

# Εργασία 3

**ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΑΛΓΟΡΙΘΜΙΚΑ ΠΡΟΒΛΗΜΑΤΑ**

**Mantzouranis Gewrgios sdi1700076**

**Mixopoulos Mixalis sdi1700091**

21 / 01 / 2021

## A Ερώτημα.

Το A ερώτημα το έχουμε χωρίσει σε **A1** και **A2** για να καλύψουμε και τα 2 κομμάτια του ερωτήματος, τόσο ανά χρονοσειρά όσο και για το σύνολο των χρονοσειρών.

Στο **A1** δημιουργούμε ένα μοντέλο και για κάθε χρονοσειρά γίνεται **fit** και **predict**. Επίσης, υπάρχει η μεταβλητή **look\_back\_value**, η οποία μπορεί και αλλάζει κάθε φορά και η αλλαγή της οποίας επηρεάζει τα αποτελέσματα του training.

Σε γενικές γραμμές, αυτό που παρατηρήσαμε είναι ότι το πλήθος των **epochs, batch**, επηρεάζουν σημαντικά τον χρόνο του training και ίσως όχι τόσο το αποτέλεσμα του **predict**. Αυτό που επίσης επηρεάζει τον χρόνο σε συνδυασμό με τα προηγούμενα είναι και η μεταβλητή **look\_back\_value**. Όσο μεγαλύτερη η τιμή της, τόσο πιο πολύ αργή και το fitting. Οι τιμές που δοκιμάσαμε για την τελευταία ξεκινούσαν από **60**, που ήταν και η αρχική τιμή του **tutorial** που μας στείλατε, στη συνέχεια κατεβήκαμε στο **10, 5, 3** και **1**, όπου και το fitting έγινε αρκετά πιο γρήγορο.

Όσον αφορά τα **LSTM layers**, πηραματιστήκαμε με αρκετά νούμερα **units** και αριθμό **στρωμάτων**. Ο μεγαλύτερος αριθμός στρωμάτων που προσθέσαμε ήταν 4 και ο μικρότερος 1. Ο χρόνος επίσης αυξανόταν, όταν τα στρώματα γίνονταν περισσότερα. Για τον αριθμό **μονάδων**, οι περισσότερες που προσθέσαμε ήταν 128 και οι λιγότερες 32 ενώ παίξαμε κυρίως με τους αριθμούς 50 και 64.

Ο μέσος χρόνος που χρειάζεται ένα epoch να γίνει trained είναι περίπου στα **2s** ενώ το loss κυμαίνεται στο **0.002 – 0.0005**

Μερικά από τα παραδείγματα των γραφικών παραστάσεων predicted και real values είναι τα εξής:

Compile με **optimizer: adam**

```
+ Κώδικας + Κείμενο
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.models import load_model
import os.path

# create a model
def createModel(lag):

    model = Sequential()
    # Adding the first LSTM layer and some Dropout regularisation
    model.add(LSTM(units = 50, return_sequences = True, input_shape = (lag, 1)))
    model.add(Dropout(0.2))
    # Adding a second LSTM layer and some Dropout regularisation
    model.add(LSTM(units = 50, return_sequences = True))
    model.add(Dropout(0.2))
    # Adding a third LSTM layer and some Dropout regularisation
    model.add(LSTM(units = 50, return_sequences = True))
    model.add(Dropout(0.2))
    # Adding a fourth LSTM layer and some Dropout regularisation
    model.add(LSTM(units = 50))
    model.add(Dropout(0.2))
    # Adding the output layer
    model.add(Dense(units = 1))

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

    return model

def processData(data: np.array, lag, size):

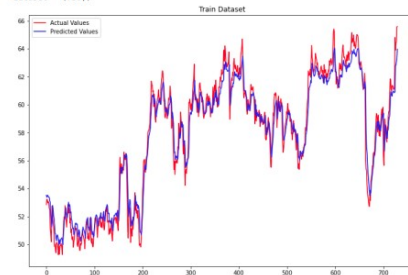
    X,Y = [], []

    for i in range(lag, size):
        X.append(data[i-lag:i, 0])
        Y.append(data[i, 0])

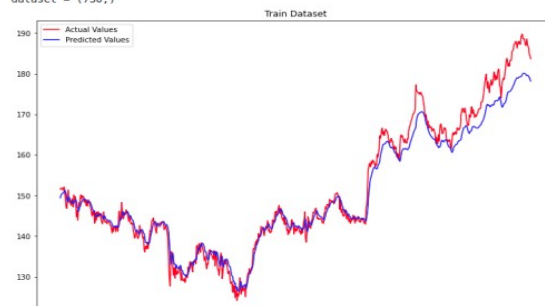
    return np.array(X), np.array(Y)

# load dataset
```

```
91/91 [=====] - 1s 8ms/step - loss: 6.2693e-04
Epoch 96/100
91/91 [=====] - 1s 8ms/step - loss: 5.3672e-04
Epoch 97/100
91/91 [=====] - 1s 8ms/step - loss: 5.5638e-04
Epoch 98/100
91/91 [=====] - 1s 8ms/step - loss: 5.7294e-04
Epoch 99/100
91/91 [=====] - 1s 8ms/step - loss: 5.6068e-04
Epoch 100/100
91/91 [=====] - 1s 8ms/step - loss: 5.7782e-04
test=> (740, 1)
predicted = (730, 1)
dataset = (730,)
```



```
91/91 [=====] - 1s 7ms/step - loss: 4.4503e-04
Epoch 94/100
91/91 [=====] - 1s 7ms/step - loss: 4.7782e-04
Epoch 95/100
91/91 [=====] - 1s 7ms/step - loss: 4.5352e-04
Epoch 96/100
91/91 [=====] - 1s 7ms/step - loss: 5.1369e-04
Epoch 97/100
91/91 [=====] - 1s 7ms/step - loss: 4.8971e-04
Epoch 98/100
91/91 [=====] - 1s 7ms/step - loss: 4.9925e-04
Epoch 99/100
91/91 [=====] - 1s 7ms/step - loss: 5.4594e-04
Epoch 100/100
91/91 [=====] - 1s 7ms/step - loss: 4.7566e-04
test=> (740, 1)
predicted = (730, 1)
dataset = (730,)
```



Untitled0.ipynb

Αρχείο Επεξεργασία Προβολή Εισαγωγή Χρόνος εκτέλεσης (runtime) Εργασία Βοήθεια Όλες οι αλλαγές αποθηκεύτηκαν

Σχόλιο Κουνοποίηση

RAM Δίσκος

Επεξεργασία

Αρχείο

models

my\_model.h5

sample\_data

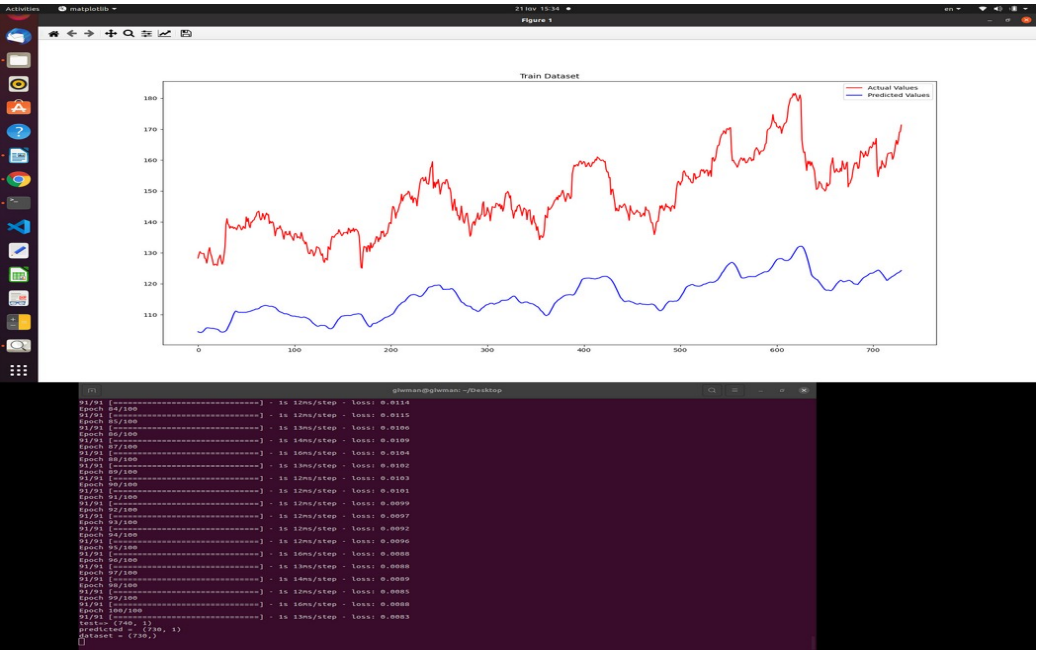
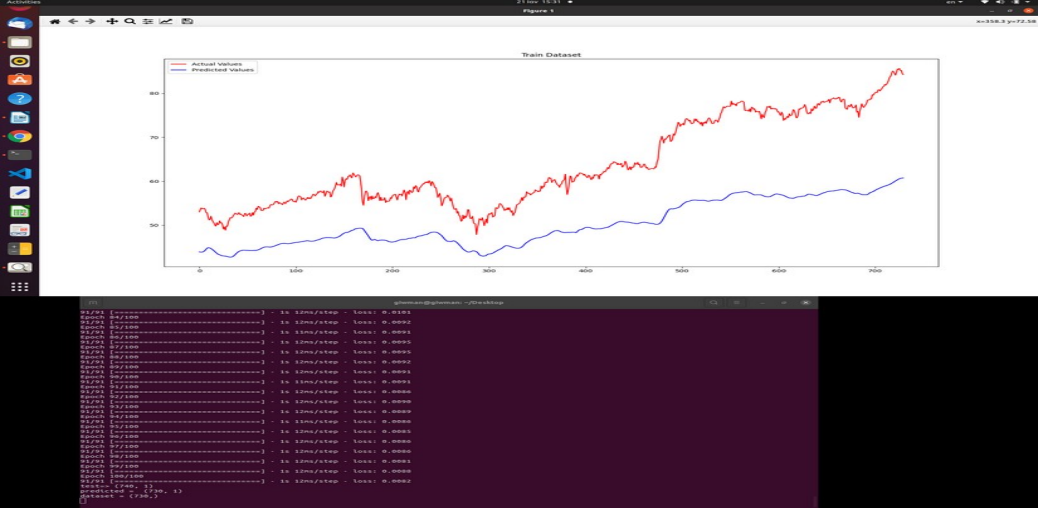
nasdaq2007\_17.csv

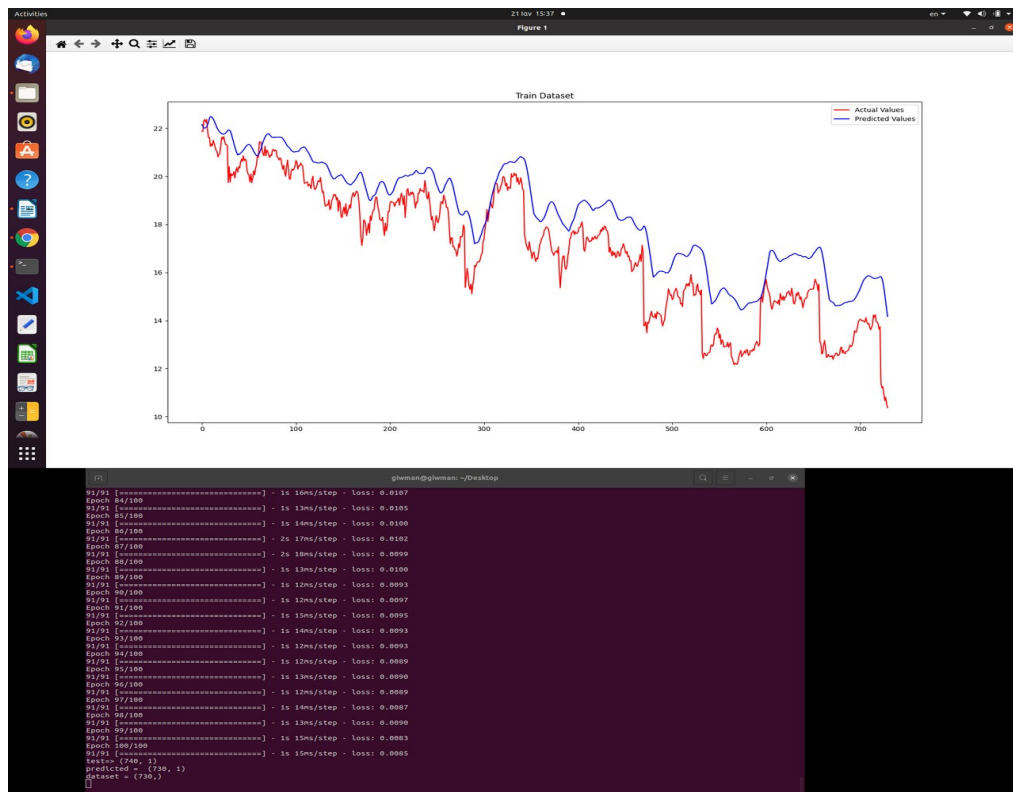
+ Κώδικας + Κείμενο

```
91/91 [=====] - 1s 7ms/step - loss: 6.0371e-04
Epoch 94/100
91/91 [=====] - 1s 7ms/step - loss: 6.5452e-04
Epoch 95/100
91/91 [=====] - 1s 7ms/step - loss: 6.4323e-04
Epoch 96/100
91/91 [=====] - 1s 7ms/step - loss: 6.1473e-04
Epoch 97/100
91/91 [=====] - 1s 7ms/step - loss: 6.0311e-04
Epoch 98/100
91/91 [=====] - 1s 7ms/step - loss: 5.6914e-04
Epoch 99/100
91/91 [=====] - 1s 7ms/step - loss: 6.3455e-04
Epoch 100/100
91/91 [=====] - 1s 7ms/step - loss: 6.0849e-04
test=> (740, 1)
predicted = (730, 1)
dataset = (730,)
```

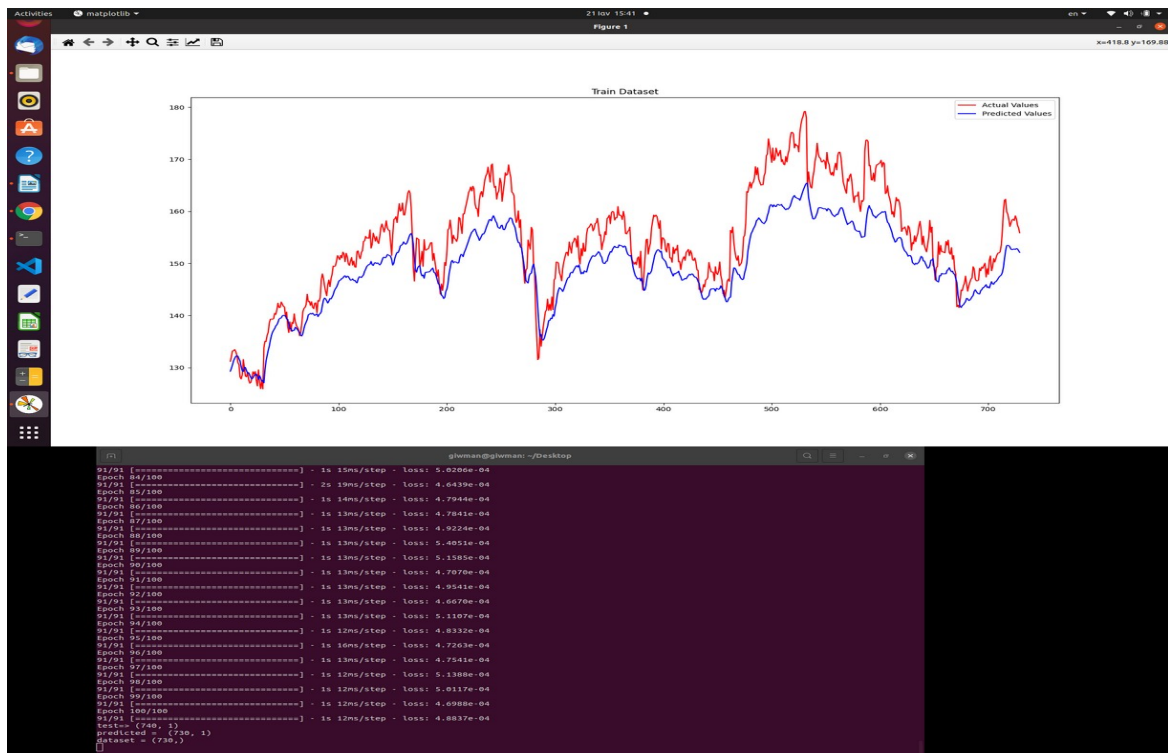
Train Dataset

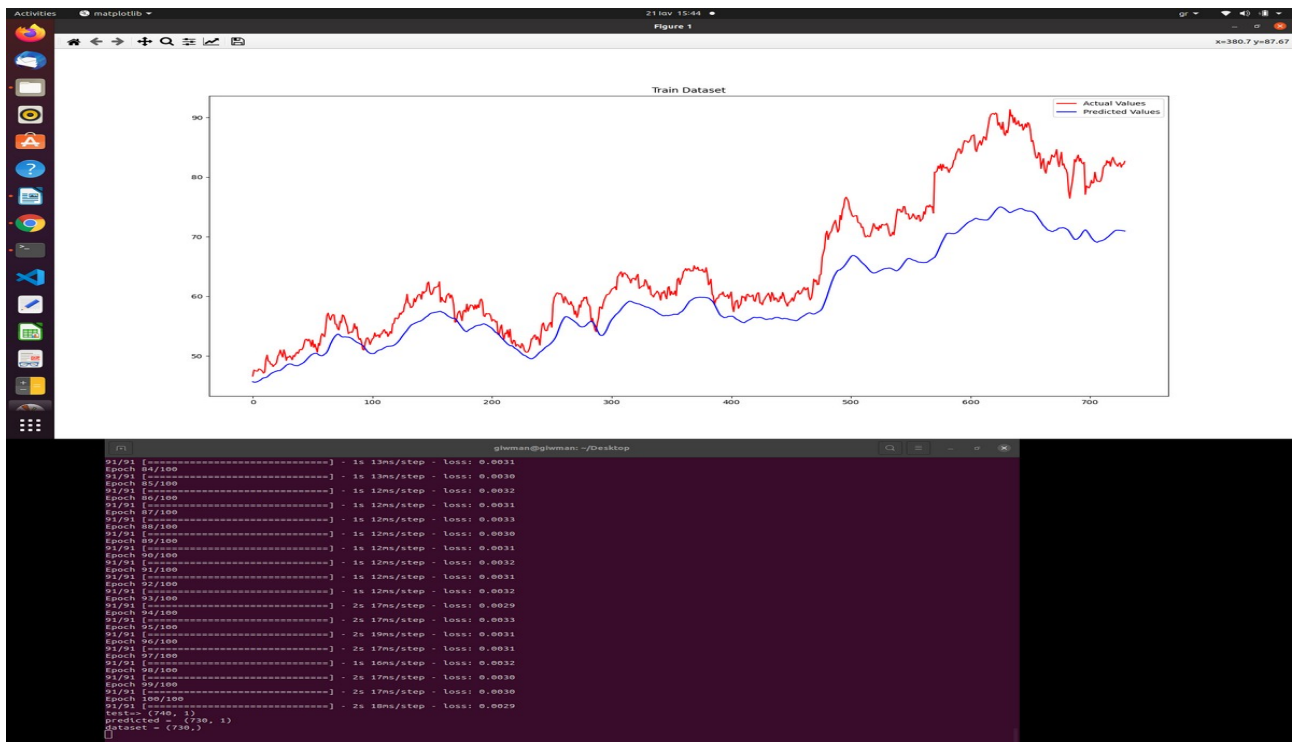
## Compile με **adadelta**:





Compile  $\mu$  Rmsprop:





Από τα παραπάνω παραδείγματα και από δικούς μας πειραματισμούς, παρατηρήσαμε ότι οι όποιες σημαντικές διαφοροποιήσεις μεταξύ των αποτελεσμάτων, έχουν να κάνουν σε μεγάλο βαθμό και από τις **optimization functions**. Αν και δεν τις δοκιμάσαμε όλες, διαπιστώσαμε ότι τα “καλύτερα” αποτελέσματα τα πετύχαμε με την **adam**.

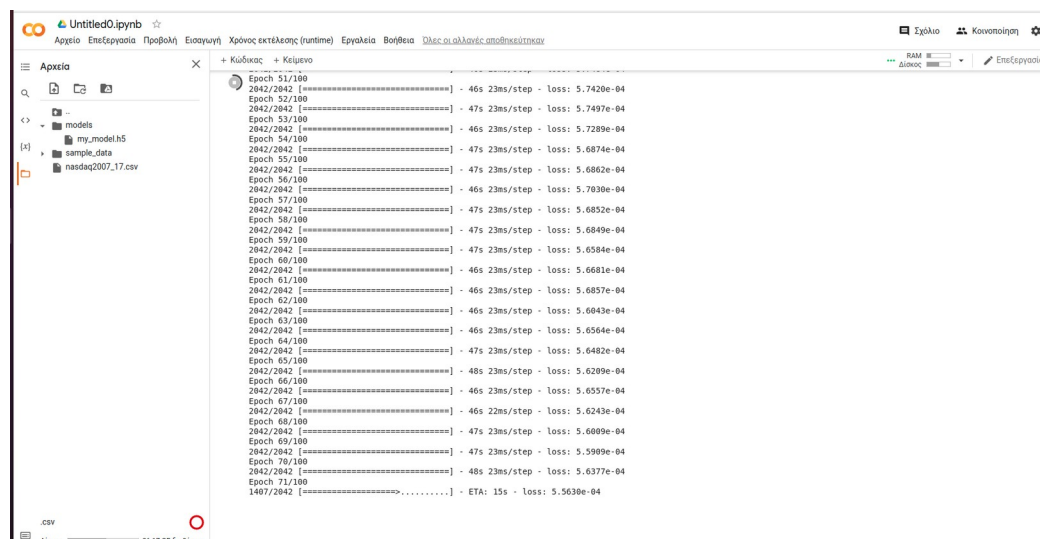
Για το **A2** και το **training στο σύνολο των χρονοσειρών**, αρχικά δημιουργούμε το **training set** κρατώντας για κάθε χρονοσειρά το 80% της. Κάνουμε scale και στη συνέχεια ενώνουμε όλες τις χρονοσειρές σε ένα μεγάλο set, το οποίο θα αποτελείται από το 80% κάθε χρονοσειράς \* 360 που είναι όλες οι χρονοσειρές. Μόλις γίνει αυτό, ξεκινάμε το fit με ένα μεγαλύτερο batch size από πριν για να καταφέρουμε να μειώσουμε τον χρόνο, ο οποίος αυτή τη φορά ανεβαίνει σημαντικά, εξαιτίας του μεγάλου όγκου δεδομένων. Αυτή τη φορά κάθε **epoch**, χρειάζεται περίπου 45s για να ολοκληρωθεί με βάση το τελευταίο μοντέλο που έχουμε δημιουργήσει. Για το prediction, επιλέγουμε n τυχαίες χρονοσειρές με τις οποίες γίνεται το predict και στο τέλος εμφανίζουμε n plots.

Όσον αφορά τις υπερπαραμέτρους που αναφέραμε και στο **A1**, οι τιμές δεν αλλάζουν σημαντικά. Συνεχίσαμε να πειραματιζόμαστε πάνω στις ίδιες τιμές. Αυτό που άλλαξε σίγουρα αυτή τη φορά είναι ο αριθμός των **epoch και batch size** και αυτό γιατί αυξήθηκε και ο χρόνος του training.

Ακόμα παρατηρήσαμε ότι έχοντας 3 layers LSTM και παίζοντας με τους αριθμούς 64, 50 και 32 καταφέραμε να έχουμε αρκετά καλό loss κοντά στο 0.0005, φυσικά σε συνδυασμό με τους αριθμούς των υπόλοιπων υπερπαραμέτρων.

Τέλος, να σημειώσουμε ότι για αυτό το μοντέλο, το fit μας πήρε πάνω από μια ώρα.

Κάποια από τα αποτελέσματα είναι τα εξής:

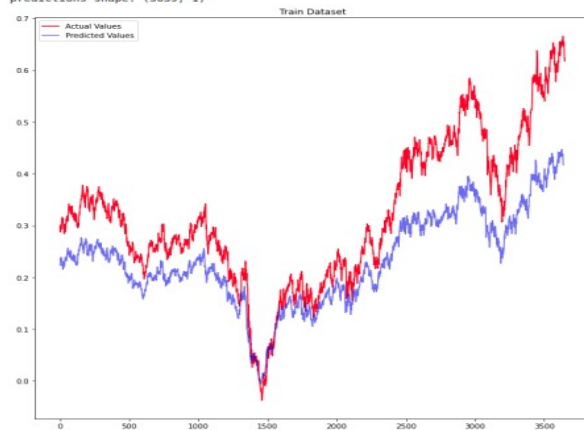
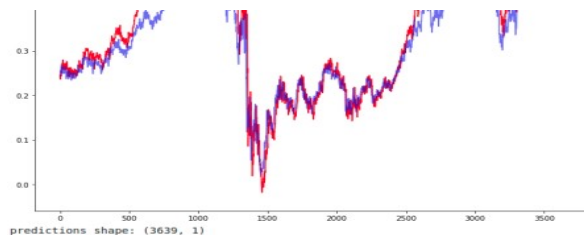
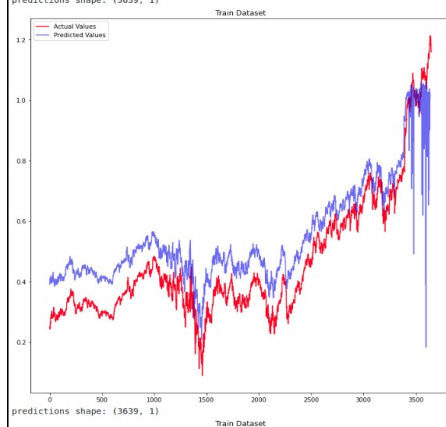


```
Epoch: 51/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.7420e-04
Epoch: 52/100
2042/2042 [=====] - 47s 23ms/step - loss: 5.7497e-04
Epoch: 53/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.7289e-04
Epoch: 54/100
2042/2042 [=====] - 47s 23ms/step - loss: 5.6874e-04
Epoch: 55/100
2042/2042 [=====] - 47s 23ms/step - loss: 5.6862e-04
Epoch: 56/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.7030e-04
Epoch: 57/100
2042/2042 [=====] - 47s 23ms/step - loss: 5.6852e-04
Epoch: 58/100
2042/2042 [=====] - 47s 23ms/step - loss: 5.6849e-04
Epoch: 59/100
2042/2042 [=====] - 47s 23ms/step - loss: 5.6584e-04
Epoch: 60/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.6681e-04
Epoch: 61/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.6857e-04
Epoch: 62/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.6843e-04
Epoch: 63/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.6564e-04
Epoch: 64/100
2042/2042 [=====] - 47s 23ms/step - loss: 5.6482e-04
Epoch: 65/100
2042/2042 [=====] - 48s 23ms/step - loss: 5.6209e-04
Epoch: 66/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.6557e-04
Epoch: 67/100
2042/2042 [=====] - 46s 22ms/step - loss: 5.6243e-04
Epoch: 68/100
2042/2042 [=====] - 47s 23ms/step - loss: 5.6009e-04
Epoch: 69/100
2042/2042 [=====] - 47s 23ms/step - loss: 5.5909e-04
Epoch: 70/100
2042/2042 [=====] - 48s 23ms/step - loss: 5.6377e-04
Epoch: 71/100
1487/2042 [=====] - ETA: 15s - loss: 5.5630e-04
```

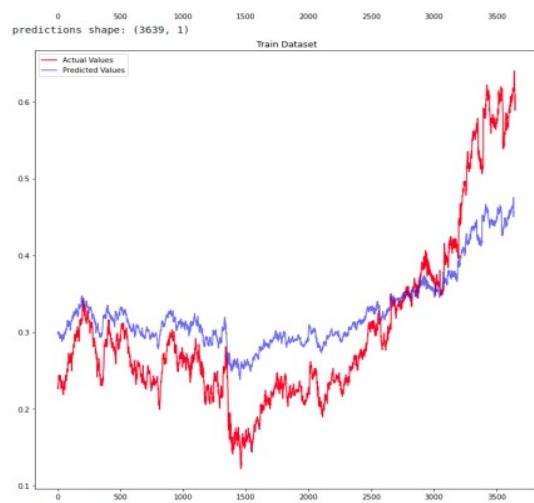
```

2042/2042 [=====] - 40s 23ms/step - loss: 5.4230e-04
Epoch 92/100
2042/2042 [=====] - 47s 23ms/step - loss: 5.4150e-04
Epoch 93/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.4597e-04
Epoch 94/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.4287e-04
Epoch 95/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.4345e-04
Epoch 96/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.4401e-04
Epoch 97/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.4400e-04
Epoch 98/100
2042/2042 [=====] - 49s 24ms/step - loss: 5.4231e-04
Epoch 99/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.4216e-04
Epoch 100/100
2042/2042 [=====] - 46s 23ms/step - loss: 5.3342e-04
predictions shape: (3639, 1)

```







## Εργασία 3 : Β

Section 1 : In this section we are getting the data for the csv file .  
We are creating the stocks set .

Section 2 : We calculating the train set and the test set ,so we are separating the sets.

Section 3 : Then we scale these sets using the MinMaxScaler.

Section 4 : Then b/c in the exercise we need to use a set of TimeSeries to feed to the NN . So we concatenate a set of time series in order to train the NN.

Section 5 : Then we split the concatenated-test set and the scaled-test-series into subsequences.

**The things until now were standard now we are getting into  
The experimenting stage .**

Section 6 : We create an Autoencoder in Keras : using the train – subsequences shape . we experimentef with a lot of hyperparameters.

Here are some examples.

B.pybnb

☆

File

Edit

View

Insert

Runtime

Tools

Help

All changes saved

Comment

Share

⚙️

M

Table of contents

✕

+ Code

+ Text

✓ RAM 8

Disk

Editing

^

Step 1 : getting and saving the data appropriately.

Step 2: Calculating Train and Test sets.

Step 3 : Let's scale our Data.

Step 4 : Concatenate the chosen set of training stocks.

Step 5 : Split data into subSequences .

Step 6 : Let's create our Autoencoder in Keras and Train our Model -- Experimenting Stage --

Section

✓ [99] model.compile(loss='mae', optimizer='adam')

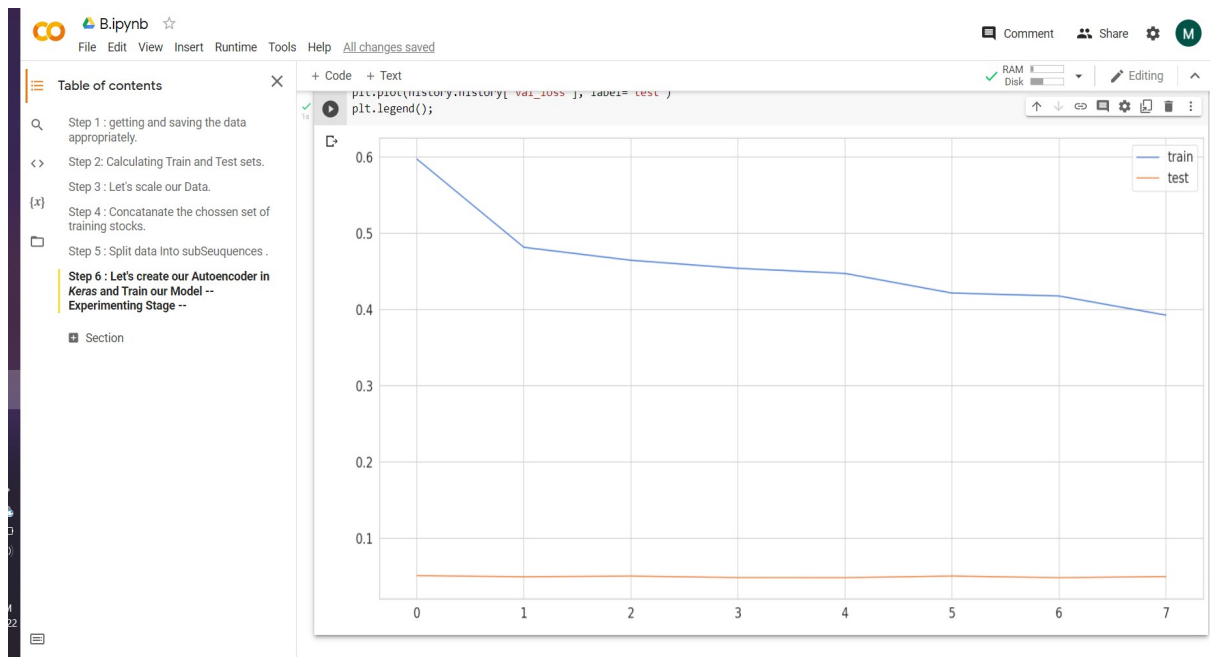
✓ ▶

history = model.fit(  
    X\_train, y\_train,  
    epochs=8,  
    batch\_size=128,  
    validation\_split=0.1,  
    shuffle=False  
)

Epoch 1/8  
1027/1027 [=====] - 62s 57ms/step - loss: 0.5973 - val\_loss: 0.0511  
Epoch 2/8  
1027/1027 [=====] - 58s 56ms/step - loss: 0.4815 - val\_loss: 0.0496  
Epoch 3/8  
1027/1027 [=====] - 58s 56ms/step - loss: 0.4646 - val\_loss: 0.0506  
Epoch 4/8  
1027/1027 [=====] - 58s 57ms/step - loss: 0.4540 - val\_loss: 0.0485  
Epoch 5/8  
1027/1027 [=====] - 58s 57ms/step - loss: 0.4473 - val\_loss: 0.0484  
Epoch 6/8  
1027/1027 [=====] - 58s 57ms/step - loss: 0.4216 - val\_loss: 0.0506  
Epoch 7/8  
1027/1027 [=====] - 58s 57ms/step - loss: 0.4177 - val\_loss: 0.0483  
Epoch 8/8  
1027/1027 [=====] - 58s 57ms/step - loss: 0.3926 - val\_loss: 0.0499

8m 25s

completed at 2:14 PM



B.ipy nb

File Edit View Insert Runtime Tools Help All changes saved

Files

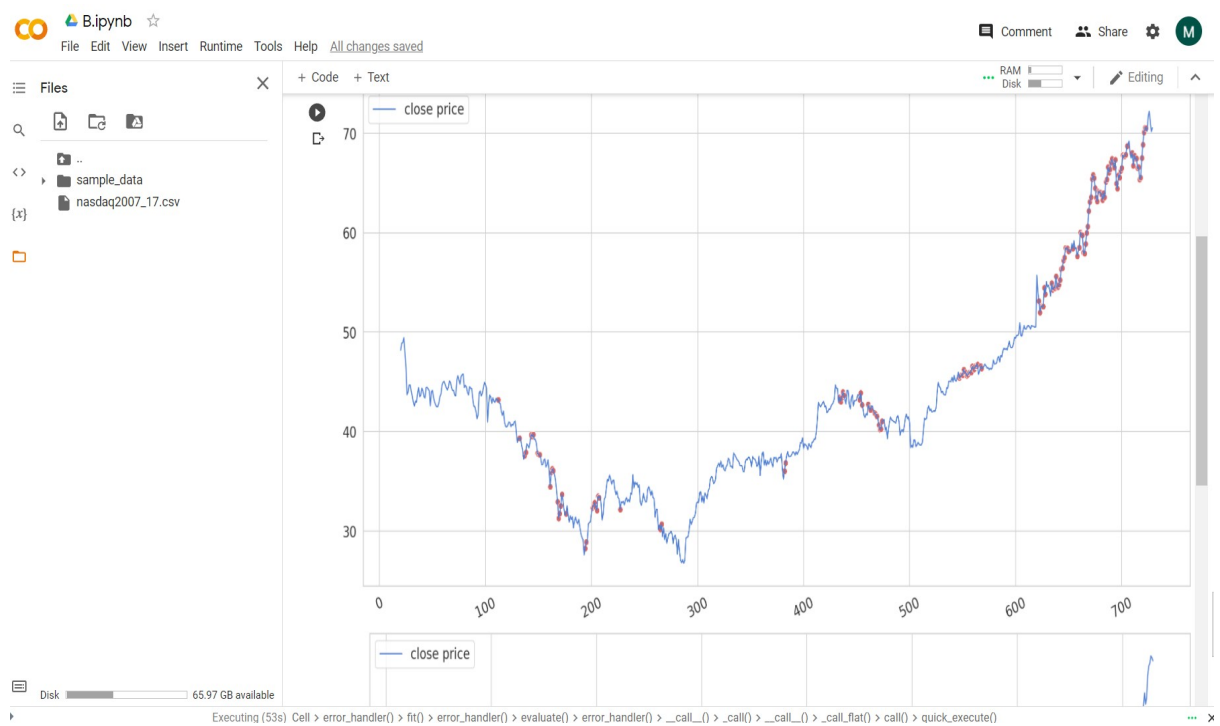
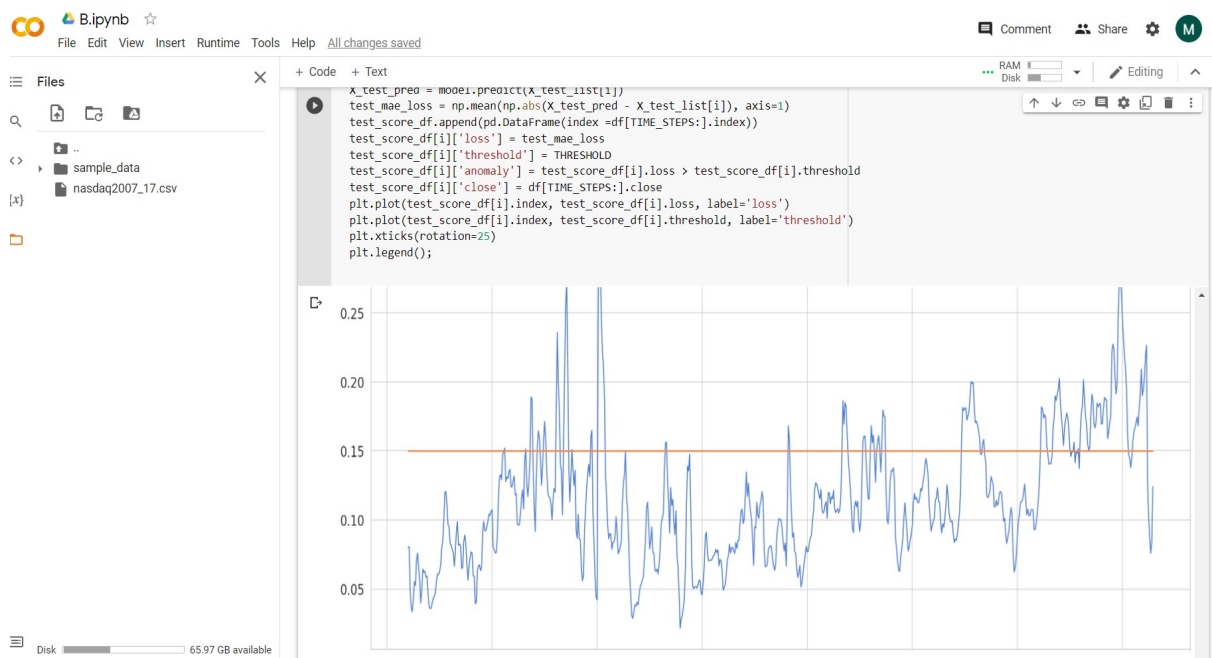
- sample\_data
- nasdaq2007\_17.csv

```
model = keras.Sequential()
model.add(keras.layers.LSTM(
    units=64,
    input_shape=(X_train.shape[1], X_train.shape[2])
))
model.add(keras.layers.Dropout(rate=0.2))
model.add(keras.layers.RepeatVector(n=X_train.shape[1]))
model.add(keras.layers.LSTM(units=64, return_sequences=True))
model.add(keras.layers.Dropout(rate=0.2))
model.add(
    keras.layers.TimeDistributed(
        keras.layers.Dense(units=X_train.shape[2])
    )
)
model.compile(loss='mae', optimizer='adam')
```

```
[ ] history = model.fit(
    x_train, y_train,
    epochs=5,
    batch_size=128,
    validation_split=0.1,
    shuffle=False
)
```

Epoch 1/5  
822/822 [=====] - 52s 59ms/step - loss: 0.7265 - val\_loss: 0.0418  
Epoch 2/5  
822/822 [=====] - 47s 57ms/step - loss: 0.5885 - val\_loss: 0.0401  
Epoch 3/5  
822/822 [=====] - 47s 57ms/step - loss: 0.5719 - val\_loss: 0.0388  
Epoch 4/5  
822/822 [=====] - 47s 57ms/step - loss: 0.5542 - val\_loss: 0.0415  
Epoch 5/5  
822/822 [=====] - 47s 57ms/step - loss: 0.5410 - val\_loss: 0.0377

Disk 65.97 GB available





Tools Help

Comment Share M

+ Code + Text

RAM Disk Editing

```
[16] model = keras.Sequential()
model.add(keras.layers.LSTM(
    units=64,
    input_shape=(X_train.shape[1], X_train.shape[2])
))
model.add(keras.layers.Dropout(rate=0.6))
model.add(keras.layers.RepeatVector(n=X_train.shape[1]))
model.add(keras.layers.Dropout(rate=0.6))
model.add(keras.layers.LSTM(units=64, return_sequences=True))
model.add(keras.layers.Dropout(rate=0.6))
model.add(
    keras.layers.TimeDistributed(
        keras.layers.Dense(units=X_train.shape[2])
    )
)
model.compile(loss='mae', optimizer='adam')
```

5m

▶ history = model.fit(
 X\_train, y\_train,
 epochs=8,
 batch\_size=128,
 validation\_split=0.5,
 shuffle=False
)

📄

Epoch 1/8  
571/571 [=====] - 39s 69ms/step - loss: 0.1047 - val\_loss: 0.0958  
Epoch 2/8  
571/571 [=====] - 40s 70ms/step - loss: 0.0985 - val\_loss: 0.1408  
Epoch 3/8  
571/571 [=====] - 40s 70ms/step - loss: 0.1088 - val\_loss: 0.1031  
Epoch 4/8  
571/571 [=====] - 40s 70ms/step - loss: 0.1002 - val\_loss: 0.0981  
Epoch 5/8  
571/571 [=====] - 41s 73ms/step - loss: 0.0970 - val\_loss: 0.1227  
Epoch 6/8

silable

6s completed at 12:14 PM

+ Code + Text

RAM Disk Editing

Our Autoencoder should take a sequence as input and outputs a sequence of the same shape. Here's how to build such a simple model in Keras:

```
model = keras.Sequential()
model.add(keras.layers.LSTM(
    units=64,
    input_shape=(X_train.shape[1], X_train.shape[2])
))
model.add(keras.layers.Dropout(rate=0.4))
model.add(keras.layers.RepeatVector(n=X_train.shape[1]))
model.add(keras.layers.Dropout(rate=0.4))
model.add(keras.layers.LSTM(units=64, return_sequences=True))
model.add(keras.layers.Dropout(rate=0.4))
model.add(
    keras.layers.TimeDistributed(
        keras.layers.Dense(units=X_train.shape[2])
    )
)
model.compile(loss='mae', optimizer='adam')
```

8m

▶ history = model.fit(
 X\_train, y\_train,
 epochs=12,
 batch\_size=128,
 validation\_split=0.5,
 shuffle=False
)

📄

Epoch 1/12  
571/571 [=====] - 45s 79ms/step - loss: 0.1269 - val\_loss: 1.0011  
Epoch 2/12  
571/571 [=====] - 46s 81ms/step - loss: 0.0967 - val\_loss: 1.0437  
Epoch 3/12

silable

8m 59s completed at 12:26 PM

+ Code + Text

RAM Disk

Editing

0s

history = model.fit(  
X\_train, y\_train,  
epochs=12,  
batch\_size=128,  
validation\_split=0.5,  
shuffle=False  
)

8m

Epoch 1/12  
571/571 [=====] - 45s 79ms/step - loss: 0.1269 - val\_loss: 1.0011  
Epoch 2/12  
571/571 [=====] - 46s 81ms/step - loss: 0.0967 - val\_loss: 1.0437  
Epoch 3/12  
571/571 [=====] - 45s 79ms/step - loss: 0.1059 - val\_loss: 1.0239  
Epoch 4/12  
571/571 [=====] - 45s 78ms/step - loss: 0.1036 - val\_loss: 1.0218  
Epoch 5/12  
571/571 [=====] - 45s 79ms/step - loss: 0.1045 - val\_loss: 1.0147  
Epoch 6/12  
571/571 [=====] - 45s 79ms/step - loss: 0.1018 - val\_loss: 1.0079  
Epoch 7/12  
571/571 [=====] - 45s 79ms/step - loss: 0.1026 - val\_loss: 0.9859  
Epoch 8/12  
571/571 [=====] - 44s 78ms/step - loss: 0.0965 - val\_loss: 0.9957  
Epoch 9/12  
571/571 [=====] - 45s 78ms/step - loss: 0.0960 - val\_loss: 1.0030  
Epoch 10/12  
571/571 [=====] - 44s 78ms/step - loss: 0.0946 - val\_loss: 1.0393  
Epoch 11/12  
571/571 [=====] - 45s 78ms/step - loss: 0.0996 - val\_loss: 0.9931  
Epoch 12/12  
571/571 [=====] - 45s 78ms/step - loss: 0.0930 - val\_loss: 1.0177

able

✓ 2s  
[19] plt.plot(history.history['loss'], label='train')  
plt.plot(history.history['val\_loss'], label='test')

✓ 8m 59s completed at 12:26 PM

+ Code + Text

RAM Disk

Editing

0s

Our Autoencoder should take a sequence as input and outputs a sequence of the same shape. Here's how to build such a simple model in Keras:  
  
model = keras.Sequential()  
model.add(keras.layers.LSTM(  
units=64,  
input\_shape=(X\_train.shape[1], X\_train.shape[2])  
))  
model.add(keras.layers.Dropout(rate=0.5))  
model.add(keras.layers.RepeatVector(n=X\_train.shape[1]))  
model.add(keras.layers.Dropout(rate=0.5))  
model.add(keras.layers.LSTM(units=64, return\_sequences=True))  
model.add(keras.layers.Dropout(rate=0.5))  
model.add(  
keras.layers.TimeDistributed(  
keras.layers.Dense(units=X\_train.shape[2])  
)  
)  
model.compile(loss='mae', optimizer='adam')

7m

history = model.fit(  
X\_train, y\_train,  
epochs=8,  
batch\_size=128,  
validation\_split=0.4,  
shuffle=False  
)

able

Epoch 1/8  
685/685 [=====] - 55s 75ms/step - loss: 0.2438 - val\_loss: 1.1099  
Epoch 2/8  
685/685 [=====] - 51s 74ms/step - loss: 0.1448 - val\_loss: 1.0203  
Epoch 3/8

✓ 7m 25s completed at 12:37 PM



Section 7 : we re finding and plotting tth anomalies for each time series.