ניתוח מכירות ופילוח שוק של חנות iTunes

גבריאל מזרחי | עילאי חביב | רונן לסניק

תוכן עניינים

1. מבוא

רקע והקשר :

בפרויקט זה אנו עוסקים בניתוח מכירות ופילוח שוק של חנות iTunes.
חנות iTunes היא אחת מהפלטפורמות הגדולות ביותר למכירת מוזיקה דיגיטלית, עם מיליוני שירים ואלבומים זמינים לרכישה בכל רחבי העולם. מכירות המוזיקה בחנות iTunes מהוות מקור הכנסה חשוב לאמנים ולחברות תקליטים, ומספקות נתונים עשירים שמאפשרים לנתח מגמות ותבניות בשוק המוזיקה הדיגיטלית.

2. מסקנות

במהלך הפרויקט הגענו למספר תובנות מרכזיות:

זיהוי דפוסי מכירה עיקריים:
זיהינו מגמות במכירות שירים ואלבומים, כולל הפלייליסטים הפופולריים והשירים המובילים בז'אנרים שונים. ממצאים אלה יכולים לסייע למנהלי השיווק ולמחלקות המכירה למקד את מאמציהם במוצרים הפופולריים ביותר ולהתאים את האסטרטגיות השיווקיות בהתאם.

מערכת המלצות מותאמת אישית:
פיתחנו מערכת שממליצה ללקוחות על שירים ואלבומים בהתבסס על העדפותיהם המוזיקליות והרכישות הקודמות שלהם. מערכת זו מאפשרת חווית משתמש מותאמת אישית ומשפרת את שביעות הרצון של הלקוחות, מה שיכול להוביל להגדלת המכירות.

השפעות עונתיות במכירות:
זיהינו מגמות עונתיות במכירות ז'אנרים שונים והבנו כיצד עונות השנה משפיעות על הביקוש למוזיקה. תובנות אלה מאפשרות למנהלי המכירות להיערך טוב יותר לקראת עונות השיא ולתכנן את המלאי בהתאם.

המלצות אסטרטגיות:
ניתוח הנתונים אפשר לנו להציע מספר המלצות לשיפור האסטרטגיות השיווקיות. לדוגמה, אנו ממליצים להתמקד בקמפיינים פרסומיים ממוקדים במהלך עונות השיא, ולהשתמש במערכת ההמלצות כדי להציע ללקוחות מוצרים נוספים שמתאימים להעדפותיהם האישיות.

3. ניתוח נתונים

2.3 - עיבוד נתונים - בפייתון

ביצענו התקנה של הספריות pandas ו-SQLAlchemy, וייבאנו אותן לפרויקט.
יצרנו חיבור למסד הנתונים PostgreSQL באמצעות הפונקציה create_engine.
החיבור נבדק על ידי הרצת שאילתה לדוגמה.

קראנו את קובץ המחלקות באמצעות הפונקציה pd.read_csv וטענו את הנתונים ל-DataFrame.
בנוסף, קראנו שני קבצים בפורמט JSON באמצעות הפונקציה pd.read_json.
כל אחד מהקבצים נטען ל-DataFrame נפרד לעיבוד נוסף.

חיברנו את שני ה-DataFrames של קובצי ה-JSON באמצעות הפונקציה pd.concat על ציר השורות (axis=0).
מחקנו עמודות מיותרות כמו sub_dep_id ו-sub_dep_name בעזרת הפונקציה drop.

חישבנו את התקציב הכולל לכל מחלקה ראשית באמצעות הפונקציות groupby ו-sum, ויצרנו DataFrame חדש בשם total_budget.

המרנו את עמודת department_id בשתי הטבלאות (department ו-total_budget) לטיפוס int כדי להבטיח התאמה במיזוג.

פיצלנו עמודת קלט בטבלת המחלקות על סמך התו המפריד -, ויצרנו שתי עמודות נפרדות עם כותרות מתאימות.
מחקנו את העמודה המקורית לשמירה על מבנה נתונים נקי.

חיברנו את הטבלאות department ו-total_budget באמצעות הפונקציה pd.merge על בסיס עמודת department_id.

המיזוג בוצע בשיטת inner join,יצר DataFrame סופי בשם updated_department_budget.

העלינו את הנתונים המעובדים למסד הנתונים PostgreSQL.
הנתונים נטענו לטבלה בשם department_budget בסכמת stg באמצעות הפונקציה to_sql.

3. ניתוח נתונים

2.3 - עיבוד נתונים - בפייתון - API CURRENCIES

שלפנו תאריכים ייחודיים מתוך טבלת stg.invoice במסד הנתונים PostgreSQL באמצעות שאילתת SQL.
התוצאה נטענה ל-DataFrame בשם dates_df, ולאחר מכן בדקנו את סוגי העמודות והפורמט של עמודת התאריכים.
עמודת התאריכים הומרה לפורמט datetime לשם נוחות ודיוק בעיבוד.
הודפסו מספר התאריכים הייחודיים ובוצעה בדיקה לוודא שהנתונים תקינים.

ייבאנו את ספריית requests לעבודה עם API, והגדרנו את פרטי ה-API, כולל מפתח הגישה, כתובת הבסיס, והפרמטרים הנדרשים לבקשות.
בנוסף, יצרנו רשימה ריקה בשם exchange_rates לשמירת נתוני שערי החליפין, וסט של תאריכים שכבר טופלו כדי למנוע עיבוד כפול.

הרצנו לולאת for שעוברת על כל תאריך ב-dates_df
עבור כל תאריך, נבדק אם הוא כבר טופל: אם לא, בוצעה קריאה ל-API לשליפת שער ההמרה בין דולר לשקל עבור אותו תאריך.
אם הבקשה הצליחה, שער ההמרה נשלף ונשמר ברשימה exchange_rates, לצד פרטי התאריך והמטבע.
במקרים שבהם לא הוחזרו נתונים או שהבקשה נכשלה, הודפסה הודעה מתאימה.
לאחר השלמת הלולאה, הודפס מספר שערי החליפין שנשלפו, וכן הוצגו מספר דוגמאות מהנתונים.
בוצעה בדיקה לזהות שערי חליפין לא תקינים (שליליים או אפסיים), והודפסה אזהרה במידת הצורך.

לאחר מכן, המרה של רשימת הנתונים ל-DataFrame בשם rates_df איפשרה המשך עיבוד וטעינה.בסיום, הנתונים המעובדים נטענו למסד הנתונים PostgreSQL לטבלה בשם exchange_rates בסכמת stg.

הטעינה בוצעה עם הגדרה לשמור על הנתונים הקיימים ולהוסיף את החדשים באמצעות הפונקציה to_sql.

3. ניתוח נתונים

3.3 - יצירת DWH בSQL

1.סכמת DWH:
יצירת סכמת DWH (רק במידה ולא קיימת).

2.DIM_CURRENCY:
השאילתה יוצרת טבלה חדשה בשם dim_currency בסכמה החדשה שיצרנו.
לוקחת את כל המידע שיצרנו בטבלת exchange_rates שמציגה את שער ההמרה מדולר לשקל לפי התאריכים הרלוונטים (לפי מה שקיימים בdata).

השתמשנו בפעולת create table שיוצרת טבלה חדשה בסכמה שאנחנו רושמים לפי ה"שאילתא" שבנינו. הוספנו את הסטרינג if not exist כדי למנוע "כפילויות".

הפונקציה create table שימשה אותנו לבניית כל טבלה רצויה במשימה הזו והכנסתה לסכמה הרצויה

3.DIM_PLAYLIST:
השאילתה יוצרת טבלה חדשה (crate table) בשם dim_playlist, שבה שמורים רק הפלייליסטים שיש להם שירים.
השתמשנו ב-inner join כדי לחבר בין טבלת playlisttrack לטבלת playlist על סמך עמודת המפתח המשותפת playlistid.
החיבור נעשה כך שכל שורה מתוך טבלת playlisttrack תתאים לפלייליסט שמופיע בטבלת playlist, ונקבל את הנתונים המתאימים משתי הטבלאות עבור כל שיר(track).

inner join מחזיר רק את השורות שיש להן התאמה בשתי הטבלאות. כלומר, אם יש פלייליסט שאין לו שירים (אין התאמה ב-playlisttrack), הוא לא ייכלל בתוצאה

4.DIM_CUSTOMER:
את בניית עמודות השמות (פרטי ומשפחה) עשינו על ידי שרשור של שתי מחרוזות שאנחנו יוצרים - הראשונה האות הראשונה בכל שם כאות גדולה לפי שליפת תת מחרוזת באמצעות substring עם חיבור של כל המשך השם כאותיות קטנות ושליפת האותיות הרלוונטיות בעזרת פונקציות אורך המחרוזת ודילוג על האות הראשונה.
אותה פעולה נבצע לעמודת שם משפחה.
שימוש בפונקציית lower ו upper בשביל לשלוף את הסטרינג בהתאם לדרישה.
תו ראשון יהיה באות גדולה (upper) והמשך הstring יהיה באותיות קטנות (lower).

את בניית עמודת הדומיין עשינו על ידי שליפת מחרוזת מצד ימין (right), באורך המחושב לפי אורך המחרוזת הכוללת (length) פחות המיקום של ה-@ שחושב במחרוזת (position), כדי לקבל את התווים שמופיעים אחרי ה-@(לא כולל המיקום שלו).
נתנו לעמודה את השם domain (בעזרת השימוש - as).
את כל מה שבנינו הכנסו לתוך- CTE, עם עמודת המפתח customerid כדי שיה לנו את המפתח הייחודי עבור כל לקוח ויצירת קשר בין הטבלה של customer בשימוש בinner join ב CTE.

3.3 - יצירת DWH בSQL

4.DIM_CUSTOMER - המשך:

יצרנו שאילתא ששולפת את כל העמודות מ-customer מלבד השמות הישנים. מה-CTE החדש שלפנו את השמות המעודכנים ועדכנו את שם העמודה לשם המקורי מטבלת customer.
הוספנו את הדומיין שחתכנו מתוך המייל בתור עמודת domain.
סידרנו את העמודות בסדר נוח והגיוני:
עמודת domain באה לאחר המייל.
שם הפרטי והמשפחה באו במקום ה"מקורים" בטבלת customer
ואת כל שאר העמודות מcustomer.

בעזרת הפעולות האלה יצרנו טבלה חדשה בשם dim_customer בתוך בסיס הנתונים dwh עם השימוש בcreate table - רק במידה ולא קיים.

5. DIM_EMPLOYEE:

התחלנו בבניית שאילתא שתביא את כל העמודות מהטבלה employee.
עשינו את הinner join על סמך המפתח הייחודי dpartment_id שבטבלת department_budget והחיבור שלו לעמודה זהה בטבלת employee שרק שם העמודה הוא בלי הקו התחתון והוא משמש כמפתח משני (foreign key) בטבלת employee.
כתבנו את החיבור של הjoin על שמות העמודות שמתאימות בכל טבלה הוספנו את העמודות הרצויות ל select וכך נוסיף אותם לטבלה שניצור.
לאחר מכן ניגשנו לבניית העמודה הנדרשת - חישוב זמן ההעסקה של כל עובד וייצוג בשנים.
חישבנו את זמן העסקה עבור כל עובד על ידי העיקרון שבו כל עובד שנמצא בטבלה עדיין (היום) עובד בחברה.
בעזרת פעולת age שמחשבת את הפרש בין שני תאריכים עשינו חישוב של התאריך הנוכחי (בעזרת current_date ששולף עבורנו את התאריך הנוכחי) לבין התאריך שבו העובד התחיל את העסקה שלו בחברה (לפי עמודת hiredate מטבלת employee).
חילצנו מהתוצאה של ההפרש רק את השנה בעזרת extract year from (השנה מהתאריך שיצרנו) ואליו בעזרת concat (||) חיברנו את ה years-string עם הפרדה של רווח כדי שיהיה ברור למי שקורה מה המספר מייצג.
כך בנינו את העמודה והוספנו אותה לטבלה שלנו תחת השם hire_length.
המשכנו לבנות את העמודה שתחתוך את הדומיין מכתובת המייל, כמו שעשינו בעמודת הלקוחות:
שליפת מחרוזת מצד ימין (right) באורך המחושב לפי אורך המחרוזת הכוללת (length) פחות המיקום של ה-@ (position) במחרוזת, כדי להבטיח שהחיתוך יכלול רק את התווים שמופיעים אחרי ה-@ (לא כולל המיקום שלו). נתנו לעמודה את השם domain (בעזרת השימוש - as).
לאחר מכן, לבניית העמודה עבור זיהוי האם העובד הוא מנהל השתמשנו בפעולת case, שמאפשרת בדיקה שלפיה אם עמודת employeeid של העובד מופיעה בעמודת reportsto של עובדים אחרים, אז העובד נחשב כמנהל.
את הבדיקה ביצענו על ידי שאילתת משנה שבודקת את כל הערכים הייחודיים בעמודת reportsto בטבלת stg.employee, תוך שימוש בסינון where reportsto is not null, שמוודא שלא נכלול ערכים ריקים.
אם נמצא שהערך של employeeid מופיע בעמודת reportsto, העמודה החדשה תקבל את הערך 1.
אם לא נמצא, העמודה תקבל את הערך 0.
כך בנינו עמודה בשם is_manager, שמציינת עבור כל עובד האם הוא מוגדר כמנהל (1) או לא (0).
בעזרת השאילתא המלאה שבנינו יצרנו בעזרת create table טבלה חדשה סכמה הרצויה.

3. ניתוח נתונים

3.3 - יצירת DWH בSQL

6. DIM_TRACK:

ביצענו בדיקה על כל העמודות באמצעות information_schema לפני יצירת הטבלה, כדי לוודא שכל המידע הנדרש נכלל ושאף שדה לא הושמט בטעות.
יצרנו את טבלת dim_track שמרכזת את כל המידע על שירים.
התחלנו בשליפת כל העמודות מטבלת track, כפי שנדרש במשימה.
חיברנו את הטבלה לטבלאות album, artist, genre ו-mediatype באמצעות פעולת join, תוך שימוש במפתחות משותפים.
כל חיבור נבדק כדי למנוע כפילויות מיותרות של עמודות או שורות.
חישוב משך השיר בשניות ובפורמט MM:SS : השתמשנו ב-round לעיגול משך השיר לשניות.
פורמט MM:SS נבנה באמצעות floor לחישוב הדקות המלאות, lpad להצגת השניות בשתי ספרות, ו-concat לחיבורם יחד.
השדות היחידים שלא נכללו הם שדות כפולים שנוצרו מחיבורים לטבלאות אחרות או מידע מיותר שגרם לשכפול שורות.
לדוגמה: שדות כפולים שמקורם בחיבורים לטבלאות album, artist, genre ו-mediatype.
שורות כפולות שנגרמו מחיבורים לא מדויקים נמנעו על ידי שימוש ב-distinct או חיבור מדויק יותר.
בסיום התהליך יצרנו את הטבלה בעזרת create table if not exists, כדי למנוע יצירת כפילויות.
כל השדות הובאו מתוך מטרה לשמר את המידע המקורי בהתאם לדרישות.
השדות שהיו "לא רלוונטיים", כמו bytes, composer, ו-last_update, נכללו בטבלה לפי הדרישה להביא את "כל השדות". שדות שלא נכללו בטבלה הסופית הן רק שדות כפולים ושורות מיותרות שנגרמו מחיבורים.

7. FACT_INVOICE:

בבניית טבלת Fact_invoice, הרעיון המרכזי הוא לכלול נתונים מדידים ומזהים שמייצגים את הפעולות העסקיות המרכזיות.
הטבלה מתמקדת בסכומים, תאריכים, ומפתחות זרים שמחברים אותה לטבלאות מימד כמו Dim_customer.
נתונים שאינם רלוונטיים לניתוח, כמו פרטי כתובת, לא נכללו בטבלה כדי לשמור על מבנה פשוט ויעיל, בהתאם למודל Star Schema.
הטבלה כוללת עמודות שנבחרו בקפידה כדי לייצג את המידע החשוב לניתוחים עסקיים.
מזהה החשבונית (invoice_id) מאפשר זיהוי ברור של כל פעולה עסקית, ומזהה הלקוח (customer_id) מחבר את החשבונית לנתוני הלקוח שבטבלת Dim_customer.
תאריך יצירת החשבונית (invoice_date) עוזר לנתח מגמות מכירה לאורך זמן, וסכום החשבונית הכולל (total) הוא נתון מדיד חשוב לניתוחים פיננסיים.
המבנה של הטבלה ממוקד, שומר רק על הנתונים הנחוצים, ומאפשר לנתח את המידע בצורה קלה וברורה.
אם בעתיד יתברר שיש צורך להוסיף שדות או להסיר נתונים שאינם בשימוש, מבנה הטבלה יכול להשתנות בהתאם לצורך.

3. ניתוח נתונים

3.3 - יצירת DWH בSQL

8.FACT_INVOICELINE:

בבניית טבלת Fact_invoiceline שמרנו על העמודות החשובות ביותר לניתוחים עסקיים, תוך הקפדה על פשטות ויעילות.
הקריטריונים לבחירת העמודות התבססו על נתונים מדידים ומזהים, שתומכים בניתוחי מכירות ומשתלבים היטב עם טבלאות אחרות במודל הנתונים.
עמודות שאינן משמשות ישירות לניתוחים, כמו חישובים שניתן לבצע בשאילתות או שדות טכניים, הוסרו כדי לשמור על מבנה ברור ונגיש.
הטבלה כוללת את מזהה שורת החשבונית invoicelineid, שמאפשר מעקב פרטני אחרי כל שורה, ואת מזהה החשבונית invoiceid, שמחבר את הטבלה לטבלת Fact_invoice ומאפשר ניתוח נתונים ברמת החשבונית.
מזהה השיר trackid מקשר את שורת החשבונית לנתוני השיר בטבלת Dim_track.
בנוסף, נשמרו עמודות מחיר היחידה unitprice וכמות היחידות שנרכשו quantity, המהוות נתונים פיננסיים מרכזיים לניתוחי מכירות והכנסות.
עמודת last_update נשמרה כדי לאפשר מעקב אחרי שינויים או עדכונים במידע.
עיצוב הטבלה מייצג את עקרונות מודל Star Schema, בו טבלאות FACT כוללות נתונים מדידים ומזהים בלבד, לצורך ניתוחים מדויקים וממוקדים.
עמודות כמו line_total, המבוססות על חישוב שאפשר לבצע בשאילתות, הוסרו, כמו גם עמודות טכניות שאינן נחוצות לניתוחים עסקיים.
המבנה הזה מבטיח גמישות להוספת עמודות נוספות במידה ויידרשו בעתיד, תוך שמירה על ניתוחים יעילים ופשוטים.

3. ניתוח נתונים

4.3 - ניתוח מידע בSQL

1. המטרה הייתה לזהות את הפלייליסטים עם מספר השירים המקסימלי והמינימלי ולחשב את ממוצע השירים בכל הפלייליסטים.
יצרנו טבלה זמנית cte_track_count שחישבה את מספר השירים הייחודיים לכל פלייליסט באמצעות group by.
לאחר מכן, יצרנו שתי טבלאות זמניות נוספות: max_playlist, שמציגה את הפלייליסט עם מספר השירים המקסימלי, ו-min_playlist, שמציגה את הפלייליסט עם מספר השירים המינימלי.
חיברנו את שתי הטבלאות באמצעות union all עם עמודת category שמסמנת את סוג הפלייליסט (מקסימום או מינימום).
חישבנו את ממוצע השירים בטבלה הזמנית cte_track_count והוספנו אותו לתוצאה הסופית.
התוצאה כוללת את מזהי הפלייליסטים עם מספר השירים הקיצוניים ואת ממוצע השירים בכל הפלייליסטים.

2. החלוקה של השירים לפי קבוצות מכירה בוצעה על ידי יצירת שאילתה שבחנה את מספר המכירות הכולל לכל שיר, בהתבסס על הנתונים הקיימים בטבלאות fact_invoiceline ו-fact_invoice.
החיבור בין הטבלאות נעשה באמצעות join על עמודת המפתח המשותפת invoiceid, כאשר טבלת fact_invoiceline מכילה את מזהי השירים (trackid) ואת פרטי המכירה, ואילו טבלת fact_invoice מספקת מידע נוסף על החשבוניות.
לצורך חישוב כמות המכירות הכוללת לכל trackid השתמשנו בפונקציית count כדי לספור את מספר השורות הקשורות לכל שיר.
הנתונים קובצו לפי trackid באמצעות group by כדי להבטיח שכל מזהה שיר יופיע פעם אחת עם מספר המכירות הכולל שלו.
התוצאה הבסיסית כללה את מזהי השירים (trackid) ואת סך המכירות שלהם (total_sales).
לאחר מכן, עטפנו את השאילתה בתת-שאילתה נוספת כדי ליצור עמודה חדשה בשם sales_group, המחלקת את השירים לקבוצות בהתאם למספר המכירות שלהם. החלוקה לקבוצות בוצעה בעזרת פונקציית case, כך ששירים ללא מכירות שובצו לקבוצה 0, שירים עם מכירות בין 1 ל-5 שובצו לקבוצה 1-5, שירים עם מכירות בין 6 ל-10 שובצו לקבוצה 5-10, ושירים עם מכירות מעל 10 הוקצו לקבוצה 10<.
התוצאה המתקבלת כללה שלוש עמודות: trackid, total_sales ו-sales_group.

לבסוף, מיינו את השירים לפי total_sales בסדר יורד, כך שהשירים עם מספר המכירות הגבוה ביותר מופיעים בתחילת הרשימה.
השאילתה אינה כוללת עמודת שם שיר, מכיוון שטבלה זו אינה מכילה מידע כזה.
אם יידרש בעתיד מידע נוסף כמו שמות שירים או ז'אנרים, ניתן לשלב טבלאות אחרות באמצעות join נוסף על trackid. התוצאה הסופית היא טבלה מסודרת, המאפשרת ניתוח ברור של מכירות השירים, תוך חלוקה אינטואיטיבית לקבוצות על בסיס כמות המכירות.

3. ניתוח נתונים

4.3 - ניתוח מידע בSQL

3.א התחלנו בחישוב סך המכירות לכל מדינה באמצעות טבלה זמנית (CTE) בשם country_sales, שמרכזת את סך המכירות לכל מדינה מתוך טבלת fact_invoice, בהתבסס על התאמה ללקוחות מתוך טבלת dim_customer.
החישוב בוצע בעזרת פונקציית sum לעמודת total וקיבוץ הנתונים לפי עמודת country.
יצרנו שתי טבלאות זמניות נוספות: top_5_countries מציגה את 5 המדינות עם סך המכירות הגבוה ביותר, באמצעות מיון לפי total_sales בסדר יורד ושימוש בהגבלה ל-5 שורות הגבוהות ביותר.
bottom_5_countries: מציגה את 5 המדינות עם סך המכירות הנמוך ביותר, באמצעות מיון לפי total_sales בסדר עולה ושימוש בהגבלת 5 שורות נמוכות ביותר.
לבסוף, שילבנו את שתי התוצאות לשאילתה אחת באמצעות union all, תוך הוספת עמודת קטגוריה (category), שמבדילה בין Top 5 ל-Bottom 5.
התוצאה הסופית מויינה לפי הקטגוריה, ובתוכה לפי סך המכירות בסדר יורד.

3.ב בנינו טבלה זמנית (genre_sales) שמרכזת את סך המכירות של כל ז'אנר בכל מדינה, על ידי חיבור בין מספר טבלאות:חיברנו את טבלת fact_invoiceline, שמכילה נתונים על כמות המוצרים שנמכרו והמחיר שלהם, עם טבלת dim_track, שמספקת את שם הז'אנר של כל שיר.
חיברנו את התוצאה לטבלת dim_customer, שמספקת את שם המדינה של הלקוח.
החישוב בוצע באמצעות פונקציית sum על המכפלה של quantity ו-unitprice, עם קיבוץ לפי country ו-genre_name.
בנוסף, יצרנו טבלה זמנית נוספת בשם total_country_sales, שמרכזת את סך המכירות הכולל של כל מדינה מתוך עמודת total שבטבלת fact_invoice.טבלה זו שימשה אותנו לחישוב האחוזים.

לבסוף, השתמשנו בנתוני שתי הטבלאות לחישוב:אחוזי המכירות של כל ז'אנר מתוך סך המכירות הכולל של אותה מדינה.
דירוג מקומי לכל ז'אנר בתוך כל מדינה, באמצעות פונקציית rank().התוצאה הסופית כוללת את שם המדינה, שם הז'אנר, סך המכירות של הז'אנר (עם סימון $), האחוזים (עם סימון %), ודירוג הז'אנר בתוך כל מדינה.

3. ניתוח נתונים

4.3 - ניתוח מידע בSQL

4. התחלנו בחישוב מספר הלקוחות בכל מדינה מתוך טבלת dwh.dim_customer.
יצרנו טבלה זמנית שמרכזת את מספר הלקוחות בכל מדינה (customer_count) באמצעות הפונקציה COUNT תוך קיבוץ הנתונים לפי עמודת country.
כדי להתמודד עם מדינות בעלות לקוח יחיד בלבד, הוספנו עמודה חדשה (country_label) שבה תויגו מדינות אלו כ-"Other" באמצעות CASE.
תוצאה זו מהווה בסיס לניתוח נתוני ההזמנות והמכירות.
בשלב הבא, יצרנו טבלה זמנית שמרכזת מידע על כל לקוח מתוך טבלת dwh.fact_invoice.
חישבנו את מספר ההזמנות (order_count), ממוצע הסכום להוצאה (avg_total_per_customer), וסך ההוצאות לכל לקוח.
החישוב בוצע באמצעות הפונקציות COUNT, AVG, ו-SUM, תוך קיבוץ הנתונים לפי עמודת customerid. שלב זה מספק תובנות על פעילות הלקוחות ותורם לבניית תמונה כללית על פעילותם.
לאחר מכן, שילבנו את התוצאות לניתוח כלל-מדינתי. יצרנו טבלה זמנית שמשלבת את מספר הלקוחות בכל מדינה עם נתוני ההזמנות והמכירות של הלקוחות.
החיבור בין הטבלאות בוצע באמצעות LEFT JOIN בין טבלת customer_count_per_country לטבלת הלקוחות (dwh.dim_customer) ולנתוני ההזמנות (customer_aggregates).
החישובים כללו מספר לקוחות בכל מדינה (customer_count), ממוצע ההזמנות לכל לקוח (avg_orders_per_customer), וממוצע הסכומים לכל לקוח (avg_total_per_customer).
נעשה שימוש ב-COALESCE לטיפול בערכים ריקים, מה שמבטיח פלט מסודר ומלא.

לבסוף, שלפנו את התוצאה הסופית הכוללת את שם המדינה (או "Other" עבור מדינות עם לקוח יחיד), מספר הלקוחות בכל מדינה, ממוצע ההזמנות לכל לקוח, וממוצע הסכומים לכל לקוח.
התוצאה ממוינת לפי שם המדינה ומאפשרת לזהות מדינות בעלות ביצועים עסקיים גבוהים בהשוואה לאחרות.

5. באנליזה זו יצרנו מערכת המלצות מותאמת אישית ללקוחות, המבוססת על העדפותיהם המוזיקליות והשירים שהם רכשו בעבר.
מטרת האנליזה הייתה לזהות שירים שהלקוח טרם רכש ולהציע לו רשימת המלצות חדשות המבוססות על שני הז'אנרים המועדפים עליו.
התחלנו בחישוב כמות השירים שנרכשו בכל ז'אנר עבור כל לקוח.
את הנתונים ריכזנו באמצעות CTE (Common Table Expression), מה שאפשר לנו לפשט את הניתוח ולהתמקד בז'אנרים המובילים בלבד.
לאחר מכן, זיהינו את שני הז'אנרים המועדפים על כל לקוח על פי כמות השירים שנרכשו.
ניתחנו את השירים הפופולריים ביותר בתוך כל אחד מהז'אנרים המובילים, תוך שימוש בדירוג מבוסס RANK. עבור כל ז'אנר, בחרנו את שלושת השירים המובילים. כדי להבטיח שכל שיר ברשימת ההמלצות יהיה חדש ורלוונטי ללקוח, סיננו את השירים שהלקוח כבר רכש, תוך שימוש בפקודת NOT IN ובנתוני הרכישות הקיימים.

לבסוף, יצרנו רשימה סופית של שישה שירים מומלצים לכל לקוח: שלושה מכל אחד משני הז'אנרים המועדפים עליו. כל שיר ברשימה כולל את שם הז'אנר, מזהה השיר ושם השיר, כך שההמלצות נגישות ומותאמות אישית.
התהליך מדגים שימוש אפקטיבי בטכניקות ניתוח מתקדמות, כגון דירוגים וסינון, ליצירת מערכת המלצות יעילה המבוססת על נתוני העבר של הלקוחות.

3. ניתוח נתונים

4.3 - ניתוח מידע בSQL

6. הצג את 3 הז'אנרים הפופולריים ביותר שצפויים לחוות עלייה במכירות בכל עונה, בהתבסס על ניתוח נתונים קיימים ומגמות נוכחיות.
הצג את סך הרווחים המצטבר לכל ז'אנר, את כמות הלקוחות שרכשו שירים בכל ז'אנר, והצג חיזוי למגמות מכירות לפי מדינות, תוך חלוקה לפי עונות השנה.
המטרה באנליזה זו הייתה לזהות את שלושת הז׳אנרים הפופולריים ביותר שצפויים לחוות עלייה במכירות בכל עונה, בהתבסס על נתוני מכירות היסטוריים ומגמות נוכחיות.
האנליזה שואפת לספק תובנות מעמיקות על ביצועי הז׳אנרים השונים, להציג את סך הרווחים המצטבר ואת כמות הלקוחות הייחודיים לכל ז׳אנר, ולבחון את מגמות הצמיחה העונתיות תוך התמקדות בחלוקה לפי מדינות.
תובנות אלו יכולות לשמש ככלי אסטרטגי לקבלת החלטות שיווקיות ומיקוד מאמצים בז׳אנרים בעלי פוטנציאל הצמיחה הגבוה ביותר.
בתחילת התהליך, חילקנו את נתוני המכירות לעונות השנה על בסיס תאריכי רכישה.
באמצעות פונקציות SQL, הומרו חודשי השנה לעונות: חורף, אביב, קיץ וסתיו.
לכל רכישה זוהה הז׳אנר של השיר הנרכש, המחיר ליחידה, כמות היחידות שנמכרו, ומזהה הלקוח.
שילוב נתונים אלו יצר בסיס נתונים מרכזי המאפשר ניתוח רווחים ולקוחות עבור כל ז׳אנר ועונה.

בהמשך, חישבנו את סך הרווחים הכולל (total_revenue) ואת מספר הלקוחות הייחודיים (unique_customers) לכל ז׳אנר בכל עונה.
החישובים בוצעו באמצעות פונקציות SUM ו-COUNT(DISTINCT) לצורך הפקת נתונים מדויקים ותובנות ברמת המיקרו.
לאחר מכן, יצרנו טבלה מדרגת את הז׳אנרים בכל עונה על פי סך הרווחים, תוך שימוש בפונקציית RANK() לחישוב הדירוג של כל ז׳אנר בעונה הרלוונטית.

לבסוף, סיננו את הנתונים כדי להציג את שלושת הז׳אנרים המובילים בכל עונה. הנתונים שהתקבלו הוצגו בטבלה מסכמת הכוללת את שם הז׳אנר, העונה, סך הרווחים, ומספר הלקוחות הייחודיים.
התוצאות הציגו שלושת הז׳אנרים המובילים בכל עונה, תוך פירוט של הביצועים הכלכליים ומספר הלקוחות הייחודיים לכל ז׳אנר.
מגמות עונתיות ברורות שעלו מהנתונים הצביעו על ז׳אנרים שמתחזקים בכל עונה, וסיפקו תובנות על הזדמנויות שיווקיות שיכולות למקד את מאמצי השיווק בהתאם לעונה ולז׳אנר.

3. ניתוח נתונים

5.3 - ניתוח מידע בפייתון

ייבאנו את הספריות Pandas, SQLAlchemy ו-Matplotlib לצורך ניתוח נתונים ויצירת גרפים.
יצרנו חיבור למסד הנתונים PostgreSQL עם פרטי גישה, ושלפנו את כל הטבלאות הנדרשות מתוך הסכמות שהוגדרו מראש.
כל טבלה נטענה ל-DataFrame נפרד, כך שכל הנתונים זמינים לעיבוד בפייתון.
בשלב זה, נתוני הטבלאות המרכזיות כמו לקוחות, חשבוניות ושירים מוכנים לשימוש באנליזות.
הפונקציה create_bar_chart פותחה ככלי מרכזי ליצירת גרפי עמודות דינמיים וברורים, המותאמים לצרכים מגוונים כמו השוואת משתנים, חלוקה לקבוצות, ושימוש בעיצובים מותאמים אישית.
הרעיון המרכזי הוא לשלב פשטות וגמישות, כך שכל משתמש יוכל ליצור גרפים בקלות ובהתאם לצרכים הייחודיים שלו.
הפונקציה מתחילה בהכנת ציר גרפי (Axis), ואם לא סופק ציר מותאם, היא מגדירה אחד כברירת מחדל. ציר ה-X מסודר כך שכל ערך בעמודה הנבחרת יופיע בצורה מסודרת וברורה.
לגרפים מורכבים יותר, הפונקציה מאפשרת חלוקה לקבוצות, כמו לפי שנים או קטגוריות.
כל קבוצה מוצגת בעמודות נפרדות עם אפשרות להתאים צבעים שונים, מה שמוסיף מימד ויזואלי ברור וקריא.
אם נדרשת השוואה בין משתנים, כמו נתוני מכירות במטבעות שונים, הפונקציה תציג עמודות מקבילות בצבעים שונים עם תוויות שמסבירות את הערכים.

בסיום, מתבצע עיצוב סופי של הגרף: הוספת כותרת, תוויות לצירים, וסיבוב תוויות ציר ה-X לשיפור הקריאות.
אם יש קבוצות או משתנים נוספים, נוסף מקרא אוטומטי שמסביר את הנתונים.
הפונקציה מספקת כלי גמיש ופשוט להצגת נתונים בצורה ברורה, אינטואיטיבית, ומותאמת אישית לכל תרחיש.

3. ניתוח נתונים

5.3 - ניתוח מידע בפייתון

1. באנליזה הזו חישבנו שלושה פרמטרים עיקריים: כמות האלבומים לכל אמן, כמות השירים לכל אמן, וכמות השירים לכל ז'אנר.
כל ניתוח בוצע בצורה ממוקדת ותוצאותיו הוצגו באמצעות הפונקציה create_bar_chart, שנבנתה מראש כדי לספק תצוגה גמישה וברורה.
תחילה חישבנו את כמות האלבומים לכל אמן.
באמצעות groupby פילחנו את הנתונים לפי מזהה ושם האמן, וספרנו את כמות האלבומים הייחודיים עם nunique.
לאחר מיון בסדר יורד ובחירת חמשת האמנים המובילים, השתמשנו בפונקציה ליצירת גרף עמודות בצבע כחול.
הפונקציה התאימה בקלות את הכותרות, הצבעים והמבנה, ואיפשרה להציג את המידע בצורה ברורה. בשלב הבא ניתחנו את כמות השירים לכל אמן.
תהליך זה היה דומה מאוד, אך הפעם חישבנו את כמות השירים באמצעות count.
התוצאות הוצגו שוב בגרף עמודות, הפעם בצבע ירוק, תוך שימוש חוזר בפונקציה שלנו.
ההתאמה האישית של מאפייני הגרף, כמו השמות והצבעים, בוצעה בקלות בזכות הגמישות של הפונקציה. לבסוף חישבנו את כמות השירים לכל ז'אנר.
בעזרת אותו מבנה לוגי קיבצנו את הנתונים לפי מזהה ושם הז'אנר, וספרנו את השירים.
המידע הוצג בגרף כתום, כששוב הפונקציה שלנו ייצרה גרף אחיד ומותאם אישית במהירות ובפשטות.
לסיום, שילבנו את שלושת הגרפים ב-Subplots אחד.
שימוש בלולאה אפשר לנו להעביר את הנתונים ואת מאפייני כל גרף בצורה דינמית לפונקציה, שהפיקה שלושה גרפים נפרדים ומסודרים היטב.
תהליך זה הדגים את הגמישות הגבוהה של הפונקציה ואת היכולת שלה להתאים את עצמה למבנים מורכבים כמו Subplots, מבלי לשנות את הקוד.
באנליזה הזו הפונקציה הוכיחה את יעילותה ואיפשרה להציג את התוצאות בצורה קריאה, אחידה ומדויקת. השימוש בה חסך זמן ומנע טעויות, תוך שמירה על רמת ביצוע מקצועית.

3. ניתוח נתונים

5.3 - ניתוח מידע בפייתון

2. באנליזה זו זיהינו את חמשת הלקוחות עם סכומי הרכישות הגבוהים ביותר, תוך השוואה בין סכומי הרכישות בדולרים (USD) לשקלים (ILS).
התהליך כלל חיבור טבלאות, חישוב מדדים כספיים ותצוגה גרפית מותאמת אישית.
תחילה, המרת עמודת התאריכים בטבלת currency לפורמט datetime אפשרה חיבור חלק בין הטבלה לשאר הנתונים.
חיברנו את טבלת invoice, המכילה פרטי חשבוניות, עם טבלת customer, שמספקת פרטים על הלקוחות, ובנוסף חיברנו את נתוני שערי ההמרה מ-currency על בסיס תאריכי החשבוניות.
יצרנו עמודה חדשה בשם customer_name, שאיחדה את השם הפרטי ושם המשפחה של כל לקוח.
חישבנו את סכומי הרכישות בשקלים בעזרת עמודת total_ils, שממירה את הסכומים משקלים לדולרים על פי שער ההמרה, כדי לאפשר השוואה בין המטבעות.
בשלב הניתוח, קבוצנו את הנתונים לפי מזהה ושם הלקוח וסכמנו את סך הרכישות בדולרים ושקלים.
מיינו את הלקוחות לפי סכום הרכישות בדולרים ובחרנו את חמשת הלקוחות המובילים.
הנתונים נשמרו במבנה ברור, המאפשר להציג אותם בצורה נגישה.
התוצאה הוצגה בגרף עמודות שהציג את סכומי הרכישות בכל מטבע עבור כל לקוח.
השימוש בפונקציה create_bar_chart הפך את תהליך יצירת הגרף לנוח וברור, תוך התאמה אישית של צבעים ותוויות. כך הודגשו ההבדלים בין סכומי הרכישות במטבעות השונים, בצורה שמאפשרת הבנה אינטואיטיבית של הממצאים.

3. באנליזה זו חישבנו את סכומי המכירות החודשיים לפי שנים, תוך שימוש בנתוני החשבוניות מטבלת invoice.
תחילה, הוספנו עמודות חדשות לנתונים, אחת עבור השנה (year) ואחת עבור החודש (month), שהופקו מתוך עמודת התאריך invoicedate בעזרת המרה לפורמט datetime ושימוש במתודות dt.year. ו-dt.month..
לאחר מכן, קבוצנו את הנתונים לפי שנה וחודש, וחישבנו את סכומי המכירות הכוללים לכל שילוב של שנה וחודש באמצעות הפונקציה .sum(). תוצאות הקיבוץ נשמרו כ-DataFrame חדש בשם monthly_sales, עם מבנה ברור ושם עמודה מותאם לתיאור נתוני המכירות.
לצורך הצגת הממצאים בצורה ויזואלית, השתמשנו בפונקציה create_bar_chart, שפותחה מוקדם יותר, ליצירת גרף עמודות המציג את סכומי המכירות החודשיים. הפונקציה חולקת את הנתונים לקבוצות לפי השנים, כאשר לכל שנה הוקצה צבע ייחודי מתוך רשימה מוגדרת מראש. על ציר ה-X מוצגים החודשים, ועל ציר ה-Y סכומי המכירות החודשיים. הוספנו כותרות, תוויות לצירים והתאמות עיצוב כמו סיבוב התוויות בציר ה-X, כדי להבטיח קריאות ונוחות שימוש. הגרף מספק תובנה אינטואיטיבית וברורה על הביצועים החודשיים לאורך השנים.

3. ניתוח נתונים

5.3 - ניתוח מידע בפייתון

4. באנליזה זו נבדקה הקורלציה בין אורך השיר למכירות, תוך שימוש בנתוני השירים והחשבוניות.
תחילה, חיברנו את הטבלאות track ו-invoiceline בעזרת איחוד פנימי (merge), שבו חיברנו בין מזהה השיר (track_id) לבין נתוני המכירות.
לאחר מכן, חישבנו את סכום המכירות לכל שיר באמצעות הכפלת מחיר היחידה (unitprice) בכמות הנמכרת (quantity), והוספנו את הנתון כעמודה חדשה בשם sales.

בשלב הבא, חישבנו את הקורלציה בין אורך השיר בשניות (track_duration_seconds) לבין סכום המכירות בעזרת הפונקציה .corr(), ותוצאות החישוב הודפסו לצורך ניתוח.להמחשה ויזואלית, השתמשנו בספריית seaborn ליצירת גרף פיזור, שבו כל נקודה מייצגת שיר עם ציר X שמציג את אורך השיר בציר Y את סכום המכירות.
מעל הנקודות הוספנו קו רגרסיה בצבע אדום, הממחיש את מגמת הקשר בין המשתנים.
הגרף עוצב בצורה קריאה וברורה, כולל תוויות לצירים, כותרת, ורשת רקע להדגשה.
הניתוח הגרפי יחד עם ערך הקורלציה מספקים הבנה מעמיקה על טיב הקשר בין אורך השיר למכירות.

5. באנליזה זו פיתחנו מערכת המלצות מותאמת אישית ללקוחות המבוססת על הז'אנרים והשירים שהם רכשו בעבר.
תהליך ההמלצות כלל מספר שלבים לוגיים, שמטרתם לזהות העדפות לקוחות ולהציע להם שירים שטרם נרכשו.
התחלנו בלחבר את הטבלאות invoice ו-invoiceline כדי ליצור מאגר נתונים מלא של רכישות שירים, שכלל פרטים על השירים, המחירים, וכמויות הרכישה.
בשלב הבא, שילבנו את נתוני הלקוחות על ידי איחוד נוסף עם טבלת customer, כך שניתן יהיה לשייך כל רכישה ללקוח המתאים.לאחר מכן, עברנו על כל לקוח בטבלת הלקוחות. עבור כל לקוח, זיהינו את הז'אנרים המועדפים עליו על ידי קיבוץ נתוני הרכישות לפי ז'אנרים, ומציאת שני הז'אנרים המובילים על בסיס מספר השירים שנרכשו בכל ז'אנר.
מידע זה שימש בסיס להמלצות ממוקדות יותר.בהמשך, סיננו את השירים שלא נרכשו על ידי הלקוח על מנת להבטיח שכל שיר בהמלצות יהיה חדש ללקוח.
מתוך רשימה זו, בחרנו שלושה שירים פופולריים מכל אחד משני הז'אנרים המועדפים, בהתבסס על סדר השירים בטבלת הנתונים.לבסוף, יצרנו רשימת המלצות, שבה לכל לקוח יש עד שישה שירים מומלצים (שלושה מכל ז'אנר מועדף).
ההמלצות כוללות מידע על ז'אנר השיר, מזהה השיר ושם השיר, ומוצגות ב-DataFrame לסקירה נוחה.

3. ניתוח נתונים

5.3 - ניתוח מידע בפייתון

6. האנליזה ששאלנו :"הצג את 10 האמנים עם הפוטנציאל הגבוה ביותר לעלייה בפופולריות בחודשים הקרובים, בהתבסס על הנתונים ומגמות שקיימות בהם.
הצג את סכום הרווח המצטבר עבור כל אמן, ואת כמות האנשים הייחודיים שרכשו את השירים שלו.
פרט את הגישה לחישוב הפוטנציאל." :
האנליזה נועדה לזהות את עשרת האמנים בעלי הפוטנציאל הגבוה ביותר לעלייה בפופולריות בחודשים הקרובים.
השיטה מתבססת על ניתוח נתוני מכירות, מגמות קצרות וארוכות טווח, והתנהגות לקוחות.
המטרה היא לספק כלי מבוסס נתונים לקבלת החלטות שיווקיות ואסטרטגיות, תוך הדגשת אמנים בעלי פוטנציאל צמיחה יוצא דופן.
חיברנו טבלאות רלוונטיות – שירים, מכירות ולקוחות – תוך שימוש במפתחות זרים ליצירת מסגרת נתונים מרכזית לניתוח.
נתוני התאריכים הומרו לפורמט המאפשר חילוץ חודשים ושנים לצורך ניתוח מגמות.
חישבנו הכנסות חודשיות לכל אמן על ידי קיבוץ וסיכום נתוני מחירים וכמויות.
ניתוח המגמות התבצע באמצעות ממוצע נייד (שלושה חודשים) וממוצע מצטבר מתעדכן, שסימנו חודשים עם עליות משמעותיות.
סיכמנו את ההכנסות בחודשים אלו, וניתחנו את מספר הלקוחות הייחודיים וסך ההכנסות לכל אמן.
מדד "blow up potential", המשלב משקלות למגמות קצרות (70%) וארוכות טווח (30%), שימש לדירוג האמנים.
תוצאות האנליזה הוצגו בטבלת סיכום עם המחשה גרפית שמציגה את היחס בין ההכנסות למספר הלקוחות.
התהליך כולו ממוקד בזיהוי אמנים מבטיחים, והוא מספק בסיס מדויק להחלטות אסטרטגיות המבוססות על נתונים.
ניתן להרחיב גישה זו לשימושים נוספים כמו חיזוי מגמות עתידיות או הערכת השפעת קמפיינים שיווקיים.

4. סיכום

בפרויקט זה עסקנו בניתוח מכירות ופילוח שוק של חנות iTunes, תוך שימוש בכלים מתקדמים לניתוח נתונים כמו SQL ופייתון.
מטרת הפרויקט הייתה להבין את דפוסי המכירה העיקריים, ליצור מערכת המלצות מותאמת אישית, ולזהות מגמות עונתיות במכירות.

במהלך הפרויקט הצלחנו להגיע למספר תובנות משמעותיות, כולל זיהוי הפלייליסטים והשירים הפופולריים ביותר, ניתוח ההשפעות של עונות השנה על הביקוש למוזיקה, ופיתוח מערכת המלצות שמאפשרת חווית משתמש מותאמת אישית.
הניתוחים שביצענו הובילו אותנו להמלצות אסטרטגיות לשיפור הקמפיינים השיווקיים ולהגדלת המכירות.

הפרויקט מציג את הכוח של ניתוח נתונים מתקדם בקבלת החלטות שיווקיות ובשיפור הביצועים העסקיים.
אנו מאמינים שהתובנות שהגענו אליהן יסייעו למנהלי השיווק ולמחלקות המכירה לשפר את האסטרטגיות שלהם ולמקסם את הרווחים.

5. גישה לבסיס נתונים

Host :
aws-0-us-west-1.pooler.supabase.com

Database name :
postgres

Port :
6543

User :
postgres.xrkcvsdwwwcxhjkjuayc

Database Password :
pivsaj-fajkug-1jyxHe

1.6 קוד פייתון - עיבוד נתונים

Code :

```
# התקנה במידה וצריך
! pip install SQLAlchemy
! pip install pandas
```

Response :

```
Requirement already satisfied: SQLAlchemy in /usr/local/lib/python3.10/dist-packages (2.0.36)
Requirement already satisfied: typing-extensions>=4.6.0 in /usr/local/lib/python3.10/dist-packages (from
SQLAlchemy) (4.12.2)
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from SQLAlchemy)
(3.1.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-packages (from pandas)
(1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from
pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas)
(2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas)
(2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-
dateutil>=2.8.2->pandas) (1.16.0)
```

Code :

```
# ייבוא ספריות
import pandas as pd
from pandas import Series, DataFrame

import sqlalchemy
from sqlalchemy import create_engine
```

Code :

```
# חיבור למסד נתונים:
engine = create_engine("postgresql+psycopg2://
postgres.xrkcvsdwwwcxhjkjuayc:jisxi6-dafbic-tubVyb@aws-0-us-
west-1.pooler.supabase.com:6543/postgres")
```

Code :

```
# בדיקת חיבור
try:
    with engine.connect() as connection:
        print("החיבור למסד הנתונים הצליח!")
except Exception as e:
    print(f"שגיאה בחיבור למסד הנתונים: {e}”)
```

Response :

```
# החיבור למסד הנתונים הצליח!
```

1.6 קוד פייתון - עיבוד נתונים

Code :

```
# בדיקת חיבור
query = "select * from stg.artist limit 5;"
df_artist = pd.read_sql(query, engine)
df_artist
```

Response :

	artistid	name	last_update
0	1	AC/DC	2023-01-04 12:05:33.609691
1	2	Accept	2023-01-04 12:05:33.611359
2	3	Aerosmith	2023-01-04 12:05:33.612373
3	4	Alanis Morissette	2023-01-04 12:05:33.613312
4	5	Alice In Chains	2023-01-04 12:05:33.614249

Code :

```
#קריאת קבצים
department = pd.read_csv("/content/raw-department.txt")
department
```

Response :

	department_id-department_name
0	1-General
1	2-Sales Support
2	3-IT

Code :

```
department_budget = pd.read_json("/content/raw-department-budget.txt",
lines=True)
department_budget
```

Response :

	sub_dep_id	sub_dep_name	department_id	budget
0	1	managers	1	3000
1	2	managers2	1	1500
2	1	sales support john	2	2000
3	2	sales support joe	2	1000
4	3	sales support johnson	2	2500
5	4	sales support eduards	2	2500

1.6 קוד פייתון - עיבוד נתונים

Code :

```
# חיבור שתי טבלאות (מחלקות ותקציב) על פי השורות (axis=0)
all_department_budget = pd.concat([department_budget,
department_budget_2], axis=0)
all_department_budget
```

Response :

	sub_dep_id	sub_dep_name	department_id	budget
0	1	managers	1	3000
1	2	managers2	1	1500
2	1	sales support john	2	2000
3	2	sales support joe	2	1000
4	3	sales support johnson	2	2500
5	4	sales support eduards	2	2500
0	1	IT purchases	3	2000
1	2	IT maintenance	3	1500
2	3	IT other	3	1000

Code :

```
#מחיקת עמודות מיותרות (עמודת תת מחלקה, עמודת מנהל תת מחלקה)
all_department_budget.drop(columns=['sub_dep_id', 'sub_dep_name'],
inplace=True)
all_department_budget
```

Response :

	department_id	budget
0	1	3000
1	1	1500
2	2	2000
3	2	1000
4	2	2500
5	2	2500
0	3	2000
1	3	1500
2	3	1000

1.6 קוד פייתון - עיבוד נתונים

Code :

```
#סך תקציב לכל מחלקה
total_budget = all_department_budget.groupby("department_id",
as_index=False)["budget"].sum()
total_budget
```

Response :

	department_id	budget
0	1	4500
1	2	8000
2	3	4500

Code :

```
#שינוי שמות העמודות
total_budget.columns = ["department_id", "total_budget"]
total_budget
```

Response :

	department_id	total_budget
0	1	4500
1	2	8000
2	3	4500

Code :

```
department
```

Response :

	department_id-department_name
0	1-General
1	2-Sales Support
2	3-IT

1.6 קוד פייתון - עיבוד נתונים

Code :

```
#והוספת כותרות dataframe שמירה ב delimiter , פיצול עמודה על סמך ה
department[['department_id', 'department_name']] =
department['department_id-department_name'].str.split('-', expand=True)
department
```

Response :

	department_id-department_name	department_id	department_name
0	1-General	1	General
1	2-Sales Support	2	Sales Support
2	3-IT	3	IT

Code :

```
#מחיקת עמודות מקוריות
department = department.drop('department_id-department_name', axis=1)
department
```

Response :

	department_id	department_name
0	1	General
1	2	Sales Support
2	3	IT

Code :

```
# המרת department_id ל int בשתי הטבלאות
department["department_id"] = department["department_id"].astype(int)
total_budget["department_id"] =
total_budget["department_id"].astype(int)

# חיבור בין הטבלאות ויצרת dataframe חדש
updated_department_budget = pd.merge(department, total_budget,
on="department_id", how="inner")
updated_department_budget
```

Response :

	department_id	department_name	total_budget
0	1	General	4500
1	2	Sales Support	8000
2	3	IT	4500

1.6 קוד פייתון - עיבוד נתונים

Code :
```
#הורדת הטבלה
updated_department_budget.to_sql(
    name="department_budget",  # שם הטבלה ב-PostgreSQL
    con=engine,                # החיבור ל-Database
    schema="stg",              # שם הסכמה
    if_exists="append",        # מוסיף נתונים לטבלה קיימת
    index=False                # לא מוסיף אינדקס כעמודה
)
```
Response :

```
3
```

Code :
```
# שאילתא sql לשליפת תאריכים ייחודיים
dates_df = pd.read_sql_query(
    """
    select distinct date(invoicedate) as date
    from stg.invoice
    where invoicedate is not null
    order by date;
    """,
    engine
)
```
Code :
```
print(dates_df.head())
```
Response :

```
         date
0  2018-01-01
1  2018-01-02
2  2018-01-03
3  2018-01-04
4  2018-01-05
```

Code :
```
#בדיקת סוג עמודות בdataframe
print(dates_df.dtypes)
```
Response :

```
date    object
dtype: object
```

1.6 קוד פייתון - עיבוד נתונים

Code :
```
# בדיקת נתונים ייחודיים
print(f"Number of unique dates: {len(dates_df)}")
```
Response :

```
Number of unique dates: 1592
```

Code :
```
# המרת עמודת date לערכים מסוג datetime.date
dates_df["date"] = pd.to_datetime(dates_df["date"])
```
Code :
```
# בדיקה ממוקדת לפורמט עמודת date ב dataframe
print(dates_df["date"].dtypes)
```
Response :

```
datetime64[ns]
```

Code :
```
# ייבוא ספרייה עבור תקשורת עם הAPI
import requests
```

Code :
```
# פרטי הAPI-
api_key = 'fca_live_U0iQHftaOfbTqsNeICa8uyxl77es5sG9yOM3TPLJ'
base_url = 'https://api.freecurrencyapi.com/v1/latest'
```

Code :
```
#רשימה להשמירת שערי החליפין
exchange_rates = []
```

Code :
```
# יצירת סט של תאריכים שכבר יש נתונים עבורם
existing_dates = {rate["date"] for rate in exchange_rates}
```

1.6 קוד פייתון - עיבוד נתונים

Code :

```
# בניית לולאה לשליפת שערי חליפין
for date in dates_df["date"]:
    if date in existing_dates:  # בדיקה אם התאריך כבר טופל
        print(f"Skipping date: {date} (already processed)")
        continue  # דילוג על תאריך שכבר קיים

    print(f"Fetching data for date: {date}")  # הודעה לתיעוד
    params = {  # הגדרת פרמטרים לבקשת ה-API
        "apikey": api_key,  # מפתח גישה ל-API
        "base_currency": "USD",  # מטבע בסיס
        "currencies": "ILS",  # מטבע יעד
        "date": date  # התאריך הנוכחי
    }
    # קריאה ל-API
    response = requests.get(base_url, params=params)

    if response.status_code == 200:  # בדיקה אם הבקשה הצליחה
        rates = response.json().get("data", {})  # שליפת נתונים מהתשובה
        ils_rate = rates.get("ILS")  # שליפת שער ההמרה לשקל
        if ils_rate is not None:  # בדיקה אם קיבלנו שער המרה
            exchange_rates.append({"date": date, "currency": "ILS", "rate":
ils_rate})  # הוספת הנתונים לרשימה
            existing_dates.add(date)  # הוספת התאריך לרשימת התאריכים הקיימים
        else:
            print(f"No exchange rate data returned for date: {date}")  # הודעה אם
אין נתונים
    else:
        print(f"Failed to fetch data for date: {date}. Status code:
{response.status_code}")  # הודעה אם הבקשה נכשלה
```

Response :

```
Fetching data for date: 2018-01-01 00:00:00
Fetching data for date: 2018-01-02 00:00:00
Fetching data for date: 2018-01-03 00:00:00
Fetching data for date: 2018-01-04 00:00:00
Fetching data for date: 2018-01-05 00:00:00
Fetching data for date: 2018-01-06 00:00:00
Fetching data for date: 2018-01-07 00:00:00
Fetching data for date: 2018-01-09 00:00:00
Fetching data for date: 2018-01-10 00:00:00
Fetching data for date: 2018-01-11 00:00:00
Fetching data for date: 2018-01-12 00:00:00
Fetching data for date: 2018-01-13 00:00:00
Fetching data for date: 2018-01-15 00:00:00
Fetching data for date: 2018-01-16 00:00:00
Fetching data for date: 2018-01-17 00:00:00
Fetching data for date: 2018-01-18 00:00:00
Fetching data for date: 2018-01-19 00:00:00
Fetching data for date: 2018-01-20 00:00:00
Fetching data for date: 2018-01-21 00:00:00
Fetching data for date: 2018-01-22 00:00:00
Fetching data for date: 2018-01-23 00:00:00
Failed to fetch data for date: 2018-01-23 00:00:00. Status code: 429
Fetching data for date: 2018-01-24 00:00:00
Failed to fetch data for date: 2018-01-24 00:00:00. Status code: 429
Fetching data for date: 2018-01-25 00:00:00
...
Failed to fetch data for date: 2018-10-13 00:00:00. Status code: 429
Fetching data for date: 2018-10-14 00:00:00
Failed to fetch data for date: 2018-10-14 00:00:00. Status code: 429
Fetching data for date: 2018-10-16 00:00:00
```

1.6 קוד פייתון - עיבוד נתונים

Code :

```
#בדיקת תקינות נתוני ההמרה והמחשת התוצאות
print(f"Number of exchange rates fetched: {len(exchange_rates)}")  #
הדפסת מספר השערים שהתקבלו

# הדפסת מספר דוגמאות של נתונים
print("Example of fetched exchange rates:")
for rate in exchange_rates[:5]:  # מציג את 5 הנתונים הראשונים
    print(rate)  # הדפסת כל נתון של שער ההמרה

# בדיקה אם הנתונים תקינים (לוודא שאין שערים שליליים או אפסיים)
for rate in exchange_rates:
    if rate["rate"] <= 0:
        print(f"Warning: Invalid exchange rate for {rate['date']} - rate
is {rate['rate']}")
```

Code :

```
# יצירת DF מהתוצאות
rates_df = pd.DataFrame(exchange_rates)
```

Code :

```
#בדיקת תקינות נתוני ההמרה והמחשת התוצאות
print("Exchange rats Dataframe:")
rates_df.head()
```

Response :

Exchange rats Dataframe:

	date	currency	rate
0	2018-01-01	ILS	3.64392
1	2018-01-02	ILS	3.64392
2	2018-01-03	ILS	3.64392
3	2018-01-04	ILS	3.64392
4	2018-01-05	ILS	3.64392

Code :

```
# טעינת הנתונים לטבלת postgresql
table_name = "exchange_rates"  # שם הטבלה בה יישמרו הנתונים
schema = "stg"  # הסכמה בה נמצאת הטבלה

# מוסיף את הנתונים לטבלה קיימת מבלי למחוק את הנתונים הישנים-טעינת הנתונים ל postgresql
rates_df.to_sql(table_name, engine, schema=schema, if_exists="append",
index=False)
```

2.6 קוד SQL - בניית DWH

```
--#1

create schema if not exists dwh;-- יצירת סכמה במידה ולא קיימת

----------------------------------------
--#2
--DIM_CURRENCY

create table if not exists dwh.dim_currency as (
select *
from stg.exchange_rates
);

--הצגת הטבלה
select *
from dwh.dim_currency


----------------------------------------
--#3
--DIM_PLAYLIST

create table if not exists dwh.dim_playlist as (
select distinct playlisttrack.*,
                playlist.name
from stg.playlisttrack
join stg.playlist
on playlisttrack.playlistid = playlist.playlistid
);

--הצגת הטבלה
select *
from dwh.dim_playlist
```

2.6 קוד SQL - בניית DWH

```sql
--#4
--DIM_CUSTOMER


create table if not exists dwh.dim_customer as (
with cte_right_name_v_domain as (
select customerid,
       firstname,
       concat(
            upper(substring(firstname, 1, 1)),
            lower(substring(firstname, 2, length(firstname)))
            )as name_v,
       lastname,
       concat(
          upper(substring(lastname, 1, 1)),
          lower(substring(lastname, 2, length(lastname)))
            )as lastname_v,
        email,
        right(email, length(email) - position('@' in email)) as domain
from stg.customer
)

select cte_right_name_v_domain.customerid,
       cte_right_name_v_domain.name_v as firstname,
       cte_right_name_v_domain.lastname_v as lastname,
       cte_right_name_v_domain.email,
       cte_right_name_v_domain.domain,
       stg.customer.company,
       stg.customer.address,
       stg.customer.city,
       stg.customer.state,
       stg.customer.country,
       stg.customer.postalcode,
       stg.customer.phone,
       stg.customer.fax,
       stg.customer.supportrepid,
       stg.customer.last_update
from cte_right_name_v_domain
join stg.customer
on cte_right_name_v_domain.customerid = stg.customer.customerid
);

--הצגת הטבלה
select *
from dwh.dim_customer
```

2.6 קוד SQL - בניית DWH

```
--#5
--DIM_EMPLOYEE

create table if not exists dwh.dim_employee as (
select distinct employee.*,
       department_budget.department_name ,
       department_budget.total_budget,
       extract (year from age(current_date, employee.hiredate)) ||' '||
'Years' as hire_length,
       right(employee.email, length(employee.email) - position('@' in
employee.email)) as domain,
       case
           when employee.employeeid in (select distinct reportsto from
stg.employee where reportsto is not null)
           then '1'
           else '0'
       end as is_manager
from stg.employee
join stg.department_budget
on stg.employee.departmentid = stg.department_budget.department_id
);


select *
from dwh.dim_employee
```

2.6 קוד SQL - בניית DWH

```
-#6. -DIM_TRACK
--הצגת כל העמודות לצורך השוואה ובדיקה לפני יצירת הטבלה
select table_name,
       column_name,
       data_type
from information_schema.columns
where table_schema = 'stg'
  and table_name in ('track', 'album', 'artist', 'genre', 'mediatype')
order by table_name, ordinal_position;
--בניית הטבלה
create table if not exists dwh.dim_track as (
select
    t.trackid as track_id,
    t.name as track_name,
    a.albumid as album_id,
    a.title as album_title,
    a.last_update as album_last_update,
    ar.artistid as artist_id,
    ar.name as artist_name,
    ar.last_update as artist_last_update,
    g.genreid as genre_id,
    g.name as genre_name,
    g.last_update as genre_last_update,
    mt.mediatypeid as mediatype_id,
    mt.name as mediatype_name,
    mt.last_update as mediatype_last_update,
    t.milliseconds as track_duration_milliseconds,
    round(t.milliseconds / 1000.0, 2) as track_duration_seconds,
    concat(
        floor(t.milliseconds / 60000)::text,
        ':',
        lpad(floor((t.milliseconds % 60000) / 1000)::text, 2, '0')
    ) as track_duration_formatted,
    t.unitprice as track_unit_price,
    t.composer as track_composer,
    t.bytes as track_size_bytes,
    t.last_update as track_last_update
from stg.track t
    join stg.album a
        on t.albumid = a.albumid
    join stg.artist ar
        on a.artistid = ar.artistid
    join stg.genre g
        on t.genreid = g.genreid
    join stg.mediatype mt
        on t.mediatypeid = mt.mediatypeid
);

--הצגת הטבלה
select *
from dwh.dim_track dt
```

2.6 קוד SQL - בניית DWH

```
--#7
--FACT_INVOICE

select *
from stg.invoice

-- בדיקה האם הטבלה fact_invoice יצירה שלה - אם לא .כבר קיימת.
create table if not exists dwh.fact_invoice as (

    -- שליפת הנתונים הנדרשים מתוך טבלת המקור stg.invoice
select invoiceid,        -- מזהה ייחודי של כל חשבונית
        customerid,       -- מזהה הלקוח של החשבונית
        invoicedate,      -- תאריך יצירת החשבונית
        total             -- הסכום הכולל של החשבונית
from stg.invoice     -- מקור המידע הוא טבלת  stg
);

-- הצגת כל הנתונים מתוך הטבלה החדשה
select *
from dwh.fact_invoice;

-----------------------------------------
--#8

select *
from stg.invoiceline

-- FACT_INVOICELINE:
--יצירת טבלה עם עמודות נבחרות בלבד
create table if not exists dwh.fact_invoiceline as (
    select
        invoicelineid,  -- מזהה ייחודי לכל שורת חשבונית
        invoiceid,      -- מזהה החשבונית שאליה השורה שייכת
        trackid,        -- מזהה השיר שנרכש
        unitprice,      -- מחיר ליחידה עבור המוצר בשורה זו
        quantity,       -- מספר היחידות שנרכשו
        last_update     -- תאריך העדכון האחרון של הרשומה
    from stg.invoiceline
);

-- הצגת הנתונים מהטבלה החדשה
select *
from dwh.fact_invoiceline;

-----------------------------------------
```

3.6 קוד SQL - ניתוח מידע

```
--sql analysis

-----------------------------------------------1
-- יצירת טבלה זמנית עם מספר השירים הייחודיים לכל פלייליסט
with cte_track_count as (
    select
        playlistid,
        count(distinct trackid) as track_count -- מספר השירים הייחודיים
בכל פלייליסט
    from dwh.dim_playlist
    group by playlistid
),
-- טבלה זמנית למציאת הפלייליסט עם הכי הרבה שירים
max_playlist as (
    select
        'max' as category,
        playlistid,
        track_count -- המספר המקסימלי של שירים
    from cte_track_count
    where track_count = (select max(track_count) from cte_track_count)
),
-- טבלה זמנית למציאת הפלייליסט עם הכי מעט שירים
min_playlist as (
    select
        'min' as category,
        playlistid,
        track_count -- המספר המינימלי של שירים
    from cte_track_count
    where track_count = (select min(track_count) from cte_track_count)
),
-- איבור תוצאות הפלייליסטים עם הכי הרבה והכי מעט שירים
combined_results as (
    select * from max_playlist
    union all
    select * from min_playlist
)
-- שליפה סופית עם ממוצע השירים לכל הפלייליסטים
select
    category, -- קטגוריה: מקסימום או מינימום
    playlistid, -- מזהה הפלייליסט
    track_count, -- מספר השירים
    (select avg(track_count) from cte_track_count) as avg_track_count --
ממוצע מספר השירים
from combined_results;
```

3.6 קוד SQL - ניתוח מידע

```
-----------------------------------------2
-- תחילה, נאסוף את כמות המכירות לכל שיר.
-- כדי להתאים בין fact_invoiceline ו-fact_invoice אנו מחברים את הטבלאות
מכירות לחשבוניות.
select
    il.trackid, -- מזהה ייחודי של השיר
    count(il.invoicelineid) as total_sales -- סך המכירות (מספר פעמים
שהשיר נמכר)
from
    dwh.fact_invoiceline il -- טבלת השורות בחשבונית
join
    dwh.fact_invoice i -- טבלת החשבוניות הראשית
on
    il.invoiceid = i.invoiceid -- התאמת כל שורת מכירה לחשבונית הראשית
group by
    il.trackid; -- קיבוץ לפי מזהה השיר לקבלת סך המכירות.
-- כעת נעטוף את השאילתה הפנימית ונחלק את השירים לקבוצות בהתאם לכמות
המכירות.
select
    trackid, -- מזהה ייחודי של השיר
    total_sales, -- סך המכירות לשיר
    -- בהתאם לכמות המכירות CASE חלוקת השירים לקבוצות לפי
    case
        when total_sales = 0 then '0' -- קבוצה: לא נמכר כלל
        when total_sales between 1 and 5 then '1-5' -- קבוצה: 1 עד 5
מכירות
        when total_sales between 6 and 10 then '5-10' -- קבוצה: 6 עד 10
מכירות
        else '10<' -- קבוצה: מעל 10 מכירות
    end as sales_group -- קבוצה אליה השיר משתייך
from (
    -- שאילתה פנימית לאיסוף נתוני המכירות
    select
        il.trackid, -- מזהה ייחודי של השיר
        count(il.invoicelineid) as total_sales -- סך המכירות
    from
        dwh.fact_invoiceline il
    join
        dwh.fact_invoice i
    on
        il.invoiceid = i.invoiceid
    group by
        il.trackid
) as sales_data -- שם זמני לשאילתה הפנימית
order by
    total_sales desc; -- סדר התוצאות מהשיר עם הכי הרבה מכירות להכי מעט.
```

3.6 קוד SQL - ניתוח מידע

```sql
----------------------------------------3
----A
-- חישוב סך המכירות לכל מדינה
with country_sales as (
    select
        c.country,
        sum(i.total) as total_sales
    from
        dwh.fact_invoice i
    join
        dwh.dim_customer c
    on
        i.customerid = c.customerid
    group by
        c.country
),
-- בחירת 5 המדינות עם המכירות הגבוהות ביותר
top_5_countries as (
    select
        country,
        total_sales
    from
        country_sales
    order by
        total_sales desc
    limit 5
),
-- בחירת 5 המדינות עם המכירות הנמוכות ביותר
bottom_5_countries as (
    select
        country,
        total_sales
    from
        country_sales
    order by
        total_sales asc
    limit 5
)
-- שילוב והצגת התוצאה הסופית
select
    country,
    total_sales,
    'Top 5' as category
from
    top_5_countries

union all

select
    country,
    total_sales,
    'Bottom 5' as category
from
    bottom_5_countries

order by
    category,
    total_sales desc;
```

3.6 קוד SQL - ניתוח מידע

```
----------------------------------------3
--b
 -- חישוב סך המכירות של כל ז'אנר בכל מדינה
with genre_sales as (
    select
        c.country,
        t.genre_name,
        sum(il.quantity * il.unitprice) as genre_sales
    from
        dwh.fact_invoice i
    join
        dwh.fact_invoiceline il
    on
        i.invoiceid = il.invoiceid
    join
        dwh.dim_customer c
    on
        i.customerid = c.customerid
    join
        dwh.dim_track t
    on
        il.trackid = t.track_id
    group by
        c.country, t.genre_name
),
-- חישוב סך המכירות הכולל בכל מדינה
total_country_sales as (
    select
        c.country,
        sum(i.total) as total_sales
    from
        dwh.fact_invoice i
    join
        dwh.dim_customer c
    on
        i.customerid = c.customerid
    group by
        c.country
)
-- חישוב אחוזי המכירות לכל ז'אנר ודירוג גלובלי
select
    gs.country,
    gs.genre_name,
    gs.genre_sales || ' $' as genre_sales,
    round(gs.genre_sales * 100.0 / tcs.total_sales, 2) || '%' as sales_percentage,
    rank() over (partition by gs.country order by gs.genre_sales desc) as
country_sales_rank
from
    genre_sales gs
join
    total_country_sales tcs
on
    gs.country = tcs.country
order by
    country_sales_rank;
```

3.6 קוד SQL - ניתוח מידע

```
----------------------------------------4
-- יצירת טבלה זמנית לספירת לקוחות ותיוג מדינות
with customer_count_per_country as (
    select
        country,
        count(customerid) as customer_count, -- ספירת הלקוחות בכל מדינה
        case
            when count(customerid) = 1 then 'other' -- מדינה עם לקוח יחיד
מתויגת כ-'other'
            else country -- שמירת שם המדינה המקורי
        end as country_label -- עמודת תיוג חדשה למדינה
    from dwh.dim_customer
    group by country
),
-- טבלה זמנית שמכילה מידע מסוכם על לקוחות
customer_aggregates as (
    select
        customerid,
        count(invoiceid) as order_count, -- מספר ההזמנות ללקוח
        avg(total) as avg_total_per_customer -- ממוצע סכום ההזמנות ללקוח
    from dwh.fact_invoice
    group by customerid
),
-- טבלה זמנית שמסכמת נתונים לפי מדינות
country_aggregates as (
    select
        ccp.country_label as country, -- שם המדינה או התיוג 'other'
        count(c.customerid) as customer_count, -- מספר הלקוחות במדינה
        coalesce(avg(ca.order_count), 0) as avg_orders_per_customer, -- ממוצע
ההזמנות ללקוח
        coalesce(avg(ca.avg_total_per_customer), 0) as avg_total_per_customer
-- ממוצע הסכום הכולל ללקוח
    from customer_count_per_country ccp
    left join dwh.dim_customer c on ccp.country_label = c.country -- חיבור
למדינות בטבלת הלקוחות
    left join customer_aggregates ca on c.customerid = ca.customerid -- חיבור
להזמנות הלקוחות
    group by ccp.country_label
)
-- שליפה מסכמת עם מיון לפי מדינה
select
    country,
    customer_count,
    avg_orders_per_customer,
    avg_total_per_customer
from country_aggregates
order by country;
```

3.6 קוד SQL - ניתוח מידע

```
 ------------------------------------------5
-- טבלה זמנית לחישוב הכנסות שנתיות לכל עובד
with revenue_per_employee as (
    select
        e.employeeid,
        e.hire_length1,
        count(c.customerid) as customer_count,
        extract(year from f.invoicedate) as _year, -- חילוץ השנה מתאריך החשבונית
        sum(f.total) as year_total -- סיכום ההכנסות השנתיות
    from dwh.dim_employee e
    join dwh.dim_customer c on c.supportrepid = e.employeeid -- חיבור עובדים
ללקוחותיהם
    join dwh.fact_invoice f on c.customerid = f.customerid
    group by e.employeeid, e.hire_length1, extract(year from f.invoicedate)
),
-- חישוב הכנסות השנה הקודמת לכל עובד
yoy_cal as (
    select
        employeeid,
        hire_length1,
        customer_count,
        _year,
        year_total,
        lag(year_total) over (partition by employeeid order by _year asc) as
previous_year_total -- פונקציית lag להשגת ערך השנה הקודמת
    from revenue_per_employee
),
-- חישוב אחוז השינוי משנה לשנה
percentage_growth as (
    select
        employeeid,
        hire_length1,
        _year,
        customer_count,
        year_total,
        previous_year_total,
        case
            when previous_year_total is not null and previous_year_total != 0 then
                (year_total - previous_year_total) * 100.0 / previous_year_total --
חישוב אחוז הצמיחה
            else
                null
        end as yoy_growth_percentage
    from yoy_cal
)
-- שליפה סופית ומיון לפי עובד ושנה
select
    employeeid as employee_id,
    hire_length1 as hire_length,
    _year as year,
    customer_count,
    year_total,
    previous_year_total,
    yoy_growth_percentage
from percentage_growth
order by employeeid, _year;
```

3.6 קוד SQL - ניתוח מידע

```
 ------------------------------------------6
-- יצירת חלוקה לעונות והוספת עמודות עבור מכירות ונתוני לקוחות
with seasonal_sales as (
    select
        dt.genre_name,
        extract(month from fi.invoicedate) as month,
        case
            when extract(month from fi.invoicedate) in (12, 1, 2) then 'winter'
            when extract(month from fi.invoicedate) in (3, 4, 5) then 'spring'
            when extract(month from fi.invoicedate) in (6, 7, 8) then 'summer'
            when extract(month from fi.invoicedate) in (9, 10, 11) then 'fall'
        end as season,
        fil.unitprice * fil.quantity as sale_amount,
        fi.customerid
    from
        dwh.dim_track dt
    join
        dwh.fact_invoiceline fil on dt.track_id = fil.trackid
    join
        dwh.fact_invoice fi on fil.invoiceid = fi.invoiceid
)

-- חישוב סיכומי מכירות ומספר לקוחות ייחודיים לפי ז'אנר ועונה
, genre_summary as (
    select
        genre_name,
        season,
        sum(sale_amount) as total_revenue,
        count(distinct customerid) as unique_customers
    from
        seasonal_sales
    group by
        genre_name, season
)

-- דירוג ז'אנרים בעונות לפי סך הרווחים
, top_genres as (
    select
        genre_name,
        season,
        total_revenue,
        unique_customers,
        rank() over (partition by season order by total_revenue desc) as rank
    from
        genre_summary
)

-- סינון ל-3 הז'אנרים המובילים בכל עונה והצגת התוצאות
select
    genre_name,
    season,
    total_revenue,
    unique_customers
from
    top_genres
where rank <= 3
order by
    season, total_revenue desc;
```

4.6 קוד פייתון - ניתוח מידע

Code :

```
# ייבוא ספריות

import pandas as pd

from sqlalchemy import create_engine

import matplotlib.pyplot as plt

import seaborn as sns
```

Code :

```
# חיבור למסד נתונים:
engine = create_engine("postgresql+psycopg2://
postgres.xrkcvsdwwwcxhjkjuayc:jisxi6-dafbic-tubVyb@aws-0-us-
west-1.pooler.supabase.com:6543/postgres")
```

Code :

```
#יצירת df לכל טבלה
currency = pd.read_sql("select * from dwh.dim_currency", engine)
track = pd.read_sql("select * from dwh.dim_track", engine)
playlist = pd.read_sql("select * from dwh.dim_playlist", engine)
customer  = pd.read_sql("select * from dwh.dim_customer", engine)
employee = pd.read_sql("select * from dwh.dim_employee", engine)
invoice = pd.read_sql("select * from dwh.fact_invoice", engine)
invoiceline = pd.read_sql("select * from dwh.fact_invoiceline", engine)
```

4.6 קוד פייתון - ניתוח מידע

Code :

```
def create_bar_chart(data, x_col, y_col, title, xlabel, ylabel, color, rotation=45, ax=None, y_col2=None,
color2=None, label1=None, label2=None, group_col=None, group_colors=None):
    """
    פונקציה ליצירת גרף בר מותאם אישית.
    פרמטרים:
    - data: DataFrame שמכיל את הנתונים
    - x_col: שם העמודה עבור ציר ה-x
    - y_col: שם העמודה הראשונה עבור ציר ה-y
    - y_col2: (אופציונלי) שם העמודה השנייה להשוואה (לדוגמה: usd מול ils)
    - group_col: (אופציונלי) עמודת קבוצה (לדוגמה: שנה)
    - title: כו תרת הגרף
    - xlabel: תווית עבור ציר ה-x
    - ylabel: תווית עבור ציר ה-y
    - color: צבע העמודות עבור y_col (אם אין group_col)
    - group_colors: (אופציונלי) רשימת צבעים עבור הקבוצות ב-group_col
    - color2: (אופציונלי) צבע העמודות עבור y_col2
    - label1: (אופציונלי) תווית לעמודות y_col
    - label2: (אופציונלי) תווית לעמודות y_col2
    - rotation: זווית הסיבוב של התוויות בציר ה-x (ברירת מחדל: 45 מעלות)
    - ax: אובייקט של subplot לציור הגרף (ברירת מחדל: None)
    """
    if ax is None:
        ax = plt.gca()  # שימוש בציר הנוכחי אם לא סופק ax

    x = range(len(data[x_col].unique()))  # מיקום העמודות לפי הערכים ב-x_col

    if group_col:
        groups = data[group_col].unique()  # רשימת הקבוצות (לדוגמה, שנים)
        bar_width = 0.8 / len(groups)  # התאמת רוחב העמודות לפי מספר הקבוצות

        for i, group in enumerate(groups):
            subset = data[data[group_col] == group]  # סינון הנתונים לפי הקבוצה הנוכחית
            ax.bar(
                [pos + i * bar_width - 0.4 + bar_width / 2 for pos in x],  # הזזת עמודות לכל קבוצה
                subset[y_col],
                width=bar_width,
                color=group_colors[i % len(group_colors)] if group_colors else color,  # בחירת צבע
                label=f"{group_col}: {group}"
            )
    else:
        # גרף לעמודה הראשונה
        ax.bar(
            [pos - 0.2 if y_col2 else pos for pos in x],  # אם יש עמודה שנייה, הזז את העמודות
            data[y_col],
            width=0.4 if y_col2 else 0.8,  # התאמת רוחב העמודות
            color=color,
            label=label1 or y_col
        )

    # גרף לעמודה השנייה (אם קיימת)
    if y_col2:
        ax.bar(
            [pos + 0.2 for pos in x],
            data[y_col2],
            width=0.4,
            color=color2,
            label=label2 or y_col2
        )

    # התאמות לגרף
    ax.set_title(title)
    ax.set_xticks(x)
    ax.set_xticklabels(data[x_col].unique(), rotation=rotation)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.legend()
```

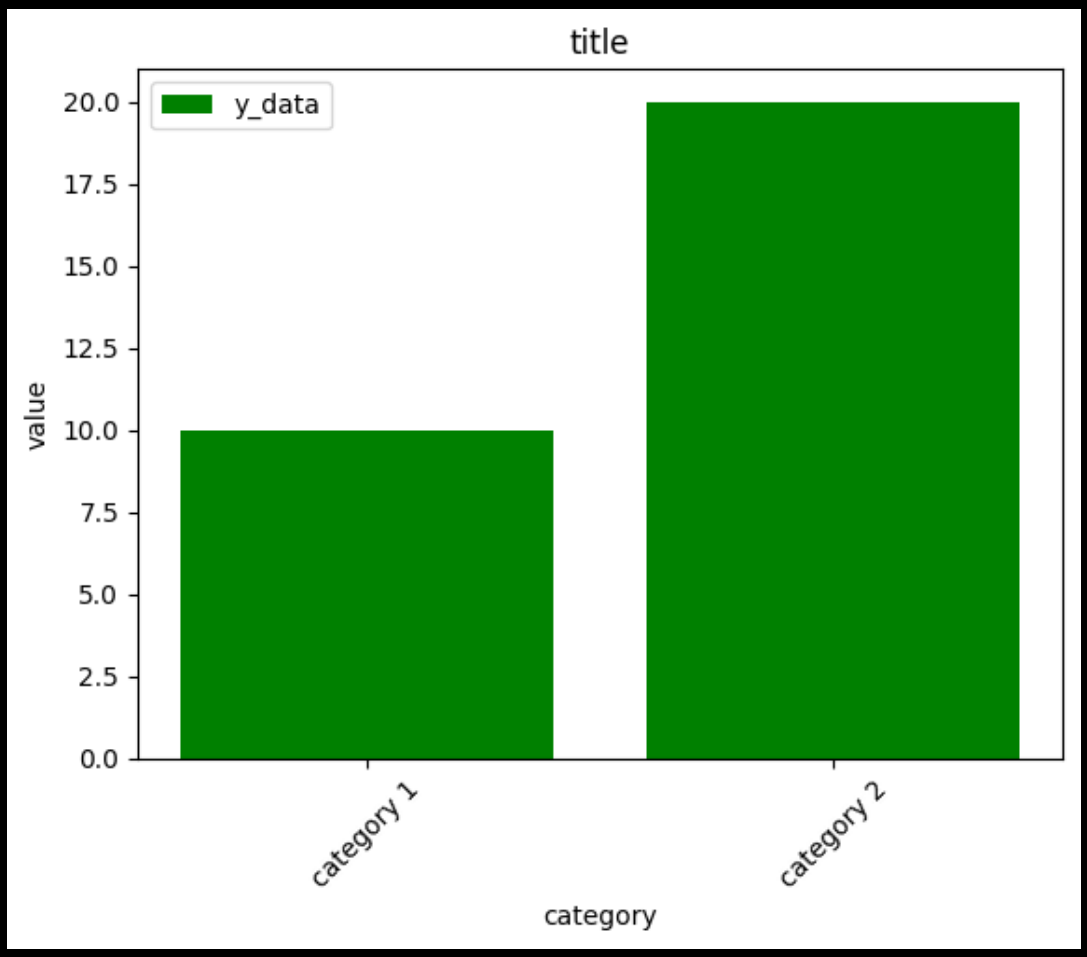
4.6 קוד פייתון - ניתוח מידע

Code :

```
# דוגמת שימוש בפונקציה ליצירת גרף בר
# יצירת DataFrame עם נתוני דוגמה
sample_data = pd.DataFrame({
    'x_data': ["category 1", "category 2"],  # נתונים עבור ציר ה-X
    'y_data': [10, 20]  # נתונים עבור ציר ה-Y
})

# קריאה לפונקציה עם נתוני הדוגמה
create_bar_chart(
    data=sample_data,                # DataFrame עם הנתונים
    x_col='x_data',                  # עמודה עבור ציר ה-X
    y_col='y_data',                  # עמודה עבור ציר ה-Y
    title="title",       # כותרת הגרף
    xlabel="category",               # תווית ציר ה-X
    ylabel="value",                  # תווית ציר ה-Y
    color='green'                    # צבע העמודות
)
```

Response :



4.6 קוד פייתון - ניתוח מידע

Code :

```
top_artists_albums = (
    track.groupby(['artist_id', 'artist_name'])['album_id']  # קיבוץ לפי מזהה ושם האמן
    .nunique()  # חישוב כמות האלבומים הייחודיים
    .reset_index()  # איפוס האינדקס ליצירת DataFrame
    .rename(columns={'album_id': 'album_count'})  # שינוי שם העמודה
    .sort_values('album_count', ascending=False)  # מיון לפי כמות האלבומים בסדר יורד
    .head(5)  # בחירת חמשת האמנים המובילים
)
# הצגת הנתונים
print(top_artists_albums)
```

Response :

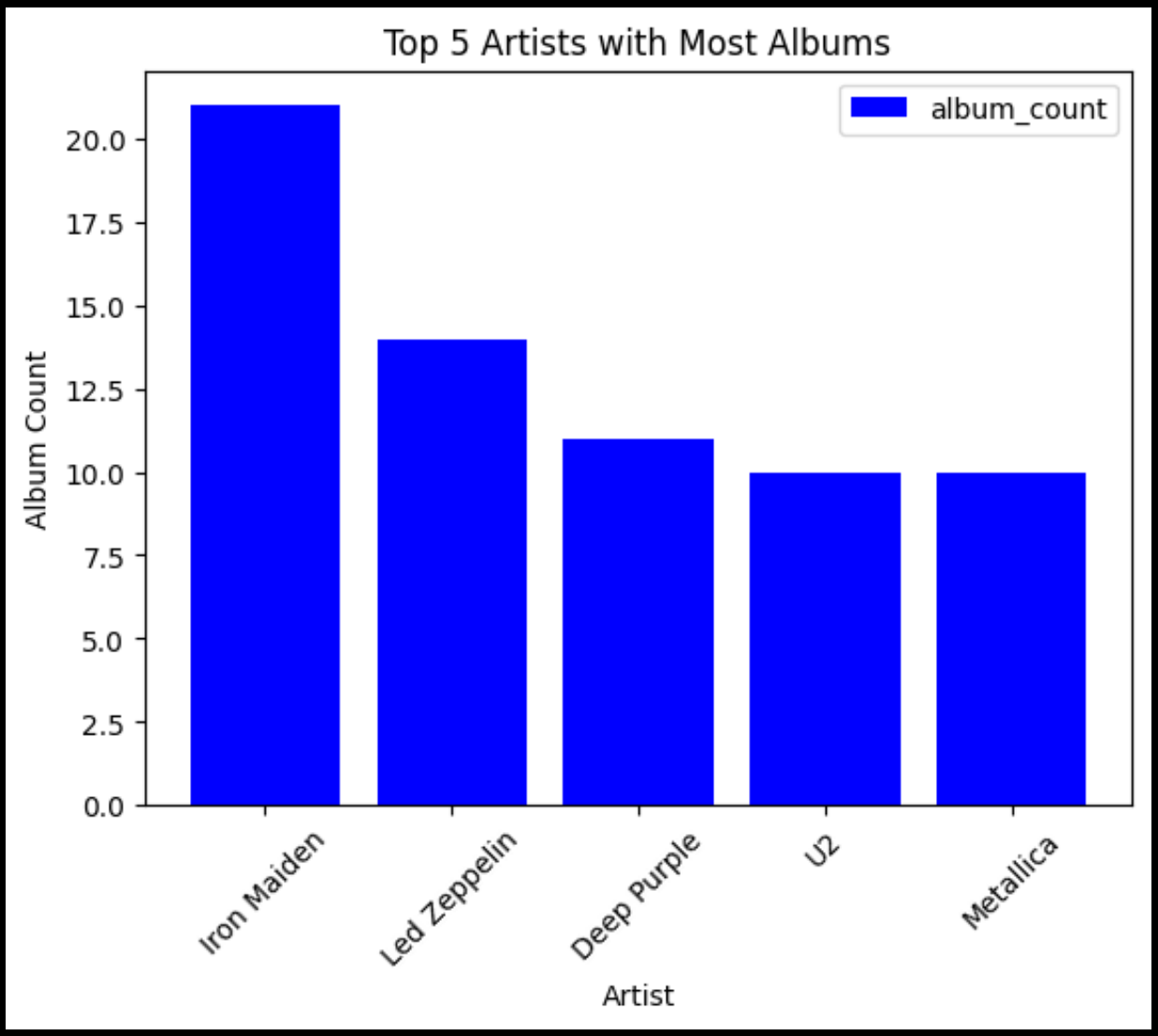
	artist_id	artist_name	album_count
58	90	Iron Maiden	21
21	22	Led Zeppelin	14
38	58	Deep Purple	11
114	150	U2	10
30	50	Metallica	10

4.6 קוד פייתון - ניתוח מידע

Code :

```
# יצירת גרף המציג את חמשת האמנים עם הכי הרבה אלבומים
create_bar_chart(top_artists_albums, 'artist_name', 'album_count',
'Top 5 Artists with Most Albums', 'Artist', 'Album Count', 'blue')
```

Response :



4.6 קוד פייתון - ניתוח מידע

Code :

```
top_artists_tracks = (
    track.groupby(['artist_id', 'artist_name'])['track_id']  # קיבוץ לפי מזהה ושם האמן
    .count()  # ספירת מספר השירים עבור כל אמן
    .reset_index()  # איפוס האינדקס ליצירת DataFrame
    .rename(columns={'track_id': 'track_count'})  # שינוי שם העמודה
    .sort_values('track_count', ascending=False)  # מיון לפי כמות השירים בסדר יורד
    .head(5)  # בחירת חמשת האמנים המובילים
)
# הצגת הנתונים
print(top_artists_tracks)
```

Response :

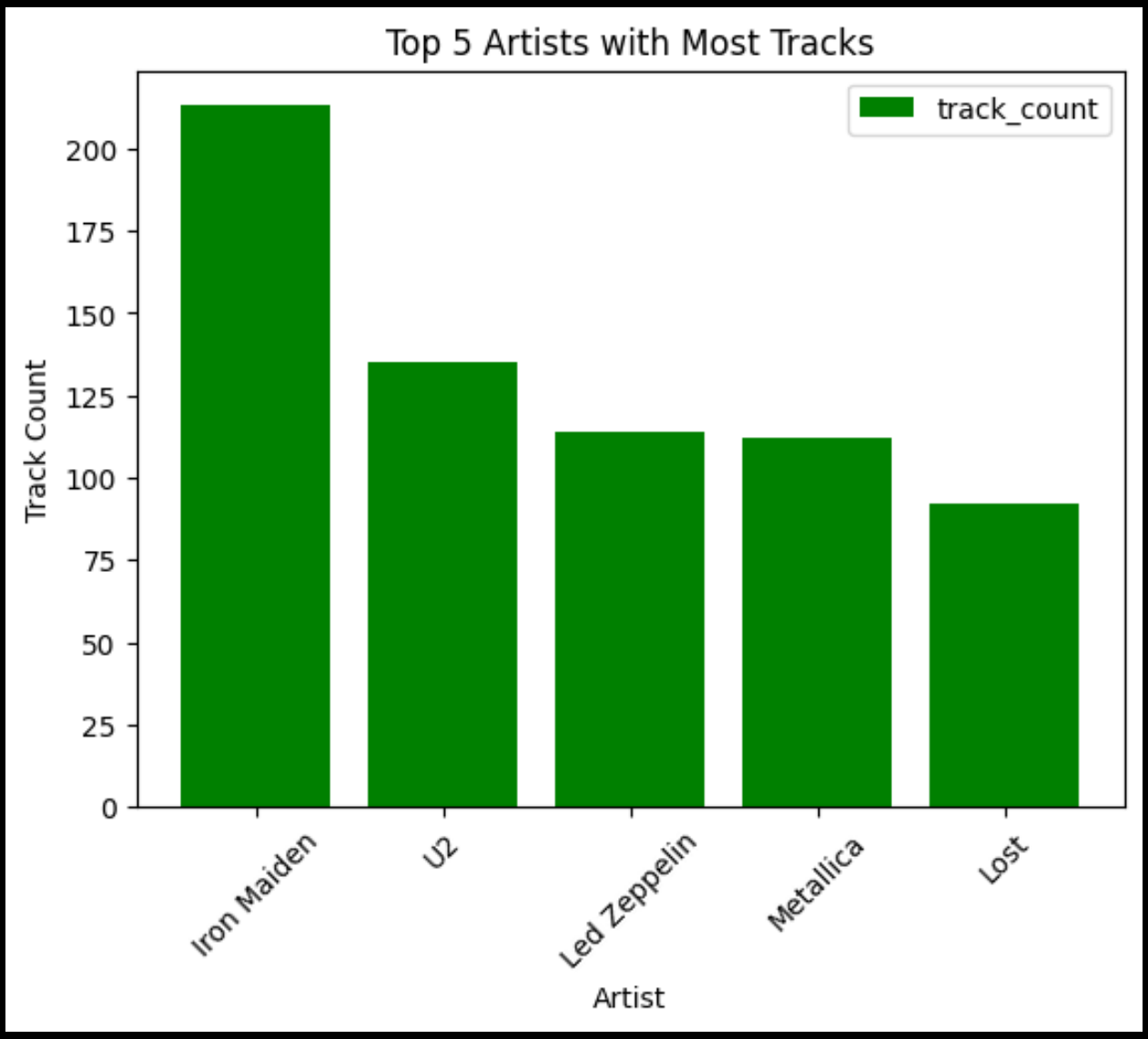
	artist_id	artist_name	track_count
58	90	Iron Maiden	213
114	150	U2	135
21	22	Led Zeppelin	114
30	50	Metallica	112
113	149	Lost	92

4.6 קוד פייתון - ניתוח מידע

Code :

```
# יצירת גרף המציג את חמשת האמנים עם הכי הרבה שירים
create_bar_chart(top_artists_tracks, 'artist_name', 'track_count',
'Top 5 Artists with Most Tracks', 'Artist', 'Track Count', 'green')
```

Response :



4.6 קוד פייתון - ניתוח מידע

Code :

```
top_genres_tracks = (
    track.groupby(['genre_id', 'genre_name'])['track_id']  # קיבוץ לפי
מזהה ושם הז'אנר
    .count()  # ספירת מספר השירים עבור כל ז'אנר
    .reset_index()  # איפוס האינדקס ליצירת DataFrame
    .rename(columns={'track_id': 'track_count'})  # שינוי שם העמודה
    .sort_values('track_count', ascending=False)  # מיון לפי כמות השירים
בסדר יורד
    .head(5)  # בחירת חמשת הז'אנרים המובילים
)

# הצגת הנתונים
print(top_genres_tracks)
```

Response :

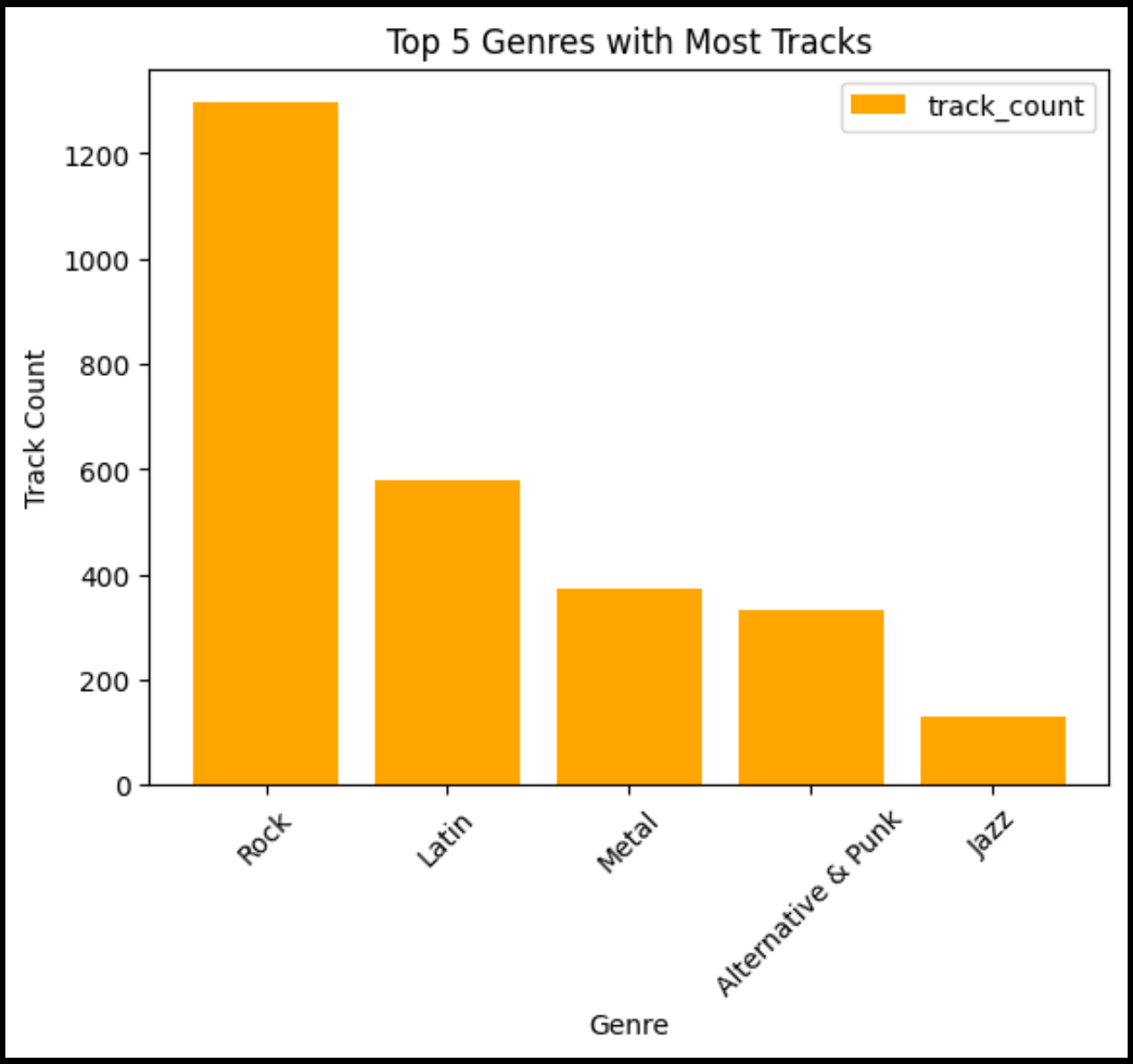
	genre_id	genre_name	track_count
0	1	Rock	1297
6	7	Latin	579
2	3	Metal	374
3	4	Alternative & Punk	332
1	2	Jazz	130

4.6 קוד פייתון - ניתוח מידע

Code :

```
# יצירת גרף המציג את חמשת הז'אנרים עם הכי הרבה שירים
create_bar_chart(top_genres_tracks, 'genre_name', 'track_count',
'Top 5 Genres with Most Tracks', 'Genre', 'Track Count', 'orange')
```

Response :



4.6 קוד פייתון - ניתוח מידע

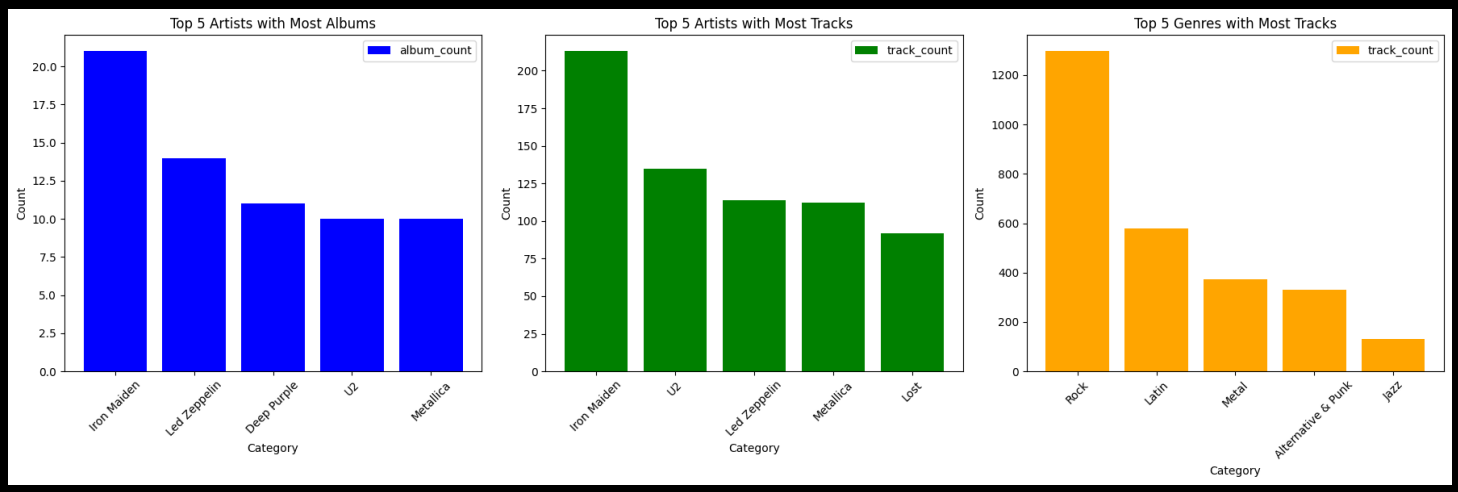
Code :

```
# יצירת Subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 6))  # שלושה גרפים בשורה אחת

# שימוש בלולאה לציור הגרפים
for ax, (data, x_col, y_col, title, color) in zip(
    axes,
    [
        (top_artists_albums, 'artist_name', 'album_count', 'Top 5 Artists with Most
Albums', 'blue'),
        (top_artists_tracks, 'artist_name', 'track_count', 'Top 5 Artists with Most
Tracks', 'green'),
        (top_genres_tracks, 'genre_name', 'track_count', 'Top 5 Genres with Most
Tracks', 'orange'),
    ]
):
    create_bar_chart(data, x_col, y_col, title, 'Category', 'Count', color, ax=ax)

# התאמת מרווחים והצגת הגרפים
plt.tight_layout()
plt.show()
```

Response :



4.6 קוד פייתון - ניתוח מידע

Code :

```
# המרת עמודת התאריך בטבלת currency לפורמט datetime
currency['date'] = pd.to_datetime(currency['date'])

# חיבור בין invoice ל-customer
customer_sales = invoice.merge(
    customer, how='inner', left_on='customerid', right_on='customerid'  # מיזוג טבלת הלקוחות עם טבלת החשבוניות
)

# חיבור נתוני שערי המרה מ-dim_currency
customer_sales = customer_sales.merge(
    currency, how='inner', left_on='invoicedate', right_on='date'  # חיבור לפי תאריך החשבונית
)

# יצירת עמודה חדשה לשם הלקוח המלא
customer_sales['customer_name'] = customer_sales['firstname'] + ' ' +
customer_sales['lastname']

# הוספת עמודת רכישות בשקלים
customer_sales['total_ils'] = customer_sales['total'] * customer_sales['rate']
# המרת סכומים מש"ח לדולר לפי שער ההמרה

# חישוב סכום הרכישות הכולל בדולרים ושקלים לכל לקוח
top_customers = (
    customer_sales.groupby(['customerid', 'customer_name'])[['total',
'total_ils']]  # קיבוץ לפי מזהה ושם הלקוח
    .sum()  # חישוב סכומי הרכישות
    .reset_index()  # איפוס האינדקס
    .rename(columns={'total': 'total_usd', 'total_ils': 'total_ils'})  # שינוי שם העמודות
    .sort_values('total_usd', ascending=False)  # מיון לפי סכום הרכישות בדולרים
    .head(5)  # בחירת חמשת הלקוחות המובילים
)

# הצגת הנתונים
print(top_customers)
```

Response :

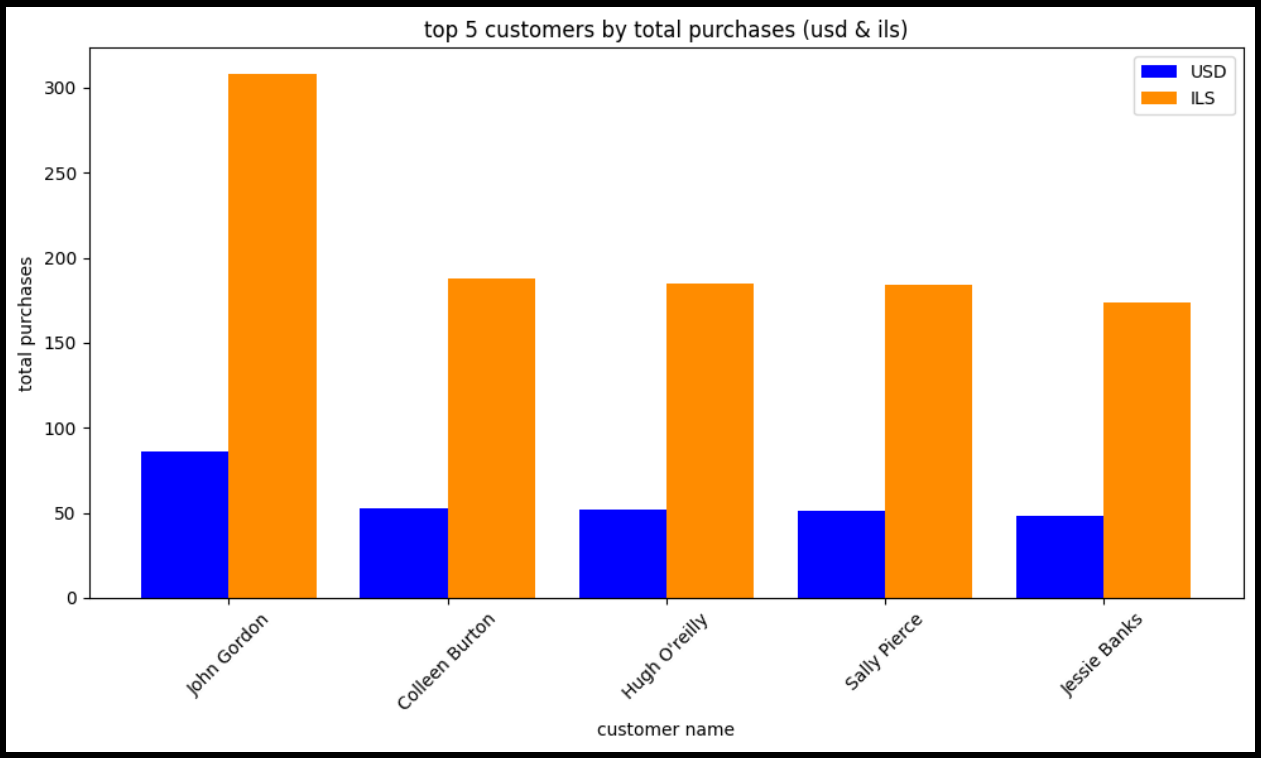
```
     customerid   customer_name  total_usd   total_ils
22           23     John Gordon      86.13  308.098275
256         285  Colleen Burton      52.53  187.906363
43           46   Hugh O'reilly      51.64  184.705306
201         225    Sally Pierce      51.53  184.328461
246         273    Jessie Banks      48.52  173.563772
```

4.6 קוד פייתון - ניתוח מידע

Code :

```
# יצירת גרף המציג את 5 הלקוחות עם סכומי הרכישות הגבוהים ביותר בשקלים
ובדולרים
fig, ax = plt.subplots(figsize=(10, 6))
# שימוש בפונקציה ליצירת גרף עם שני משתנים (USD ו-ILS)
create_bar_chart(
    data=top_customers,
    x_col='customer_name',
    y_col='total_usd',
    y_col2='total_ils',  # הוספת עמודה שנייה להשוואה
    color='blue',
    color2='darkorange',  # צבעים לעמודות
    label1='USD',
    label2='ILS',
    title='top 5 customers by total purchases (usd & ils)',
    xlabel='customer name',
    ylabel='total purchases',
    rotation=45,
    ax=ax
)
# התאמת עיצוב
plt.tight_layout()
plt.show()
```

Response :



4.6 קוד פייתון - ניתוח מידע

Code :

```
# יצירת עמודות חדשות לשנה ולחודש מתוך עמודת התאריך
invoice['year'] = pd.to_datetime(invoice['invoicedate']).dt.year
invoice['month'] = pd.to_datetime(invoice['invoicedate']).dt.month

# קיבוץ הנתונים לפי שנה וחודש וחישוב סכום המכירות
monthly_sales = (
    invoice.groupby(['year', 'month'])['total']  # קיבוץ לפי שנה וחודש
    .sum()  # חישוב סכום המכירות
    .reset_index()  # איפוס האינדקס ליצירת DataFrame
    .rename(columns={'total': 'monthly_sales'})  # שינוי שם העמודה
)

# הצגת הנתונים
print(monthly_sales.head(5))
```

Response :

	year	month	monthly_sales
0	2018	1	201.01
1	2018	2	237.68
2	2018	3	233.66
3	2018	4	262.45
4	2018	5	227.76

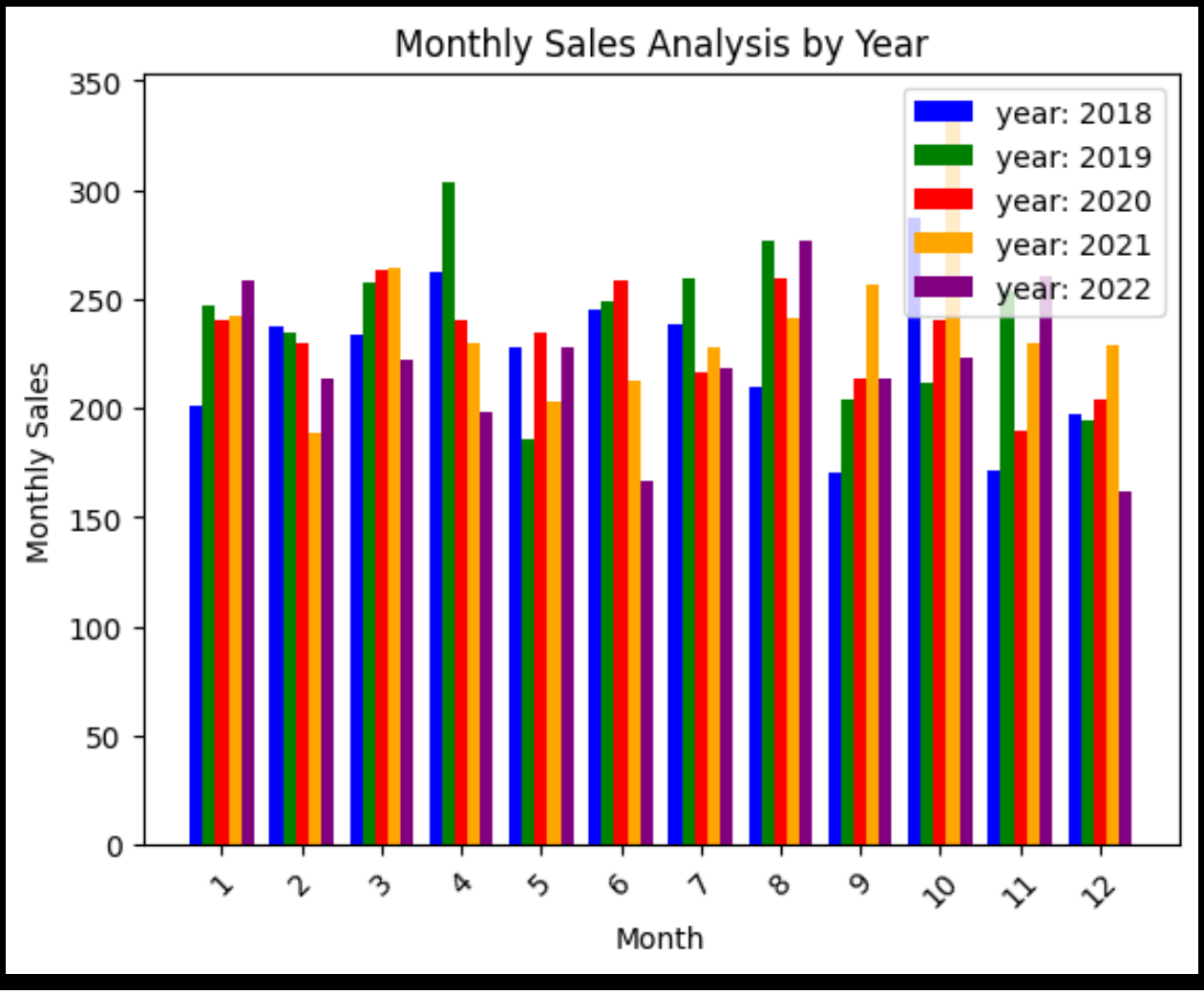
4.6 קוד פייתון - ניתוח מידע

Code :

```
# רשימת צבעים לכל שנה
colors = ['blue', 'green', 'red', 'orange', 'purple']

# יצירת גרף
create_bar_chart(
    data=monthly_sales,
    x_col='month',
    y_col='monthly_sales',
    title='Monthly Sales Analysis by Year',  # כותרת גרף באנגלית
    xlabel='Month',  # תווית ציר ה-X באנגלית
    ylabel='Monthly Sales',  # תווית ציר ה-Y באנגלית
    color='gray',  # ברירת מחדל אם אין group_col
    group_col='year',  # עמודת הקבוצות
    group_colors=colors,  # צבעים לקבוצות
    rotation=45
)
plt.show()
```

Response :



4.6 קוד פייתון - ניתוח מידע

Code :

```
# חיבור נתונים בין track ל-invoiceline
track_sales = invoiceline.merge(
    track, how='inner', left_on='trackid', right_on='track_id'  # איחוד לפי track_id
)

# יצירת עמודה לחישוב המכירות (unitprice * quantity)
track_sales['sales'] = track_sales['unitprice'] *
track_sales['quantity']

# בדיקת הקורלציה בין אורך השיר (track_duration_seconds) למכירות (sales)
correlation =
track_sales['track_duration_seconds'].corr(track_sales['sales'])
print(correlation)
```

Response :

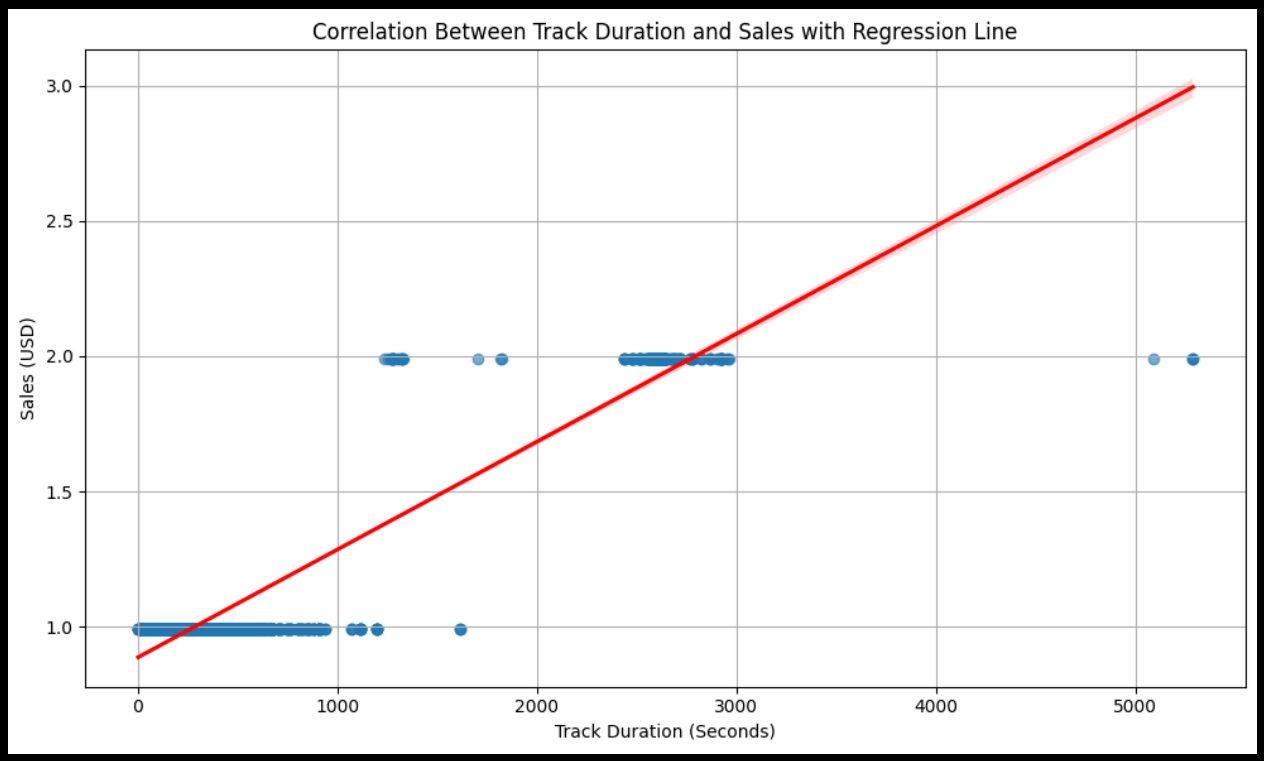
```
0.9621344288657433
```

4.6 קוד פייתון - ניתוח מידע

Code :

```
# יצירת גרף פיזור עם קו רגרסיה
plt.figure(figsize=(10, 6))
sns.regplot(x='track_duration_seconds', y='sales', data=track_sales,
scatter_kws={'alpha':0.6}, line_kws={'color':'red'})
plt.title("Correlation Between Track Duration and Sales with Regression
Line")
plt.xlabel("Track Duration (Seconds)")
plt.ylabel("Sales (USD)")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Response :



4.6 קוד פייתון - ניתוח מידע

Code :

```
# מיזוג של טבלאות invoice ו-invoiceline על trackid
purchased_songs_df = invoiceline.merge(track, left_on='trackid',
right_on='track_id', how='inner')
# שליפת כל הרכישות של כל הלקוחות
customer_purchases_df = invoice.merge(purchased_songs_df, left_on='invoiceid',
right_on='invoiceid', how='inner')
# יצירת רשימה לאחסון ההמלצות
recommendations_list = []
```

Code :

```
# עבור כל לקוח
for customer_id in customer['customerid']:
    # שליפת כל הרכישות של הלקוח
    customer_purchases =
customer_purchases_df[customer_purchases_df['customerid'] == customer_id]

    # זיהוי הז'אנרים של השירים שנרכשו על ידי הלקוח
    customer_genres =
customer_purchases.groupby('genre_name').size().sort_values(ascending=False).h
ead(2).index.tolist()

    # סינון השירים שלא נרכשו
    not_purchased_songs_df =
track[~track['track_id'].isin(customer_purchases['trackid'])]

    # סינון השירים לפי הז'אנרים המועדפים
    genre_filtered_songs =
not_purchased_songs_df[not_purchased_songs_df['genre_name'].isin(customer_genr
es)]

    # שליפת שלושה שירים פופולריים מכל ז'אנר
    for genre in customer_genres:
        popular_songs =
genre_filtered_songs[genre_filtered_songs['genre_name'] == genre].head(3)

        # הוספת ההמלצות לרשימה
        for song_id in popular_songs['track_id']:
            recommendations_list.append({
                'customer_id': customer_id,
                'genre': genre,
                'track_id': song_id,
                'track_name': popular_songs[popular_songs['track_id'] ==
song_id]['track_name'].values[0]
            })
# יצירת DataFrame עם ההמלצות
recommendations_df = pd.DataFrame(recommendations_list)
```

4.6 קוד פייתון - ניתוח מידע

Code :

```
# הצגת ה-DataFrame עם ההמלצות
recommendations_df
```

Response :

	customer_id	genre	track_id	track_name
0	1	Rock	1	For Those About To Rock (We Salute You)
1	1	Rock	2	Balls to the Wall
2	1	Rock	3	Fast As a Shark
3	1	Latin	205	Jorge Da Capadócia
4	1	Latin	206	Prenda Minha
...	...	...	...	...
3910	656	Rock	2	Balls to the Wall
3911	656	Rock	3	Fast As a Shark
3912	657	Latin	205	Jorge Da Capadócia
3913	657	Latin	206	Prenda Minha
3914	657	Latin	207	Meditação

3915 rows × 4 columns

4.6 קוד פייתון - ניתוח מידע

הצג את 10 האמנים עם הפוטנציאל הגבוה ביותר לעלייה בפופולריות בחודשים הקרובים, בהתבסס על הנתונים ומגמות שקיימות בהם. הצג את סכום הרווח המצטבר עבור כל אמן, ואת כמות האנשים הייחודיים שרכשו את השירים שלו.

Code :

```
# 1. חיבור הטבלאות (שירים, מכירות, לקוחות)
artist_sales = (
    invoiceline
    .merge(track, how="inner", left_on="trackid", right_on="track_id")
    .merge(invoice, how="inner", left_on="invoiceid",
right_on="invoiceid")
    .merge(customer, how="inner", left_on="customerid",
right_on="customerid")
)

# 2. המרת תאריך וחילוץ month + year
artist_sales['invoicedate'] =
pd.to_datetime(artist_sales['invoicedate'])
artist_sales['month'] = artist_sales['invoicedate'].dt.month
artist_sales['year'] = artist_sales['invoicedate'].dt.year

# 3. חישוב רווח חודשי (monthly_revenue) לכל אמן
artist_revenue = (
    artist_sales
    .groupby(['artist_name', 'year', 'month'])['unitprice']
    .sum()
    .reset_index()
    .rename(columns={'unitprice': 'monthly_revenue'})
)

# 4. ממוצע נייד לטווח קצר (3 חודשים)
artist_revenue['rolling_3m_mean'] = (
    artist_revenue
    .groupby("artist_name")['monthly_revenue']
    .transform(lambda x: x.rolling(window=3, min_periods=1).mean())
)

# 5. ממוצע מצטבר לטווח ארוך (Expanding), תוך הזזת חודש (shift)
artist_revenue['expanding_mean'] = (
    artist_revenue
    .groupby("artist_name")['monthly_revenue']
    .transform(lambda x: x.shift(1).expanding().mean())
)
```

4.6 קוד פייתון - ניתוח מידע

Code :

```
# 6. סימון עלייה – קצר/ארוך
artist_revenue['increase_potential_short'] = (
    artist_revenue['monthly_revenue'] > artist_revenue['rolling_3m_mean']
)
artist_revenue['increase_potential_long'] = (
    artist_revenue['monthly_revenue'] > artist_revenue['expanding_mean']
)
# 7. צבירת הרווחים בחודשים שבהם הייתה עלייה בכל טווח
short_term_revenue = (
    artist_revenue[artist_revenue['increase_potential_short']]
    .groupby('artist_name')['monthly_revenue']
    .sum()
    .reset_index()
    .rename(columns={'monthly_revenue': 'short_term_growth_revenue'})
)
long_term_revenue = (
    artist_revenue[artist_revenue['increase_potential_long']]
    .groupby('artist_name')['monthly_revenue']
    .sum()
    .reset_index()
    .rename(columns={'monthly_revenue': 'long_term_growth_revenue'})
)
# 8. חישוב סך המכירות (total_revenue) ומספר הרוכשים הייחודיים
total_revenue = (
    artist_sales
    .groupby('artist_name')['unitprice']
    .sum()
    .reset_index()
    .rename(columns={'unitprice': 'total_revenue'})
)
unique_customers = (
    artist_sales
    .groupby('artist_name')['customerid']
    .nunique()
    .reset_index()
    .rename(columns={'customerid': 'unique_customers'})
)
# 9. בניית טבלת סיכום (איחוד הנתונים)
artist_summary = (
    total_revenue
    .merge(unique_customers, on="artist_name", how="left")
    .merge(short_term_revenue, on="artist_name", how="left")
    .merge(long_term_revenue, on="artist_name", how="left")
)
# מילוי ערכי NaN אם אין חודשים חיוביים ב-0
artist_summary['short_term_growth_revenue'] =
artist_summary['short_term_growth_revenue'].fillna(0)
artist_summary['long_term_growth_revenue'] =
artist_summary['long_term_growth_revenue'].fillna(0)

# 10. חישוב מדד "התפוצצות" משולב (ניתן לשנות משקלות כרצונך)
artist_summary['blow_up_potential'] = (
    0.7 * artist_summary['short_term_growth_revenue'] +
    0.3 * artist_summary['long_term_growth_revenue']
)
```

4.6 קוד פייתון - ניתוח מידע

```
Code :
# 11. בחירת טופ 10 לפי המדד המשולב
top_10_artists = (
    artist_summary
    .sort_values('blow_up_potential', ascending=False)
    .head(10)
)
# 12. הצגת הטבלה (רק אחת) – עם סך מכירות, רוכשים ייחודיים, וערכי הצמיחה
print("\nTop 10 Artists Expected to Increase Sales & Popularity:\n")
print(top_10_artists[[
    'artist_name',
    'total_revenue',
    'unique_customers',
    'short_term_growth_revenue',
    'long_term_growth_revenue',
    'blow_up_potential'
]])
# 13. יצירת גרף עמודות כפול להמחשה (לדוגמה, total_revenue מול unique_customers)
fig, ax = plt.subplots(figsize=(14, 8))
create_bar_chart(
    data=top_10_artists,
    x_col='artist_name',
    y_col='total_revenue',
    y_col2='unique_customers',
    title="Top 10 Artists – Potential for Growth",
    xlabel="Artist Name",
    ylabel="Revenue / Unique Customers",
    color='blue',
    color2='orange',
    label1="Total Revenue",
    label2="Unique Customers",
    rotation=45,
    ax=ax
)
plt.tight_layout()
plt.show()
```

Response is in the next page

4.6 קוד פייתון - ניתוח מידע

Response :

```
Top 10 Artists Expected to Increase Sales & Popularity:

         artist_name  total_revenue  unique_customers  \
69       Iron Maiden         949.41               430
85              Lost         610.93               215
82      Led Zeppelin         493.02               291
95         Metallica         494.01               298
37       Deep Purple         400.95               271
156               U2         408.87               259
115        Pearl Jam         306.90               216
51     Faith No More         235.62               174
159  Various Artists         248.49               191
49      Eric Clapton         234.63               179

     short_term_growth_revenue  long_term_growth_revenue  blow_up_potential
69                      560.34                    464.31            531.531
85                      324.37                    386.06            342.877
82                      307.89                    276.21            298.386
95                      313.83                    255.42            296.307
37                      266.31                    258.39            263.934
156                     256.41                    263.34            258.489
115                     190.08                    217.80            198.396
51                      172.26                    134.64            160.974
159                     141.57                    157.41            146.322
49                      142.56                    145.53            143.451
```