

## ***Firas Heib Portfolio***

### **iTunes Data Warehouse and Dashboard Project #**

#### **Department Budget Integration**

In the **Dim\_Employee.sql** model, we integrated department budget data with employee data to provide insights into the relationship between employees and department budgets.

#### **Key Actions:**

##### **Joining Department Data:**

The department table included department\_id, department name, and budget.

This table was joined with the employee table using the department\_id field.

##### **Calculating Department Budgets:**

Each employee was assigned the budget of their respective department.

This allowed budget details to be included in the employee dimension table.

```

import pandas as pd
from sqlalchemy import create_engine

# קריאת הקבצים ל-Pandas DataFrames
departments = pd.read_csv(r'C:\Python data\raw-department.txt', delimiter=';')
department_budget1 = pd.read_json(r'C:\Python data\raw-department-budget.txt', lines=True)
department_budget2 = pd.read_json(r'C:\Python data\raw-department-budget2.txt')

# הצגת התוצאות
print(departments.head())
print(department_budget1.head())
print(department_budget2.head())

```

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

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

sub_dep_id	sub_dep_name	department_id	budget
0	1 IT purchases	3	2000
1	2 IT maintenance	3	1500
2	3 IT other	3	1000

```

# של התקציבים DataFrames איחוד התוצאות בין שני ה
merged_budget = pd.concat([department_budget1, department_budget2])

# שמירת התוצאה לקובץ JSON
merged_budget.to_json(r'C:\Python data\merged_department_budget.json', orient='records')

# על פי הדרישה delimiter איחוד עם
merged_budget_delimited = pd.merge(departments, merged_budget, on='department_id', how='left')

# שמירת התוצאה לקובץ CSV עם delimiter '|'
merged_budget_delimited.to_csv(r'C:\Python data\merged_department_budget_delimited.txt', sep='|', index=False)

# department_id על התוצאות לפי Join ביצוע
joined_data = pd.merge(departments, merged_budget, on='department_id', how='inner')

# הצגת התוצאות המשותפות
print(joined_data.head())

```

department_id	department_name	sub_dep_id	sub_dep_name	budget
0	1 General	1	managers	3000
1	1 General	2	managers2	1500
2	2 Sales Support	1	sales support john	2000
3	2 Sales Support	2	sales support joe	1000
4	2 Sales Support	3	sales support johnson	2500

```

# יצירת Postgres לטבלה
engine = create_engine('postgresql://postgres:postgres@localhost:5432/chinook')

# Postgres לטבלה ב-DataFrame העלאת
joined_data.to_sql('stg_department_budget', con=engine, if_exists='replace', index=False)

print("Data uploaded successfully to Postgres.")

```

Data uploaded successfully to Postgres.

## ## 1. Dimensional Models (DBT)

**\*\*Dim\_Customer.sql\*\***

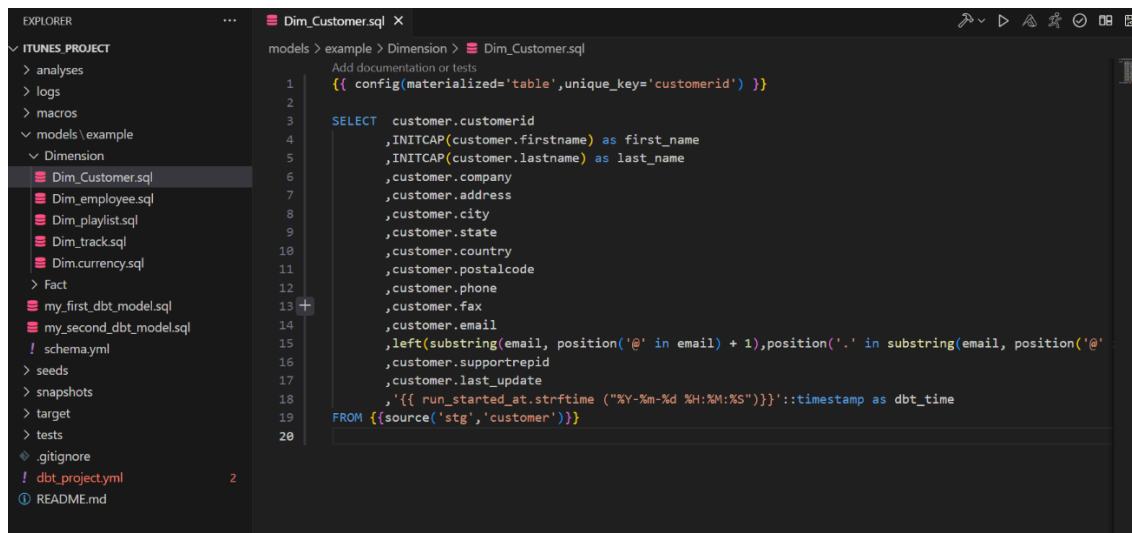
Goal: **\*\* Create a Customer Dimension table with customer details such as .name, address, phone number, and email domain extraction**

**\*\*Key Actions\*\*** -

.Standardized customer names to title case -

.Extracted email domains -

.Added a timestamp column ( ` dbt\_time` ) for process tracking -



```
1  {{ config(materialized='table', unique_key='customerid') }}
2
3  SELECT customer.customerid
4         ,INITCAP(customer.firstname) as first_name
5         ,INITCAP(customer.lastname) as last_name
6         ,customer.company
7         ,customer.address
8         ,customer.city
9         ,customer.state
10        ,customer.country
11        ,customer.postalcode
12        ,customer.phone
13        ,customer.fax
14        ,customer.email
15        ,left(substring(email, position('@' in email) + 1),position('.') in substring(email, position('@'
16        ,customer.supportrepid
17        ,customer.last_update
18        , '{{ run_started_at.strftime ("%Y-%m-%d %H:%M:%S") }}'::timestamp as dbt_time
19  FROM {{source('stg','customer')}}
20
```

**\*\*Dim\_Employee.sql\*\* ###**

Goal:**\*\* Create an Employee Dimension table, including department budgets \*\*** -  
.and identifying managers

**\*\*Key Actions\*\*** -

.Calculated years of employment for each employee -

.Identified managers using a case condition -

.Merged department budget data with employee details -

```

models > example > Dimension > Dim_employee.sql
Add documentation or tests
1  {{ config(materialized='table', unique_key='employeeid') }}
2
3  SELECT emp.*
4         ,db.department_name
5         ,db.budget
6         ,EXTRACT(YEAR FROM age(CURRENT_DATE, emp.hiredate)) AS years_in_company
7         ,left(substring(email, position('@' in email) + 1),position('.') in substring(email, position('@'
8         ,case when emp.employeeid in (select distinct reportsto from stg.employee) then 1
9         else 0
10        end as is_manager
11        , '{{ run_started_at.strftime ("%Y-%m-%d %H:%M:%S") }}'::timestamp as dbt_time
12  FROM {{source('stg','employee')}} emp
13  JOIN {{source('stg','department_budget')}} db on emp.departmentid = db.department_id

```

**\*\*Dim\_Playlist.sql\*\* ###**

Goal:\*\* Create a Playlist Dimension table to analyze playlists and track \*\* -  
 .updates

\*\*Key Actions\*\* -

.Combined playlist and track details using a join -

.Implemented incremental updates with a timestamp condition -

```

models > example > Dimension > Dim_playlist.sql
Add documentation or tests
1  select pl.playlistid as playlist_id
2         ,pl.name as playlist_name
3         ,pl.last_update as last_update
4         ,plt.trackid as track_id
5         ,plt.last_update as track_last_update
6         , '{{ run_started_at.strftime ("%Y-%m-%d %H:%M:%S") }}'::timestamp as dbt_time
7  from {{source('stg','playlist')}} as pl
8  left join {{source('stg','playlisttrack')}} plt on pl.playlistid = plt.playlistid
9  where 1=1
10         {% if is_incremental() %}
11         and plt.last_update::timestamp > (select max(pl.last_update) from {{this}})
12         {% endif %}
13
14

```

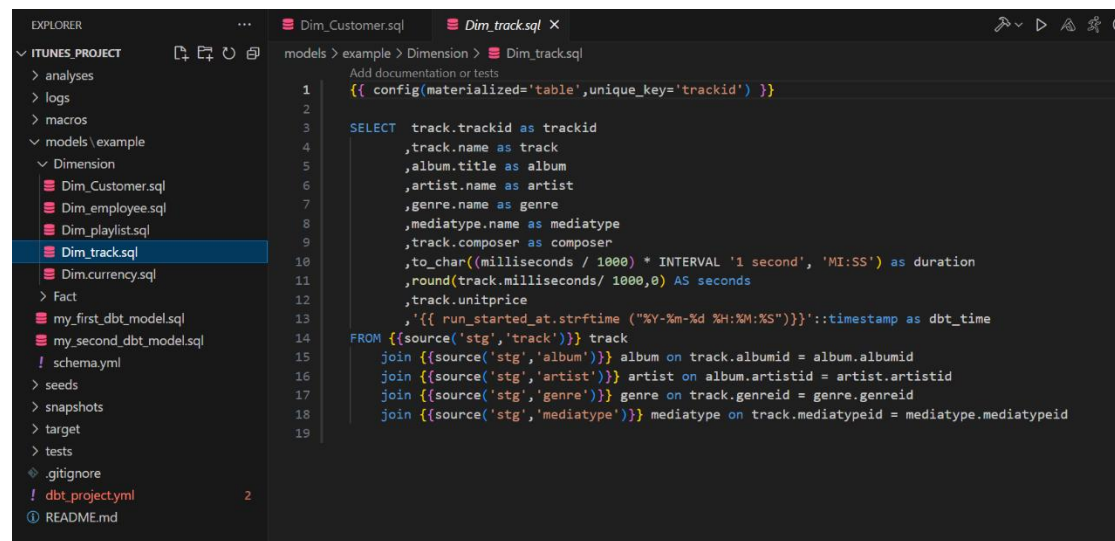
**\*\*Dim\_Track.sql\*\* ###**

**Goal:\*\* Create a Track Dimension table with details like genre, album, artist, \*\* -  
.track duration, and price**

**\*\*Key Actions\*\* -**

.Converted track duration from milliseconds to minutes:seconds format -

.Merged data from genre, album, and media type tables -



```
1  {{ config(materialized='table', unique_key='trackid') }}
2
3  SELECT track.trackid as trackid
4         ,track.name as track
5         ,album.title as album
6         ,artist.name as artist
7         ,genre.name as genre
8         ,mediatype.name as mediatype
9         ,track.composer as composer
10        ,to_char((milliseconds / 1000) * INTERVAL '1 second', 'MI:SS') as duration
11        ,round(track.milliseconds/ 1000,0) AS seconds
12        ,track.unitprice
13        , '{{ run_started_at.strftime ("%Y-%m-%d %H:%M:%S") }}'::timestamp as dbt_time
14  FROM {{source('stg','track')}} track
15       join {{source('stg','album')}} album on track.albumid = album.albumid
16       join {{source('stg','artist')}} artist on album.artistid = artist.artistid
17       join {{source('stg','genre')}} genre on track.genreid = genre.genreid
18       join {{source('stg','mediatype')}} mediatype on track.mediatypeid = mediatype.mediatypeid
19
```

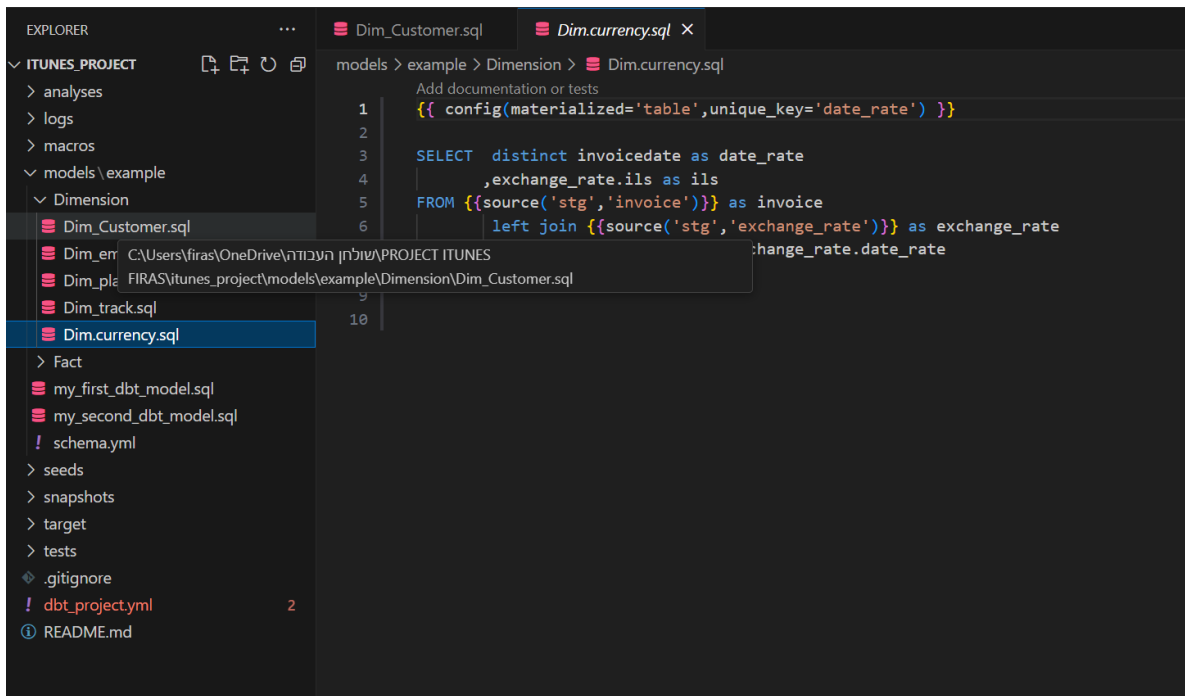
**\*\*Dim\_Currency.sql\*\* ###**

**Goal:\*\* Build a Currency Dimension table to incorporate historical exchange \*\* -  
.rates between USD and ILS**

**\*\*Key Actions\*\* -**

.Integrated exchange rates with sales data based on date -

.Prepared data for currency-based sales analysis -



## Fact Models .2 ##

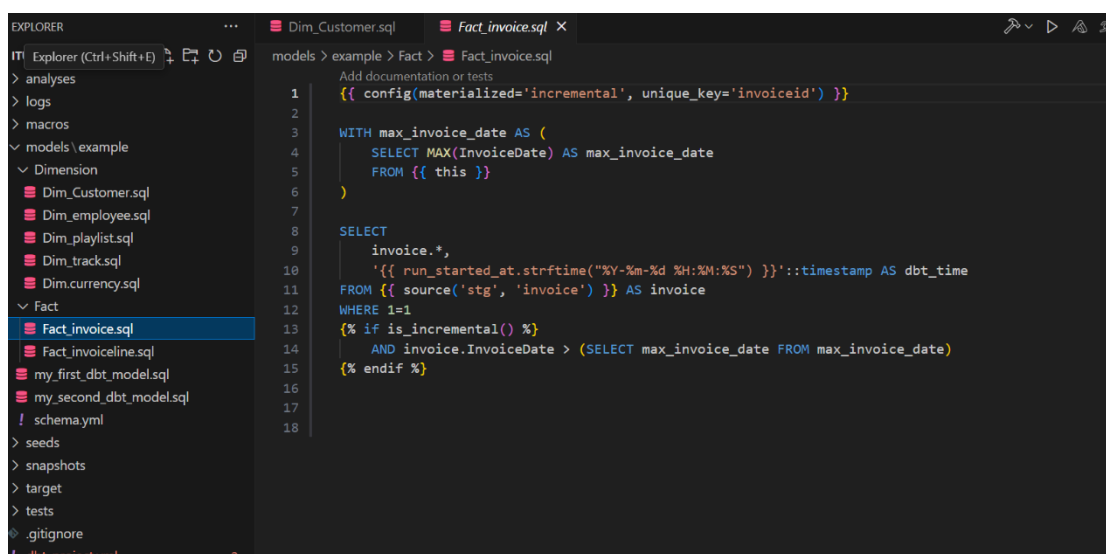
**\*\*Fact\_Invoice.sql\*\* ###**

.Goal: **\*\* Create an Invoice Fact table for high-level sales transactions\*\*** -

**\*\*Key Actions\*\*** -

.Implemented incremental updates based on the latest invoice date -

.Added process timestamp ( `dbt\_time` ) -



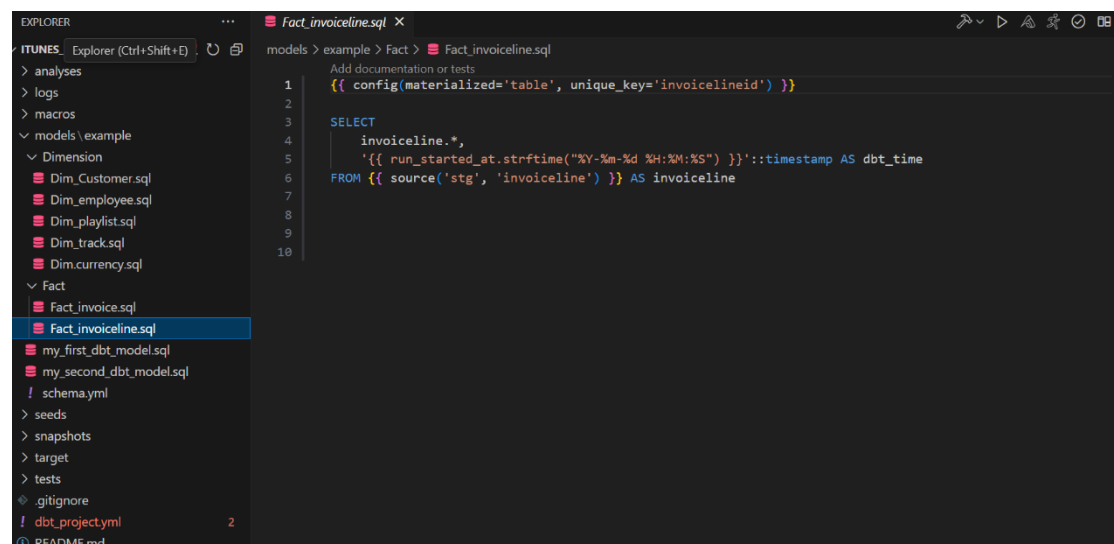
**\*\*Fact\_InvoiceLine.sql\*\* ###**

Goal:**\*\* Create an Invoice Line Fact table for detailed sales transaction \*\* -**  
**.analysis**

**\*\*Key Actions\*\* -**

**.Stored individual line items of invoices, including quantities and prices -**

**.Added a timestamp column for tracking updates -**



---

**API Integration for Currency Conversion .3 ##**

**\*\*API\_Currency.ipynb\*\* ###**

Goal:**\*\* Fetch and integrate exchange rates between USD and ILS using an \*\* -**  
**.external API**

**\*\*Key Actions\*\* -**

**.Retrieved historical exchange rates (2018-2022) via the Alpha Vantage API -**

.Stored data in a PostgreSQL database as a staging table -

.Prepared data for use in the `Dim\_Currency` model -

```
API_Currency.ipynb X
C:\Users> firas > OneDrive > שולחן העבודה > PROJECT ITUNES FIRAS > API_Currency.ipynb > ...
+ Code + Markdown | ▶ Run All | Clear All Outputs | Outline ...
Python 3.11.9

import requests
import pandas as pd
from sqlalchemy import create_engine
from datetime import datetime

# פרטי API
api_key = "DPFAXBCEFOHECJY"
start_date = "2018-01-01"
end_date = "2022-12-22"
base_currency = "USD"
target_currency = "ILS"

# פונקציה למשיכת נתונים מ-API
def get_daily_rates(start_date, end_date, base_currency, target_currency, api_key):
    url = f"https://www.alphavantage.co/query?function=FX_DAILY&from_symbol={base_currency}&to_symbol={target_currency}&outputsize=full&apikey={api_key}"
    response = requests.get(url)

    # בדיקה אם הבקשה הצליחה
    if response.status_code == 200:
        rates = response.json()['Time Series FX (Daily)']
        # סינון התאריכים
        filtered_rates = {date: rate for date, rate in rates.items() if start_date <= date <= end_date}
        return filtered_rates
    else:
        print(f"Error fetching data from API. Status code: {response.status_code}")
        return {}

# לקבלת שיערי המרה API-קריאה ל
daily_rates = get_daily_rates(start_date, end_date, base_currency, target_currency, api_key)

# Dataframe-עיבוד הנתונים ופירוט ל
converted_amounts = [{'date_rate': pd.to_datetime(date), 'usd': 1, 'ils': float(rate['4. close'])} for date, rate in daily_rates.items()]
df = pd.DataFrame(converted_amounts)
print(df)
```

```
date_rate  usd  ils
0  2022-12-22  1  3.4902
1  2022-12-21  1  3.4630
2  2022-12-20  1  3.4626
3  2022-12-19  1  3.4416
4  2022-12-16  1  3.4580
...
1294 2018-01-05  1  3.4281
1295 2018-01-04  1  3.4436
1296 2018-01-03  1  3.4481
1297 2018-01-02  1  3.4475
1298 2018-01-01  1  3.4666

[1299 rows x 3 columns]
```

```
# שנה את פרטי החתוכיות בהתאם למחסן הנתונים שלך (PostgreSQL-יצירת חיבור ל
engine = create_engine('postgresql://postgres:postgres@localhost/chinook')

# בסכמה 'stg' טעינת הנתונים לטבלת
df.to_sql('dim_currency', con=engine, if_exists='append', index=False, schema='stg')

print("Data loaded successfully into 'exchange_rate' table.")
```

Python

Data loaded successfully into 'exchange\_rate' table.

## ## 4. Business Intelligence (Power BI Dashboards)

**\*\*Dashboard 1: Sales Overview\*\* ###**

**\*\*Highlights\*\* -**



Top 5 Genres:\*\* Displayed the most popular music genres based on track \*\* -  
.and album counts

.Top 5 Countries by Sales:\*\* Analyzed sales by geographic location\*\* -

.Customer Spending:\*\* Identified top customers based on total purchases\*\* -

.Tooltip functionality added for genre-specific insights -

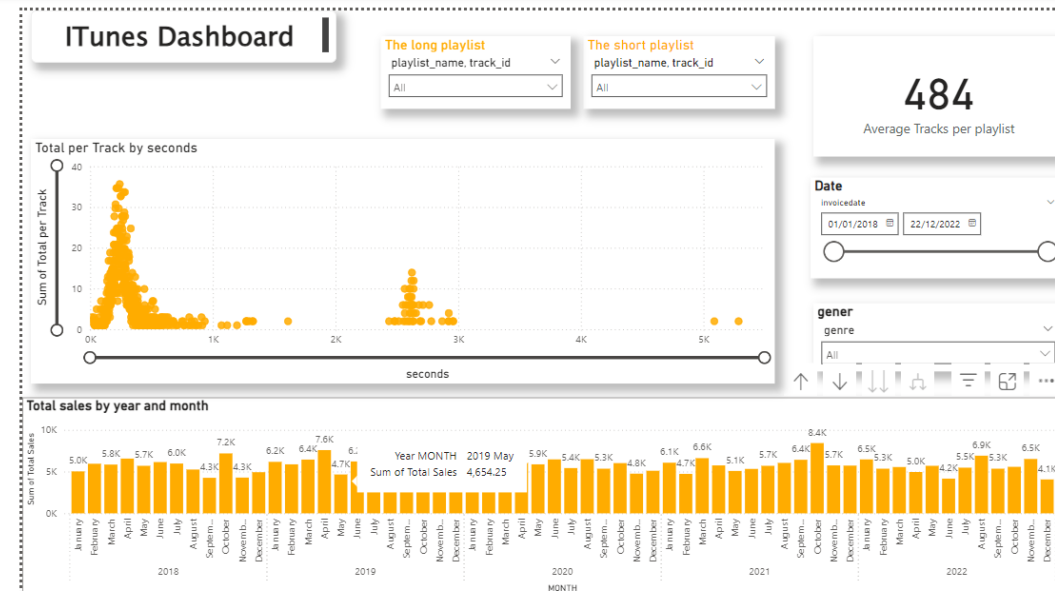
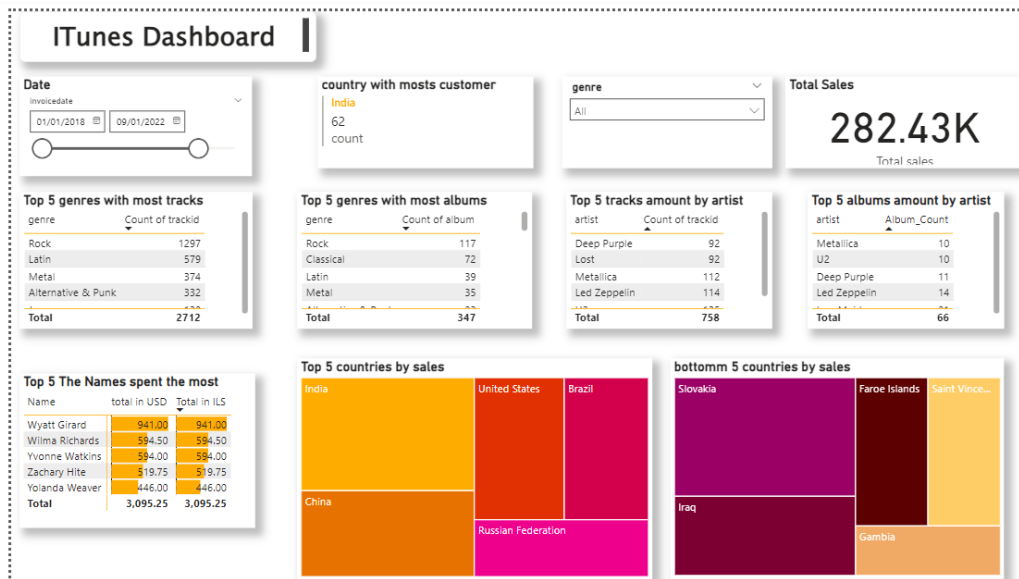
## **\*\*Dashboard 2: Track Analysis\*\* ###**

\*\*:.Highlights\*\* -

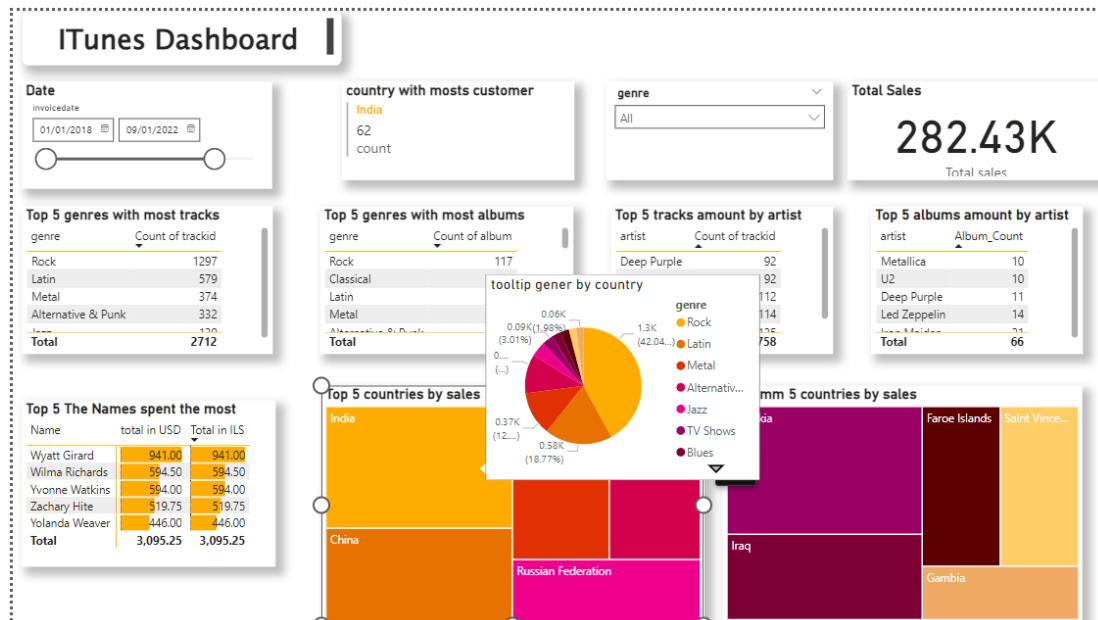
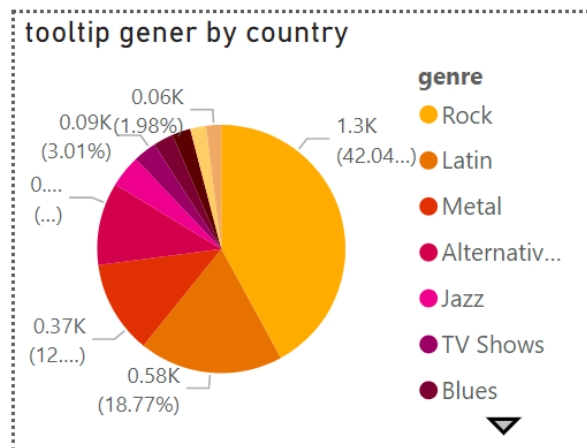
.Track Duration:\*\* Visualized distribution of track lengths in seconds\*\* -

Monthly Sales Trends:\*\* Analyzed sales patterns over time (monthly \*\* -  
.granularity)

.Added slicers for dynamic filtering by date and genre -



TOOLTIP:



Summary ##

:This project showcases end-to-end ETL and analytics capabilities

Data Engineering:\*\* Building a Star Schema with DBT, incorporating \*\* -  
.Dimensional and Fact tables

API Integration:\*\* Fetching external data for currency conversion and storing \*\* -  
.it in PostgreSQL

Data Visualization:\*\* Developing interactive dashboards in Power BI to \*\* -  
.provide actionable insights