1. XGBoost Model

Methodology

Data & Features:

- **Dataset**: Historical AAPL stock data (1980–2024) with engineered features:
  - `50_Day_MA` (50-day moving average).
  - `200_Day_MA` (200-day moving average).
  - `RSI` (14-day Relative Strength Index).
  - `Volatility` (7-day rolling standard deviation of daily returns).
- **Target Variable**: `Close` price.

Implementation:

- **Train-Test Split**: 80-20 split with `random_state=42` for reproducibility.
- **Scaling**: Features normalized to [0, 1] using `MinMaxScaler`.
- **Model**: `XGBRegressor` with hyperparameters:
  - `n_estimators=200`, `learning_rate=0.1`, `max_depth=5`, `subsample=0.8`.

Strengths

- **High Accuracy**: Explains 99% of variance in prices, indicating exceptional predictive power.
- **Robustness**: Built-in regularization (e.g., `max_depth`, `subsample`) prevents overfitting.
- **Feature Handling**: Efficiently captures non-linear relationships (e.g., momentum via `RSI`).

Weaknesses

- **Complexity**: Less interpretable than simpler models like linear regression.
- **Computational Cost**: Requires careful tuning of hyperparameters for optimal performance.

Suitability

XGBoost is ideal for stock price prediction due to its ability to model complex, non-linear trends in noisy financial data. Its high accuracy and generalization make it suitable for both short-term trading and long-term forecasting.

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("XGBoost - MSE:", mse)
print("XGBoost - R²:", r2)
print("XGBoost - RMSE:", np.sqrt(mse))
```

```
[*********************100%***********************]  1 of 1 completed
XGBoost - MSE: 1.9298205375671387
XGBoost - R²: 0.9992537498474121
XGBoost - RMSE: 1.3891798075005044
```

```
[26] y_train_pred = xgb_model.predict(X_train_scaled)
     train_r2 = r2_score(y_train, y_train_pred)
     train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))

     print("Train R²:", train_r2)
     print("Train RMSE:", train_rmse)
```

```
Train R²: 0.9997512102127075
Train RMSE: 0.783907908543761
```

2. Random Forest Model

Methodology

Data & Features:

- Same dataset and features as XGBoost.

Implementation:

- **Train-Test Split**: 80-20 split *without shuffling* to preserve temporal order.
- **Scaling**: Features normalized using `MinMaxScaler`.
- **Model**: `RandomForestRegressor` with hyperparameters:
  - `n_estimators=100`, `max_depth=5`, `min_samples_split=10`.

Strengths

- **Ensemble Learning**: Aggregates predictions from multiple trees to reduce variance.
- **Interpretability**: Provides feature importance scores (e.g., `200_Day_MA` is most influential).
- **Stability**: Less prone to overfitting than individual decision trees.

Weaknesses

- **Bias Toward Averages**: Struggles with extreme price movements due to majority voting.
- **Speed**: Slower inference compared to XGBoost for large datasets.

Suitability

Random Forest is well-suited for exploratory analysis and scenarios where interpretability is critical. While slightly less accurate than XGBoost, it serves as a reliable baseline model for financial forecasting.

```
train_mse = mean_squared_error(y_train, y_train_pred_rf)
train_rmse = np.sqrt(train_mse)
train_r2 = r2_score(y_train, y_train_pred_rf)

print(" Random Forest - Test Set")
print("R²:", test_r2)
print("RMSE:", test_rmse)

print(" Random Forest - Train Set")
print("R²:", train_r2)
print("RMSE:", train_rmse)
```

```
 Random Forest - Test Set
R²: -1.4951782002731249
RMSE: 101.64204655749414
 Random Forest - Train Set
R²: 0.9978666266347549
RMSE: 0.31595455356792507
```

3. Support Vector Machine (SVM)

Methodology

Data & Features:

- Identical dataset and features as previous models.

Implementation:

- **Train-Test Split**: 80-20 split with `random_state=42`.
- **Scaling**: Features scaled to [0, 1] using `MinMaxScaler` (essential for SVM).
- **Model**: `SVR` with `kernel='rbf'`, `C=100`, `epsilon=0.1`.

Strengths

- **Kernel Flexibility**: RBF kernel captures complex patterns in price movements.
- **Regularization**: `C=100` balances margin maximization and error tolerance.

Weaknesses

- **Sensitivity to Scaling**: Performance degrades without proper normalization.
- **Hyperparameter Tuning**: Requires extensive optimization (e.g., `C`, `gamma`).
- **Computational Cost**: Training becomes slow for large datasets.

Suitability

SVM underperforms compared to tree-based models in this context, likely due to the high noise and non-stationarity of stock data. It may excel in smaller datasets with clearer patterns or after advanced feature engineering.

```python
# Display results
print("SVR - Test Set")
print(f"R²: {test_r2}")
print(f"RMSE: {test_rmse}")

print(" SVR - Train Set")
print(f"R²: {train_r2}")
print(f"RMSE: {train_rmse}")
```

```
SVR - Test Set
R²: 0.9985022720865208
RMSE: 1.9680223985395333
 SVR - Train Set
R²: 0.9984247915039558
RMSE: 1.9725955528293235
```

Conclusion

- **XGBoost**: Best for high-precision forecasting, balancing accuracy and robustness.
- **Random Forest**: Ideal for interpretable, stable predictions with moderate computational demands.
- **SVM**: Limited utility here but valuable for methodological comparisons or specialized use cases.