

# Τεχνητή Νοημοσύνη

Εργασία: Πρόβλεψη μετοχών με αλγορίθμους  
μηχανικής μάθησης

10/12/2023

Φοιτητές:

Γεροβασίλης Κωνσταντίνος, ΜΠΠΛ21008

Κυπραίος Χαρίτων, ΜΠΠΛ21042

## Περιεχόμενα

Γενικές Πληροφορίες για την Τεχνητή Νοημοσύνη και τη Μηχανική Μάθηση .....	2
Τεχνικά Χαρακτηριστικά και Εργαλεία Εκπόνησης της Εργασίας .....	3
Τρόπος Εκτέλεσης του κώδικα .....	4
Κώδικας Αλγόριθμου Γραμμικής Παλινδρόμησης .....	10
Κώδικας Αλγόριθμου Δέντρων Απόφασης.....	15
Κώδικας Αλγόριθμου Νευρωνικών Δικτύων .....	23
Παραμετροποιήσεις αλγορίθμων με την προσθήκη δεικτών.....	31
Κώδικας αλγορίθμου γραμμικής παλινδρόμησης με προσθήκη των δεικτών MACD, Bollinger Bands και RSI .....	31
Κώδικας αλγορίθμου δέντρων απόφασης με προσθήκη των δεικτών MACD, Bollinger Bands και RSI. ....	39
Πηγές .....	48

Η συγκεκριμένη εργασία έχει ως στόχο την πρόβλεψη μετοχών με την χρήση αλγορίθμων μηχανικής μάθησης. Στην συγκεκριμένη εργασία χρησιμοποιούνται 3 αλγόριθμοι μηχανικής μάθησης:

1. Ο αλγόριθμος Γραμμικής Παλινδρόμησης (Linear Progression).
2. Ο αλγόριθμος Δέντρων Αποφάσεων (Decision Trees).
3. Ο αλγόριθμος Νευρωνικών Δικτύων (Neural Networks).

## Γενικές Πληροφορίες για την Τεχνητή Νοημοσύνη και τη Μηχανική Μάθηση

Η Τεχνητή Νοημοσύνη (AI) αναφέρεται στη γενική ικανότητα των υπολογιστών να μιμούνται την ανθρώπινη σκέψη και να εκτελούν εργασίες σε πραγματικά περιβάλλοντα. Περιλαμβάνει ευρεία γκάμα λειτουργιών όπως η κατανόηση της γλώσσας, η αναγνώριση μοτίβων, η επίλυση προβλημάτων και η μάθηση. Η AI χρησιμοποιεί διάφορα εργαλεία και τεχνικές, όπως η βαθιά μάθηση, τα νευρωνικά δίκτυα, η υπολογιστική όραση και η επεξεργασία φυσικής γλώσσας.

Η Μηχανική Μάθηση είναι ένα υποσύνολο της AI, που εστιάζει στη χρήση δεδομένων και αλγορίθμων για να επιτρέψει στους υπολογιστές να μαθαίνουν από την εμπειρία και να βελτιώνουν την απόδοσή τους με την πάροδο του χρόνου. Σε αντίθεση με τον παραδοσιακό προγραμματισμό, η ML επιτρέπει σε ένα σύστημα να μαθαίνει και να προσαρμόζεται σε νέα δεδομένα ανεξάρτητα. Οι αλγόριθμοι ML εκπαιδεύονται χρησιμοποιώντας μεγάλα σύνολα δεδομένων και βελτιώνουν την ακρίβεια και την αποτελεσματικότητά τους καθώς επεξεργάζονται περισσότερα δεδομένα.

Η βαθιά μάθηση, η οποία είναι μια πιο προηγμένη μέθοδος μηχανικής μάθησης, χρησιμοποιεί μεγάλα νευρωνικά δίκτυα (τα οποία λειτουργούν όπως ο εγκέφαλος ενός ανθρώπου) για την ανάλυση περίπλοκων μοτίβων στα δεδομένα. Αυτά τα μοντέλα μπορούν να εκτελούν εργασίες όπως αναγνώριση εικόνων και ομιλίας με υψηλό βαθμό ακρίβειας.

Συνολικά, ενώ η Τεχνητή Νοημοσύνη περιλαμβάνει την ευρύτερη έννοια των μηχανών που είναι ικανές να εκτελούν εργασίες με τρόπο που θεωρούμε "έξυπνο", η

μηχανική μάθηση είναι μια τρέχουσα εφαρμογή της Τεχνητής Νοημοσύνης βασισμένη στην ιδέα ότι πρέπει να μπορούμε να δίνουμε στις μηχανές πρόσβαση σε δεδομένα και να τους επιτρέπουμε να μαθαίνουν μόνες τους.

## Τεχνικά Χαρακτηριστικά και Εργαλεία Εκπόνησης της Εργασίας

Τα τελευταία χρόνια προκειμένου να επιλυθούν διάφορα προβλήματα που απαιτούνταν η χρήση της Τεχνητής Νοημοσύνης και της Μηχανικής Μάθησης, χρησιμοποιήθηκαν ή αναπτύχθηκαν διάφορες γλώσσες προγραμματισμού που εξυπηρετούσαν αυτόν τον σκοπό. Εκτός των γλωσσών προγραμματισμού, σημαντικό ρόλο παίζουν και οι βιβλιοθήκες που χρησιμοποιήθηκαν ή αναπτύχθηκαν προκειμένου να συνεισφέρουν σε αυτόν το σκοπό.

Οι κυριότερες γλώσσες προγραμματισμού που χρησιμοποιούνται στον τομέα της Μηχανικής Μάθησης και της Τεχνητής Νοημοσύνης είναι:

1. Python: Είναι η πιο δημοφιλής γλώσσα λόγω της ευκολίας μάθησης, της ευέλικτης σύνταξης, και της μεγάλης ποικιλίας βιβλιοθηκών και εργαλείων διαθέσιμων για την ανάπτυξη εφαρμογών AI. Επίσης, διαθέτει εξαιρετικά εργαλεία οπτικοποίησης δεδομένων, που είναι ζωτικής σημασίας στην ανάπτυξη AI.
2. Java: Είναι μια δημοφιλής γενικής χρήσης γλώσσα με μεγάλη κοινότητα προγραμματιστών. Είναι στατικά τυπική, πράγμα που σημαίνει ότι μπορείτε να εντοπίζετε λάθη νωρίς και να τρέχετε προγράμματα γρηγορότερα. Ωστόσο, η Java μπορεί να είναι πολύ λεπτομερής και να έχει απότομη καμπύλη μάθησης.
3. C++: Είναι μια δημοφιλής γλώσσα για εφαρμογές με κρίσιμες απαιτήσεις απόδοσης και διαχείρισης μνήμης. Η ταχύτητα και η αποδοτικότητά της την καθιστούν ιδανική για τη χρήση στη μηχανική μάθηση.
4. JavaScript: Είναι μια δημοφιλής γλώσσα για την ανάπτυξη ιστοσελίδων και είναι γνωστή για τη χρήση της σε βιβλιοθήκες μηχανικής μάθησης όπως η TensorFlow.js.
5. R: Είναι μια δημοφιλής στατιστική γλώσσα προγραμματισμού μεταξύ των επιστημόνων δεδομένων. Είναι εξαιρετική για AI με ισχυρές ανάγκες επεξεργασίας δεδομένων. Ενσωματώνει καλά με άλλες γλώσσες και διαθέτει πολλά διαθέσιμα πακέτα.
6. Julia: Σχεδιάστηκε για υψηλής απόδοσης αριθμητικό υπολογισμό και έχει σταθερή υποστήριξη για μηχανική μάθηση. Είναι μια νεότερη γλώσσα και επομένως δεν έχει μεγάλη κοινότητα υποστήριξης.
7. Lisp: Χρησιμοποιείται για την AI εδώ και πολλά χρόνια. Είναι γνωστή για την ευελιξία της και τη συμβολική, λογική προσέγγιση.

8. Prolog: Είναι μια δηλωτική γλώσσα προγραμματισμού που είναι κατάλληλη για την ανάπτυξη AI. Χρησιμοποιείται κυρίως για λογικό προγραμματισμό — τη βάση της ανάπτυξης AI.
9. Scala: Είναι μια γενικού σκοπού γλώσσα με πολλά χαρακτηριστικά κατάλληλα για την ανάπτυξη AI. Ενσωματώνεται καλά με τη Java και έχει μεγάλη κοινότητα προγραμματιστών.

Στην παρούσα εργασία, λόγω μεγαλύτερης εξοικείωσης με την Python και τις βιβλιοθήκες αυτής για προβλήματα σχετικά με Μηχανική Μάθηση, χρησιμοποιήθηκε η συγκεκριμένη γλώσσα. Η έκδοση της Python που χρησιμοποιήθηκε είναι η 3.10.13 καθώς υπάρχει καλύτερη αλληλεπίδραση με διάφορες βιβλιοθήκες και frameworks που χρειάζονταν για την ολοκλήρωση των αλγορίθμων μηχανικής μάθησης που αναπτύχθηκαν.

Για την ανάπτυξη και εκτέλεση του κώδικα, καθώς και για τη δημιουργία των αρχείων του έργου, χρησιμοποιήθηκε το Jupyter Notebook, ένα εργαλείο που προσφέρει ένα διαδραστικό περιβάλλον για την εκτέλεση κώδικα, οπτικοποίηση δεδομένων και παρουσίαση αποτελεσμάτων. Η εγκατάσταση των βιβλιοθηκών της Python και η διαχείριση των περιβαλλόντων ανάπτυξης πραγματοποιήθηκε μέσω του Anaconda, το οποίο δίνει τη δυνατότητα διαχείρισης πακέτων και περιβαλλόντων, και του Conda prompt, που επιτρέπει την αποδοτική διαχείριση και εγκατάσταση διαφορετικών εκδόσεων λογισμικού. Η συνδυασμένη χρήση του Jupyter Notebook και του Anaconda εξασφαλίζει μια ομαλή εμπειρία ανάπτυξης, *καθιστώντας τη διαδικασία ανάλυσης δεδομένων και εκπαίδευσης μοντέλων πιο αποτελεσματική.*

Ανάμεσα στις βιβλιοθήκες που χρησιμοποιήθηκαν είναι η *yfinance*, η οποία μας έδωσε τη δυνατότητα να λάβουμε δεδομένα χρηματιστηρίου. Η *Pandas* και η *Numpy* για την επεξεργασία και την ανάλυση των δεδομένων. Επιπλέον, χρησιμοποιήθηκαν η *Matplotlib* και η *Seaborn* για τη δημιουργία γραφημάτων και την καλύτερη οπτικοποίηση των δεδομένων. Τέλος η *Tensorflow* με το *Keras API* και η *Scikit-Learn*, χρησιμοποιήθηκαν για την εκπαίδευση μοντέλων μηχανικής μάθησης.

## Τρόπος Εκτέλεσης του κώδικα

Προκειμένου να εκτελεστεί ο κώδικας από τα jupyter files που έχουν δημιουργηθεί υπάρχουν 2 τρόποι. Ο πρώτος τρόπος αφορά σε περίπτωση που θέλουμε να δημιουργήσουμε ένα περιβάλλον τοπικά ενώ ο δεύτερος αξιοποιεί τους πόρους που υπάρχουν από την Google προκειμένου να εκτελέσουμε τα jupyter files. Για εκπαιδευτικούς λόγους προτείνεται ο δεύτερος τρόπος.

### 1. Μέσω της λήψης του Anaconda και δημιουργία ενός conda environment

Για τον τρόπο αυτό η διαδικασία που θα πρέπει να πραγματοποιηθεί είναι η εξής:

- Λήψη και εγκατάσταση του Anaconda. Από την επίσημη ιστοσελίδα του Anaconda μπορεί να γίνει η λήψη του στην τελευταία έκδοση ανάλογα το λειτουργικό σύστημα του χρήστη <https://www.anaconda.com/>
- Άνοιγμα του conda prompt που δημιουργήθηκε κατά την εγκατάσταση του Anaconda.
- Unzip τον φάκελο του project και μετάβαση μέσω του Conda prompt στον φάκελο όπου υπάρχουν τα .ipynb αρχεία και το requirements.txt αρχείο. Οι εντολές στον conda prompt είναι ακριβώς οι ίδιες που υπάρχουν και στα υπόλοιπα Command lines.
- Εκτέλεση της παρακάτω εντολής από το conda prompt: **conda create --prefix ./ai\_project\_env --file requirements.txt**  
Το βήμα αυτό θα πάρει λίγη ώρα και θα εμφανιστεί κάποια στιγμή το ακόλουθο  
στον conda prompt:

Εκεί πατάμε γ και έπειτα από λίγη ώρα θα ολοκληρωθεί και θα έχουμε την  
εξής εικόνα:

- Εκτελούμε την παρακάτω εντολή καθώς θα μας χρειαστεί στη συνέχεια για κάποιες βιβλιοθήκες της Python που δεν υπάρχουν μέσα στο conda environment που δημιουργήθηκε νωρίτερα.

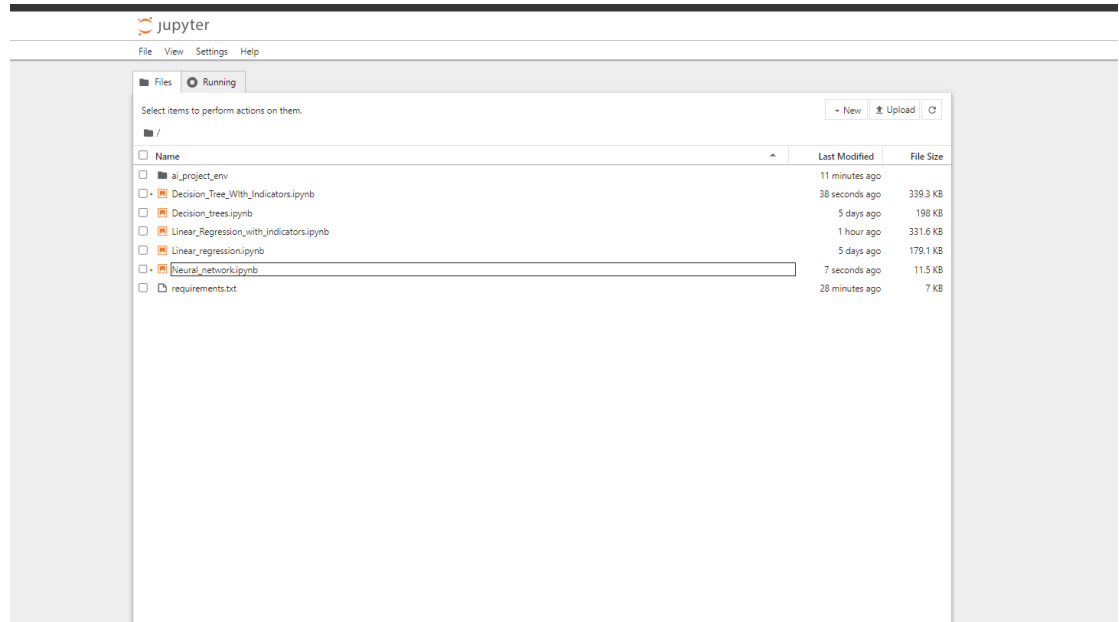
**pip install yfinance**

- Πατάμε την εντολή `conda activate <path_name>`, όπου `<path_name>` θα είναι το αντίστοιχο όπως θα δημιουργηθεί τοπικά στο περιβάλλον του χρήστη. Στο παραπάνω περιβάλλον αυτή η εντολή θα ήταν: **conda activate C:\Users\harit\Desktop\ai\_dokimastiko\_unzip\ai\_project\_env**

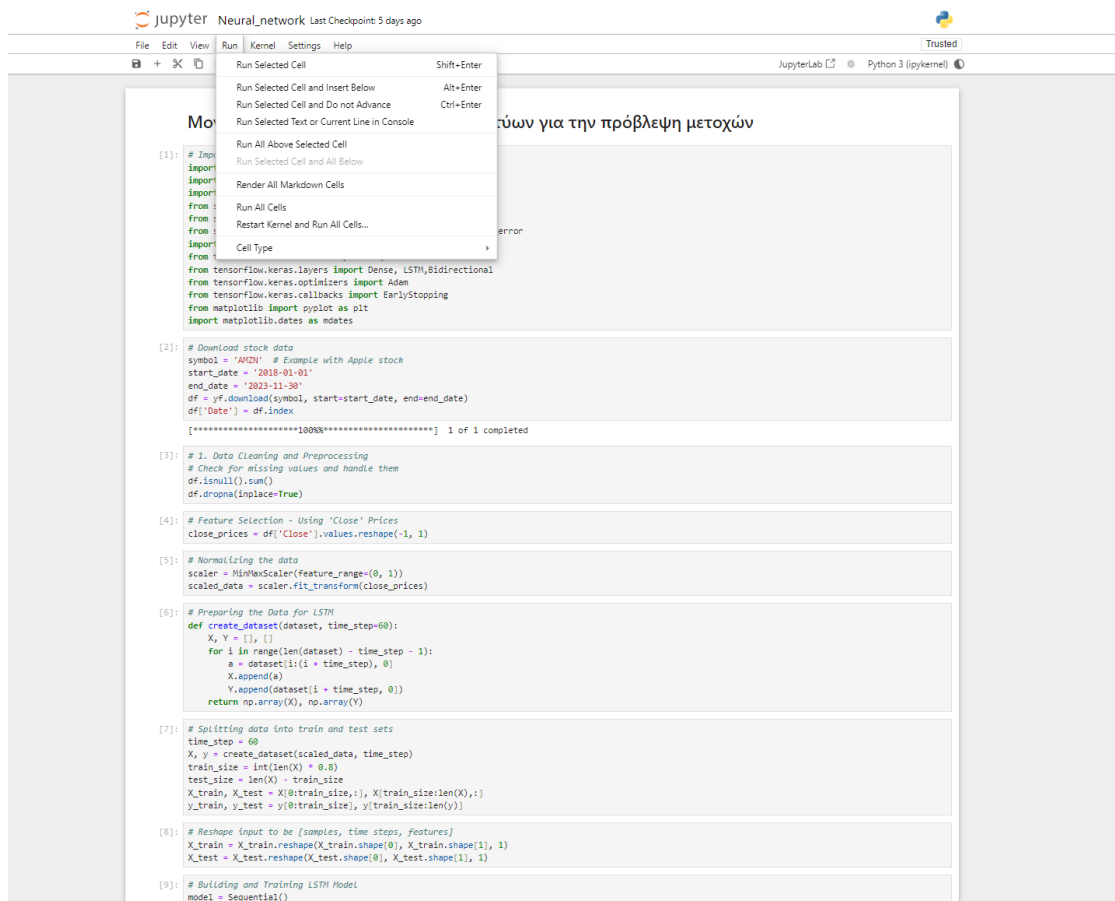
Με την παραπάνω εντολή θα μας ενεργοποιήσει το περιβάλλον και θα εμφανίσει στον conda prompt κάτι της μορφής:

```
(C:\Users\harit\Desktop\ai_dokimastiko_unzip\ai_project_env) C:\Users\harit\Desktop\ai_dokimastiko_unzip>
```

- Έπειτα πατάμε **jupyter notebook** στον conda prompt όπου θα μας ανοίξει το αντίστοιχο παράθυρο του jupyter στον browser.
- Από το εικονικό περιβάλλον που θα δούμε στον browser που θα είναι της παρακάτω μορφής, επιλέγουμε το .ipynb αρχείο που θέλουμε να εκτελέσουμε.



- Αφού ανοίξουμε το επιθυμητό jupyter file, από το Run που υπάρχει στο κεντρικό μενού πατάμε το Run All Cells.



- Δίπλα στο κάθε κελί εμφανίζει έναν αριθμό που υποδηλώνει σε ποια σειρά εκτελέστηκε το κελί. Όταν εμφανίσει αριθμό σημαίνει ότι εκτελέστηκε. Μετά από λίγο χρονικό διάστημα (τα κελιά που εμπεριέχουν την εκπαίδευση μοντέλου ή την εισαγωγή βιβλιοθηκών χρειάζονται περισσότερο χρόνο για να εκτελεστούν), όλα τα κελιά θα πρέπει να έχουν εκτελεστεί.

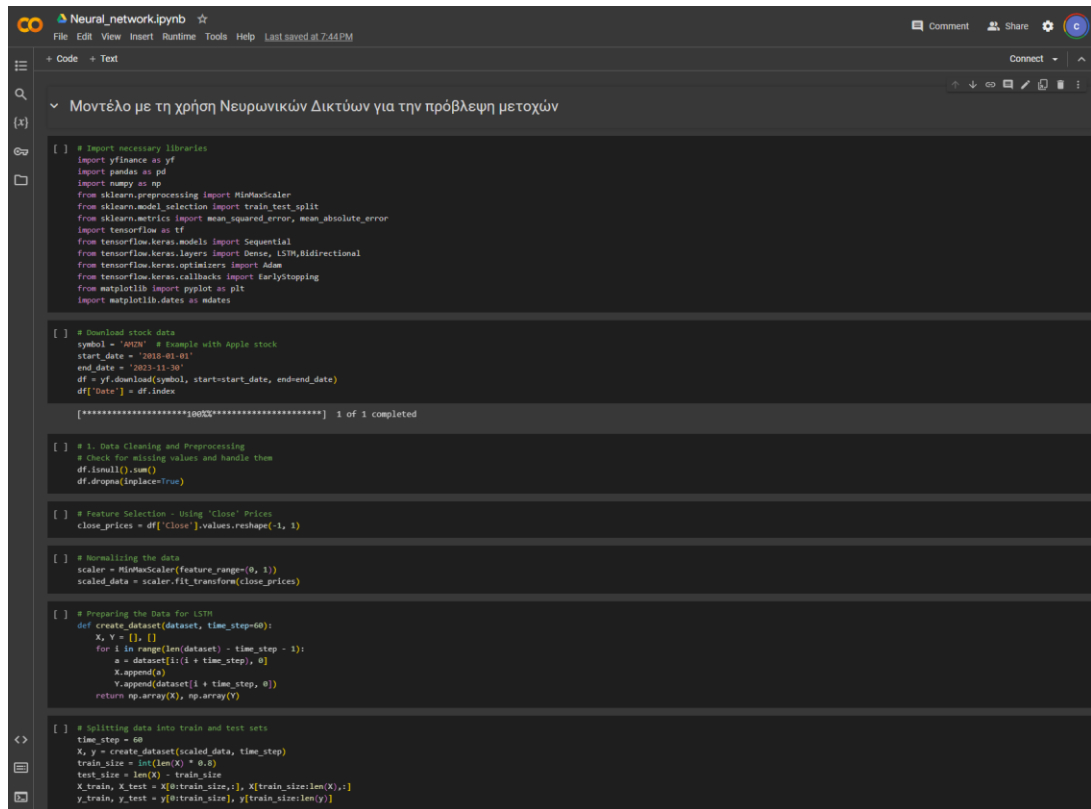
## 2. Μέσω του Google Colab.

Για τον τρόπο αυτό η διαδικασία που πρέπει να ακολουθηθεί είναι η εξής:

- Μετάβαση στο επίσημο site του google colab. <https://colab.research.google.com/>
- Μας εμφανίζει ένα παράθυρό που λέει Open notebook. Επιλέγουμε το Upload και πάμε στο τοπικό file που θέλουμε να ανοίξουμε.



- Έπειτα αφού το ανοίξει μας ανακατευθύνει σε αυτό το file.



```
[ ] # Import necessary libraries
import yfinance as yf
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Bidirectional
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from matplotlib import pyplot as plt
import matplotlib.dates as mdates

[ ] # Download stock data
symbol = 'AAPL' # Example with Apple stock
start_date = '2018-01-01'
end_date = '2023-12-30'
df = yf.download(symbol, start=start_date, end=end_date)
df['Date'] = df.index

[ ] # 1. Data Cleaning and Preprocessing
# Check for missing values and handle them
df.isnull().sum()
df.dropna(inplace=True)

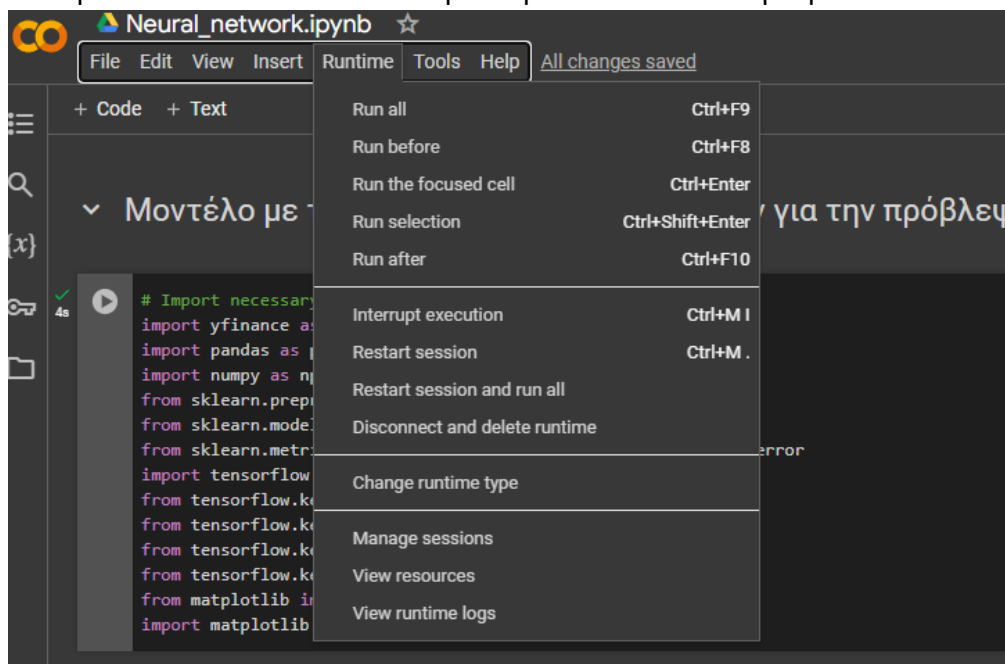
[ ] # Feature Selection - Using 'Close' Prices
close_prices = df['Close'].values.reshape(-1, 1)

[ ] # Normalizing the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(close_prices)

[ ] # Preparing the Data for LSTM
def create_dataset(dataset, time_step=60):
    X, Y = [], []
    for i in range(len(dataset) - time_step - 1):
        a = dataset[i:(i + time_step), 0]
        X.append(a)
        Y.append(dataset[i + time_step, 0])
    return np.array(X), np.array(Y)

[ ] # Splitting data into train and test sets
time_step = 60
X, y = create_dataset(scaled_data, time_step)
train_size = int(len(X) * 0.8)
test_size = len(X) - train_size
X_train, X_test = X[0:train_size, :], X[train_size:len(X), :]
y_train, y_test = y[0:train_size], y[train_size:len(y)]
```

- Πατάμε Runtime από το κεντρικό μενού και επιλέγουμε το Run All



- Όπως και στο jupyter θα μας εμφανίζει με νούμερο το κελί δίπλα όταν θα έχει εκτελεστεί.

```

File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

▼ Μοντέλο με τη χρήση Νευρωνικών Δικτύων για την πρόβλεψη μετοχών

[1] # Import necessary libraries
import yfinance as yf
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Bidirectional
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from matplotlib import pyplot as plt
import matplotlib.dates as mdates

[2] # Download stock data
symbol = 'AAPL' # Example with Apple stock
start_date = '2018-01-01'
end_date = '2023-11-30'
df = yf.download(symbol, start=start_date, end=end_date)
df['Date'] = df.index

[*****] 1 of 1 completed

[3] # 1. Data Cleaning and Preprocessing
# Check for missing values and handle them
df.isnull().sum()
df.dropna(inplace=True)

[4] # Feature Selection - Using 'Close' Prices
close_prices = df['Close'].values.reshape(-1, 1)

[5] # Normalizing the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(close_prices)

[6] # Preparing the Data for LSTM
def create_dataset(dataset, time_step=60):
    X, Y = [], []
    for i in range(len(dataset) - time_step - 1):
        a = dataset[i:(i + time_step), 0]
        X.append(a)
        Y.append(dataset[i + time_step, 0])
    return np.array(X), np.array(Y)

[7] # Splitting data into train and test sets
time_step = 60
X, y = create_dataset(scaled_data, time_step)
train_size = int(len(X) * 0.8)
test_size = len(X) - train_size
X_train, X_test = X[0:train_size, :], X[train_size:len(X), :]
y_train, y_test = y[0:train_size], y[train_size:len(y)]
  
```

- Σε περίπτωση που θέλουμε να εκτελέσουμε ένα κελί μεμονωμένα, επιλέγουμε το κελί και πατάμε Shift+Enter
- Τα κελιά που έχουν εισαγωγή βιβλιοθηκών όπως και εκπαίδευση μοντέλων, παίρνουν λίγο περισσότερο χρόνο προκειμένου να εκτελεστούν.

Στην συνέχεια θα αναλυθούν οι κώδικες για κάθε ένα αλγόριθμοι, καθώς και τα αποτελέσματα που δίνουν.

Να σημειωθεί πως όλοι οι αλγόριθμοι δουλεύουν με μία συγκεκριμένη μετοχή, αυτή της AMAZON και όλα τα αποτελέσματα αφορούν αυτή την μετοχή. Συνεπώς τα αποτελέσματα είναι αποδιδόμενα με βάση την συγκεκριμένη μετοχή. Είναι πιθανό πως με κάποια άλλη μετοχή ο κάθε αλγόριθμος ίσως έχει διαφορετικά αποτελέσματα.

Το χρονικό περιθώριο που έχουμε βάλει τους αλγορίθμους να τραβάνε δεδομένα είναι τα τελευταία 5 χρόνια.

## Κώδικας Αλγόριθμου Γραμμικής Παλινδρόμησης

Παρακάτω φαίνεται ο κώδικας του αλγορίθμου Γραμμικής Παλινδρόμησης (Input):

Πάνω από κάθε εντολή υπάρχουν και τα απαραίτητα σχόλια ώστε να εξηγείται σε κάθε γραμμή τι κάνει ο αντίστοιχος κώδικας.

```
# Εισαγωγή απαραίτητων βιβλιοθηκών
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
```

Στις πρώτες γραμμές του κώδικα γίνεται η εισαγωγή των βιβλιοθηκών που χρειάζονται για τον αλγόριθμο.

```
symbol = 'AMZN' # Παράδειγμα με μετοχή Amazon
start_date = '2018-11-30'
end_date = '2023-11-30'
df = yf.download(symbol, start=start_date, end=end_date)
```

Στην συνέχεια επιλέγεται η μετοχή που θα αναλυθεί, η οποία είναι η Amazon, και καθορίζονται οι ημερομηνίες της μετοχής που θα κατεβάσουμε μέσω του yfinance.

```
# Καθαρισμός και επεξεργασία δεδομένων
df.dropna(inplace=True) # Διαγραφή απουσιάζουσων τιμών
df['Return'] = df['Close'].pct_change() # Υπολογισμός ημερήσιας
απόδοσης
df = df.reset_index()
```

Επιπλέον γίνεται καθορισμός και επεξεργασία δεδομένων, διαγράφοντας τις απουσιάζουσες τιμές και υπολογίζοντας την ημερήσια απόδοση.

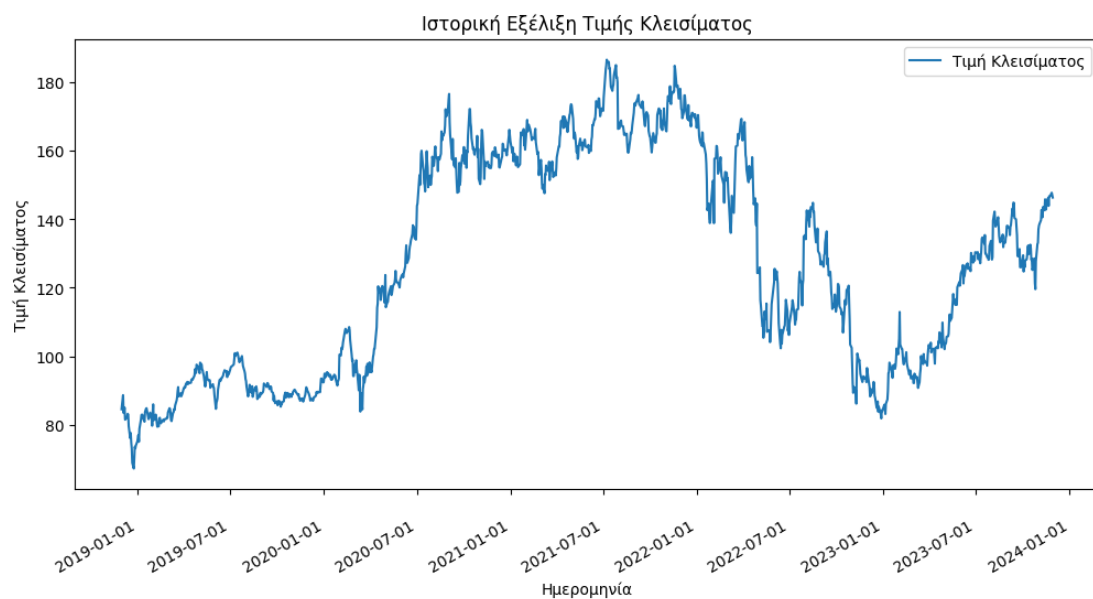
```
plt.figure(figsize=(12, 6))
# Μετατροπή της στήλης 'Date' σε datetime
df['Date'] = pd.to_datetime(df['Date'])
plt.plot(df['Date'], df['Close'], label='Τιμή Κλεισίματος')
plt.title('Ιστορική Εξέλιξη Τιμής Κλεισίματος')
plt.xlabel('Ημερομηνία')
plt.ylabel('Τιμή Κλεισίματος')
plt.legend()
plt.gcf().autofmt_xdate() # rotate στον άξονα x για καλύτερη ανάγνωση
των ημερομηνιών

# Βελτιστοποίηση του άξονα x για να εμφανίζονται μόνο οι μήνες
Ιανουάριος και Ιούλιος
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(bymonth=(1, 7)))
plt.gca().xaxis.set_tick_params(which='major', pad=15)

plt.show()
```

Σε αυτό το κομμάτι του κώδικα παρουσιάζεται η οπτικοποίηση των δεδομένων. Στην πρώτη γραμμή καθορίζονται οι διαστάσεις του διαγράμματος το οποίο θα έχει την μορφή plot. Μετά γίνεται η μετατροπή της στήλης Date σε datetime και ορίζονται τα ονόματα του τίτλου, του κάθετου και του οριζόντιου άξονα που θα έχει το διάγραμμα. Επιπλέον γίνεται rotate στον άξονα x και βελτιστοποίηση του άξονα x.

Το διάγραμμα που εμφανίζεται είναι το εξής(output):



```
# Δημιουργία μεταβλητών για το μοντέλο μηχανικής μάθησης
df['Previous_Close'] = df['Close'].shift(1) # Τιμή κλεισίματος
προηγούμενης ημέρας
df.dropna(inplace=True) # Διαγραφή απουσιάζουσων τιμών μετά τη
μετατόπιση
```

Ακολουθεί η δημιουργία μεταβλητών για το μοντέλο μηχανικής μάθησης και αντικαθιστούνται οι τιμές του προηγούμενου κλεισίματος με του τωρινού καθώς επίσης γίνεται και διαγραφή απουσιάζουσων τιμών μετά από αυτή την μετατόπιση.

```
# Διαχωρισμός σε σετ εκπαίδευσης και δοκιμής
X = df[['Date', 'Previous_Close']]
y = df['Close']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, shuffle=False)
```

. Στη συγκεκριμένη περίπτωση, το `X` περιέχει δύο στήλες από τ

Στο συγκεκριμένο σημείο του κώδικα γίνεται διαχωρισμός σε σετ εκπαίδευσης και δοκιμής. Εδώ το `x` περιέχει δύο στήλες από το πλαίσιο δεδομένων `df`: τη στήλη 'Date' και τη στήλη 'Previous\_Close', ενώ το `y` περιέχει τη στήλη 'Close'. Μετά οι μεταβλητές `x` και `y` εκπαιδεύονται και δοκιμάζονται για τον έλεγχο του μοντέλου σε ένα τεστάρισμα μεγέθους 20%.

```
# Εκπαίδευση του μοντέλου
model = LinearRegression()
model.fit(X_train.drop('Date', axis=1), y_train)

# Πρόβλεψη τιμών
predictions = model.predict(X_test.drop('Date', axis=1))
```

Στην συνέχεια γίνεται η εκπαίδευση του μοντέλου με την χρήση του αλγορίθμου γραμμικής παλινδρόμησης και στην συνέχεια η πρόβλεψη που θα βγάλει το μοντέλο με τον συγκεκριμένο αλγόριθμο.

```
# Αξιολόγηση του μοντέλου
```

```
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
print(f'MSE: {mse}')
print(f'MAE: {mae}')
```

Επιπλέον γίνεται αξιολόγηση του μοντέλου με την χρήση 2 μετρικών, του mse και mae.

Το Mean Squared Error (MSE) είναι μια μετρική που μετρά τη μέση τετραγωνική απόκλιση μεταξύ των πραγματικών και προβλεπόμενων τιμών. Τα ορίσματα της στην συγκεκριμένη περίπτωση είναι τα εξής:

y\_test: Οι πραγματικές τιμές της μεταβλητής κατάστασης από το σετ δοκιμής.

predictions: Οι προβλεπόμενες τιμές της μεταβλητής κατάστασης από το μοντέλο.

Το Mean Absolute Error (MAE) είναι μια μετρική που μετρά τη μέση απόλυτη απόκλιση μεταξύ των πραγματικών και προβλεπόμενων τιμών. Τα ορίσματα της στην συγκεκριμένη περίπτωση είναι τα εξής:

y\_test: Οι πραγματικές τιμές της μεταβλητής κατάστασης από το σετ δοκιμής.

predictions: Οι προβλεπόμενες τιμές της μεταβλητής κατάστασης από το μοντέλο.

Τα αποτελέσματα αυτών των 2 μετρικών είναι τα εξής:

MSE: 5.1452285628148715

MAE: 1.7473207387118779

### 1. MSE (Mean Squared Error):

- Η τιμή του MSE που παρέχετε (5.1452285628148715) είναι το μέσο τετραγωνικό σφάλμα μεταξύ των πραγματικών και προβλεπόμενων τιμών.
- Όσο μικρότερη είναι η τιμή του MSE, τόσο καλύτερη είναι η απόδοση του μοντέλου.
- Σε αυτήν την περίπτωση, μια τιμή 5.145 δείχνει ότι υπάρχει κάποια διασπορά στα τετραγωνικά σφάλματα.

### 2. MAE (Mean Absolute Error):

- Η τιμή του MAE που παρέχετε (1.7473207387118779) είναι το μέσο απόλυτο σφάλμα μεταξύ των πραγματικών και προβλεπόμενων τιμών.

- Όσο μικρότερη είναι η τιμή του MAE, τόσο καλύτερη είναι η απόδοση του μοντέλου.
- Σε αυτήν την περίπτωση, μια τιμή 1.747 δείχνει ότι υπάρχει κάποιο απόλυτο σφάλμα, αλλά η απόκλιση δεν είναι πολύ μεγάλη.

Συνολικά, κρίνοντας από τις τιμές που παρέχονται, μπορούμε να διαπιστώσουμε ότι το μοντέλο έχει κάποιο βαθμό σφάλματος στις προβλέψεις του, αλλά εξακολουθεί να παρουσιάζει μια ανοχή.

```
plt.figure(figsize=(12, 6))
plt.plot(X_test['Date'], y_test, color='blue', label='Πραγματικές τιμές') # Πραγματικές τιμές
plt.plot(X_test['Date'], predictions, color='orange', label='Προβλεπόμενες τιμές') # Προβλεπόμενες τιμές
plt.title('Πραγματικές και προβλεπόμενες τιμές της μετοχής Amazon')
plt.xlabel('Ημερομηνία')
plt.ylabel('Τιμή Μετοχής')
plt.legend()
plt.gcf().autofmt_xdate()
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=6))

plt.show()
```

Στο τέλος του κώδικα γίνεται η οπτικοποίηση των δεδομένων μεταξύ πραγματικής τιμής της μετοχής και των προβλέψεων που έδωσε ο κώδικας.

Η πρώτη γραμμή δημιουργεί ένα σχήμα μεγέθους 12x6. Στην συνέχεια δημιουργείται η γραφική αναπαράσταση τόσο των πραγματικών (μπλε χρώμα), όσο και των προβλεπόμενων(πορτοκαλί χρώμα) τιμών.

Επιπλέον δίνεται ο τίτλος που θα έχει το διάγραμμα, και οι ετικέτες που θα έχουν ο οριζόντιος και ο κάθετος άξονας.

Οι τελευταίες εντολές ενεργοποιούν την αυτόματη μορφοποίηση των ημερομηνιών στον άξονα x, για να είναι πιο ευανάγνωστες, ορίζουν τον τρόπο μορφοποίησης των ημερομηνιών και το διάστημα εμφάνισης των σημείων στον άξονα x (σε αυτήν την περίπτωση, κάθε 6 μήνες).

Τέλος δίνεται η εντολή για να εμφανιστεί το διάγραμμα, το οποίο είναι το εξής(output):



Αυτό που παρατηρείται είναι πως οι προβλεπόμενες τιμές που έδωσε ο αλγόριθμος σε σχέση με τις πραγματικές είναι κοντά, αλλά η απόκλιση που υπάρχει είναι εμφανές ότι δεν κρύβεται. Συμπερασματικά θα μπορούσε να αποτελεί ένα μέτριο δείγμα για πρόβλεψη της συγκεκριμένης μετοχής με περιθώριο βελτιστοποίησης.

## Κώδικας Αλγορίθμου Δέντρων Απόφασης

Παρακάτω φαίνεται ο κώδικας του αλγορίθμου Δέντρων Απόφασης (Input):

Πάνω από κάθε εντολή υπάρχουν και τα απαραίτητα σχόλια ώστε να εξηγείται σε κάθε γραμμή τι κάνει ο αντίστοιχος κώδικας.

```
import yfinance as yf
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
```

Στις πρώτες γραμμές του κώδικα γίνεται η εισαγωγή των βιβλιοθηκών που χρειάζονται για τον αλγόριθμο.



```
# Λήψη δεδομένων μετοχής από το yfinance
symbol = 'AMZN' # Παράδειγμα με μετοχή Amazon
start_date = '2018-11-30'
end_date = '2023-11-30'
df = yf.download(symbol, start=start_date, end=end_date)
```

Στην συνέχεια επιλέγεται η μετοχή που θα αναλυθεί, η οποία είναι η Amazon, και καθορίζονται οι ημερομηνίες της μετοχής που θα κατεβάσουμε μέσω του yfinance.

```
df.dropna(inplace=True) # Αφαίρεση γραμμών με NaN
```

Η συγκεκριμένη εντολή χρησιμοποιείται για να αφαιρέσει τις γραμμές από ένα πλαίσιο δεδομένων που περιέχουν τιμές NaN (Not a Number).

```
# Προετοιμασία δεδομένων
df['Date'] = df.index
X = df[['Open', 'High', 'Low', 'Volume', 'Adj Close']]
y = df['Close']
```

Στην συνέχεια γίνεται η προετοιμασία των δεδομένων.

Αρχικά δημιουργεί μια νέα στήλη 'Date' στο πλαίσιο δεδομένων df και την γεμίζει με τις ημερομηνίες που χρησιμοποιούνται ως δείκτες (index) του πλαισίου δεδομένων.

Έπειτα δημιουργεί ένα νέο πλαίσιο δεδομένων X, το οποίο περιλαμβάνει τα χαρακτηριστικά που χρησιμοποιούνται για την εκπαίδευση του μοντέλου. Τα χαρακτηριστικά αυτά είναι οι στήλες 'Open', 'High', 'Low', 'Volume', και 'Adj Close' του αρχικού πλαισίου δεδομένων df.

Ακόμη δημιουργεί τη μεταβλητή κατάστασης y, που περιέχει τις τιμές της στήλης 'Close' του αρχικού πλαισίου δεδομένων df. Η μεταβλητή αυτή αντιπροσωπεύει την τιμή που θέλουμε να προβλέψουμε με το μοντέλο.

```
# Διαχωρισμός σε train και test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Δημιουργία και εκπαίδευση μοντέλου
model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)

# Υπολογισμός της πρόβλεψης
predictions = model.predict(X_test)
```

Σ αυτό το κομμάτι οι μεταβλητές  $x$  και  $y$  εκπαιδεύονται και δοκιμάζονται για τον έλεγχο του μοντέλου σε ένα τεστάρισμα μεγέθους 20%.

Επιπλέον γίνεται η δημιουργία του μοντέλου του αλγορίθμου των δέντρων απόφασης καθώς και η εκπαίδευση του και στην συνέχεια υπολογίζεται η πρόβλεψη που δίνει το μοντέλο.

```
# Σορτάρισμα των δεδομένων για την εμφάνιση των προβλέψεων
test_data_sorted = X_test.copy()
test_data_sorted['Predictions'] = predictions
test_data_sorted = test_data_sorted.sort_index()
```

Σε αυτό το σημείο του κώδικα γίνεται σορτάρισμα των δεδομένων για την εμφάνιση των προβλέψεων. Αρχικά δημιουργεί μια αντιγραφή του πλαισίου δεδομένων `X_test` και την αποθηκεύει στη μεταβλητή `test_data_sorted`.

Έπειτα προσθέτει μια νέα στήλη με το όνομα 'Predictions' στο πλαίσιο δεδομένων `test_data_sorted` και τη γεμίζει με τις προβλεπόμενες τιμές που προέκυψαν από το μοντέλο.

Τέλος ταξινομεί το πλαίσιο δεδομένων `test_data_sorted` βάσει του δείκτη (index), ώστε οι προβλέψεις (και τα αντίστοιχα χαρακτηριστικά) να εμφανίζονται με χρονολογική σειρά μετά την εκτέλεση αυτής της γραμμής κώδικα.

Συνολικά, αυτός ο κώδικας δημιουργεί ένα νέο πλαίσιο δεδομένων `test_data_sorted` που περιέχει τις προβλεπόμενες τιμές σε χρονολογική σειρά για ευκολότερο έλεγχο.

```
# Αξιολόγηση του μοντέλου με τις μετρικές MSE και MAE
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)

print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
```

Επιπλέον γίνεται αξιολόγηση του μοντέλου με την χρήση 2 μετρικών, του `mse` και `mae`.

Το Mean Squared Error (MSE) είναι μια μετρική που μετρά τη μέση τετραγωνική απόκλιση μεταξύ των πραγματικών και προβλεπόμενων τιμών. Τα ορίσματα της στην συγκεκριμένη περίπτωση είναι τα εξής:

y\_test: Οι πραγματικές τιμές της μεταβλητής κατάστασης από το σετ δοκιμής.

predictions: Οι προβλεπόμενες τιμές της μεταβλητής κατάστασης από το μοντέλο.

Το Mean Absolute Error (MAE) είναι μια μετρική που μετρά τη μέση απόλυτη απόκλιση μεταξύ των πραγματικών και προβλεπόμενων τιμών. Τα ορίσματα της στην συγκεκριμένη περίπτωση είναι τα εξής:

y\_test: Οι πραγματικές τιμές της μεταβλητής κατάστασης από το σετ δοκιμής.

predictions: Οι προβλεπόμενες τιμές της μεταβλητής κατάστασης από το μοντέλο.

Τα αποτελέσματα αυτών των 2 μετρικών είναι τα εξής:

MSE: 0.07453622392102492

MAE: 0.14141430930485802

### **1. Mean Squared Error (MSE):**

- Η τιμή MSE που παρέχετε (0.07453622392102492) είναι το μέσο τετραγωνικό σφάλμα μεταξύ των πραγματικών και προβλεπόμενων τιμών.
- Όσο μικρότερη είναι η τιμή του MSE, τόσο καλύτερη είναι η απόδοση του μοντέλου.
- Σε αυτήν την περίπτωση, μια τιμή 0.074 δείχνει ότι οι προβλέψεις του μοντέλου δεν έχουν κάποιο ιδιαίτερο επίπεδο τετραγωνικού σφάλματος και μπορεί να θεωρηθεί ως αποδεκτό, αφού η τιμή είναι πολύ κοντά στο 0.

### **2. Mean Absolute Error (MAE):**

- Η τιμή MAE που παρέχετε (0.14141430930485802) είναι το μέσο απόλυτο σφάλμα μεταξύ των πραγματικών και προβλεπόμενων τιμών.
- Όσο μικρότερη είναι η τιμή του MAE, τόσο καλύτερη είναι η απόδοση του μοντέλου.
- Σε αυτήν την περίπτωση, μια τιμή 0.141 δείχνει ότι υπάρχει κάποιο μικρό απόλυτο σφάλμα στις προβλέψεις, αλλά η απόκλιση δεν είναι πολύ μεγάλη.

Συνολικά, οι τιμές αυτές μπορεί να εκτιμηθούν ως αποδεκτές, αλλά εξαρτάται και από το πλαίσιο του προβλήματος και τη φύση των δεδομένων που χρησιμοποιήθηκαν για την εκπαίδευση του μοντέλου.

```
# Παρουσίαση αποτελεσμάτων
plt.figure(figsize=(10,6))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=6))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
plt.plot(df.index, df['Close'], label='Πραγματικές Τιμές',
color='blue', linewidth=2)
plt.scatter(test_data_sorted.index, test_data_sorted['Predictions'],
label='Προβλεπόμενες Τιμές', color='orange', s=10)
plt.xlabel('Ημερομηνία')
plt.ylabel('Τιμή')
plt.title('Πρόβλεψη τιμής για την μετοχή της Amazon με Decision Tree')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Ο παραπάνω κώδικας χρησιμοποιείται για την οπτικοποίηση των δεδομένων. Στην πρώτη γραμμή κώδικα δημιουργείται ένα νέο σχήμα για το γράφημα, ορίζοντας το μέγεθος του ως 10X6. Στην δεύτερη γραμμή κώδικα Ορίζει τον κυρίαρχο τοποθετητή (locator) για τον άξονα x, έτσι ώστε οι ετικέτες του άξονα να είναι κυρίως σε κάθε 6 μήνες. Στην τρίτη γραμμή κώδικα ορίζει τον μορφοποιητή (formatter) του κυρίαρχου τοποθετητή, ώστε οι ημερομηνίες να εμφανίζονται σε μορφή έτος-μήνας.

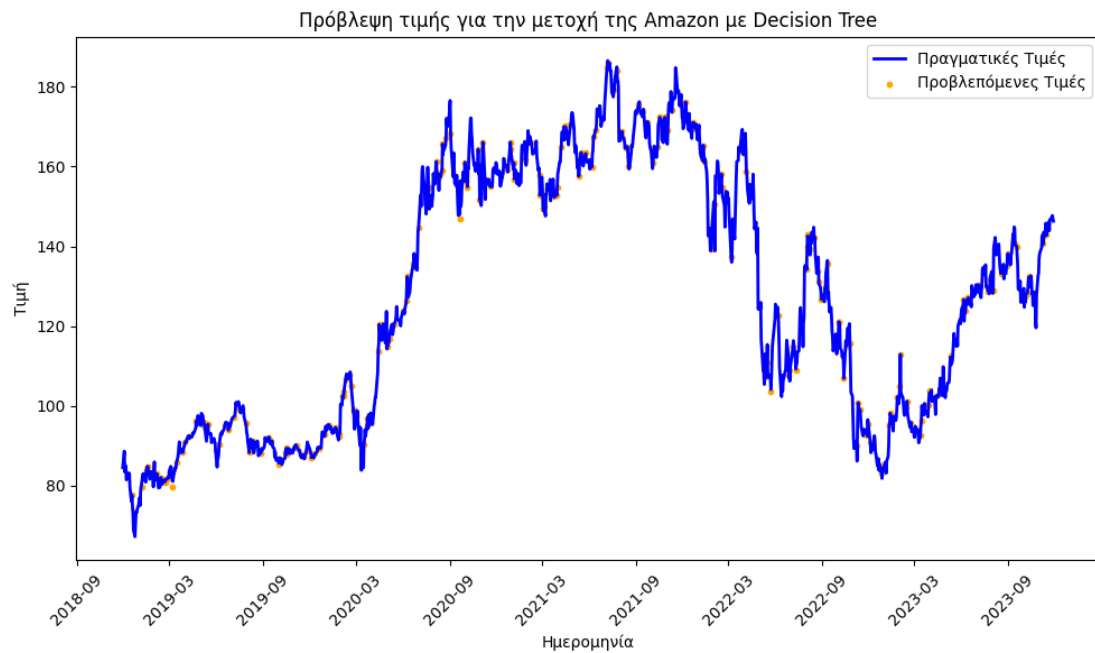
Έπειτα σχεδιάζει τη γραφική παράσταση των πραγματικών τιμών (στήλη 'Close') ως γραμμή με χρώμα μπλε και πάχος γραμμής 2.

Στην συνέχεια ορίζονται οι ετικέτες του άξονα x και του άξονα y, καθώς και ο τίτλος του διαγράμματος. Ακόμη εμφανίζει το κλειδί του γραφήματος, περιλαμβάνοντας τις ετικέτες 'Πραγματικές Τιμές' και 'Προβλεπόμενες Τιμές' και περιστρέφει τις ετικέτες του άξονα x κατά 45 βαθμούς για καλύτερη αναγνωσιμότητα.

Τέλος βελτιστοποιεί την διάσταση και την τοποθέτηση των στοιχείων για να αποφευχθεί τυχόν κοπή τους και η τελευταία εντολή είναι για την εμφάνιση του διαγράμματος.

Συνολικά, ο κώδικας δημιουργεί ένα γράφημα που περιλαμβάνει τις πραγματικές τιμές και τις προβλεπόμενες τιμές, επιτρέποντας την οπτικοποίηση της απόδοσης του μοντέλου.

Το γράφημα που προκύπτει είναι το εξής(output):



Αυτό που παρατηρείται είναι ότι οι προβλεπόμενες με τις πραγματικές τιμές ταυτίζονται σχεδόν σε όλο το μήκος του διαγράμματος και υπάρχει κάποιες πολύ μικρές αποκλίσεις. Αυτό δείχνει σε συνδυασμό και με τις μετρικές mse και mae ότι ο αλγόριθμος έχει πολύ καλά ποσοστά πρόβλεψης για την συγκεκριμένη μετοχή και ότι ίσως αποτελεί ένα καλό εργαλείο πρόβλεψης για την συγκεκριμένη μετοχή στην συγκεκριμένη περίοδο.

```
# Υπολογίζουμε το μέσο όρο και την τυπική απόκλιση των τελευταίων 30
# ημερών για τα features
mean_values = df[['Open', 'High', 'Low', 'Volume', 'Adj
Close']].tail(30).mean()
std_values = df[['Open', 'High', 'Low', 'Volume', 'Adj
Close']].tail(30).std()

# Προσδιορίζουμε την τελευταία ημερομηνία των δεδομένων
last_date = df.index[-1]

# Δημιουργούμε ένα νέο DataFrame για τις μελλοντικές ημερομηνίες
future_dates = pd.date_range(start=last_date, periods=30, freq='D')
future_df = pd.DataFrame(index=future_dates)

# Προσθέτουμε τυχαία διακύμανση στις μέσες τιμές για κάθε feature
for feature in mean_values.index:
    future_df[feature] = mean_values[feature] + np.random.normal(0,
std_values[feature], size=len(future_dates))
```

```
# Κάνουμε προβλέψεις χρησιμοποιώντας το μοντέλο
future_predictions = model.predict(future_df[['Open', 'High', 'Low',
'Volume', 'Adj Close']])

# Εκτυπώνουμε τις προβλέψεις
future_df['Predictions'] = future_predictions
```

Ξεκινώντας να αναλύουμε τον παραπάνω κώδικα γραμμή-γραμμή, στις πρώτες γραμμές του κώδικα υπολογίζουμε το μέσο όρο και την τυπική απόκλιση των τελευταίων 30 ημερών για τα features. Συγκεκριμένα με την πρώτη εντολή επιλέγονται τα τελευταία 30 δείγματα για κάθε χαρακτηριστικό (Open, High, Low, Volume, Adj Close) από το DataFrame(df) και στην δεύτερη εντολή υπολογίζεται ο μέσος όρος (mean\_values) και η τυπική απόκλιση (std\_values) για κάθε χαρακτηριστικό.

Μετά γίνεται ο προσδιορισμός της τελευταίας ημερομηνίας των δεδομένων και στην μεταβλητή last\_date εκχωρείται η τελευταία ημερομηνία από τον δείκτη df.

Έπειτα δημιουργούμε ένα νέο DataFrame για τις μελλοντικές ημερομηνίες, όπου η πρώτη εντολή δημιουργεί μία σειρά μελλοντικών ημερομηνιών για τις επόμενες 30 ημέρες και η δεύτερη εντολή δημιουργεί ένα νέο DataFrame (το future\_df) με αυτές τις ημερομηνίες ως δείκτη.

Μετά προσθέτουμε μία τυχαία διακύμανση στις μέσες τιμές για κάθε feature. Αυτό γίνεται με μία εντολή επανάληψης (for...in)

Στην συνέχεια κάνουμε προβλέψεις χρησιμοποιώντας το συγκεκριμένο μοντέλο και εκτυπώνουμε τα δεδομένα σαν στήλη(predictions) στο future\_df.

```
# Ρυθμίζουμε τον οριζόντιο άξονα (x-axis) για να εμφανίζει ημερομηνίες
κάθε έξι μήνες
locator = mdates.MonthLocator(interval=6)
formatter = mdates.DateFormatter('%Y-%m')

# Δημιουργία του διαγράμματος
plt.figure(figsize=(14, 7))
plt.gca().xaxis.set_major_locator(locator)
plt.gca().xaxis.set_major_formatter(formatter)

# Πραγματικές τιμές
plt.plot(df.index, df['Close'], label='Πραγματικές Τιμές',
color='blue')
```

```
# Προβλεπόμενες τιμές για το μέλλον
plt.plot(future_df.index, future_df['Predictions'],
label='Προβλεπόμενες Τιμές', color='orange', linestyle='--')

plt.xlabel('Ημερομηνία')
plt.ylabel('Τιμή')
plt.title('Πρόβλεψη τιμής για την μετοχή της Amazon με Προσομοίωση
Μελλοντικών Δεδομένων')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

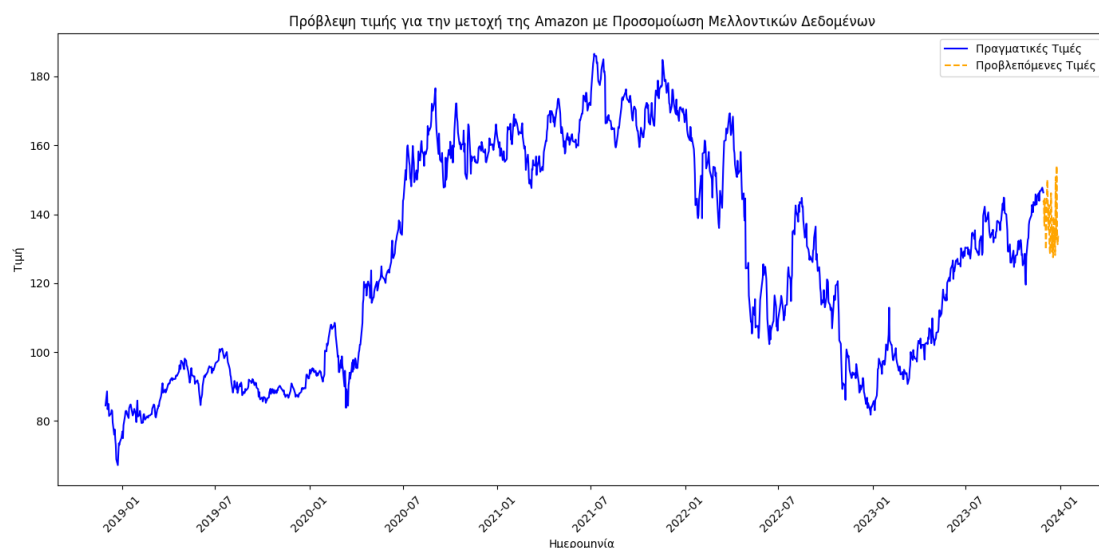
Το συγκεκριμένο κομμάτι κώδικα ξεκινάει με την δημιουργία ενός locator που ορίζει τα σημεία επιλογής κάθε 6 μήνες και την δημιουργία ενός formatter που μορφοποιεί τις ημερομηνίες στη μορφή έτος-μήνας.

Μετά γίνεται η δημιουργία του διαγράμματος με διαστάσεις 14X7.

Μετά γίνεται ο σχεδιασμός των πραγματικών τιμών(στήλη Close) στο διάγραμμα, οι οποίες θα απεικονίζονται με μπλε χρώμα, καθώς και ο σχεδιασμός των προβλεπόμενων τιμών(στήλη Predictions) οι οποίες θα απεικονίζονται με πορτοκαλί χρώμα.

Τέλος καθορίζονται τα ονόματα του τίτλου, της κάθετης και της οριζόντιας ετικέτας, εμφανίζεται το κλειδί του γραφήματος και εμφανίζεται η τελική μορφή του.

Συνεπώς το διάγραμμα που προκύπτει είναι το εξής(output):



Στο συγκεκριμένο διάγραμμα φαίνεται η κίνηση της μετοχής με μπλε χρώμα την τελευταία 5ετία. Στην συνέχεια βλέπουμε ότι το διάγραμμα συνεχίζει με μία διακεκομμένη πορτοκαλί γραμμή, η οποία στην ουσία είναι η πρόβλεψη που δίνει ο

αλγόριθμος για την συγκεκριμένη μετοχή. Δείχνει δηλαδή το πώς θα κινηθεί η μετοχή το επόμενο διάστημα.

Αυτό που παρατηρείται είναι ότι δεν υπάρχει κάποια σταθερή πορεία που θα ακολουθήσει η μετοχή, αλλά αντιθέτως υπάρχει μεγάλη διακύμανση στην μελλοντική τιμή της μετοχής.

Συνεπώς ο συγκεκριμένος αλγόριθμος ενώ είδαμε πως είχε πολύ καλά αποτελέσματα στις προβλέψεις που έκανε για τις μέχρι τώρα τιμές, για τις μελλοντικές τιμές φαίνεται να παρουσιάζει μεγάλες διακυμάνσεις, πράγμα το οποίο δεν αποτελεί και τον πιο αξιόπιστο παράγοντα για την πρόβλεψη μιας μετοχής, καθώς και για το ποια πορεία θα ακολουθήσει μία μετοχή.

## Κώδικας Αλγορίθμου Νευρωνικών Δικτύων

Τα νευρωνικά δίκτυα λειτουργούν μιμούμενα τη δομή και τη λειτουργία του ανθρώπινου εγκεφάλου, αποτελούμενα από σύνδεσμούς που ονομάζονται νευρώνες. Κάθε νευρώνας σε ένα νευρωνικό δίκτυο λαμβάνει εισόδους, τις επεξεργάζεται με βάση ένα σύνολο βαρών και παράγει μία έξοδο. Η διαδικασία εκμάθησης σε ένα νευρωνικό δίκτυο συνήθως επιτυγχάνεται μέσω ενός μηχανισμού ανάδρασης που ονομάζεται backpropagation. Σε αυτήν τη διαδικασία, το δίκτυο συγκρίνει την πραγματική έξοδο με την επιθυμητή και προσαρμόζει τα βάρη των συνδέσεων με βάση τη διαφορά αυτή, βελτιώνοντας σταδιακά την ακρίβειά του.

Ο παρακάτω κώδικας αναπτύσσει ένα μοντέλο πρόβλεψης τιμών μετοχών με τη χρήση νευρωνικών δικτύων, ειδικότερα του τύπου LSTM (Long Short-Term Memory). Το LSTM (ένας ειδικός τύπος νευρωνικού δικτύου που είναι καλό για την επεξεργασία ακολουθιών δεδομένων όπως χρονοσειρές) χρησιμοποιείται για να αναγνωρίσει τα πρότυπα στις τιμές των μετοχών και να κάνει προβλέψεις.

```
# Εισαγωγή βιβλιοθηκών
import yfinance as yf
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Bidirectional
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from matplotlib import pyplot as plt
import matplotlib.dates as mdates
```

Αρχικά εισάγουμε τις απαραίτητες βιβλιοθήκες που θα χρειαστούν στη συνέχεια. Πιο συγκεκριμένα, η yfinance για τη λήψη δεδομένων μετοχών, η pandas και η numpy για



την επεξεργασία των δεδομένων, η sklearn για την προεπεξεργασία, η tensorflow για τη δημιουργία και εκπαίδευση του νευρωνικού δικτύου, και η matplotlib για την οπτικοποίηση των δεδομένων.

```
# Λήψη δεδομένων από το Yahoo Finance
symbol = 'AMZN' # Παράδειγμα μετοχής για την Amazon
start_date = '2018-01-01'
end_date = '2023-11-30'
df = yf.download(symbol, start=start_date, end=end_date)
df['Date'] = df.index
```

Κάνουμε λήψη των δεδομένων για τη μετοχή της AMAZON μέσω του yfinance. Ορίζουμε το όνομα της μετοχής όπως περιμένει να το λάβει η βιβλιοθήκη yfinance. Δηλώνουμε αρχική ημερομηνία για τα δεδομένα που θα λάβουμε και τελική ημερομηνία. Κάνουμε λήψη των δεδομένων και τα καταχωρούμε σε ένα DataFrame της Pandas. Έπειτα λόγω ότι το index είναι της μορφής της ημερομηνίας, δημιουργούμε μία επιπλέον κολώνα για το Date.

```
# Καθαρισμός των δεδομένων και προεπεξεργασία
df.isnull().sum()
df.dropna(inplace=True)
```

Κάνουμε καθαρισμό και προεπεξεργασία των δεδομένων. Αρχικά υπολογίζουμε τον αριθμό των null ή NaN σε κάθε στήλη του DataFrame. Αυτό βοηθά στον εντοπισμό στηλών που έχουν ατελή δεδομένα. Η εντολή df.dropna(inplace=True) αφαιρεί όλες τις γραμμές στο DataFrame που περιέχουν έστω και μία null ή NaN τιμή. Η παράμετρος inplace=True σημαίνει ότι η αλλαγή γίνεται απευθείας στο αρχικό DataFrame, αντί να δημιουργείται ένα νέο DataFrame.

```
close_prices = df['Close'].values.reshape(-1, 1)
```

Μετασχηματίζουμε τις τιμές κλεισίματος σε έναν δισδιάστατο πίνακα numpy.

```
# Κανονικοποίηση δεδομένων
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(close_prices)
```

Κανονικοποιούμε τα δεδομένα. Αρχικά δημιουργούμε ένα αντικείμενο MinMaxScaler από τη βιβλιοθήκη scikit-learn. Ο MinMaxScaler κανονικοποιεί τα δεδομένα στην κλίμακα που ορίζεται στο feature\_range. Στην περίπτωση αυτή, τα δεδομένα θα κλιμακωθούν σε ένα εύρος από 0 έως 1. Στην συνέχεια εφαρμόζεται αυτός ο scaler

στον πίνακα `close_prices` από προηγουμένως. Αποτέλεσμα αυτού είναι ένας νέος πίνακας με τις κλιμακωμένες τιμές.

```
# Προετοιμασία δεδομένων για το μοντέλο
def create_dataset(dataset, time_step=60):
    X, Y = [], []
    for i in range(len(dataset) - time_step - 1):
        a = dataset[i:(i + time_step), 0]
        X.append(a)
        Y.append(dataset[i + time_step, 0])
    return np.array(X), np.array(Y)
```

Στη συνέχεια κάνουμε προετοιμασία των δεδομένων για την εκπαίδευση του μοντέλου. Συγκεκριμένα, δημιουργείται μια λίστα εισόδων (X) και μια λίστα εξόδων (Y) για το μοντέλο. Για κάθε σημείο στο σετ δεδομένων, λαμβάνει μια ακολουθία τιμών (μήκους `time_step`, εδώ ορίζεται ως 60) και την αντιστοιχίζει με την επόμενη τιμή στη σειρά.

```
# Διαχωρισμός σε train και test set
time_step = 60
X, y = create_dataset(scaled_data, time_step)
train_size = int(len(X) * 0.8)
test_size = len(X) - train_size
X_train, X_test = X[0:train_size,:], X[train_size:len(X),:]
y_train, y_test = y[0:train_size], y[train_size:len(y)]
```

Κάνουμε διαχωρισμό σε train και test set. Χρησιμοποιούμε την `create_dataset` από το προηγούμενο βήμα για να δημιουργήσουμε την λίστα εισόδων και την λίστα εξόδων του μοντέλου μας. Ορίζουμε το μέγεθος του συνόλου εκπαίδευσης να είναι στο 80% ενώ του test στο υπόλοιπο 20% και διαχωρίζουμε τα δεδομένα εξόδου σε σύνολα εκπαίδευσης και test.

```
# Μετατροπή σε 3D μορφή για το LSTM
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

Αυτό το κομμάτι κώδικα μετατρέπει τα σετ δεδομένων εκπαίδευσης και δοκιμής σε τρισδιάστατες δομές, οι οποίες είναι απαραίτητες για την είσοδο σε ένα LSTM (Long Short-Term Memory) νευρωνικό δίκτυο. Οι LSTM απαιτούν τρισδιάστατα δεδομένα με διαστάσεις (αριθμός δειγμάτων, βήματα χρόνου, χαρακτηριστικά). Εδώ, το `X_train.shape[0]` είναι ο αριθμός των δειγμάτων, το `X_train.shape[1]` είναι ο αριθμός των βημάτων χρόνου, και το 1 στο τέλος υποδηλώνει ότι κάθε βήμα χρόνου έχει ένα

χαρακτηριστικό (σε αυτή την περίπτωση, την κλιμακωμένη τιμή κλεισίματος της μετοχής).

```
# Χτίσιμο του μοντέλου LSTM
model = Sequential()
model.add(Bidirectional(LSTM(50, return_sequences=True), input_shape=(time_step, 1)))
model.add(Bidirectional(LSTM(50)))
model.add(Dense(25))
model.add(Dense(1))
```

Εδώ φτιάχνουμε το νευρωνικό δίκτυο για το μοντέλο LSTM. Αρχικά αρχικοποιούμε ένα σειριακό μοντέλο. Έπειτα προσθέτουμε μία bidirectional LSTM στρώση με 50 κόμβους όπου δηλώνουμε ότι η έξοδος της στρώσης θα είναι μία ακολουθία. Προσθέτουμε άλλη μία bidirectional LSTM στρώση με 50 κόμβους. Προσθέτουμε μία Dense στρώση με 25 νευρώνες και μία τελική Dense στρώση με έναν νευρώνα για την πρόβλεψη ενός μοναδικού στοιχείου εξόδου.

```
# Εκπαίδευση του μοντέλου
adam_optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=adam_optimizer, loss='mean_squared_error')
model.fit(X_train, y_train, batch_size=1, epochs=5) # Εκπαίδευση για 5 εποχές
```

✓ 3m 21.6s

Epoch 1/5  
1141/1141 [=====] - 50s 36ms/step - loss: 0.0045  
Epoch 2/5  
1141/1141 [=====] - 39s 34ms/step - loss: 0.0017  
Epoch 3/5  
1141/1141 [=====] - 35s 31ms/step - loss: 0.0013  
Epoch 4/5  
1141/1141 [=====] - 37s 33ms/step - loss: 0.0011  
Epoch 5/5  
1141/1141 [=====] - 40s 35ms/step - loss: 9.1509e-04

Έπειτα εκπαιδεύουμε το LSTM μοντέλο. Δημιουργούμε έναν optimizer Adam με ένα προκαθορισμένο ρυθμό μάθησης (learning rate) 0.001. Κάνουμε compile το μοντέλο με τον προηγούμενος ορισμένο optimizer και χρησιμοποιώντας το mean squared error ως συνάρτηση απώλειας. Η συνάρτηση απώλειας χρησιμοποιείται για να εκτιμήσει το πόσο καλά το μοντέλο προβλέπει τα δεδομένα κατά τη διάρκεια της εκπαίδευσης. Τέλος εκπαιδεύουμε το μοντέλο στο σύνολο των δεδομένων για 5 εποχές (epochs). Κάθε εποχή είναι μια πλήρης διέλευση του συνόλου δεδομένων εκπαίδευσης, και η εκπαίδευση προσαρμόζει τα βάρη του δικτύου για να μειώσει την απώλεια. Επίσης βλέπουμε στο δικό μας παράδειγμα πόση ώρα έκανε για να κάνει την εκπαίδευση για κάθε epoch και το mean squared error για κάθε μία από αυτές.

```
# Αξιολόγηση του μοντέλου για την πρόβλεψη του μοντέλου στο train και test set
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
```

✓ 5.9s

36/36 [=====] - 5s 40ms/step  
9/9 [=====] - 0s 27ms/step

Έπειτα κάνουμε αξιολόγηση στο μοντέλο για την πρόβλεψη του μοντέλου στο train και test set. Με το batch size ίσο με 1 που του δώσαμε νωρίτερα το 36/36 που εμφανίζει κατά το output δείχνει ότι είχαμε 36 δείγματα στα input data. Αντίστοιχα για τα test το 9/9 δείχνει ότι είχαμε 9 δείγματα στα input data.

```
# Μετατροπή των δεδομένων σε αρχική μορφή
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
```

Μετατρέπουμε τα προβλεπόμενα δεδομένα ξανά στην αρχική μορφή που είχαν.

```
# Υπολογισμός του start index
test_start_index = len(scaled_data) - len(test_predict) - 1
```

Υπολογίζουμε τον δείκτη (index) από τον οποίο θα ξεκινήσει η εισαγωγή των προβλεπόμενων τιμών του test set στο συνολικό dataset.

```
trainPredictPlot = np.empty_like(scaled_data)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[time_step:len(train_predict) + time_step, :] = train_predict

testPredictPlot = np.empty_like(scaled_data)
testPredictPlot[:, :] = np.nan
testPredictPlot[test_start_index:test_start_index + len(test_predict), :] = test_predict
```

Έπειτα οργανώνουμε τα δεδομένα για τις προβλεπόμενες τιμές για τα train και test δεδομένα, προετοιμάζοντας τα για την οπτικοποίηση που θα ακολουθήσει σε επόμενο βήμα.

```
# Υπολογισμός του MAE, MSE και RMSE για το train set
train_mae = mean_absolute_error(y_train, scaler.inverse_transform(model.predict(X_train)))
train_mse = mean_squared_error(y_train, scaler.inverse_transform(model.predict(X_train)))
train_rmse = np.sqrt(train_mse)

✓ 1.8s

36/36 [=====] - 1s 24ms/step
36/36 [=====] - 1s 22ms/step

# Υπολογισμός του MAE, MSE και RMSE για το test set
test_mae = mean_absolute_error(y_test, scaler.inverse_transform(model.predict(X_test)))
test_mse = mean_squared_error(y_test, scaler.inverse_transform(model.predict(X_test)))
test_rmse = np.sqrt(test_mse)

✓ 0.7s

9/9 [=====] - 0s 27ms/step
9/9 [=====] - 0s 22ms/step
```

Υπολογίζουμε το mean absolute error (Μέσο Απόλυτο Σφάλμα), mean squared error (Μέσο Τετραγωνικό Σφάλμα) και το root mean squared error (Ρίζα του Μέσου Τετραγωνικού Σφάλματος).

```
print("Training Data Evaluation:")
print("MAE:", train_mae)
print("MSE:", train_mse)
print("RMSE:", train_rmse)

✓ 0.0s

Training Data Evaluation:
MAE: 124.71025471656876
MSE: 16754.227081689394
RMSE: 129.438120666554

print("\nTesting Data Evaluation:")
print("MAE:", test_mae)
print("MSE:", test_mse)
print("RMSE:", test_rmse)

✓ 0.0s

Testing Data Evaluation:
MAE: 115.57398267978468
MSE: 13687.041678582149
RMSE: 116.9916308057211
```

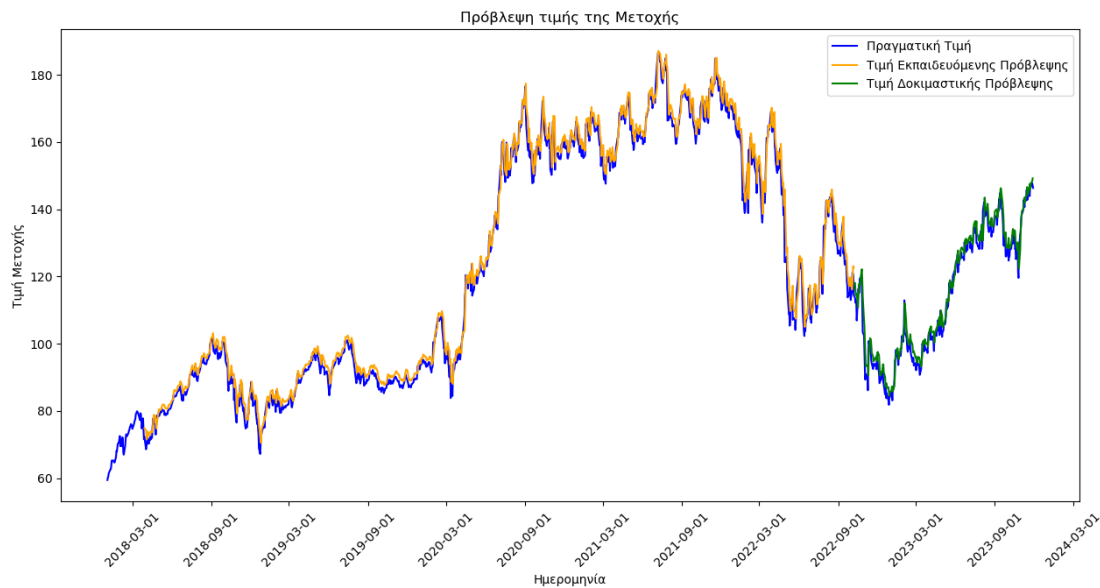
Εδώ βλέπουμε τα αποτελέσματα που πήραμε από τις μετρήσεις του μέσου απόλυτου σφάλματος, του μέσου τετραγωνικού σφάλματος και της ρίζας του μέσου τετραγωνικού σφάλματος. Όσο χαμηλότερες είναι αυτές οι τιμές, τόσο καλύτερη είναι η ακρίβεια του μοντέλου. Σε αυτή την περίπτωση, βλέπουμε ότι το μοντέλο έχει περίπου το ίδιο σφάλμα στα σετ εκπαίδευσης και δοκιμής, πράγμα που υποδηλώνει μια ομοιόμορφη απόδοση και ότι στο μοντέλο δεν έχουμε overfit στα δεδομένα εκπαίδευσης.

```
# Οπτικοποίηση των αποτελεσμάτων
plt.figure(figsize=(15,7))
plt.plot(df.index, scaler.inverse_transform(scaled_data), label='Πραγματική Τιμή', color='blue')
plt.plot(df.index, trainPredictPlot, label='Τιμή Εκπαιδευόμενης Πρόβλεψης', color='orange')
plt.plot(df.index, testPredictPlot, label='Τιμή Δοκιμαστικής Πρόβλεψης', color='green')

plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=6))

plt.xlabel('Ημερομηνία')
plt.ylabel('Τιμή Μετοχής')
plt.title('Πρόβλεψη τιμής της Μετοχής')
plt.legend()
plt.xticks(rotation=45)
plt.show()
```

Έπειτα κάνουμε οπτικοποίηση των δεδομένων σε ένα plot διάγραμμα και το output που παίρνουμε είναι το παρακάτω.



Από το διάγραμμα φαίνεται ότι οι προβλέψεις του μοντέλου ακολουθούν στενά την πραγματική κίνηση των τιμών της μετοχής, τόσο στα δεδομένα εκπαίδευσης όσο και στα δεδομένα δοκιμής. Αυτό υποδεικνύει ότι το μοντέλο έχει μάθει τα πρότυπα στα δεδομένα αρκετά καλά για να παράγει ακριβείς προβλέψεις. Η καλή συμπεριφορά του μοντέλου στο test set δείχνει ότι μπορεί να γενικεύσει καλά σε νέα δεδομένα που δεν έχει δει κατά την εκπαίδευση.

```
# Πρόβλεψη τιμών για τις επόμενες 5 μέρες
future_days = 5
future_predictions = []

# Παίρνουμε τα δεδομένα για την τελευταία μέρα
last_time_step_data = scaled_data[-time_step:]

# Μετατροπή των δεδομένων σε 3D μορφή για το LSTM
current_batch = last_time_step_data.reshape((1, time_step, 1))

# Πρόβλεψη τιμών για τις επόμενες 5 μέρες
for i in range(future_days):
    future_pred = model.predict(current_batch)[0]
    future_predictions.append(future_pred)
    current_batch = np.append(current_batch[:,1:,:], [[future_pred]], axis=1)

# Αντιστροφή της κανονικοποίησης
future_predictions = scaler.inverse_transform(future_predictions)

# Δημιουργία Pandas Series για τις μελλοντικές ημερομηνίες και τις προβλεπόμενες τιμές
future_dates = pd.date_range(start=df.index[-1], periods=future_days, freq='B') # 'B' denotes business day frequency
future_dates_series = pd.Series(future_dates, name='Date')
future_predictions_series = pd.Series(np.reshape(future_predictions, (future_days,)), name='Predicted Close')

# Συνδυασμός των dates και των προβλεπόμενων τιμών σε ένα DataFrame
future_predictions_df = pd.concat([future_dates_series, future_predictions_series], axis=1)

✓ 04s

1/1 [=====] - 0s 30ms/step
1/1 [=====] - 0s 32ms/step
1/1 [=====] - 0s 32ms/step
1/1 [=====] - 0s 32ms/step
1/1 [=====] - 0s 33ms/step
```

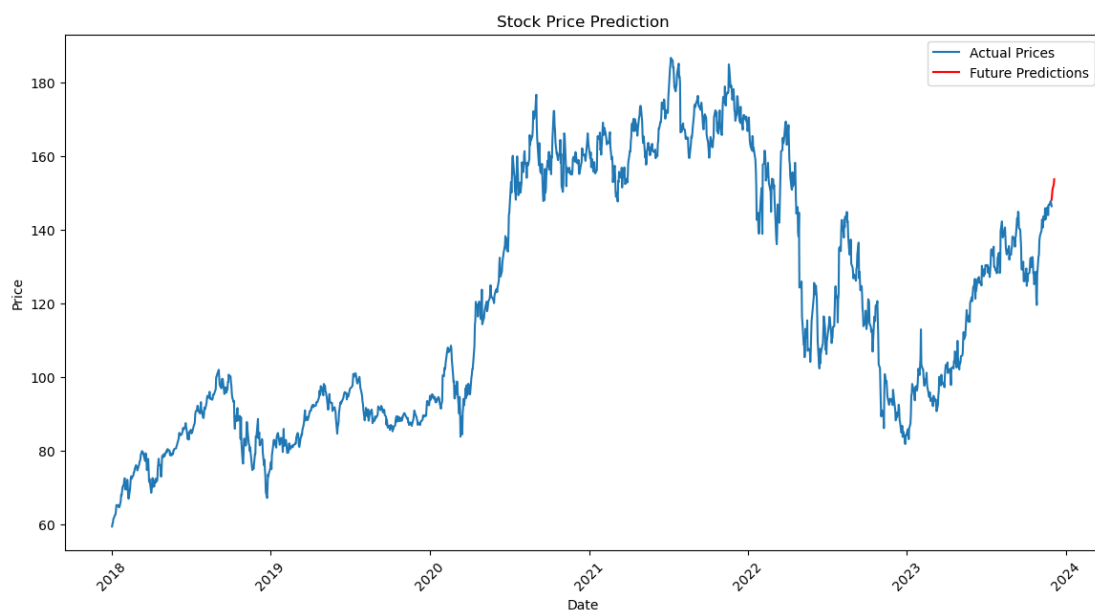
Έπειτα κάνουμε δοκιμή για να δούμε σε δεδομένα 5 νέων ημερών τι πρόβλεψη μπορεί να κάνει το εκπαιδευόμενο LSTM μοντέλο.

```
# Οπτικοποίηση των προβλεπόμενων τιμών
plt.figure(figsize=(14,7))
plt.plot(df['Close'], label='Actual Prices')

plt.plot(future_predictions_df.set_index('Date'), label='Future Predictions', color='red')

plt.title('Stock Price Prediction')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.xticks(rotation=45)
plt.show()
```

Τέλος κάνουμε και μία οπτικοποίηση των δεδομένων που προκύπτουν από την πρόβλεψη των δεδομένων για τις επόμενες 5 μέρες. Το διάγραμμα που προκύπτει είναι το ακόλουθο.



Από το παραπάνω διάγραμμα βλέπουμε ότι η πρόβλεψη δείχνει ότι η τιμή της μετοχής για την AMAZON θα αυξηθεί τις επόμενες 5 μέρες. Ως προς την πρόβλεψη της τιμής της μετοχής εκτός των άλλων πρέπει να λάβουμε υπόψη και άλλα συμβάντα στην αγορά καθώς επίσης είναι αναγκαία η θεμελιώδης και τεχνική ανάλυση προκειμένου να λάβουμε την απόφαση μας.

Για το μοντέλο των νευρωνικών δικτύων αυτό που παρατηρήθηκε είναι ότι κάθε φορά μπορεί να μας δώσει και κάποια διαφορετική πρόβλεψη ανάλογα με τις παραμετροποιήσεις που γίνονται κατά την εκπαίδευση του μοντέλου διαφοροποιώντας τις παραμέτρους που λαμβάνει το μοντέλο. Μετά από αρκετές προσπάθειες η τελική μορφή που πήρε φαίνεται να ανταποκρίνεται σε καλό βαθμό στην πρόβλεψη. Παρόλα αυτά με περισσότερες προσπάθειες και περισσότερα δεδομένα ίσως διαφοροποιηθεί και πάλι η πρόβλεψη και πρέπει να γίνουν παραμετροποιήσεις.

Παρακάτω παρέχονται κάποιες πληροφορίες ως προς τους νευρώνες, τις στρώσεις καθώς και την αρχιτεκτονική του νευρωνικού δικτύου.

**Νευρώνες:** Το μοντέλο σας περιέχει δύο διδιευθυντικές LSTM στρώσεις (bidirectional LSTM layers) με 50 νευρώνες η κάθε μία, μια πυκνή στρώση (dense layer) με 25 νευρώνες, και μια τελική πυκνή στρώση με έναν νευρώνα. Οι διδιευθυντικές LSTM στρώσεις αποτελούνται από δύο ξεχωριστές LSTM στρώσεις που διαδραματίζουν τα δεδομένα σε δύο κατευθύνσεις (προς τα εμπρός και προς τα πίσω), οπότε έχουμε συνολικά 100+100 νευρώνες για αυτές τις δύο στρώσεις.

**Στρώσεις:** Συνολικά, υπάρχουν τέσσερις στρώσεις, από τις οποίες οι δύο είναι διδιευθυντικές LSTM στρώσεις και οι δύο είναι πυκνές στρώσεις. Από αυτές, οι LSTM στρώσεις θεωρούνται ως κρυφές στρώσεις (hidden layers), ενώ η τελευταία πυκνή στρώση θεωρείται ως η φανερή στρώση (output layer) επειδή παράγει την τελική πρόβλεψη.

**Αρχιτεκτονική Μοντέλου:** Η αρχιτεκτονική του μοντέλου είναι σειριακή (Sequential). Σε αυτή την αρχιτεκτονική, οι στρώσεις συνδέονται αυστηρά η μία μετά την άλλη. Τα δεδομένα εισέρχονται από την είσοδο, διαδοχικά περνούν μέσα από κάθε στρώση και τελικά παράγεται η έξοδος.

## Παραμετροποιήσεις αλγορίθμων με την προσθήκη δεικτών

Κώδικας αλγορίθμου γραμμικής παλινδρόμησης με προσθήκη των δεικτών MACD, Bollinger Bands και RSI

Πάνω από κάθε εντολή υπάρχουν και τα απαραίτητα σχόλια ώστε να εξηγείται σε κάθε γραμμή τι κάνει ο αντίστοιχος κώδικας.

Επειδή μερικά κομμάτια του κώδικα είναι κοινά με τον αρχικό κώδικα, θα αναλυθούν και θα επεξηγηθούν μόνο τα επιπρόσθετα κομμάτια του κώδικα.

Παρακάτω φαίνεται ο κώδικας του αλγορίθμου Γραμμικής Παλινδρόμησης με την προσθήκη των τριών δεικτών (Input):

```
# Εισαγωγή απαραίτητων βιβλιοθηκών
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
```



```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
```

```
# Λήψη δεδομένων
symbol = 'AMZN'
start_date = '2018-11-30'
end_date = '2023-11-30'
df = yf.download(symbol, start=start_date, end=end_date)
```

```
# Καθαρισμός και επεξεργασία δεδομένων
df.dropna(inplace=True)
df['Return'] = df['Close'].pct_change()
df = df.reset_index()
df['Date'] = pd.to_datetime(df['Date'])
```

```
# Συνάρτηση για υπολογισμό RSI
def calculate_rsi(data, column_name, period=14):
    delta = data[column_name].diff(1)
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)
    avg_gain = gain.rolling(window=period, min_periods=1).mean()
    avg_loss = loss.rolling(window=period, min_periods=1).mean()
    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))
    return rsi
```

Η παραπάνω συνάρτηση υπολογισμού RSI (Relative Strength Index) χρησιμοποιείται για να υπολογίσει το RSI ενός συγκεκριμένου χαρακτηριστικού (π.χ., τιμές μετοχής) σε ένα DataFrame.

Η μεταβλητή `delta` γυπολογίζει τη διαφορά μεταξύ διαδοχικών τιμών του συγκεκριμένου χαρακτηριστικού (`column_name`). Η `diff(1)` σημαίνει ότι υπολογίζει τη διαφορά με την προηγούμενη τιμή.

Στην συνέχεια δημιουργεί δύο νέες στήλες, `gain` και `loss`, όπου η στήλη `gain` παίρνει τις θετικές τιμές του `delta` και η στήλη `loss` παίρνει τις αρνητικές τιμές του `delta`. Όταν το `delta` είναι μη θετικό, οι αντίστοιχες τιμές των `gain` και `loss` είναι μηδέν.

Μετά υπολογίζονται οι μέσοι όροι για περιόδους κέρδους και απώλειας. Το `window=period` καθορίζει το παράθυρο του κινούμενου μέσου, και το `min_periods=1` υποδεικνύει ότι το λιγότερο πολύ ένα δείγμα απαιτείται για να υπολογιστεί η μέση τιμή.

Ακόμη υπολογίζεται το σχετικό δυναμικό, που είναι ο λόγος του κινητού μέσου των κερδών προς το κινητό μέσο των απωλειών.

Τέλος υπολογίζεται το RSI με την χρήση του `rs` (σχετικού δυναμικού) και επιστρέφει την τιμή του.

```
# Συνάρτηση για υπολογισμό MACD
def calculate_macd(data, column_name, short_window=12, long_window=26,
signal_window=9):
    short_ema = data[column_name].ewm(span=short_window,
adjust=False).mean()
    long_ema = data[column_name].ewm(span=long_window,
adjust=False).mean()
    data['MACD'] = short_ema - long_ema
    data['Signal_Line'] = data['MACD'].ewm(
        span=signal_window, adjust=False).mean()
    return data
```

Αρχικά υπολογίζει το Short EMA για το συγκεκριμένο χαρακτηριστικό (`column_name`) χρησιμοποιώντας τον αριθμό περιόδων (`span=short_window`) για τον υπολογισμό του EMA.

Έπειτα υπολογίζει το Long EMA χρησιμοποιώντας τον αριθμό περιόδων (`span=long_window`).

Ο υπολογισμός των παραπάνω χρειάζεται για τον υπολογισμό του MACD που προκύπτει από την αφαίρεση του `short_ema` από το `long_ema`.

Μετά υπολογίζει τη γραμμή σήματος (`Signal Line`) ως EMA του MACD, χρησιμοποιώντας τον αριθμό περιόδων (`span=signal_window`) και τέλος επιστρέφει το DataFrame `data` που περιέχει τις νέες στήλες 'MACD' και 'Signal\_Line'.

```
# Συνάρτηση για υπολογισμό Bollinger Bands
def calculate_bollinger_bands(data, column_name, window=20, num_std=2):
    data['Rolling_Mean'] = data[column_name].rolling(
        window=window, min_periods=1).mean()
    data['Upper_Band'] = data['Rolling_Mean'] + \
```

```

        (data[column_name].rolling(window=window, min_periods=1).std()
* num_std)
        data['Lower_Band'] = data['Rolling_Mean'] - \
            (data[column_name].rolling(window=window, min_periods=1).std()
* num_std)
        return data

```

Αρχικά η συνάρτηση υπολογίζει τον κινούμενο μέσο όρο με το συγκεκριμένο χαρακτηριστικό (column\_name) χρησιμοποιώντας ένα παράθυρο κινητού μέσου μεγέθους. Το min\_periods=1 υποδεικνύει ότι τουλάχιστον ένα δείγμα απαιτείται για να υπολογιστεί ο μέσος όρος.

Μετά υπολογίζει το άνω Bollinger Band ως το άνω όριο του εύρους μεταξύ του κινούμενου μέσου όρου και του ανωτάτου Bollinger Band. Χρησιμοποιεί τον κινητό τυπικό απόκλιση (rolling(window=window, min\_periods=1).std()) πολλαπλασιασμένο με τον αριθμό των τυπικών αποκλίσεων (num\_std).

Τέλος υπολογίζει το κάτω Bollinger Band ως το κάτω όριο του εύρους μεταξύ του κινούμενου μέσου όρου και του κατωτάτου Bollinger Band και επιστρέφει το DataFrame data που περιέχει τις νέες στήλες 'Rolling\_Mean', 'Upper\_Band' και 'Lower\_Band'.

```

# Προσθήκη δεικτών RSI, MACD και Bollinger Bands
df['RSI'] = calculate_rsi(df, 'Close')
df = calculate_macd(df, 'Close')
df = calculate_bollinger_bands(df, 'Close')

```

Σε αυτό το σημείο του κώδικα, γίνεται ο υπολογισμός και η προσθήκη των 3 δεικτών στο DataFrame (df).

```

# Δημιουργία μεταβλητών για το μοντέλο μηχανικής μάθησης
df['Previous_Close'] = df['Close'].shift(1)
df.dropna(inplace=True)

```

```

# Διαχωρισμός σε σετ εκπαίδευσης και δοκιμής
X = df[['Date', 'Previous_Close']]
y = df['Close']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, shuffle=False)

```

```
# Εκπαίδευση μοντέλου
model = LinearRegression()
model.fit(X_train.drop('Date', axis=1), y_train)
```

```
# Πρόβλεψη τιμών
predictions = model.predict(X_test.drop('Date', axis=1))
```

```
# Αξιολόγηση μοντέλου
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
print(f'MSE: {mse}')
print(f'MAE: {mae}')
```

Τα αποτελέσματα αυτών των 2 μετρικών είναι τα εξής:

MSE: 6.091115402222176

MAE: 1.8708078991466957

### 3. MSE (Mean Squared Error):

- Η τιμή του MSE που παρέχετε (6.091115402222176) είναι το μέσο τετραγωνικό σφάλμα μεταξύ των πραγματικών και προβλεπόμενων τιμών.
- Όσο μικρότερη είναι η τιμή του MSE, τόσο καλύτερη είναι η απόδοση του μοντέλου.
- Σε αυτήν την περίπτωση, μια τιμή 6.091 δείχνει ότι υπάρχει κάποια διασπορά στα τετραγωνικά σφάλματα.

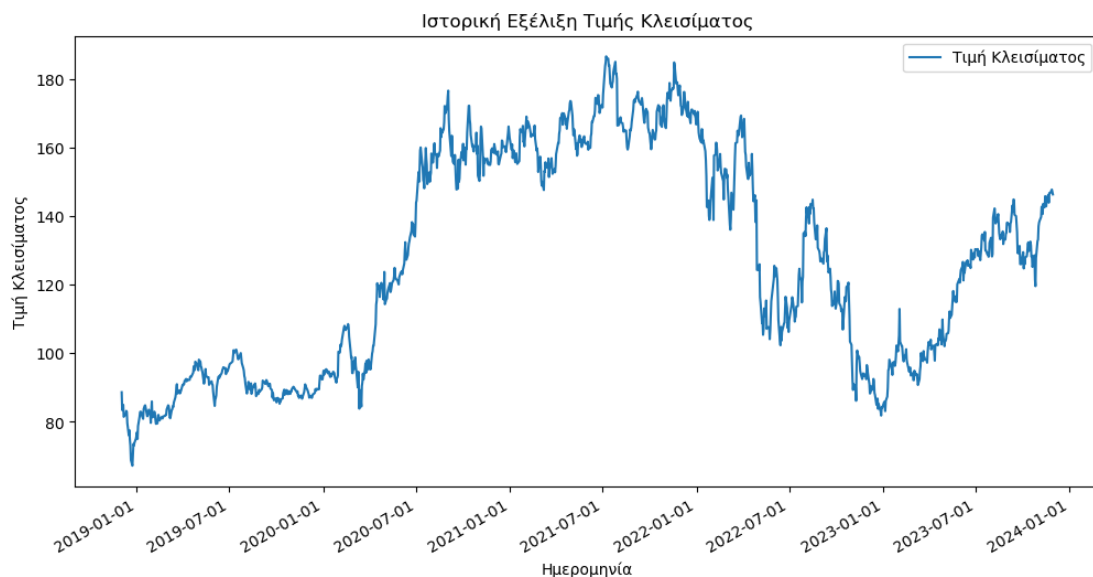
### 4. MAE (Mean Absolute Error):

- Η τιμή του MAE που παρέχετε (1.8708078991466957) είναι το μέσο απόλυτο σφάλμα μεταξύ των πραγματικών και προβλεπόμενων τιμών.
- Όσο μικρότερη είναι η τιμή του MAE, τόσο καλύτερη είναι η απόδοση του μοντέλου.
- Σε αυτήν την περίπτωση, μια τιμή 1.870 δείχνει ότι υπάρχει κάποιο απόλυτο σφάλμα, αλλά η απόκλιση δεν είναι πολύ μεγάλη.

Συνολικά, κρίνοντας από τις τιμές που παρέχονται, μπορούμε να διαπιστώσουμε ότι το μοντέλο μαζί με την προσθήκη των 3 δεικτών έχει ελαφρώς ανεβασμένες τιμές MSE και MAE. Συνεπώς και πάλι έχει κάποιο βαθμό σφάλματος στις προβλέψεις του, αλλά εξακολουθεί να παρουσιάζει μια ανοχή.

```
# Σχεδίαση ιστορικής εξέλιξης τιμής, δεικτών και προβλεπόμενων τιμών
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Close'], label='Τιμή Κλεισίματος')
plt.title('Ιστορική Εξέλιξη Τιμής Κλεισίματος')
plt.xlabel('Ημερομηνία')
plt.ylabel('Τιμή Κλεισίματος')
plt.legend()
plt.gcf().autofmt_xdate()
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(bymonth=(1, 7)))
plt.show()
```

Το γράφημα που προκύπτει είναι το εξής(output):



Το παραπάνω διάγραμμα απεικονίζει την πορεία της μετοχής Amazon την τελευταία 5ετία.

```
# RSI
plt.subplot(3, 1, 1)
plt.plot(df['Date'], df['RSI'], label='RSI', color='purple')
plt.axhline(y=70, color='r', linestyle='--', label='Overbought (70)')
plt.axhline(y=30, color='g', linestyle='--', label='Oversold (30)')
plt.title('Relative Strength Index (RSI)')
plt.legend()

# MACD
```

```

plt.subplot(3, 1, 2)
plt.plot(df['Date'], df['MACD'], label='MACD', color='blue')
plt.plot(df['Date'], df['Signal_Line'], label='Signal Line',
color='orange')
plt.title('MACD (Moving Average Convergence Divergence)')
plt.legend()

# Bollinger Bands
plt.subplot(3, 1, 3)
plt.plot(df['Date'], df['Close'], label='Τιμή Κλεισίματος')
plt.plot(df['Date'], df['Upper_Band'], label='Upper Band', color='red')
plt.plot(df['Date'], df['Lower_Band'], label='Lower Band',
color='green')
plt.fill_between(df['Date'], df['Upper_Band'], df['Lower_Band'],
color='lightgray', alpha=0.4, label='Bollinger Bands')
plt.title('Bollinger Bands')
plt.xlabel('Ημερομηνία')
plt.ylabel('Τιμή Κλεισίματος')
plt.legend()

plt.tight_layout()
plt.show()

```

Η λειτουργία του παραπάνω κώδικα είναι η οπτικοποίηση των 3 δεικτών.

- Για τον RSI:

Χρησιμοποιεί τη συνάρτηση `plt.subplot(3, 1, 1)` για να ορίσει ένα υποδιάγραμμα στο οποίο θα σχεδιαστεί ο RSI. Στη συνέχεια, σχεδιάζει τη γραφική παράσταση του RSI και προσθέτει οριζόντιες γραμμές στα επίπεδα 70 και 30 ως σημεία Overbought και Oversold, αντίστοιχα. Τέλος, προσθέτει τίτλο και λεζάντα.

- Για τον MACD:

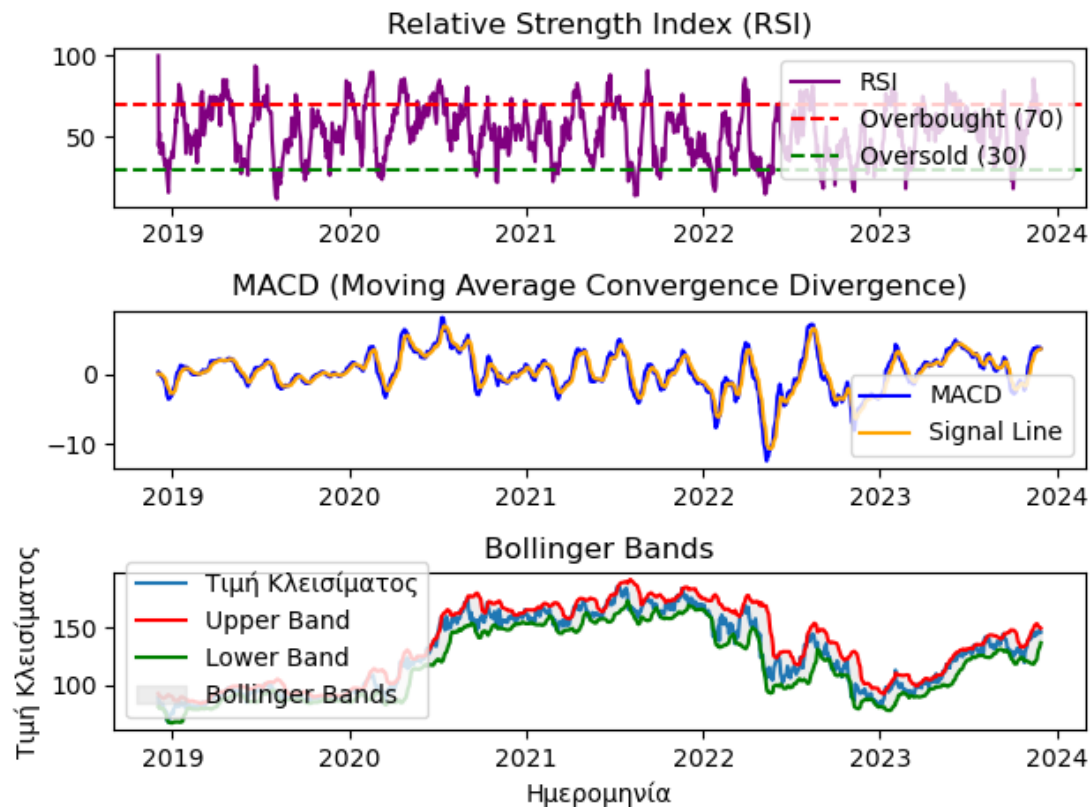
Χρησιμοποιεί τη συνάρτηση `plt.subplot(3, 1, 2)` για να ορίσει ένα υποδιάγραμμα στο οποίο θα σχεδιαστεί το MACD και η γραμμή σήματος. Στη συνέχεια, σχεδιάζει τις γραφικές παραστάσεις για το MACD και τη γραμμή σήματος, και προσθέτει τίτλο και λεζάντα.

- Για τον Bollinger Bands:

Χρησιμοποιεί τη συνάρτηση `plt.subplot(3, 1, 3)` για να ορίσει ένα υποδιάγραμμα στο οποίο θα σχεδιαστούν τα Bollinger Bands. Στη συνέχεια, σχεδιάζει τις γραφικές παραστάσεις για την τιμή κλεισίματος, τα Bollinger Bands, και προσθέτει μια γεμάτη περιοχή μεταξύ του Upper και Lower Band. Τέλος, προσθέτει τίτλο, άξονες x και y, και λεζάντα.

Τέλος το `plt.tight_layout()` χρησιμοποιείται για να διασφαλίσει ότι τα διαγράμματα δεν επικαλύπτονται και το `plt.show()` εμφανίζει το τελικό γράφημα.

Συνεπώς το διάγραμμα που προκύπτει είναι το εξής(output):



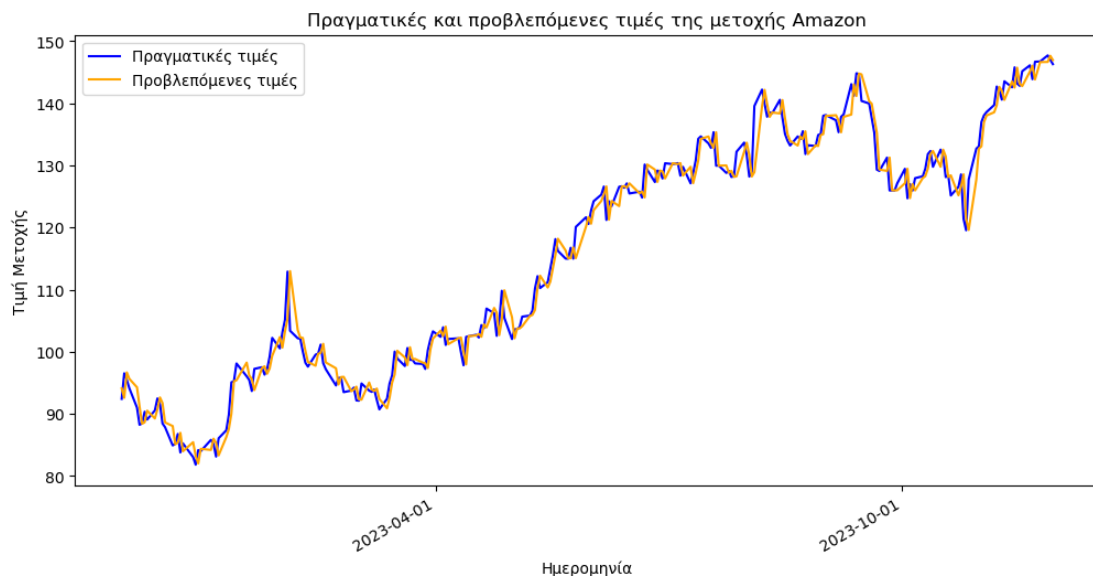
Αυτά τα 3 διαγράμματα απεικονίζουν τις κινήσεις των δεικτών RSI, MACD και Bollinger Bands. Αυτοί οι 3 δείκτες είναι τεχνικοί και χρηματοοικονομικοί δείκτες και χρειάζονται εξειδικευμένη γνώση για να μπορεί κάποιους να τους διαβάσει και να κατανοήσει τι απεικονίζουν.

Συνεπώς η προσθήκη τους στον συγκεκριμένο αλγόριθμο, σίγουρα βοηθάει τον χρήστη να έχει επιπλέον γνώση για να βγάλει την άποψη του σχετικά με το που θα κινηθεί η συγκεκριμένη μετοχή, ωστόσο η αλήθεια είναι πως ταυτόχρονα απαιτεί και εξειδικευμένη γνώση.

```
# Εμφάνιση πραγματικών και προβλεπόμενων τιμών  
plt.figure(figsize=(12, 6))
```

```
plt.plot(X_test['Date'], y_test, color='blue', label='Πραγματικές τιμές')
plt.plot(X_test['Date'], predictions, color='orange', label='Προβλεπόμενες τιμές')
plt.title('Πραγματικές και προβλεπόμενες τιμές της μετοχής Amazon')
plt.xlabel('Ημερομηνία')
plt.ylabel('Τιμή Μετοχής')
plt.legend()
plt.gcf().autofmt_xdate()
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=6))

plt.show()
```



Αυτό που παρατηρείται είναι πως οι προβλεπόμενες τιμές που έδωσε ο αλγόριθμος σε σχέση με τις πραγματικές είναι κοντά, αλλά η απόκλιση που υπάρχει είναι εμφανές ότι δεν κρύβεται. Συμπερασματικά θα μπορούσε να αποτελεί ένα μέτριο δείγμα για πρόβλεψη της συγκεκριμένης μετοχής με περιθώριο βελτιστοποίησης.

Κώδικας αλγορίθμου δέντρων απόφασης με προσθήκη των δεικτών MACD, Bollinger Bands και RSI.

Πάνω από κάθε εντολή υπάρχουν και τα απαραίτητα σχόλια ώστε να εξηγείται σε κάθε γραμμή τι κάνει ο αντίστοιχος κώδικας.



Επειδή μερικά κομμάτια του κώδικα είναι κοινά με τον αρχικό κώδικα του αλγορίθμου δέντρων απόφασης, θα αναλυθούν και θα επεξηγηθούν μόνο τα επιπρόσθετα κομμάτια του κώδικα.

Παρακάτω φαίνεται ο κώδικας του αλγορίθμου Δέντρων Απόφασης με την προσθήκη των τριών δεικτών (Input):

```
import yfinance as yf
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
```

```
# Λήψη δεδομένων
symbol = 'AMZN'
start_date = '2018-11-30'
end_date = '2023-11-30'
df = yf.download(symbol, start=start_date, end=end_date)
```

```
# Καθαρισμός και επεξεργασία δεδομένων
df.dropna(inplace=True)
df['Return'] = df['Close'].pct_change()
df = df.reset_index()
df['Date'] = pd.to_datetime(df['Date'])
```

```
# Συνάρτηση για υπολογισμό RSI
def calculate_rsi(data, column_name, period=14):
    delta = data[column_name].diff(1)
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)
    avg_gain = gain.rolling(window=period, min_periods=1).mean()
    avg_loss = loss.rolling(window=period, min_periods=1).mean()
    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))
    return rsi
```

Η παραπάνω συνάρτηση υπολογισμού RSI (Relative Strength Index) χρησιμοποιείται για να υπολογίσει το RSI ενός συγκεκριμένου χαρακτηριστικού (π.χ., τιμές μετοχής) σε ένα DataFrame.

Η μεταβλητή delta υπολογίζει τη διαφορά μεταξύ διαδοχικών τιμών του συγκεκριμένου χαρακτηριστικού (column\_name). Η diff(1) σημαίνει ότι υπολογίζει τη διαφορά με την προηγούμενη τιμή.

Στην συνέχεια δημιουργεί δύο νέες στήλες, gain και loss, όπου η στήλη gain παίρνει τις θετικές τιμές του delta και η στήλη loss παίρνει τις αρνητικές τιμές του delta. Όταν το delta είναι μη θετικό, οι αντίστοιχες τιμές των gain και loss είναι μηδέν.

Μετά υπολογίζονται οι μέσοι όροι για περιόδους κέρδους και απώλειας. Το window=period καθορίζει το παράθυρο του κινούμενου μέσου, και το min\_periods=1 υποδεικνύει ότι το λιγότερο πολύ ένα δείγμα απαιτείται για να υπολογιστεί η μέση τιμή.

```
# Συνάρτηση για υπολογισμό MACD
def calculate_macd(data, column_name, short_window=12, long_window=26,
signal_window=9):
    short_ema = data[column_name].ewm(span=short_window,
adjust=False).mean()
    long_ema = data[column_name].ewm(span=long_window,
adjust=False).mean()
    data['MACD'] = short_ema - long_ema
    data['Signal_Line'] = data['MACD'].ewm(
        span=signal_window, adjust=False).mean()
    return data
```

Αρχικά υπολογίζει το Short EMA για το συγκεκριμένο χαρακτηριστικό (column\_name) χρησιμοποιώντας τον αριθμό περιόδων (span=short\_window) για τον υπολογισμό του EMA.

Έπειτα υπολογίζει το Long EMA χρησιμοποιώντας τον αριθμό περιόδων (span=long\_window).

Ο υπολογισμός των παραπάνω χρειάζεται για τον υπολογισμό του MACD που προκύπτει από την αφαίρεση του short\_ema από το long\_ema.

Μετά υπολογίζει τη γραμμή σήματος (Signal Line) ως EMA του MACD, χρησιμοποιώντας τον αριθμό περιόδων (span=signal\_window) και τέλος επιστρέφει το DataFrame data που περιέχει τις νέες στήλες 'MACD' και 'Signal\_Line'.

```
# Συνάρτηση για υπολογισμό Bollinger Bands
def calculate_bollinger_bands(data, column_name, window=20, num_std=2):
    data['Rolling_Mean'] = data[column_name].rolling(
        window=window, min_periods=1).mean()
    data['Upper_Band'] = data['Rolling_Mean'] + \
        (data[column_name].rolling(window=window, min_periods=1).std()
* num_std)
```

```

data['Lower_Band'] = data['Rolling_Mean'] - \
    (data[column_name].rolling(window=window, min_periods=1).std()
 * num_std)
return data

```

Αρχικά η συνάρτηση υπολογίζει τον κινούμενο μέσο όρο με το συγκεκριμένο χαρακτηριστικό (column\_name) χρησιμοποιώντας ένα παράθυρο κινητού μέσου μεγέθους. Το min\_periods=1 υποδεικνύει ότι τουλάχιστον ένα δείγμα απαιτείται για να υπολογιστεί ο μέσος όρος.

Μετά υπολογίζει το άνω Bollinger Band ως το άνω όριο του εύρους μεταξύ του κινούμενου μέσου όρου και του ανωτάτου Bollinger Band. Χρησιμοποιεί τον κινητό τυπικό απόκλιση (rolling(window=window, min\_periods=1).std()) πολλαπλασιασμένο με τον αριθμό των τυπικών αποκλίσεων (num\_std).

Τέλος υπολογίζει το κάτω Bollinger Band ως το κάτω όριο του εύρους μεταξύ του κινούμενου μέσου όρου και του κατωτάτου Bollinger Band και επιστρέφει το DataFrame data που περιέχει τις νέες στήλες 'Rolling\_Mean', 'Upper\_Band' και 'Lower\_Band'.

```

# Προσθήκη δεικτών RSI, MACD και Bollinger Bands
df['RSI'] = calculate_rsi(df, 'Close')
df = calculate_macd(df, 'Close')
df = calculate_bollinger_bands(df, 'Close')

```

Σε αυτό το σημείο του κώδικα, γίνεται ο υπολογισμός και η προσθήκη των 3 δεικτών στο DataFrame (df).

```

# Δημιουργία μεταβλητών για το μοντέλο μηχανικής μάθησης
df['Previous_Close'] = df['Close'].shift(1)
df.dropna(inplace=True)

```

```

# Διαχωρισμός σε σετ εκπαίδευσης και δοκιμής
X = df[['Date', 'Previous_Close']]
y = df['Close']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, shuffle=False)

# Δημιουργία και Εκπαίδευση μοντέλου
model = DecisionTreeRegressor()

```

```
model.fit(X_train.drop('Date', axis=1), y_train)

# Πρόβλεψη τιμών
predictions = model.predict(X_test.drop('Date', axis=1))
```

Σ αυτό το κομμάτι οι μεταβλητές  $x$  και  $y$  εκπαιδεύονται και δοκιμάζονται για τον έλεγχο του μοντέλου σε ένα τεστάρισμα μεγέθους 20%.

Επιπλέον γίνεται η δημιουργία του μοντέλου του αλγορίθμου των δέντρων απόφασης καθώς και η εκπαίδευση του και στην συνέχεια υπολογίζεται η πρόβλεψη που δίνει το μοντέλο.

```
# Αξιολόγηση μοντέλου
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error: {mse}')
```

Η αξιολόγηση του μοντέλου για τον αλγόριθμο δέντρων απόφασης θα γίνει μόνο με την μετρική MSE και χωρίς την MAE. Ο λόγος είναι ότι ο κώδικας έβγαζε κάποιο σφάλμα στην εκτέλεση της μετρικής MAE, με αποτέλεσμα να μείνει εκτός κώδικα.

Το Mean Squared Error (MSE) είναι μια μετρική που μετρά τη μέση τετραγωνική απόκλιση μεταξύ των πραγματικών και προβλεπόμενων τιμών. Τα ορίσματα της στην συγκεκριμένη περίπτωση είναι τα εξής:

$y_{\text{test}}$ : Οι πραγματικές τιμές της μεταβλητής κατάστασης από το σετ δοκιμής.

$\text{predictions}$ : Οι προβλεπόμενες τιμές της μεταβλητής κατάστασης από το μοντέλο.

Τα αποτελέσματα της είναι:

MSE: 17.033557350343067

#### ▪ Mean Squared Error (MSE):

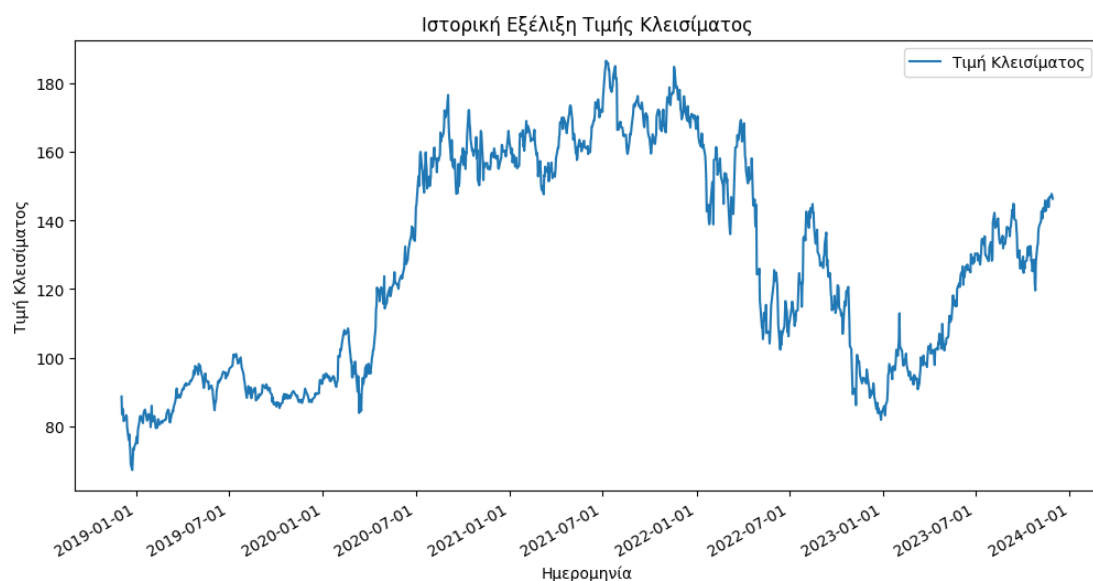
- Η τιμή MSE που παρέχετε (17.033557350343067) είναι το μέσο τετραγωνικό σφάλμα μεταξύ των πραγματικών και προβλεπόμενων τιμών.
- Όσο μικρότερη είναι η τιμή του MSE, τόσο καλύτερη είναι η απόδοση του μοντέλου.

- Σε αυτήν την περίπτωση, μια τιμή 17.033 δείχνει ότι οι προβλέψεις του μοντέλου έχουν μεγάλο επίπεδο τετραγωνικού σφάλματος και αυτό δεν μπορεί να θεωρηθεί ως αποδεκτό, αφού η τιμή είναι πολύ μακριά τόσο από το 0 όσο και από την τιμή του MSE (αρχική τιμή: 0.0745) στον αρχικό αλγόριθμο δέντρων απόφασης που δεν εφαρμόσαμε τους δείκτες.

Συνολικά, η τιμή αυτή δεν μπορεί να εκτιμηθεί ως αποδεκτή, καθώς αποκλίνει παρά πολύ από το μοντέλο και με αυτό τον τρόπο το θεωρεί και ως αναξιόπιστο.

```
# Σχεδίαση ιστορικής εξέλιξης τιμής, δεικτών και προβλεπόμενων τιμών
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Close'], label='Τιμή Κλεισίματος')
plt.title('Ιστορική Εξέλιξη Τιμής Κλεισίματος')
plt.xlabel('Ημερομηνία')
plt.ylabel('Τιμή Κλεισίματος')
plt.legend()
plt.gcf().autofmt_xdate()
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(bymonth=(1, 7)))
plt.show()
```

Το διάγραμμα που προκύπτει είναι το εξής(output):



```

# RSI
plt.subplot(3, 1, 1)
plt.plot(df['Date'], df['RSI'], label='RSI', color='purple')
plt.axhline(y=70, color='r', linestyle='--', label='Overbought (70)')
plt.axhline(y=30, color='g', linestyle='--', label='Oversold (30)')
plt.title('Relative Strength Index (RSI)')
plt.legend()

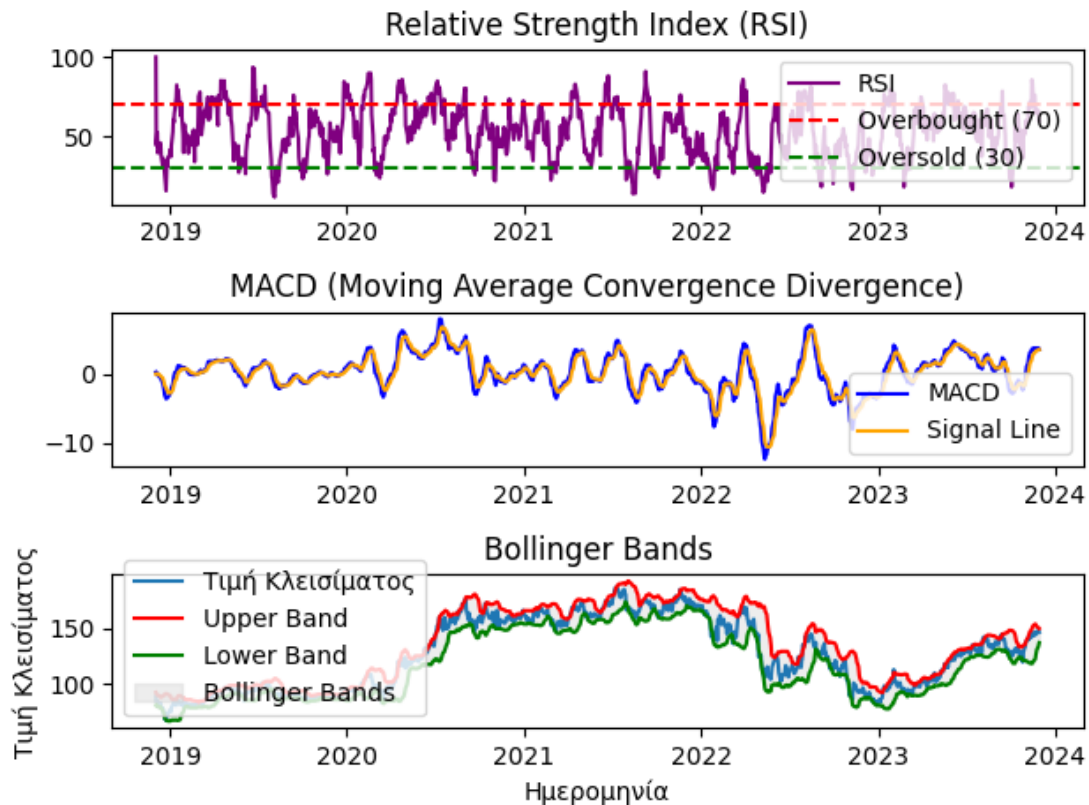
# MACD
plt.subplot(3, 1, 2)
plt.plot(df['Date'], df['MACD'], label='MACD', color='blue')
plt.plot(df['Date'], df['Signal_Line'], label='Signal Line',
color='orange')
plt.title('MACD (Moving Average Convergence Divergence)')
plt.legend()

# Bollinger Bands
plt.subplot(3, 1, 3)
plt.plot(df['Date'], df['Close'], label='Τιμή Κλεισίματος')
plt.plot(df['Date'], df['Upper_Band'], label='Upper Band', color='red')
plt.plot(df['Date'], df['Lower_Band'], label='Lower Band',
color='green')
plt.fill_between(df['Date'], df['Upper_Band'], df['Lower_Band'],
color='lightgray', alpha=0.4, label='Bollinger Bands')
plt.title('Bollinger Bands')
plt.xlabel('Ημερομηνία')
plt.ylabel('Τιμή Κλεισίματος')
plt.legend()

plt.tight_layout()
plt.show()

```

Τα διαγράμματα που προκύπτουν για τους 3 δείκτες είναι τα εξής (output) είναι τα εξής:



Αυτά τα 3 διαγράμματα απεικονίζουν τις κινήσεις των δεικτών RSI, MACD και Bollinger Bands. Αυτοί οι 3 δείκτες είναι τεχνικοί και χρηματοοικονομικοί δείκτες και χρειάζονται εξειδικευμένη γνώση για να μπορεί κάποιους να τους διαβάσει και να κατανοήσει τι απεικονίζουν.

```
# Εμφάνιση πραγματικών και προβλεπόμενων τιμών
plt.figure(figsize=(12, 6))
plt.plot(X_test['Date'], y_test, color='blue', label='Πραγματικές τιμές')
plt.plot(X_test['Date'], predictions, color='orange', label='Προβλεπόμενες τιμές')
plt.title('Πραγματικές και προβλεπόμενες τιμές της μετοχής Amazon')
plt.xlabel('Ημερομηνία')
plt.ylabel('Τιμή Μετοχής')
```

```
plt.legend()
plt.gcf().autofmt_xdate()
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=6))

plt.show()
```



Αυτό που παρατηρείται είναι πως οι προβλεπόμενες τιμές που έδωσε ο αλγόριθμος σε σχέση με τις πραγματικές σε κάποια σημεία είναι κοντά, σε κάποια άλλα έχουν μία μέτρια απόκλιση και σε κάποια άλλα έχουν μεγάλη απόκλιση. Συνεπώς δεν μπορεί να αποτελεί ένα αξιόπιστο δείγμα για πρόβλεψη της συγκεκριμένης μετοχής και επιτρέπει περαιτέρω τροποποιήσεις για να βελτιστοποιήσει τις προβλέψεις του.



## Πηγές

Για τον Αλγόριθμο Γραμμικής Παλινδρόμησης:

- [yfinance Documentation](#)
- [Pandas Documentation](#)
- [Matplotlib Documentation](#)
- [Seaborn Documentation](#)
- [scikit-learn Documentation](#)
- [Time Series Analysis with Pandas](#)
- [Introduction to Linear Regression in Python](#)
- [Machine Learning Mastery](#)
- [Towards Data Science](#)

Για τον Αλγόριθμο Δέντρων Απόφασης:

- [Scikit-Learn Decision Trees](#)
- [Decision Tree Regression in Python](#)
- [A Guide to Decision Trees for Beginners](#)
- [Decision Trees: A Visual Introduction for Beginners](#)
- [Understanding Decision Trees for Classification \(Python\)](#)

Για τον Αλγόριθμο Νευρωνικών Δικτύων:

- Πως λειτουργούν τα νευρωνικά δίκτυα: <https://www.explainthatstuff.com/introduction-to-neural-networks.html>
- Πώς λειτουργεί ο MinMaxScaler: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>
- Κατανόηση του LSTM Network: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- Sequential model: [https://keras.io/guides/sequential\\_model/](https://keras.io/guides/sequential_model/)
- LSTM και πώς να φτιάξουμε τα δεδομένα για αυτό: <https://machinelearningmastery.com/reshape-input-data-long-short-term-memory-networks-keras/>

- LSTM για time series: <https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/>
- Adam optimizer: [https://www.tensorflow.org/api\\_docs/python/tf/keras/optimizers/Adam](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam)
- Model training με keras: [https://keras.io/api/models/model\\_training\\_apis/](https://keras.io/api/models/model_training_apis/)

Για τους δείκτες:

- [Using Python and RSI to Generate Trading Signals](#)
- [Algorithmic Trading with MACD in Python](#)
- [How To Calculate Bollinger Bands Of A Stock With Python](#)
- [Bollinger Bands with Python](#)

Επιπλέον Πηγές:

- Διαφορές Τεχνητής Νοημοσύνης με Μηχανική Μάθηση: [Artificial Intelligence \(AI\) vs. Machine Learning | Columbia AI](#)
- Τι είναι η Μηχανική Μάθηση: [Machine learning, explained | MIT Sloan](#)
- Γλώσσες Προγραμματισμού για την Τεχνητή Νοημοσύνη και Μηχανική Μάθηση: [Best Programming Language for AI Development in 2023 \[Updated\] \(hackr.io\)](#) , <https://www.codecademy.com/resources/blog/machine-learning-programming-languages/> , <https://bootcamp.berkeley.edu/blog/ai-programming-languages/>
- Python: [Welcome to Python.org](#)
- Anaconda: [Anaconda | The World's Most Popular Data Science Platform](#)
- Conda: [Getting started with conda — conda 23.11.1.dev10 documentation](#)
- Jupyter Notebook: <https://jupyter.org/>
- Google Colab: <https://colab.research.google.com/>
- Numpy: [NumPy](#)
- Tensorflow: <https://www.tensorflow.org/>