TITLE- Stock Prediction Using LSTM

Team Members-

1. Sudarshan Iyer(22011101113)
2. Sajan A(22011101098)

DATASET DESCRIPTION-

The dataset encompasses a rich historical perspective, capturing the evolution of stock prices for Apple, Google, Microsoft, and Amazon, key players in the technology sector. It spans a substantial timeframe, allowing for in-depth analysis of long-term trends and patterns in stock performance.

Each entry in the dataset not only includes the essential attributes of Date, Open, High, Low, Close, Adj Close, and Volume but also holds valuable information about the market sentiment and investor behavior during specific periods. This nuanced view enables researchers and analysts to explore the impact of various market events, such as earnings announcements, product launches, regulatory changes, or economic shifts, on stock prices.

Moreover, the inclusion of Adjusted Close prices provides a holistic understanding of stock performance by accounting for corporate actions like dividends, stock splits, and other adjustments. This ensures that the dataset accurately reflects the true value of the stocks over time, facilitating more reliable analysis and decision-making.

Additionally, the dataset may incorporate metadata or external factors that could potentially influence stock prices, such as industry news, competitor performance, macroeconomic indicators, and geopolitical events. Integrating such contextual information enhances the dataset's utility for comprehensive analysis and predictive modeling, enabling deeper insights into the underlying dynamics of the stock market.

Furthermore, the availability of high-frequency trading data allows for granular examination of intra-day price movements and trading volumes, providing valuable insights into market liquidity, volatility, and investor sentiment on a more microeconomic level.

Overall, the dataset serves as a valuable resource for researchers, investors, and analysts seeking to gain actionable insights into the behavior of technology stocks, uncovering underlying trends, identifying trading opportunities, and mitigating risks in the dynamic world of finance.

CODE:

```
%pip install pandas_datareader
%pip install yfinance
import pandas as pd
import numpy as np
from pandas_datareader.data import DataReader
import yfinance as yf
from pandas_datareader import data as pdr
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
%matplotlib inline

yf.pdr_override()

# For time stamps
from datetime import datetime
from keras.models import load_model
model = load_model("lstm_model.h5")
# The tech stocks we'll use for this analysis
tech_list = ['AAPL', 'GOOG', 'TSLA', 'AMZN']

# Set up End and Start times for data grab
tech_list = ['AAPL', 'GOOG', 'TSLA', 'AMZN']

end = datetime.now()
start = datetime(end.year - 1, end.month, end.day)
```

```
for stock in tech_list:
    globals()[stock] = yf.download(stock, start, end)



company_list = [AAPL, GOOG, TSLA, AMZN]
company_name = ["APPLE", "GOOGLE", "TESLA", "AMAZON"]

for company, com_name in zip(company_list, company_name):
    company["company_name"] = com_name

df = pd.concat(company_list, axis=0)
df.tail(10)
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Adj Close'].plot()
    plt.ylabel('Adj Close')
    plt.xlabel(None)
    plt.title(f"Closing Price of {tech_list[i - 1]}")

plt.tight_layout()
# We'll use pct_change to find the percent change for each day
for company in company_list:
    company['Daily Return'] = company['Adj Close'].pct_change()

# Then we'll plot the daily return percentage
fig, axes = plt.subplots(nrows=2, ncols=2)
```

```
fig.set_figheight(10)
fig.set_figwidth(15)

AAPL['Daily Return'].plot(ax=axes[0,0], legend=True, linestyle='--', marker='o')
axes[0,0].set_title('APPLE')

GOOG['Daily Return'].plot(ax=axes[0,1], legend=True, linestyle='--', marker='o')
axes[0,1].set_title('GOOGLE')

TSLA['Daily Return'].plot(ax=axes[1,0], legend=True, linestyle='--', marker='o')
axes[1,0].set_title('TESLA')

AMZN['Daily Return'].plot(ax=axes[1,1], legend=True, linestyle='--', marker='o')
axes[1,1].set_title('AMAZON')

fig.tight_layout()
# Grab all the closing prices for the tech stock list into one DataFrame

closing_df = pdr.get_data_yahoo(tech_list, start=start, end=end)['Adj Close']

# Make a new tech returns DataFrame
tech_rets = closing_df.pct_change()
tech_rets.head()
plt.figure(figsize=(12, 10))

plt.subplot(2, 2, 1)
sns.heatmap(tech_rets.corr(), annot=True, cmap='summer')
plt.title('Correlation of stock return')
```

```
plt.subplot(2, 2, 2)
sns.heatmap(closing_df.corr(), annot=True, cmap='summer')
plt.title('Correlation of stock closing price')
#how much value of risk we put by investing in that particular stock.
import numpy as np
import matplotlib.pyplot as plt

# Assuming `tech_rets` is a DataFrame containing returns of different tech stocks

rets = tech_rets.dropna()

area = np.pi * 20

plt.figure(figsize=(10, 8))
plt.scatter(rets.mean(), rets.std(), s=area)
plt.xlabel('Expected return')
plt.ylabel('Risk')

for label, x, y in zip(rets.columns, rets.mean(), rets.std()):
    plt.annotate(label, xy=(x, y), xytext=(50, 50), textcoords='offset points', ha='right', va='bottom',
                 arrowprops=dict(arrowstyle='-', color='blue', connectionstyle='arc3,rad=-0.3'))
    # Print risk values for each stock
    print(f"Risk for {label}: {y}")

plt.show()
# Get the stock quote
def get_stock_choice():
    # List of available stock symbols
    stock_list = ['AAPL', 'GOOG', 'TSLA', 'AMZN']
```

```
    # Prompt the user to select a stock
    print("Which stock would you like to see?")
    print("Available options:", ", ".join(stock_list))

    # Get user input and ensure it's a valid stock symbol
    while True:
        stock_choice = input("Enter the stock symbol: ").strip().upper()
        if stock_choice in stock_list:
            return stock_choice

company =  get_stock_choice()
df = pdr.get_data_yahoo(company , start='2012-01-01', end=datetime.now())
# Show teh data
Df
plt.figure(figsize=(16,6))
plt.title('Close Price History')
plt.plot(df['Close'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.show()
# Create a new dataframe with only the 'Close column
data = df.filter(['Close'])
# Convert the dataframe to a numpy array
dataset = data.values
# Get the number of rows to train the model on
training_data_len = int(np.ceil( len(dataset) * .95 ))
```

```
from sklearn.preprocessing import MinMaxScaler


scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)


# Create the training data set
# Create the scaled training data set
train_data = scaled_data[0:int(training_data_len), :]
# Split the data into x_train and y_train data sets
x_train = []
y_train = []


for i in range(60, len(train_data)):
   x_train.append(train_data[i-60:i, 0])
   y_train.append(train_data[i, 0])
   if i<= 61:
      print(x_train)
      print(y_train)
      print()


# Convert the x_train and y_train to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)


# Reshape the data
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
# x_train.shape
# Create the testing data set
# Create a new array containing scaled values from index 1543 to 2002
test_data = scaled_data[training_data_len - 60: , :]
```

```
# Create the data sets x_test and y_test
x_test = []
y_test = dataset[training_data_len:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

# Convert the data to a numpy array
x_test = np.array(x_test)

# Reshape the data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1 ))

# Get the models predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

# Get the root mean squared error (RMSE)
rmse = np.sqrt(np.mean(((predictions - y_test) ** 2)))
rmse
```

STOCK PRICE PREDICTION USING LSTM

```
# Plot the data
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions
# Visualize the data
plt.figure(figsize=(16,6))
plt.title('Model Using LSTM for ' + company + ' Stock')
plt.xlabel('Date', fontsize=18)
```

plt.ylabel('Close Price USD ($)', fontsize=18)

plt.plot(train['Close'])

plt.plot(valid[['Close', 'Predictions']])

plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')

plt.show()

STOCK PRICE PREDICTION USING SVR

Import SVM from scikit-learn

from sklearn.svm import SVR

1. Prepare the data

We'll use the same 'Close' prices data as before

data = df.filter(['Close'])

dataset = data.values

Scale the data

scaler = MinMaxScaler(feature_range=(0,1))

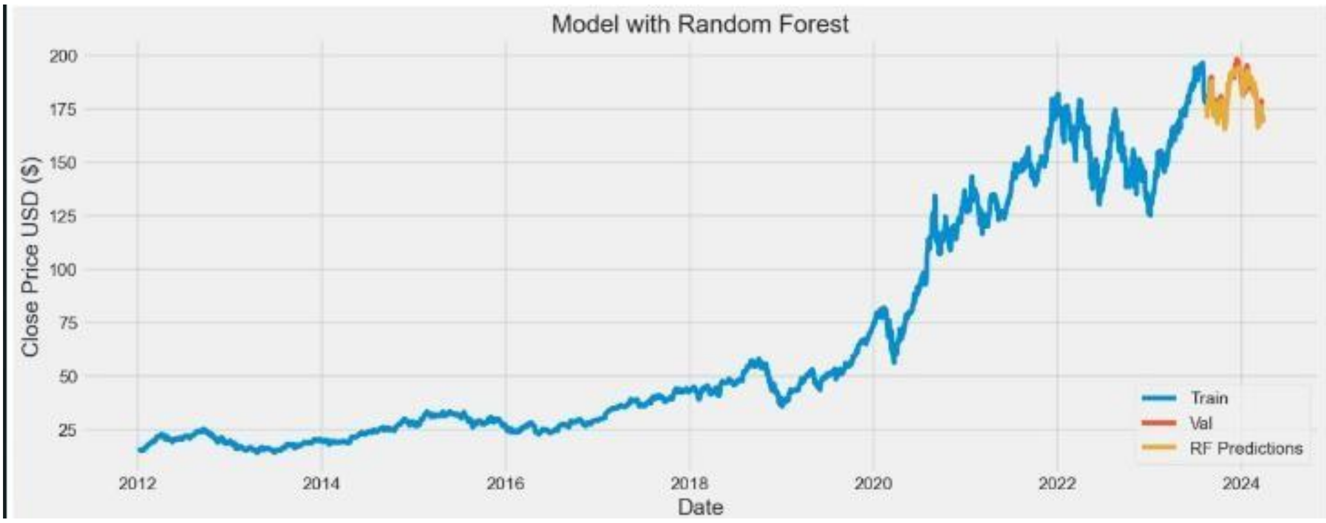
scaled_data = scaler.fit_transform(dataset)

```
# Prepare the training data
training_data_len = int(np.ceil(len(dataset) * 0.95))
train_data = scaled_data[0:training_data_len, :]
x_train = []
y_train = []

for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1]))

# 2. Train the SVM model
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
svr_rbf.fit(x_train, y_train)

# 3. Prepare the testing data
test_data = scaled_data[training_data_len - 60:, :]
x_test = []
y_test = dataset[training_data_len:, :]

for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1]))

# 4. Make predictions using the trained model
```

```
svm_predictions = svr_rbf.predict(x_test)

svm_predictions = svm_predictions.reshape(-1, 1)

svm_predictions = scaler.inverse_transform(svm_predictions)


# 5. Visualize the predictions

train = data[:training_data_len]

valid = data[training_data_len:]

valid['SVM Predictions'] = svm_predictions


plt.figure(figsize=(16,6))

plt.title('Model with SVR')

plt.xlabel('Date', fontsize=18)

plt.ylabel('Close Price USD ($)', fontsize=18)

plt.plot(train['Close'])

plt.plot(valid[['Close', 'SVM Predictions']])

plt.legend(['Train', 'Val', 'SVM Predictions'], loc='lower right')

plt.show()
```

STOCK PRICE PREDICTION USING RANDOM FOREST.

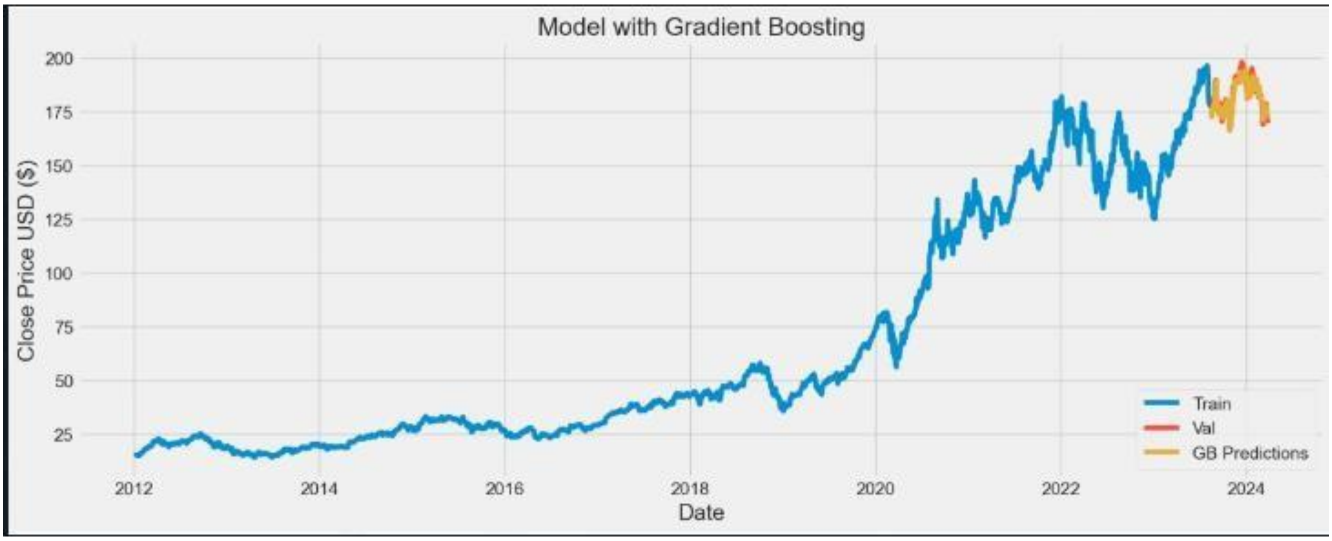
```
from sklearn.ensemble import RandomForestRegressor

# Train the Random Forest model
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(x_train, y_train)

# Make predictions using the trained Random Forest model
rf_predictions = rf_regressor.predict(x_test)
rf_predictions = rf_predictions.reshape(-1, 1)
rf_predictions = scaler.inverse_transform(rf_predictions)

# Visualize the predictions made by Random Forest
valid['RF Predictions'] = rf_predictions

plt.figure(figsize=(16,6))
plt.title('Model with Random Forest')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.plot(train['Close'])
plt.plot(valid[['Close', 'RF Predictions']])
plt.legend(['Train', 'Val', 'RF Predictions'], loc='lower right')
plt.show()
```



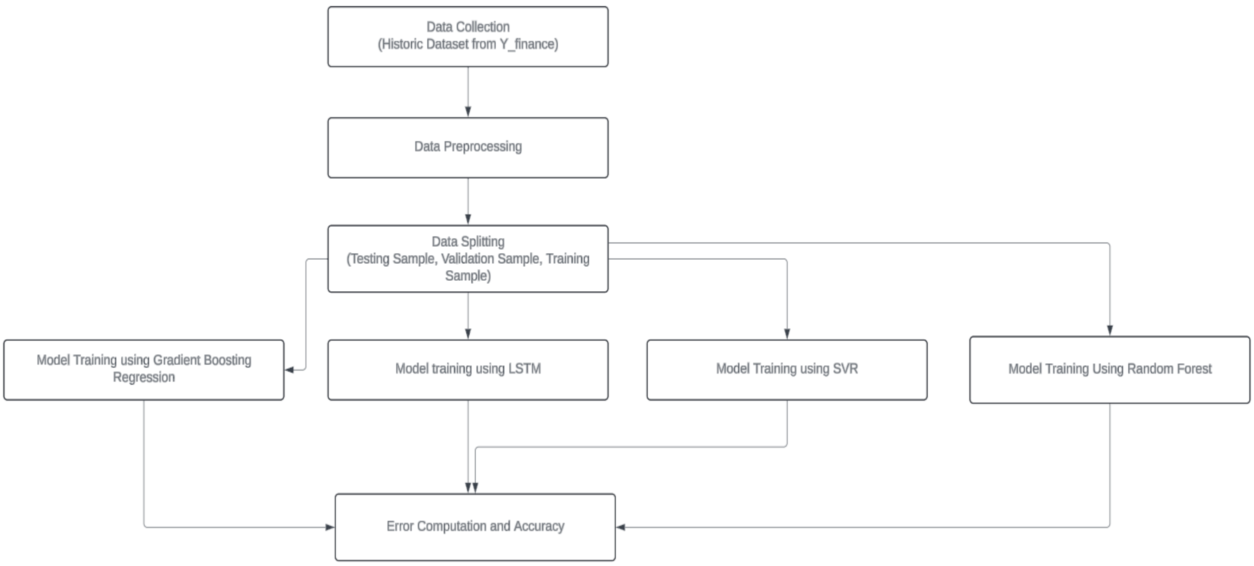
STOCK PRICE PREDICTION USING GRADIENT BOOSTING REGRESSOR

```
from sklearn.ensemble import GradientBoostingRegressor

# Train the Gradient Boosting model
gb_regressor = GradientBoostingRegressor(n_estimators=100, random_state=42)
gb_regressor.fit(x_train, y_train)

# Make predictions using the trained Gradient Boosting model
gb_predictions = gb_regressor.predict(x_test)
gb_predictions = gb_predictions.reshape(-1, 1)
gb_predictions = scaler.inverse_transform(gb_predictions)

# Visualize the predictions made by Gradient Boosting
valid['GB Predictions'] = gb_predictions

plt.figure(figsize=(16,6))
plt.title('Model with Gradient Boosting')
plt.xlabel('Date', fontsize=18)
```

```
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.plot(train['Close'])
plt.plot(valid[['Close', 'GB Predictions']])
plt.legend(['Train', 'Val', 'GB Predictions'], loc='lower right')
plt.show()
```



DIAGRAM



COMPARISON OF RESULTS WITH 3 OTHER DIFFERENT MODELS.

- When compared with the LSTM model, the RANDOM FOREST AND GRADIENT BOOSTING REGRESSOR gives more accurate predictions.
- While LSTM gave a decent prediction with an average error of 4.96 for all the four stocks, SVR could not provide accurate results and varied vastly for each stock.
- LSTM MODEL tasks were computationally intensive when compared to other models.
- Due to Ensemble learning in Random Forest, the risk of overfitting is reduced, hence provide better and accurate results.
- Gradient boosting Regressor gave the overall best results as it handles non-linear relationship well.
- Since a very higher dimensional space was not used and due to its high sensitivity to the choice of kernel and hyperparameters, SVR was not very effective here.

```
LSTM Metrics:
MSE: 24.467048390531072
MAE: 4.011696809257558
RMSE: 4.946417733120715

SVR Metrics:
MSE: 141.5549257409681
MAE: 10.559970848931725
RMSE: 11.89768573046742

Random Forest Metrics:
MSE: 7.350229437044029
MAE: 2.195120206346693
RMSE: 2.711130656579286

Gradient Boosting Metrics:
MSE: 9.526925621723333
MAE: 2.469062964780886
RMSE: 3.0865718235160724
```

In summary, the selection of the most suitable model for stock value prediction hinges on a comprehensive consideration of factors such as dataset characteristics, computational resources, and the specific requirements of the prediction task, with LSTM, Random Forest, and Gradient Boosting Regressor emerging as prominent candidates warranting careful consideration.

CONCLUSION:

In conclusion, for tasks such as stock value prediction, our analysis underscores the efficacy of LSTM as a robust model, demonstrating its capability to generate highly accurate predictions. Furthermore, both Random Forest and Gradient Boosting Regressor exhibited exceptional performance, consistently delivering minimal prediction errors and thereby establishing themselves as leading contenders in this domain.

However, it is noteworthy that while Random Forest and Gradient Boosting Regressor excelled in minimizing errors, their comparative performance with LSTM in terms of overall prediction accuracy warrants further investigation, particularly in the context of different datasets and feature selections.

Conversely, the Support Vector Regressor (SVR) model, while a viable option in certain scenarios, displayed comparatively inferior predictive capabilities in our evaluation, highlighting its limitations in capturing the intricacies of stock value dynamics effectively.

References:

https://www.sciencedirect.com/science/article/pii/S1877050920307924

https://www.researchgate.net/publication/341482418_A_Survey_on_Stock_Market_Prediction_Using_Machine_Learning_Techniques