Stock Forecasting with Machine Learning

Caila Hanus, Paulette Petracco, Dani Bar-Lavi, Kayode Ayenioye

Background

The stock market's dynamic and volatile nature presents both significant opportunities and considerable risks. Accurately forecasting stock prices can give investors a competitive edge, helping them make informed decisions and optimize their portfolios. Our project aims to leverage machine learning techniques to enhance the accuracy of stock price predictions and assess the associated risks.

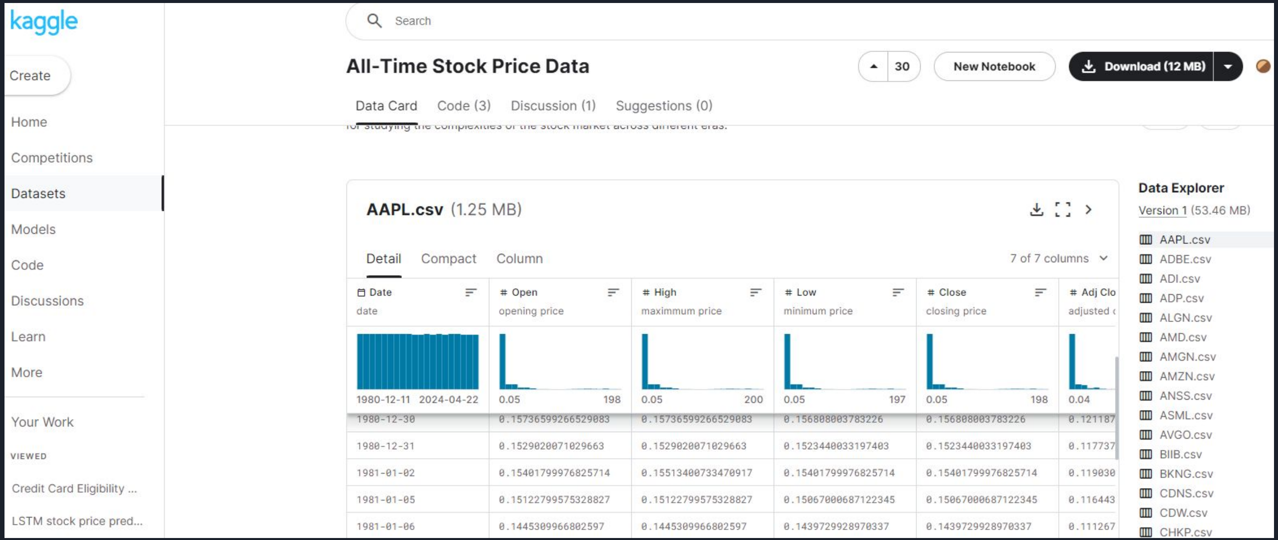
Main Research Questions:

- Predicting Stock Market Movement:
  - Short-Term Predictions: Where will the stock be tomorrow? Next month?
  - Long-Term Predictions: Where will the stock be next year? Next five years?
- Sentiment Analysis:
  - Financial News Impact: Analyze the sentiment of financial news and its impact on stock prices.
  - Social Media Influence: Examine how social media sentiment affects stock price predictions.

Data (Extraction)

Dataset: https://www.kaggle.com/datasets/hchsmost/test-dataset

- 77 .csv files including historical stock price data spanning various time periods
- Records that span multiple decades to explore the dynamics of different stocks, industries, and market sectors

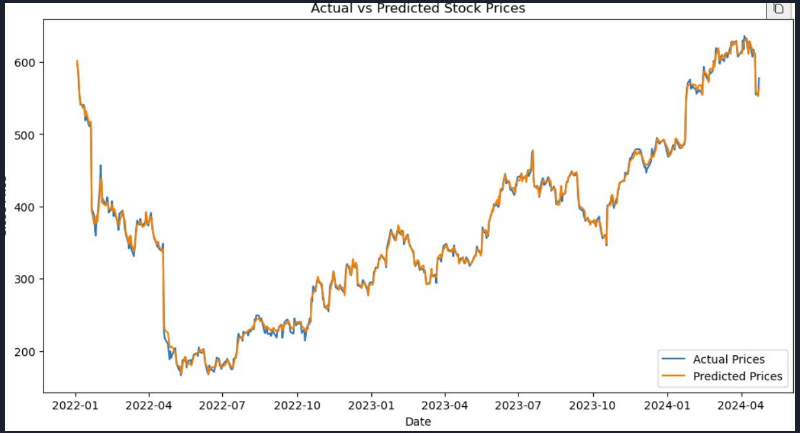


Short Term Forecasting

Model Troubleshooting - One Month Predictions

NFLX Dataset:

- 5517 rows of data ranging from 5/23/2002 to 4/23/2022 or 20 years of stock data
- Aimed to get predictions for the next month of data



Model Optimization

Things experimented with:

- Data Preprocessing:
  - Handling missing values and Feature scaling/normalization
  - Ex. Experimenting with different cutoff dates.
  - Ex. Using training sets of specified dates.
- Feature Engineering:
  - Creation of lagged features and additional features
  - Ex. Monthly Averages, Day-to-Day Changes, Volatility.

```python
# Load the data
data = pd.read_csv('Resources/NFLX.csv')

# Convert the 'Date' column to datetime format
data['Date'] = pd.to_datetime(data['Date'])

# Ensure the data is sorted by date
data = data.sort_values(by='Date')
```
✓ 0.0s

```python
# Creating moving average features
data['MA_5'] = data['Close'].rolling(window=5).mean()
data['MA_10'] = data['Close'].rolling(window=10).mean()

# Drop the rows with NaN values (caused by rolling windows)
data = data.dropna()

# Define the target variable and features
features = ['Open', 'High', 'Low', 'Volume', 'MA_5', 'MA_10']
target = 'Close'

# Split the data into training and testing sets
train_data = data[data['Date'] < '2022-01-01']
test_data = data[data['Date'] >= '2022-01-01']

X_train = train_data[features]
y_train = train_data[target]
X_test = test_data[features]
y_test = test_data[target]
```
✓ 0.0s

Model Optimization

More Experimenting:

- Model Selection and Comparison:
  - Linear Regression vs. Random Forest Regression.
  - Performance metrics (e.g., MSE, R-squared).
- Epoch-based Training:
  - Using neural networks.
  - Impact of different architectures.
- Train Model
- Create Visualizations

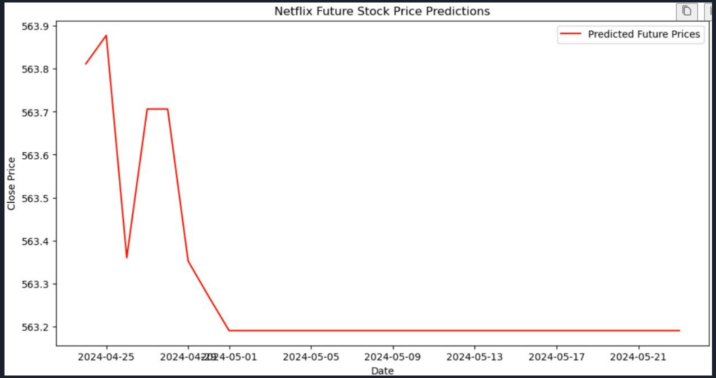
```python
# Initialize the model
model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train, y_train)

# Make predictions
predictions = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error: {mse}')
r2_score = r2_score(y_test, predictions)
print(f'R squared: {r2_score}')
```

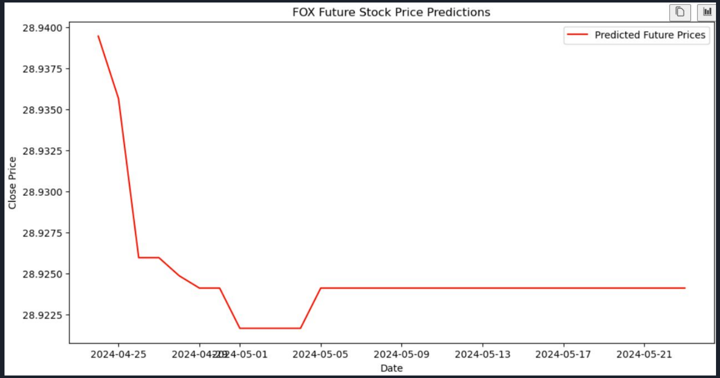
✓ 9.5s

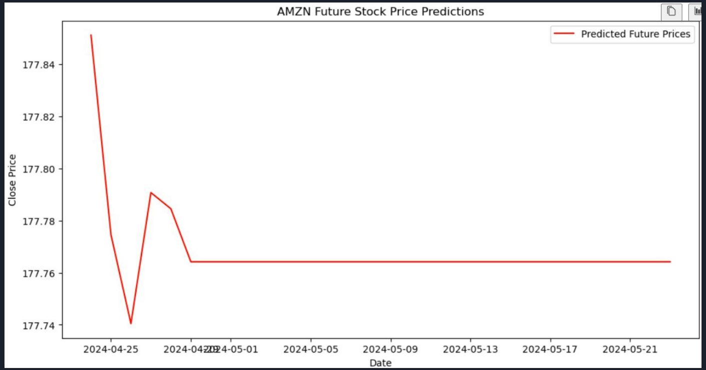
```
Mean Squared Error: 25.547305957802443
R squared: 0.9982296463424087
```

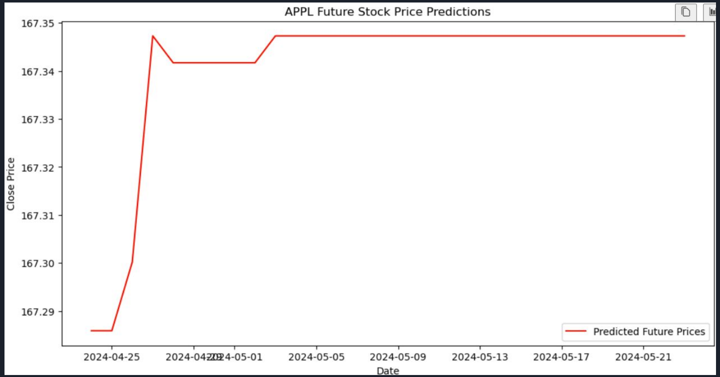
```python
# Predict future prices for the next month
def predict_next_month(model, last_data, features, days=30):
    future_predictions = []
    last_date = last_data['Date'].max()

    for i in range(days):
        last_row = last_data.iloc[-1]
        new_row = last_row.copy()

        # Move the date forward by one day
        new_row['Date'] = last_date + pd.Timedelta(days=1)

        # Predict the next closing price
        prediction = model.predict([last_row[features]])[0]
        new_row['Close'] = prediction

        # Update the moving averages
        last_data = pd.concat([last_data, new_row.to_frame().T], ignore_index=True)
        last_data['MA_5'] = last_data['Close'].rolling(window=5).mean()
        last_data['MA_10'] = last_data['Close'].rolling(window=10).mean()

        # Append the prediction to the future predictions list
        future_predictions.append(new_row)

        # Update the last_date
        last_date = new_row['Date']

        # Remove NaN values that may appear during the moving average calculation
        last_data = last_data.dropna()

    return pd.DataFrame(future_predictions)

# Prepare the last data point for prediction
last_data = data.copy()

# Predict future prices
future_predictions = predict_next_month(model, last_data, features)
```

One Month Forecasting by Day

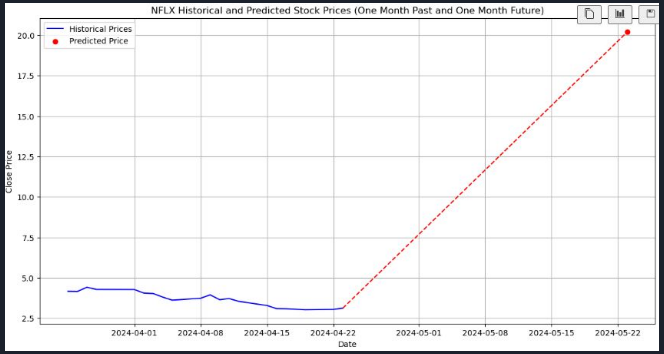








One Month Comparison to Forecasted



Predicting Stock Prices in the Health and Beauty Sector

The stocks analyzed in the 5 year model are:

- Align Technology, Inc. (ALGN)
- Lululemon Athletica Inc. (LULU)
- Peloton Interactive, Inc. (PTON)
- Ulta Beauty, Inc. (ULTA)

ALGN	Align Technology, Inc.	$255.70	+$0.16	↑0.063%
LULU	Lululemon Athletica Inc	$318.26	+$0.40	↑0.13%
PTON	Peloton Interactive Inc	$3.64	-$0.020	↓0.55%
ULTA	Ulta Beauty Inc	$382.50	-$0.11	↓0.029%

By leveraging 10 years of historical data, feature engineering, and neural networks, I attempted to forecast stock prices for the next five years.

Data Loading and Preparation

- Created a Library
- Pulled in the 77 files
- Combined the files

```
# Define directory containing the csv files
dir_path = Path("../Resources")

# create an empty list to hold the dataframes
dfs = []

# loop through each file in the directory
for file in os.listdir(dir_path):
    # check if the file is a csv file
    if file.endswith(".csv"):
        # extract the ticker symbol from the file name (assuming the file name is the ticker symbol)
        ticker = file.replace(".csv", "")
        # Read the file into a DataFrame
        stocks_df = pd.read_csv(dir_path / file)
        # Add a column to the DataFrame to store the ticker symbol
        stocks_df["Ticker"] = ticker
        # add the dataframe to the list
        dfs.append(stocks_df)

# concatenate the dataframes in the list
combined_stocks_df = pd.concat(dfs, ignore_index=True)

# Display the combined DataFrame to verify
print(combined_stocks_df.head())

# change the type in 'date' column to datetime
combined_stocks_df["Date"] = pd.to_datetime(combined_stocks_df["Date"])
```

Engineering and Model Training

Engineering

```python
stocks_to_analyze = ['ALGN', 'LULU', 'PTON', 'ULTA']

# Initialize dictionaries to hold the MSE and predictions for each stock symbol
mse_dict = {}
predictions_dict = {}

# Iterate through each stock in the list of specific stocks
for stock in stocks_to_analyze:
    # Select the stock's data
    stock_data = stocks_data_filtered[stocks_data_filtered["Ticker"] == stock]

    # Display the selected stock data
    print(f"Selected stock data for {stock}:")
    print(stock_data.head())

    # Set the date as the index
    stock_data.set_index("Date", inplace=True)

    # Define a feature in the data for previous date closing prices
    stock_data["Previous Day Close"] = stock_data["Close"].shift(1)

    # Define a feature in the data for the volume
    stock_data["Volume Difference"] = stock_data["Volume"].diff()

    # Drop rows with NaN values
    stock_data = stock_data.dropna()
```

Model Training

```python
# Define the features (X) and the target (y) variables for training purposes
X = stock_data[["Previous Day Close", "Volume Difference"]]
y = stock_data["Close"].values.reshape(-1, 1)

# Split the data into training and testing sets chronologically
split = int(0.7 * len(X))
X_train = X[: split]
X_test = X[split:]
y_train = y[: split]
y_test = y[split:]

# Normalize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the neural network model
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model with epochs
model.fit(X_train_scaled, y_train, epochs=50, batch_size=32, validation_split=0.2)

# Make predictions using the testing data
predictions = model.predict(X_test_scaled)
```

Insights

Future predictions

Mean Squared Error

```
Stock Ticker,Mean Squared Error
ALGN,138.9862474570898
LULU,80.46665080758463
PTON,248.35520640453953
ULTA,113.22467256840763
```

ALGN

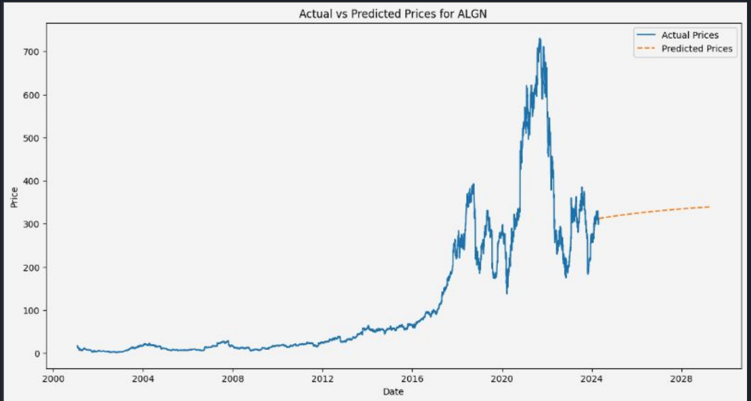
```
Date,Close
2024-04-30,312.58243
2024-05-31,313.27362
2024-06-30,313.95377
2024-07-31,314.62308
2024-08-31,315.2817
2024-09-30,315.92987
2024-10-31,316.5677
2024-11-30,317.19534
2024-12-31,317.813
2025-01-31,318.42078
2025-02-28,319.01892
2025-03-31,319.6075
2025-04-30,320.1867
2025-05-31,320.75668
2025-06-30,321.31754
2025-07-31,321.8695
2025-08-31,322.4126
2025-09-30,322.94708
2025-10-31,323.47305
2025-11-30,323.99057
2025-12-31,324.49988
2026-01-31,325.0011
2026-02-28,325.4943
2026-03-31,325.9796
2026-04-30,326.4572
2026-05-31,326.9272
2026-06-30,327.38968
2026-07-31,327.8448
2026-08-31,328.2926
2026-09-30,328.7333
2026-10-31,329.16702
2026-11-30,329.5938
2026-12-31,330.01382
2027-01-31,330.42706
2027-02-28,330.83374
2027-03-31,331.23395
2027-04-30,331.62772
2027-05-31,332.01526
2027-06-30,332.39664
2027-07-31,332.7719
2027-08-31,333.1412
2027-09-30,333.50455
2027-10-31,333.86215
2027-11-30,334.21405
2027-12-31,334.56036
2028-01-31,334.90115
2028-02-29,335.23648
2028-03-31,335.56647
2028-04-30,335.89117
2028-05-31,336.21072
2028-06-30,336.52518
2028-07-31,336.83463
2028-08-31,337.13916
2028-09-30,337.4388
2028-10-31,337.7337
2028-11-30,338.0239
2028-12-31,338.30945
2029-01-31,338.59048
2029-02-28,338.86697
2029-03-31,339.1391
```

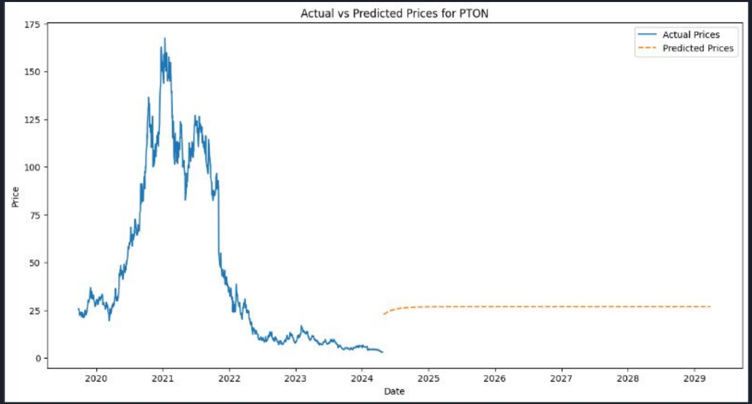
```python
# Predict the future values iteratively
future_closes = []
for date in future_dates_df.index:
    # prepare the input data for prediction
    input_data = pd.DataFrame({"Previous Day Close": [previous_close], "Volume Difference": [volume_difference]})
    input_data_scaled = scaler.transform(input_data)
    # Make the prediction
    predicted_close = model.predict(input_data_scaled)[0][0]
    # Append the predicted close to the list
    future_closes.append(predicted_close)
    # Update previous_close for the next iteration
    previous_close = predicted_close

# Add the predictions to the dataframe
future_dates_df["Close"] = future_closes

# Store the predictions in the dictionary
predictions_dict[stock] = future_dates_df

# Display the predictions for each stock ticker
for stock, future_dates_df in predictions_dict.items():
    print(f"Predictions for {stock}:")
    print(future_dates_df)
```

```python
# Define the base path as the parent directory of "Notebooks" and "Resources"
base_path = os.path.abspath(os.path.join(os.getcwd(), os.pardir))

# Create a directory named "Predictions" within the base path if it doesn't exist
predictions_folder = os.path.join(base_path, "Health and Beauty Five Year Predictions")
if not os.path.exists(predictions_folder):
    os.makedirs(predictions_folder)

# Save the Predictions results to individual files in the "Predictions" folder
for stock, future_dates_df in predictions_dict.items():
    prediction_file_path = os.path.join(predictions_folder, f"predictions_{stock}.csv")
    future_dates_df.to_csv(prediction_file_path)

print(f"Prediction results saved to the folder: {predictions_folder}")

# Create a directory named "MSE" within the base path if it doesn't exist
mse_folder = os.path.join(base_path, "Health and Beauty Five Year MSE Output")
if not os.path.exists(mse_folder):
    os.makedirs(mse_folder)

# Define the path to save the MSE results file
mse_file_path = os.path.join(mse_folder, "mse_results.csv")

# Save the MSE results to a file in the "MSE" folder
mse_df = pd.DataFrame(list(mse_dict.items()), columns=["Stock Ticker", "Mean Squared Error"])
mse_df.to_csv(mse_file_path, index=False)

print(f"MSE results saved to: {mse_file_path}")
```

Reviewing The Last Ten Years of Data to Predict The Next Five

5 Year - Actual vs Predicted





Modeling Big Stock Data

- Wrote one code to iterate a model over the historical data of 77 stocks
- Using nesting for-loops, the model is able to run separately over each stock and give separate precision and accuracy scores for the model as it pertains to each stock
- Created three slightly different versions of my code to create 6 month, 12 month, and five year projections. I’ll be focusing on 12 month projections for this presentation.

Process

1. Data Loading and Preprocessing
   a. Filter data by date
   b. Feature engineering–create new features for previous days close and volume

```
# Use a for loop to iterate data_load over each CSV in the Data folder to get individual results for each stock
for stock in os.listdir(data_path):
    if stock.endswith('.csv'):
        file_path = os.path.join(data_path, stock)
        data = data_load(file_path)

        # Filter this stock's data to only use entries from January 1st, 2020 onward
        start_date = datetime(2020, 1, 1)
        data_filtered_2020s = data[data['Date'] >= start_date].copy()

        # Define new features in the data for the previous day's close and previous day's volume
        data_filtered_2020s.loc[:, 'Prev_Day_Close'] = data_filtered_2020s['Close'].shift(1)
        data_filtered_2020s.loc[:, 'Prev_Day_Vol'] = data_filtered_2020s['Volume'].shift(1)

        # Drop rows with NaN values using dropna()
        data_filtered_2020s = data_filtered_2020s.dropna()

        # Ensure data is sorted by date
        data_filtered_2020s = data_filtered_2020s.sort_values(by='Date')
```

Process

2. Preparing Data for Testing:

- Feature/Target Definition
- Data Splitting for Testing (Chronological)

```
# Define features (X) and target (y) for training purposes
X = data_filtered_2020s[['Prev_Day_Close', 'Prev_Day_Vol']]
y = data_filtered_2020s['Close']

# Split data into test and training sets chronologically
cutoff_date = datetime(2023, 12, 31)
X_train = X[data_filtered_2020s['Date'] <= cutoff_date]
y_train = y[data_filtered_2020s['Date'] <= cutoff_date]
X_test = X[data_filtered_2020s['Date'] > cutoff_date]
y_test = y[data_filtered_2020s['Date'] > cutoff_date]
```

Process

3. Model Training

4. Model Evaluation

```
# Train stock_model
stock_model = RandomForestRegressor(n_estimators=100, random_state=42)
stock_model.fit(X_train, y_train)

# Evaluate the stock_model using mse (mean squared error)
y_pred = stock_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f'{stock}: Mean Squared Error: {mse}')
```

Process

5. Make Future Predictions (Iteratively)

```python
# Make full year predictions for this stock
future_dates = pd.date_range(start='2024-07-01', end='2025-7-01', freq='B')  # 'B' restricts to business days
future_data = pd.DataFrame({
    'Date': future_dates
})

# Initialize with the last known close
last_known_close = data_filtered_2020s['Close'].iloc[-1]

# Define the range for random volume generation
min_vol = min(data_filtered_2020s['Volume'])
max_vol = max(data_filtered_2020s['Volume'])

# Iteratively predict each day's closing price using the previous day's projected close
projected_closes = []
for i in range(len(future_dates)):
    if i == 0:
        prev_close = last_known_close
    else:
        prev_close = projected_closes[-1]

    prev_vol = np.random.uniform(low=min_vol, high=max_vol)

    future_data.loc[i, 'Prev_Day_Close'] = prev_close
    future_data.loc[i, 'Prev_Day_Vol'] = prev_vol

    # Predict the closing price
    pred_close = stock_model.predict([[prev_close, prev_vol]])[0]
    projected_closes.append(pred_close)

future_data['Projected_Close'] = projected_closes
future_data['Stock'] = stock.replace('.csv', '')  # Add stock identifier
stock_predictions.append(future_data)
```

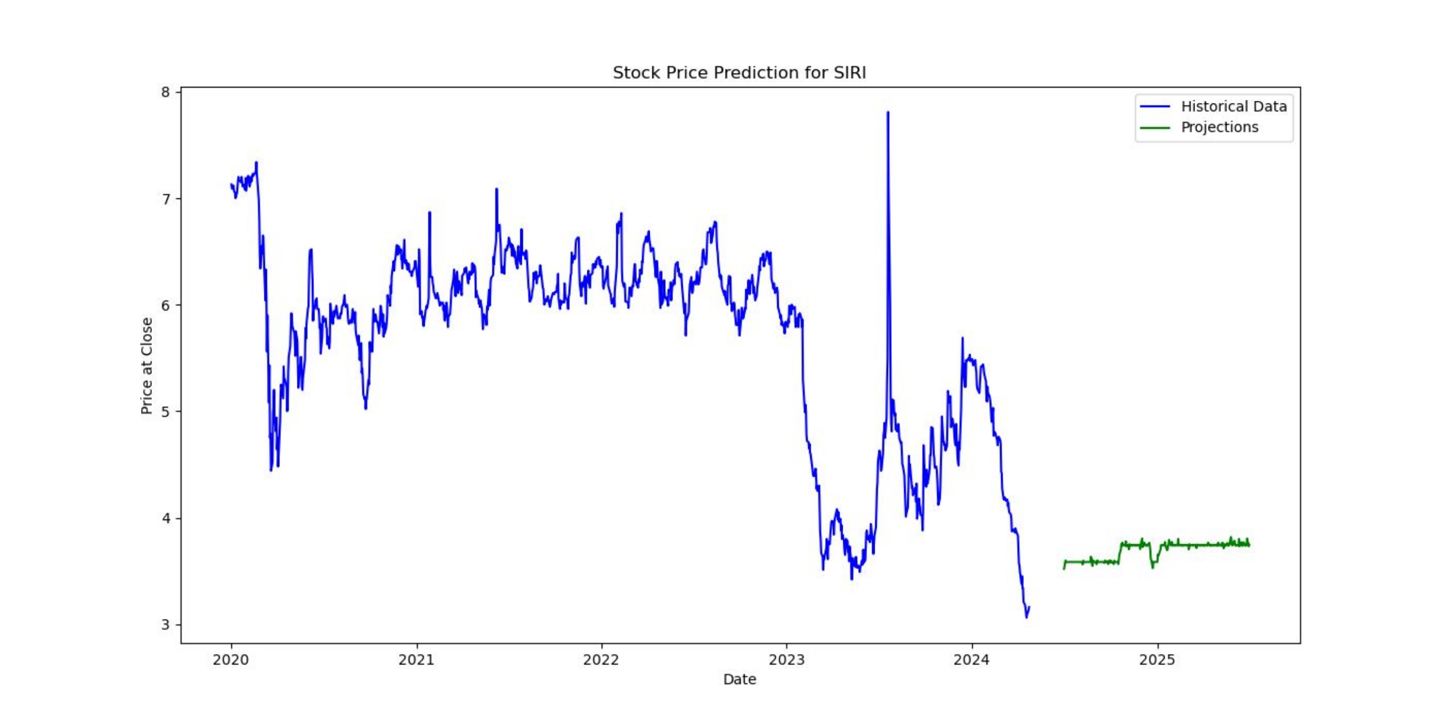
Process

6. Concatenating and saving projection data

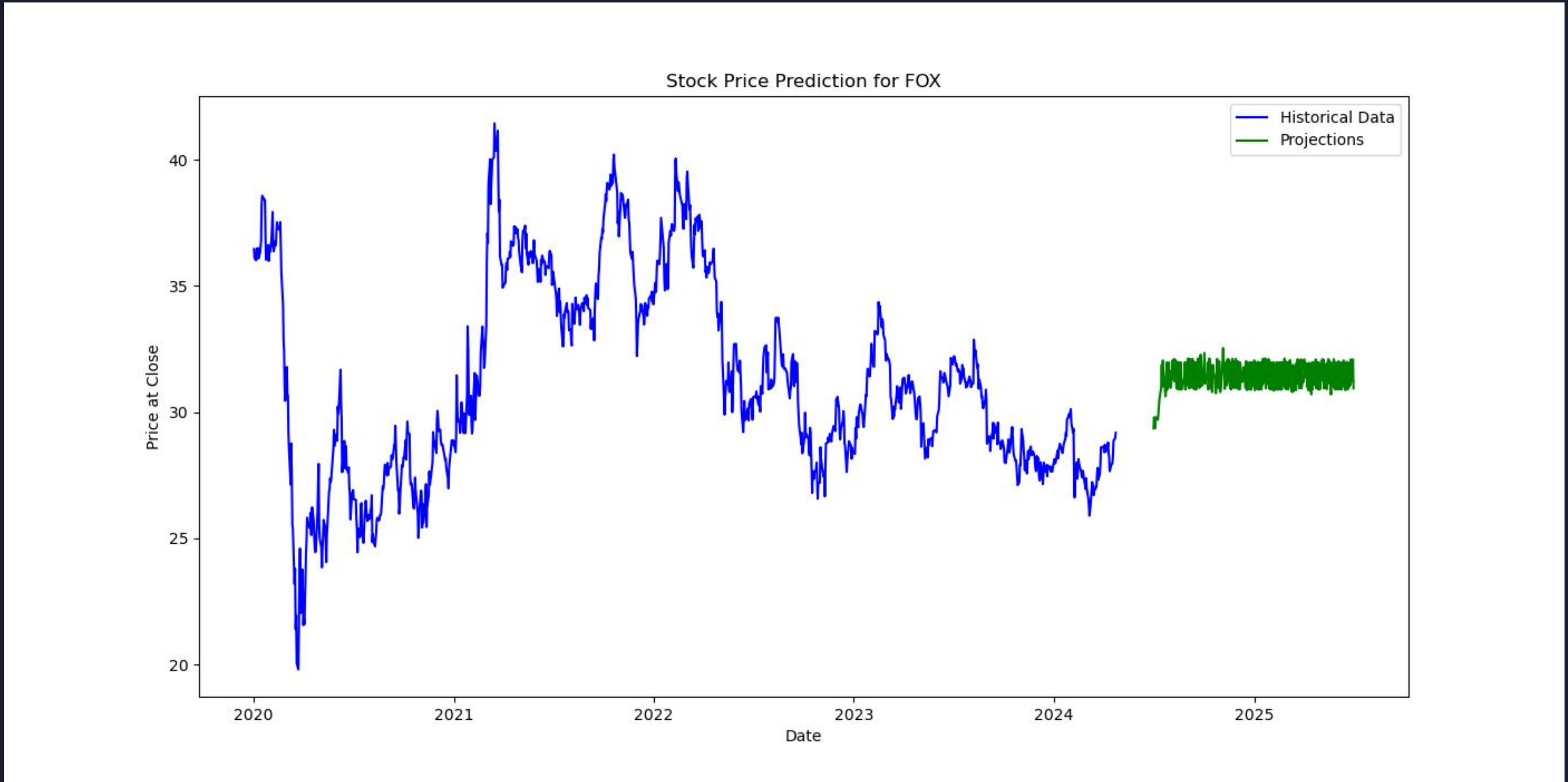
```
# Concatenate all predictions into one DataFrame
stock_predictions_df = pd.concat(stock_predictions, ignore_index=True)

print(stock_predictions_df.head())

# Save predictions to a CSV file
stock_predictions_df.to_csv('fullyr_future_stock_predictions.csv', index=False)
print('Stock projections exported to fullyr_future_stock_predictions.csv')
```

Visualizations

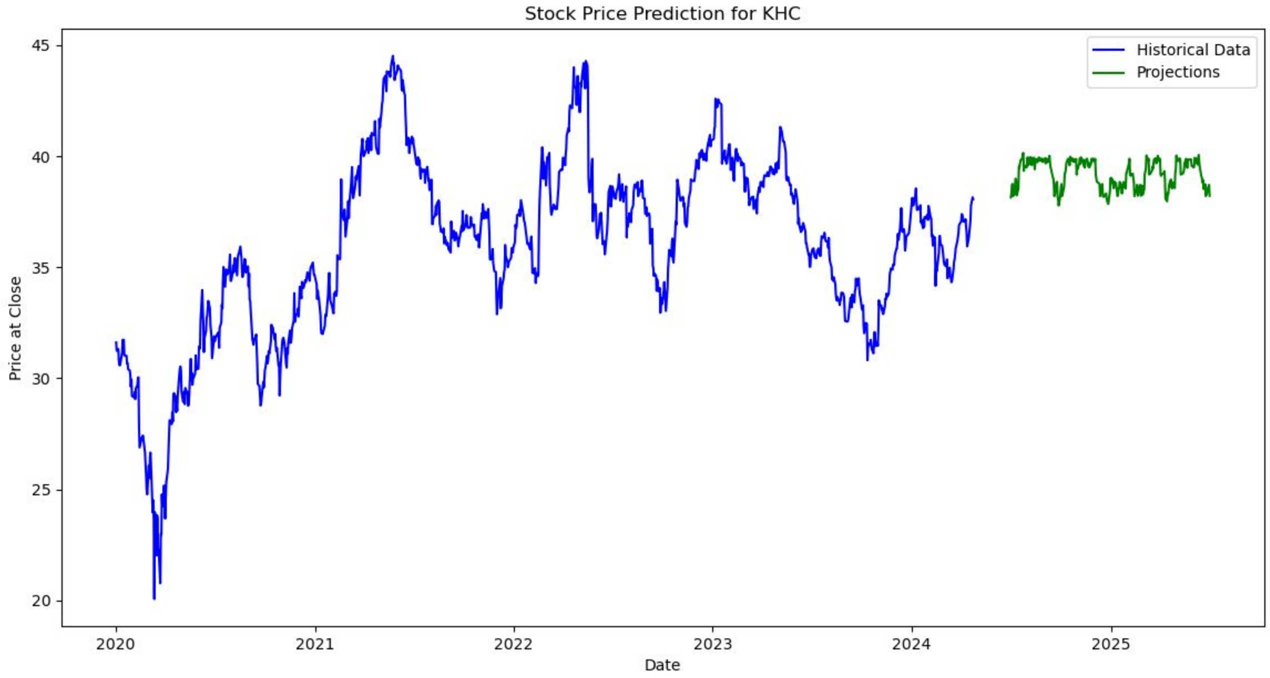
These are the projections for the top 3 lowest MSE stocks I analyzed, starting with Sirius XM.



20th Century Fox

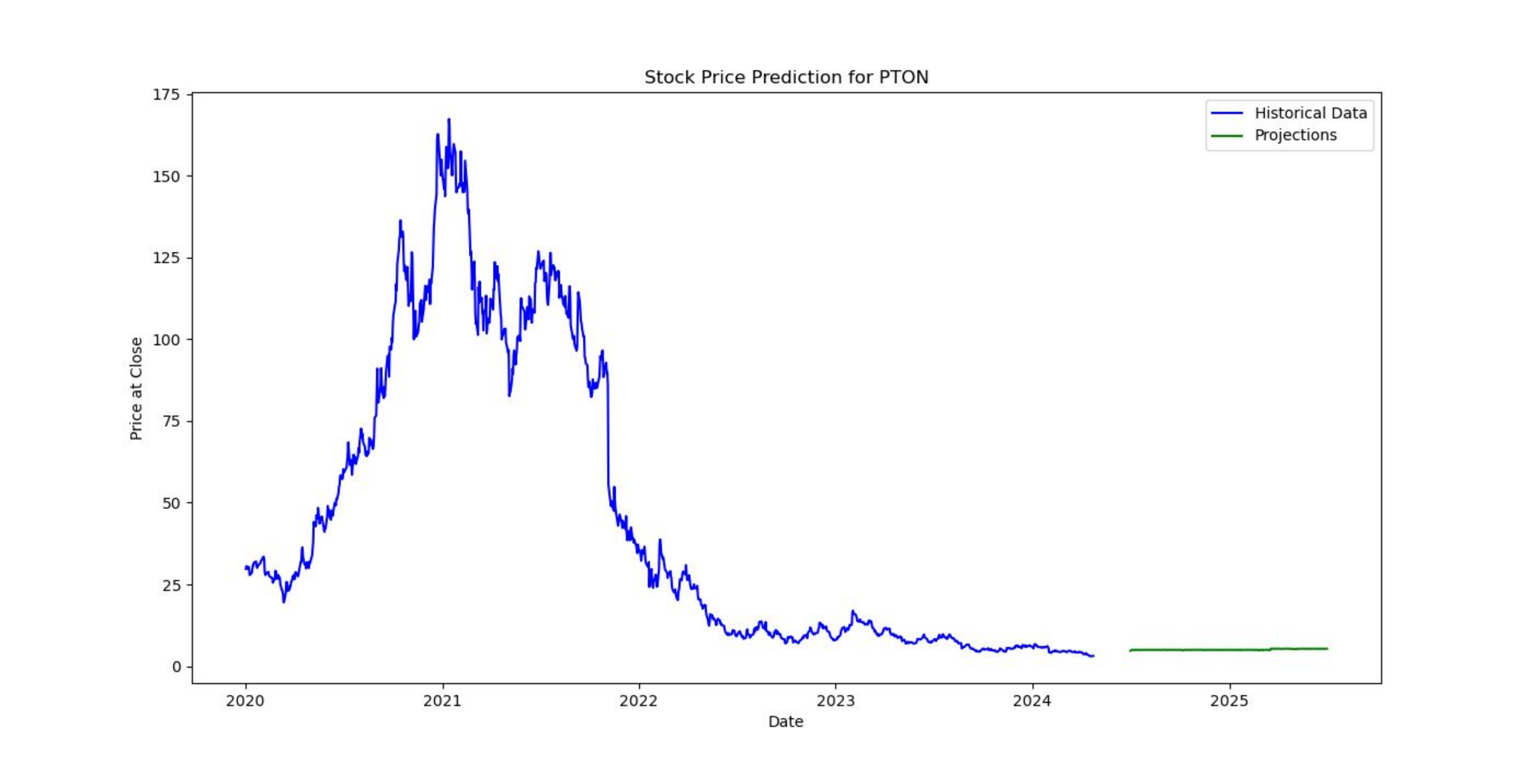


Kraft Heinz



Stock Price Prediction for KHC
Historical Data
Projections
Price at Close
45
40
35
30
25
20
2020
2021
2022
2023
2024
2025
Date

Peloton and Data Storytelling



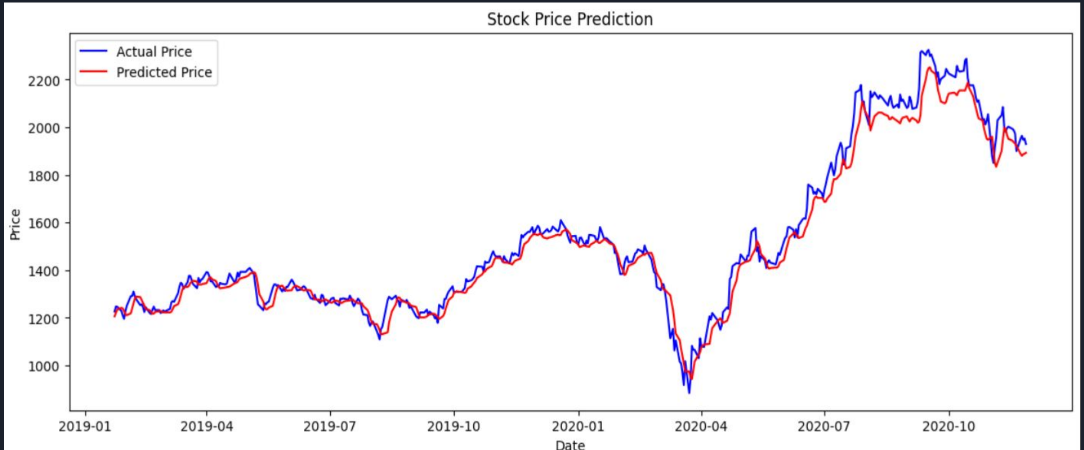
Reliance Stock Price Prediction

Historical Stock Prices

Volatility Assessment

Sentiment Analysis

5-Year Stock Price Projection



Why Sentiment Analysis of Financial News and Social Media?

- Analyzing market sentiment from news articles, social media, etc
- Creating market sentiment indicators indicating bullish or bearish trends
- Understanding how positive or negative news influences investor behaviour
- Monitoring sentiments on social media platforms to gauge public perception and sentiment towards stocks or the market