1. 🌞 Project Motivation & Problem Statement

- **Challenge**: Predicting stock prices is hard due to market volatility and non-linear behaviors.
- **Use Cases**: Valuable for personal investing, algorithmic trading, and financial forecasting systems.
- **Objective**: Forecast **Apple (AAPL)** daily closing prices from **2013 to 2018**, using a **deep learning** model.
- **Goal**: Minimize prediction error (e.g., MSE, RMSE) and ensure the model generalizes well to unseen data.

---

2. 🧰 Tools, Libraries & Setup

- **Data Handling**: Pandas, NumPy
- **Visualization**: Matplotlib, Seaborn
- **Deep Learning**: TensorFlow and Keras
- **Preprocessing**: sklearn.preprocessing.MinMaxScaler
- **Utilities**:
    - datetime for date handling
    - warnings.filterwarnings("ignore") to suppress non-critical warnings

---

3. 📥 Data Loading & Preprocessing

1. **Load Dataset**

python

CopyEdit

data = pd.read_csv('all_stocks_5yr.csv', delimiter=',', on_bad_lines='skip')

    - Contains 5 years of stock data (Apple + others).
    - Skips malformed rows during import.

2. **Convert date to datetime**

python

CopyEdit

data['date'] = pd.to_datetime(data['date'])

3. **Filter Apple Stock Data**

python

CopyEdit

apple = data[data['Name']=='AAPL']

- Isolates AAPL entries for focused analysis.

4. **Select Training Range**

python

CopyEdit

train_data = apple[apple['date'] < '2018-01-01']

training_count = ceil(len(train_data)*0.95)

- Uses ~95% of AAPL data for training, remainder for testing.

---

4. 📈 Exploratory Data Analysis (EDA)

- **Open vs Close Prices**:
  Visualize trends and compare price movements over time.
- **Volume Analysis**:
  Chart trading volume to understand market activity and investor interest.
- **Trend Plot for AAPL**:
  Shows overall upward/downward movements from 2013–2018 — important for modeling.
- **Structural Data Checks**:
  Use .info(), .sample(), and .shape() to ensure data consistency, types, and completeness.

---

5. 🔄 Feature Engineering

- **Use Closing Price Only**:
  Extract it as close_data = apple[['close']], then convert to NumPy array.
- **Normalize Data**:

python

CopyEdit

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

scaled_data = scaler.fit_transform(dataset)

- **Create 60-Day Sliding Windows**:
  - For each day *t*, use the previous 60 days as features (x_train)
  - The label (y_train) is the day *t* closing price

python

CopyEdit

x_train, y_train = [], []

```
for i in range(60, len(train_data)):
    x_train.append(scaled_data[i-60:i, 0])
    y_train.append(scaled_data[i, 0])
x_train = np.reshape(x_train, (..., 60, 1))
```

---

6. 🧠 LSTM Model Architecture

- **Layer Structure**:
  - LSTM (64 units, return_sequences=True)
  - LSTM (64 units)
  - Dense (32 units)
  - Dropout (0.5) — helps prevent overfitting
  - Dense Output (1 unit) — predicts the closing price
- **Why LSTM?**
  Handles sequential data well and mitigates vanishing gradient issues.
- **Stacked Architecture**:
  Two layers of 64 LSTM units capture deeper temporal patterns.

---

7. ⚙️ Model Compilation & Training

- **Compilation**:

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

  - Adam optimizer: adaptive and efficient
  - MSE loss: common for regression tasks

- **Training**:

```
python
CopyEdit
history = model.fit(x_train, y_train, epochs=10)
```

  - Fit for 10 epochs (can be extended to 50–100 for better performance)

---

8. 🔍 Testing & Prediction

- **Prepare Test Sequences**:
  Start 60 days before test set, then build similar 60-day sliding windows.
- **Predict & Scale Back**:

python

CopyEdit

preds = model.predict(x_test)

predictions = scaler.inverse_transform(preds)

- **Evaluate**:

python

CopyEdit

mse = np.mean((predictions - y_test)**2)

rmse = sqrt(mse)

---

9. 📊 Results & Visualization

- **Plotting**:
  - Training segment
  - Real prices from the test set
  - Predicted values
- **Performance Insight**:
  - How close predictions follow actual prices
  - Identify leads/delays in the model
  - Assess overall fit and any divergence

---

10. ✅ Key Takeaways

- LSTM effectively captures time-dependent patterns in stock prices.
- A 60-day look-back window strikes a good balance between historical context and model complexity.
- Normalizing data ensures stable training.
- Although training for 10 epochs gives a good start, more epochs improve accuracy.
- Visualizations are vital for interpreting model performance and guiding improvements.

🎯 Future Enhancements

1. Integrate **sentiment analysis** via news or social media to capture market sentiment.
2. Explore **Transformer-based models** (e.g., Time2Vec, Temporal Fusion Transformer).
3. Expand to **multi-stock/multivariate prediction** using features like volume, moving averages.
4. Implement **real-time forecasting dashboards** for live trading environments.
5. Add **auto-retraining mechanisms** for continuous learning as new data arrives.

💬 Final Thoughts

Your project presents a robust workflow from data ingestion to predictive modeling and evaluation. You successfully demonstrated how LSTM, a deep learning method, can be applied effectively to financial time-series data. Each phase—data quality checks, preprocessing, model architecture, evaluation, and visual interpretation—contributes to building reliable forecasting systems. Your roadmap for future work shows thoughtful awareness of how to increase model sophistication and real-world relevance.

If you'd like, I can help convert this into polished slides or a detailed report. Let me know!

IMP: LSTM

📘 What is LSTM?

LSTM (Long Short-Term Memory) is a type of Recurrent Neural Network (RNN) designed to learn long-term dependencies in time series data. It helps solve issues of vanishing gradients and remembers important patterns across long sequences — ideal for stock price prediction.

🧠 Model Architecture Breakdown:

🔸 **1. Stacked LSTM Layers (2 Layers of 64 Units)**

- **Definition: A "stacked LSTM" means more than one LSTM layer is placed one after the other to allow the model to learn both low-level and high-level temporal patterns.**
- **return_sequences=True:**
    - **In the first LSTM layer, we use return_sequences=True to pass the full output sequence to the next LSTM layer.**
    - **This is necessary when stacking multiple LSTM layers.**
- **Why 64 Units?**

- Each LSTM layer has 64 memory cells (units), which control how much past information is remembered.
- More units allow the model to learn more complex patterns, but also increase computational cost.

🔸 2. Dense Layer (32 Units)

- After the LSTM layers, the output is flattened and passed to a fully connected dense layer with 32 neurons.
- Purpose: Combines the learned temporal features and starts preparing them for prediction.
- Activation: Typically ReLU is used here for non-linearity.

🔸 3. Dropout Layer (Rate = 0.5)

- Dropout is used for regularization to prevent overfitting.
- With dropout rate of 0.5, half of the neurons are randomly disabled during training to improve generalization.
- Helps the model perform better on unseen data.

🔸 4. Output Layer (1 Unit)

- This is a single neuron output, as we are doing a regression task (predicting stock price).
- Activation: Usually linear (activation=None) since we are predicting continuous values.

📊 Visual Guide (model.summary())

Here's a textual version of what the model summary looks like:

txt

CopyEdit

_____

Layer (type) Output Shape Param #

==========

LSTM (64 units) (None, 60, 64) X1

LSTM (64 units) (None, 64) X2

Dense (32 units) (None, 32) X3

Dropout (0.5) (None, 32) 0

Dense (1 unit) (None, 1) 33

==========

Total params: ~X (depends on input shape)

📝 **Notes:**

- **The input shape is typically (batch_size, time_steps, features). E.g., (None, 60, 1) if you use 60-day window with 1 feature (close price).**
- **Each LSTM layer has trainable parameters calculated using: (4×units×(units+inputdim+1))(4 × units × (units + input_dim + 1))(4×units×(units+inputdim+1))**
  This is because each LSTM cell has 4 gates (input, forget, cell, output).

✅ **Benefits of This Architecture**

- **Captures temporal dependencies in stock data using LSTM layers.**
- **Reduces overfitting using dropout.**
- **Provides a smooth regression output for predicting stock prices.**
- **Stacked layers help learn hierarchical time series patterns.**

📚 **Based on GeeksforGeeks & Best Practices**

According to GFG and TensorFlow practices:

- **Always use return_sequences=True if stacking LSTM layers.**
- **Use Dropout after dense or LSTM layers to reduce overfitting.**
- **Final layer for time series regression should be a single neuron with no activation.**

Model Traing:-

Prepare Test Sequences:

“Start 60 days before test set, then build similar 60-day sliding windows.”

📌 **Context:**

In LSTM-based stock price prediction, we use sequences (or windows) of historical data to predict the next value.
For example:

- **If you want to predict the stock price on Day 61, you use data from Day 1 to Day 60 as input.**

🔍 **What does this line mean?**

Let's break it down step by step:

✅ **Step 1: Train-Test Split**

- **You already have split your data into training and testing parts.**
- **Let's say your training data ends at Day 200.**
- **Your test data starts at Day 201.**

✅ **Step 2: Why Start 60 Days Before Test Set?**

- **LSTM needs 60 previous days to predict the next day's price.**
- **So to predict stock price on Day 201, you need data from Day 141 to 200.**

📌 **That's why we start 60 days before the first test day — to have a complete input sequence.**

✅ **Step 3: Build Sliding Windows**

- **A sliding window is a moving 60-day sequence:**
  - **First window: Days 141–200 → Predict Day 201**
  - **Next window: Days 142–201 → Predict Day 202**
  - **And so on…**

Each window slides 1 day forward until all test predictions are prepared.

🧠 **Summary:**

- **Start 60 days before the test dataset starts.**
- **Create overlapping 60-day sequences (windows).**
- **These sequences become input to the LSTM to predict stock prices for each day in the test set.**

📈 **Example:**

Window #	Input Days	Predicting
1	141 → 200	Day 201
2	142 → 201	Day 202
3	143 → 202	Day 203
…	…	…