STOCK MOVEMENT ANALYSIS

This project uses Python to scrape stock-related data from Telegram, preprocesses the data, and trains a machine learning model to predict stock movements. The notebook demonstrates scraping, preprocessing, training, and evaluation.

Prerequisites:

- Python 3.8 or higher

- Jupyter Notebook or Google Colab

- Libraries: `telethon`, `nest_asyncio`, `pandas`, `numpy`, `scikit-learn`

Setup Instructions:

1. Install required Python libraries using:

```bash
pip install telethon nest_asyncio pandas numpy scikit-learn
```

2. Obtain Telegram API credentials (`api_id` and `api_hash`) from [Telegram](https://my.telegram.org/).

Running the Notebook:

2. Follow the cells step by step:

  -Library Installation: Install dependencies.

  - Telegram Scraping: Input your `api_id` and `api_hash` to connect to Telegram.

  - Data Preprocessing: Prepare the data for model training.

  - Model Training: Train the `RandomForestClassifier` on processed data.

  - Evaluation: Evaluate the model's performance with accuracy metrics.

Project Workflow:

1. Data Scraping:

  - Connects to a Telegram channel to fetch stock-related messages.

  - Extracts relevant data (e.g., stock prices).

Here we are installing set of libraries

Step 1: !pip install telethon

Step 2: !pip install nest_asyncio

Step 3: import nest_asyncio

from telethon import TelegramClient

import asyncio

```
import os

# Apply nest_asyncio to allow nested event loops
nest_asyncio.apply()

# Replace with your values
api_id = '24784796'
api_hash = 'c3b056cb039b706a1284973a3bf42435'
group_or_channel = '@STOCKGAINERSS'

# Define session file name
session_file = 'my_session.session'

async def fetch_messages():
    # Use an async context manager
    async with TelegramClient(session_file, api_id, api_hash) as client:
        # Fetch messages asynchronously
        messages = await client.get_messages(group_or_channel, limit=100)

        # Process and print messages
        for message in messages:
            print(f"Date: {message.date}, Content: {message.text}")

# Call the async function using `await` in Colab
await fetch_messages()
```

Step 4: import pandas as pd

```
from telethon import TelegramClient

# Replace with your values
api_id = '24784796'
api_hash = 'c3b056cb039b706a1284973a3bf42435'
```

```
group_or_channel = '@STOCKGAINERSS'  # Replace with your correct username or ID
session_file = 'my_session.session'

# Function to fetch messages
async def fetch_messages():
    async with TelegramClient(session_file, api_id, api_hash) as client:
        # Fetch messages
        messages = await client.get_messages(group_or_channel, limit=100)

        # Process the messages into a list of dictionaries
        data = [{"date": msg.date, "text": msg.text} for msg in messages]

        # Convert the list of dictionaries into a DataFrame
        df = pd.DataFrame(data)

        # Save to CSV
        df.to_csv("stock_data.csv", index=False)
        print("Data saved to stock_data.csv")

# Run the fetch_messages function
await fetch_messages()
```

Note: By the executing the above code will extract data from telegram public channels regarding about stocks and the extracted data will be saved CSV format for further analysis

2. Data Preprocessing:

- Cleans and formats the scraped data.
- Extracts numerical values and removes outliers/missing values.

Step 1:

```
# Load the CSV file into a DataFrame
df = pd.read_csv("stock_data.csv")

# Display the first few rows of the DataFrame
```

print(df.head())

Step 2:

```
from google.colab import files

# Download the file to your local machine
files.download('stock_data.csv')
```

Step 3:

```
def extract_price(text):
    # Ensure the text is a string before attempting to split
    if isinstance(text, str):  # Check if text is a string
        try:
            return float(text.split()[0])  # Attempt to extract the price from the first word
        except (ValueError, IndexError):
            return np.nan  # If extraction fails, return NaN
    return np.nan  # If the text is not a string, return NaN
```

Step 4:

```
# Inspect the unique values or data types in the 'text' column
print(df['text'].dtype)  # Check data type of the 'text' column
print(df['text'].head())  # Preview the first few values
```

Step 5:

```
# Apply the updated function to create the 'Price' column
df['Price'] = df['text'].apply(extract_price)

# Drop rows with missing prices
df = df.dropna(subset=['Price'])

# Continue with the rest of your process...
```

3. Model Training:

- Splits data into training and testing sets.
- Trains a `RandomForestClassifier` to predict stock movement.

Step 1:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load data (assuming 'stock_data.csv' contains stock prices)
df = pd.read_csv("stock_data.csv")

# Check if 'Price' column already exists
if 'Price' not in df.columns:
    # Function to extract price from text, handling non-numeric cases
    def extract_price(text):
        # Check if text is a string and if not, try converting to string
        if not isinstance(text, str):
            text = str(text)
        try:
            # Attempt to split the text and convert the first word to float
            return float(text.split()[0])
        except (ValueError, IndexError):
            # If conversion fails or there's no first word, return NaN
            return np.nan

    # Apply the function to create the 'Price' column
    df['Price'] = df['text'].apply(extract_price)

# Drop rows with missing prices
df = df.dropna()

# Create the target variable - Predict if price goes up or down
df['Price Change'] = np.where(df['Price'].shift(-1) > df['Price'], 1, 0)  # 1 if price goes up, 0 if down
```

```python
# Features and target
X = df[['Price']]  # You might need to engineer features from 'text'
y = df['Price Change']

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

4. Evaluation:

   - Computes accuracy and generates classification reports.

Step 1:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize the model
model = RandomForestClassifier(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
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

5. Results Export:

- Saves processed data and predictions for further analysis.

Step 1:

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
import yfinance as yf  # For fetching stock data

# Download historical stock data
stock_data = yf.download('AAPL', start='2020-01-01', end='2023-01-01')

# Preprocess the stock data (e.g., create moving averages)
stock_data['5_day_MA'] = stock_data['Close'].rolling(window=5).mean()
stock_data['50_day_MA'] = stock_data['Close'].rolling(window=50).mean()
stock_data['Volume'] = stock_data['Volume']

# Create the target variable (1 if price goes up, 0 if down)
stock_data['Price Change'] = np.where(stock_data['Close'].shift(-1) > stock_data['Close'], 1, 0)

# Drop rows with missing values
stock_data = stock_data.dropna()

# Features and target
X = stock_data[['Close', '5_day_MA', '50_day_MA', 'Volume']]
y = stock_data['Price Change']

# Feature scaling
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Build and train the model
model = RandomForestClassifier(n_estimators=100, random_state=42)
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(f"Accuracy: {accuracy}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Final Report :

```
Accuracy: 0.4859154929577465
Classification Report:
              precision    recall  f1-score   support

           0       0.48      0.47      0.47        70
           1       0.49      0.50      0.50        72

    accuracy                           0.49       142
   macro avg       0.49      0.49      0.49       142
weighted avg       0.49      0.49      0.49       142
```

Notes:

- Ensure the Telegram group/channel is accessible.

- Use the notebook's visualization for better insights into data and model performance.