

תיק פרויקט לעבודת גמר (5 יח"ל) - למידת מכונה

שם בית הספר: מקיף מופת בסמת - חיפה

סמל מוסד: 344101

שם התלמיד: איתי אביעד

מספר ת"ז: 214891665

נושא העבודה: למידה במסחר במניות

שם פרויקט: traider

תוכן עניינים

2.....	תוכן עניינים
3.....	מבוא
5.....	איסוף הכנה וניתוח הנתונים
11.....	בנייה ואימון המודל
22.....	תוצאות האימון
29.....	יישום
32.....	מדריך למפתח
47.....	תיעוד מדריך למשתמש
49.....	רפלקציה
50.....	ביבליוגרפיה

מבוא

הרקע לפרויקט:

הפרויקט החל מתוך הצורך למצוא פתרון חדשני ומותאם אישית למסחר אוטומטי בשווקים פיננסיים. החלטתי לערוך מחקר מעמיק שיכלול גם טכניקות מתקדמות של למידת מכונה ורשתות נוירונים, במטרה לפתח מערכת שמסוגלת לנתח בזמן אמת נתונים פיננסיים ולהגיב בהתאם.

תהליך המחקר:

במסגרת המחקר המקדים, עברתי מספר שלבים:

1. **סקירה ספרותית:** בחנתי מחקרים ומודלים קיימים בתחום המסחר האוטומטי.
2. **ניתוח אתגרי השוק:** זיהיתי אתגרים מרכזיים כמו התנודתיות הגבוהה, חוסר הוודאות והצורך בהחלטות מהירות ומדויקות.
3. **ניסויים ובדיקות:** ערכתי השוואות בין הפתרונות השונים במטרה לזהות את היתרונות והחסרונות של כל גישה.

אתגרים מרכזיים:

- **דינאמיות וחוסר ודאות:** השוק הפיננסי משתנה במהירות, מה שמעלה את רמת הקושי בבניית מערכת שמצליחה לחזות ולהגיב בזמן אמת.
- **התמודדות עם נתונים רעשניים:** נתוני השוק כוללים רעש והפרעות שמקשות על זיהוי המגמות האמיתיות.
- **ביצועים ומהירות:** הצורך בקבלת החלטות מהירות מחייב את המערכת להיות יעילה מאוד מבחינת עיבוד נתונים בזמן אמת.

הפתרונות שנבחנו במסגרת המחקר המקדים:

במהלך המחקר בדקתי מספר גישות טכניות:

- **שילוב אינדיקטורים טכניים:** ניסיתי לשלב אינדיקטורים שונים (כמו ממוצעים נעים) כחלק מקלט המודל, במטרה לשפר את יכולת הזיהוי של דפוסי שוק.
- **התאמת פרמטרים ואופטימיזציה:** ערכתי ניסויים בשינוי פרמטרים שונים כמו שיעור הלמידה, קצב ההתפחחות של האקספלורציה ועוד, כדי להגיע לאיזון מיטבי בין חקר וניצול המידע.

בסופו של דבר, המחקר המקדים סייע לי לזהות גישה יעילה לפיתוח מערכת מסחר אוטומטית שתוכל להתמודד עם האתגרים המורכבים של השוק, באופן שמתאים לאופי הפרויקט.

הספריות בהן השתמשתי:

```
1 import yfinance as yf
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 import torch
6 import torch.nn as nn
7 import torch.optim as optim
8 from sklearn.preprocessing import MinMaxScaler
9 from sklearn.metrics import mean_absolute_error, mean_squared_error
10 from sklearn.model_selection import train_test_split
11 from torch.utils.data import DataLoader, TensorDataset
12 from IPython.display import clear_output
```

איסוף הכנה וניתוח הנתונים

1. תיאור מבנה הנתונים Dataset

מקור הנתונים:

הנתונים נאספו ממקורות אמינים בתחום הפיננסי - Yahoo Finance. בחרתי במקור זה בזכות הזמינות של נתונים היסטוריים איכותיים ועדכניים, אשר כוללים מידע חשוב למסחר כגון מחירים, תאריכים ונפח מסחר. את הנתונים הורדתי באמצעות קטע קוד מיוחד אשר מוריד את הנתונים באמצעות ספריית yfinance בפייטון.

```
1 # Download stock data
2 df = yf.download(TICKER, period="max", interval="1d", rounding=True, prepost=True)
```

כך הורדתי את נתוני המחירים (כאשר הם מעוגלים לרמה מסוימת - 2 ספרות אחרי הנקודה העשרונית). הקובץ מכיל את המחירים מיום ההשקה של המניה המבוקשת לשוק ההון ועד היום הנוכחי.

מבנה הנתונים:

הקובץ שנאסף הוא קובץ CSV הכולל מספר עמודות. כל שורה בקובץ הפלט מייצגת מחירי פתיחה, סגירה, נמוך ביותר ביום, גבוה ביותר ביום, תאריך ונפח מסחר באותו היום.

- Date - תאריך (יומי -יום, חודש, שנה).
- Close - מחיר הסגירה של המניה בסוף יום המסחר.
- High - המחיר הגבוה ביותר של המניה ביום המסחר.
- Low - המחיר הנמוך ביותר של המניה ביום המסחר.
- Open - מחיר הפתיחה של המניה בתחילת יום המסחר.
- Volume - נפח מסחר באותו היום (מספר המניות שעברו בעלים באותו היום).
- EMA - ממוצע נע אקספוננציאלי. זהו סוג מסוים של ממוצע נע שנותן יותר משקל למחירים עדכניים מאשר למחירי עבר (חושב באמצעות בקוד).

Price	Close	High	Low	Open	Volume	EMA
Ticker	AMZN	AMZN	AMZN	AMZN	AMZN	
Date						
1997-05-15	0.10	0.12	0.10	0.12	1443120000	0.100000
1997-05-16	0.09	0.10	0.09	0.10	294000000	0.096667
1997-05-19	0.09	0.09	0.08	0.09	122136000	0.094444
1997-05-20	0.08	0.09	0.08	0.09	109344000	0.089630
1997-05-21	0.07	0.08	0.07	0.08	377064000	0.083086
...
2025-05-16	205.59	206.85	204.37	206.85	43318500	204.941589
2025-05-19	206.16	206.62	201.26	201.65	34314800	205.347726
2025-05-20	204.07	205.59	202.65	204.63	29470400	204.921817
2025-05-21	201.12	203.46	200.06	201.61	42460900	203.654545
2025-05-22	203.10	205.76	200.16	201.38	38830200	203.469697

7050 rows × 6 columns

2. תיאור וניתוח הנתונים הגולמיים

ניתוח ראשוני:

בשלב האיסוף, נבחנו הנתונים הגולמיים.

ניקיון הנתונים: בוצע תהליך ניקוי שבו טופלו ערכים חסרים, והובטח שהנתונים מסודרים לפי סדר זמן ולפי שמות עמודות נכונים.



אתגרים שנתקלתי בהם:

- **התאמה לאינדיקטורים טכניים:** ראיתי צורך לשלב אינדיקטורים טכניים, ולכן היה עלי לבצע פעולות חישוב בקוד לשם שיפור המידע הזמין למודל. לכן הייתי צריך לכתוב קטע קוד שיחשב את האינדיקטור הטכני של ממוצע נע אקספוננציאלי (לפי 5 ימי המסחר הקודמים), ויוסיף אותם בעמודה הרלוונטית ל-Dataset לאימון. חישוב ה-EMA:



Formula for Exponential Moving Average (EMA)

לפי הנוסחה:

$$EMA_{\text{Today}} = \left(\text{Value}_{\text{Today}} * \left(\frac{\text{Smoothing}}{1 + \text{Days}} \right) \right) + EMA_{\text{Yesterday}} * \left(1 - \left(\frac{\text{Smoothing}}{1 + \text{Days}} \right) \right)$$

where:

EMA = Exponential moving average

While there are many possible choices for the smoothing factor, the most common choice is:

- Smoothing = 2

That gives the most recent observation more weight. If the smoothing factor is increased, more recent observations have more influence on the EMA.

3. תיאור תהליך הכנת ה Dataset לאימון

נירמול המידע

ישנו צורך בנירמול ה-Dataset. זאת משום שמחירי המניות ב-Dataset נתונים במחירים אמיתיים, שיכולים להיות שונים בהרבה אחד מהשני. לכן יש צורך בנרמול המחירים לערכים בין 0 ל-1, ובכך נקבל טווח מחירים יותר אמין לשם אימון המודל. קוד פייטון אשר לוקח את מחיר הסגירה של המניה בכל יום (Close) ואת הממוצע הנע האקספוננציאלי (EMA):

A code block with a dark background and light-colored text. It contains four lines of Python code. The first line is a comment. The second line defines a list of features. The third line creates a MinMaxScaler object. The fourth line uses the scaler to fit and transform the data for the specified features.

```
1 # Normalize features
2 features = ["Close", "EMA"]
3 scaler = MinMaxScaler()
4 scaled_data = scaler.fit_transform(df[features])
```


יצירת תוויות לשם אימון

לשם אימון המודל, יש צורך בתוויות (labels) מתאימות בהתאם לרצון החיזוי מהמודל. כלומר, בשביל שהמודל יחזה את מחיר המניה למחרת, יש צורך לאמן אותו על תוויות אימון אשר מתארות את מחיר המניה ביום הבא (שידוע משום שאנו מאמנים את המודל על מחירי עבר של המניות, ולא על מחירים עתידיים).

כמו כן, יש צורך לחלק את המחירים הנתונים ב-dataset ל-sequence. זאת כדי לאפשר למודל להתאמן על מחירים של המניה במשך מספר ימים לפני התאריך הרצוי לחיזוי, ולא רק על פי היום האחרון לפני התאריך האחרון לחיזוי.

לשם כך כתבתי פונקציה ב-python אשר מקבלת את המידע (הפרמטר data) ואת האורך הרצוי של כל sequence (הפרמטר seq_length). הפונקציה מחלקת את המידע הנתון ל-sequence באורך הרצוי, ובעבור כל אחד מהם נותנת תווית מתאימה לאימון, כלומר, מחיר המניה היום בעוקב ליום האחרון ב-sequence.

בשביל המחירים השתמשתי במחיר ה-Close של המניה באותו היום, משום שזהו המחיר העדכני ביותר של המניה בסוף כל יום.

```
1 # Prepare sequences
2 def create_sequences(data, seq_length):
3     xs, ys = [], []
4     for i in range(seq_length, len(data)):
5         xs.append(data[i - seq_length:i])
6         ys.append(data[i, features.index("Close")]) # Predicting the 'Close' price
7     return np.array(xs), np.array(ys)
```

```
1 X, y = create_sequences(scaled_data, SEQ_LENGTH)
```

חלוקת הנתונים לנתוני אימון ונתוני בדיקה (train, test)

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1-TRAIN_TEST_RATIO, shuffle=False)
2
```

טעינת הנתונים למבני נתונים של pytorch

לשם אימון המודל, יש להעביר את הנתונים ל-Tensor של Pytorch בהם Pytorch יכול להשתמש ביעילות בתהליך אימון המודל:

```
1 # Convert to PyTorch tensors
2 X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
3 y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
4 X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
5 y_test_tensor = torch.tensor(y_test, dtype=torch.float32)
6
7 # Create DataLoader instances
8 train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
9 test_dataset = TensorDataset(X_test_tensor, y_test_tensor)
10 train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
11 test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)
```

בנייה ואימון המודל

תיאור המודל

המודל בו בחרתי להשתמש בפרויקט הוא מסוג LSTM - Long Short-Term Memory.

LSTM הוא סוג של רשת עצבית חוזרת (RNN) שמסוגלת ללמוד ולהתחשב בקשרים ארוכי טווח בתוך סדרות זמן. הוא שומר מידע חשוב לאורך רצף זמן ומסנן מידע פחות רלוונטי, מה שהופך אותו למועיל במיוחד בחיזוי נתונים כמו מחירי מניות או טקסט.

LSTM מתאים במיוחד לפרויקט שלי כי הוא מסוגל ללמוד דפוסים לאורך זמן ברצפים של מחירי מניות - כמו מגמות, תנודתיות, או תגובה לאירועים. בניגוד לרגרסיה ליניארית שמתחשבת רק בקלט הנוכחי, LSTM מנצל מידע היסטורי (למשל 60 הימים האחרונים) כדי לבצע תחזית מדויקת יותר של המחיר הבא.

המודל בקוד

```
1 # LSTM Model Definition
2 class LSTMModel(nn.Module):
3     def __init__(self, input_size, hidden_size, num_layers, dropout):
4         super(LSTMModel, self).__init__()
5         self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True, dropout=dropout)
6         self.fc = nn.Linear(hidden_size, 1)
7
8     def forward(self, x):
9         out, _ = self.lstm(x)
10        out = out[:, -1, :] # Take the output from the last time step
11        out = self.fc(out)
12        return out
```

:Class

מחלקה שיורשת מ-`nn.Module`, מחלקת הבסיס לכל המודלים ב-Pytorch.

:__init__

פונקציית האתחול של המודל.

פרמטרים:

- `input_size` – כמה פיצ'רים נכנסים בכל צעד זמן (למשל: Close ו-EMA, אז 2).
- `hidden_size` – מספר הנוירונים בשכבת ה-LSTM הפנימית.
- `num_layers` – כמה שכבות LSTM מוערמות אחת על השנייה.
- `dropout` – אחוז דרופאאוט. זוהי טכניקה למניעת למידת יתר (overfitting) ברשתות נוירונים. במהלך האימון, בכל איטרציה, היא "מכבה" באקראי אחוז מסוים מהנוירונים כך שלא יתרמו לחישוב. זה מאלץ את הרשת ללמוד ייצוגים כלליים יותר ולא להסתמך על נוירונים מסוימים בלבד.

שכבות:

שכבת LSTM שמקבלת רצפים ומחזירה את הפלט בכל צעד זמן.

שכבת Fully Connected שמקבלת את הפלט האחרון של ה-LSTM (כלומר מהיום האחרון ברצף), וממירה אותו לניבוי בודד: המחיר הצפוי למחר.

:forward

```
1 out, _ = self.lstm(x)
2 out = out[:, -1, :] # Take the output from the last time step
3 out = self.fc(out)
4 return out
```

שורה 1:

- מעבירה את הקלט לשכבת ה-LSTM.
- out: מכיל את הפלט מכל צעד זמן.
- _: מכיל את מצבי ה-hidden וה-cell האחרונים (לא משתמשים בהם כאן).

שורה 2:

- לוקחים את הפלט של צעד הזמן האחרון בלבד מתוך כל הרצף (למשל, היום ה-60).

שורה 3:

- מעבירים את הפלט דרך השכבה הליניארית כדי לקבל את מחיר הסגירה החזוי.

אימון המודל:

אתחול

```
1 # Initialize
2 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
3 model = LSTMModel(input_size=len(features), hidden_size=HIDDEN_SIZE, num_layers=NUM_LAYERS, dropout=DROPOUT)
4 model = model.to(device)
5 criterion = nn.MSELoss()
6 optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
7
```

בקטע הזה מגדירים את המכשיר להרצה (GPU אם קיים, אחרת CPU), יוצרים מופע של המודל LSTMModel, מגדירים פונקציית Loss מסוג MSE (שגיאת ממוצע ריבועית), ו-optimizer מסוג Adam לעדכון המשקלים של המודל.

פונקציית השגיאה - MSELoss:

פונקציה זו מחשבת את ממוצע ריבועי ההפרשים בין התחזיות לערכים האמיתיים, וכך מודדת עד כמה התחזיות מדויקות. היא מענישה שגיאות גדולות יותר באופן חזק יותר, ועוזרת לשפר את איכות התחזית.

ייעול ההתכנסות - Adam:

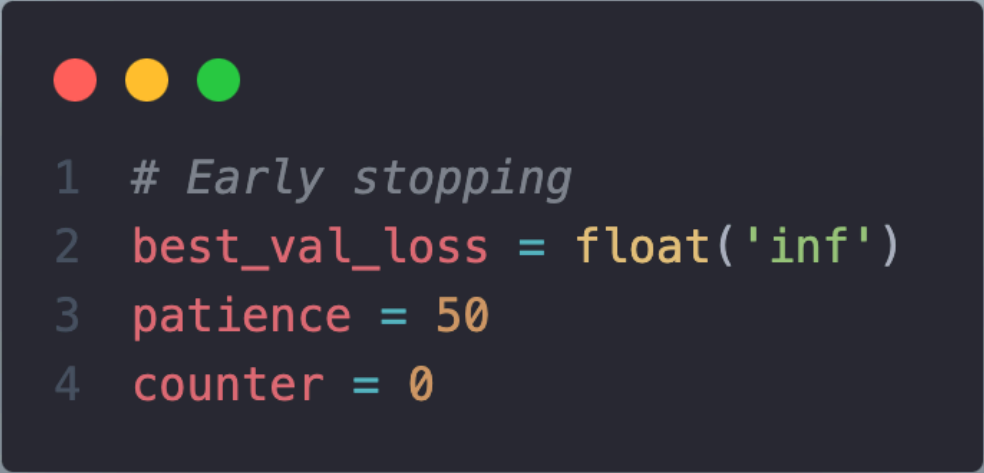
אופטימיזר Adam מתאים את קצב הלמידה באופן דינמי לכל פרמטר, מה שמאפשר למודל להתכנס מהר יותר וללא צורך בכיוון ידני מדויק. הוא מבוסס על שמירה של ממוצעים נעים של גרדיאנטים ומרובעים שלהם כדי לשפר את העדכונים.

יצירת לוגים להיסטוריית Loss

```
1 # Logging
2 train_loss_log = []
3 val_loss_log = []
```

שומרים את ערכי ה-loss לאורך ה-epoch עבור אימון וולידציה כדי שאפשר יהיה לנתח ולצייר את גרף הביצועים מאוחר יותר.

הגדרת Early Stopping



```
1 # Early stopping
2 best_val_loss = float('inf')
3 patience = 50
4 counter = 0
```

משתמשים ב-Early Stopping כדי לעצור את האימון אם המודל לא משתפר לאורך זמן. שומרים את האיבוד הטוב ביותר בוולידציה ומונים כמה epochs עברו ללא שיפור.

לולאת האימון לכל Epoch

```
1 for epoch in range(EPOCHS):
2     model.train()
3     batch_train_losses = []
4
5     for X_batch, y_batch in train_loader:
6         X_batch, y_batch = X_batch.to(device), y_batch.to(device)
7         optimizer.zero_grad()
8         outputs = model(X_batch).squeeze()
9         loss = criterion(outputs, y_batch)
10        loss.backward()
11        optimizer.step()
12        batch_train_losses.append(loss.item())
```

פותרים לולאה שרצה על פני מספר אפוקים (סיבובי אימון). כל פעם מאפסים את רשימת ה-Losses של האימון.

עוברים על כל באץ' (מקבץ דוגמאות) ב-`train_loader`, מחשבים ניבוי, מחשבים שגיאה, מבצעים Backpropagation, ומעדכנים את המשקלים. כל שגיאה נרשמת ברשימת ה-Losses.


```

1 # Validation
2 model.eval()
3 batch_val_losses = []
4 with torch.no_grad():
5     for X_batch, y_batch in test_loader:
6         X_batch, y_batch = X_batch.to(device), y_batch.to(device)
7         outputs = model(X_batch).squeeze()
8         loss = criterion(outputs, y_batch)
9         batch_val_losses.append(loss.item())

```

מעבירים את המודל למצב הערכה (eval) כדי לנטרל Dropout וכו', ואז מעריכים את הביצועים על סט הבדיקה תוך מניעת שמירת גרדיאנטים.

חישוב Losses

```

1 avg_train_loss = np.mean(batch_train_losses)
2 avg_val_loss = np.mean(batch_val_losses)
3 train_loss_log.append(avg_train_loss)
4 val_loss_log.append(avg_val_loss)
5
6 clear_output(wait=True)
7 print(f"Epoch [{epoch+1}/{EPOCHS}] | Train Loss: {avg_train_loss:.6f} | Val Loss: {avg_val_loss:.6f}")

```

מחשבים את ה-Loss הממוצע ל-EPOCH הנוכחי, שומרים אותם בלוג, ומדפיסים את התוצאה למסך.

Early Stopping

```
1 # Early stopping
2 if avg_val_loss < best_val_loss:
3     best_val_loss = avg_val_loss
4     counter = 0
5     torch.save(model.state_dict(), "best_model.pth")
6 else:
7     counter += 1
8     if counter >= patience:
9         print("Early stopping triggered.")
10        break
```

אם המודל השתפר (ה-Loss על הוולידציה ירד), שומרים את המודל כ"הכי טוב" ומאפסים את המונה. אם לא - מגדילים את המונה. אם עברו יותר מדי אפוקים בלי שיפור (בהתאם ל-patience), עוצרים את האימון.

כל קוד האימון ביחד

```
1 for epoch in range(EPOCHS):
2     model.train()
3     batch_train_losses = []
4
5     for X_batch, y_batch in train_loader:
6         X_batch, y_batch = X_batch.to(device), y_batch.to(device)
7         optimizer.zero_grad()
8         outputs = model(X_batch).squeeze()
9         loss = criterion(outputs, y_batch)
10        loss.backward()
11        optimizer.step()
12        batch_train_losses.append(loss.item())
13
14    # Validation
15    model.eval()
16    batch_val_losses = []
17    with torch.no_grad():
18        for X_batch, y_batch in test_loader:
19            X_batch, y_batch = X_batch.to(device), y_batch.to(device)
20            outputs = model(X_batch).squeeze()
21            loss = criterion(outputs, y_batch)
22            batch_val_losses.append(loss.item())
23
24    avg_train_loss = np.mean(batch_train_losses)
25    avg_val_loss = np.mean(batch_val_losses)
26    train_loss_log.append(avg_train_loss)
27    val_loss_log.append(avg_val_loss)
28
29    clear_output(wait=True)
30    print(f"Epoch [{epoch+1}/{EPOCHS}] | Train Loss: {avg_train_loss:.6f} | Val Loss: {avg_val_loss:.6f}")
31
32    # Early stopping
33    if avg_val_loss < best_val_loss:
34        best_val_loss = avg_val_loss
35        counter = 0
36        torch.save(model.state_dict(), "best_model.pth")
37    else:
38        counter += 1
39        if counter >= patience:
40            print("Early stopping triggered.")
41            break
```

:Hyper Parameters

```
1 # Configuration
2 TICKER = "AMZN" # Stock ticker symbol
3 SEQ_LENGTH = 60 # Sequence length for LSTM input
4 BATCH_SIZE = 32 # Batch size for training
5 EPOCHS = 100 # Number of training epochs
6 LEARNING_RATE = 0.002 # Learning rate for optimizer
7 HIDDEN_SIZE = 128
8 TRAIN_TEST_RATIO = 0.8 # Ratio of training to testing data
9 DROPOUT = 0.01
10 NUM_LAYERS = 3
```

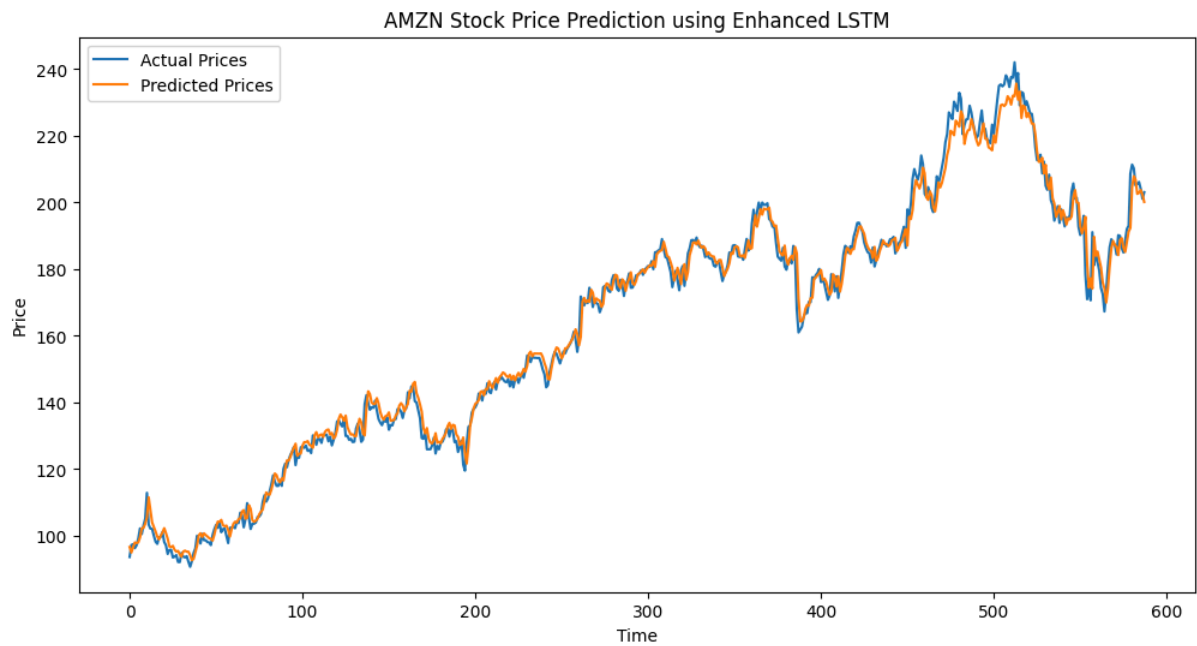
- TICKER - סימול המניה שנשתמש בה.
- SEQ_LENGTH - מספר הימים ההיסטוריים שכל דוגמה מזינה ל-LSTM.
- BATCH_SIZE - מספר הדוגמאות שכל באץ' יכיל באימון.
- EPOCHS - מספר האפוקים הכולל לאימון המודל.
- LEARNING_RATE - קצב הלמידה של האופטימיזר.
- HIDDEN_SIZE - מספר הנירונים בשכבה הנסתרת של ה-LSTM.
- TRAIN_TEST_RATIO - היחס בין כמות הנתונים לאימון ובדיקה בחלוקת הנתונים.
- DROPOUT - אחוז הדרופאאוט (השמטת ניורונים באימון) כדי למנוע overfitting.
- NUM_LAYERS - מספר שכבות LSTM במודל.

שיפורים שנעשו ב-Hyper Parameters

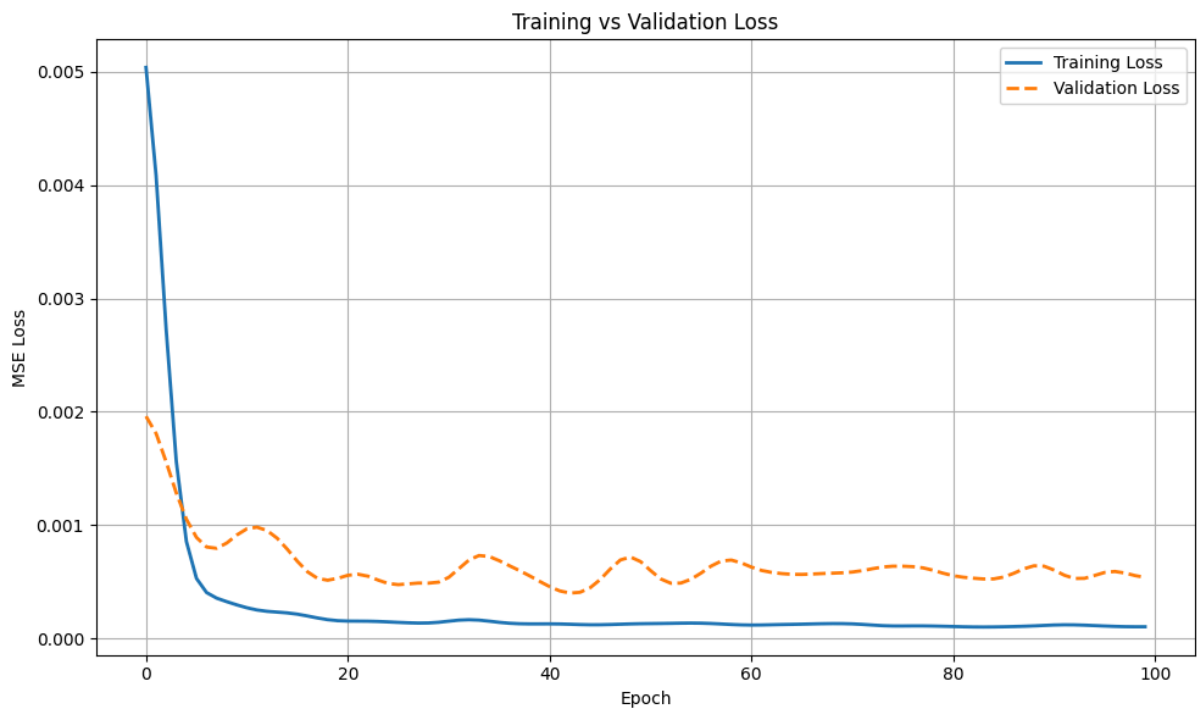
- **SEQ_LENGTH** - הייתי צריך למצוא ערך שיתחשב במספיק מחירי עבר כדי שידע לתת ערך מתאים על פי המחירים, אך לא יותר מידי מחירי עבר שכן מחירים מאוד רחוקים לא בהכרח משפיעים על המחיר לניבוי. 60 ימים נתן תוצאה טובה.
- **BATCH_SIZE** - חיפשתי ערך מתאים ו-32 נמצא כערך שנותן תוצאה טובה. ערכים גדולים/קטנים יותר נתנו תוצאות פחות טובות.
- **EPOCHS** - אני מאפשר למודל לבצע 100 איפוקים, וזה מספיק כפי שניתן לראות בגרף ה-Loss של האימון המופיע בהמשך.
- **LEARNING_RATE** - יש צורך לתת ערך מספיק קטן כדי שה-GD יצליח למצוא נקודת מינימום מקומית טובה מספיק. בו זמנית הערך צריך להיות קטן במידה כך שאינו קטן מידי וגורם לאימון להיות ארוך מידי.
- **HIDDEN_SIZE** - לפי השימושים ב-LSTM באינטרנט, מצאתי שהערך 128 הוא ערך נפוץ וטוב. מדדתי גם בעצמי עם ערכים גדולים וקטנים יותר והערך הזה נתן תוצאות טובות.
- **TRAIN_TEST_RATIO** - ערך נפוץ מאוד הוא 0.8 ובו השתמשתי לאימון ובדיקה של תוצאות המודל.
- **DROPOUT** - גם כן ערך נפוץ הוא 0.01 והוא נתן תוצאות טובות לפי הבדיקות שביצעתי.
- **NUM_LAYERS** - הערך 3 (כלומר 3 שכבות LSTM) גרמו למודל להיות מורכב במידה, לא יותר מידי ולא פחות מידי.

תוצאות האימון

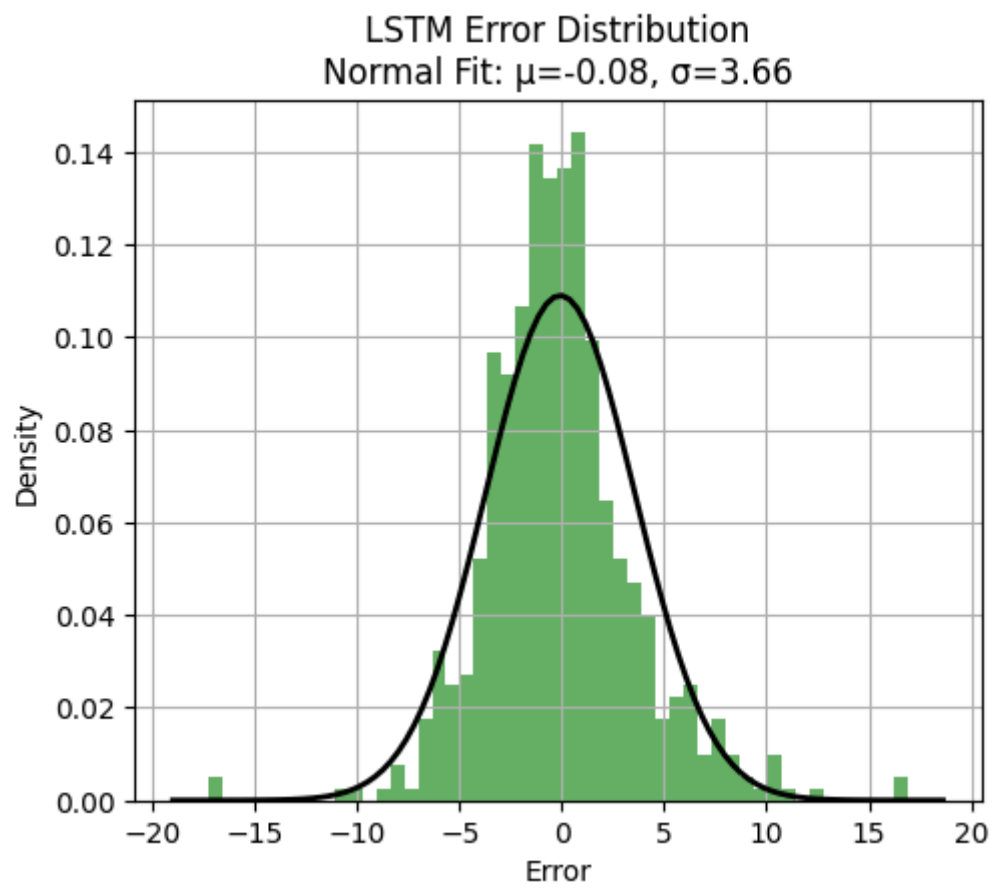
תוצאות חיזוי המודל על נתוני ה-test:



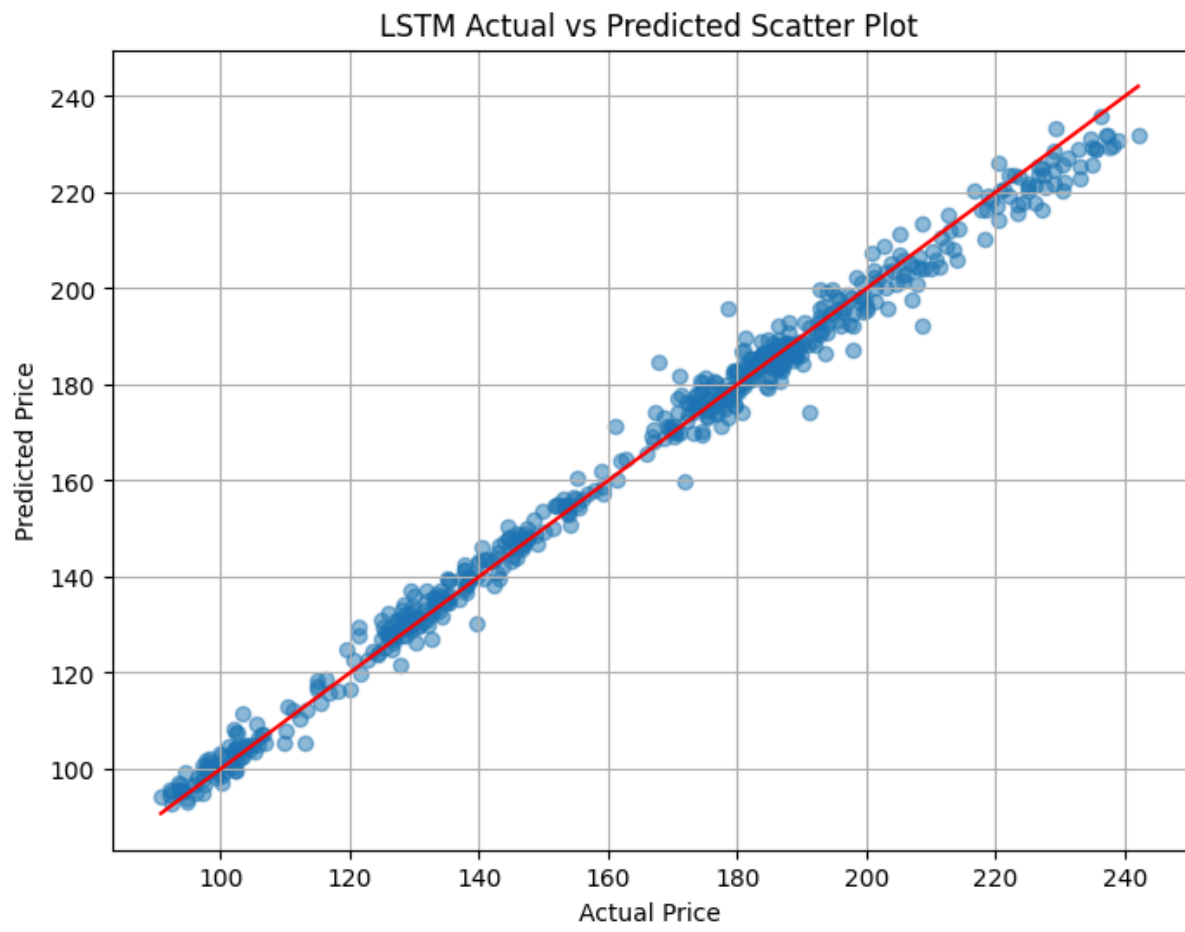
ה-Loss לפי מספר ה-Epoch על נתוני האימון ועל נתוני הוולידציה:



התפלגות שגיאה עבור תוצאות הבדיקה מהמודל:



מחיר אמיתי לעומת מחיר ניבוי לפי המודל:



הערכת המודל לפי מודל פשוט יותר - Linear Regression
בשביל לבדוק את תוצאות המודל מזווית מבט שונה, החלטתי לאמן מודל נוסף באמצעות Linear Regression, מודל שאמור להיות פשוט יותר. כלומר, באמצעות אותו המידע בדיוק, לאמן עוד מודל מסוג שונה אשר ינסה לבצע את אותן פעולות ניבוי כמו מודל ה-LSTM שבניתי.
בחרתי במודל Linear Regression משום שהוא פשוט יותר ונפוץ מאוד במודלים פשוטים של ניבוי, וגם בפרוייקטים המנסים לנבא מחירי מניות.

הספריות הנחוצות הן אותן הספריות כמו מקודם, בתוספת:

```
1 from sklearn.linear_model import LinearRegression
```

הכנת ה-Dataset לאימון:

```
1 # Use only the last time-step of each sequence as features
2 X_lr_train = X_train[:, -1, :] # shape: (n_samples, n_features)
3 X_lr_test  = X_test[:, -1, :]
```

אימון המודל:



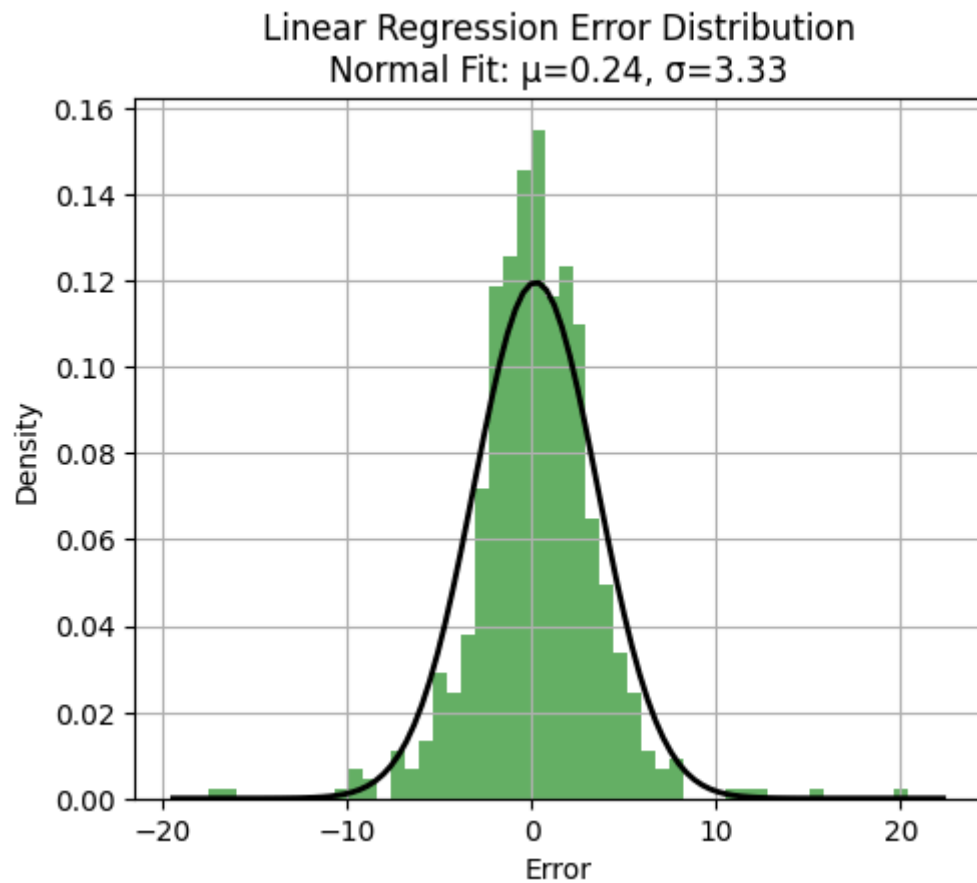
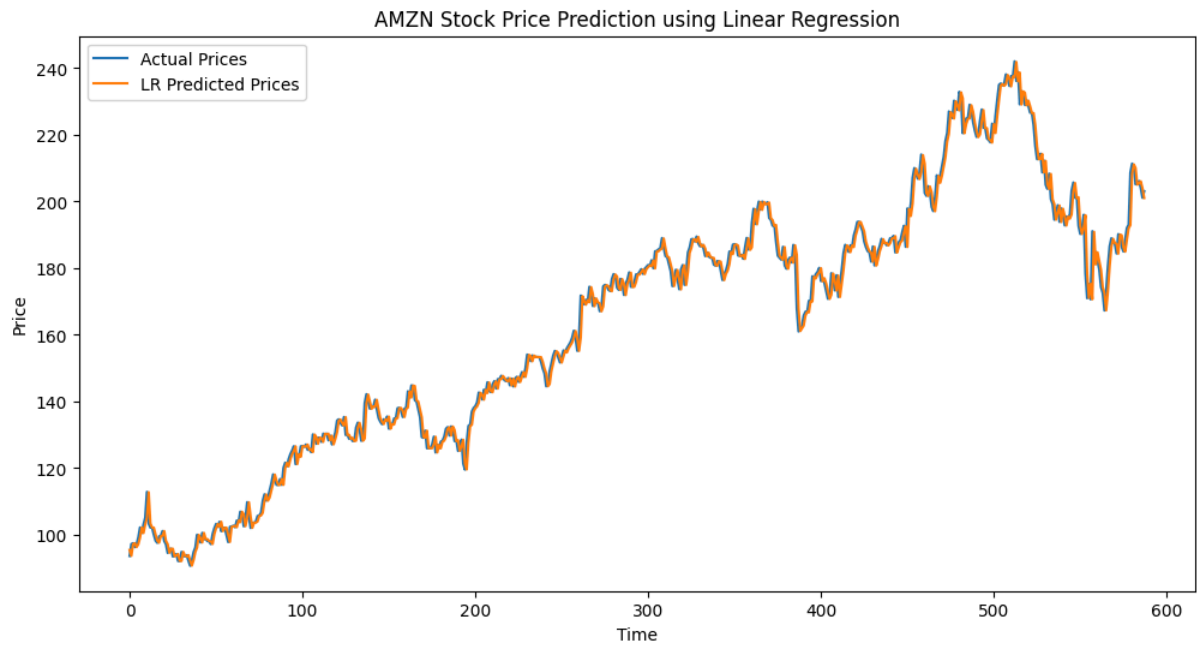
```
1 lr_model = LinearRegression()
2 lr_model.fit(X_lr_train, y_train)
```

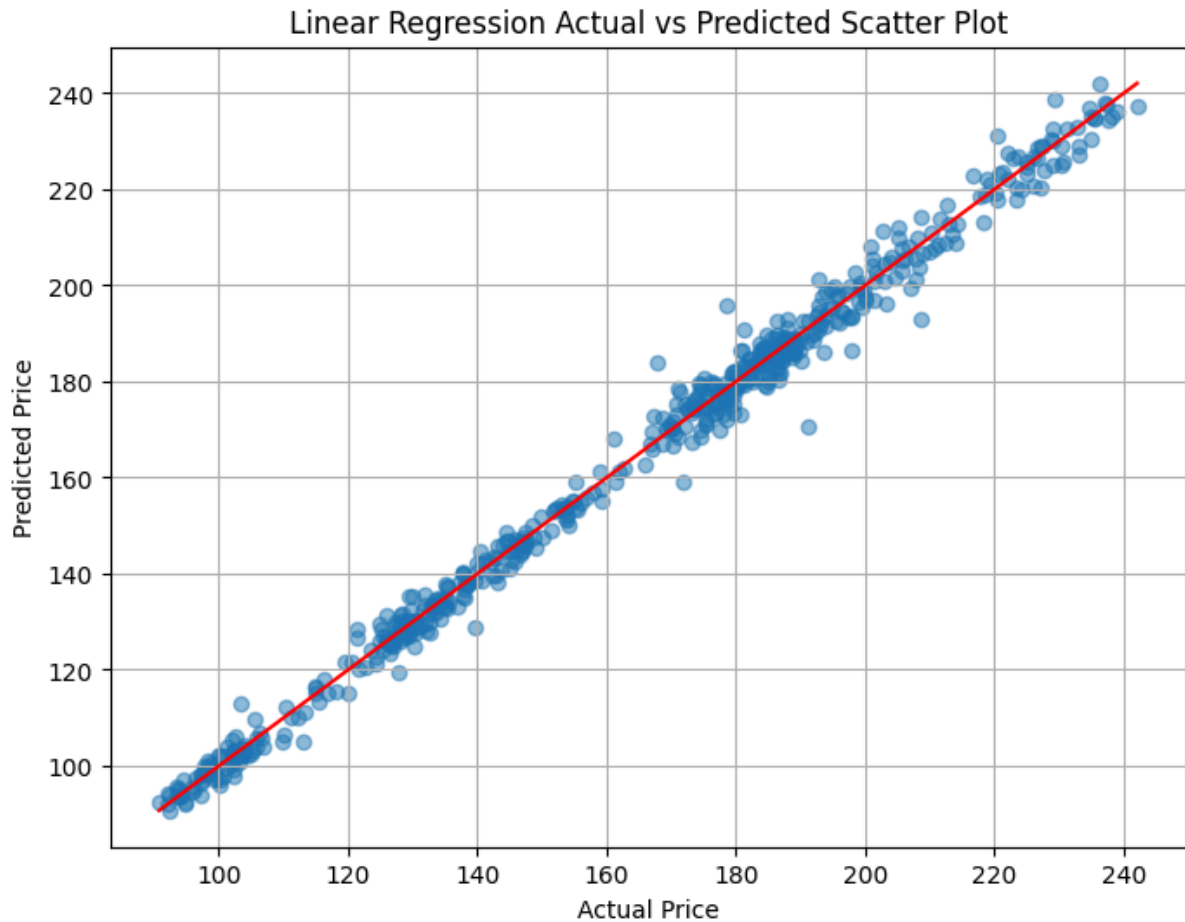
ניבוי על ה-test:



```
1 lr_preds_scaled = lr_model.predict(X_lr_test)
2
3 # Inverse-transform scaled predictions and true values
4 zeros = np.zeros((len(lr_preds_scaled), len(features)))
5 zeros[:, features.index("Close")] = lr_preds_scaled
6 lr_preds = scaler.inverse_transform(zeros[:, features.index("Close")])
7
8 zeros[:, features.index("Close")] = y_test
9 lr_actuals = scaler.inverse_transform(zeros[:, features.index("Close")])
```

תוצאות האימון של מודל ה-Linear Regression





מסקנות שלב האימון

כפי שניתן לראות לפי תוצאות מודל ה-LSTM לפי הגרפים, התוצאות הינן מספקות ביותר ומראות רמת ניבוי גבוהה.

לעומת זאת, מודל ה-Linear Regression, שהוא פשוט יותר, הגיע לביצועים טובים גם כן, ואף טובים יותר ממודל ה-LSTM במקרים מסוימים. זאת כנראה כתוצאה מפשטות המודל, וסביר להניח שעבור Dataset ושיטות אימון מסובכות יותר, מודל ה-LSTM היה מצליח טוב יותר ממודל ה-Linear Regression, שכן הוא מתוחכם יותר ויכול להגיע לרמות סיבוכיות גבוהות יותר.

יישום

היישום שפתיחתי במסגרת מימוש ממשק משתמש בפרויקט הוא אתר המאפשר למשתמש לבחור מניה משוק ההון, ותאריך (בהווה או בעבר) בו הוא רוצה לדעת האם כדאי לו לקנות/למכור את אותה המניה. לאחר בחירה המשתמש יקבל את התוצאה שהמודל שאימנתי נותן עבור הנתונים המבוקשים של המשתמש, ויחזיר למשתמש תשובה האם כדאי לו לקנות/למכור את המניה המבוקשת בתאריך המבוקש של המשתמש.

1. תיאור והסבר כיצד היישום משתמש במודל

היישום שפיתחתי משתמש במודל שאומן באמצעות שליחת בקשות POST מה client אל ה server הכתוב באמצעות ספריית flask בפייטון. כאשר ה server מקבל בקשת HTTP POST מהלקוח, הוא מריץ את הקוד שמבצע פעולת evaluation לנתונים שהוכנסו על ידי המשתמש, לפי המודל המאומן, ומחזיר את התשובה שהחוזרה על ידי המודל. כמובן שלפני כן היישום טוען את המודל המאומן ומוודא שהוא קיים ושאפשר להשתמש בו לשם הערכה של הנתונים שהמשתמש הכניס על ידי המודל.

תחילת הפונקציה המטפלת בבקשת POST הנשלחת מהלקוח אל השרת עם הנתונים המתאימים בשביל לבצע הערכה מתאימה (syntax של ספריית flask בפייטון):

```
1 @app.route("/evaluate", methods=["POST"])
2 def evaluate():
3     data = request.get_json()
4     symbol = data.get("symbol")
5     date_str = data.get("date")
```

2. תיאור הטכנולוגיה שעל פיה מומש ממשק המשתמש

הטכנולוגיה שעל פיה מומש ממשק המשתמש היא טכנולוגיה ידועה ויחסית פשוטה, אך מאוד יעילה בשפת התכנות פייטון, באמצעות ספריית flask. Flask היא ספרייה ליישומי אינטרנט שנכתבה בפייטון. היא מסווגת כ "microframework" מכיוון שהיא אינה דורשת כלים או ספריות. כמו כן, בממשק המשתמש שבניתי יש שימוש נרחב בטכנולוגיות אתרים ידועות של HTML5 ו-CSS. בנוסף, נעשה שימוש בטכנולוגית Tradingview Widgets, בשביל להציג גרף ניתוח מחירי עבר של המניה המבוקשת, אשר מתעדכן למניה המבוקשת על ידי המשתמש כאשר המשתמש בוחר את סמל המניה בשוק ההון.

3. תיאור קוד הקולט את ה-DATA שעליו יבוצע החיזוי

קלט סמל המניה והתאריך המבוקש על ידי המשתמש, באתר המוצג ללקוח:

```
1 document.getElementById('date').addEventListener('change', evaluate);
2
3 function evaluate() {
4     var date = document.getElementById('date').value;
5     var errorDiv = document.getElementById('error');
6     var resultBtn = document.getElementById('resultBtn');
7     var priceInfo = document.getElementById('price-info');
8
9     errorDiv.textContent = '';
10    priceInfo.textContent = '';
11    resultBtn.disabled = true;
12    resultBtn.className = 'trade-button disabled-button';
13    resultBtn.textContent = 'Result';
14
15    if (!currentTicker || !date) return;
16
17    fetch('/evaluate', {
18        method: 'POST',
19        headers: { 'Content-Type': 'application/json' },
20        body: JSON.stringify({ symbol: currentTicker, date: date })
21    })
22    .then(response => response.json().then(data => ({ status: response.status, data })))
23    .then(obj => {
24        if (obj.status !== 200) {
25            resultBtn.textContent = 'Error';
26            errorDiv.textContent = obj.data.error;
27        } else {
28            const data = obj.data;
29            // Show predicted price
30            priceInfo.textContent = `Predicted Price: ${data.predicted_price}`;
31            if (data.actual_price !== undefined) {
32                priceInfo.textContent += ` | Actual Price: ${data.actual_price}`;
33            }
34
35            // Show decision
36            resultBtn.textContent = data.decision;
37            resultBtn.disabled = false;
38            resultBtn.className = 'trade-button ' + (data.decision === 'Buy' ? 'buy-button' : 'sell-button');
39        }
40    })
41    .catch(err => {
42        resultBtn.textContent = 'Error';
43        errorDiv.textContent = 'Server error.';
44    });
45 }
```

קבלת המידע המתאים באמצעות שימוש בספריית yfinance בפייטון, עבור התאריך והמניה המתאימה שביקש המשתמש, חישוב ה-EMA המתאים:

```
1 @app.route("/evaluate", methods=["POST"])
2 def evaluate():
3     data = request.get_json()
4     symbol = data.get("symbol")
5     date_str = data.get("date")
6
7     # Validate date format
8     try:
9         chosen_date = datetime.datetime.strptime(date_str, "%Y-%m-%d").date()
10    except Exception:
11        return jsonify({"error": "Invalid date format"}), 400
12
13    # Download historical data (300 days before chosen date)
14    start_date = chosen_date - datetime.timedelta(days=300)
15    end_date = chosen_date.strftime("%Y-%m-%d")
16    ticker = yf.Ticker(symbol)
17    df = ticker.history(start=start_date.strftime("%Y-%m-%d"), end=end_date)
18
19    if df.empty or df.shape[0] < 60:
20        return jsonify({"error": "Not enough historical data or stock not found"}), 400
21
22    # Prepare last 60 days of Close and EMA for LSTM input
23    seq_length = 60
24    df["EMA"] = df["Close"].ewm(span=20, adjust=False).mean()
25
26    features = ["Close", "EMA"]
27    data_features = df[features].tail(seq_length).to_numpy()
```

התאמה למבנה נתונים המתאים לחיזוי:

```
1 # Normalize with MinMaxScaler (fit on current data)
2 scaler = MinMaxScaler()
3 data_scaled = scaler.fit_transform(data_features)
4
5 input_tensor = (
6     torch.FloatTensor(data_scaled).unsqueeze(0).to("cpu")
7 ) # shape: (1, 60, 2)
```

מדריך למפתח

:traider.ipynb

קובץ המחברת המכיל קוד פייטון.

בתוכו כלולים החלקים של שלב איסוף, הכנה וניתוח הנתונים, בנוסף לשלב בניית ואימון המודל.

להלן תוכן הקובץ בתצורה ישירה (קוד פייטון):

```
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader, TensorDataset
from IPython.display import clear_output
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm

# Configuration
TICKER = "AMZN" # Stock ticker symbol
SEQ_LENGTH = 60 # Sequence length for LSTM input
BATCH_SIZE = 32 # Batch size for training
EPOCHS = 100 # Number of training epochs
LEARNING_RATE = 0.002 # Learning rate for optimizer
HIDDEN_SIZE = 128
TRAIN_TEST_RATIO = 0.8 # Ratio of training to testing data
DROPOUT = 0.01
NUM_LAYERS = 3

# Download stock data
df = yf.download(TICKER, period="max", interval="1d", rounding=True,
prepost=True)
df.dropna(inplace=True)
df["EMA"] = df["Close"].ewm(span=5, adjust=False).mean()
df.dropna(inplace=True)

print(len(df))

print(df)
```



```

# Normalize features
features = ["Close", "EMA"]
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df[features])

# Prepare sequences
def create_sequences(data, seq_length):
    xs, ys = [], []
    for i in range(seq_length, len(data)):
        xs.append(data[i - seq_length:i])
        ys.append(data[i, features.index("Close")]) # Predicting the
'Close' price
    return np.array(xs), np.array(ys)

X, y = create_sequences(scaled_data[-3000:], SEQ_LENGTH)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=1-TRAIN_TEST_RATIO, shuffle=False)

# Convert to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32)

# Create DataLoader instances
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False)

# LSTM Model Definition
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, dropout):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
batch_first=True, dropout=dropout)
        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        out, _ = self.lstm(x)
        out = out[:, -1, :] # Take the output from the last time step
        out = self.fc(out)
        return out

```

```
# Initialize
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = LSTMModel(input_size=len(features), hidden_size=HIDDEN_SIZE,
num_layers=NUM_LAYERS, dropout=DROPOUT).to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)

# Logging
train_loss_log = []
val_loss_log = []

# Early stopping
best_val_loss = float('inf')
patience = 200
counter = 0
```

```

for epoch in range(EPOCHS):
    model.train()
    batch_train_losses = []

    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        optimizer.zero_grad()
        outputs = model(X_batch).squeeze()
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()
        batch_train_losses.append(loss.item())

    # Validation
    model.eval()
    batch_val_losses = []
    with torch.no_grad():
        for X_batch, y_batch in test_loader:
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)
            outputs = model(X_batch).squeeze()
            loss = criterion(outputs, y_batch)
            batch_val_losses.append(loss.item())

    avg_train_loss = np.mean(batch_train_losses)
    avg_val_loss = np.mean(batch_val_losses)
    train_loss_log.append(avg_train_loss)
    val_loss_log.append(avg_val_loss)

    clear_output(wait=True)
    print(f"Epoch [{epoch+1}/{EPOCHS}] | Train Loss:
{avg_train_loss:.6f} | Val Loss: {avg_val_loss:.6f}")

    # Early stopping
    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        counter = 0
        torch.save(model.state_dict(), "best_model.pth")
    else:
        counter += 1
        if counter >= patience:
            print("Early stopping triggered.")
            break

    # Load the best model
    model.load_state_dict(torch.load("best_model.pth"))

```

```

# Evaluation
model.eval()
predictions = []
actuals = []
with torch.no_grad():
    for X_batch, y_batch in test_loader:
        X_batch = X_batch.to(device)
        outputs = model(X_batch).squeeze().cpu().numpy()
        predictions.extend(outputs)
        actuals.extend(y_batch.numpy())

# Inverse transform predictions and actuals
predictions = np.array(predictions)
actuals = np.array(actuals)
zeros = np.zeros((len(predictions), len(features)))
zeros[:, features.index("Close")] = predictions
predictions_inverse = scaler.inverse_transform(zeros)[:,
features.index("Close")]

zeros[:, features.index("Close")] = actuals
actuals_inverse = scaler.inverse_transform(zeros)[:,
features.index("Close")]

# Calculate evaluation metrics
mae = mean_absolute_error(actuals_inverse, predictions_inverse)
rmse = np.sqrt(mean_squared_error(actuals_inverse, predictions_inverse))
print(f"Mean Absolute Error: {mae:.2f}")
print(f"Root Mean Squared Error: {rmse:.2f}")

# Plot predictions vs actuals
plt.figure(figsize=(12, 6))
plt.plot(actuals_inverse, label="Actual Prices")
plt.plot(predictions_inverse, label="Predicted Prices")
plt.xlabel("Time")
plt.ylabel("Price")
plt.title(f"{TICKER} Stock Price Prediction using Enhanced LSTM")
plt.legend()
plt.show()

from scipy.ndimage import gaussian_filter1d

```

```

def plot_loss_curves(train_losses, val_losses):
    plt.figure(figsize=(10, 6))
    smoothed_train = gaussian_filter1d(train_losses, sigma=2)
    smoothed_val = gaussian_filter1d(val_losses, sigma=2)

    plt.plot(smoothed_train, label='Training Loss', linewidth=2)
    plt.plot(smoothed_val, label='Validation Loss', linewidth=2,
linestyle='--')
    plt.title('Training vs Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('MSE Loss')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

plot_loss_curves(train_loss_log, val_loss_log)

# Function to analyze prediction errors
def analyze_errors(actual, predicted, model_name=""):
    errors = actual - predicted
    plt.figure(figsize=(12, 10))

    # Error distribution with normal curve
    plt.subplot(2, 2, 1)
    count, bins, ignored = plt.hist(errors, bins=50, density=True,
alpha=0.6, color='g')

    # Fit normal distribution
    mu, std = norm.fit(errors)
    xmin, xmax = plt.xlim()
    x = np.linspace(xmin, xmax, 100)
    p = norm.pdf(x, mu, std)

    plt.plot(x, p, 'k', linewidth=2)
    plt.xlabel('Error')
    plt.ylabel('Density')
    plt.title(f'{model_name} Error Distribution\nNormal Fit:  $\mu={mu:.2f}$ ,
 $\sigma={std:.2f}$ ')
    plt.grid(True)

```

```

# Scatter plot to show correlation between actual and predicted values
def scatter_actual_vs_predicted(actual, predicted, model_name=""):
    plt.figure(figsize=(8, 6))
    plt.scatter(actual, predicted, alpha=0.5)
    plt.xlabel('Actual Price')
    plt.ylabel('Predicted Price')
    plt.title(f'{model_name} Actual vs Predicted Scatter Plot')
    plt.plot([min(actual), max(actual)], [min(actual), max(actual)],
color='red') # y = x line
    plt.grid(True)
    plt.show()

# Linear Regression

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Use only the last time-step of each sequence as features
X_lr_train = X_train[:, -1, :] # shape: (n_samples, n_features)
X_lr_test  = X_test[:, -1, :]

# Train and predict
lr_model = LinearRegression()
lr_model.fit(X_lr_train, y_train)
lr_preds_scaled = lr_model.predict(X_lr_test)

# Inverse-transform scaled predictions and true values
zeros = np.zeros((len(lr_preds_scaled), len(features)))
zeros[:, features.index("Close")] = lr_preds_scaled
lr_preds = scaler.inverse_transform(zeros[:, features.index("Close")])

zeros[:, features.index("Close")] = y_test
lr_actuals = scaler.inverse_transform(zeros[:, features.index("Close")])

# Compute and print metrics
lr_mae = mean_absolute_error(lr_actuals, lr_preds)
lr_rmse = np.sqrt(mean_squared_error(lr_actuals, lr_preds))
print(f"Linear Regression MAE: {lr_mae:.2f}")
print(f"Linear Regression RMSE: {lr_rmse:.2f}")

```

```
# Plot Linear Regression Predictions vs Actual Prices
```

```
plt.figure(figsize=(12, 6))
plt.plot(lr_actuals, label="Actual Prices")
plt.plot(lr_preds, label="LR Predicted Prices")
plt.xlabel("Time")
plt.ylabel("Price")
plt.title(f"{TICKER} Stock Price Prediction using Linear Regression")
plt.legend()
plt.show()
```

```
# Plot LSTM vs Linear Regression results
```

```
analyze_errors(actuals_inverse, predictions_inverse, "LSTM")
analyze_errors(lr_actuals, lr_preds, "Linear Regression")
```

```
scatter_actual_vs_predicted(actuals_inverse, predictions_inverse,
                             "LSTM")
scatter_actual_vs_predicted(lr_actuals, lr_preds, "Linear Regression")
```

:~index.html

קובץ HTML5 המכיל את ה-Frontend של חלק היישום.

בתוכו מופיע כל התצוגה למשתמש ושליחת הבקשות ל-Backend.

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Traider</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        body {
            background-color: #1e222d;
            color: #fff;
            font-family: Arial, sans-serif;
            text-align: center;
            padding: 20px;
        }

        .trade-button {
            color: #fff;
            font-size: 16px;
            font-weight: bold;
            text-transform: uppercase;
            border: none;
            border-radius: 5px;
            background: transparent;
            padding: 12px 20px;
            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
            transition: transform 0.2s, box-shadow 0.2s;
            cursor: pointer;
            margin: 10px auto;
            display: inline-block;
            min-width: 120px;
        }

        .buy-button {
            color: #28a745;
            border: 2px solid #28a745;
        }

        .sell-button {
            color: #dc3545;
            border: 2px solid #dc3545;
        }
    </style>

```



```

    }

    .disabled-button {
        opacity: 0.5;
        cursor: not-allowed;
    }

    .error {
        color: red;
    }

    #price-info {
        margin-top: 10px;
        font-size: 18px;
    }
</style>
<!-- Load TradingView library -->
<script type="text/javascript"
src="https://s3.tradingview.com/tv.js"></script>
</head>

<body>
    <h1>Traider</h1>

    <!-- Ticker Input -->
    <div style="margin-top: 10px;">
        <label for="ticker">Ticker Symbol:</label>
        <input type="text" id="ticker" value="{{ widget_stats.symbol
    }}">
        <button onclick="updateWidget()">Update Chart</button>
    </div>

    <!-- TradingView widget container -->
    <div id="tradingview_chart" style="height:400px; width:75%; margin:
auto;"></div>

    <!-- Date input -->
    <div style="margin-top: 10px;">
        <label for="date">Select Date:</label>
        <input type="date" id="date">
    </div>

    <!-- Result button -->
    <div style="margin-top: 20px;">
        <button id="resultBtn" class="trade-button disabled-button"
disabled>Result</button>

```

```

</div>

<div id="price-info"></div>
<div id="error" class="error"></div>

<script>
    var currentTicker = document.getElementById('ticker').value;

    function initWidget() {
        document.getElementById('tradingview_chart').innerHTML = "";
        new TradingView.widget({
            "container_id": "tradingview_chart",
            "autosize": true,
            "symbol": currentTicker,
            "interval": "{{ widget_stats.interval }}",
            "timezone": "Etc/UTC",
            "theme": "dark",
            "style": "1",
            "locale": "en",
            "withdateranges": true,
            "hide_side_toolbar": false,
            "allow_symbol_change": false,
            "details": true,
            "hotlist": true,
            "calendar": false,
            "support_host": "https://www.tradingview.com"
        });
    }

    initWidget();

    function updateWidget() {
        currentTicker = document.getElementById('ticker').value;
        initWidget();
        evaluate();
    }

    document.getElementById('date').addEventListener('change',
evaluate);

    function evaluate() {
        var date = document.getElementById('date').value;
        var errorDiv = document.getElementById('error');
        var resultBtn = document.getElementById('resultBtn');
        var priceInfo = document.getElementById('price-info');

```

```

        errorDiv.textContent = '';
        priceInfo.textContent = '';
        resultBtn.disabled = true;
        resultBtn.className = 'trade-button disabled-button';
        resultBtn.textContent = 'Result';

        if (!currentTicker || !date) return;

        fetch('/evaluate', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ symbol: currentTicker, date: date
    })
    })
        .then(response => response.json().then(data => ({
status: response.status, data })))
        .then(obj => {
            if (obj.status !== 200) {
                resultBtn.textContent = 'Error';
                errorDiv.textContent = obj.data.error;
            } else {
                const data = obj.data;
                // Show predicted price
                priceInfo.textContent = `Predicted Price:
${data.predicted_price}`;
                if (data.actual_price !== undefined) {
                    priceInfo.textContent += ` | Actual Price:
${data.actual_price}`;
                }

                // Show decision
                resultBtn.textContent = data.decision;
                resultBtn.disabled = false;
                resultBtn.className = 'trade-button ' +
(data.decision === 'Buy' ? 'buy-button' : 'sell-button');
            }
        })
        .catch(err => {
            resultBtn.textContent = 'Error';
            errorDiv.textContent = 'Server error.';
        }));
    }
</script>
</body>

</html>

```

:app.py

קובץ פייטון המכיל את ה-Backend של חלק היישום. הוא מקבל את הבקשות מחלק ה-Frontend שגשלחות לפי בקשות המשתמש. מתאים את הנתונים שהתקבלו מהמשתמש לטעינה למודל. מעביר את הנתונים למודל, ומחזיר למשתמש את התוצאות שחזה המודל.

```
import datetime

import numpy as np
import torch
import torch.nn as nn
import yfinance as yf
from flask import Flask, jsonify, render_template, request
from sklearn.preprocessing import MinMaxScaler

app = Flask(__name__)

# Define the same LSTM model class used during training
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, dropout):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(
            input_size, hidden_size, num_layers, batch_first=True,
            dropout=dropout
        )
        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        out, _ = self.lstm(x)
        out = out[:, -1, :] # Take the last output
        out = self.fc(out)
        return out

@app.route("/")
def index():
    widget_stats = {"symbol": "AMZN", "interval": "D"}
    return render_template("index.html", widget_stats=widget_stats)

@app.route("/evaluate", methods=["POST"])
def evaluate():
    data = request.get_json()
    symbol = data.get("symbol")
    date_str = data.get("date")

    # Validate date
    try:
```

```

        chosen_date = datetime.datetime.strptime(date_str,
"%Y-%m-%d").date()
    except Exception:
        return jsonify({"error": "Invalid date format"}), 400

    today = datetime.date.today()
    if chosen_date > today:
        return jsonify({"error": "Date is in the future"}), 400

    # Download historical data
    start_date = chosen_date - datetime.timedelta(days=300)
    end_date = chosen_date.strftime("%Y-%m-%d")
    try:
        ticker = yf.Ticker(symbol)
        df = ticker.history(start=start_date.strftime("%Y-%m-%d"),
end=end_date)
    except Exception as e:
        print(f"Error downloading data: {e}")
        return jsonify({"error": "Error downloading data"}), 500

    if df.empty or df.shape[0] < 60:
        return jsonify({"error": "Not enough historical data or stock
not found"}), 400

    # Prepare last 60 days of Close and EMA for LSTM input
    seq_length = 60
    df["EMA"] = df["Close"].ewm(span=20, adjust=False).mean()

    features = ["Close", "EMA"]
    data_features = df[features].tail(seq_length).to_numpy()

    # Normalize with MinMaxScaler (fit on current data)
    scaler = MinMaxScaler()
    data_scaled = scaler.fit_transform(data_features)

    input_tensor = (
        torch.FloatTensor(data_scaled).unsqueeze(0).to("cpu")
    ) # shape: (1, 60, 2)

    # Load trained model
    input_size = 2
    hidden_size = 128
    num_layers = 3
    dropout = 0.01

    try:

```

```

        model = LSTMModel(input_size, hidden_size, num_layers,
dropout).to("cpu")
    except Exception as e:
        print(f"Model init error: {e}")
        return jsonify({"error": "Model not loaded"}), 500

    try:
        model.load_state_dict(
            torch.load("model/best_model.pth",
map_location=torch.device("cpu"))
        )
    except Exception as e:
        print(f"Model load error: {e}")
        return jsonify({"error": "Model weights not found"}), 500

    model.eval()

    # Predict and inverse transform
    try:
        with torch.no_grad():
            pred_norm = model(input_tensor).item()

            dummy_row = np.zeros((1, len(features))) # zeros for all
features
            dummy_row[0, features.index("Close")] = pred_norm # put
predicted close value

            pred_inverse = scaler.inverse_transform(dummy_row)[0,
features.index("Close")]
        except Exception as e:
            print(f"Prediction error: {e}")
            return jsonify({"error": "Prediction failed"}), 500

    # Determine buy/sell decision
    last_close = df["Close"].iloc[-1]
    decision = "Buy" if pred_inverse > last_close else "Sell"

    # Return response
    response = {"predicted_price": round(pred_inverse, 2), "decision":
decision}

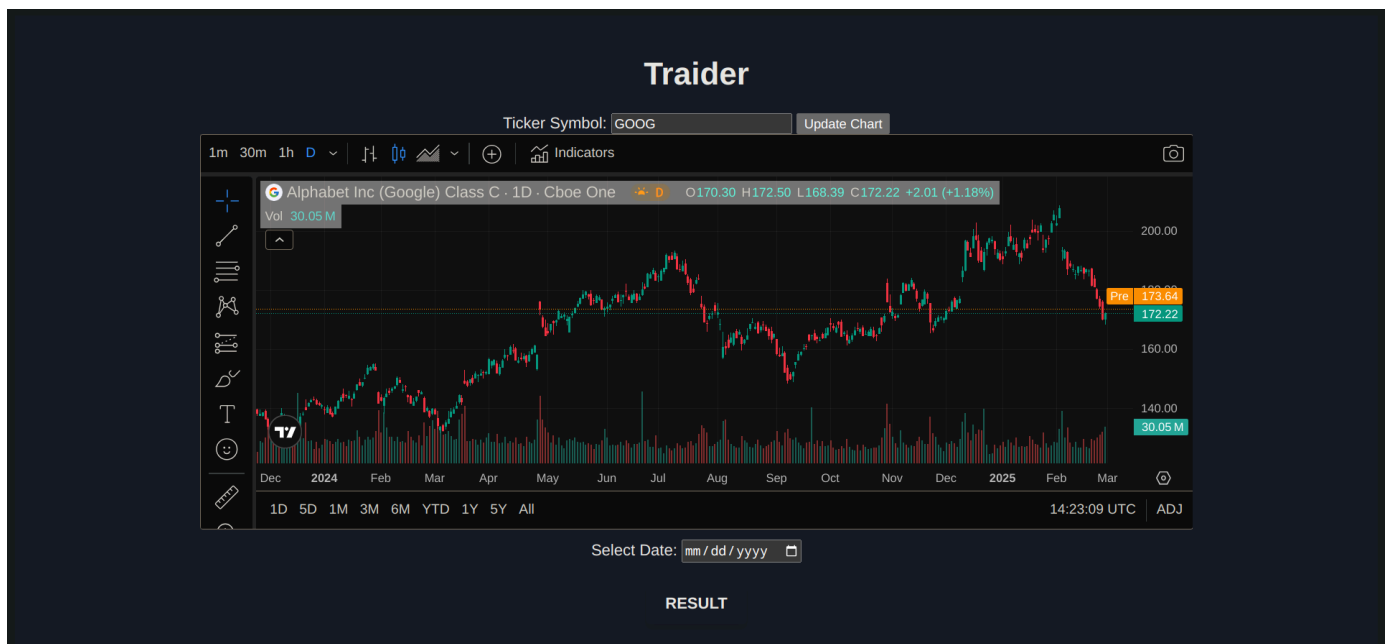
    return jsonify(response)

if __name__ == "__main__":
    app.run(debug=True)

```

תיעוד מדריך למשתמש

היישום שפיתחתי משתמש במסך אחד בלבד, המציג את כל האפשרויות המוצעות למשתמש ביישום. האפשרויות הללו כוללות: בחירת סמל מניה משוק ההון. בחירת תאריך רצוי עבורו המודל ינתח את כדאיות קניה/מכירה של המניה שנבחרה בתאריך שנבחר. גרף TradingView המציג את מחירי העבר של המניה לשם נוחות וניתוח נוסף של המניה על ידי הלקוח, במקום אחד, בלי צורך בשינוי מסך.



הסבר עבור כל אלמנט תצורה:

הכנסת סמל מניה - תיבת input בה ניתן לכתוב את סמל המניה הייחודי לה, כאשר המניה צריכה להיות מניה מאחד משווקי ההון בעולם. במידה והמניה לא קיימת תוצג הודעת שגיאה, המתארת למשתמש כי סמל המניה שהוכנס אינו קיים, במקום בו כתוב "RESULT" (בתמונה מעלה).

עדכון טבלה - לאחר בחירת סמל מניה והקלדתו בתיבת הטקסט הנ"ל, על המשתמש ללחוץ על הכפתור לעדכון הגרף המוצג ופרמטר סמל המניה המועבר למודל לשם הערכה. על הכפתור כתוב: "Update Chart", שכן הוא יעדכן את המניה המוצגת בטבלת ה TradingView מטה.

גרף מניה - גרף TradingView Widgets המציג את מחירי העבר של המניה הנבחרת ומאפשר הצגת נתונים נוספים על המניה לשם אפשרויות ניתוח עצמי על ידי המשתמש, כגון הוספת

אינדיקטורים, ניתוח נתונים פיננסיים ועוד (כל הנתונים והניתוחים הללו הם באחריות TradingView ואינן חלק מהפיתוחים של פרויקט זה).

בחירת תאריך - תיבת בחירה המאפשרת למשתמש לבחור תאריך. התאריך הנבחר על ידי המשתמש הוא התאריך שינותח על ידי המודל (לאחר שיבחר), ולגביו יוחזר ניתוח המודל לגבי פעולות קניה/מכירה של המניה הנבחרת (בתאריך הנבחר). במידה ונבחר תאריך בעתיד, תוצג הודעת שגיאה למשתמש במקום בו כתוב "RESULT" (בתמונה מעלה). בנוסף אם התאריך הנבחר הוא לפני תאריך ה-IPO של המניה (התאריך בו חברה פרטית מציעה בפעם הראשונה את מניותיה למכירה לציבור), כלומר אין נתוני מחירי מניה לפני תאריך זה, גם כן תוצג הודעות שגיאה למשתמש באותו המקום (בדומה לשאר הודעות השגיאה).

רפלקציה

בתהליך העבודה על הפרויקט נחשפתי למגוון אתגרים והזדמנויות למידה במספר תחומים:

- **אינטגרציה בין טכנולוגיות**

למדתי לשלב בצורה חלקה בין מודל למידת מכונה, מערכת backend ב-Flask, ושילוב עם רכיב גרפי (TradingView Widget). החוויה הזו המחישה לי עד כמה חשוב להבין את אופן הפעולה והאינטגרציה בין כל רכיבי המערכת השונים.

- **פשטות מול פונקציונליות**

למרות הרצון לשמור על פתרון פשוט ומינימלי, נדרשתי להוסיף תכונות חיוניות כגון עדכון ידני של הסמל, טיפול בקלטי משתמש, והצגת הודעות שגיאה בצורה ברורה. זאת על מנת להבטיח חווית משתמש טובה ואמינה.

- **התמודדות עם נתונים בזמן אמת**

העבודה עם נתונים היסטוריים באמצעות yfinance והערכתם בזמן אמת ליצירת החלטת "Buy" או "Sell" חשפה בפניי את הקשיים בניהול נתונים, כמו זמינות נתונים, טיפול בערכים חריגים והתאמת המודל לתנאי השוק המשתנים.

- **למידה ושיפור מתמיד**

הפרויקט אפשר לי להבין לעומק את חשיבות ניהול הפרויקט, תיעוד נכון ומסודר, ותכנון מוקדם של ממשקי המשתמש. למדתי גם על הדרכים לשפר את ביצועי המודל ועל החשיבות של בדיקות ושיפורים מתמשכים.

בסך הכל, הפרויקט היה עבורי חוויה מעצימה שבה למדתי על אתגרי הפיתוח בשילוב בין למידת מכונה, פיתוח ווב ועיצוב חוויית משתמש. אני רואה בפרויקט זה בסיס ייחודי ומשמעותי להמשך התפתחות והעמקה בתחומים אלו, ואני מתכוון להמשיך ולפתח את המערכת עם שיפורים נוספים בעתיד.

ביבליוגרפיה

Flask - <https://flask.palletsprojects.com/en/stable/>

Python - <https://docs.python.org/3/>

Pytorch Adam -

<https://docs.pytorch.org/docs/stable/generated/torch.optim.Adam.html>

Pytorch LSTM - <https://docs.pytorch.org/docs/stable/generated/torch.nn.LSTM.html>

Linear Regression -

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Pytorch MSELoss -

<https://docs.pytorch.org/docs/stable/generated/torch.nn.MSELoss.html>

TradingView Widgets - <https://www.tradingview.com/widget-docs/widgets/>

Matplotlib docs - <https://matplotlib.org/stable/api/index>