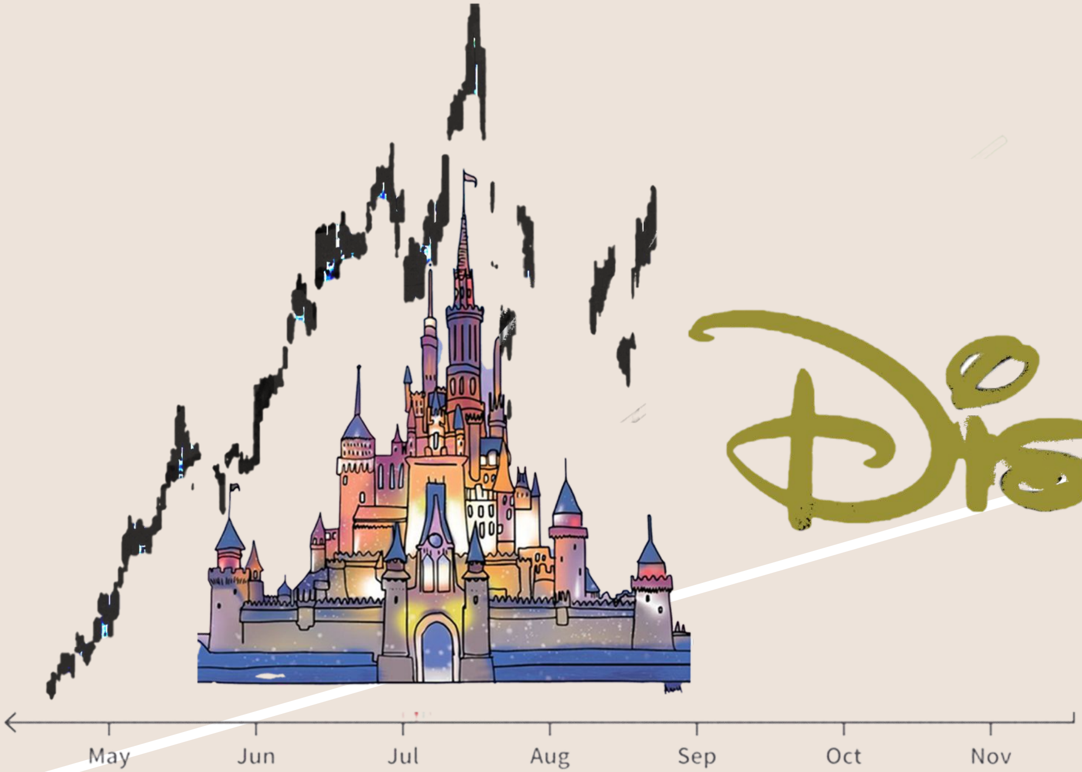


RNNs Predicting Stock of Disney

Gavin Wentzel Hao Zhang Dylan Lai

Recurrent Neural Network

1. Time-Series Modeling

- RNNs excel at capturing temporal patterns in sequential data like stock prices.

2. Memory of Past Trends

- Retains stock prices from previous days to understand context over time.

3. Non-linear Relationship Handling

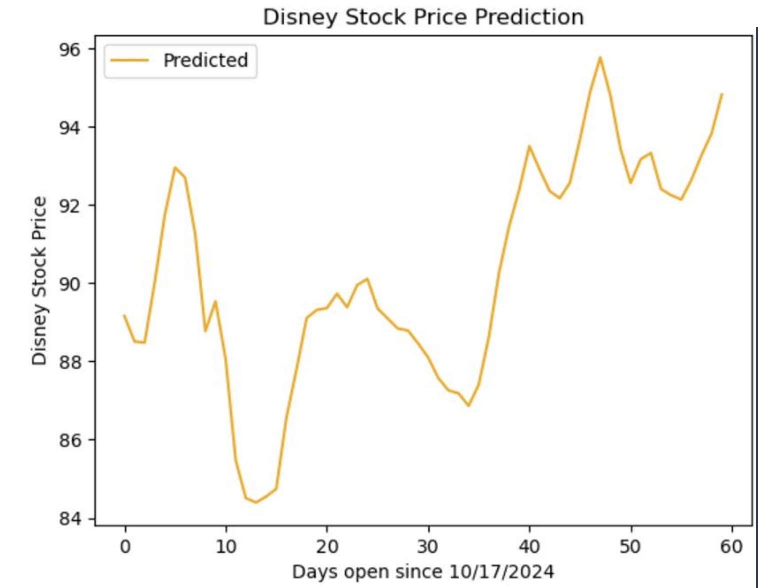
- Models complex market dynamics influenced by multiple factors, such as sentiment and volatility.
- Reveal hidden patterns and associations that would otherwise be difficult or nearly impossible for humans to identify with linear methods

4. Multivariate Input Support

- Can integrate multiple variables (e.g., prices, trading volume, economic indicators) for better predictions.

Goal

- Generate a general outline predicting future stock price movement
- Develop Trade Entry and Exit Strategy
- **Profit** $$



Languages Used

Python: Primary programming language for the project.

Libraries:

- **NumPy**: Numerical computations.
- **Pandas**: Data manipulation and preprocessing.
- **Matplotlib**: Data visualization.
- **Scikit-learn**: Data scaling and evaluation metrics.

- **TensorFlow**: Building and training the RNN model

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
from tensorflow.keras.optimizers import SGD
```

Dataset

Source: Kaggle.

Content: Historical stock prices of Disney from 1999 to October 2024.

Variables: Open Price, Close Price, High Price, Low Price, Volume

Format: CSV file imported as a Pandas DataFrame for analysis.

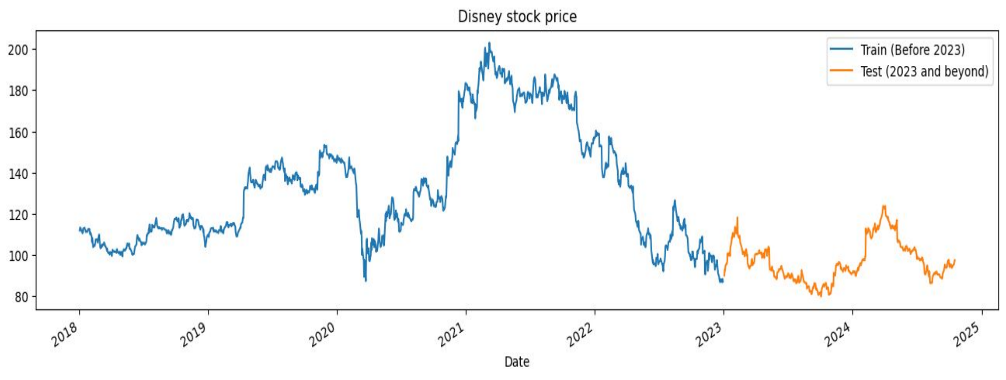
Focus:

- Used **high prices** to predict the **next high price** using a Long Short Term Memory and Gated Recurrent Unit

```
print(dataset.describe())
```

	Open Price	High Price	Low Price	Close Price	Volume
count	6281.000000	6281.000000	6281.000000	6281.000000	6.281000e+03
mean	67.436753	68.087816	66.735660	67.428240	9.810510e+06
std	44.669931	45.029558	44.237155	44.624743	6.085865e+06
min	13.800000	14.100000	13.480000	13.770000	1.487900e+06
25%	28.910000	29.380000	28.550000	29.000000	6.380600e+06
50%	43.610000	44.080000	43.260000	43.790000	8.340051e+06
75%	103.220000	104.150000	102.100000	103.260000	1.142185e+07
max	200.185000	203.020000	195.400000	201.910000	1.166250e+08

Dataset Scaling and Modification




```python
#divides the training and testing sets
def train_test_split(dataset, tstart, tend):
    train = dataset.loc[f"{tstart}":f"{tend}", "High Price"].values
    test = dataset.loc[f"{tend+1}":, "High Price"].values
    return train, test
training_set, test_set = train_test_split(dataset, tstart, tend)
num_features = 1
```

```python
#scales the training set
sc = MinMaxScaler(feature_range=(0, 1))
training_set = training_set.reshape(-1, num_features)
training_set_scaled = sc.fit_transform(training_set)
```

```python
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        end_ix = i + n_steps
        if end_ix > len(sequence) - 1:
            break
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)

#splits the training set into predictor and prediction segments
n_steps = 60
# split into samples
X_train, y_train = split_sequence(training_set_scaled, n_steps)
```

Training

```
#displays information about the lstm model
model_lstm = Sequential()
model_lstm.add(LSTM(units=125, activation="tanh", input_shape=(n_steps, num_features)))
model_lstm.add(Dense(units=num_features))
# Compiling the model
model_lstm.compile(optimizer="RMSprop", loss="mse")
model_lstm.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 125)	63,500
dense (Dense)	(None, 1)	126

Total params: 63,626 (248.54 KB)

Trainable params: 63,626 (248.54 KB)

Non-trainable params: 0 (0.00 B)

```
In [14]: model_lstm.fit(X_train, y_train, epochs=50, batch_size=32)
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 8.0626e-04
Epoch 42/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 7.0012e-04
Epoch 43/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 7.0750e-04
Epoch 44/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 6.7793e-04
Epoch 45/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 8.2062e-04
Epoch 46/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 7.1232e-04
Epoch 47/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 7.3169e-04
Epoch 48/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 7.9386e-04
Epoch 49/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 7.5055e-04
Epoch 50/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 6.2414e-04
```

```
#same for gru
model_gru = Sequential()
model_gru.add(GRU(units=125, activation="tanh", input_shape=(n_steps, num_features)))
model_gru.add(Dense(units=num_features))
# Compiling the RNN
model_gru.compile(optimizer="RMSprop", loss="mse")

model_gru.summary()
```

Model: "sequential_13"

Layer (type)	Output Shape	Param #
gru_4 (GRU)	(None, 125)	48,000
dense_13 (Dense)	(None, 1)	126

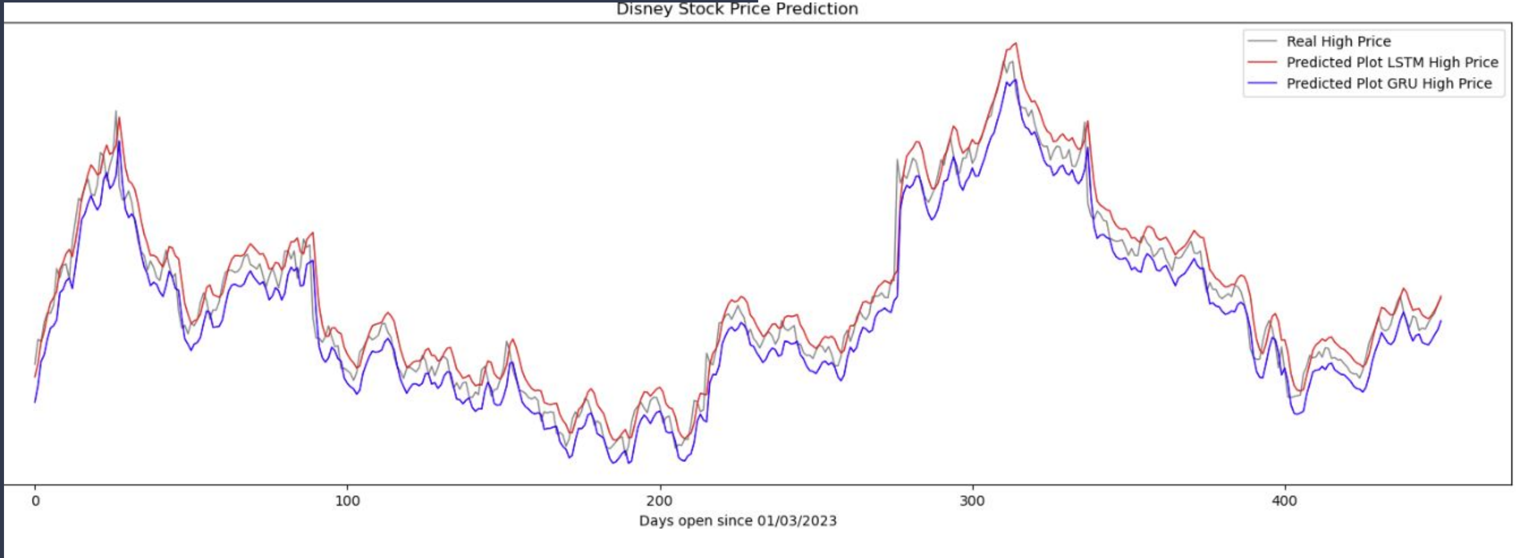
Total params: 48,126 (187.99 KB)

Trainable params: 48,126 (187.99 KB)

Non-trainable params: 0 (0.00 B)

```
model_gru.fit(X_train, y_train, epochs=50, batch_size=32)
38/38 ━━━━━━━━━━━━━━━━━━━━ 10s 147ms/step - loss: 6.4953e-04
Epoch 42/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 10s 135ms/step - loss: 7.7769e-04
Epoch 43/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 6s 143ms/step - loss: 5.6075e-04
Epoch 44/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 10s 137ms/step - loss: 7.6533e-04
Epoch 45/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 5s 140ms/step - loss: 5.4499e-04
Epoch 46/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 6s 149ms/step - loss: 6.6725e-04
Epoch 47/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 10s 132ms/step - loss: 6.4558e-04
Epoch 48/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 6s 143ms/step - loss: 7.2674e-04
Epoch 49/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 6s 143ms/step - loss: 5.2221e-04
Epoch 50/50
38/38 ━━━━━━━━━━━━━━━━━━━━ 6s 147ms/step - loss: 6.1471e-04
```

Demo – Prediction vs. Real Data




```
return_rmse(test_set,predicted_stock_price_lstm)
return_rmse(test_set,predicted_stock_price_gru)
```

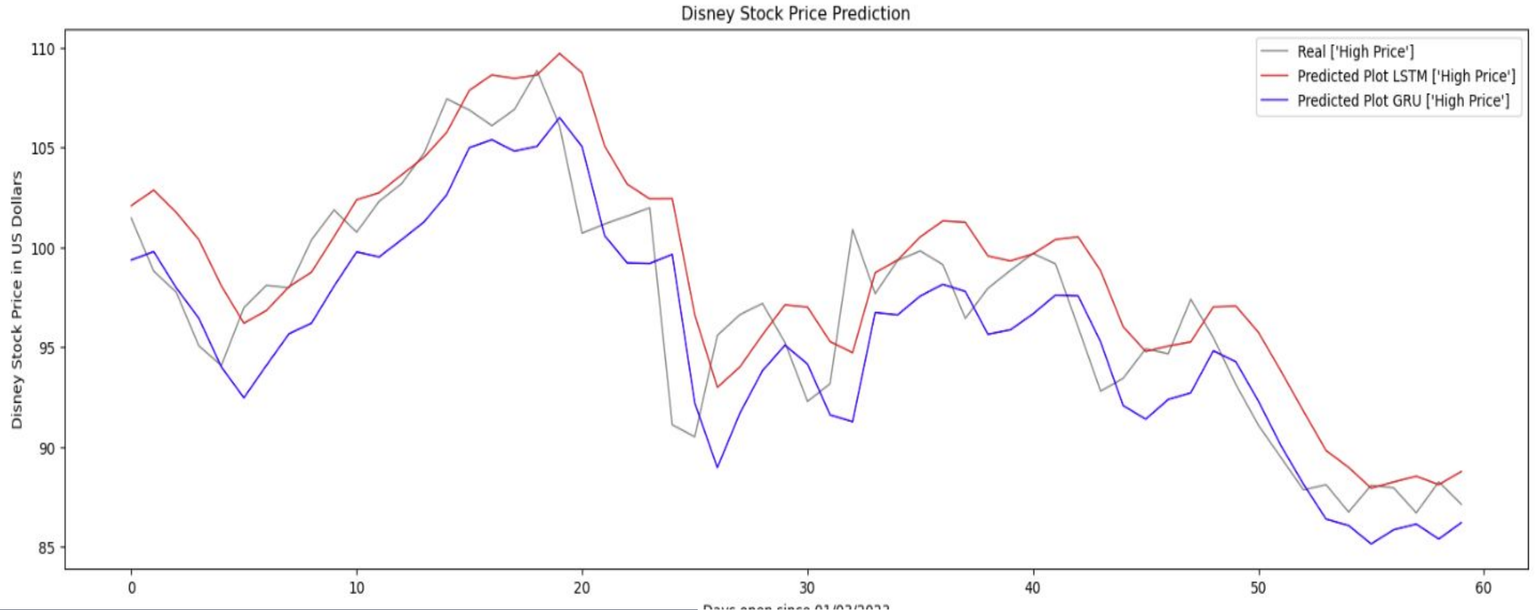
```
The root mean squared error is 2.24.
The root mean squared error is 2.43.
```

Demo – Prediction vs. Real Data

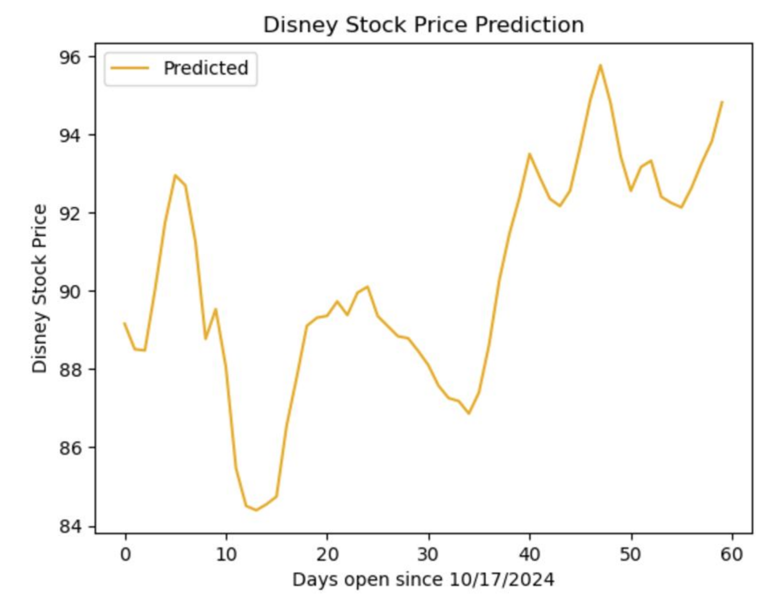
```
return_rmse(training_set[-futurePredicts:],newPredicts_LSTM)
return_rmse(training_set[-futurePredicts:],newPredicts_GRU)
```

```
The root mean squared error is 9.21.
The root mean squared error is 13.01.
```

```
comparePlots(small_test_set,[newPredicts_LSTM,newPredicts_GRU],['red','blue'],['LSTM','GRU'],["High Price"])
```



Demo – Prediction vs. Real Data



Results

- The RNN model successfully captured the overall **trend** in Disney's stock price movements.
- Despite the **irregular shape** and noise in the distribution of data points, the model was able to predict a **fairly accurate shape** for future high prices.
- Highlights the model's strength in learning patterns from complex, non-linear time-series data.
- Minor deviations in individual predictions were observed, likely due to market volatility and random fluctuations

Limitations

Influence of External Factors:

- Stock prices are influenced by more than just historical trends, such as:
  - Economic news, geopolitical events, and company announcements.
  - Market sentiment, investor behavior, and macroeconomic indicators.
- Including these external factors in the model is challenging due to their complexity and unpredictability.

Data Dependency:

- Over-reliance on historical prices can lead to limited accuracy in highly volatile markets.

Conclusion

RNN Effectiveness:

- Successfully modeled trends in Disney's stock prices, demonstrating its ability to handle sequential, non-linear data.
- Despite challenges, the predictions closely matched the general shape of future price trends.

Challenges Faced:

- External factors influencing stock prices are difficult to incorporate into the model.
- The irregular distribution of data points posed limitations on prediction accuracy.

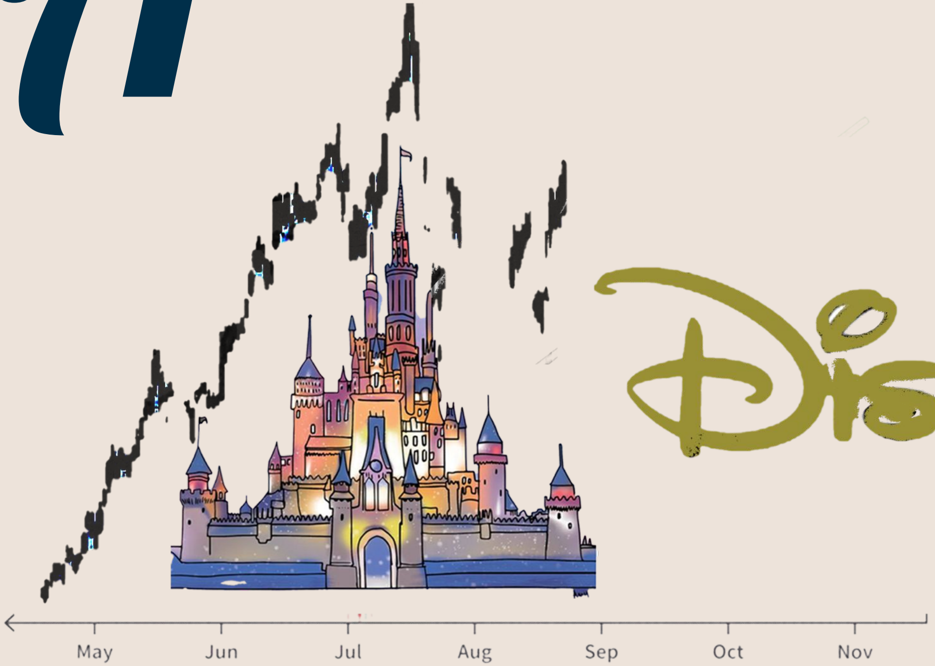
Future Scope:

- Enhance the model by integrating external data sources like market news or sentiment analysis.
- Explore advanced architectures such as GRUs or hybrid models for improved performance.

Takeaway:

- RNNs are powerful for stock price prediction but must be complemented by additional features for real-world applications

Q & A



Disney