TITLE : REAL TIME FOREX TRADING DASHBOARD

# ABSTRACT:

In an increasingly dynamic global economy, access to real-time and accurate foreign exchange (Forex) data has become crucial for informed decision-making. This project presents a **Real-Time Forex Trading Dashboard** focused on major currencies benchmarked against the Indian Rupee (INR). The solution integrates automated data ingestion from the TwelveData API, structured storage using Google BigQuery, and interactive visualization through Power BI. It enables trend analysis, threshold-based alerts, historical data modeling, and OLAP operations such as slicing, dicing, drill-down, and pivoting. By leveraging a robust, scalable architecture, the dashboard ensures seamless, minimal-intervention data handling and provides businesses with actionable insights to navigate the complexities of currency exchange fluctuations effectively.

# INTRODUCTION:

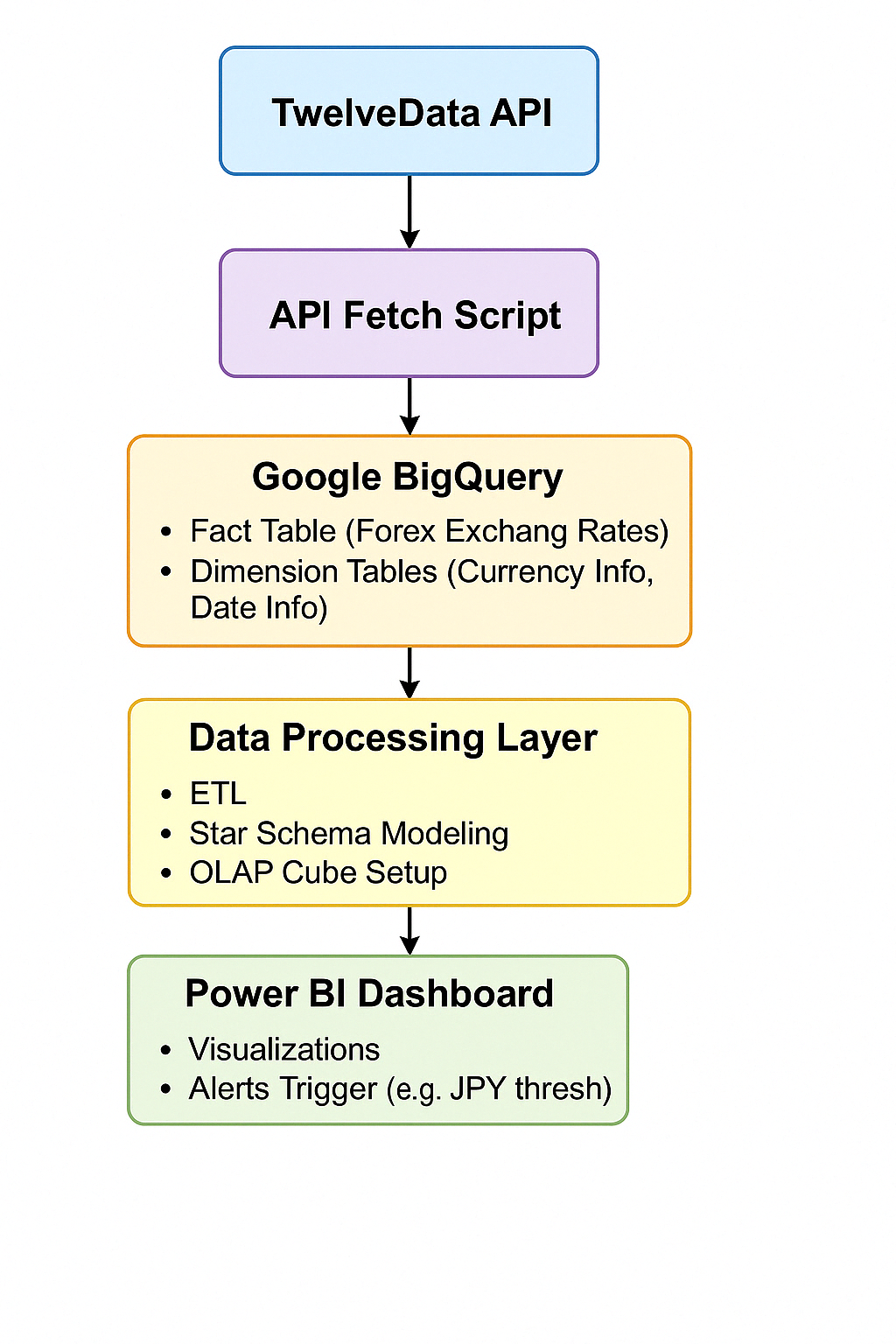
In today's volatile global economy, timely insights into currency exchange rates can significantly impact business strategies and financial decisions.

This project establishes a real-time Forex analytics system focused on major currencies benchmarked against the Indian Rupee (INR).

The solution is designed not only to monitor exchange rates but also to support business decision-making through trend analysis, threshold-based alerts, and historical data modeling.

Built on a scalable, analytical framework, this system enables deeper exploration of forex trends with minimal manual intervention.

# SYSTEM DESIGN AND ARCHITECTURE:



The above diagram represent system that is structured in a layered manner to ensure efficient data flow, storage, analysis, and business decision-making support.

**Layer 1: Data Source**

TwelveData API provides real-time forex rates for multiple currencies (EUR, USD, JPY, AUD, NZD, PLN, GBP, CAD, CHF) against INR. It acts as the primary and reliable source of data for the system.

**Layer 2: Data Ingestion**

An automated API fetch script retrieves live forex data at scheduled intervals. The script parses, cleans, and prepares the data for storage, ensuring consistency and accuracy without manual effort.

**Layer 3: Data Storage**

Google BigQuery serves as the central data warehouse. Data is organized into a star schema with fact and dimension tables, enabling quick access and simplified querying.

**Layer 4: Data Processing and Modeling**

Data from Google BigQuery is loaded into Power Bi using Direct query where ETL processes clean and transform the data to fit analytical needs. An OLAP cube structure is built conceptually to enable operations like drill-down, slice, and dice across currencies and time.

**Layer 5: Data Visualization and Alerts**

Power BI is used to create dynamic dashboards, visualize forex trends, and set real-time alerts (such as thresholds for JPY movements). This provides actionable insights for users in a clear and interactive manner.

The overall architecture ensures a seamless flow from raw data to real-time business intelligence, supporting timely and informed decision-making.

# IMPLEMENTATION DETAILS:

## Data Ingestion

The data ingestion process involves fetching foreign exchange rate data from the TwelveData API and preparing it for storage in Google BigQuery. Here's how it's implemented:

1. **API Integration**:
   * Used Python with the requests library to make API calls to TwelveData
   * Implemented rate limiting and error handling to manage API quotas
2. **Data Transformation Pipeline**:
   * Raw JSON responses are parsed and normalized into structured dataframes
   * Currency pairs are standardized to ensure consistency (e.g., EUR/INR, USD/INR)
   * Timestamps are converted to a unified timezone (UTC) and formatted consistently

## Workflows

1. **Daily Data Collection Workflow**:
   * Triggered at scheduled intervals (daily at market close)
   * Extracts latest exchange rates for all currency pairs against INR
   * Updates both fact and dimension tables
2. **Historical Data Backfill Workflow**:
   * One-time process to populate historical data
   * Handles large data volumes through batched processing
   * Implements checkpointing for recovery from failures

## Alert Engine

1. **Configuration**:
   * Threshold values for JPY/INR are stored in a configuration table
   * Support for both upper and lower bounds
   * Ability to configure multiple thresholds with different severity levels
2. **Monitoring Process**:
   * Dedicated process checks the latest JPY/INR rates against thresholds
   * Implemented using BigQuery scheduled queries that evaluate the latest data
   * Alert messages include contextual information (current rate, threshold crossed, time)

## Data Modeling

1. **Fact Table (fact\_exchange):**
   * Contains Foreign keys to all dimension tables
2. **Dimension Tables:**
   * dim\_date: Time stamp, year, month, day, time\_id
   * dim\_NZD: nzd\_id, nzd\_rate
   * dim\_USD: usd\_id. usd\_rate
   * dime\_AUD: aud\_id, aud\_rate
   * dim\_CAD: cad\_id, cad\_rate
   * dim\_GBP: gbp\_id, gbp\_rate
3. **Schema Design Considerations:**
   * Optimized for query performance with appropriate partitioning
   * Implemented appropriate indexing strategies
   * Denormalized certain attributes for query efficiency

## OLAP Implementation

The OLAP capabilities are implemented through:

1. **Data Cube Creation:**
   * Materialized views in BigQuery that pre-aggregate data
   * Dimensions include: Currency pair, Time period, Rate metrics
   * Hierarchies defined (e.g., Year > Month > Day)
2. **OLAP Operations:**
   * Roll-up: Aggregating from day to month to quarter to year
   * Drill-down: Navigating from year to specific days
   * Slice: Analyzing specific currencies at different time periods
   * Dice: Examining multiple currencies across different time frames
   * Pivot table: Analyzing currencies across various time periods in table format
3. **Query Optimization:**
   * Pre-aggregated common query patterns
   * Implemented caching strategies
   * Created specific materialized views for frequent analytical queries

# CODE DOCUMENTATION:

## Table Creation and Data Ingestion:

from flask import Flask

import requests

import uuid

from datetime import datetime

from google.cloud import bigquery

app = Flask(\_\_name\_\_)

PROJECT\_ID = "bi-project-455515"

DATASET\_ID = "Forex"

CURRENCIES = ["EUR", "GBP", "CAD", "PLN", "USD", "NZD", "JPY", "CHF", "AUD", "RUB", "BRL"]

client = bigquery.Client()

def create\_dim\_table(currency):

table\_id = f"{PROJECT\_ID}.{DATASET\_ID}.dim\_{currency}"

schema = [

bigquery.SchemaField("id", "STRING", mode="REQUIRED"),

bigquery.SchemaField(f"{currency}\_rate", "FLOAT", mode="REQUIRED")

]

table = bigquery.Table(table\_id, schema=schema)

try:

client.get\_table(table\_id)

print(f"Table dim\_{currency} already exists.")

except Exception:

client.create\_table(table)

print(f"Created table: dim\_{currency}")

def create\_date\_table():

table\_id = f"{PROJECT\_ID}.{DATASET\_ID}.dim\_date"

schema = [

bigquery.SchemaField("time\_id", "STRING", mode="REQUIRED"),

bigquery.SchemaField("timestamp", "TIMESTAMP", mode="REQUIRED"),

bigquery.SchemaField("year", "INTEGER", mode="REQUIRED"),

bigquery.SchemaField("month", "INTEGER", mode="REQUIRED"),

bigquery.SchemaField("day", "INTEGER", mode="REQUIRED") ]

table = bigquery.Table(table\_id, schema=schema)

try:

client.get\_table(table\_id)

print("Date table already exists.")

except Exception:

client.create\_table(table)

print("Created table: dim\_date")

def create\_fact\_table():

fields = [bigquery.SchemaField("time\_id", "STRING", mode="REQUIRED")]

for currency in CURRENCIES:

fields.append(bigquery.SchemaField(f"{currency}\_id", "STRING", mode="REQUIRED"))

table\_id = f"{PROJECT\_ID}.{DATASET\_ID}.fact\_exchange"

table = bigquery.Table(table\_id, schema=fields)

try:

client.get\_table(table\_id)

print("Fact table already exists.")

except Exception:

client.create\_table(table)

print("Created fact table.")

def insert\_into\_dim(currency, rate):

table\_id = f"{PROJECT\_ID}.{DATASET\_ID}.dim\_{currency}"

unique\_id = str(uuid.uuid4())

row = {"id": unique\_id, f"{currency}\_rate": rate}

job = client.load\_table\_from\_json([row], table\_id)

job.result()

return unique\_id

def insert\_into\_date(timestamp):

table\_id = f"{PROJECT\_ID}.{DATASET\_ID}.dim\_date"

time\_id = str(uuid.uuid4())

dt = datetime.strptime(timestamp, "%Y-%m-%dT%H:%M:%S.%f")

row = {

"time\_id": time\_id,

"timestamp": timestamp,

"year": dt.year,

"month": dt.month,

"day": dt.day }

job = client.load\_table\_from\_json([row], table\_id)

job.result()

return time\_id

def insert\_into\_fact(time\_id, id\_map):

row = {"time\_id": time\_id}

for currency, cid in id\_map.items():

row[f"{currency}\_id"] = cid

table\_id = f"{PROJECT\_ID}.{DATASET\_ID}.fact\_exchange"

job = client.load\_table\_from\_json([row], table\_id)

job.result()

@app.route("/fetch", methods=["GET"])

def fetch\_and\_store():

url = "http://apilayer.net/api/live" #live historical

params = {

"access\_key": "a45be475cbcc78bd6f262c605c4c6f36",

"currencies": ",".join(CURRENCIES),

"source": "INR",

"format": 1, }

create\_fact\_table()

create\_date\_table()

for currency in CURRENCIES:

create\_dim\_table(currency)

response = requests.get(url, params=params)

if response.status\_code == 200:

data = response.json()

if data.get("success"):

quotes = data["quotes"]

timestamp = datetime.utcnow().isoformat()

time\_id = insert\_into\_date(timestamp)

id\_map = {}

for key, rate in quotes.items():

currency = key.replace("INR", "")

if currency in CURRENCIES:

unique\_id = insert\_into\_dim(currency, rate)

if unique\_id:

id\_map[currency] = unique\_id

insert\_into\_fact(time\_id, id\_map)

return "✅ Exchange rates fetched and stored."

else:

return f"API error: {data.get('error', {}).get('info', 'Unknown error')}"

else:

return f"Failed to fetch data: Status Code {response.status\_code}"

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

app.run(debug=True)

## Pivot table script :

pivot\_table = df.pivot\_table(

    index='year',

    columns='month',

    values=['JPY\_rate', 'NZD\_rate', 'PLN\_rate'],

    aggfunc='mean')

pivot\_table = pivot\_table.round(4)

pivot\_table.columns = [f'{curr},{str(month).zfill(2)}' for curr, month in pivot\_table.columns]

pivot\_table = pivot\_table.reset\_index()

table = ax.table(

    cellText=pivot\_table.values,

    colLabels=pivot\_table.columns,

    cellLoc='center',

    loc='center’)

table.auto\_set\_font\_size(False)

table.set\_fontsize(8)

table.scale(1.2, 1.2)

plt.show()

## Date Cube:

pivot\_table = pd.pivot\_table(

    df,

    index=['year', 'month'],

    values=['BRL\_rate', 'AUD\_rate', 'EUR\_rate', 'GBP\_rate'],

    aggfunc='mean'

).reset\_index()

pivot\_table['time'] = pivot\_table['year'] \* 100 + pivot\_table['month']

x\_labels = pivot\_table['time']

y\_labels = ['BRL\_rate', 'AUD\_rate', 'EUR\_rate', 'GBP\_rate']

fig = plt.figure(figsize=(12, 8))

ax = fig.add\_subplot(111, projection='3d')

\_x = np.arange(len(x\_labels))

\_y = np.arange(len(y\_labels))

\_xx, \_yy = np.meshgrid(\_x, \_y)

x, y = \_xx.ravel(), \_yy.ravel()

top = pivot\_table[y\_labels].values.T.ravel()

bottom = np.zeros\_like(top)

width = depth = 0.5

ax.bar3d(x, y, bottom, width, depth, top, shade=True)

plt.tight\_layout()

plt.show()

## Alerts:

df['timestamp'] = pd.to\_datetime(df['timestamp'])

threshold\_jpy = 1.75  # Example threshold for JPY rate

df['JPY\_alert'] = df['JPY\_rate'] > threshold\_jpy

fig, ax = plt.subplots(figsize=(12, 6))  # Adjusted size for better clarity

ax.plot(df['timestamp'], df['JPY\_rate'], label='JPY Rate', color='blue')

ax.scatter(df[df['JPY\_alert']]['timestamp'], df[df['JPY\_alert']]['JPY\_rate'], color='red', label='JPY Threshold Crossed', zorder=5)

plt.xticks(rotation=45, ha='right')

plt.title('JPY Rate with Alerts for Threshold Crossings (Month-Day)')

plt.xlabel('Month-Day')

plt.ylabel('JPY Rate')

plt.legend()

plt.tight\_layout()

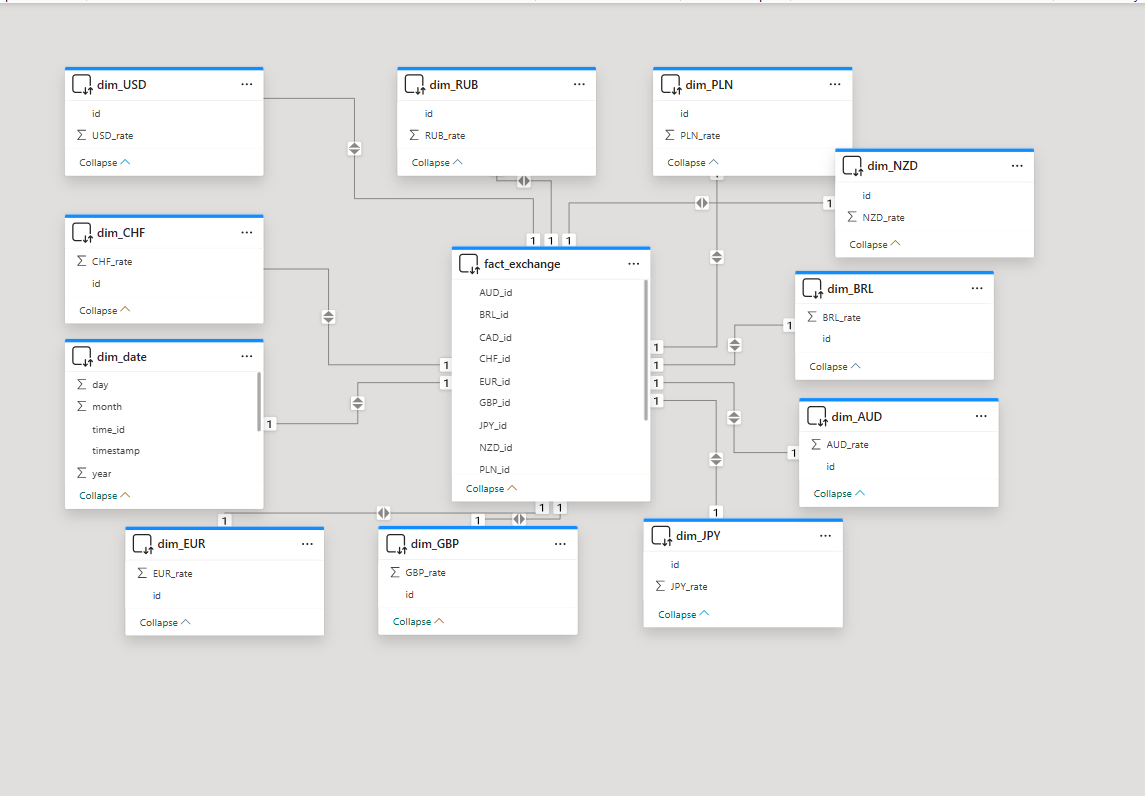
plt.show()

# TESTING AND EVALUATION RESULT:

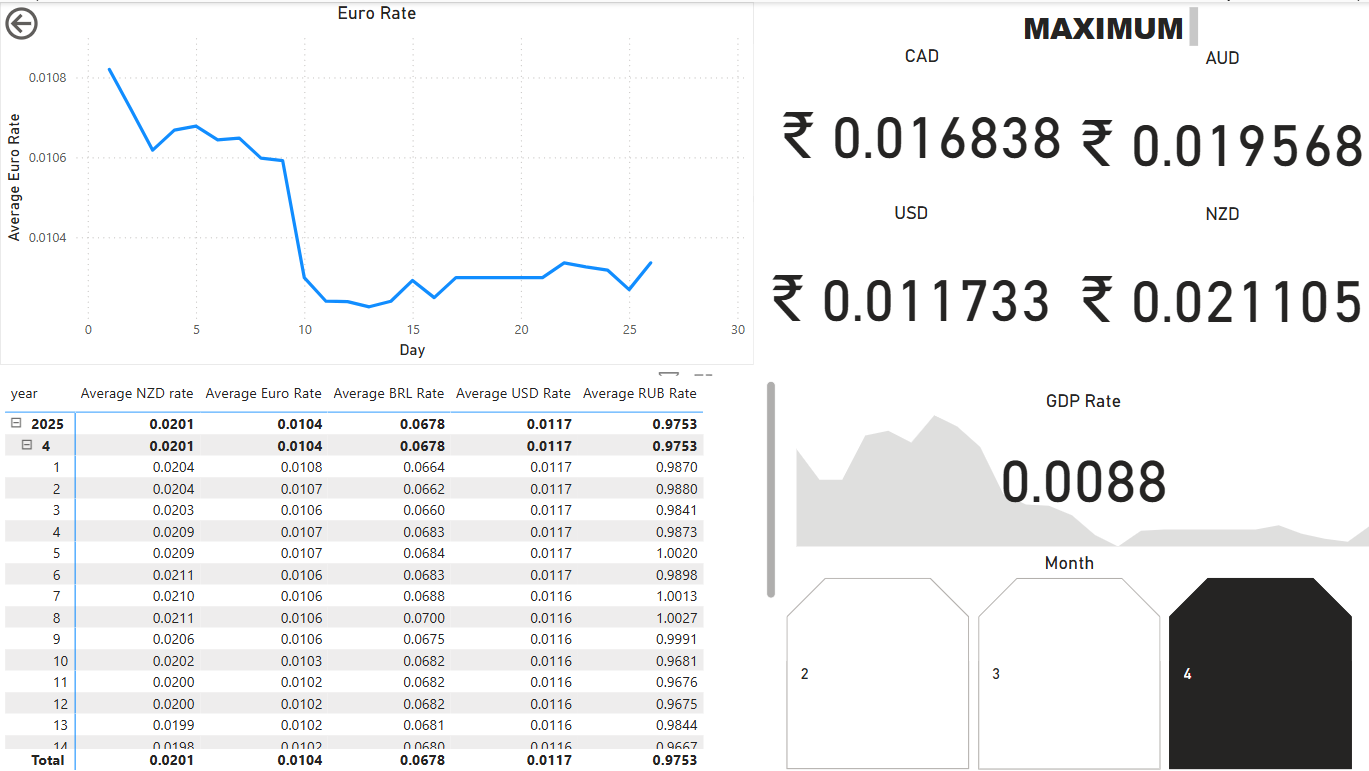
|  |  |  |
| --- | --- | --- |
| **Area** | **Test** | **Result** |
| |  | | --- | | API Connectivity |  |  | | --- | |  | | Success | Data Retrieved in 30 seconds |
| |  | | --- | | Data Loading in Bigquery |  |  | | --- | |  | | Success | Data loaded in 30 seconds |
| |  | | --- | | Star Schema Integrity |  |  | | --- | |  | | Success | 1 to 1 Connection established between schemas |
| Dynamic Data Updating | Success | Data is refreshed 30 seconds once automatically |
| |  | | --- | | Alert Accuracy |  |  | | --- | |  | | Success | JPY alert triggered above threshold |

# OUTPUT:

## Star Schema:

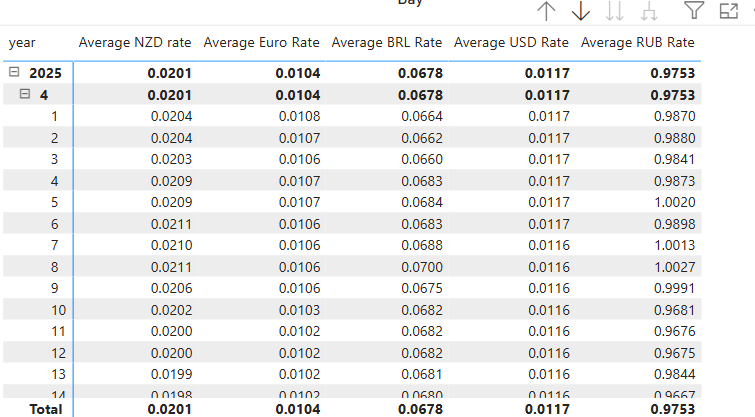


## Home Dashboard:

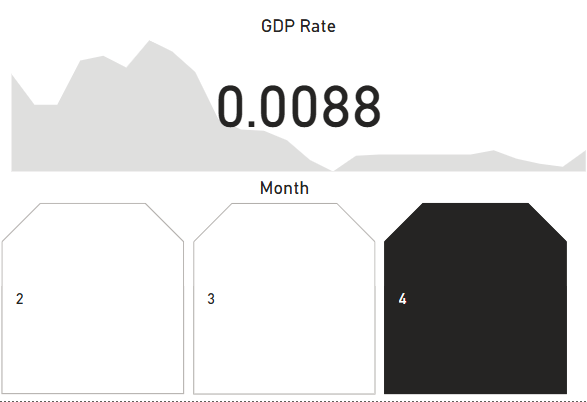


## OLAP Operations

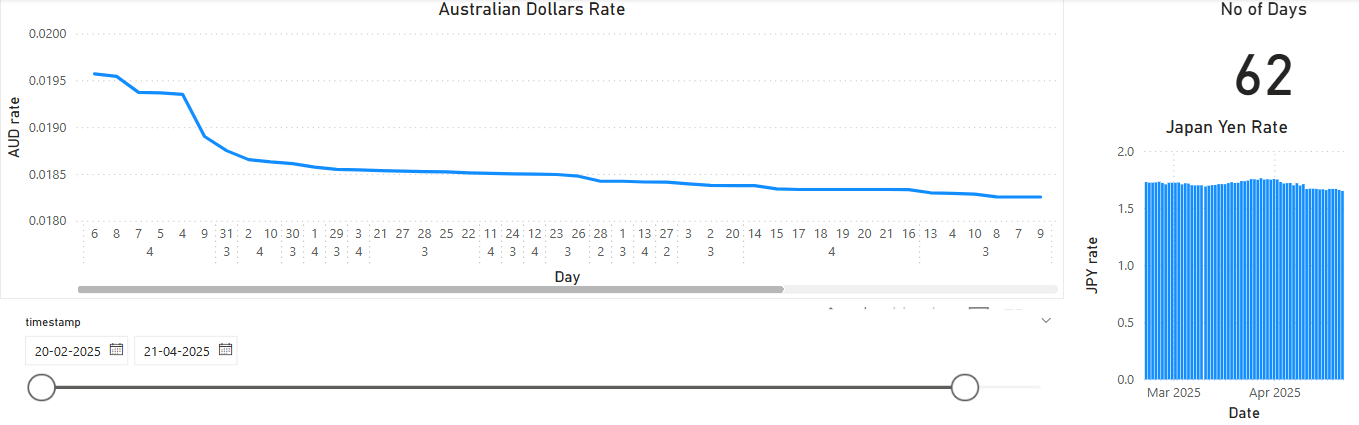
1. Roll up and Drill Down

****

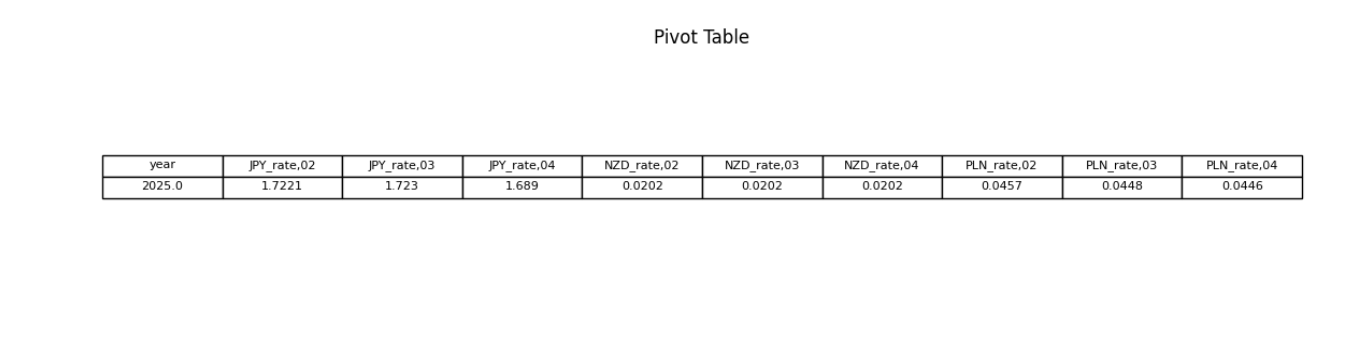
1. Slicing



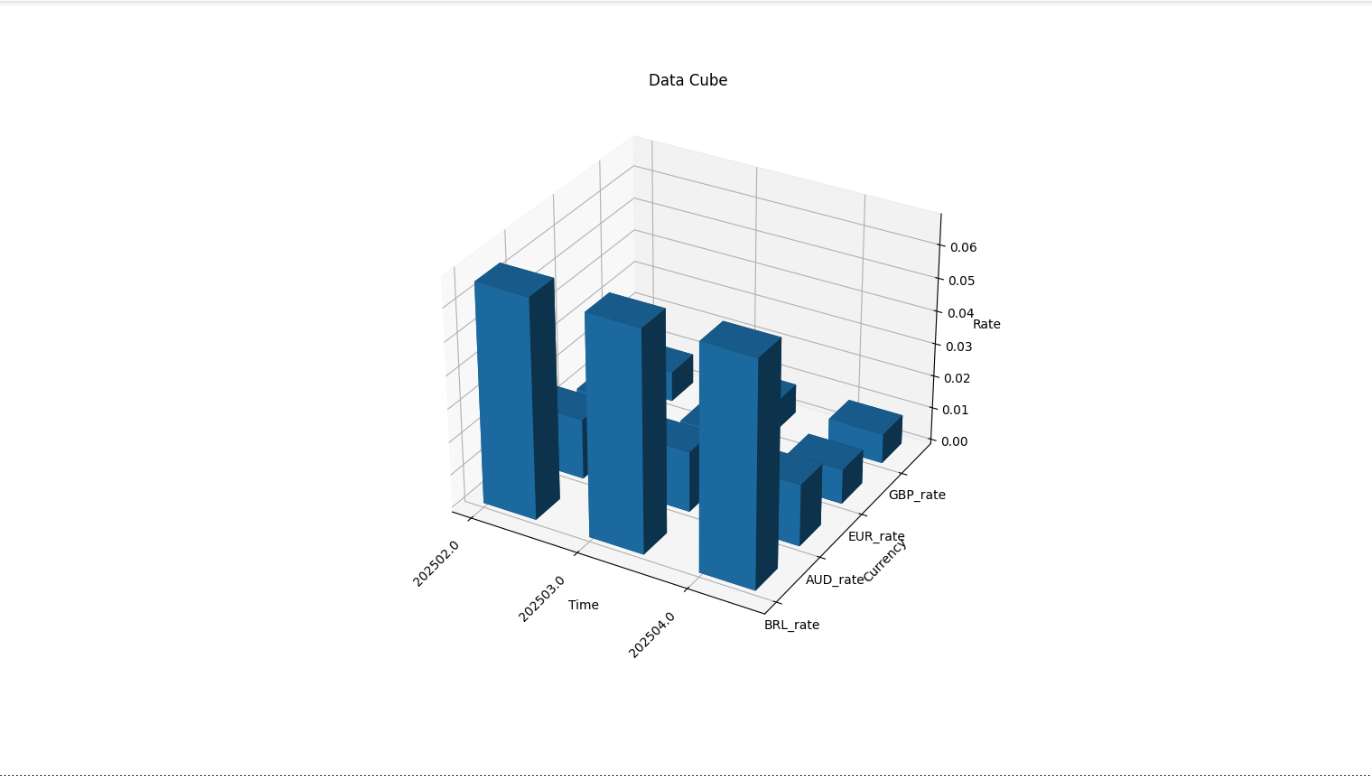
1. Dicing

****

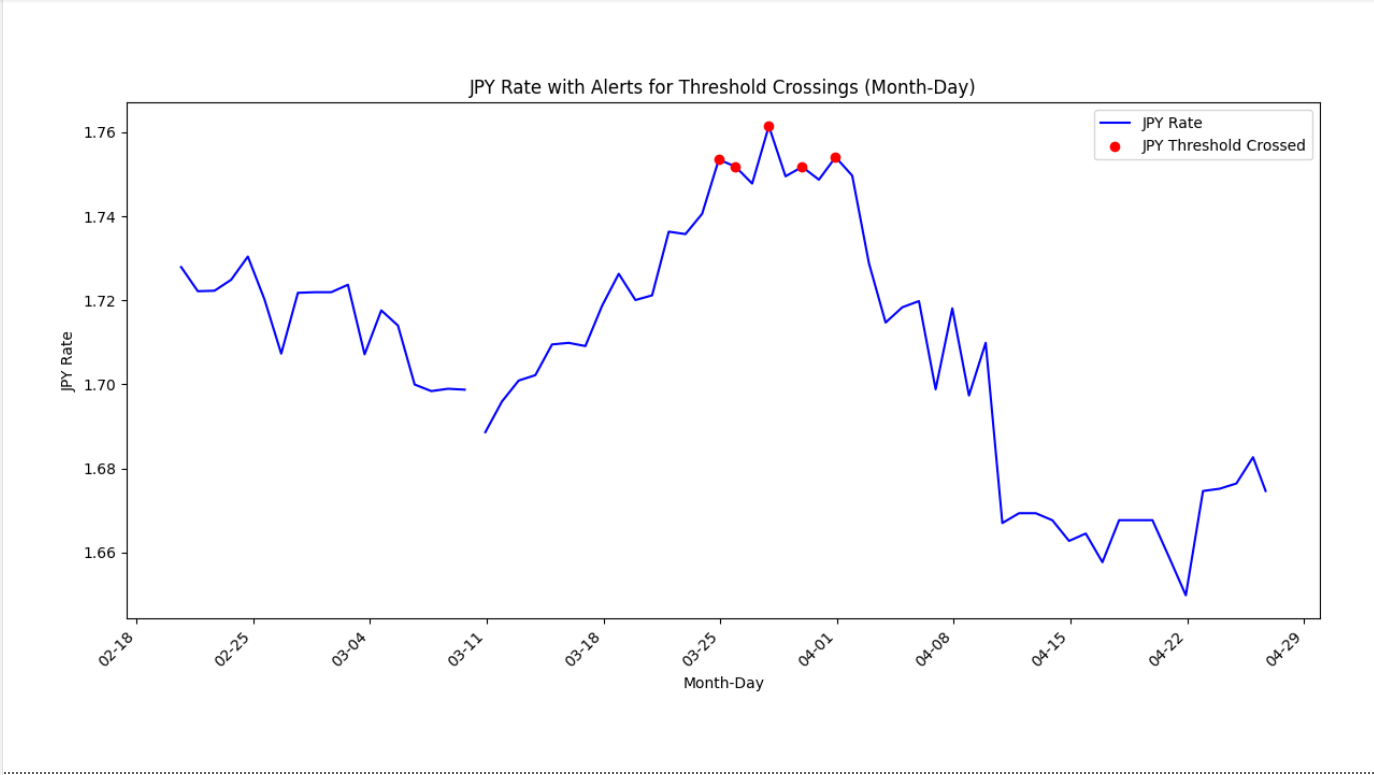
1. Pivot Table

****

## Data Cube:

****

## Alerts:

****