I .Introduction

Stock price prediction has long been a focal point in the world of finance, captivating investors, traders, and financial analysts alike. The ability to foresee future stock prices with a high degree of accuracy is an invaluable asset in making informed investment decisions. Traditionally, forecasting stock prices has relied on fundamental analysis, technical indicators, and statistical methods.

However, as the financial markets evolve and the volume of data generated grows exponentially, traditional methods often fall short in capturing the complex patterns and subtleties present in stock price movements. In this era of rapid technological advancements, we stand on the precipice of a new frontier in stock price prediction

Leveraging the power of advanced deep learning techniques, including Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM) networks, and attention mechanisms, we embark on a journey to revolutionize the way we forecast stock prices.

This endeavor seeks not only to predict stock prices with higher accuracy but also to uncover previously undetectable trends, correlations, and patterns that have the potential to redefine investment strategies.

The dataset provided to us from Kaggle, containing Microsoft's lifetime stock prices, serves as the foundation for our exploration into the application of cutting-edge deep learning techniques in stock price prediction.

This dataset is a valuable resource with a rich history of stock price movements, enabling us to develop, test, and validate innovative models that can be deployed in real-world trading scenarios.

In this document, we will detail the complete process of transitioning from a traditional approach to stock price prediction to an innovative, deep learning-based methodology. We will walk through each step of our journey, from data preprocessing and feature engineering to model development and evaluation.

We will also explore the nuances of combining CNN, LSTM, and attention mechanisms in our model architecture, aiming for superior predictive performance. Our pursuit is to unlock the potential of deep learning in forecasting stock prices, thereby providing investors and financial professionals with a robust and sophisticated tool for making more informed decisions in the ever-dynamic financial markets.

Through this innovative approach, we aim to enhance our understanding of the intricacies of stock price movements and contribute to the evolution of financial analytics.

FEATURE ENGINEERING

Importing Libraries: The code begins by importing the necessary libraries, primarily Pandas for data manipulation.

Date Formatting: It converts the 'Date' column to a datetime format and ensures that the DataFrame is sorted in chronological order based on the date.

Lag Features: The script creates lag features for the 'Close' prices up to 5 days in the past, which can be useful for time series forecasting
.
Moving Averages: It calculates two types of moving averages, the 10-day Simple Moving Average (SMA) and the 10-day Exponential Moving Average (EMA), providing different ways to smooth the data.

Relative Strength Index (RSI): The code defines a function to compute the Relative Strength Index (RSI), a momentum oscillator that measures the speed and change of price movements. It then applies this function to generate the RSI_14 feature.

Bollinger Bands: Another function is defined to calculate Bollinger Bands, which consist of an upper and lower band that help identify potential overbought or oversold conditions in the data.

Handling Missing Values: To account for missing values created by the rolling functions, the script fills them with zeros.

Display Data: The code concludes by printing the first few rows of the DataFrame, allowing you to inspect the newly created features and their values.

This feature engineering process is a crucial step in preparing financial data for machine learning or statistical analysis, as it helps capture important patterns and signals in the time series.

```
import pandas as pd

df['Date'] = pd.to_datetime(df['Date'])

df = df.sort_values(by='Date')


for i in range(1, 6):  # Create lags up to 5 days
    df[f'Close_Lag_{i}'] = df['Close'].shift(i)

# Calculate moving averages

df['SMA_10'] = df['Close'].rolling(window=10).mean()  # 10-day simple moving average
df['EMA_10'] = df['Close'].ewm(span=10, adjust=False).mean()  # 10-day exponential moving average

# Calculate relative strength index (RSI)
def calculate_rsi(data, window=14):
    delta = data['Close'].diff()
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)
    avg_gain = gain.rolling(window=window, min_periods=1).mean()
    avg_loss = loss.rolling(window=window, min_periods=1).mean()
    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))
    return rsi

df['RSI_14'] = calculate_rsi(df)

# Calculate Bollinger Bands
def calculate_bollinger_bands(data, window=20, num_std_dev=2):
    rolling_mean = data['Close'].rolling(window=window).mean()
    rolling_std = data['Close'].rolling(window=window).std()
    upper_band = rolling_mean + (rolling_std * num_std_dev)
    lower_band = rolling_mean - (rolling_std * num_std_dev)
    return upper_band, lower_band

df['Upper_Bollinger_Band'], df['Lower_Bollinger_Band'] = calculate_bollinger_bands(df)

# Fill missing values (due to rolling functions)
df.fillna(0, inplace=True)

print(df.head())
```

OP:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549	1031788800
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783	308160000
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899	133171200
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.064224	67766400
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.063107	47894400

	Close_Lag_1	Close_Lag_2	Close_Lag_3	Close_Lag_4	Close_Lag_5	SMA_10
0	0.000000	0.000000	0.000000	0.000000	0.0	0.0
1	0.097222	0.000000	0.000000	0.000000	0.0	0.0
2	0.100694	0.097222	0.000000	0.000000	0.0	0.0
3	0.102431	0.100694	0.097222	0.000000	0.0	0.0
4	0.099826	0.102431	0.100694	0.097222	0.0	0.0

	EMA_10	RSI_14	Upper_Bollinger_Band	Lower_Bollinger_Band
0	0.097222	0.000000	0.0	0.0
1	0.097853	100.000000	0.0	0.0
2	0.098686	100.000000	0.0	0.0
3	0.098893	66.662401	0.0	0.0
4	0.098747	54.544503	0.0	0.0

MODEL TRAINING

In machine learning, "model training" is the process of teaching a machine learning model to make predictions or decisions based on data. This training process involves the following key steps:

Data Collection: Gather and prepare a dataset that consists of input features (also known as independent variables or predictors) and corresponding target values (also known as labels or dependent variables). This dataset is used to train and evaluate the model.

Model Selection: Choose a specific machine learning algorithm or model architecture that is suitable for your problem. The choice of model depends on the type of task (classification, regression, clustering, etc.) and the nature of the data.

Training the Model: The model is fed with the training data, and it learns to make predictions by adjusting its internal parameters based on the provided examples. During training, the model tries to minimize the difference between its predictions and the actual target values.

Hyperparameter Tuning: Fine-tune the model's hyperparameters, which are settings that are not learned from the data but can significantly affect the model's performance. This can involve techniques like cross-validation to find the best hyperparameters.

Validation and Testing: Split the dataset into training and validation sets to evaluate the model's performance during training. After tuning, test the model on a separate test set to assess its generalization to new, unseen data.

Performance Evaluation: Use appropriate evaluation metrics to measure how well the model performs on the validation and test data. The choice of metrics depends on the specific task (e.g., accuracy, mean squared error, F1-score, etc.).

Model Deployment: Once the model is trained and evaluated, it can be deployed for making predictions on new, real-world data.

Model training is a crucial part of the machine learning workflow, as the quality of the trained model largely depends on the quality of the data, the choice of model, and the tuning process. The goal is to create a model that can make accurate predictions on new data and solve the intended problem effectively.

REGRESSION

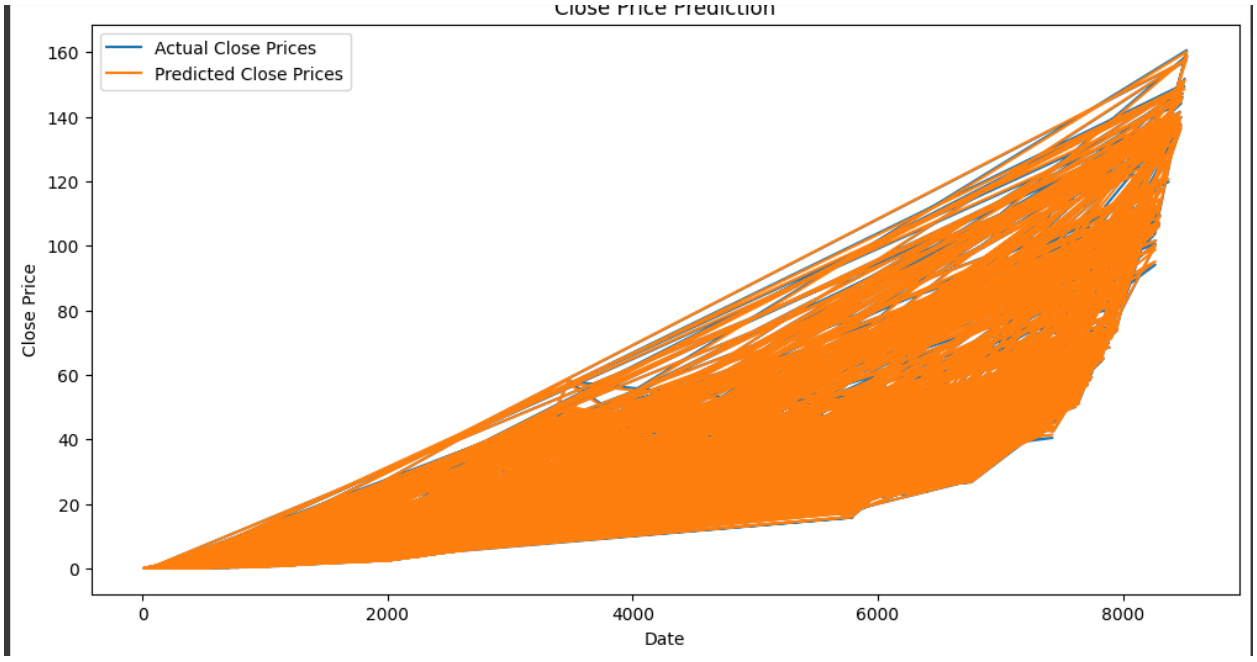
Linear regression is a supervised machine learning technique used for modeling the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

data = pd.read_csv('MSFT.csv')

# Extract features and target variable
features = data[['Open', 'High', 'Low', 'Adj Close', 'Volume']]
target = data['Close']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2,
random_state=42)
```

```
model = LinearRegression()

model.fit(X_train, y_train)


y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Visualize the results
plt.figure(figsize=(12, 6))
plt.plot(y_test.index, y_test.values, label='Actual Close Prices')
plt.plot(y_test.index, y_pred, label='Predicted Close Prices')
plt.legend()
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Close Price Prediction')
plt.show()
```

OP;



RANDOM FOREST:

Random Forest is an ensemble machine learning technique that combines multiple decision trees to make more accurate predictions. It's versatile for both classification and regression tasks, offering robustness against overfitting and high performance, and can handle a wide range of data types and complex relationships in the data.

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error


data = pd.read_csv('MSFT.csv')


features = data[['Open', 'High', 'Low', 'Adj Close', 'Volume']]
target = data['Close']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2,
random_state=42)

# Create a Random Forest regressor
model = RandomForestRegressor(n_estimators=100, random_state=42)

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

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

OP:
Mean Squared Error: 0.04266101126021738

NAVIE BAYES

Naive Bayes is a simple yet effective classification algorithm based on Bayes' theorem. It assumes that features are conditionally independent, making it computationally efficient and particularly useful for text classification, spam detection, and recommendation systems. Despite its simplicity, Naive Bayes often achieves competitive results and is suitable for handling high-dimensional datasets.

CODE;

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

data = pd.read_csv('MSFT.csv')

data['Close_Direction'] = (data['Close'] - data['Close'].shift(1)) > 0
features = data[['Open', 'High', 'Low', 'Adj Close', 'Volume']]
target = data['Close_Direction']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2,
random_state=42)

# Create a Naive Bayes model
model = GaussianNB()

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

# Make predictions
y_pred = model.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

OP:

Accuracy: 0.4961876832844575

KMEANS;

K-Means is an unsupervised clustering algorithm that partitions data into K distinct clusters based on similarity. It's widely used for pattern recognition, image segmentation, and customer segmentation. K-Means works by iteratively assigning data points to the nearest cluster center and updating the centers to minimize the sum of squared distances, aiming to find natural groupings in the data.

CODE:

```
import pandas as pd
from sklearn.cluster import KMeans

data = pd.read_csv('MSFT.csv')

features = data[['Open', 'High', 'Low', 'Adj Close', 'Volume']]

k=3
model = KMeans(n_clusters=k)

data['Cluster'] = model.fit_predict(features)
```

OP:

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning warnings.warn(

```
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

data = pd.read_csv('MSFT.csv')

data['Close_Direction'] = (data['Close'] - data['Close'].shift(1)) > 0
features = data[['Open', 'High', 'Low', 'Adj Close', 'Volume']]
target = data['Close_Direction']

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2,
random_state=42)

k=5
model = KNeighborsClassifier(n_neighbors=k)
# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model (classification)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

OP:

```
Accuracy: 0.4879765395894428
```

RANDOM FOREST CLASSIFIER

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
data = pd.read_csv('MSFT.csv')

data['Close_Direction'] = (data['Close'] - data['Close'].shift(1)) > 0
features = data[['Open', 'High', 'Low', 'Adj Close', 'Volume']]
target = data['Close_Direction']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2,
random_state=42)

# Create a Random Forest classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

OP:

```
Accuracy: 0.6780058651026393
```

MODEL EVALUATION

RANDOM FOREST

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


data = pd.read_csv('MSFT.csv')

data['Close_Direction'] = (data['Close'] - data['Close'].shift(1)) > 0
features = data[['Open', 'High', 'Low', 'Adj Close', 'Volume']]
target = data['Close_Direction']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2,
random_state=42)

# Create a Random Forest classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Make predictions
y_pred = model.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Display evaluation metrics
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", confusion)
print("Classification Report:\n", classification_report_str)
```

OP:

```
Accuracy: 0.6780058651026393
Accuracy: 0.6780058651026393
Confusion Matrix:
 [[587 271]
 [278 569]]
Classification Report:
              precision    recall  f1-score   support

       False       0.68      0.68      0.68       858
        True       0.68      0.67      0.67       847

    accuracy                           0.68      1705
   macro avg       0.68      0.68      0.68      1705
weighted avg       0.68      0.68      0.68      1705
```

NAVIE BAYES

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

data = pd.read_csv('MSFT.csv')

data['Close_Direction'] = (data['Close'] - data['Close'].shift(1)) > 0
features = data[['Open', 'High', 'Low', 'Adj Close', 'Volume']]
target = data['Close_Direction']
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2,
random_state=42)
# Create a Naive Bayes model
model = GaussianNB()
# Train the model
model.fit(X_train, y_train)
# MaKe predictions
y_pred = model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)
# Display evaluation metrics
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", confusion)
print("Classification Report:\n", classification_report_str)
```

OP:

```
Accuracy: 0.4961876832844575
Accuracy: 0.4961876832844575
Confusion Matrix:
 [[123 735]
 [124 723]]
Classification Report:
              precision    recall  f1-score   support

       False       0.50      0.14      0.22       858
        True       0.50      0.85      0.63       847

    accuracy                           0.50      1705
   macro avg       0.50      0.50      0.42      1705
weighted avg       0.50      0.50      0.42      1705
```

KMEANS:

```
from sklearn.metrics import silhouette_score, davies_bouldin_score
import pandas as pd
from sklearn.cluster import KMeans

data = pd.read_csv('MSFT.csv')

features = data[['Open', 'High', 'Low', 'Adj Close', 'Volume']]

k=3
model = KMeans(n_clusters=k)

data['Cluster'] = model.fit_predict(features)

# Evaluate the K-Means model using the Silhouette Score
silhouette_avg = silhouette_score(features, data['Cluster'])
print("Silhouette Score:", silhouette_avg)
# Evaluate the K-Means model using the Davies-Bouldin Index
davies_bouldin = davies_bouldin_score(features, data['Cluster'])
print("Davies-Bouldin Index:", davies_bouldin)
```

OP:

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto'
in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
Silhouette Score: 0.5634415591941591
Davies-Bouldin Index: 0.5877296116966805
```

KNN

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
data = pd.read_csv('MSFT.csv')
data['Close_Direction'] = (data['Close'] - data['Close'].shift(1)) > 0
```

```
features = data[['Open', 'High', 'Low', 'Adj Close', 'Volume']]
target = data['Close_Direction']
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2,
random_state=42)
k=5
model = KNeighborsClassifier(n_neighbors=k)
# Train the model
model.fit(X_train, y_train)
# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model (classification)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Make predictions
y_pred = model.predict(X_test)
# Evaluate the k-NN model
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)
# Display evaluation metrics
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", confusion)
print("Classification Report:\n", classification_report_str)
```

OP:

```
Accuracy: 0.4879765395894428
Accuracy: 0.4879765395894428
Confusion Matrix:
 [[407 451]
 [422 425]]
Classification Report:
              precision    recall  f1-score   support

       False       0.49      0.47      0.48       858
        True       0.49      0.50      0.49       847

    accuracy                           0.49      1705
   macro avg       0.49      0.49      0.49      1705
weighted avg       0.49      0.49      0.49      1705
```