חלק א - סעיף ב

**שאלה 3:**

במקרה שבו הנתונים אינם מגיעים כקובץ מלא אלא כזרם (stream) רציף של נתונים, יש לבצע עיבוד בזמן אמת תוך כדי צריכת הנתונים. כלומר, אי אפשר להחזיק את כל המידע בזיכרון כי אולי יהיו מליוני שורות בהמשך ולכן צריך לעשות את כל החישובים (כמו ממוצע שעתי או דו שעתי) תוך כדי...ולכן הרעיון המרכזי הוא – במקום לשמור את כל הנתונים , אני שומרת רק את מה שצריך בשביל לחשב את הממוצע השעתי וכשזמן עובר- אני זורקת את השורות שכבר לא רלוונטיות כך תמיד יש "חלון נגלל" של השעתיים האחרונות נניח ( אם החלטנו ללכת על ממוצע דו שעתי) ואז כל פעם כשמגיעה שורה חדשה אני בודקת אם צריך לעדכן את הממוצע ואם כן שומרת אותו. וזה טוב כי כך אנו לא מחזיקים את כל השורות בזיכרון ואפשר לחשב את הממוצע בreal time בלי לחכות לקובץ שלם, זה מתאים למערכות שמנטרות מידע "חי"...(מזג אויר, תעבורת רשת וכו..) פתרון כזה דורש עיבוד אינקרמנטלי, תוך שימוש במשאבים קבועים (כגון זיכרון וזמן עיבוד) גם כאשר כמות הנתונים מצטברת ללא הגבלה.

וכעת, אכתוב פירוט בשלבים של צורת הפתרון-

אגדיר חלון זמן נגלל (sliding window) בגודל של שעתיים(נניח).

אשתמש במבנה נתונים מסוג deque או queue לאחסון רשומות בטווח הזמן הרלוונטי בלבד.

עם הגעת כל רשומה חדשה, אעדכן את רשימת הרשומות הפעילות, ואסיר רשומות ישנות שאינן בטווח הזמן.

אחזיק משתנים כגון סכום (sum) ומספר מופעים (count) כדי לחשב את הממוצע השעתי מבלי לשמור את כל הנתונים.

הפלט יתעדכן בזמן אמת (real-time) לכל שעה מלאה שעברה.

היתרונות המרכזיים:

ניצול זיכרון קבוע – אין צורך לשמור את כל ההיסטוריה- כי אם נשמור את כל ההיסטוריה (שכבר לא רלוונטית לנו) הזיכרון במחשב עלול להגמר...ולכן אנו שומרים רק את מה שאנחנו צריכים כדי לחשב את הממוצע המבוקש (שעתיים וכו..) וכל השאר- זורקים.

מתאים למערכות stream כמו Kafka, MQTT או socket. שאלו הן מערכות שמעבירות נתונים ב"שידור חי" ולא דרך קובץ.. כמו משחקים, צ'אט בין שרת ללקוח, פרוטוקולים להעברת מידע בין חיישנים ועוד...

ולכן המערכת צריכה להיות מוכנה לתרחישים מסוג זה של לדעת לעבד שורה שורה מבלי לחכות לקובץ שלם.

פתרון רספונסיבי המתאים לדשבורדים, ניטור והתרעות.- פתרון רספונסיבי זהו פתרון שמגיב מהר כלומר אם רוצים לראות את המידע מתעדכן כל שניה זה מצוין, לא צריך לחכות שיסיימו לקרוא קובץ שלם. טוב למערכות שמנטרות בזמן אמת נתונים (רשתות ותקשורת, מעבר של חבילות ברשת ועוד..)

**שאלה 4:**

כדי להתאים את הקוד לפורמט Parquet, אשתמש בפונקציית pandas.read\_parque () לצורך קריאה, וב־to\_parquet() לצורך שמירה. אזהה את סוג הקובץ לפי סיומת (.parquet/.csv) או לפי סדר עדיפויות (Parquet עדיף), כך שהקוד יתמוך בגמישות בפורמטים השונים.

השוואה בין הפורמטים – CSV לעומת Parquet:

מבנה- parquet הוא מבנה של עמודות בעוד שקובץ csv הוא במבנה של שורות

דחיסה- קובץ parquet הוא דחוס מאד (שזה מועדף לזיכרון) ועל כן גודל הקובץ האופייני הוא קטן משמעותית בעוד שcsv אינו דחוס או דחוס בצורה מוגבלת ולכן גודלו בזיכרון- גדול...

מהירות קריאה- בקובץ parquet הגישה לנתונים היא מהירה במיוחד בגלל הגישה לפי העמודות ולכן הקריאה מהירה בעוד שבcsv הקריאה היא איטית יחסית..

תמיכה בטיפוסים וסכימות - קובץ parquet תומך בכל סוגי הטיפוסים ולכן שומר טיפוסי מידע מדויקים ועל כן שומר את הסכימה באופן מפורש כלומר כל עמודה כוללת את הטיפוס ולכן גם קל לשלב עם מסדי נתונים וכלים מתקדמים וזה חשוב כשעובדים עם data ענק וצריכים דיוק באיך שמפרשים את המידע , בעוד שקובץ csv שומר את כל הערכים כמחרוזות ולא שומר סכימות...

סה''כ קובץ parquet גם גמיש לתשתיות של big data הוא אידיאלי לעבודה עם spark(פלטפורמת קוד פתוח שמעבדת נתונים בbig data) ועוד בעוד שקובץ csv מוגבל לכל אלה...

סה''כ--- יתרונות Parquet:

חיסכון במקום אחסון משמעותי – תוצאה של דחיסה לפי עמודות.

ביצועים גבוהים: במיוחד כשנדרש לקרוא רק חלק מהעמודות

שילוב קל עם מערכות Big Data ותמיכה בענן (S3, Azure Data Lake וכו').

יעיל יותר לעיבוד מקבילי או כשעובדים עם קבצים מאוד גדולים.

בפרויקט הנוכחי, העדפתי תחילה קבצי Parquet אם קיימים, ובהיעדרם עברתי ל־CSV או Excel תוך המרה. הגישה הזו מאפשרת קוד גמיש ותומך בפורמטים מגוונים מבלי לדרוש שינוי מצד המשתמש.