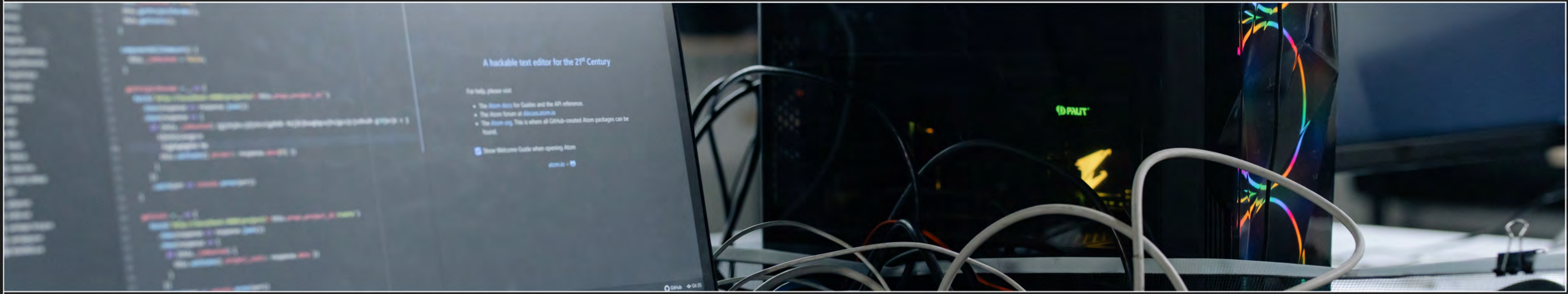
Search

Team 7



Andrada-Ioana Ilie; Ivan Sapozhnikov; Elena Strimbeanu-Iacoban; Amir Vakili

Search

Explaining the libraries

```
import math
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import yfinance as yf
```

What are the libraries that we are importing?
math is used for mathematical operations such as rounding up values. numpy provides efficient numerical operations, especially for arrays, while pandas helps handle structured data in DataFrames. Sklearn.preprocessing.MinMaxScaler is used later for scaling data between 0 and 1, which helps with LSTM model training. Additionally, matplotlib.pyplot helps visualize stock data and yfinance fetches stock data from Yahoo Finance.

Search

Loading the data

df = yf.download('AAPL', start="2012-01-01", end='2025-02-10')

Yahoo Finance (yfinance) is used to download stock data for Apple (AAPL). The dataset starts from January 1, 2012, and ends on February 10, 2025. Also df will contain historical stock prices, including Open, High, Low, Close, Volume, and Adjusted Close.

data = df[['Close']] # Select only 'Close' column

Instead of using all stock price data, the code selects only the 'Close' price, which is typically used in stock price forecasting. 'Close' price is the final trading price of the stock for each day and is widely used in technical analysis.

Search

Arranging the data

dataset = data.values

Converts the Pandas DataFrame into a NumPy array. LSTMs in TensorFlow/Keras require numerical arrays instead of DataFrames.

training_data_len = math.ceil(len(dataset) * 0.8)

Defining the Training Data Length Determines how much of the data should be used for training (80% of the dataset). math.ceil() ensures the number is rounded up to the nearest whole number, so the training set includes enough data points.

Search

Methodology: Fix Data Leakage

Why is data leakage a problem? Data leakage happens when information from outside the training dataset is used to create the model, leading to **overly optimistic** performance estimates during training. This can cause the model to fail when making real-world predictions because it has already "seen" future data.

How does fitting the scaler only on training data help? MinMaxScaler calculates statistics (e.g., min/max values) from the data and applies transformations. If we fit the scaler on the entire dataset, it indirectly exposes the model to information from future data (i.e., the test set), leading to **unrealistic performance expectations.**

Instead, by fitting the scaler only on the training data, we ensure that the test set remains truly "unseen," mimicking how the model would encounter **new data in real-world** scenarios.
This step is **crucial in time series forecasting**, where preserving the temporal order of data is necessary to avoid future information leaking into the past.

Search

Methodology: Fix Data Leakage

```
scaler = MinMaxScaler(feature_range=(0,1))
```

MinMaxScaler is a tool from the **sklearnlibrary** used for normalization. **feature_range=(0,1)** specifies that we want to scale the data to a range between 0 and 1. This is a common choice. The variable scaler now holds this scaling tool, ready to be used.

Fitting the Scaler on Training Data:

```
train_data = dataset[:training_data_len]  # Only training portion
scaler.fit(train_data)  # Fit only on training data
```

train_data is created by slicing the original dataset (dataset) and taking only the portion designated for training. **scaler.fit(train_data)** is the crucial step. It calculates the minimum and maximum values only from the training data. These values will be used to perform the scaling. This is essential to prevent data leakage. Transforming the Data:

```
scaled_train_data = scaler.transform(train_data)
scaled_test_data = scaler.transform(dataset[training_data_len:])
```

Search

Vali

Explaining the code

Initialization:
x_train, y_train = [], []: Two empty lists, x_train and y_train, are created to store the training data and target values, respectively.

Looping through the Data:
for i in range(60, len(scaled_train_data)):
This loop iterates through the scaled_train_data, starting from the 60th data point to the end of the data. The loop variable i represents the current index.

Creating Input Sequences (x_train):
x_train.append(scaled_train_data[i-60:i, 0])
For each index i, a slice of the scaled_train_data is taken from index i-60 to i (exclusive). This slice represents a sequence of **60 previous data points**. The 0 in **[:, 0]**indicates that we are selecting only the **first column** (which typically represents the 'Close' price in this type of financial data). This sequence is then appended to the x_trainlist.

Search

Vali

Explaining the code

Creating Target Values (y_train):

y_train.append(scaled_train_data[i, 0]): The target value for the current input sequence is the data point at index i (again, from the first column). This value is appended to the y_train list. In essence, this code is doing the following:

It's creating a dataset where each input (x_train) is a sequence of 60 previous stock prices. The corresponding output (y_train) for each input sequence is the next stock price in the dataset. This structure allows the LSTM model to **learn patterns from historical data** and make predictions about future stock prices based on those patterns.

Building the model

```
model = Sequential()
model.add(LSTM(50, input_shape=(x_train.shape[1], 1), return_sequences=True))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))
```

First line initializes a sequential model. A sequential model is a linear stack of layers, where you add layers one after the other.
The second line adds the first LSTM (Long Short-Term Memory) layer to the model. LSTMs are a type of recurrent neural network (RNN) that are well-suited for processing sequential data like time series. First parameter (**50**) specifies the number of LSTM units (or cells) in the layer. More units can capture more complex patterns but might increase computational cost.

input_shape=(x_train.shape[1], 1), return_sequences=True))

This defines the shape of the input data expected by the LSTM layer. It's derived from the shape of your training data
return_sequences=True

This is important for stacking LSTM layers. It means that this layer will output a sequence, which can be fed as input to another LSTM layer.

Search

Optimizer and loss

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

This line is a crucial step in preparing the machine learning model for training. It essentially configures the learning process of the model. Here's a breakdown of the two key parts: optimizer and loss.

optimizer: This specifies the algorithm that will be used to update the model's internal parameters (weights and biases) during training. The goal of the optimizer is to find the best set of parameters that minimize the prediction error.

loss: This defines the function that measures the difference between the model's predictions and the actual values (ground truth) in your training data. The goal of training is to minimize this loss function.

'mean_squared_error: This is a common loss function for regression problems (predicting continuous values, like stock prices in this case). It calculates the average of the squared differences between predicted and actual values. This penalizes larger errors more heavily

Search

Making predictions

predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

Making predictions with the data
1.Uses the trained LSTM model to predict future stock prices based on x_test (test data).
2.When training the model, the data was scaled between 0 and 1 using MinMaxScaler to help the model learn better. This step converts predictions back to the original stock price range so that the results are interpretable.

1.train = data[:training_data_len]
2.valid = data[training_data_len:]

Splitting data into Training and Validation sets

1. The first 80% of the data, used to train the LSTM model
2. The remaining 20% of the data, used to evaluate the model's performance.

Search

Vali

Explaining the code

```
1.valid = valid.iloc[:len(predictions)]
2.valid['Predictions'] = predictions.flatten()
```

Matching valid dataset with predictions

1. Ensures valid has the same number of rows as predictions. This is needed because the LSTM model may output fewer predictions than the total test data (due to how sequences are structured)
2. Stores the model's predictions in the valid DataFrame

```
print(len(valid))
print(len(predictions))
```

Verifies that valid and predictions have the same length. Ensures no mismatches that could cause issues when visualizing or analyzing results.

Search

Setting up the plot

```
plt.figure(figsize=(16,8))
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
```

Setting up the plot
It creates a large figure (16x8 inches) for better visibility and adds title and labels to make the chart easy to interpret.

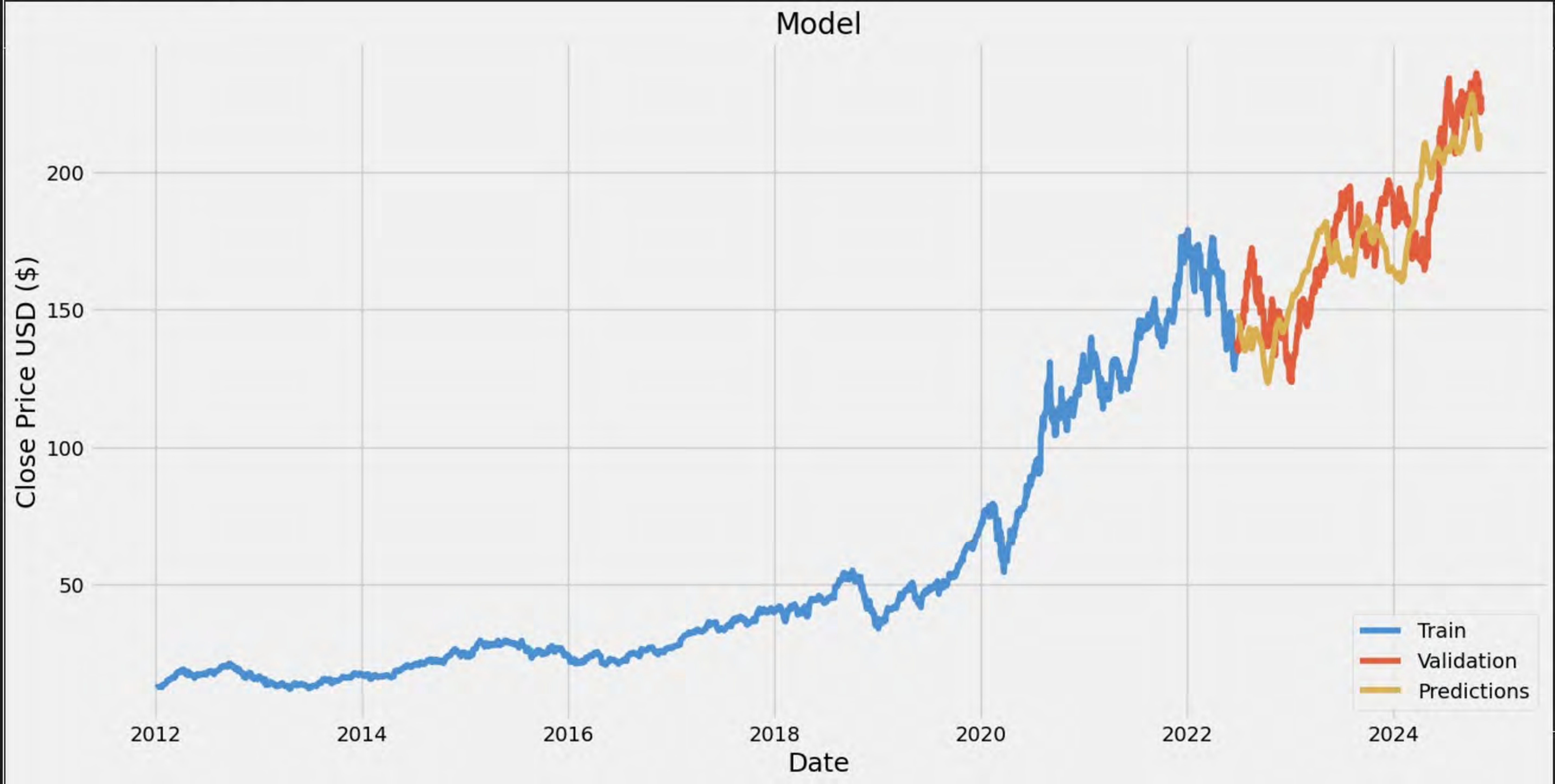
```
plt.plot(train['Close'], label='Train')
plt.plot(valid['Close'], label='Validation')
plt.plot(valid['Predictions'], label='Predictions')
```

Plotting the stock price data
train['Close'] Represents historical training data.
valid['Close'] Represents actual closing prices for the test period.
valid['Predictions'] Represents model's predicted prices.
This visually compares the predictions to actual prices, helping assess model accuracy.

Search

Vali

The Plot



Search

Thank you for listening!