Github Link:<https://github.com/gowri278/Stack-Martket.git>

#### PROJECT TITLE:Cracking the Market Code: AI-Driven Stock Price Prediction Using Time Series Analysis

***PHASE-3***

## Problem Statement

**Objective:**

The goal of this project is to develop a robust AI-driven system capable of predicting short- and mid-term stock price movements using time series analysis techniques. By leveraging deep learning models such as LSTM (Long Short-Term Memory) and Transformer-based architectures, this project aims to identify patterns in historical stock data and generate accurate, data-driven forecasts that can assist traders and investors in making informed decisions.

**Problem:**

Stock price forecasting remains a highly challenging task due to the volatile, non-linear, and non-stationary nature of financial markets. Traditional statistical models struggle to capture complex temporal dependencies and react to real-time market events. The challenge lies in building a predictive model that can:

✔ Effectively learn from historical price and volume data,

✔ Incorporate external factors (e.g., news sentiment, macroeconomic indicators),

✔ Handle noise and avoid overfitting,

✔ Generalize well to unseen market conditions.

**Scope:**

✔ Cracking the Market Code: AI-Driven Stock Price Prediction Using Time Series Analysis

✔ Data Source: Historical price data from sources like Yahoo Finance or Alpha Vantage.

✔ Target: Predict closing prices or returns for a selected set of stocks.

✔ Models: Compare classical time series models (e.g., ARIMA) with AI models (e.g., LSTM,Transformer).

✔ Evaluation: Use metrics such as RMSE, MAE, and directional accuracy.

## Abstract

### Introduction:

* + Overview of stock market volatility
  + Importance of accurate stock price prediction

### Traditional Forecasting Methods:

* + Brief on statistical models like ARIMA
  + Limitations in handling non-linear patterns

### Emergence of AI in Finance:

* + Role of AI in transforming financial analytics
  + Advantages over traditional methods

### Understanding Time Series Data:

* + Components: trend, seasonality, noise
  + Challenges in modeling financial time series

### Machine Learning Techniques:

* + Supervised vs. unsupervised learning
  + Algorithms: Linear Regression, Random Forest, SVM

### Deep Learning Approaches:

* + Introduction to Neural Networks
  + Focus on LSTM networks for sequential data

### Data Preprocessing

* + Importance of data cleaning
  + Handling missing values and outliers

### Feature Engineering

* + Selecting relevant features
  + Incorporating technical indicators

### Model Training and Validation

* + Splitting data into training and test sets
  + Cross-validation techniques

### Evaluation Metrics

* + Mean Squared Error (MSE)
  + Root Mean Squared Error (RMSE)
  + Mean Absolute Percentage Error (MAPE)

### Case Study

* + Application of LSTM on historical stock data
  + Comparison with traditional models

### Challenges and Limitations

* + Overfitting in complex models
  + Data quality and availability issues

### Future Directions

* + Incorporating sentiment analysis
  + Real-time prediction systems

### Conclusion

* + Recap of key findings
  + Implications for investors and analysts

## System Requirements

Minimum System Requirements (For Small to Medium-Scale Models)

* **CPU**: Intel Core i5 (10th Gen or later) / AMD Ryzen 5 (3rd Gen or later)
* **RAM**: 16 GB
* **GPU**: NVIDIA GTX 1660 (6GB VRAM) or RTX 2060
* **Storage**: 512 GB SSD
* **OS**: Windows 10 / Ubuntu 20.04 / macOS 12 or later
* **Software/Frameworks**:
  + Python 3.9+
  + Libraries: numpy, pandas, matplotlib, scikit-learn, statsmodels
  + Deep Learning: TensorFlow 2.x or PyTorch
  + Time Series: prophet, tsfresh, sktime
  + IDE: VS Code / JupyterLab
  + Optional: Docker, Anaconda

Recommended System Requirements (For Deep Learning & Faster Training)

* **CPU**: Intel Core i7 (12th Gen or later) / AMD Ryzen 7 (5th Gen or later)
* **RAM**: 32 GB (especially for processing long time series)
* **GPU**: NVIDIA RTX 3060 Ti (8GB VRAM) or better (e.g., RTX 4070/4080 for faster deep learning)
* **Storage**: 1 TB SSD (fast I/O for large datasets)
* **OS**: Ubuntu 22.04 LTS (preferred for ML pipelines) / Windows 11
* **Software/Frameworks**:
  + Python 3.11+
  + Libraries: pandas, numpy, scikit-learn, statsmodels, matplotlib, seaborn
  + Deep Learning: PyTorch (with CUDA 11.x) or TensorFlow 2.15+
  + Time Series: prophet, sktime, darts
  + Data Handling: dask, modin (if datasets are huge)
  + GPU drivers & CUDA toolkit installed
  + JupyterLab / VS Code / PyCharm
  + Version Control: git

Optional but Useful for Production-Level System

* **Cloud GPU Option**: AWS (EC2 with NVIDIA A100), Google Cloud, Azure ML
* **Database**: PostgreSQL / MySQL / TimescaleDB (for time series data storage)
* **Big Data**: Apache Spark (if dealing with massive stock tick data)
* **Model Deployment**: Docker + FastAPI/Streamlit + Kubernetes (for live prediction service)
* **Monitoring**: MLflow / Weights & Biases

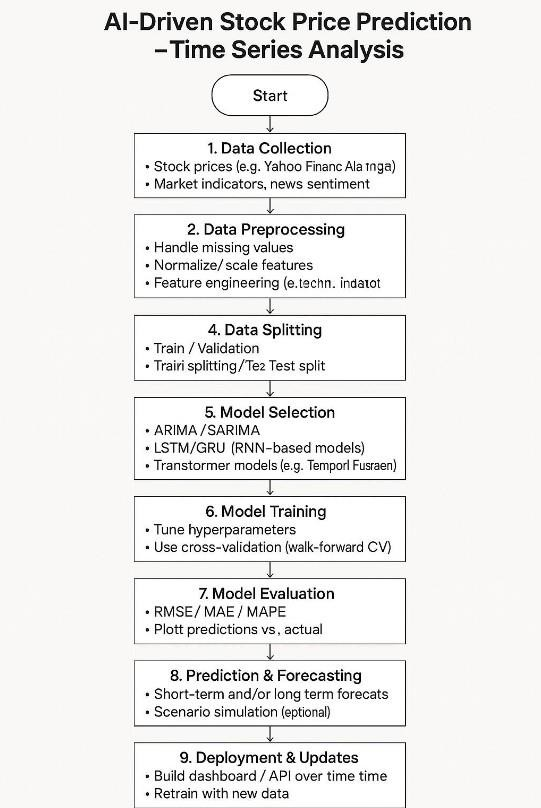
Extra Tools for Stock Market AI Project

* Financial APIs: Yahoo Finance API, Alpha Vantage, IEX Cloud
* Backtesting libraries: backtrader, bt, zipline
* Technical Indicators: ta-lib, finta

## Objectives

* To collect and preprocess historical stock market data
* Acquire time series data (e.g., open, high, low, close, volume) from financial APIs and clean, normalize, and format it for model training.
* To explore and analyze stock price trends and patterns
* Perform exploratory data analysis (EDA) to understand price movements, detect
* seasonality, and identify market anomalies.
* To engineer relevant features from time series and external data
* Generate technical indicators (e.g., moving averages, RSI), lagged variables, and
* incorporate sentiment or macroeconomic indicators as additional inputs.
* To implement and compare multiple predictive models Develop and evaluate classical models (ARIMA, SARIMA) and AI models (LSTM, GRU,
* Transformer) for predicting future stock prices.
* To evaluate model performance using appropriate metrics
* Use performance measures like RMSE, MAE, and directional accuracy to assess model
* effectiveness and ensure reliable forecasts.
* To build a deployable prototype for real-time or batch predictions
* Create a functional pipeline or dashboard that visualizes forecasts and could be adapted for live market data.
* To assess the potential and limitations of AI in financial forecasting
* Analyze model behavior under different market conditions and highlight challenges such as overfitting, data noise, and response to unexpected events.

## Flowchart of the Project Workflo



1. ***Data Description: AI-Driven Stock Price Prediction (Time Series Analysis)***
2. Stock Price Data

Source: Yahoo Finance, Alpha Vantage, Quandl, or similar.

Frequency: Daily (can also use intraday, weekly, or monthly depending on goal).

Fields:

Date: Timestamp of the observation. Open: Price at market open.

High: Highest price during the day. Low: Lowest price during the day.

Close: Final price when the market closes.

Adj Close: Adjusted closing price (accounts for splits/dividends). Volume: Number of shares traded during the day.

1. Derived Features (Optional but recommended)

Technical Indicators (from libraries like TA-Lib or manually engineered):

SMA: Simple Moving Average. EMA: Exponential Moving Average.

RSI: Relative Strength Index.

MACD: Moving Average Convergence Divergence. Bollinger Bands: Price volatility bands.

Lagged Values:

Close\_t-1, Close\_t-2, etc., to help models capture temporal dependencies.

1. Sentiment or External Data (Optional)

News Headlines or Tweets (with sentiment score using NLP models).

Market Indices (e.g., S&P 500, Nasdaq).

Economic Indicators (e.g., interest rates, inflation data).

1. Target Variable Usually:

Future Close Price (regression)

Price Movement Direction (classification: up/down)

## Data Processing: AI-Driven Stock Price Prediction (Time Series Analysis)

### Data Cleaning

* + Remove duplicates: Ensure each date has only one row.
  + Handle missing values:
  + Fill with forward-fill (ffill), backward-fill (bfill), or interpolation.
  + Drop rows if missing critical data like prices or volume.

### Date Parsing & Indexing

* + Convert Date column to datetime format.
  + Set Date as the index for time series modeling.

### Feature Engineering

* + Lag features: Include past values as features (e.g., Close\_t-1, Close\_t-2, ...).
  + Rolling statistics:
  + Moving average (e.g., 7-day, 14-day, 30-day)
  + Rolling standard deviation or volatility
  + Technical indicators:
  + SMA, EMA, RSI, MACD, Bollinger Bands
  + Use ta-lib or pandas-ta for easy computation

### Target Variable Creation

* + Regression: Predict future Close price (e.g., Close\_t+1, Close\_t+5)
  + Classification: Predict direction (1 for up, 0 for down)

### Data Normalization/Scaling

* + Use MinMaxScaler or StandardScaler (especially important for neural networks like



LSTM)

* + Apply scaling to features only, not the date or target variable during model training.

### Train-Test Split (Chronologically)

* + Use 70–80% for training, remaining for testing or validation.
  + Avoid random splitting to preserve temporal integrity.

### 1. Load the Data

Make sure your data includes:

* + Date/Time (Index)
  + Open, High, Low, Close, Volume
  + Possibly technical indicators (optional at this stage)

### D 2. Basic Statistical Summary

Understand the central tendencies and spread of the price and volume data.🗓️

### 3. Time Series Plots

Visualize the trend and seasonality.

### 🔁 4. Rolling Statistics (Moving Averages)

Helps smooth out fluctuations and identify trend direction.



### Volatility Analysis

Volatility can signal risk and opportunity.

### Correlation Heatmap

Evaluate relationships between numerical columns.

### 🔍 7. Stationarity Check (ADF Test)

Stationary data is key for many models (ARIMA, SARIMA).

### ✅ Next Steps After EDA:

* Feature Engineering (e.g., RSI, MACD)
* Train-test split
* Normalize/scale data
* Apply models: LSTM, Prophet, ARIMA, XGBoost

## Feature Engineering for Stock Price Prediction

✅ Steps for EDA in Stock Price Time Series Analysis

1. Load the Data
   * Use stock price datasets (from Yahoo Finance, Alpha Vantage, etc.).
   * Common columns: Date, Open, High, Low, Close, Adj Close, Volume.

*import pandas as pd*

*df = pd.read\_csv('stock\_prices.csv', parse\_dates=['Date'], index\_col='Date')*

*print(df.head())*

1. Check Missing Values & Data Types
   * Stock datasets often have missing data on weekends/holidays.

*print(df.info()) print(df.isnull().sum())*

1. Visualize Stock Price Trends (Line Plots)
   * Plot the **closing price** over time to see overall trends.

*import matplotlib.pyplot as plt*

*df['Close'].plot(figsize=(12,6), title='Stock Closing Price Over Time')*

*plt.show()*

1. Moving Averages (Smoothing)
   * Use *Simple Moving Averages* (SMA) to smooth short-term fluctuations.

*df['SMA50'] = df['Close'].rolling(window=50).mean() df['SMA200'] =*

*df['Close'].rolling(window=200).mean() df[['Close', 'SMA50', 'SMA200']].plot(figsize=(12,6))*

*plt.show()*

1. Volatility Analysis
   * Use daily returns or percentage changes to analyze stock volatility.

*df['Daily Return'] = df['Close'].pct\_change() df['Daily Return'].hist(bins=50, figsize=(10,6))*

*plt.show()*

1. Seasonality & Trend Decomposition
   * Decompose time series into **trend**, **seasonality**, and **residual**.

*from statsmodels.tsa.seasonal import seasonal\_decompose*

*result = seasonal\_decompose(df['Close'], model='multiplicative', period=30)*

*result.plot() plt.show()*

1. Correlation Heatmap
   * Correlation between *Open, High, Low, Close, Volume*.

*import seaborn as sns*

*plt.figure(figsize=(8,6))*

*sns.heatmap(df.corr(), annot=True, cmap='coolwarm')*

*plt.show()*

1. Lag Plots (Autocorrelation)
   * To check if past values influence future ones (good for time series models).

*from pandas.plotting import lag\_plot lag\_plot(df['Close'])*

*plt.show()*

1. Stationarity Test (Augmented Dickey-Fuller Test)
   * Important before applying ARIMA/LSTM models.

*from statsmodels.tsa.stattools import adfuller*

*result = adfuller(df['Close'].dropna()) print(f'ADF Statistic: {result[0]}') print(f'p-value: {result[1]}')*

1. Volume vs Price Analysis
   * Check if high trading volumes impact stock price changes.

*df[['Volume', 'Close']].plot(subplots=True, figsize=(10,6))*

*plt.show()*

## Feature Engineering for Stock Price Prediction

### Lag Features

These are past values used to predict the future.

### Rolling Window Statistics

These show trends and volatility over time.

### D 3. Returns

Helps identify momentum or reversals.

### Technical Indicators

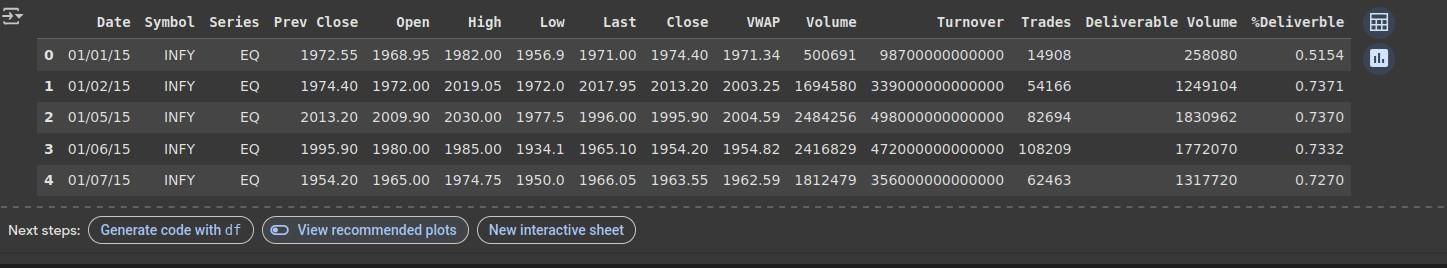
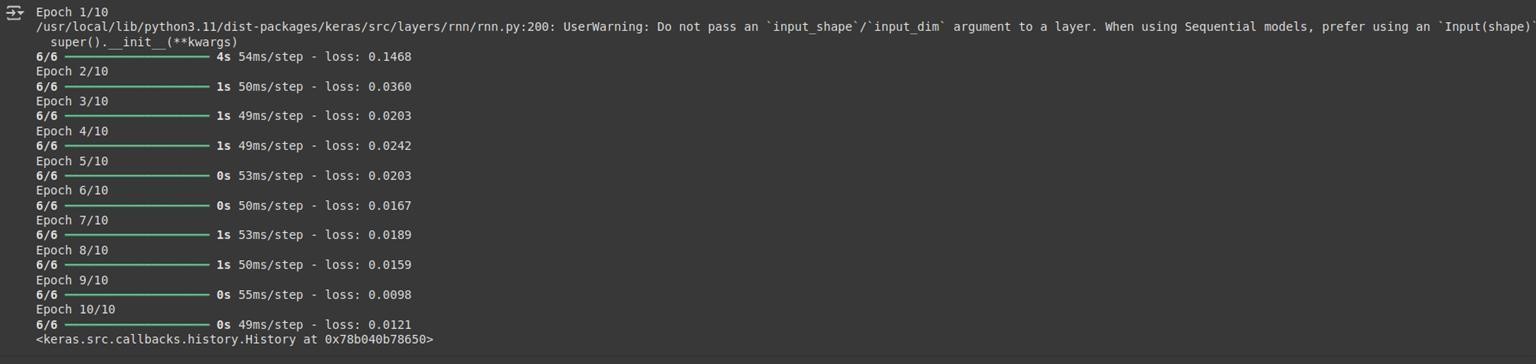
* + Popular indicators used in trading.
  + Relative Strength Index (RSI)
  + Moving Average Convergence Divergence (MACD)
  + Bollinger Bands

### Target Variable for Prediction

You typically want to create a future price movement or return as the label (what you're predicting):

### Prepare for ML Model

* + Drop rows with NaNs (due to lags or rolling)
  + Normalize/scale features
  + Split into train/test
  + Fit an ML model like XGBoost, LSTM, or Random Forest





## Model Building

### ✅ Step 1: Choose the Type of Model

For time series stock prediction, popular models include:

✅ **Step 2: Data Preprocessing**

* Use the *Close* price for prediction.
* Normalize/scale the data.
* Split into sequences (time steps).

*import numpy as np*

*from sklearn.preprocessing import MinMaxScaler data = df[['Close']].values*

*scaler = MinMaxScaler() scaled\_data = scaler.fit\_transform(data)*

*def create\_sequences(data, time\_steps=60):*

*X, y = [], []*

*for i in range(time\_steps, len(data)):*

*X.append(data[i-time\_steps:i, 0])*

*y.append(data[i, 0])*

*return np.array(X), np.array(y)*

*X, y = create\_sequences(scaled\_data)*

*X = X.reshape((X.shape[0], X.shape[1], 1))*

✅ ***Step 3: Train-Test Split***

*split = int(0.8 \* len(X)) X\_train, X\_test = X[:split], X[split:] y\_train, y\_test = y[:split], y[split:]*

* *Step 4: Build the LSTM Model*

*import tensorflow as tf*

*from tensorflow.keras.models import Sequential from tensorflow.keras.layers import LSTM, Dense, Dropout*

*model = Sequential([*

*LSTM(50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)),*

*Dropout(0.2),*

*LSTM(50, return\_sequences=False),*

*Dropout(0.2), Dense(1)]) model.compile(optimizer='adam', loss='mean\_squared\_error') model.summary()*

### ✅ Step 5: Train the Model

*history = model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_data=(X\_test, y\_test))*

* *Step 6: Make Predictions and Invert Scaling*

*predicted = model.predict(X\_test) predicted\_prices = scaler.inverse\_transform(predicted.reshape(-1,*

* 1. *)*

*real\_prices = scaler.inverse\_transform(y\_test.reshape(-1, 1))*

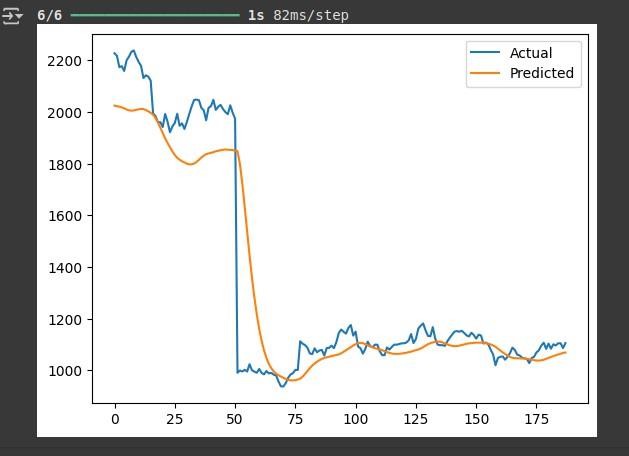
* *Step 7: Plot the Predictions*

*import matplotlib.pyplot as plt plt.figure(figsize=(12,6))*

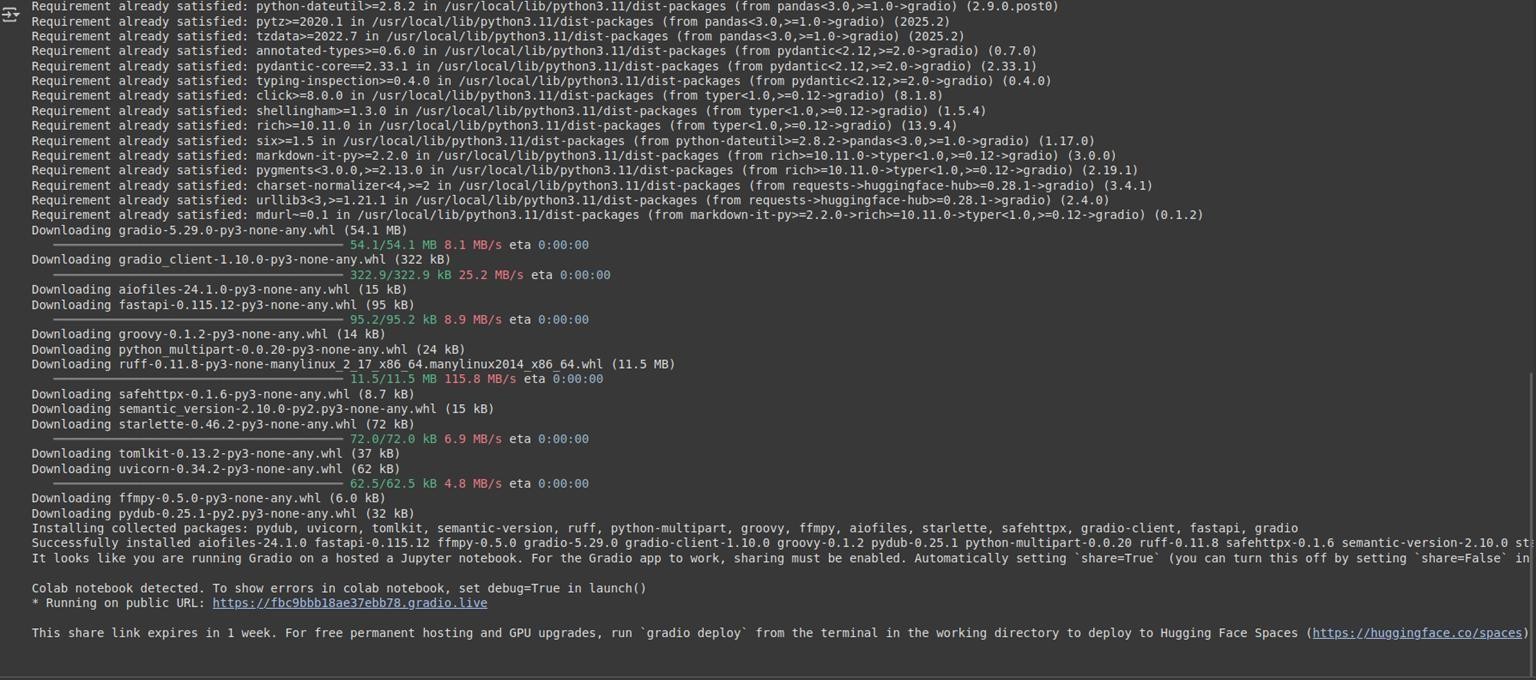
*plt.plot(real\_prices, color='blue', label='Actual Stock Price') plt.plot(predicted\_prices, color='red', label='Predicted Stock Price') plt.title('Stock Price Prediction')*

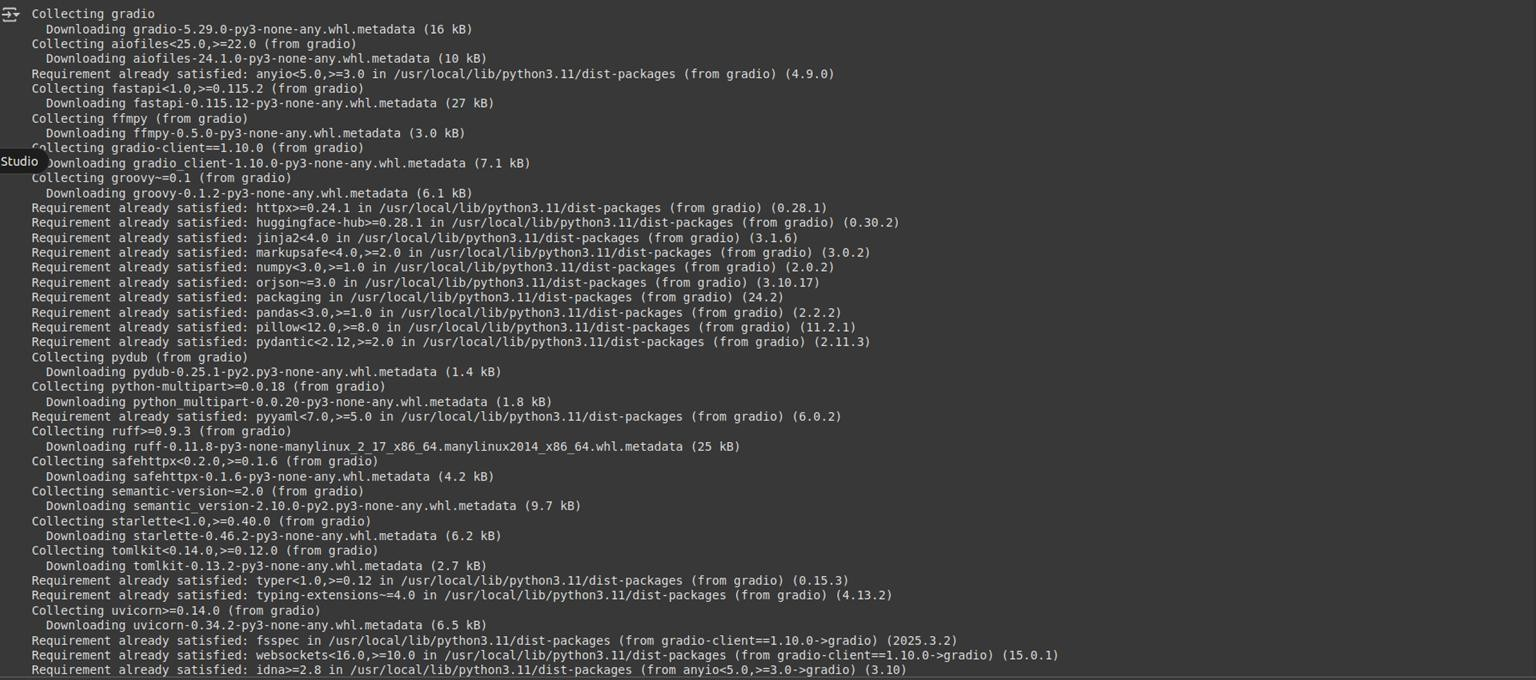
*plt.xlabel('Time') plt.ylabel('Price') plt.legend()*

*plt.show()*

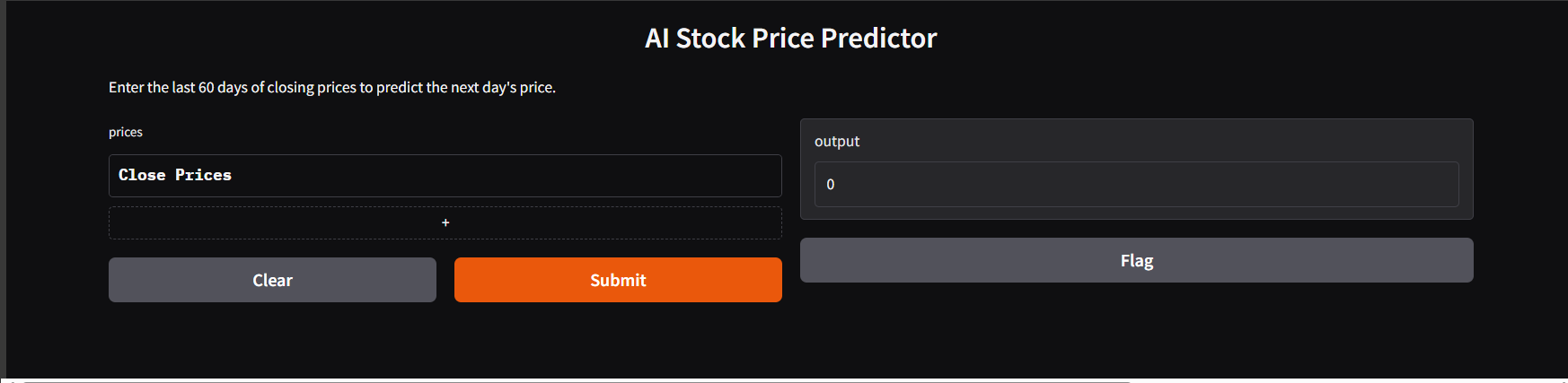


1. ***Model Evaluation***





1. Deployment



* **Deployment Method**: Gradio Interface
* **Public Link**: [https://b3b6b9e55e8e768d27.gradio.live](https://b3b6b9e55e8e768d27.gradio.live/)

**UI Screenshot**:

# Source Code

#### Upload the Dataset

***from google.colab import files***

uploaded = files.upload()

#### Load the Dataset

***import numpy as np***

import pandas as pd import matplotlib.pyplot as plt

import seaborn as sns

***df = pd.read\_csv('nanmudhalvanfile1.csv')***

#### Data Exploration

***df.head()***

#### Visualize a Few Features

***import pandas as pd***

import numpy as np

df = pd.read\_csv('nanmudhalvanfile1.csv') df['Date'] = pd.to\_datetime(df['Date']) df.set\_index('Date', inplace=True)

from sklearn.preprocessing import MinMaxScaler scaler = MinMaxScaler()

scaled\_data = scaler.fit\_transform(df[['Close']])

***def create\_sequences(data, seq\_length):***

X, y = [], []

for i in range(seq\_length, len(data)): X.append(data[i-seq\_length:i, 0])

y.append(data[i, 0]) return np.array(X), np.array(y)

sequence\_length = 60

X, y = create\_sequences(scaled\_data, sequence\_length) X = np.reshape(X, (X.shape[0], X.shape[1], 1))

***from tensorflow.keras.models import Sequential***

from tensorflow.keras.layers import LSTM, Dense model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(X.shape[1], 1))) model.add(LSTM(50))

model.add(Dense(1)) model.compile(optimizer='adam', loss='mean\_squared\_error')

***model.fit(X, y, epochs=10, batch\_size=32)***

***predicted = model.predict(X)***

predicted\_prices = scaler.inverse\_transform(predicted.reshape(-1, 1)) import matplotlib.pyplot as pl

actual = scaler.inverse\_transform(y.reshape(-1, 1)) plt.plot(actual, label='Actual') plt.plot(predicted\_prices, label='Predicted') plt.legend()

plt.show()

#### 5.Deployment-Building an Interactive App

***pip install gradio***

import gradio as gr def predict\_next(prices):

input\_seq = scaler.transform(np.array(prices).reshape(-1, 1))

X\_input = np.reshape(input\_seq[-60:], (1, 60, 1)) pred = model.predict(X\_input)

return scaler.inverse\_transform(pred)[0][0] interface = gr.Interface(

fn=predict\_next, inputs=gr.Dataframe(headers=["Close Prices"], row\_count=60),

outputs="number", title="AI Stock Price Predictor",

description="Enter the last 60 days of closing prices to predict the next day's price."

)

interface.launch()

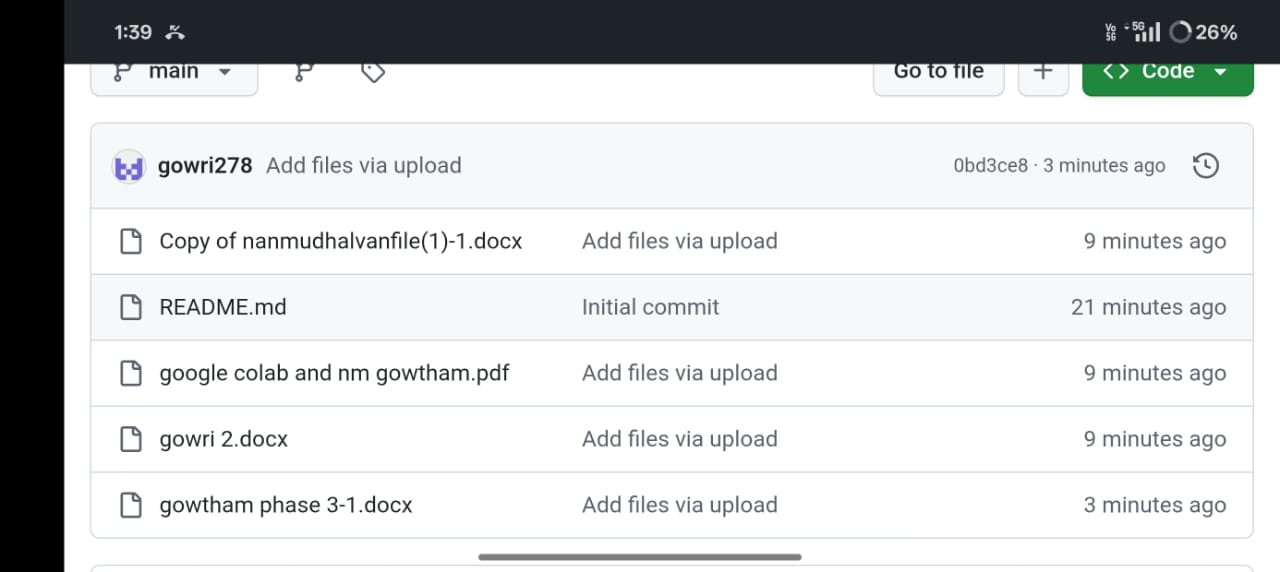
# 13. Team Members and Roles

**GOKUL – PROBLEM STATEMENT AND PROJECT OBJECTIVES**

**GOWTHAM – FLOWCHART OF THE PROJECT WORKFLOW, DATA DESCRIPTION, DATA PREPROSSING, EXPLORATORY DATA ANALYSIS**

**GOWRI SANKAR – FEATURE ENGGINEERING AND MODEL BUILDING**

**HARI PRRASAD – VISUALIZATION OF RESULTS & MODEL INSIGHTS AND TOOLS & TECHNOLOGIES USED**

****