📘 FINAL PROJECT REPORT

Project Title: LSTM-Based Stock Price Predictor with ARIMA and SARIMA Comparative Models

GitHub Repo: LSTM-STOCK-PREDICTOR

📝 Abstract

This project presents a hybrid time-series forecasting system that combines **deep learning (LSTM)** with classical statistical models (**ARIMA** and **SARIMA**) to predict stock prices. The system is deployed using **Streamlit** and demonstrates how different models behave on stock market data (specifically AAPL stock). The goal is to compare the performance of these models and visualize their predictions.

🎯 Objectives

- To implement an LSTM-based model for stock price prediction.
- To compare LSTM with classical time-series models: ARIMA and SARIMA.
- To visualize predictions and error metrics in an interactive Streamlit app.
- To enable data upload and real-time forecasting on new stock data.

📁 Project Structure

```
LSTM-STOCK-PREDICTOR/
├── app/
│   └── streamlit_app.py
├── model/
│   ├── lstm_model.py
│   ├── arima_model.py
│   └── sarima_model.py
├── data/
│   └── AAPL.csv
├── requirements.txt
└── README.md
```

. 🛠️ Technologies Used

- **Frontend**: Streamlit
- **Backend & Model**: Python (Pandas, NumPy, Scikit-learn, TensorFlow, Keras, Statsmodels)
- **Visualization**: Matplotlib, Plotly
- **Version Control**: Git & GitHub
- **Hosting Environment**: Streamlit Cloud

📊 Data Used

- **Dataset**: AAPL stock data in CSV format.
- **Fields**: Date, Open, High, Low, Close, Volume
- **Preprocessing**:
    - Missing values dropped
    - Only Close prices used for modeling
    - Normalization (MinMaxScaler) applied for LSTM input

---

1. Models Implemented

1.1. 🔁 LSTM (Long Short-Term Memory)

- Implemented using TensorFlow/Keras.
- Sequence-to-one architecture: uses previous time steps to predict the next.
- Features:
    - MinMaxScaler normalization
    - 80-20 Train-Test split

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

- Loss: mean_squared_error
- Optimizer: adam
- Epochs: configurable in code (e.g., 10)
- Metrics used: RMSE

1.2. 📈 ARIMA (AutoRegressive Integrated Moving Average)

- Implemented with statsmodels.tsa.arima_model.ARIMA
- Best (p,d,q) parameters chosen using AIC/BIC.
- Limitations handled by differencing non-stationary series.

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(train_data, order=(5,1,0))
model_fit = model.fit()
```

1.3. 📊 SARIMA (Seasonal ARIMA)

- Extended version of ARIMA for handling seasonality.
- Implemented with SARIMAX from statsmodels.
- Seasonal order defined as (P, D, Q, s) in addition to (p, d, q).

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
model = SARIMAX(train_data, order=(1,1,1), seasonal_order=(1,1,1,12))
model_fit = model.fit()
```

2. Evaluation Metrics

Model	RMSE (example)
LSTM	~3.02
ARIMA	~3.85
SARIMA	~3.74

RMSE (Root Mean Squared Error) is the primary metric used to compare performance.

---

3. Streamlit Web App Features

- Upload custom stock CSV files
- View historical data charts
- Compare LSTM vs ARIMA vs SARIMA predictions
- Interactive sliders and inputs
- Real-time graphs using Matplotlib and Plotly

---

4. Execution Flow

1. **User selects a stock dataset** from data/AAPL.csv
2. **Data is cleaned and scaled**
3. **User chooses prediction model (LSTM, ARIMA, or SARIMA)**
4. **Prediction results and future forecasts** are plotted
5. **Streamlit app displays charts & metrics**

---

5. Warnings and Errors Handled

- **ValueWarning**: Unsupported index — mitigated by converting Date column to DatetimeIndex
- **UserWarnings in SARIMA**: Non-stationary starting parameters — initialized to zero
- **Missing CUDA libraries**: Ignored since CPU-only execution is sufficient for the app
- **FileNotFoundError:** Resolved by mounting the correct path /data/AAPL.csv
- **ModuleNotFoundError:** Fixed by adding statsmodels to requirements.txt
- **TensorFlow GPU warnings:** Safe to ignore if using CPU only

---

6. Deployment

- **Local**: via streamlit run app/streamlit_app.py
- **Cloud**: Deployable on platforms like Streamlit Cloud, Heroku, or Render
- **Dependencies**: Managed with requirements.txt

7. Requirements

```
txt
CopyEdit
numpy
pandas
scikit-learn
tensorflow
keras
streamlit
matplotlib
plotly
statsmodels
```

---

7. Conclusion

This project successfully demonstrates the power of LSTM in time-series forecasting and contextualizes its performance alongside ARIMA and SARIMA models. The interactive app makes it easy for users to experiment with different models and data inputs.

---

8. Future Enhancements

- Add Prophet and XGBoost models for comparison
- Integrate news sentiment analysis
- Improve hyperparameter tuning using AutoML