Scenario: You are a data engineer for a hedge fund that trades based on technical analysis. Your team needs a robust pipeline to process and analyze large volumes of historical stock data efficiently. You are tasked with building a data pipeline that cleans, transforms, and aggregates Open, High, Low, and Close (OHLC) data for further analysis and model development.

Data Source:

•⁠ ⁠Your company receives daily OHLC data feeds for multiple stocks in various formats (e.g., CSV, JSON).

•⁠ ⁠Eg: Yahoo Finance/yfinance - try 5-10 equities as a sample dataset

Tasks:

1.⁠ ⁠Data Ingestion:

•⁠ ⁠Develop a script that can ingest OHLC data feeds from various sources and formats.

•⁠ ⁠Validate the data integrity (e.g., check for missing values, outliers, data type consistency).

•⁠ ⁠Standardize the data format to a common structure (e.g., pandas DataFrame).

2.⁠ ⁠Data Cleaning:

•⁠ ⁠Identify and handle missing values (e.g., imputation, removal).

•⁠ ⁠Detect and correct outliers using statistical methods or domain knowledge.

•⁠ ⁠Address any inconsistencies in timestamps or date formats.

3.⁠ ⁠Data Transformation:

•⁠ ⁠Calculate technical indicators based on OHLC data (e.g., moving averages, Bollinger Bands, Relative Strength Index).

•⁠ ⁠Apply feature engineering techniques to create new features relevant for your trading strategy (e.g., volatility measures, price patterns).

•⁠ ⁠Resample the data based on desired frequencies (e.g., daily to hourly).

4.⁠ ⁠Data Validation:

•⁠ ⁠Implement unit tests to ensure the pipeline's functionality and data integrity.

•⁠ ⁠Monitor the pipeline for errors and data quality issues.

5.⁠ ⁠Data Storage:

•⁠ ⁠Use a simple DB to store this (such as Sqlite, Mysql etc)

•⁠ ⁠Partition the data by year, month, or another relevant category for efficient querying.

•⁠ ⁠Optimize the data format for fast retrieval and analysis (e.g., columnar format).

Bonus:

•⁠ ⁠Implement data compression techniques to reduce storage costs.

•⁠ ⁠Integrate the pipeline with a visualization tool to explore the data interactively.

•⁠ ⁠Develop data quality checks and alerts to proactively identify and address issues.

Deliverables:

•⁠ ⁠A well-documented Python script implementing the data pipeline.

•⁠ ⁠Example output showing cleaned and transformed data for a specific period.

•⁠ ⁠(Optional) Visualization or report showcasing key insights from the processed data.

Technologies:

•⁠ ⁠Python libraries: pandas, numpy, Dask (for large datasets), data validation libraries (e.g., pytest)

•⁠ ⁠Cloud storage or data warehouse (e.g., AWS S3, Google Cloud Storage, Snowflake) (optional)

•⁠ ⁠Visualization tools (e.g., Tableau, Power BI) (optional)

Code:

task.py

import pandas as pd

import numpy as np

import yfinance as yf

from datetime import datetime

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler

import sqlite3

# Function to ingest OHLC data from Yahoo Finance

def fetch\_stock\_data(ticker, start\_date, end\_date):

df = yf.download(ticker, start=start\_date, end=end\_date)

return df

# Function to handle missing values

def handle\_missing\_values(df):

# Impute missing values using mean or any other strategy

imputer = SimpleImputer(strategy='mean')

df\_filled = pd.DataFrame(imputer.fit\_transform(df), columns=df.columns, index=df.index)

return df\_filled

# Function to detect and correct outliers

def handle\_outliers(df):

# Implement outlier detection and correction techniques

# For example, remove rows where prices deviate significantly from the mean

return df

# Function to standardize date format

def standardize\_date\_format(df):

# Convert index to datetime format if not already

if not isinstance(df.index, pd.DatetimeIndex):

df.index = pd.to\_datetime(df.index)

return df

# Function to calculate technical indicators

def calculate\_technical\_indicators(df):

# Implement calculation of technical indicators such as moving averages, RSI, etc.

# Example:

df['SMA\_20'] = df['Close'].rolling(window=20).mean()

return df

# Function to resample data

def resample\_data(df, frequency='D'):

# Resample the data based on desired frequency (default: daily)

df\_resampled = df.resample(frequency).agg({'Open':'first', 'High':'max', 'Low':'min', 'Close':'last', 'Volume':'sum'})

return df\_resampled

# Function to store data in SQLite database

def store\_data\_in\_db(df, db\_name='stock\_data.db', table\_name='stock\_data'):

conn = sqlite3.connect(db\_name)

df.to\_sql(table\_name, conn, if\_exists='replace')

conn.close()

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

tickers = ['AAPL', 'MSFT', 'GOOGL']

start\_date = '2023-01-01'

end\_date = '2024-01-01'

for ticker in tickers:

# Ingest data

df = fetch\_stock\_data(ticker, start\_date, end\_date)

# Data cleaning

df = handle\_missing\_values(df)

df = handle\_outliers(df)

df = standardize\_date\_format(df)

# Data transformation

df = calculate\_technical\_indicators(df)

df = resample\_data(df)

# Data storage

store\_data\_in\_db(df)

Stack\_data.db is created.

Output:

