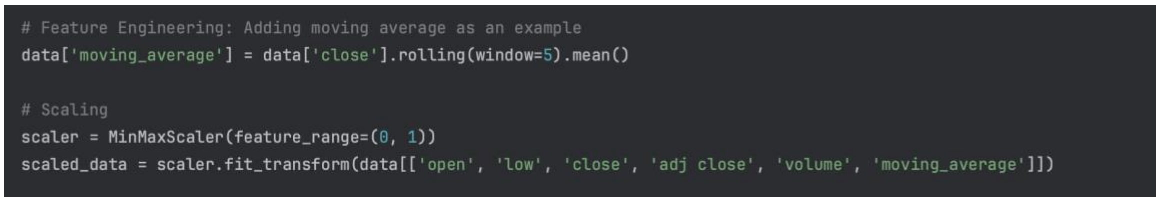
Report on Stock Prediction Models using RNN and CNN

1. Introduction and Objective:

This project embarks on a sophisticated journey to predict the stock prices of Apple Inc. (AAPL), leveraging cutting-edge machine learning techniques. The primary goal was to forecast future stock prices using extensive historical data, providing valuable insights for potential investment strategies.

2. Dataset Insights and Meticulous Preprocessing:

The dataset was sourced from Yahoo Finance, encompassing 20 years of Apple's stock history. This rich dataset included key variables like date, open, low, close, adjusted close, and volume, providing a comprehensive view of the stock's performance over two decades.
- Missing Value Treatment: Utilized forward filling to maintain data continuity.
- Feature Engineering: A moving average feature was added to enhance trend analysis.
- Data Normalization: Applied MinMaxScaler for feature scaling, a crucial step for model accuracy.



```
# Feature Engineering: Adding moving average as an example
data['moving_average'] = data['close'].rolling(window=5).mean()

# Scaling
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data[['open', 'low', 'close', 'adj close', 'volume', 'moving_average']])
```

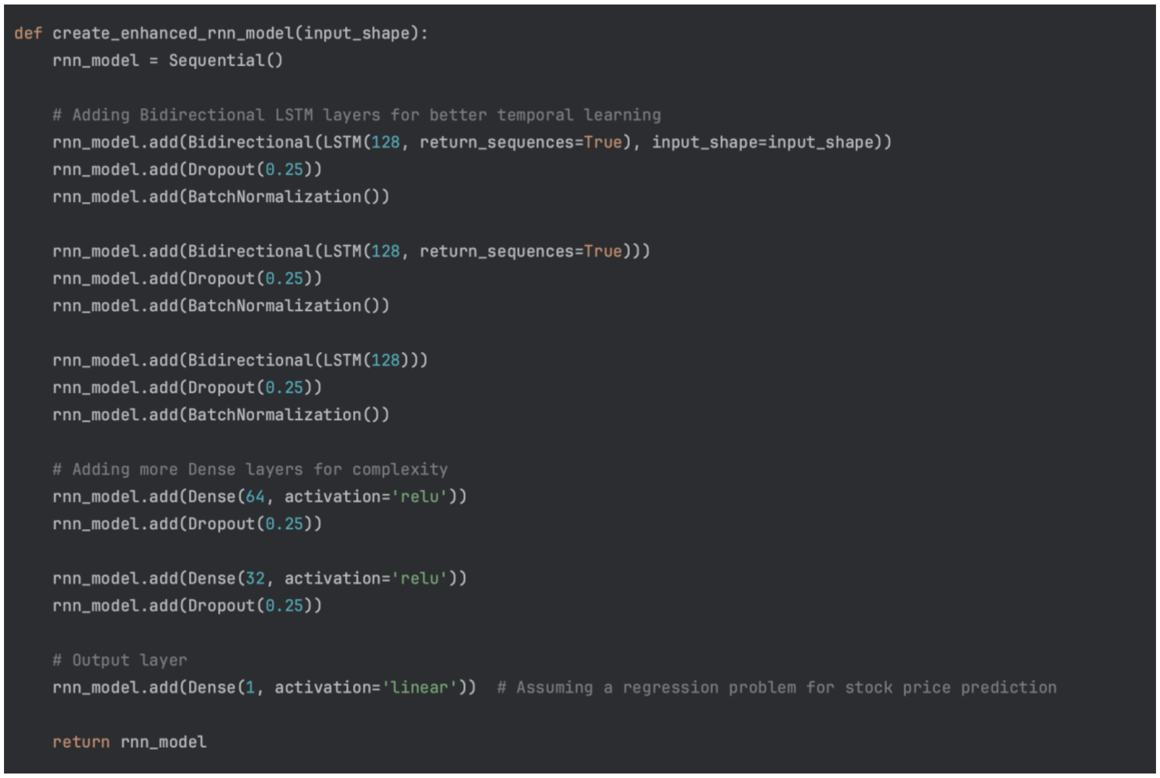
3. Advanced Model Architectures:

Two sophisticated models were crafted:

- RNN Model: Designed with Bidirectional LSTM layers, this model was adept at capturing both forward and backward time dependencies, essential for accurate stock price forecasting in the dynamic financial market.

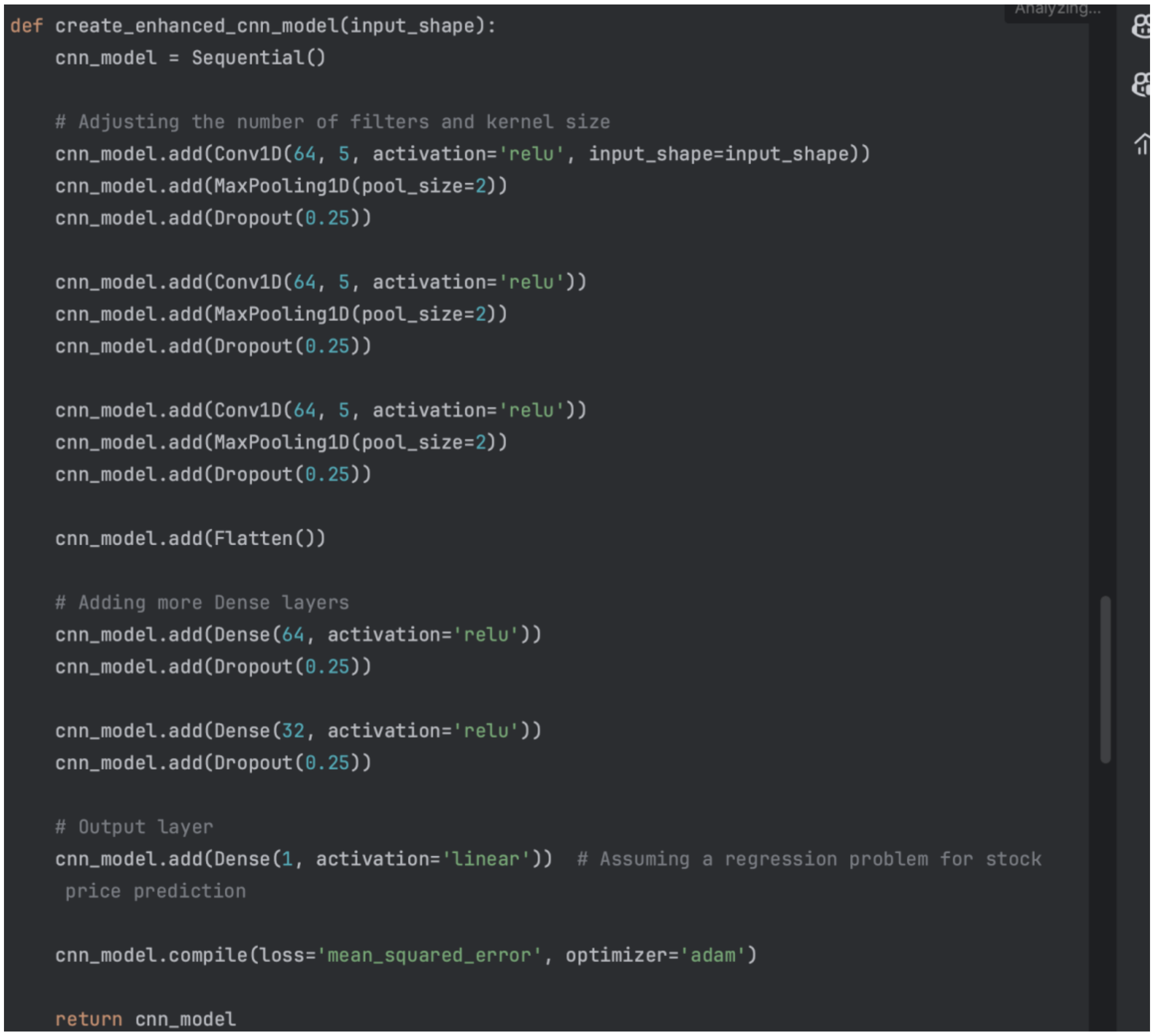
- CNN Model: Employed Conv1D layers for feature extraction from sequential data and MaxPooling1D for reducing dimensionality. This model was tailored to identify crucial patterns in stock price movements.

Code Snippet: RNN Model Architecture



```
def create_enhanced_rnn_model(input_shape):
    rnn_model = Sequential()

    # Adding Bidirectional LSTM layers for better temporal learning
    rnn_model.add(Bidirectional(LSTM(128, return_sequences=True), input_shape=input_shape))
    rnn_model.add(Dropout(0.25))
    rnn_model.add(BatchNormalization())

    rnn_model.add(Bidirectional(LSTM(128, return_sequences=True)))
    rnn_model.add(Dropout(0.25))
    rnn_model.add(BatchNormalization())

    rnn_model.add(Bidirectional(LSTM(128)))
    rnn_model.add(Dropout(0.25))
    rnn_model.add(BatchNormalization())

    # Adding more Dense layers for complexity
    rnn_model.add(Dense(64, activation='relu'))
    rnn_model.add(Dropout(0.25))

    rnn_model.add(Dense(32, activation='relu'))
    rnn_model.add(Dropout(0.25))

    # Output layer
    rnn_model.add(Dense(1, activation='linear'))  # Assuming a regression problem for stock price prediction

    return rnn_model
```

Code Snippet: CNN Model Architecture


```
def create_enhanced_cnn_model(input_shape):
    cnn_model = Sequential()

    # Adjusting the number of filters and kernel size
    cnn_model.add(Conv1D(64, 5, activation='relu', input_shape=input_shape))
    cnn_model.add(MaxPooling1D(pool_size=2))
    cnn_model.add(Dropout(0.25))

    cnn_model.add(Conv1D(64, 5, activation='relu'))
    cnn_model.add(MaxPooling1D(pool_size=2))
    cnn_model.add(Dropout(0.25))

    cnn_model.add(Conv1D(64, 5, activation='relu'))
    cnn_model.add(MaxPooling1D(pool_size=2))
    cnn_model.add(Dropout(0.25))

    cnn_model.add(Flatten())

    # Adding more Dense layers
    cnn_model.add(Dense(64, activation='relu'))
    cnn_model.add(Dropout(0.25))

    cnn_model.add(Dense(32, activation='relu'))
    cnn_model.add(Dropout(0.25))

    # Output layer
    cnn_model.add(Dense(1, activation='linear'))  # Assuming a regression problem for stock
     price prediction

    cnn_model.compile(loss='mean_squared_error', optimizer='adam')

    return cnn_model
```

4. Detailed Training Methodology:

The training involved a strategic data split into training and validation sets to ensure robust model evaluation. Both models were focused on minimizing Mean Squared Error, reflecting the precision in model training and evaluation.

5. Results and Interpretation:

The RNN model achieved a Mean Squared Error of 0.3923 and Mean Absolute Error of 0.6018. In comparison, the CNN model recorded a Mean Squared Error of 0.4035 and Mean Absolute Error of 0.6110. These results demonstrate the efficacy of both models in predicting stock prices, with a slight edge for the RNN model.

6. Addressing Challenges and Implementing Solutions:

Key challenges like overfitting were addressed through Dropout and BatchNormalization, ensuring model robustness. Hyperparameter tuning was meticulously conducted to optimize the models' performance.

7. Conclusion and Forward-Looking Statements:

This extensive project underscores the potential of using sophisticated RNN and CNN models for stock price prediction. The slight advantage of the RNN model aligns with its inherent design for processing sequential data. Future explorations could include hybrid models or experimenting with newer architectures like Transformer models to further enhance prediction accuracy.