLUSTER-2 {centroid,abx, }

To clustering με το συμπιεσμένο dataset:

Algorithm Lloyds

CLUSTER-0 {centroid,aphb, aig, amzn, azo, }

CLUSTER-1 {centroid,amat, admp, aiv, amat, antm, avp, axgn, bc, bb, aapl, adbe, adp, aes, apa, au, axp, ba, bax, aa, abx, adsk, amgn, apc, bac, bbb, abt, aee, aem, aep, atro, bby, abc, aet, amd, apd, ati, bbt, a, aaba, adi, adm, afl, all, an, ash, avy, }

CLUSTER-2 {centroid,admp, agn, }

Και με την χρήση Mean Frechet έχουμε παρόμοια αποτελέσματα με τα δύο αρχεία.

7.clustering LSH με Mean Vector:

To clustering με το αρχικό dataset:

Algorithm: Range Search LSH

CLUSTER-0 {centroid,admp, aig, aphb, }

CLUSTER-1 {centroid,bbby, abx, aee, aep, afl, amzn, apc, au, axp, bac, bbbby, bby, bc, adbe, adi, adp, adsk, agn, aiv, avp, bbt, aet, adm, amgn, antm, apa, ash, bb, azo, ati, all, ba, aa, aaba, apd, avy, }

CLUSTER-2 {centroid,aapl, aapl, atro, axgn, a, abt, amd, an, abc, aem, aes, amat, bax, }

To clustering με το συμπιεσμένο dataset:

Algorithm: Range Search LSH

CLUSTER-0 {centroid,abc, adp, amgn, adbe, axp, agn, aet, antm, apa, apc, apd, amzn, ba, aapl, }

CLUSTER-1 {centroid,apc, aphb, azo, aig, }

CLUSTER-2 {centroid,a, aee, aep, aaba, bax, bby, abx, adi, amat, avp, aes, amd, afl, bbbby, aem, aa, axgn, a, abt, adsk, all, atro, au, bac, bc, adm, ash, ati, aiv, abc, admp, avy, an, bb, bbt, }

Και πάλι έχουμε παρόμοια συγκέντρωση των διανυσμάτων στα clusters.

8.clustering Hypercube με Mean Vector:

To clustering με το αρχικό dataset:

Algorithm: Range Search Hypercube

CLUSTER-0 {centroid,amzn, a, aaba, abc, abt, abx, adbe, adi, adm, adp, adsk, aee, aem, aep, aet, afl, aiv, amat, amd, an, apc, ash, avp, bax, bb, bbt, bby, aapl, aes, agn, atro, bac, bbb, bc, ati, axgn, all, amzn, au, axp, }

CLUSTER-1 {centroid,aphb, }

CLUSTER-2 {centroid,aa, admp, aig, amgn, antm, apa, apd, avy, ba, azo, }

To clustering με το συμπιεσμένο dataset:

Algorithm: Range Search Hypercube

CLUSTER-0 {centroid,amgn, aphb, azo, aig, }

CLUSTER-1 {centroid,abt, bby, aaba, aiv, bax, abt, adm, au, avp, ash, bac, bc, aee, amat, ati, bbb, amd, axgn, aa, apa, an, atro, bbt, a, abx, aes, bb, adi, admp, aem, aep, }

CLUSTER-2 {centroid,aaba, adbe, aet, all, amgn, antm, avy, agn, adp, apc, axp, abc, aapl, adsk, afl, amzn, apd, ba, }

Silhouette: [0.254576,0.500829,0.363831,0.373079]

Τα αποτελέσματα είναι ξανά καλά , ενώ έχουμε και αρκετα καλό silhouette για το συμπιεσμένο dataset.

## 9.clustering LSH Mean\_Frechet

To clustering με το αρχικό dataset:

Algorithm: Range Search LSH\_Frechet

CLUSTER-0 {centroid,bbby, aapl, abc, abt, abx, adm, adsk, aem, aes, amat, an, ash, atro, axgn, bax, bb, admp, }

CLUSTER-1 {centroid,aphb, }

CLUSTER-2 {centroid,a, a, aet, aig, amd, ati, azo, aa, amgn, amzn, antm, ba, bbb, aaba, adbe, adi, adp, aee, aep, afl, agn, aiv, all, apa, apc, apd, au, avp, avy, axp, bac, bbt, bby, bc, }

Silhouette: [0.0326603,0,0.0412423,0.0246342]

To clustering με το συμπιεσμένο dataset:

Algorithm: Range Search LSH\_Frechet

CLUSTER-0 {centroid,bax, aapl, adm, afl, amat, bc, aem, apc, a, atro, agn, amzn, an, axp, bb, au, adi, adbe, aet, amgn, apd, avp, avy, axgn, bby, aes, abx, aee, all, bac, bbb, bbt, antm, bax, abc, abt, aep, aig, amd, azo, aiv, ati, ba, aaba, adp, adsk, apa, aphb, aa, admp, ash, }

CLUSTER-1 {centroid,adbe, }

CLUSTER-2 {centroid,amzn, }

Silhouette: [0.328115,0,0,0.109372]

και εδώ τα αποτελέσματα είναι σχετικά καλά ,ενώ παραδόξως έχουμε καλύτερο silhouette για το συμπιεσμένο dataset.