|BIG DATA QUIZ 7 - DATA STREAMS|

|SETUP|

Questions 1-4: Code for the setup is written below. API Key is being pulled from the 'keys.py' file.
Question 5: The stock data for each stock are stored in .csv files and are being submitted separately

```
try:
    import twelvedata
except ModuleNotFoundError:
    import os
    os.system("pip install
twelvedata[pandas,matplotlib,plotly,websocket-client]")
    import twelvedata

import pandas as pd
import time
from datetime import datetime, timedelta
from datetime import datetime
import os
import sys

#Getting my api key from a separate file
current_dir = os.path.dirname(os.path.abspath('keys.py'))
if current_dir not in sys.path:
    sys.path.append(current_dir)
from keys import twelveDataKey as api_key

# function to get stock data
def collectStockData(symbol, api_key, start_date, end_date):
    td = twelvedata.TDClient(api_key)
    # Pull data for the stock symbol between start_date and end_date
at 15-minute intervals
    data = td.time_series(symbol=symbol, interval="15min",
start_date=start_date, end_date=end_date, outputsize=5000)
    return data.as_pandas()

# I split the last 4.5 years into 7 consecutive time periods and ran
this code
# on each one and then compiled all the csvs together for each stock
# I did this to avoid going over my per minute API usage
start_date_str = "2020-11-17"
end_date_str = "2021-07-18"
time_section = 6

stocks = ['AAPL', 'MSFT', 'IBM']
all_stock_data = {}

for stock in stocks:
```

```
    print(f"Fetching data for {stock} from {start_date_str} to
{end_date_str}")
    all_stock_data[stock] = collectStockData(stock, api_key,
start_date_str, end_date_str)

download_dir = "C:/Users/jacks/Downloads/"


# Saving data as CSV
for stock, data in all_stock_data.items():
    file_path = os.path.join(download_dir,
f"{stock}_data{time_section}.csv")
    data.to_csv(file_path)
    print(f"Data for {stock} saved to {file_path}")

Fetching data for AAPL from 2020-04-17 to 2020-11-17
Fetching data for MSFT from 2020-04-17 to 2020-11-17
Fetching data for IBM from 2020-04-17 to 2020-11-17
Data for AAPL saved to C:/Users/jacks/Downloads/AAPL_data6.csv
Data for MSFT saved to C:/Users/jacks/Downloads/MSFT_data6.csv
Data for IBM saved to C:/Users/jacks/Downloads/IBM_data6.csv
```

|ALGROTHMIC STOCK TRADING|

1. If I wanted to purchase 1,000,000 shares of a stock without letting the market know there are a few strategies I could employ. The first way might be to do a large portion of the trade through a broker in the form of a block trade. Something like this would likely be arranged privately and then the brokers would go and use their networks to get it done. Another tactic I would employ would be using trading strategies like Volume Weighted Average Price, where I reduce the impact on the market by distributing the trades across various time periods in order to match the average market price. Similarily, a Time Weighted Average Price strtegy by distributing the trades over an even period of time no matter the volume fluctuations involved. Finally, I could employ a routing system that routes parts of the larger order to different venues which not only reduces the volume each venue has to handle but also allows them to try for the lowest buying prices.

2. If I was looking to identify somene that was dumping a certain stock there are also a few different tactics that I would try and use. Naturally, the first thing I woul dbe keeping an eye on is the trading volume, specifically looking for spikes over short periods or consistently high averages over longer periods relative to the past average for the stock. Comparing the relative price of the stock to the trading volume could also help, as a stcok having a high volume but it's price still consistently dropping would indicate to me that it is being dumped. Finally, I would also utilize the Volume Weighted Avergae Price and compare it to the price the stock is being sold at, because selling at a price significantly below the Volume Weighted AVerage Price would be strong sig that soemone is urgently trying to get rid of a

stock, becuase they don't care about the money they are losing selling below the average price as long as they get the stock sold.

| TECHNICAL ANALYSIS OF STOCK TRADING|

For the final question I used the code below (new-stock-price-feeder.py) and ran in with spark in a cluster. I was getting an error with setting up the cluster for a while, but once I finally got the code to run it would stop after only comparing the moving averages a couple times. I don't have the last buy sell recommendations, but here is still the code I used to run all the commands in questions 11-14.

```
#new-stock-price-feeder.py
try:
    import twelvedata
except ModuleNotFoundError:
    import os
    os.system("pip install
twelvedata[pandas,matplotlib,plotly,websocket-client]")
    import twelvedata

import pandas as pd
import time
from datetime import datetime, timedelta
from datetime import datetime
import os
import sys
import requests

# Set your TwelveData API key
current_dir = os.path.dirname(os.path.abspath('keys.py'))
if current_dir not in sys.path:
    sys.path.append(current_dir)
from keys import twelveDataKey as api_key

# Define stock symbols and settings
symbols = ["MSFT", "AAPL"]
start_date = "2020-01-01"
end_date = "2020-12-31"
interval = 5
init_delay_seconds = 30

#Using the Twelvedata API
def fetch_historical_data(symbol, start_date, end_date):
    url = f"https://api.twelvedata.com/time_series"
    params = {
        "symbol": symbol,
        "interval": "1day",
        "start_date": start_date,
        "end_date": end_date,
```

```
        "apikey": API_KEY,
    }
    response = requests.get(url, params=params)
    response.raise_for_status()
    data = response.json()

    if "values" in data:
        df = pd.DataFrame(data["values"])
        df["datetime"] = pd.to_datetime(df["datetime"])
        df.set_index("datetime", inplace=True)
        df = df.sort_index()
        return df["close"].astype(float)
    else:
        print(f"Error fetching data for {symbol}: {data.get('message',
'Unknown error')}", file=sys.stderr)
        return pd.Series()

# Fetch data for both stocks
print("Fetching historical data...", file=sys.stderr)
msft_prices = fetch_historical_data("MSFT", start_date, end_date)
aapl_prices = fetch_historical_data("AAPL", start_date, end_date)

if msft_prices.empty or aapl_prices.empty:
    print("Failed to fetch data. Exiting.", file=sys.stderr)
    sys.exit(1)

#Combining data into a single df
prices_df = pd.DataFrame({"MSFT": msft_prices, "AAPL": aapl_prices})
prices_df.dropna(inplace=True)  # Drop rows with missing data

#Printing initializatin info
print(f"Sending daily MSFT and AAPL prices from {start_date} to
{end_date}...", file=sys.stderr)
print(f"... each day's data sent every {interval} seconds ...",
file=sys.stderr)
print(f"... beginning in {init_delay_seconds} seconds ...",
file=sys.stderr)

time.sleep(init_delay_seconds)

for date, row in prices_df.iterrows():
    print(f"{date.date()}\t{row['MSFT']:.4f}\t{row['AAPL']:.4f}",
flush=True)
    time.sleep(interval)
```