PROJECT REPORT: Google Stock Price Modeling

PROJECT OBJECTIVE:

Predict Google (GOOGL) closing stock prices using deep learning models (RNN, LSTM, GRU)

trained on historical data (2010-2022) and evaluated on 2023 test data.

LIBRARIES USED:

1. DATA MANIPULATION & ANALYSIS:

- pandas: Data loading, manipulation, and analysis
- numpy: Numerical computations and array operations

2. VISUALIZATION:

- matplotlib: Static plotting and visualization
- seaborn: Statistical data visualization
- plotly: Interactive charts (Scatter, Line, Candlestick, Box plots, etc.)

3. MACHINE LEARNING & PREPROCESSING:

- scikit-learn (sklearn): MinMaxScaler, StandardScaler, train_test_split, metrics (RMSE, MAE, R^2)
- tensorflow/keras: Deep learning framework for building neural networks
- statsmodels: Time-series analysis, seasonal decomposition, ACF/PACF, ADF test

4. TIME-SERIES UTILITIES:

- pmdarima: AutoARIMA and time-series forecasting tools
- ta: Technical analysis indicators

5. UTILITIES:

- datetime: Date and time handling
- itertools: Hyperparameter grid generation
- os, warnings: System and warning management

DATA PIPELINE:

STEP 1 - DATA LOADING:

- Training data: Google_Stock_Train (2010-2022).csv
- Test data: Google_Stock_Test (2023).csv
- Columns: Date, Open, High, Low, Close, Volume

STEP 2 - EXPLORATORY DATA ANALYSIS (EDA):

- Time-series plots of closing prices
- Volume analysis (daily, monthly aggregation)
- Distribution analysis (returns, price changes)
- Correlation heatmap
- Moving averages (50-day, 10-day)
- Candlestick charts
- Year/month/weekday patterns

STEP 3 - PREPROCESSING:

- Convert Date to datetime format
- Extract temporal features: Year, Month, Day, Weekday, IsMonthStart, IsMonthEnd
- Select Close price (column index 4) as target variable
- Normalize Close price using MinMaxScaler (range 0-1)
- Create supervised sequences using sliding window (60 timesteps):
  * Input: previous 60 days' closing prices
  * Output: next day's closing price
- Shape transformation: (samples, 60) → (samples, 60, 1) for LSTM/RNN/GRU input

STEP 4 - TRAIN/VALIDATION SPLIT:

- Training: 88% of sequences
- Validation: 12% of sequences (via validation_split parameter)

MODEL ARCHITECTURES:

MODEL 1 - SIMPLE RNN (SimpleRNN):

Architecture:

SimpleRNN(300 units) + Dropout(0.2)

→ SimpleRNN(100 units) + Dropout(0.2)

→ SimpleRNN(100 units) + Dropout(0.2)

→ SimpleRNN(100 units) + Dropout(0.2)

→ Dense(1)

Configuration:

- Optimizer: Adam
- Loss: Mean Squared Error (MSE)
- Epochs: 100
- Batch size: 32
- Callbacks: EarlyStopping (patience=15), ReduceLROnPlateau (patience=7)

MODEL 2 - LONG SHORT-TERM MEMORY (LSTM) - Advanced:

Architecture:

Conv1D(32 filters, kernel=3, causal padding) + BatchNorm + Dropout(0.12)

→ Bidirectional LSTM(256 units) + LayerNorm + Dropout(0.18)

→ LSTM(128 units) + LayerNorm + Dropout(0.15) [with residual connection]

→ LSTM(128 units) + Dropout(0.12)

→ Attention Block (time-distributed weights)

→ GlobalAveragePooling1D

→ Dense(64, ReLU) + Dropout(0.12)

→ Dense(1, Linear)

Features:

- Conv1D front-end captures local temporal patterns
- Bidirectional LSTM processes sequences in both directions
- Residual connections improve gradient flow
- Attention mechanism weights important timesteps
- Layer normalization stabilizes training
- L2 regularization (1e-5) prevents overfitting

Configuration:

- Optimizer: Adam (learning_rate=0.001)
- Loss: Mean Squared Error (MSE)
- Metrics: MAE
- Epochs: 150
- Batch size: 32
- Callbacks: EarlyStopping (patience=20), ReduceLROnPlateau (patience=8), ModelCheckpoint

MODEL 3 - GATED RECURRENT UNIT (GRU):

Architecture:

GRU(300 units) + Dropout(0.2)
→ GRU(100 units) + Dropout(0.2)
→ GRU(100 units) + Dropout(0.2)
→ GRU(100 units) + Dropout(0.2)
→ Dense(1)

Configuration:

- Optimizer: Adam
- Loss: Mean Squared Error (MSE)
- Metrics: MAE
- Epochs: 150
- Batch size: 32

- Callbacks: EarlyStopping (patience=20), ReduceLROnPlateau (patience=8), ModelCheckpoint

EVALUATION METHODOLOGY:

1. TEST SET CREATION:
   - Combine last 60 days of training data + all test data
   - Create sequences using same scaler fitted on training data
   - Ensure consistency between train and test preprocessing

2. PREDICTION GENERATION:
   - Pass test sequences through trained models
   - Inverse transform scaled predictions back to original price range

3. METRICS COMPUTED:
   - RMSE: Root Mean Squared Error (penalizes large errors)
   - MAE: Mean Absolute Error (average magnitude of errors)
   - R^2: Coefficient of determination (model fit quality)
   - MAPE: Mean Absolute Percentage Error (percentage-based error)

4. VISUALIZATION:
   - Actual vs. Predicted prices (line plots)
   - Ensemble average (mean of all model predictions)
   - Residual analysis
   - Error distribution

HYPERPARAMETER GRID SEARCH:

Tested combinations:

- Sequence lengths: [30, 60, 90] days
- Batch sizes: [16, 32]
- Learning rates: [0.001, 0.0005]

KEY RESULTS INTERPRETATION:

- Lower MSE/RMSE → Better model accuracy
- R^2 close to 1.0 → Model captures trend well
- R^2 close to 0.0 → Model performs worse than baseline
- Ensemble predictions often reduce overfitting compared to single model

---

BEST PRACTICES IMPLEMENTED:

✓ Reproducibility: Seeds set for NumPy and TensorFlow

✓ Proper scaling: MinMaxScaler fitted on train data only

✓ Regularization: Dropout, L2 regularization, early stopping

✓ Callbacks: Monitor validation loss, reduce learning rate on plateau

✓ Model checkpoints: Save best weights during training

✓ Sequence consistency: Same window size (60) for train and test

✓ Temporal splitting: No data leakage (test data is future dates)