Unlocking the Power of Big Data

Professor: Hachem Sfar

Student Name: Chun-Han Chueh (Victoria)

Student ID: 22026614

GitHub link: https://github.com/VictoriaChueh/Unlocking-the-Power-of-Big-Data_Final.git

1. Data Platform Used

Databricks

2. Technologies

Databricks (using Delta tables, notebook, SQL, and Databricks dashboards...)

3. Data Source

3.1 Frankfurter API

- **URL:** https://api.frankfurter.app/
- **Provides:** Exchange rate data (daily rates of EUR against other currencies)
- **Table:** fx_rates_eur
- **Columns:** date, base, currency, rate
- **Purpose:** Time series analysis (e.g., monthly average exchange rates, rate changes)

3.2 REST Countries API

- **URL:** https://restcountries.com/
- **Provides:** Basic country information (population, region, currency)
- **Table:** countries_currency
- **Columns:** country, region, population, currency, currency_name
- **Purpose:** Analyze population distribution, regional comparisons, and currency-country relationships

4. Core Technical Requirements

4.1 Data Ingestion

API Source 1

```
# 4.1 Data Ingestion
# Import required libraries
import requests
import pandas as pd

# Parameterize the date range
start_date = "2024-01-01"
end_date   = "2024-12-31"
base_ccy   = "EUR"

# Build API URL
url =
f"https://api.frankfurter.app/{start_date}..{end_date}?from={base_cc
y}"

# Call the API
resp = requests.get(url, timeout=30)
resp.raise_for_status()
payload = resp.json()

# Flatten JSON to rows: one row per (date, currency, rate)
rows = []
for d, rate_map in (payload.get("rates") or {}).items():
    for ccy, rate in rate_map.items():
        rows.append({
            "date": d,           # string YYYY-MM-DD
            "base": payload.get("base"),
            "currency": ccy,
            "rate": float(rate)
        })

# Convert to pandas then to Spark DataFrame
pdf_fx = pd.DataFrame(rows)
df_fx = spark.createDataFrame(pdf_fx)
```

```
# Quick checks
print(f"fx rows: {df_fx.count()}")
df_fx.printSchema()
df_fx.show(5)
```

API Source 2

```
# 4.1 Data Ingestion
# Import libraries
import requests
import pandas as pd

# API endpoint: limit to required fields for efficiency
countries_url =
"https://restcountries.com/v3.1/all?fields=name,population,region,cc
a2,currencies"

# Call the API
resp = requests.get(countries_url, timeout=60)
resp.raise_for_status()
countries = resp.json()

# Normalize to tabular rows: expand one row per currency per country
rows = []
for c in countries:
    name = (c.get("name") or {}).get("common")
    population = c.get("population")
    region = c.get("region")
    cca2 = c.get("cca2")
    currencies = c.get("currencies") or {}

    # If a country has currencies, expand each; else keep one row
with None currency
    if currencies:
        for code, info in currencies.items():
            rows.append({
                "country": name,
                "cca2": cca2,
                "region": region,
```

```
            "population": population,
            "currency": code,
            "currency_name": (info or {}).get("name"),
        })
    else:
        rows.append({
            "country": name,
            "cca2": cca2,
            "region": region,
            "population": population,
            "currency": None,
            "currency_name": None,
        })

# Convert to pandas then to Spark DataFrame
pdf_cty = pd.DataFrame(rows)
df_cty = spark.createDataFrame(pdf_cty)

# Quick checks
print(f"countries rows: {df_cty.count()}")
df_cty.printSchema()
df_cty.show(5)
```

4.2 Data Cleaning, Preparation and analysis

```
# 4.2 Data Cleaning, Preparation and analysis
# Import Spark functions
from pyspark.sql.functions import col

# Clean FX data: drop nulls and cast rate
df_fx_clean = (
    df_fx
    .dropna(subset=["date", "currency", "rate"])
    .withColumn("rate", col("rate").cast("double"))
)

# Clean country data: ensure core columns and cast population
df_cty_clean = (
    df_cty
    .dropna(subset=["country"])              # core identity column
    .withColumn("population", col("population").cast("long"))
)

# Sanity check
print("fx_clean rows:", df_fx_clean.count())
print("cty_clean rows:", df_cty_clean.count())
```

4.3 Data Storage

```
# 4.3 Data Storage
# Persist cleaned dataframes as Delta managed tables
df_fx_clean.write.mode("overwrite").format("delta").saveAsTable("fx_
rates_eur")
df_cty_clean.write.mode("overwrite").format("delta").saveAsTable("co
untries_currency")

print("Tables saved: fx_rates_eur, countries_currency")
```

```
# 4.3 Data Storage
# Load tables back to verify
fx_tbl = spark.table("fx_rates_eur")
cty_tbl = spark.table("countries_currency")

fx_tbl.show(3)
cty_tbl.show(3)
```

4.4 SQL Analysis

Query 1: Monthly Average EUR→USD Exchange Rate

```
# 4.4 SQL Analysis / Query 1: Monthly Average EUR→USD Exchange Rate

%sql

SELECT
  date_format(to_date(date), 'yyyy-MM') AS month,
  AVG(rate) AS avg_rate_usd
FROM fx_rates_eur
WHERE currency = 'USD'
GROUP BY date_format(to_date(date), 'yyyy-MM')
ORDER BY month;
```

Query 2: Top 10 Most Populous Countries by Region

```
# 4.4 SQL Analysis / Query 2: Top 10 Most Populous Countries by
Region

%sql
WITH ranked AS (
  SELECT
    region,
    country,
    population,
    ROW_NUMBER() OVER (PARTITION BY region ORDER BY population DESC)
AS rn
  FROM countries_currency
  WHERE population IS NOT NULL
)
SELECT
  region,
  country,
  population
FROM ranked
WHERE rn <= 10
ORDER BY region, population DESC;
```

Query 3: Month-over-Month Change of Monthly Average EUR→USD Exchange Rate

```
# 4.4 SQL Analysis / Query 3: Month-over-Month Change of Monthly
Average EUR→USD Exchange Rate

%sql

WITH m AS (
  SELECT
    date_format(to_date(date), 'yyyy-MM') AS month,
    AVG(rate) AS avg_usd
  FROM fx_rates_eur
  WHERE currency = 'USD'
  GROUP BY date_format(to_date(date), 'yyyy-MM')
)
SELECT
  month,
  avg_usd,
  avg_usd - LAG(avg_usd) OVER (ORDER BY month) AS m2m_change
FROM m
ORDER BY month;
```

Query 4: Link Each Country's Currency to Its Average Exchange Rate

```
# 4.4 SQL Analysis / Query 4: Link Each Country's Currency to Its
Average Exchange Rate

%sql

WITH avg_fx AS (
  SELECT
    currency,
    AVG(rate) AS avg_rate_vs_eur
  FROM fx_rates_eur
  GROUP BY currency
)
SELECT
  c.region,
```

```
  c.country,
  c.currency,
  a.avg_rate_vs_eur
FROM countries_currency c
JOIN avg_fx a
  ON c.currency = a.currency
WHERE c.currency IS NOT NULL
ORDER BY c.region, c.country;
```

4.5 Notebook Analysis

```
# 4.5 Notebook Analysis
# Ensure using the correct schema
spark.sql("USE SCHEMA default")

# Load Delta tables created in previous steps
df_fx = spark.table("fx_rates_eur")
df_cty = spark.table("countries_currency")

# Inspect schemas and a few sample rows
print("=== Schemas ===")
df_fx.printSchema()     # FX: date (string), base, currency, rate
(double)
df_cty.printSchema()    # Countries: country, cca2, region,
population (long), currency, currency_name

print("=== Samples ===")
df_fx.show(5)
df_cty.show(5)
```

Descriptive Stats

```
# 4.5 Notebook Analysis / Descriptive Stats
# Descriptive statistics for FX rates (numeric column)
print("=== FX rates descriptive stats ===")
df_fx.describe(["rate"]).show()

# Descriptive statistics for population (numeric column)
print("=== Countries population descriptive stats ===")
df_cty.describe(["population"]).show()
```

GroupBy + Aggregation

```
#4.5 Notebook Analysis / GroupBy + Aggregation

from pyspark.sql.functions import avg, sum, col

# 1) Average FX rate per currency (descending)
print("=== Average FX rate per currency ===")
df_fx.groupBy("currency").agg(avg("rate").alias("avg_rate")) \
```

```
    .orderBy(col("avg_rate").desc()) \
    .show(20, truncate=False)

# 2) Total population per region (descending)
print("=== Total population per region ===")
df_cty.groupBy("region").agg(sum("population").alias("total_populati
on")) \
    .orderBy(col("total_population").desc()) \
    .show(truncate=False)
```

JOIN

```
#4.5 Notebook Analysis / JOIN

from pyspark.sql.functions import avg, col

# Join countries and FX by currency; keep only rows with currency
info on both sides
df_join = df_cty.join(df_fx, on="currency", how="inner")

# Compute average annual FX rate vs EUR per country
df_country_avg_fx = (
   df_join.groupBy("country", "region", "currency")
         .agg(avg("rate").alias("avg_rate_vs_eur"))
         .orderBy(col("avg_rate_vs_eur").desc())
)

print("=== Country-level average FX vs EUR (2024) ===")
df_country_avg_fx.show(20, truncate=False)
```

Time Series Processing

```
#4.5 Notebook Analysis / Time Series Processing: Convert Date to
DATE and Calculate Monthly Average Exchange Rate

from pyspark.sql.functions import to_date, date_format

# Cast date string to DATE type, then extract 'YYYY-MM' as month
df_fx_typed = (
   df_fx
```

```
    .withColumn("date_dt", to_date(col("date")))        # convert
string to date
    .withColumn("month", date_format(col("date_dt"), "yyyy-MM"))
)

# Monthly average for USD
df_fx_usd_monthly = (
    df_fx_typed.filter(col("currency") == "USD")
               .groupBy("month")
               .agg(avg("rate").alias("avg_rate_usd"))
               .orderBy("month")
)

print("=== Monthly average EUR→USD (2024) ===")
df_fx_usd_monthly.show(12, truncate=False)
```

Notebook Visualization

```
#4.5 Notebook Analysis / Notebook Visualization

# Visualize monthly EUR→USD averages (select line chart in the UI)
display(df_fx_usd_monthly)

# Visualize total population per region (select bar chart in the UI)
from pyspark.sql.functions import sum
df_region_pop =
df_cty.groupBy("region").agg(sum("population").alias("total_populati
on")) \
                  .orderBy(col("total_population").desc())
display(df_region_pop)
```

Save Analysis Results as a Delta Table

```
#4.5 Notebook Analysis / Save Analysis Results as a Delta Table

# Persist summary tables for dashboard consumption
df_fx_usd_monthly.write.mode("overwrite").format("delta").saveAsTabl
e("fx_usd_monthly_avg")
```

```
df_region_pop.write.mode("overwrite").format("delta").saveAsTable("r
egion_population_total")

print("Saved summary tables: fx_usd_monthly_avg,
region_population_total")
```

Parameterize Monthly Average Exchange Rate

```
# #4.5 Notebook Analysis / Parameterize Monthly Average Exchange
Rate (Dynamic Currency)

# Create a dropdown widget for currency selection
dbutils.widgets.removeAll()
dbutils.widgets.text("p_currency", "USD", "Currency (e.g., USD, JPY,
GBP)")

p_currency = dbutils.widgets.get("p_currency")

from pyspark.sql.functions import to_date, date_format, avg, col

df_fx_typed = df_fx.withColumn("date_dt", to_date(col("date"))) \
                .withColumn("month", date_format(col("date_dt"),
"yyyy-MM"))

df_fx_monthly_param = (
   df_fx_typed.filter(col("currency") == p_currency)
              .groupBy("month")
              .agg(avg("rate").alias("avg_rate"))
              .orderBy("month")
)

print(f"=== Monthly average EUR→{p_currency} (2024) ===")
display(df_fx_monthly_param)
```

4.6 Dashboard

SQL Editor

1. Monthly Average Exchange Rate (Line Chart Data)

```
-- 1) Monthly Average Exchange Rate (Line Chart Data)
-- Query/Table Name: q_fx_monthly_avg

WITH params AS (
  SELECT
    -- Fallback to safe boundaries when parameters are empty strings
or NULL
    COALESCE(NULLIF(:start_date, ''), '1900-01-01') AS start_date_s,
    COALESCE(NULLIF(:end_date,   ''), '2999-12-31') AS end_date_s,
    COALESCE(NULLIF(:ccy,        ''), 'USD')        AS ccy_s
)
SELECT
  date_format(to_date(f.date), 'yyyy-MM') AS month,
  AVG(f.rate) AS avg_rate
FROM fx_rates_eur f
CROSS JOIN params p
WHERE f.currency = p.ccy_s
  AND to_date(f.date) BETWEEN to_date(p.start_date_s) AND
to_date(p.end_date_s)
GROUP BY date_format(to_date(f.date), 'yyyy-MM')
ORDER BY month;
ion, c.country;
```

2. Month-over-Month Change of Monthly Average Exchange Rate (Table)

```
-- 2) Month-over-Month Change of Monthly Average Exchange Rate
(Table)
-- Query/Table Name: q_fx_mom_change

WITH params AS (
  SELECT
    COALESCE(NULLIF(:start_date, ''), '1900-01-01') AS start_date_s,
    COALESCE(NULLIF(:end_date,   ''), '2999-12-31') AS end_date_s,
    COALESCE(NULLIF(:ccy,        ''), 'USD')        AS ccy_s
),
m AS (
```

```
  SELECT
    date_format(to_date(f.date), 'yyyy-MM') AS month,
    AVG(f.rate) AS avg_rate
  FROM fx_rates_eur f
  CROSS JOIN params p
  WHERE f.currency = p.ccy_s
    AND to_date(f.date) BETWEEN to_date(p.start_date_s) AND
to_date(p.end_date_s)
  GROUP BY date_format(to_date(f.date), 'yyyy-MM')
)
SELECT
  month,
  avg_rate,
  avg_rate - LAG(avg_rate) OVER (ORDER BY month) AS mom_change
FROM m
ORDER BY month;
```

3. Total Population by Region (Bar Chart Data)

```
-- 3) Total Population by Region (Bar Chart Data)
-- Query/Table Name: q_population_by_region

SELECT
  region,
  SUM(population) AS total_population
FROM countries_currency
WHERE population IS NOT NULL
GROUP BY region
ORDER BY total_population DESC;
```

4. Link Each Country's Currency to Its Average Rate Against EUR (Bubble Chart Data)

```
-- 4) Link Each Country's Currency to Its Average Rate Against EUR
(Bubble Chart Data)
-- Query/Table Name: q_country_currency_avg_rate

WITH avg_fx AS (
  SELECT
```

```
    currency,
    AVG(rate) AS avg_rate_vs_eur
  FROM fx_rates_eur
  GROUP BY currency
)
SELECT
  c.region,
  c.country,
  c.population,
  c.currency,
  a.avg_rate_vs_eur
FROM countries_currency c
JOIN avg_fx a
  ON c.currency = a.currency
WHERE c.currency IS NOT NULL
ORDER BY c.region, c.country;
```

Dataset

1. Monthly Average Exchange Rate (Line Chart Data)

```
WITH params AS (
  SELECT
    COALESCE(NULLIF(:start_date, ''), '1900-01-01') AS start_date_s,
    COALESCE(NULLIF(:end_date,   ''), '2999-12-31') AS end_date_s,
    COALESCE(NULLIF(:ccy,        ''), 'USD')        AS ccy_s
)
SELECT
  date_format(to_date(f.date), 'yyyy-MM') AS month,
  AVG(f.rate) AS avg_rate
FROM fx_rates_eur f
CROSS JOIN params p
WHERE f.currency = p.ccy_s
  AND to_date(f.date) BETWEEN to_date(p.start_date_s) AND
to_date(p.end_date_s)
GROUP BY date_format(to_date(f.date), 'yyyy-MM')
ORDER BY month;
```

2. Month-over-Month Change of Monthly Average Exchange Rate (Table)

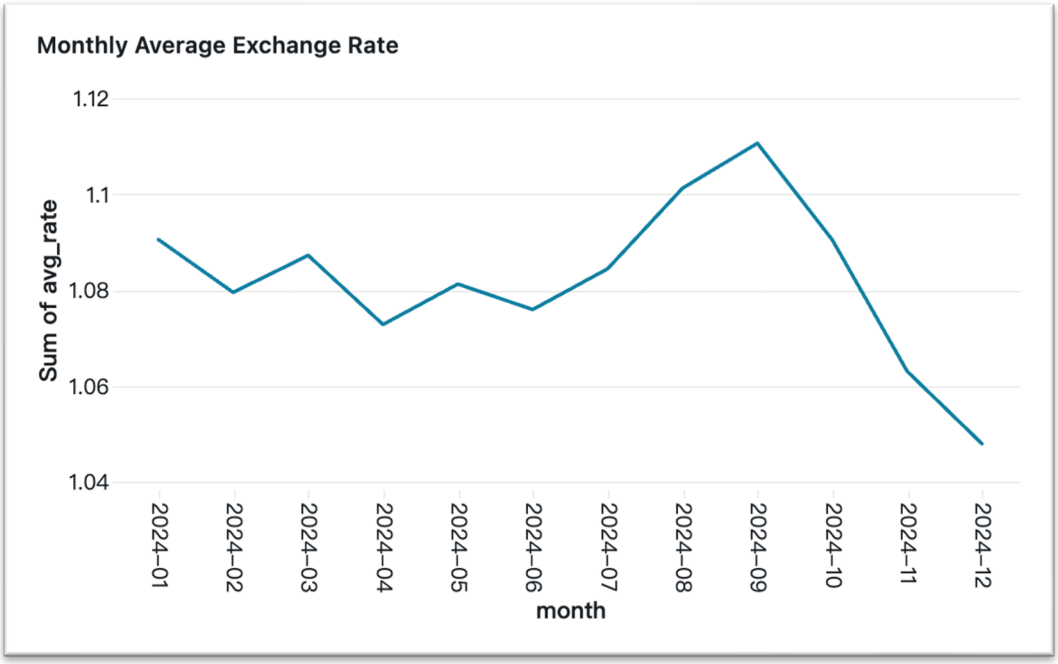
```
WITH params AS (
  SELECT
    COALESCE(NULLIF(:start_date, ''), '1900-01-01') AS start_date_s,
    COALESCE(NULLIF(:end_date,   ''), '2999-12-31') AS end_date_s,
    COALESCE(NULLIF(:ccy,        ''), 'USD')        AS ccy_s
),
m AS (
  SELECT
    date_format(to_date(f.date), 'yyyy-MM') AS month,
    AVG(f.rate) AS avg_rate
  FROM fx_rates_eur f
  CROSS JOIN params p
  WHERE f.currency = p.ccy_s
    AND to_date(f.date) BETWEEN to_date(p.start_date_s) AND
to_date(p.end_date_s)
  GROUP BY date_format(to_date(f.date), 'yyyy-MM')
)
SELECT
  month,
  avg_rate,
  avg_rate - LAG(avg_rate) OVER (ORDER BY month) AS mom_change
FROM m
ORDER BY month;
```

3. Total Population by Region (Bar Chart Data)

```
SELECT
  region,
  SUM(population) AS total_population
FROM countries_currency
WHERE population IS NOT NULL
GROUP BY region
ORDER BY total_population DESC;
```

4. *Link Each Country's Currency to Its Average Rate Against EUR (Bubble Chart Data)*

```
WITH avg_fx AS (
  SELECT
    currency,
    AVG(rate) AS avg_rate_vs_eur
  FROM fx_rates_eur
  WHERE currency IS NOT NULL
  GROUP BY currency
)
SELECT
  c.region,
  c.country,
  c.population,
  c.currency,
  a.avg_rate_vs_eur
FROM countries_currency c
JOIN avg_fx a
  ON c.currency = a.currency
WHERE c.currency IS NOT NULL
  AND c.population IS NOT NULL
ORDER BY c.region, c.country;
```

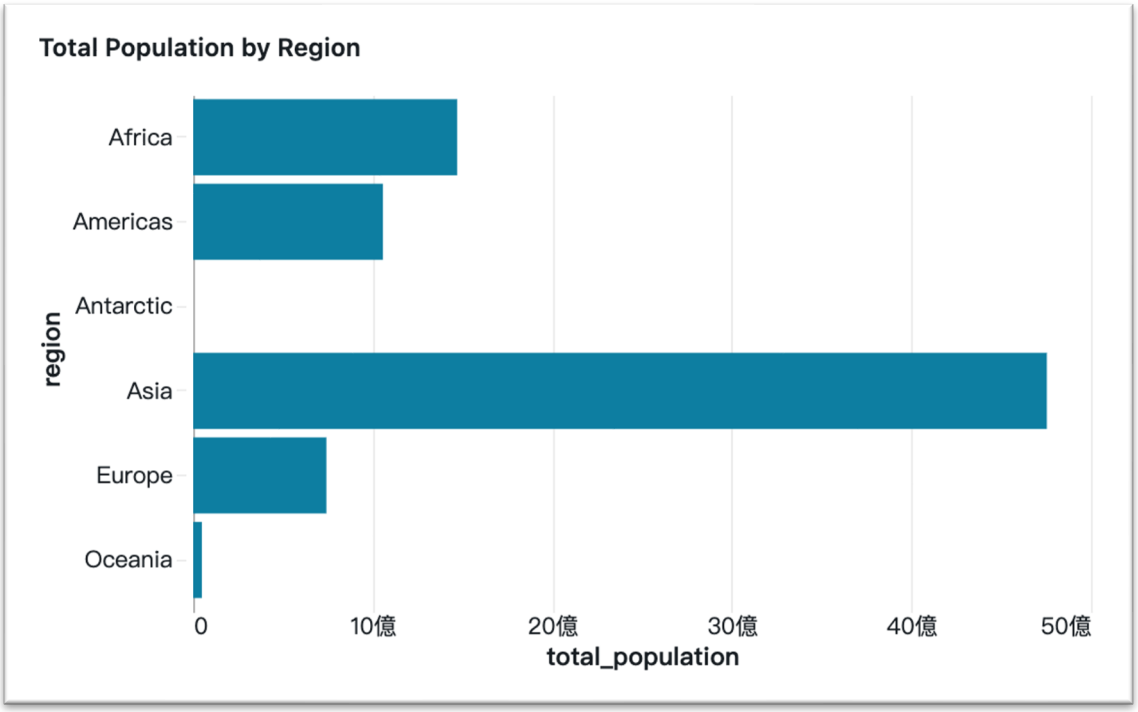


Dashboard 1_Monthly Average Exchange Rate

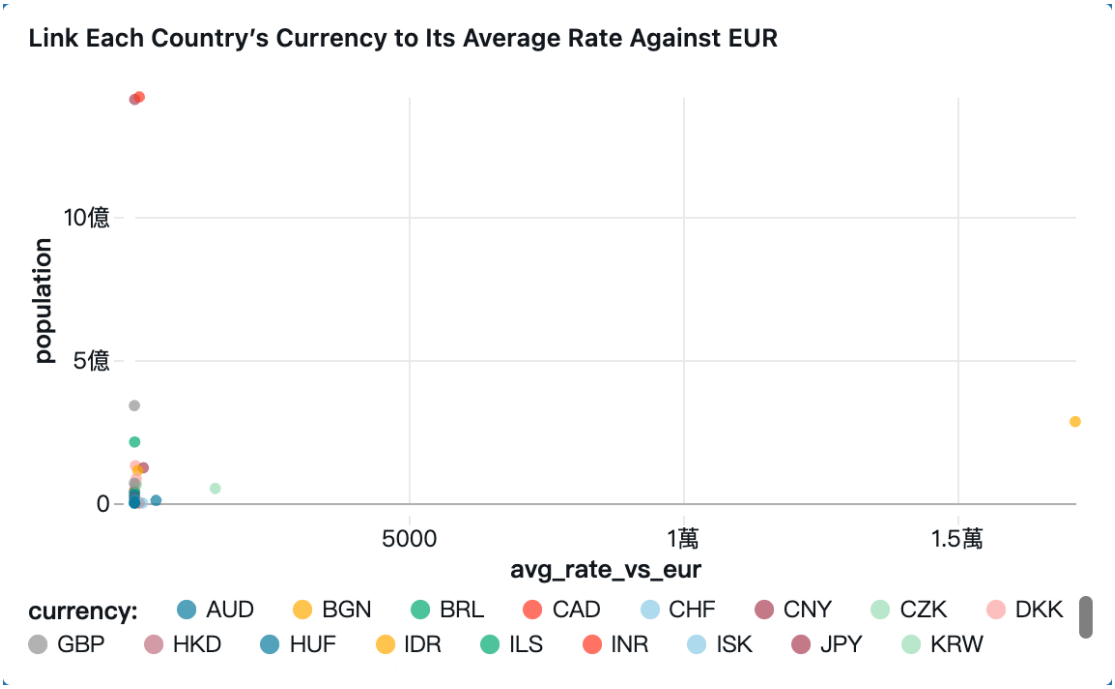
Month-over-Month Change of Monthly Average Exchange Rate

month	avg_rate	mom_change
2023-12	1.11	*null*
2024-01	1.09	-0.01
2024-02	1.08	-0.01
2024-03	1.09	0.01
2024-04	1.07	-0.01
2024-05	1.08	0.01
2024-06	1.08	-0.01
2024-07	1.08	0.01
2024-08	1.10	0.02
2024-09	1.11	0.01
2024-10	1.09	-0.02
2024-11	1.06	-0.03
2024-12	1.05	-0.02

Dashboard 2_Month-over-Month Change of Monthly Average Exchange Rate



Dashboard 3_Total Population by Region



Dashboard 4_Link Each Country's Currency to Its Average Rate Against EUR