BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Selama menjalani kegiatan magang di PT Bank Jago Tbk, Penulis ditempatkan dalam tim data dengan posisi sebagai data engineer intern. Dalam peran tersebut, Penulis memiliki tanggung jawab untuk mendukung tim dalam pengembangan dan pemeliharaan aplikasi Unit Economics Simulator. Tanggung jawab tersebut meliputi:

- Membantu dalam pengolahan dan pembersihan data untuk memastikan kualitas dan integritas data yang digunakan oleh aplikasi.
- Mendukung implementasi debugging dan perbaikan kode yang terkait dengan aplikasi Unit Economics Simulator untuk meningkatkan kinerja dan fungsionalitas.
- Melakukan pengujian unit (unit testing) dan integrasi untuk memastikan bahwa setiap komponen aplikasi berjalan dengan baik sebelum diterapkan ke sistem yang lebih besar.
- Membantu dalam proses deployment aplikasi dan pemantauan kinerja aplikasi pasca-deployment untuk mendukung kestabilan aplikasi dalam operasional sehari-hari.

3.2 Tugas yang Dilakukan

Daftar tugas yang diberikan untuk mendukung pengembangan aplikasi Unit Economics Simulator dapat dilihat pada 3.1

Tabel 3.1. Jadwal Tugas Magang

Tugas	**Minggu**
Onboarding penugasan oleh *supervisor*	1
Pembelajaran *stack* dan *tools* terkait Unit Economics Simulator	2
Pembelajaran *stack* dan *tools* terkait Unit Economics Simulator	3
Pengembangan Unit Economics Simulator	4
Pengembangan Unit Economics Simulator	5
Pengembangan Unit Economics Simulator	6
Pengembangan Unit Economics Simulator	7
Deployment Unit Economics Simulator	8
Deployment Unit Economics Simulator	9
Deployment Unit Economics Simulator	10
Pengembangan Revisi Unit Economics Simulator	11
Pengembangan Revisi Unit Economics Simulator	12

3.3 Uraian Pelaksanaan Magang

Selama pelaksanaan kerja magang, terdapat beberapa kendala yang dihadapi dan solusi yang diimplementasikan.

Seluruh data yang disajikan dalam bagian ini bersifat umum dan tidak mencakup informasi yang bersifat rahasia atau sensitif milik perusahaan tempat magang dilakukan. Data yang bersifat pribadi atau strategis tidak akan ditampilkan demi menjaga kerahasiaan dan keamanan informasi sesuai dengan ketentuan yang berlaku.

3.3.1 Mempelajari Konsep *Unit Economics*

Unit economics merupakan sebuah pendekatan yang digunakan untuk menganalisis probabilitas keuntungan dari bisnis dari level satuan terkecil seperti *Customer*, *Products*, ataupun *Transactions* [6]. Fokus utama dari *unit economics* adalah untuk memperkirakan performa finansial dengan cara mengidentifikasi pola biaya dan keuntungan yang muncul. Dengan mengerti prinsip *unit economics*, perusahaan dapat mengembangkan strategi jangka panjang, yang berdasar pada

hasil perkiraan nilai relatif biaya operational pada *unit economics*.

Dalam mempelajari *unit economics*, terdapat beberapa istilah penting yang menjadi fokus utama, antara lain:

- **Lifetime Value (LTV)**
  Lifetime value pelanggan adalah jumlah rata-rata uang yang diperoleh dari seorang pelanggan selama masa hubungannya dengan perusahaan.

- **Customer Acquisition Cost (CAC)**
  Customer acquisition cost adalah jumlah total uang yang dikeluarkan untuk memperoleh satu pelanggan melalui kegiatan pemasaran dan penjualan. Jika pengeluaran terlalu sedikit, perusahaan tidak akan mendapatkan pelanggan baru. Jika pengeluaran terlalu banyak, profitabilitas perusahaan akan terganggu. Jumlah yang tepat untuk dibelanjakan sangat terkait dengan LTV, sehingga perhitungan dan pemahaman tentang CAC sangat penting.

- **Churn Rate**
  Churn rate adalah persentase pelanggan yang membatalkan langganan dalam periode waktu tertentu.

- **Retention Rate (R)**
  Retention rate adalah kebalikan dari churn rate. Ini mengukur persentase pelanggan yang tetap berlangganan selama periode waktu tertentu.

- **Average Customer Lifetime (ACL)**
  Ini adalah rata-rata waktu yang dihabiskan oleh seorang pelanggan dalam berlangganan sebelum berhenti (churn). ACL merupakan komponen kunci dalam perhitungan LTV.

- **Jumlah Pelanggan (C)**
  Ini adalah jumlah total pelanggan yang berlangganan pada layanan selama periode yang dimaksud.

- **Jumlah Transaksi (T)**
  Tidak semua pelanggan dikenakan biaya dengan frekuensi yang sama. Metrik ini mencakup jumlah transaksi yang terjadi dalam periode tersebut dan mungkin tidak sama dengan jumlah pelanggan.

- **Total Pendapatan (TR)**
  Ini adalah jumlah total uang yang diperoleh dari pelanggan selama periode yang dimaksud.
- **Laba Kotor (GP)**
  Laba kotor adalah total pendapatan dikurangi biaya penjualan.
- **Average Order Value (AOV)**
  AOV adalah total pendapatan dibagi dengan jumlah pesanan yang terjadi dalam periode tersebut.
- **Average Gross Margin (AGM)**
  AGM adalah laba kotor dibagi dengan total pendapatan, yang mengukur persentase dari pendapatan yang tersisa setelah biaya penjualan.
- **Gross Margin per Customer Lifespan (GML)**
  GML mirip dengan LTV, namun hanya menghitung margin kotor, bukan total pendapatan.
- **Discount Rate (D)**
  Discount rate adalah tingkat pengembalian investasi (ROI) yang diperoleh dari strategi yang diterapkan.

Pemahaman yang mendalam mengenai konsep-konsep ini memungkinkan tim analisis untuk memberikan kontribusi signifikan dalam evaluasi kinerja bisnis berbasis data, serta mendukung pengambilan keputusan yang lebih tepat dan terukur dalam konteks pengembangan bisnis dan pengelolaan sumber daya perusahaan [6] [7].

3.3.2 Mempelajari Platform BigQuery Sebagai Platform Data pada PT Bank Jago Tbk

BigQuery adalah *platform data* besutan Google Cloud Platform (GCP). BigQuery dipilih oleh PT Bank Jago Tbk karena memiliki kemampuan pemrosesan data yang cepat, mendukung *syntax* SQL standar, dan mendukung banyak tipe *data source* internal maupun eksternal. Fitur-fitur tersebut memungkinkan *data analyst*, *developer* dan juga tim bisnis untuk menganalisis dan mengekstrak *insight-insight* secara efisien dari dataset yang besar [8].

Proses pembelajaran berfokus pada beberapa aspek seperti cara membuat dataset, meng*import* data dari berbagai format dan *data source* dan mengintegrasikan BigQuery dengan Google Cloud Storage dan Google Sheet. Selain itu, proses pembelajaran juga mencakup cara menjalankan *query* yang kompleks menggunakan *syntax* SQL. Fitur-fitur seperti *partioning* dan juga *clustering* juga ikut dipelajari untuk mengoptimalisasi performa dari kueri. Disisi lain *cost structure* juga turut diperkenalkan agar dapat menggunakan *resources* secara efektif.

BigQuery memiliki struktur penyimpanan yang terdiri dari proyek, dataset, dan tabel. Ini adalah unit dasar dalam BigQuery yang memungkinkan penyimpanan dan pengorganisasian data secara efisien. Setiap dataset dapat berisi beberapa tabel, dan tabel ini adalah tempat data yang akan diproses dan dianalisis.

- Proyek: Merupakan unit paling atas dalam BigQuery, yang mencakup dataset dan semua sumber daya terkait.
- Dataset: Mengelompokkan tabel-tabel terkait, memungkinkan pengelolaan dan akses data secara terorganisir.
- Tabel: Menyimpan data dalam bentuk baris dan kolom, mirip dengan database relasional. Data dapat dimasukkan dalam berbagai format (seperti CSV, JSON, Parquet).

Selain itu, BigQuery mendukung partitioning dan clustering untuk mengoptimalkan penyimpanan dan performa query.

- Partitioning memungkinkan pembagian data dalam tabel berdasarkan kolom tertentu, misalnya tanggal. Ini memungkinkan BigQuery untuk memproses data dengan lebih efisien, hanya mengambil bagian data yang relevan berdasarkan query yang dijalankan.
- Clustering membantu BigQuery untuk mengurutkan data dalam tabel berdasarkan satu atau beberapa kolom. Ini membuat pencarian data menjadi lebih cepat, karena data yang serupa dikelompokkan bersama-sama dalam penyimpanan.

Berikut merupakan contoh kode SQL yang banyak digunakan pada aplikasi Unit Economics Simulator.

1. Kode SQL untuk Agregasi Data

```
SELECT
    customer_id,
    COUNT(order_id) AS total_orders,
    SUM(order_amount) AS total_revenue
FROM `project.dataset.orders`
WHERE order_date BETWEEN '2024-01-01' AND '2024-06-30'
GROUP BY customer_id
ORDER BY total_revenue DESC;
```

Penjelasan Query:

- `SELECT customer_id, COUNT(order_id) AS total_orders, SUM(order_amount) AS total_revenue`: Bagian ini memilih kolom `customer_id` sebagai pengidentifikasi pelanggan dan dua agregat, yaitu jumlah pesanan (`COUNT(order_id)`) yang dilakukan oleh masing-masing pelanggan dan total pendapatan (`SUM(order_amount)`) yang diperoleh dari masing-masing pelanggan.
- `FROM ̀project.dataset.orders`̀: Data diambil dari tabel `orders` yang ada pada dataset `dataset` di dalam proyek `project`.
- `WHERE order_date BETWEEN '2024-01-01' AND '2024-06-30'`: Menyaring data berdasarkan tanggal pesanan, hanya mengambil pesanan yang terjadi antara 1 Januari 2024 dan 30 Juni 2024.
- `GROUP BY customer_id`: Data dikelompokkan berdasarkan `customer_id`, sehingga untuk setiap pelanggan, agregat `COUNT` dan `SUM` akan dihitung.
- `ORDER BY total_revenue DESC`: Hasil query diurutkan berdasarkan `total_revenue` dalam urutan menurun (descending), sehingga pelanggan dengan pendapatan tertinggi akan muncul terlebih dahulu.

Referensi: Kode ini digunakan untuk menghitung total pesanan dan pendapatan per pelanggan pada periode yang ditentukan.

2. Kode SQL untuk Filter Data

```
SELECT *
FROM `project.dataset.transactions`
```

```
    WHERE transaction_amount > 1000 AND status = '
completed';

```

Penjelasan Query:

- `SELECT *`: Bagian ini memilih semua kolom yang ada pada tabel `transactions`.
- `FROM ` project.dataset.transactions`:` Data diambil dari tabel `transactions` yang berada dalam dataset `dataset` di proyek `project`.
- `WHERE transaction_amount > 1000 AND status = 'completed'`: Data disaring dengan dua kondisi: transaksi yang memiliki jumlah lebih besar dari 1000 (`transaction_amount > 1000`) dan status transaksi yang sudah selesai (`status = 'completed'`).

Referensi: Kode ini digunakan untuk mengambil transaksi yang memiliki nilai lebih besar dari 1000 dan sudah selesai.

3. Kode SQL untuk Join Antar Tabel

```
SELECT
    a.customer_name,
    b.product_name,
    b.order_date
FROM `project.dataset.customers` a
INNER JOIN `project.dataset.orders` b
ON a.customer_id = b.customer_id;

```

Penjelasan Query:

- `SELECT a.customer_name, b.product_name, b.order_date`: Bagian ini memilih kolom `customer_name` dari tabel `customers`, serta kolom `product_name` dan `order_date` dari tabel `orders`.
- `FROM ` project.dataset.customersà`:` Data diambil dari tabel `customers` yang berada dalam dataset `dataset` di proyek `project`, dan tabel ini diberi alias `a` untuk referensi yang lebih mudah.

- `INNER JOIN` `project.dataset.orders`: Data dari tabel `orders` (dengan alias `b`) digabungkan menggunakan `INNER JOIN`. Hanya baris yang memiliki pasangan yang cocok di kedua tabel yang akan diambil.
- `ON a.customer_id = b.customer_id`: Kondisi penggabungan menggunakan kolom `customer_id` yang ada di kedua tabel (`customers` dan `orders`). Join ini akan mencocokkan data berdasarkan `customer_id`, menghubungkan pelanggan dengan pesanannya.

Referensi: Kode ini digunakan untuk menggabungkan data dari dua tabel (`customers` dan `orders`), sehingga menghasilkan informasi terkait nama pelanggan, produk yang dipesan, dan tanggal pesanan yang terkait.

Secara keseluruhan, mempelajari BigQuery memperkuat kemampuan dalam menghadirkan solusi analitik yang andal, cepat, dan *scaleable*. Melalui pemahaman yang lebih mendalam atas konsep-konsep ini dapat mengoptimalkan proses pengambilan keputusan yang lebih berorientasi pada data, mendukung tim untuk mengidentifikasi peluang, serta meningkatkan kinerja dan efisiensi operasional secara menyeluruh.

3.3.3 Mempelajari Platform CI/CD Harness Sebagai Platform CI/CD

Harness digunakan PT Bank Jago Tbk sebagai sebagai *platform continuous integration and continuous deployment* (CI/CD) dari PT Bank Jago Tbk. CI merupakan sebuah praktek dimana *developer* secara reguler melakukan *merge* terhadap kode yang dibuat ke dalam *shared repository*, sedangkan CD merupakan sebuah proses automasi untuk merilis kode yang sudah dites ke dalam *environment production*. Harness merupakan solusi CI/CD berbasis *cloud* yang didesain untuk mengoptimalkan pengelolaan, automasi, *build* software, *test*, dan proses *deployment*. Harness dilengkapi dengan fitur-fitur pendukung yang membuat seluruh proses CI/CD dapat dijalankan dalam satu platform yang dapat mengurangi kompleksitas dan intervensi *developer* secara manual [9].

Sebagai contoh implementasi penggunaan platform Harness dalam suatu pipeline CI/CD, berikut ini ditunjukkan potongan kode untuk membuat pipeline yang akan membangun (build) dan mendorong (push) *image* Docker ke dalam registry Docker. Kode ini menunjukkan bagaimana Harness mengelola tahapan-tahapan dalam proses otomatisasi, mulai dari pembuatan *image* Docker hingga pengunggahan ke registry yang ditentukan, seperti Docker Hub yang dapat dilihat

pada kode 3.1.

```
        pipeline:
          name: Simple Build Pipeline
          identifier: simple-build-pipeline
          stages:
            - stage:
                name: Build
                identifier: build-stage
                type: Build
                spec:
                  execution:
                    steps:
                      - step:
                          name: Build Docker Image
                          identifier: build-docker-image
                          type: Docker
                          spec:
                            image: "node:14"
                            build:
                              context: "./"
                              dockerfile: "Dockerfile"
                              tags:
                                - "latest"
                            push:
                              enabled: true
                              registry: "docker.io"
                              username: "<your-docker-username>"
                              password: "<your-docker-password>"
                      - step:
                          name: Push to Docker Hub
                          identifier: push-docker-image
                          type: DockerPush
                          spec:
                            image: "docker.io/username/app-name:
    latest"
                            registry:
                              url: "docker.io"
                              username: "<your-docker-username>"
                              password: "<your-docker-password>"

          trigger:
            type: Git
            spec:
```

```
42                 git:
43                   repoName: "my-app-repo"
44                   branch: "main"
45                   webhook:
46                     enabled: true
47
```

Listing 3.1: Contoh potongan kode Harness

Kode 3.1 merupakan implementasi sederhana dari pipeline CI/CD menggunakan platform Harness untuk membangun dan melakukan *push image* Docker ke registry Docker Hub. Berikut adalah penjelasan dari setiap bagian kode:

- **Pipeline Configuration:**
  - `name: Simple Build Pipeline` mendefinisikan nama dari pipeline ini.
  - `identifier: simple-build-pipeline` memberikan identifikasi unik untuk pipeline yang digunakan dalam sistem.
- **Stages:** Pipeline dibagi menjadi beberapa tahap (stages). Dalam kode ini, terdapat satu stage yaitu `Build` yang bertanggung jawab untuk membangun dan melakukan *push image* Docker ke Docker Hub.
  - `type: Build` mendefinisikan bahwa tahap ini bertipe build, yang akan menjalankan proses pembuatan *image* Docker.
  - `execution:` bagian ini mengatur langkah-langkah yang akan dijalankan dalam stage ini.
- **Step - Build Docker Image:**
  - `name: Build Docker Image` memberikan nama untuk langkah pertama, yang bertugas untuk membuat *image* Docker.
  - `image: "node:14"` mendefinisikan *base image* Docker yang digunakan dalam proses build, dalam hal ini adalah *image* Node.js versi 14.
  - `build:` mengatur konfigurasi build Docker, dengan `context: "./"` yang menunjukkan direktori konteks untuk build, dan `dockerfile: "Dockerfile"` yang menunjukkan lokasi Dockerfile yang digunakan dalam proses build.

- `tags`: bagian ini mendefinisikan tag untuk *image* yang dibangun, dalam hal ini menggunakan tag `"latest"`.

- **Step - Push to Docker Hub:** Setelah *image* Docker dibuat, langkah berikutnya adalah melakukan *push image* tersebut ke registry.
  - `name: Push to Docker Hub` adalah langkah yang mengupload *image* Docker yang telah dibangun ke Docker Hub.
  - `image: "docker.io/username/app-name:latest"` mendefinisikan lokasi *image* yang akan didorong ke registry Docker Hub.
  - `registry`: bagian ini berisi informasi tentang registry yang digunakan, dalam hal ini adalah Docker Hub, dan kredensial pengguna untuk autentikasi (username dan password).

- **Trigger:** Bagian ini mendefinisikan bagaimana pipeline ini akan dipicu (triggered).
  - `type: Git` menunjukkan bahwa pipeline ini akan dipicu oleh perubahan pada repositori Git.
  - `repoName: "my-app-repo"` mengacu pada nama repositori Git yang terhubung.
  - `branch: "main"` mendefinisikan cabang Git yang menjadi pemicu, dalam hal ini adalah cabang utama (main).
  - `webhook: enabled: true` mengaktifkan webhook agar pipeline dapat dijalankan secara otomatis setiap kali ada perubahan pada repositori.

Secara keseluruhan, kode ini menunjukkan alur kerja CI/CD yang efisien menggunakan platform Harness untuk membangun dan mengunggah *image* Docker ke Docker Hub secara otomatis saat ada pembaruan pada repositori Git.

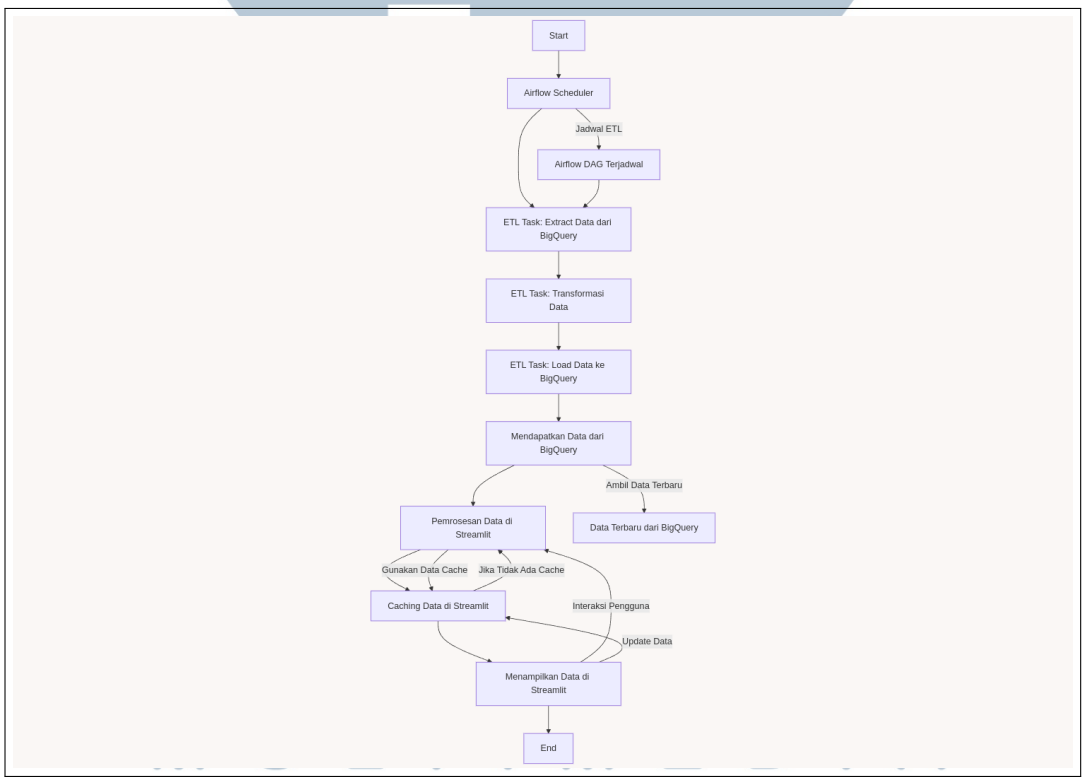
3.3.4 Pengembangan Unit Economics Simulator Menggunakan Streamlit

Streamlit adalah platform sumber terbuka yang digunakan dalam pengembangan Unit Economics Simulator untuk menyediakan antarmuka berbasis web yang interaktif dan mudah digunakan. Sebagai alat untuk visualisasi dan

pengolahan data, Streamlit memungkinkan integrasi dengan layanan lain, seperti BigQuery, untuk menampilkan hasil query dalam bentuk yang lebih mudah dipahami oleh pengguna. Dalam konteks ini, Streamlit berperan penting dalam menyajikan informasi hasil analisis data, serta mendukung pembuatan model bisnis berbasis data secara dinamis dan efisien.

Aplikasi Unit Economics Simulator ini dibangun menggunakan Streamlit untuk menyediakan pengalaman pengguna yang interaktif, memungkinkan pengguna untuk berinteraksi dengan data secara langsung, melakukan analisis, serta memodelkan variabel-variabel unit economics. Platform ini mengutamakan kemudahan penggunaan, sehingga pengguna dapat memanfaatkan fungsionalitasnya tanpa memerlukan keterampilan teknis mendalam.

Gambar berikut (3.1) memperlihatkan alur aplikasi Unit Economics Simulator menggunakan Streamlit, yang menggambarkan bagaimana data dari BigQuery diolah dan divisualisasikan dalam antarmuka web Streamlit.



Gambar 3.1. Alur Penggunaan Unit Economics Simulator Menggunakan Streamlit

Berikut merupakan penjelasan alur kerja aplikasi berdasarkan gambar 3.1.

- **Start (A)**: Proses dimulai di titik ini. Ini adalah titik awal dari alur kerja.

- **Airflow Scheduler (B)**: Airflow Scheduler bertanggung jawab untuk menjadwalkan dan memicu **ETL** (Extract, Transform, Load) sesuai dengan waktu yang telah ditentukan. *Scheduler* ini akan memulai DAG (Directed Acyclic Graph) yang mendefinisikan tugas-tugas ETL yang harus dijalankan.
- **Airflow DAG Terjadwal (K)**: **Airflow DAG** adalah sekumpulan tugas yang terdefinisi dan dijalankan dalam urutan tertentu. DAG ini mengatur kapan dan bagaimana setiap tugas dalam proses ETL dieksekusi, berdasarkan jadwal yang ditentukan.
- **ETL Task: Extract Data dari BigQuery (C)**: Langkah pertama dari proses ETL adalah **Extract**, yaitu mengambil data yang relevan dari **BigQuery**. Data diambil melalui query SQL atau API BigQuery untuk mendapatkan dataset yang diperlukan dalam aplikasi.
- **ETL Task: Transformasi Data (D)**: Setelah data berhasil diekstraksi, langkah berikutnya adalah **Transformasi**. Di sini, data diolah sesuai dengan kebutuhan aplikasi, seperti perhitungan unit economics, agregasi data, atau perubahan format data.
- **ETL Task: Load Data ke BigQuery (E)**: Langkah terakhir dari proses ETL adalah **Load**, di mana data yang sudah diproses dimuat kembali ke **BigQuery** untuk digunakan lebih lanjut dalam aplikasi Streamlit.
- **Mendapatkan Data dari BigQuery (F)**: Setelah data diproses dan dimuat, aplikasi **Streamlit** akan menarik data terbaru dari **BigQuery**. Pengambilan data ini terjadi melalui query SQL atau API BigQuery untuk mendapatkan data yang terbaru dan relevan.
- **Pemrosesan Data di Streamlit (G)**: Data yang diambil kemudian diproses lebih lanjut di dalam **Streamlit**. Pemrosesan ini bisa meliputi perhitungan unit economics, agregasi data, atau transformasi data untuk menampilkan visualisasi yang lebih jelas dan mudah dimengerti.
- **Caching Data di Streamlit (H)**: Untuk meningkatkan performa dan kecepatan aplikasi, **Streamlit** menyimpan hasil pemrosesan data yang sering digunakan dalam cache. Hal ini memastikan aplikasi tidak perlu memproses data yang sama berulang-ulang, sehingga mempercepat waktu tampilan data.

- **Menampilkan Data di Streamlit (I)**: Setelah data diproses dan cache disiapkan, hasilnya ditampilkan di **Streamlit**. Pengguna dapat melihat visualisasi, grafik, atau tabel yang menunjukkan simulasi unit economics atau hasil analisis lainnya. Pada titik ini, aplikasi siap digunakan.
- **Interaksi Pengguna dan Update Data (I → G)**: Pengguna dapat berinteraksi dengan aplikasi Streamlit untuk memperbarui data, melakukan filter, atau melihat simulasi lain. Interaksi ini akan memicu pemrosesan data baru berdasarkan input pengguna.
- **Update Cache (I → H)**: Jika ada interaksi yang menyebabkan perubahan pada data atau tampilan, cache yang ada akan diupdate agar pengguna mendapatkan data yang terbaru.

Contoh kode Python untuk membuat visualisi pada streamlit dapat dilihat pada kode 3.2.

```
import streamlit as st
import pandas as pd
import numpy as np

# Set the title of the app
st.title("Basic Data Visualization with Streamlit")

# Create a simple dataset
# Here we create a DataFrame with dates and some random values
date_range = pd.date_range(start='2023-01-01', periods=100)
data = pd.DataFrame({
    'Date': date_range,
    'Value': np.random.randn(100).cumsum()  # Cumulative sum of random numbers for a line chart
})

# Display the dataframe
st.subheader("Data Preview")
st.write(data)

# Display a line chart
st.subheader("Line Chart of Cumulative Values")
st.line_chart(data.set_index('Date')['Value'])
```

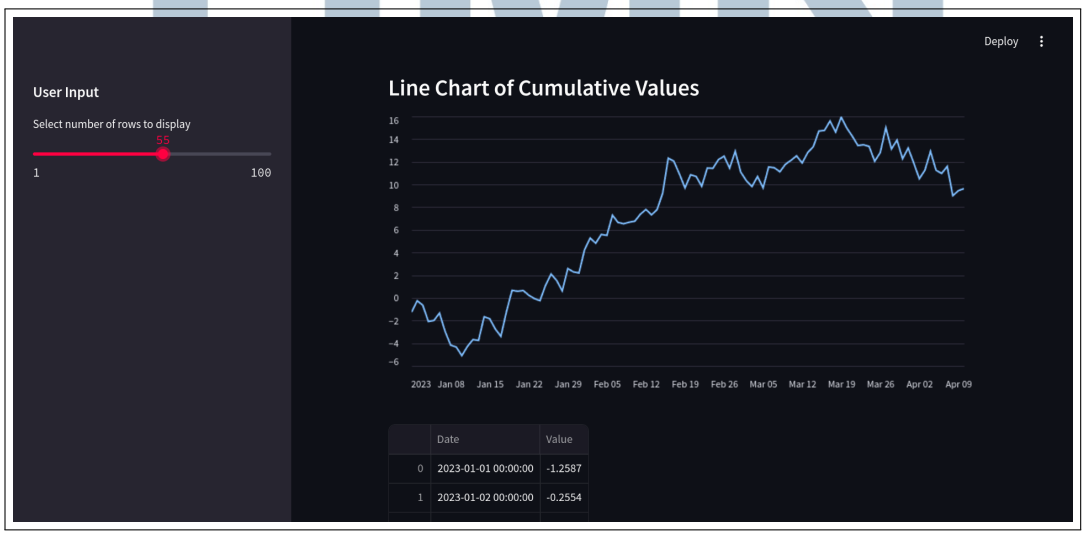
```
23
24          # You can also add some interactivity
25          st.sidebar.subheader("User Input")
26          slider_value = st.sidebar.slider("Select number of
    rows to display", 1, 100, 20)
27          st.write(data.head(slider_value))
28
29
```

Listing 3.2: Contoh potongan kode Streamlit

Pada potongan kode 3.2, beberapa komponen penting dari Streamlit dapat dilihat, di antaranya:

- Judul Aplikasi: Menggunakan st.title() untuk memberi nama pada aplikasi.
- Dataframe: Dataset sederhana yang dihasilkan menggunakan pandas dan numpy, yang terdiri dari rangkaian tanggal dan nilai kumulatif acak.
- Visualisasi: Menggunakan st.line_chart() untuk menampilkan grafik garis dari data yang telah disusun.
- Interaktivitas: Pengguna dapat memilih jumlah baris data yang ingin ditampilkan melalui slider yang ada di sidebar aplikasi dengan menggunakan st.sidebar.slider().

Output dari kode 3.2 akan menghasilkan sebuah aplikasi web interaktif yang memungkinkan pengguna untuk melihat pratinjau data dalam bentuk tabel dan grafik.



Gambar 3.2. Contoh Hasil Visualisasi Streamlit

Berdasarkan visualisasi yang dihasilkan pada 3.2, terdapat dua sumbu utama yang membentuk grafik garis:

- Sumbu X (Horizontal): Pada grafik ini, sumbu X mewakili tanggal yang terdapat dalam dataset. Sumbu ini diatur dengan menggunakan kolom Date pada DataFrame data. Setiap titik pada sumbu X menunjukkan waktu dalam format tanggal, yang dalam hal ini dimulai dari 1 Januari 2023 dan berlanjut selama 100 hari berikutnya.
- Sumbu Y (Vertikal): Sumbu Y menunjukkan nilai kumulatif yang dihitung secara acak dalam kolom Value. Nilai pada sumbu Y ini diperoleh dengan menggunakan fungsi np.random.randn(100).cumsum(), yang menghasilkan angka-angka acak yang dijumlahkan secara kumulatif. Grafik ini memperlihatkan bagaimana nilai kumulatif tersebut berkembang seiring waktu.

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala

Selama proses kerja magang di PT Bank Jago Tbk terdapat kendala berupa kurangnya dokumentasi terkait *policy* dalam melakukan *deployment* ke dalam *environment production*. Kendala ini menyebabkan proses deployment aplikasi menjadi terhambat dan membutuhkan lebih banyak waktu untuk memenuhi standar yang ada dari pihak *Site Reliability Engineer (SRE)* dan *Information Security (InfoSec)*.

3.4.2 Solusi

Solusi atas kendala tersebut adalah bertanya dengan jelas kepada Supervisor, divisi *SRE* dan *InfoSec* mengenai *policy* apa saja yang perlu dipatuhi sebelum melakukan *deployment* ke dalam *environment production*.