VERSION_1{BASE MODEL}:

Stock Prediction Model Using LSTM:

Introduction:

The goal of this Model nis to build a predictive model for stock price trends of selected Indian banks. The model uses Long Short-Term Memory (LSTM) networks, a type of recurrent neural network suited to time-series data, to forecast whether the stock price will rise or fall.

Model Overview:

- Objective: Forecast stock price trends (increase or decrease).

- Data Source: Yahoo Finance.

- Tools: Python, yfinance, pandas, Keras, TA-Lib.

- Approach: LSTM neural networks with technical indicators as features.

Code Overview:

1. Setting Up the Environment

- Importing Libraries:

- **`yfinance`** to fetch historical stock data.
- **`pandas`** for data manipulation.
- The **`bank_stocks`** list comprises the tickers of selected Indian banks.

- Defining Time Range: The period from 2013 to 2023, providing a decade of data for a comprehensive analysis.

2. Data Acquisition and Preprocessing

- Fetching Stock Data: Using `yfinance.download()` for each stock ticker.

- Purpose: To obtain historical stock data including open, close, high, low, and volume.

- **Handling Missing Data:** `fillna(method='ffill')` forward-fills missing values to maintain data continuity.

- **Saving Processed Data:** Exporting each stock's data frame to CSV for persistence and ease of access.

3. Feature Engineering

- **Technical Indicators:** Utilizing `ta.add_all_ta_features()` to add various technical indicators like RSI, MACD, and Bollinger Bands.

  - **Rationale:** These indicators are crucial for analyzing market trends and momentum, which are predictive of future price movements.

4. Building the LSTM Model

- Creating a Sequential Model: To stack LSTM layers linearly.

- **Adding LSTM Layers:**

  - **First LSTM Layer:** `return_sequences=True` for passing sequential data to the next LSTM layer.

  - **Second LSTM Layer:** To further process the sequential information.

- **Output Layer:** A `Dense` layer with a `sigmoid` activation function, predicting the probability of a price increase.

5. Model Compilation and Training

- **Compilation:** Using `binary_crossentropy` for binary classification and `adam` optimizer for efficient learning.

- **Training:** Executing `model.fit()` to train the model over specified epochs and batch size.

  - **Epochs:** The number of complete passes through the training dataset.

  - **Batch Size:** The number of samples processed before the model is updated.

6. Model Evaluation

- **Predictions:** Using `model.predict()` to forecast stock trends on the test dataset.

- **Classification Report:** Assessing model performance with precision, recall, and f1-score.

  - **Precision:** Proportion of true positive predictions.

  - **Recall:** Ability of the model to find all positive samples.

  - **F1-Score:** Harmonic mean of precision and recall.

Detailed Analysis of the Stock Prediction Model

1. **Importing Libraries and Setting Parameters:**

```
import yfinance as yf
import pandas as pd
```

- **yfinance**: Used for downloading historical stock data from Yahoo Finance.
- **pandas**: Essential for data manipulation and analysis.

```
bank_stocks = ['HDFCBANK.NS', 'SBIN.NS', 'ICICIBANK.NS',
'AXISBANK.NS', 'KOTAKBANK.NS']
start_date = '2013-01-01'
end_date = '2023-12-25'
```

- **bank_stocks**: List of stock tickers for Indian banks. These represent the dataset's subjects.
- **start_date**, **end_date**: Define the time range for the historical data.

2. **Downloading and Preprocessing Stock Data :**

```
individual_stock_data = {}
for symbol in bank_stocks:
    print(f"Downloading data for {symbol}")
    stock_df = yf.download(symbol, start=start_date, end=end_date)
    stock_df.fillna(method='ffill', inplace=True)
    stock_df.to_csv(f'{symbol}.csv')
    individual_stock_data[symbol] = stock_df
```

- **For Loop**: Iterates through each stock ticker.
- **yf.download()**: Fetches historical data for each ticker.
- **fillna()**: Forward fills missing data points to maintain data integrity.
- **to_csv()**: Saves each stock’s processed data as a CSV file.
- **Dictionary Storage**: Each DataFrame is stored in a dictionary with the ticker as the key.

3. Technical Indicators with TA-Lib

```
import numpy as np
from ta import add_all_ta_features
from statsmodels.tsa.stattools import acf
```

- **TA-Lib**: Used for calculating various technical indicators.
- **acf**: Autocorrelation function from statsmodels, to analyze the correlation of the series with itself.

3. Adding Technical Indicators

```
df = add_all_ta_features(df, open="Open", high="High", low="Low",
close="Close", volume="Volume")
```

add_all_ta_features(): Adds multiple technical indicators to the DataFrame, like RSI, MACD, and Bollinger Bands, enhancing the dataset with metrics often used in stock market analysis.

4. LSTM Neural Network Model

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(1,
len(technical_indicators))))
model.add(LSTM(units=50))
model.add(Dense(1, activation='sigmoid'))
```

- **Sequential Model**: Basis for stacking layers in Keras.
- **LSTM Layer**: Captures time-dependency in data. **return_sequences=True** for the first LSTM layer to pass sequences to the next layer.
- **Dense Layer with Sigmoid Activation**: Outputs a probability prediction (between 0 and 1) indicating the likelihood of a stock price increase.

5. Model Compilation and Training

```
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1)
```

- **compile()**: Configures the model with a loss function and optimizer.
- **binary_crossentropy** is suitable for binary classification.
- **fit()**: Trains the model.
- **epochs** define the number of times the model will work through the entire dataset.

6. Predicting and Evaluating the Model

```
yy_pred = model.predict(X_test)
classification_reports[symbol] = classification_report(y_test, y_pred,
output_dict=True)
```

- **predict()**: Generates output predictions.
- **classification_report**: Provides a detailed analysis of the model's accuracy, including precision, recall, and F1-score.

```
Classification Report for AXISBANK.NS:
              precision    recall  f1-score     support
0              0.449561  0.823293  0.581560  249.000000
1              0.443038  0.122378  0.191781  286.000000
accuracy       0.448598  0.448598  0.448598    0.448598
macro avg      0.446300  0.472835  0.386671  535.000000
weighted avg   0.446074  0.448598  0.373192  535.000000

Classification Report for HDFCBANK.NS:
              precision    recall  f1-score     support
0              0.481836  0.984375  0.646983  256.000000
1              0.666667  0.028674  0.054983  279.000000
accuracy       0.485981  0.485981  0.485981    0.485981
macro avg      0.574251  0.506524  0.350983  535.000000
weighted avg   0.578224  0.485981  0.338258  535.000000

Classification Report for ICICIBANK.NS:
              precision    recall  f1-score     support
0              0.440000  0.166667  0.241758  264.000000
1              0.494253  0.793358  0.609065  271.000000
accuracy       0.484112  0.484112  0.484112    0.484112
macro avg      0.467126  0.480012  0.425412  535.000000
weighted avg   0.467481  0.484112  0.427815  535.000000

Classification Report for KOTAKBANK.NS:
              precision    recall  f1-score     support
0              0.523220  0.628253  0.570946  269.000000
1              0.528302  0.421053  0.468619  266.000000
accuracy       0.525234  0.525234  0.525234    0.525234
macro avg      0.525761  0.524653  0.519783  535.000000
weighted avg   0.525747  0.525234  0.520069  535.000000

Classification Report for SBIN.NS:
              precision    recall  f1-score     support
0              0.000000  0.000000  0.000000  245.000000
1              0.542056  1.000000  0.703030  290.000000
accuracy       0.542056  0.542056  0.542056    0.542056
macro avg      0.271028  0.500000  0.351515  535.000000
weighted avg   0.293825  0.542056  0.381082  535.000000
```

The classification reports for each of the five bank stocks give us a detailed view of the performance of the LSTM models. Let's break down these results, focusing on the **precision**, **recall**, and **f1-score** for each class (**0** and **1**), as well as the **accuracy** and **macro avg** metrics.

1. **AXISBANK.NS:**
    - The model has a high recall but low precision for class **0**, indicating it correctly identifies most of the actual **0** cases, but also incorrectly labels many **1**s as **0**s.
    - For class **1**, both precision and recall are low, particularly recall, meaning the model struggles to correctly identify **1** cases.
    - The overall accuracy is around 44.86%, and the macro average f1-score is about 38.67%, suggesting the model performs moderately but is biased towards predicting **0**.
2. **HDFCBANK.NS:**
    - High recall but very low precision for class **0**, indicating a similar trend to AXISBANK.NS where the model predominantly predicts **0**.
    - Extremely low recall for class **1**, showing the model almost never predicts **1** correctly.
    - The accuracy is about 48.60%, but the model's effectiveness in distinguishing class **1** is poor, as reflected in the weighted average f1-score.
3. **ICICIBANK.NS:**
    - Moderate recall for class **1** and low recall for class **0**. This model seems to be better at predicting **1** than **0**.
    - The accuracy is close to 48.41%, with a macro average f1-score of about 42.54%, indicating a moderate performance with some bias.
4. **KOTAKBANK.NS:**
    - More balanced precision and recall for both classes compared to other models.
    - The accuracy is approximately 52.52%, with a macro average f1-score of around 51.98%. This suggests a relatively more balanced and better overall performance.
5. **SBIN.NS:**
    - This model shows a peculiar behavior: it has high recall for class **1** but zero precision and recall for class **0**. It seems to predict **1** for almost all instances.
    - The accuracy is 54.21%, but this is misleading as the model fails to identify any **0** cases correctly.

Conclusions:

- The models for AXISBANK.NS, HDFCBANK.NS, and ICICIBANK.NS show a bias towards predicting class **0** (with varying degrees), resulting in high recall but low precision for class **0**.
- KOTAKBANK.NS's model appears to be the most balanced among the five, with a more even distribution of precision and recall across classes.
- The model for SBIN.NS is heavily biased towards predicting class **1**, failing to identify any **0** cases.

- The accuracy and macro average f1-scores suggest moderate performance, but these models may need further tuning and possibly more complex architectures or additional features to improve their predictive power.
- The varying performances across different stocks highlight the unique characteristics and behaviors inherent in each stock's data.

Below are Improvements I am thinking of as of now:

- Experiment with different LSTM architectures or additional layers to capture more complex patterns.
- Consider using more or different technical indicators as features.
- Adjust the training parameters, like the number of epochs and batch size..
- Implement cross-validation to ensure the models' robustness.