

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN, LSTM, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

```
In [2]: ticker = "GOOGL"
df = yf.download(ticker, start="2018-01-01", end="2024-01-01")

print(f"Dataset Shape: {df.shape}")
print(f"\nDate Range: {df.index.min().date()} to {df.index.max().date()}")
print(f"\nFirst 5 rows:")
df.head()
```

[*****100%*****] 1 of 1 completed

Dataset Shape: (1509, 5)

Date Range: 2018-01-02 to 2023-12-29

First 5 rows:

Out[2]:

Price	Close	High	Low	Open	Volume
Ticker	GOOGL	GOOGL	GOOGL	GOOGL	GOOGL
Date					
2018-01-02	53.220634	53.357999	52.219411	52.219411	31766000
2018-01-03	54.128628	54.355751	53.231543	53.256336	31318000
2018-01-04	54.338890	54.751480	54.264508	54.404847	26052000
2018-01-05	55.059437	55.222590	54.638417	54.720241	30250000
2018-01-08	55.253830	55.499303	55.045057	55.094646	24644000

```
In [3]: print("=== Dataset Info ===")
print(df.info())
print("\n=== Statistical Summary ===")
print(df.describe())
print("\n=== Missing Values ===")
print(df.isnull().sum())
```

```

=== Dataset Info ===
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1509 entries, 2018-01-02 to 2023-12-29
Data columns (total 5 columns):
#   Column                Non-Null Count  Dtype
---  -
0   (Close, GOOGL)         1509 non-null   float64
1   (High, GOOGL)          1509 non-null   float64
2   (Low, GOOGL)           1509 non-null   float64
3   (Open, GOOGL)          1509 non-null   float64
4   (Volume, GOOGL)        1509 non-null   int64
dtypes: float64(4), int64(1)
memory usage: 70.7 KB
None

```

```

=== Statistical Summary ===

```

	Close	High	Low	Open	Volume
Ticker	GOOGL	GOOGL	GOOGL	GOOGL	GOOGL
count	1509.000000	1509.000000	1509.000000	1509.000000	1.509000e+03
mean	90.444293	91.426617	89.414133	90.395013	3.485950e+07
std	31.091514	31.412016	30.777227	31.099673	1.562928e+07
min	48.829918	50.191165	48.482289	48.812560	9.312000e+06
25%	59.727356	60.004066	59.073260	59.574123	2.505600e+07
50%	87.298477	88.128123	86.215936	87.377828	3.091200e+07
75%	118.718788	120.046815	117.588134	118.644405	3.990000e+07
max	148.610245	150.304226	147.678433	150.010163	1.331780e+08

```

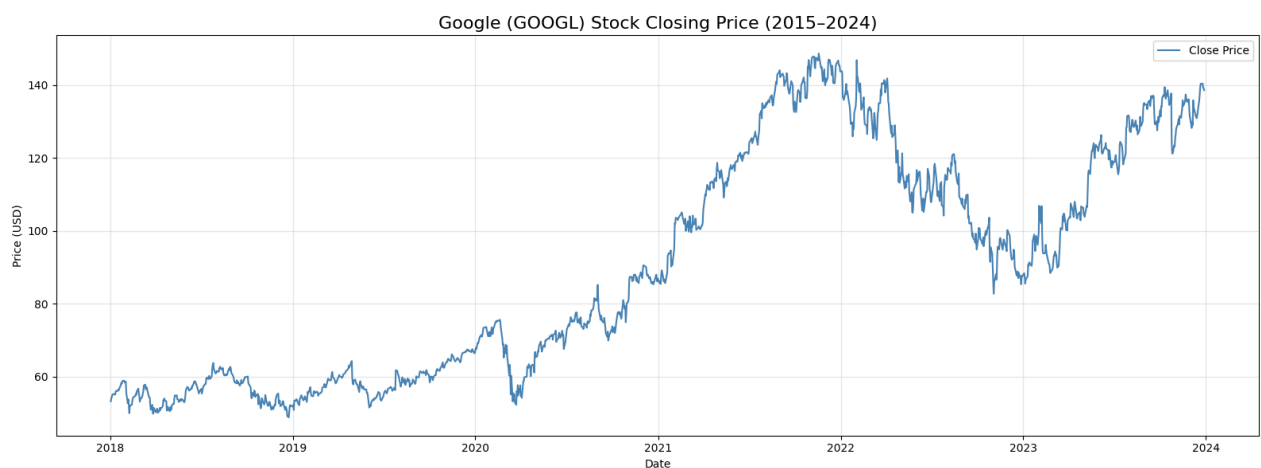
=== Missing Values ===
Price Ticker
Close GOOGL 0
High GOOGL 0
Low GOOGL 0
Open GOOGL 0
Volume GOOGL 0
dtype: int64

```

```

In [4]: plt.figure(figsize=(16, 6))
plt.plot(df.index, df['Close'], color='steelblue', linewidth=1.5, label='Close Price')
plt.title('Google (GOOGL) Stock Closing Price (2015-2024)', fontsize=16)
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

```



```

In [5]: # Use 'Close' price for prediction
data = df[['Close']].values

# Normalize data to [0, 1]
scaler = MinMaxScaler(feature_range=(0, 1))

```

```

data_scaled = scaler.fit_transform(data)

print(f"Original data range: [{data.min():.2f}, {data.max():.2f}]")
print(f"Scaled data range: [{data_scaled.min():.4f}, {data_scaled.max():.4f}]")
print(f"Total data points: {len(data_scaled)}")

```

Original data range: [48.83, 148.61]
 Scaled data range: [0.0000, 1.0000]
 Total data points: 1509

```

In [6]: def create_sequences(data, time_steps=60):
        """
        Creates (X, y) pairs where X is a window of `time_steps` days
        and y is the next day's price.
        """
        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)

TIME_STEPS = 60 # Use past 60 days to predict next day

# Train: 80%, Test: 20%
train_size = int(len(data_scaled) * 0.80)
train_data = data_scaled[:train_size]
test_data = data_scaled[train_size - TIME_STEPS:] # overlap for sequence continuity

X_train, y_train = create_sequences(train_data, TIME_STEPS)
X_test, y_test = create_sequences(test_data, TIME_STEPS)

# Reshape for RNN: [samples, time_steps, features]
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

print(f"Training samples : {X_train.shape}")
print(f"Testing samples : {X_test.shape}")

```

Training samples : (1147, 60, 1)
 Testing samples : (302, 60, 1)

```

In [7]: model = Sequential([
        SimpleRNN(units=64, return_sequences=True, input_shape=(TIME_STEPS, 1)),
        Dropout(0.2),
        SimpleRNN(units=64, return_sequences=False),
        Dropout(0.2),
        Dense(units=32, activation='relu'),
        Dense(units=1)
    ])

model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
model.summary()

```

WARNING:tensorflow:TensorFlow GPU support is not available on native Windows for TensorFlow >= 2.11. Even if CUDA/cuDNN are installed, GPU will not be used. Please use WSL2 or the TensorFlow-DirectML plugin.

D:\DL\.venv\lib\site-packages\keras\src\layers\rnn\rnn.py:199: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

```
super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 60, 64)	4,224
dropout (Dropout)	(None, 60, 64)	0
simple_rnn_1 (SimpleRNN)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense (Dense)	(None, 32)	2,080
dense_1 (Dense)	(None, 1)	33

Total params: 14,593 (57.00 KB)

Trainable params: 14,593 (57.00 KB)

Non-trainable params: 0 (0.00 B)

```
In [8]: early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    epochs=60,
    batch_size=32,
    validation_split=0.1,
    callbacks=[early_stop],
    verbose=1
)

print(f"\nTraining stopped at epoch: {len(history.history['loss'])}")
```

Epoch 34/60
 33/33 ————— 1s 22ms/step - loss: 0.0016 - mae: 0.0273 - val_loss: 0.0011 - val_mae: 0.0276
 Epoch 35/60
 33/33 ————— 1s 22ms/step - loss: 0.0016 - mae: 0.0268 - val_loss: 9.0297e-04 - val_mae: 0.0248
 Epoch 36/60
 33/33 ————— 1s 23ms/step - loss: 0.0014 - mae: 0.0251 - val_loss: 0.0022 - val_mae: 0.0393
 Epoch 37/60
 33/33 ————— 1s 21ms/step - loss: 0.0017 - mae: 0.0267 - val_loss: 9.2305e-04 - val_mae: 0.0246
 Epoch 38/60
 33/33 ————— 1s 21ms/step - loss: 0.0014 - mae: 0.0256 - val_loss: 0.0038 - val_mae: 0.0547
 Epoch 39/60
 33/33 ————— 1s 24ms/step - loss: 0.0015 - mae: 0.0262 - val_loss: 0.0015 - val_mae: 0.0312
 Epoch 40/60
 33/33 ————— 1s 21ms/step - loss: 0.0013 - mae: 0.0245 - val_loss: 0.0029 - val_mae: 0.0464
 Epoch 41/60
 33/33 ————— 1s 22ms/step - loss: 0.0017 - mae: 0.0279 - val_loss: 0.0016 - val_mae: 0.0323
 Epoch 42/60
 33/33 ————— 1s 22ms/step - loss: 0.0017 - mae: 0.0269 - val_loss: 0.0020 - val_mae: 0.0360
 Epoch 43/60
 33/33 ————— 1s 22ms/step - loss: 0.0014 - mae: 0.0257 - val_loss: 9.8204e-04 - val_mae: 0.0257
 Epoch 44/60
 33/33 ————— 1s 23ms/step - loss: 0.0013 - mae: 0.0247 - val_loss: 0.0025 - val_mae: 0.0419
 Epoch 45/60
 33/33 ————— 1s 22ms/step - loss: 0.0014 - mae: 0.0247 - val_loss: 0.0017 - val_mae: 0.0332

Training stopped at epoch: 45

```
In [9]: plt.figure(figsize=(12, 5))
plt.plot(history.history['loss'], label='Train Loss', color='royalblue')
plt.plot(history.history['val_loss'], label='Val Loss', color='tomato')
plt.title('Model Training Loss', fontsize=14)
plt.xlabel('Epoch')
plt.ylabel('MSE Loss')
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```



```
In [10]: # Predict on test set
y_pred_scaled = model.predict(X_test)

# Inverse transform to original price scale
y_pred = scaler.inverse_transform(y_pred_scaled)
y_actual = scaler.inverse_transform(y_test.reshape(-1, 1))

print(f"Predictions shape: {y_pred.shape}")
```

```
print(f"Sample predictions (first 5): {y_pred[:5].flatten().round(2)}")
print(f"Actual values (first 5): {y_actual[:5].flatten().round(2)}")
```

10/10 ————— 1s 38ms/step

Predictions shape: (302, 1)

Sample predictions (first 5): [97.15 97.78 98.58 98.36 99.46]

Actual values (first 5): [99.94 98.81 99.15 100.3 101.68]

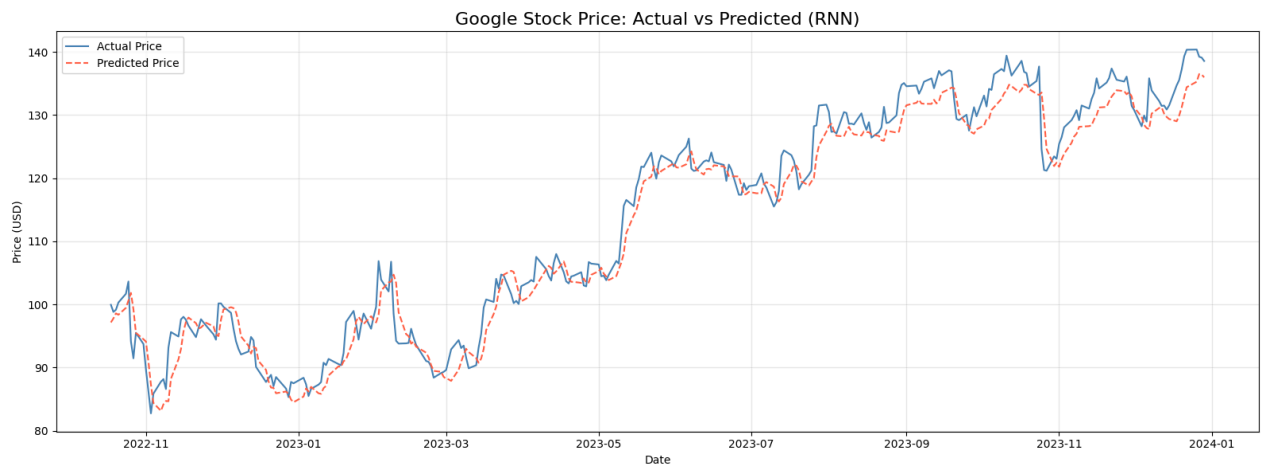
```
In [11]: mse = mean_squared_error(y_actual, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_actual, y_pred)
mape = np.mean(np.abs((y_actual - y_pred) / y_actual)) * 100

print("=" * 40)
print("      MODEL EVALUATION METRICS")
print("=" * 40)
print(f"  MSE  : {mse:.4f}")
print(f"  RMSE : {rmse:.4f}")
print(f"  MAE  : {mae:.4f}")
print(f"  MAPE : {mape:.2f}%")
print("=" * 40)
```

```
=====
      MODEL EVALUATION METRICS
=====
MSE   : 10.2773
RMSE  : 3.2058
MAE   : 2.5690
MAPE  : 2.28%
=====
```

```
In [12]: # Align dates with test predictions
test_dates = df.index[train_size:]

plt.figure(figsize=(16, 6))
plt.plot(test_dates, y_actual, label='Actual Price', color='steelblue', linewidth=1.5)
plt.plot(test_dates, y_pred, label='Predicted Price', color='tomato', linewidth=1.5, linestyle='--')
plt.title('Google Stock Price: Actual vs Predicted (RNN)', fontsize=16)
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```



```
In [13]: # Use Last TIME_STEPS days from full scaled data as seed
last_sequence = data_scaled[-TIME_STEPS:]
future_input = last_sequence.reshape(1, TIME_STEPS, 1)

future_predictions = []
n_future = 30 # days to forecast
```

```

for _ in range(n_future):
    pred = model.predict(future_input, verbose=0)
    future_predictions.append(pred[0, 0])
    # Slide window: drop oldest, append new prediction
    future_input = np.append(future_input[:, 1:, :], pred.reshape(1, 1, 1), axis=1)

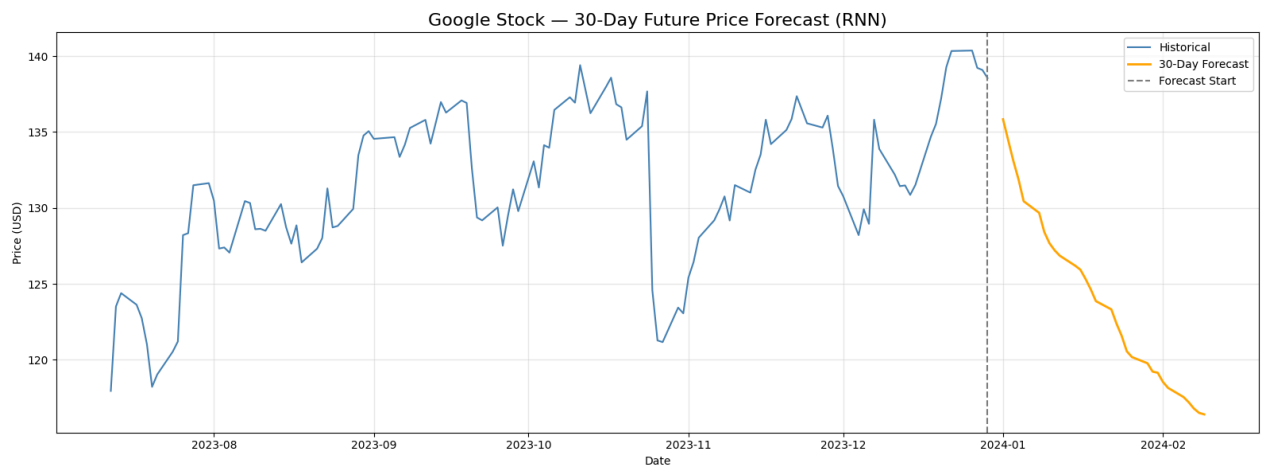
# Inverse scale
future_prices = scaler.inverse_transform(np.array(future_predictions).reshape(-1, 1))

# Generate future dates
last_date = df.index[-1]
future_dates = pd.date_range(start=last_date + pd.Timedelta(days=1), periods=n_future, freq='B')

plt.figure(figsize=(16, 6))
plt.plot(df.index[-120:], scaler.inverse_transform(data_scaled[-120:]), label='Historical', color='steelb')
plt.plot(future_dates, future_prices, label='30-Day Forecast', color='')
plt.axvline(x=last_date, color='gray', linestyle='--', label='Forecast Start')
plt.title('Google Stock - 30-Day Future Price Forecast (RNN)', fontsize=16)
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

print(f"\nForecasted price range: ${future_prices.min():.2f} - ${future_prices.max():.2f}")

```



Forecasted price range: \$116.41 - \$135.83

In []: