# BAB III
# PELAKSANAAN KERJA MAGANG

## 3.1 Kedudukan dan Koordinasi

Proses pelaksanaan pratik kerja magang yang dilakukan oleh peserta magang berperan dan memiliki kedudukan internship dalam functional unit Corporate Human Resources Kompas Gramedia Group. Selama menjalani pratik kerja magang peserta dibimbing secara langsung oleh tim People Data Analyst yaitu bersama Kak Irsa Salsabilah yang memiliki jabatan sebagai Freelance People Data Analyst, Kak Dahayu Bethari Widyandri yang memiliki jabatan sebagai Freelance People Data Analyst, kemudian Arthur sebagai internship people data analyst serta Supervisi dalam team ini ialah Mas Bramanto Ranggamukti sebagai seorang Lead People Data Analyst yang tentunya memiliki peran penting dalam struktur Kompas Gramedia Group. Pada people data analyst tidak hanya mencakup freelance, internship dan lead saja melainkan dipimpin langsung oleh CEO yaitu mas Arki Sudito yang dimana memiliki peran dalam mengarahkan dan mengelola seluruh kegiatan di dalam Growth Center. Untuk lebih detail terkait dengan struktur organisasi people data analyst terlihat pada Gambar 3.1 Struktur organisasi people data analyst.


Gambar 3.1 Struktur Organisasi People Data Analyst

Dalam struktur organisasi diatas juga memperlihatkan peran peserta magang sebagai internship people data analyst yang berkaitan erat dengan pekerjaan yang berfokus utama pada teknologi sistem informasi. Functional unit Corporate Human Resources, khususnya pada departement People Data analyst di Growth Center Kompas Gramedia Group memiliki tanggung jawab dalam bekerja sama dengan berbagai unit seperti tim developmen, tim engagement, tim discovery dan juga tim activation. People Data Analyst berperan dalam mengimplementasikan seluruh ilmu data yang nantinya dapat diolah kembali dengan memperoleh saran atau informasi kedepannya untuk dapat meningkatkan kembali kinerja produktivitas pada keseluruhan bisnis.

People Data Analyst juga bertugas dalam melakukan dan menjalankan berbagai project yang berfokus utama pada data sehingga mampu menghasilkan wawasan ataupun informasi untuk pengambilan Keputusan yang lebih baik kedepannya. Salah satu tugas utama sebagai seorang People Data Analyst ialah mampu merancang, membangun, serta mengimplementasikan visualisasi data pada sebuah dashboard pada platform Growth Center contohnya seperti membentuk sebuah diagram atau grafik yang nantinya mampu memberikan sebuag gambaran atau pemahaman yang lebih baik terkait dengan pola atau informasi yang terjadi dalam data tersebut. Jika mengalami kendala dalam report ataupun request data maka dapat dikoordinasikan kembali secara langsung dengan unit People Data Analyst untuk dilakukan perbaikan dan pemeliharaan data kembali.

Proses pratik kerja magang yang dilakukan oleh peserta magang bertanggung jawab dalam beberapa project dan request data dari unit lainnya. Seluruh proses project dan data request memiliki alur kerja yang dimulai dari disampaikan oleh Lead People Data Analyst yaitu Mas Bramanto kepada tim freelance People Data Analyst untuk menangani setiap permasalahan project, pembuatan project hingga data request. Selanjutnya setelah tim freelance menerima tugas tersebut nantinya akan ditugaskan kepada masing – masing dari tim People

Data Analyst dengan melalui meeting Weekly Team Data untuk menentukan PIC ataupun siapa yang akan menghandle tugas atau pekerjaan tersebut.

Koordinasi yang dilakukan dalam tim People Data Analyst yaitu dengan menggunakan Google Meet sebagai platform utama untuk mengadakan meeting mingguan dan melakukan discuss terkait pekerjaan yang ingin disampaikan serta dengan menggunakan Google Meet ini mampu mendukung kelancaran dalam proses bisnis. Sedangkan untuk berkoordinasi dengan tim development, tim engagement, tim discovery dan juga tim activation dapat melalui platform WhatsApp. Berikut ini jadwal koordinasi dan komunikasi yang dilakukan dalam functional unit Corporate Human Resources pada perusahaan Kompas Gramedia Group terlihat pada tabel 3.1 Koordinasi dan Komunikasi Kompas Gramedia Group.

Tabel 3.1 Koordinasi dan Komunikasi Kompas Gramedia Group

| Koordinasi | Jadwal | Deskripsi |
|---|---|---|
| Weekly Meeting All HR Expertise, SBP and Intern | Setiap Senin, Pukul 10:00 s/d 11:00 | a. Check in,<br>b. Discussion on New Weekly Handle. |
| Weekly Meeting People Data Analyst | Setiap Senin, Pukul 14:00 s/d 16:00 | a. Update pekerjaan yang telah dilakukan,<br>b. Pembagian tugas untuk project atau request data yang masuk,<br>c. Update seluruh kegiatan melalui platform Asana. |

| | | |
|---|---|---|
| Meeting Team People Data Analyst | Sesuai kebutuhan dari team People Data Analyst | Melakukan meeting terkait project yang akan dating seperti dashboard statisfaction streamlit. |

### 3.1.1 Weekly Meeting All HR Expertise, SBP and Intern

Pada functional unit Corporate Human Resources melakukan kegiatan meeting mingguan bersama dengan seluruh tim dari naungan Corporate Human Resources yang dimana Weekly meeting All HR Expertise dilakukan setiap hari Senin pada pukul 10:00 WIB sampai dengan 11:00 WIB dengan melalui platform Zoom Meeting. Meeting weekly All HR Expertise melakukan pembahasan mengenai Discussion on New Weekly handle yang dimana dalam diskusi meeting ini melakukan pembahasan mengenai kegiatan yang telah dikerjakan oleh setiap unit dalam Growth Center serta melakukan updatean mengenai kegiatan atau aktivitas apa yang akan dilakukan di minggu tersebut. Pada Gambar 3.2 memperlihatkan jadwal Weekly Meeting All HR.


Gambar 3.2 Jadwal Weekly Meeting All HR Expertise x SBP X All

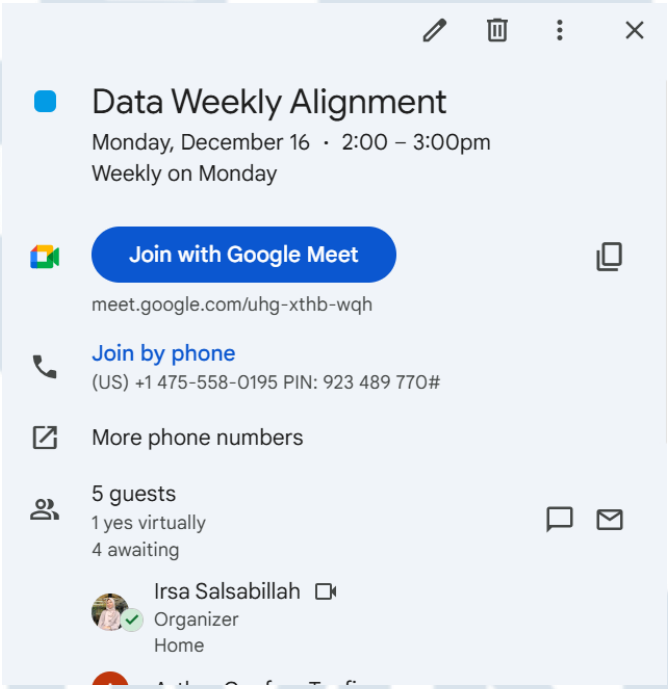
Weekly update juga menjelaskan mengenai project yang akan dilaksanakan kedepannya seperti Growth Conversation yang dimana project ini di handle oleh Kak Andini Nabila Sari yang dimana Growth Conversation ini akan dilakukan selama 3 kuater dan digunakan sebagai bentuk fasilitas pertumbuhan bagi seluruh karyawan Growth Center baik freelance maupun internship yang dimana dapat diharapkan mampu membantu seluruh karyawan dapat menemukan the best version of ourselves melalui sesi 1 on 1 dan juga dapat menciptakan 360 derajat feedback loop antara karyawan dengan streamlead, kemudian streamlead dengan manajemen.

Sebelumnya kegiatan ini telah dilakukan sehingga hanya akan melanjutkan kuarter ke 3 untuk dapat memaksimalkan tracking, maintaining dan improving performance karyawan sehingga berguna untuk menyelaraskan dengan pertumbuhan organisasi, memberikan support dan arahan untuk karyawan mencapai personal dan professional goals dan skills serta mampu meningkatkan hubungan interpersonal baik itu antara mentor dan karyawan untuk dapat memberikan pengalaman kerja yang lebih baik. Pada weekly meeting all HR ini juga memberikan penjelasan mengenai pelaksanaan Growth Conversation, timeline kegiatan hingga detail dari agenda aktivitas tersebut tetapi tidak hanya project ini saja setiap minggunya aka nada aktivitas ataupun project baru yang nantinya akan dijelaskan dalam Weekly meeting All HR Expertise.

### 3.1.2 Weekly Meeting People Data Analyst

Pada weekly meeting People Data Analyst yang dimana dilaksanakan pada setiap minggunya yaitu di hari Senin dan dapat digantikan menjadi hari berikutnya jika terdapat hari libur dan seringkali dimulai pada pukul 14:00 sampai dengan pukul 16:00. Meeting weekly

team data ini dilakukan dengan melalui platform Google Gmeet yang sebelumnya telah di set setiap minggunya melalui Google Calendar yang dilakukan oleh Kak Irsa Salsabillah sebagai organizer. Weekly meeting team data ini juga diikuti oleh seluruh karyawan team data mencakup mbak dahayu, kak irsa, Arthur dan peserta magang sebagai Internship Kompas Gramedia Group. Pada Gambar 3.3 memperlihatkan jadwal weekly meeting team People Data Analyst.




Gambar 3.3 Jadwal Weekly Meeting Team Data

Weekly meeting ini tentunya dilakukan setiap minggunya guna untuk mendiskusikan bersama terkait dengan apa saja yang telah dikerjakan pada minggu sebelumnya melakukan pembagian tugas terkait project yang akan dijalankan selanjutnya dan request data dari tim lainnya. Untuk dapat mengetahui progress pekerjaan tersebut tentunya dalam weekly meeting ini dilakukan update dengan melalui platform Asana sebagai salah satu platform penunjang koordinasi pekerjaan dalam team

data yang dimana dalam pembagian pekerjaan ini nantikan akan ditentukan siapa yang akan menghandle atau menjadi PIC dalam project tersebut. Serta dalam weekly meeting ini akan melakukan tugas rutin yaitu melakukan pembuatan report monthly baik itu assessment maupun all platform yang nantinya dari seluruh report tersebut akan dilakukan presentasi kepada seluruh karyawan Growth Center.

### 3.2 Tugas dan Uraian Kerja Magang

Pratik kerja magang yang dilaksanakan oleh peserta magang pada perusahaan Kompas Gramedia Group berlangsung selama 6 bulan yang dimulai pada periode 1 Juli 2024 sampai dengan 31 Januari 2025 dengan melibatkan peserta magang sebagaimana yang telah dicantumkan dalam Letter of Acceptance yaitu sebagai seorang Internship People Data Analyst. Peserta magang juga terlibat secara langsung dalam berbagai pekerjaan sebagai seorang people data analyst yang dimana mencakup sesi onboarding, weekly meeting HR Expertise X SBP X Intern, Weekly meeting people data analyst yang dimana telah menjalankan tugas sesuai dengan job description yang telah diberikan oleh perusahaan Kompas Gramedia Group yang dimana tugas seorang People Data Analyst mencakup:

a. Melakukan compile dan cleaning data
b. Merekomendasikan dan menerapkan tata Kelola data untuk data yang disimpan di platform Growth Center
c. Mencatat kebutuhan analisis pengguna dan buat dashboard yang sesuai dengan berbagai platform (Looker Studio/Metabase/Tableau)
d. Memastikan user memahami cara menggunakan dashboard dan menyiapkan dashboard untuk dibagikan kepada stakeholder di bawah supervisi PIC Data (contoh dengan tableau online); dan
e. Merekomendasikan cara untuk meningkatkan keandalan, efisiensi dan kualitas data.

Sehingga, seluruh job description sebagai seorang Internship People Data Analyst peserta magang memiliki peran dalam mengelola seluruh data pada seluruh platform perusahaan Kompas Gramedia Group seperti Kognisi.id, Kognisi.mykg, Discovery dan Capture sehingga dari keseluruhan data tersebut dapat diolah sehingga mampu menjadi sebuah informasi yang akurat dan tepat serta nantinya dapat membantu stakeholder dalam membuat Keputusan dan rekomendasi yang tepat untuk perkembangan perusahaan. Pada perusahaan Kompas Gramedia tools yang digunakan sebagai penunjang pekerjaan mencakup Tableau Online, Dbeaver, Ms. Excel, Ms. Power Point, Asana dan Streamlit. Berikut penjelasan terkait uraian pekerjaan yang dilakukan peserta magang dalam merancang sebuah dashboard streamlit discovery terlihat pada tabel 3.2 Uraian Aktivitas Pratik Kerja Magang pada perusahaan Kompas Gramedia Group sebagai seorang Internship People Data Analyst yaitu:

Tabel 3.2 Uraian Aktivitas Pratik Kerja Magang

| No | Job | Deskripsi Pekerjaan |
|---|---|---|
| 1. | Connect database, data Capture dan SAP | Melakukan connect database pada streamlit terhadap data – data yang diperlukan untuk membuat dashboard Discovery. |
| 2. | Page user growth | Membuat halaman user growth yang berisikan jumlah user yang mengakses Discovery dan Capture secara lifetime, his year dan this month. |
| 3. | Page demography | Membuat halaman demography yang berisikan demography user yang mengakses platform Discovery dan Capture. |
| 4. | Page result traits | Membuat halaman result traits yang berisikan distribusi jumlah seluruh user Kompas |

| | | |
|---|---|---|
| | | Gramedia group yang mengakses setiap platform |
| 5. | Page internal KG | Membuat halaman internal KG yang berisikan test taker per unit serta distribusinya dalam unit, gender, generation dan layer. |
| 6. | Page layer traits | Membuat halaman layer traits yang berisikan karyawan KG layer traits summary pada platform Discovery. |

Timeline pekerjaan yang dilaksanakan selama pratik kerja magang sebagai seorang people data analyst dalam membuat project dashboard stremlit Discovery selengkapnya dapat dilihat pada tabel 3.3 Uraian Aktivitas dan Output Project.

Tabel 3.3 Uraian Aktivitas dan Output Project

| No | Job | Minggu Ke- | Tanggal Mulai | Tanggal Selesai |
|---|---|---|---|---|
| 1. | Connect data | 1 - 2 | 01 Juli 2024 | 12 Juli 2024 |
| 2. | Page user growth | 3 - 4 | 15 Juli 2024 | 27 Juli 2024 |
| 3. | Page demography | 5 – 7 | 29 Juli 2024 | 09 Agustus 2024 |
| 4. | Page result traits | 8 - 10 | 12 Agustus 2024 | 30 Agustus 2024 |
| 5. | Page internal KG | 11 – 14 | 02 September 2024 | 30 September 2024 |
| 6. | Page layer traits | 15 – 19 | 01 Oktober 2024 | 01 November 2024 |

Tugas yang dilakukan selama proses pratik kerja magang yaitu merancang sebuah dashboard streamlit platform Discovery dan Capture agar nantinya dapat diakses oleh stakeholder secara real time. Berikut ini adalah penjelasan untuk setiap bagian yang dilakukan dalam perancangan dashboard tsreamlit discovery.

### 3.2.1 Connect Data

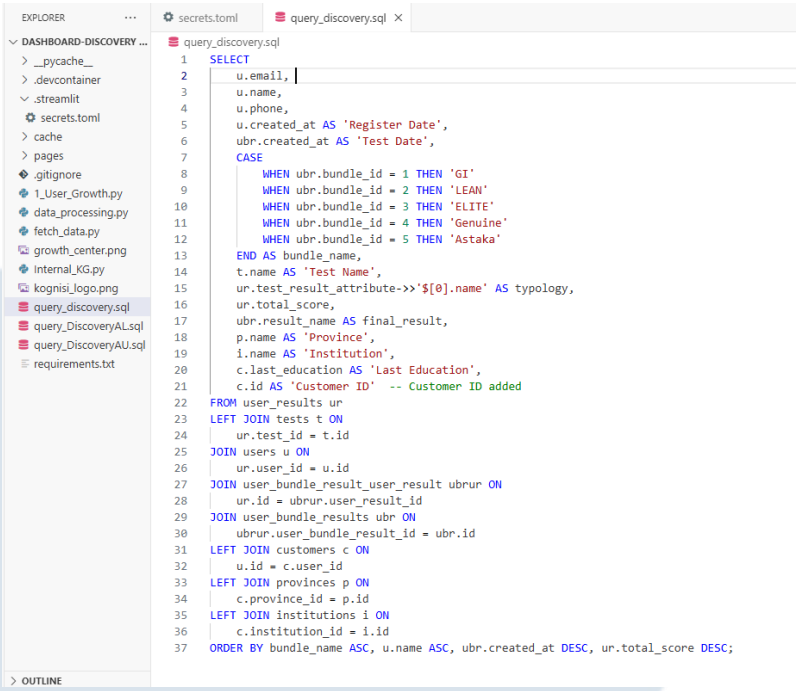
Peserta magang yang telah bergabung pada perusahaan Kompas Gramedia Group sebagai salah satu team people data analyst pada pertengahan tahun yaitu di bulan 1 Juli 2024 maka peserta magang membuat sebuah perancangan dashboard dengan menggunakan streamlit yang dimana hal ini dilakukan karena setelah peserta magang mengamati seluruh platform pembelajaran yang ada pada perusahaan Kompas Gramedia Group ternyata pada platform Discovery dan Capture sering kali dilakukan analisis secara manual dengan membuat report perbulan dan stakeholder kadangkala melakukan request terkait kebutuhan yang diinginkan. Sehingga dari kendala itu peserta magang memberikan sebuah ide dengan melakukan perancangan dashboard streamlit hingga seluruh kebutuhan stakeholder dapat dijangkau dan diakses secara real time pada dashboard streamlit discovery.

Pada minggu ke – 1 dan minggu ke – 2 peserta magang tentunya melakukan analisis terlebih dahulu terkait kebutuhan – kebutuhan yang diperlukan stakeholder kemudian dilakukan perancangan kasar terkait dengan tampilan dashboard setelah hal tersebut dilakukan maka selanjutnya peserta magang akan melakukan connect data terlebih dahulu pada dashboard streamlit untuk connect data sendiri disini peserta magang menggunakan streamlit dan melakukan penulisan log codingan pada codespace yang difasilitasi oleh streamlit. Connect data ini tentunya harus dilakukan secara hati – hati karena hal ini mencakup 3 sumber data yaitu database discovery, capture dan sap.


Gambar 3.4 Syntax Connect Data

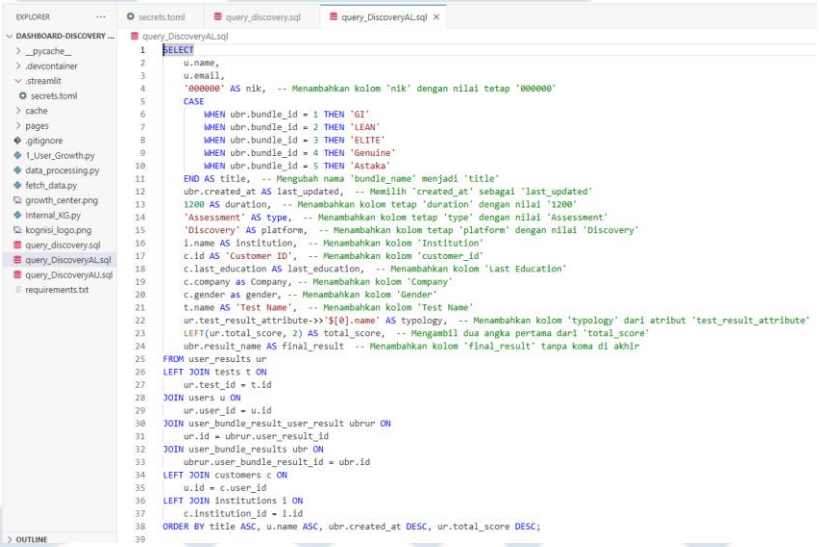
Maka dari itu untuk connect data sendiri dibuat pada secrets.toml yang terlihat pada Gambar 3.4 Syntax Connect Data. Penggunaan secrets.toml ini yang dimana merupakan file konfigurasi yang digunakan pada aplikasi streamlit untuk dapat menyimpan data sensitive seperti API keys, password, atau variabel lainnya. File ini mampu menjaga keamanan data dengan memastikan bahwa informasi sensitive tidak dicantumkan langsung di kode sumber terutama jika repositori kode diunggah ke platform public seperti GitHub.

Setelah melakukan connect data tentunya akan dilakukan pemanggilan data yang diperlukan untuk membuat dashboard streamlit untuk pemanggilan data sendiri menggunakan syntax query yang dimana menggunakan 3 syntax query yaitu Query Discovery, Query Discovery AL, dan Query Discovery AU untuk syntaxnya terlihat pada Gambar berikut ini.



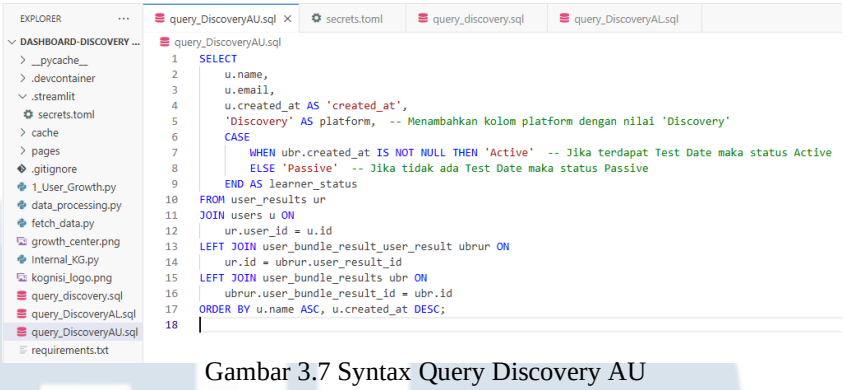
```sql
SELECT
    u.email,
    u.name,
    u.phone,
    u.created_at AS 'Register Date',
    ubr.created_at AS 'Test Date',
    CASE
        WHEN ubr.bundle_id = 1 THEN 'GI'
        WHEN ubr.bundle_id = 2 THEN 'LEAN'
        WHEN ubr.bundle_id = 3 THEN 'ELITE'
        WHEN ubr.bundle_id = 4 THEN 'Genuine'
        WHEN ubr.bundle_id = 5 THEN 'Astaka'
    END AS bundle_name,
    t.name AS 'Test Name',
    ur.test_result_attribute->>'$[0].name' AS typology,
    ur.total_score,
    ubr.result_name AS final_result,
    p.name AS 'Province',
    i.name AS 'Institution',
    c.last_education AS 'Last Education',
    c.id AS 'Customer ID'  -- Customer ID added
FROM user_results ur
LEFT JOIN tests t ON
    ur.test_id = t.id
JOIN users u ON
    ur.user_id = u.id
JOIN user_bundle_result_user_result ubrur ON
    ur.id = ubrur.user_result_id
JOIN user_bundle_results ubr ON
    ubrur.user_bundle_result_id = ubr.id
LEFT JOIN customers c ON
    u.id = c.user_id
LEFT JOIN provinces p ON
    c.province_id = p.id
LEFT JOIN institutions i ON
    c.institution_id = i.id
ORDER BY bundle_name ASC, u.name ASC, ubr.created_at DESC, ur.total_score DESC;
```

Gambar 3.5 Syntax Query Discovery



```sql
SELECT
    u.name,
    u.email,
    '000000' AS nik,  -- Menambahkan kolom 'nik' dengan nilai tetap '000000'
    CASE
        WHEN ubr.bundle_id = 1 THEN 'GI'
        WHEN ubr.bundle_id = 2 THEN 'LEAN'
        WHEN ubr.bundle_id = 3 THEN 'ELITE'
        WHEN ubr.bundle_id = 4 THEN 'Genuine'
        WHEN ubr.bundle_id = 5 THEN 'Astaka'
    END AS title,  -- Mengubah nama 'bundle_name' menjadi 'title'
    ubr.created_at AS last_updated,  -- Memilih 'created_at' sebagai 'last_updated'
    1200 AS duration,  -- Menambahkan kolom tetap 'duration' dengan nilai '1200'
    'Assessment' AS type,  -- Menambahkan kolom tetap 'type' dengan nilai 'Assessment'
    'Discovery' AS platform,  -- Menambahkan kolom tetap 'platform' dengan nilai 'Discovery'
    i.name AS institution,  -- Menambahkan kolom 'Institution'
    c.id AS 'Customer ID',  -- Menambahkan kolom 'customer_id'
    c.last_education AS last_education,  -- Menambahkan kolom 'Last Education'
    c.company as Company, -- Menambahkan kolom 'Company'
    c.gender as gender, -- Menambahkan kolom 'Gender'
    t.name AS 'Test Name',  -- Menambahkan kolom 'Test Name'
    ur.test_result_attribute->>'$[0].name' AS typology,  -- Menambahkan kolom 'typology' dari atribut 'test_result_attribute'
    LEFT(ur.total_score, 2) AS total_score,  -- Mengambil dua angka pertama dari 'total_score'
    ubr.result_name AS final_result  -- Menambahkan kolom 'final_result' tanpa koma di akhir
FROM user_results ur
LEFT JOIN tests t ON
    ur.test_id = t.id
JOIN users u ON
    ur.user_id = u.id
JOIN user_bundle_result_user_result ubrur ON
    ur.id = ubrur.user_result_id
JOIN user_bundle_results ubr ON
    ubrur.user_bundle_result_id = ubr.id
LEFT JOIN customers c ON
    u.id = c.user_id
LEFT JOIN institutions i ON
    c.institution_id = i.id
ORDER BY title ASC, u.name ASC, ubr.created_at DESC, ur.total_score DESC;
```

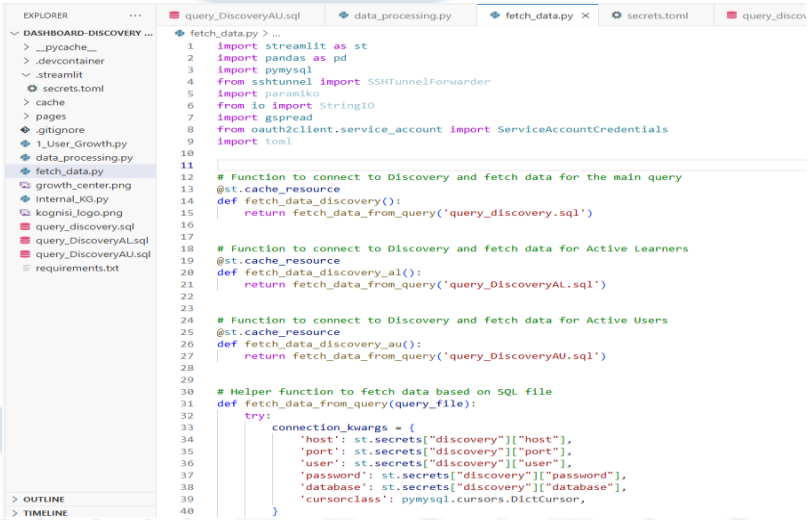
Gambar 3.6 Syntax Query Discovery AL



```sql
SELECT
    u.name,
    u.email,
    u.created_at AS 'created_at',
    'Discovery' AS platform,  -- Menambahkan kolom platform dengan nilai 'Discovery'
    CASE
        WHEN ubr.created_at IS NOT NULL THEN 'Active'  -- Jika terdapat Test Date maka status Active
        ELSE 'Passive'  -- Jika tidak ada Test Date maka status Passive
    END AS learner_status
FROM user_results ur
JOIN users u ON
    ur.user_id = u.id
LEFT JOIN user_bundle_result_user_result ubrur ON
    ur.id = ubrur.user_result_id
LEFT JOIN user_bundle_results ubr ON
    ubrur.user_bundle_result_id = ubr.id
ORDER BY u.name ASC, u.created_at DESC;
```

Gambar 3.7 Syntax Query Discovery AU

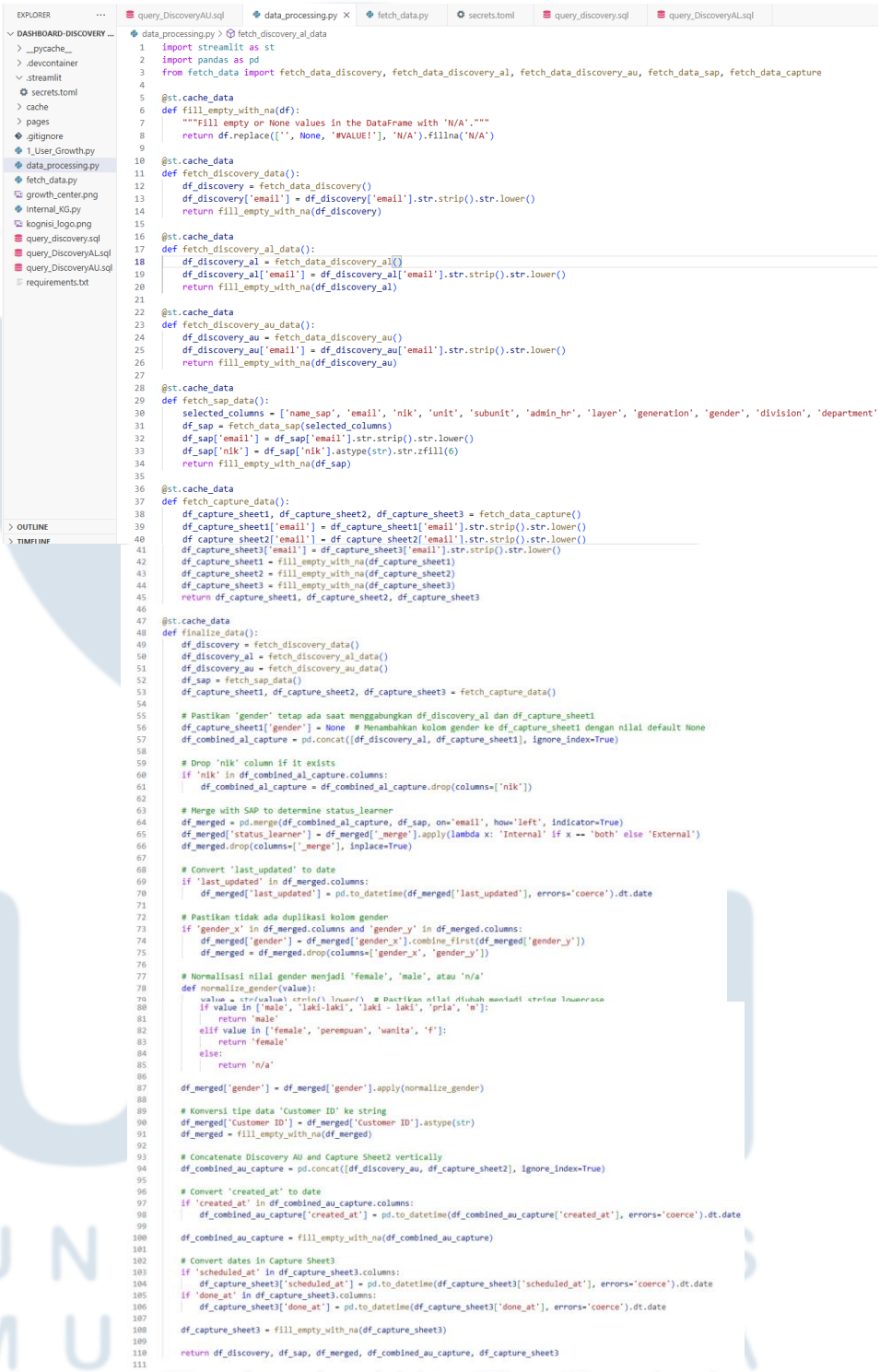
Setelah seluruh proses pemanggilan query dilakukan maka selanjutnya yaitu melakukan fetch data yang dimana merupakan proses pengambilan data dari suatu sumber seperti database, API ataupun file query yang telah dibuat sebelumnya. Hal ini biasanya melibatkan pengambilan data dari lokasi penyimpanan ekternal ke aplikasi agar data tersebut dapat diolah atau ditampilkan. Berikut syntax fetch data yang dilakukan pada pembuatan dashboard streamlit discovery terlihat pada Gambar 3.8 Syntax Fetch Data.



```python
import streamlit as st
import pandas as pd
import pymysql
from sshtunnel import SSHTunnelForwarder
import paramiko
from io import StringIO
import gspread
from oauth2client.service_account import ServiceAccountCredentials
import toml

# Function to connect to Discovery and fetch data for the main query
@st.cache_resource
def fetch_data_discovery():
    return fetch_data_from_query('query_discovery.sql')

# Function to connect to Discovery and fetch data for Active Learners
@st.cache_resource
def fetch_data_discovery_al():
    return fetch_data_from_query('query_DiscoveryAL.sql')

# Function to connect to Discovery and fetch data for Active Users
@st.cache_resource
def fetch_data_discovery_au():
    return fetch_data_from_query('query_DiscoveryAU.sql')

# Helper function to fetch data based on SQL file
def fetch_data_from_query(query_file):
    try:
        connection_kwargs = {
            'host': st.secrets["discovery"]["host"],
            'port': st.secrets["discovery"]["port"],
            'user': st.secrets["discovery"]["user"],
            'password': st.secrets["discovery"]["password"],
            'database': st.secrets["discovery"]["database"],
            'cursorclass': pymysql.cursors.DictCursor,
            }
```

Gambar 3.8 Syntax Fetch Data

Setelah proses fetch data dilakukan maka selanjutnya yaitu data processing sebagai tahapan akhir dalam melakukan connect data terlihat pada Gambar 3.9 Syntax Data Processing.



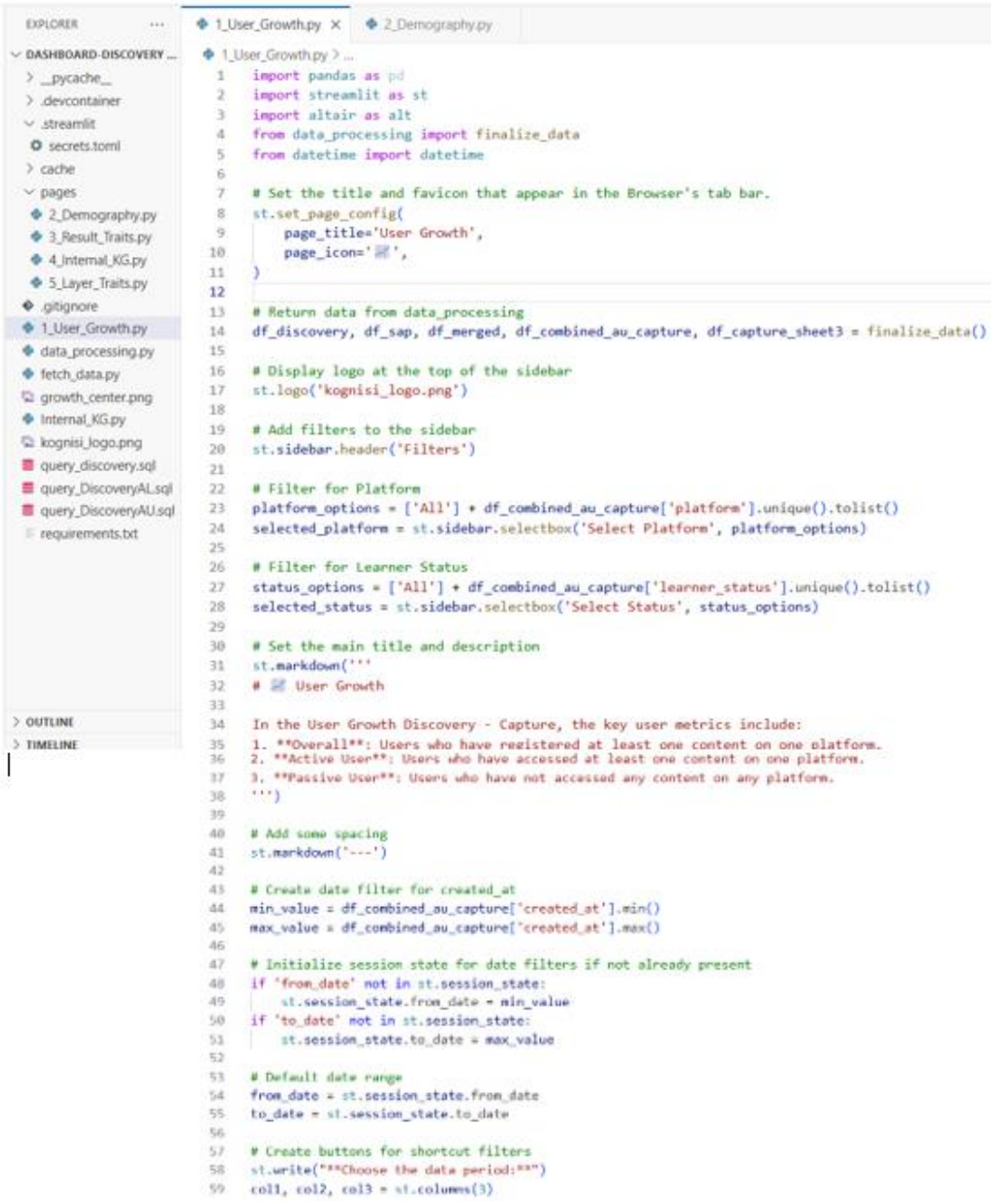

Gambar 3.9 Syntax Data Processing

Untuk perancangan dashboard streamlit discovery proses data processing dilakukan dengan tujuan utamanya yaitu:

a. Konsistensi data dengan memastikan format data seragam untuk mempermudah analisis dan penggabungan.
b. Penanganan data tidak valid dengan membersihkan data dari nilai kosong,error, atau tidak valid
c. Integrasi data dengan menggabungkan data dari berbagai sumber dengan cara yang logis dan terstruktur.
d. Kesiapan data untuk analisis yang dimana dilakukan konversi tipe data misalnya tanggal dan menstandarkan nilai kolom misalnya gender agar siap digunakan analisis atau visualisasi.
e. Pencegahan error hal ini dilakukan untuk menjamin data sudah bersih dan bebas dari duplikasi atau inkonsistensi yang dapat menyebabkan masalah saat digunakan pada aplikasi streamlit.

### 3.2.2 Page User Growth

Page user growth merupakan halaman pertama pada dashboard streamlit discovery yang dimana pada halaman ini dilakukan pada minggu ke – 3 dan minggu ke – 4 dengan membuat perancangan terlebih dahulu setelah perancangan selesai dilakukan maka akan dilanjutkan dengan implementasi yang dimana pada halaman user growth terdapat akumulasi seluruh active user secara real time baik itu active user maupun passive user dan pada halaman tersebut dapat di filter sesuai dengan yang diinginkan seperti life time, this year, this month, status maupun platform. Pada dashboard streamlit discovery halaman user growth jua menampilkan visualisasi seperti line chart register user distribustion dan bar chart platform distribution. Pada Gambar 3.10 Syntax Page User Gorwth memperlihatkan codingan pembuatan halaman tersebut.

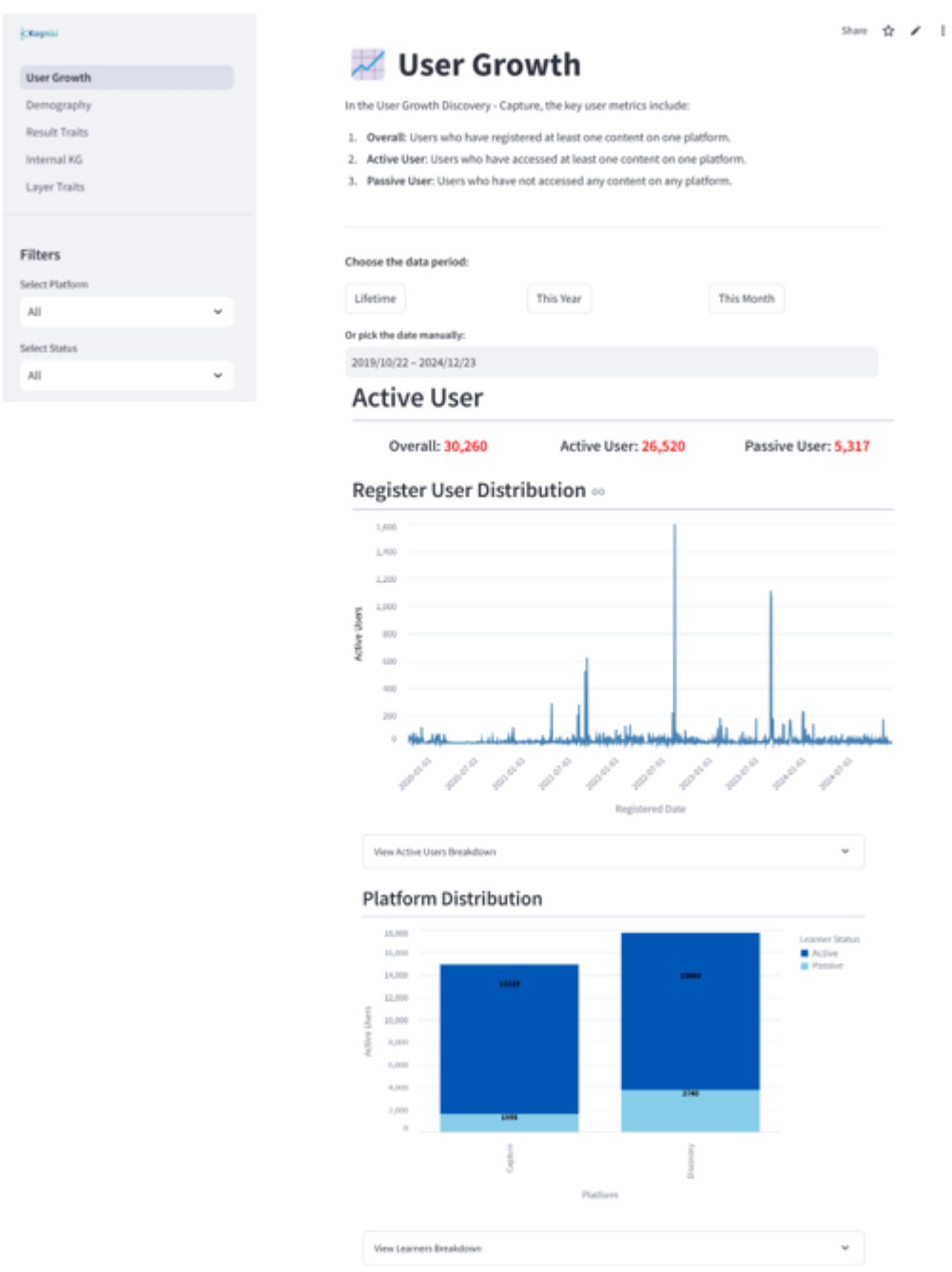


```python
with col1:
    if st.button('Lifetime'):
        st.session_state.from_date = min_value
        st.session_state.to_date = max_value

with col2:
    if st.button('This Year'):
        current_year = datetime.now().year
        st.session_state.from_date = datetime(current_year, 1, 1).date()
        st.session_state.to_date = datetime.now().date()

with col3:
    if st.button('This Month'):
        current_year = datetime.now().year
        current_month = datetime.now().month
        st.session_state.from_date = datetime(current_year, current_month, 1).date()
        st.session_state.to_date = datetime.now().date()

# Allow manual date input
from_date, to_date = st.date_input(
    '**Or pick the date manually:**',
    value=[st.session_state.from_date, st.session_state.to_date],
    min_value=min_value,
    max_value=max_value
)
st.session_state.from_date = from_date
st.session_state.to_date = to_date

# Filter the data based on the selected date range
filtered_df = df_combined_au_capture[
    (df_combined_au_capture['created_at'] <= to_date) &
    (df_combined_au_capture['created_at'] >= from_date)
]

# Further filter based on selected platform
if selected_platform != 'All':
    filtered_df = filtered_df[filtered_df['platform'] == selected_platform]

# Further filter based on selected learner status
if selected_status != 'All':
    filtered_df = filtered_df[filtered_df['learner_status'] == selected_status]

# Active Users section
st.header('Active User', divider='gray')

# Calculate the distinct counts of users based on email
total_count = filtered_df['email'].nunique()  # Total registered users (unique emails)
Active_count = filtered_df[filtered_df['learner_status'] == 'Active']['email'].nunique()  # Unique Active users
Passive_count = filtered_df[filtered_df['learner_status'] == 'Passive']['email'].nunique()  # Unique Ppassive users

# Display metrics column
col1, col2, col3 = st.columns(3)
with col1:
    st.markdown(f"<p style='font-size: 20px; text-align: center;'><strong>Overall: <span style='color: red;'>{total_count:,}</span></strong></p>",
                unsafe_allow_html=True)
with col2:
    st.markdown(f"<p style='font-size: 20px; text-align: center;'><strong>Active User: <span style='color: red;'>{Active_count:,}</span></strong></p>",
                unsafe_allow_html=True)
with col3:
    st.markdown(f"<p style='font-size: 20px; text-align: center;'><strong>Passive User: <span style='color: red;'>{Passive_count:,}</span></strong></p>",
                unsafe_allow_html=True)

# Register User Distribution
st.subheader('Register User Distribution', divider='gray')

# Create a line chart for active users based on created_at
active_user_counts = (filtered_df
                      .groupby('created_at')
```

```
                    .reset_index()

# Create a line chart for active learners
line_chart = alt.Chart(active_user_counts).mark_line(
    stroke='steelblue',  # Change the line color
    strokeWidth=2  # Increase line width for better visibility
).encode(
    x=alt.X('created_at:T', title='Registered Date',
            axis=alt.Axis(format='%Y-%m-%d', labelAngle=-45)),  # Tilt x-axis labels
    y=alt.Y('active_users:Q', title='Active Users', axis=alt.Axis(titleColor='black')),
    tooltip=[
        alt.Tooltip('created_at:T', title='Registered Date', format='%Y-%m-%d'),
        alt.Tooltip('active_users:Q', title='Active Users')
    ]
).properties(
    width=600,
    height=400
)

# Display the chart
st.altair_chart(line_chart, use_container_width=True)


# Breakdown data table for active users
with st.expander("View Active Users Breakdown"):
    st.dataframe(active_user_counts)

# Platform Distribution
st.subheader('Platform Distribution', divider='gray')

# Create a bar chart breakdown by platform and learner_status
platform_breakdown = (filtered_df
                      .groupby(['platform', 'learner_status'])
                      .agg(active_users=('email', 'nunique'))
                      .reset_index())
# Generate the stacked bar chart using Altair
bar_chart = alt.Chart(platform_breakdown).mark_bar().encode(
    x=alt.X('platform:O', title='Platform'),
    y=alt.Y('active_users:Q', title='Active Users'),
    color=alt.Color('learner_status:N', title='Learner Status', scale=alt.Scale(domain=['Active', 'Passive'], range=['#0056b3', '#87CEEB'])),
    tooltip=[
        alt.Tooltip('platform:O', title='Platform'),
        alt.Tooltip('learner_status:N', title='Learner Status'),
        alt.Tooltip('active_users:Q', title='Active Users')
    ]
).properties(
    width=600,
    height=400
)

# Add a text layer to display active user counts within each color section
text_active = bar_chart.mark_text(
    align='center',
    baseline='middle',
    dy=2,  # Positioning text slightly inside the active (blue) bar
    fontWeight='bold'
).transform_filter(
    alt.datum.learner_status == 'Active'
).encode(
    text=alt.Text('active_users:Q'),
    color=alt.value('black')  # White text for better contrast on the dark blue bar
)
text_passive = bar_chart.mark_text(
    align='center',
    baseline='middle',
    dy=5,  # Positioning text slightly inside the passive (light blue) bar
    fontWeight='bold'
).transform_filter(
    alt.datum.learner_status == 'Passive'
).encode(
    text=alt.Text('active_users:Q'),
    color=alt.value('black')  # Black text for better contrast on the light blue bar
)

# Layer the bar chart with the text labels for both active and passive users
layered_chart = bar_chart + text_active + text_passive

# Display the bar chart in Streamlit
st.altair_chart(layered_chart, use_container_width=True)

# Breakdown data table for active users
with st.expander("View Learners Breakdown"):
    st.dataframe(platform_breakdown)

```

Gambar 3.10 Syntax Page User Growth

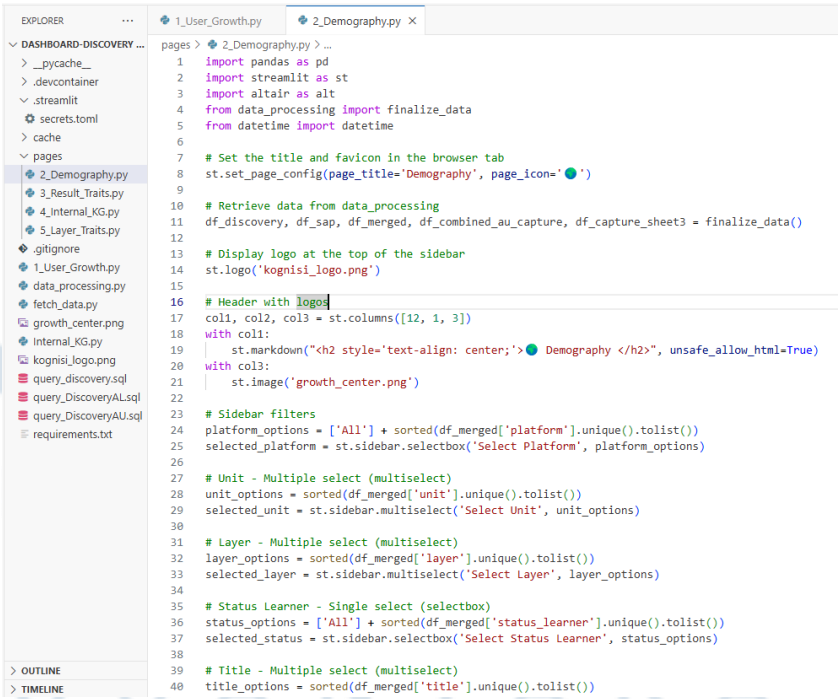
Setelah syntax tersebut dijalankan maka halaman user growth pada dashboard streamlit terlihat seperti tampilan pada Gambar 3.11 Output Page User Growth.




Gambar 3.11 Output Page User Growth

### 3.2.3 Page Demography

Page demography merupakan halaman kedua pada dashboard streamlit discovery yang dimana pada halaman ini dilakukan pada minggu ke – 5 sampai dengan minggu ke – 7 dengan membuat perancangan terlebih dahulu setelah perancangan selesai dilakukan maka akan dilanjutkan dengan implementasi yang dimana pada halaman demography terdapat akumulasi seluruh active user secara real time baik itu active user maupun passive user dan pada halaman tersebut dapat di filter sesuai dengan yang diingin seperti life time, this year, this month, platform, unit, layer, status, title dan company. Pada dashboard streamlit discovery halaman user growth juga menampilkan visualisasi seperti line chart test date distribustion dan bar chart demography distribution yang nantinya dapat difilter sesuai keinginan stakeholder. Breakdown dalam demography mencakup unit, layer, gender, company, generation, institution, dan last education. Pada Gambar 3.12 Syntax Page Demography memperlihatkan codingan pembuatan halaman tersebut.



```
selected_title = st.sidebar.multiselect('Select Title', title_options)

# Company - Multiple select (multiselect)
company_options = sorted(df_merged['Company'].unique().tolist())
selected_company = st.sidebar.multiselect('Select Company', company_options)

# Main title and description
st.markdown('''
In the Active Learners Growth Discovery - Capture, the key user metrics include:
1. **Overall**: Users test on the Discovery and Capture platforms.
2. **Internal User**: Active learners who are Kompas Gramedia Group employees.
3. **External User**: Active learners who are not Kompas Gramedia Group employees.
''')
st.markdown('---')

# Date filter setup
min_value, max_value = df_merged['last_updated'].min(), df_merged['last_updated'].max()

# Initialize session state for date filters
if 'from_date' not in st.session_state:
    st.session_state.from_date, st.session_state.to_date = min_value, max_value

# Shortcut buttons for date selection
st.write("**Choose the data period:**")
col1, col2, col3 = st.columns(3)
with col1:
    if st.button('Lifetime'):
        st.session_state.from_date, st.session_state.to_date = min_value, max_value
with col2:
    if st.button('This Year'):
        current_year = datetime.now().year
        st.session_state.from_date, st.session_state.to_date = datetime(current_year, 1, 1).date(), datetime.now().date()
with col3:
    if st.button('This Month'):
        current_year, current_month = datetime.now().year, datetime.now().month
        st.session_state.from_date, st.session_state.to_date = datetime(current_year, current_month, 1).date(), datetime.now().date()

# Ensure date selections are within valid range
st.session_state.from_date = max(st.session_state.from_date, min_value)
st.session_state.to_date = min(st.session_state.to_date, max_value)

# Manual date input
from_date, to_date = st.date_input(
    '**Or pick the date manually:**',
    value=(st.session_state.from_date, st.session_state.to_date),
    min_value=min_value,
    max_value=max_value
)

# Update session state with manual input
st.session_state.from_date, st.session_state.to_date = from_date, to_date

# Data filtering based on selected dates and platform
filtered_df = df_merged[
    (df_merged['last_updated'] >= from_date) &
    (df_merged['last_updated'] <= to_date)
]

if selected_platform != 'All':
    filtered_df = filtered_df[filtered_df['platform'] == selected_platform]

if selected_status != 'All':
    filtered_df = filtered_df[filtered_df['status_learner'] == selected_status]

if selected_unit:
    filtered_df = filtered_df[filtered_df['unit'].isin(selected_unit)]
```

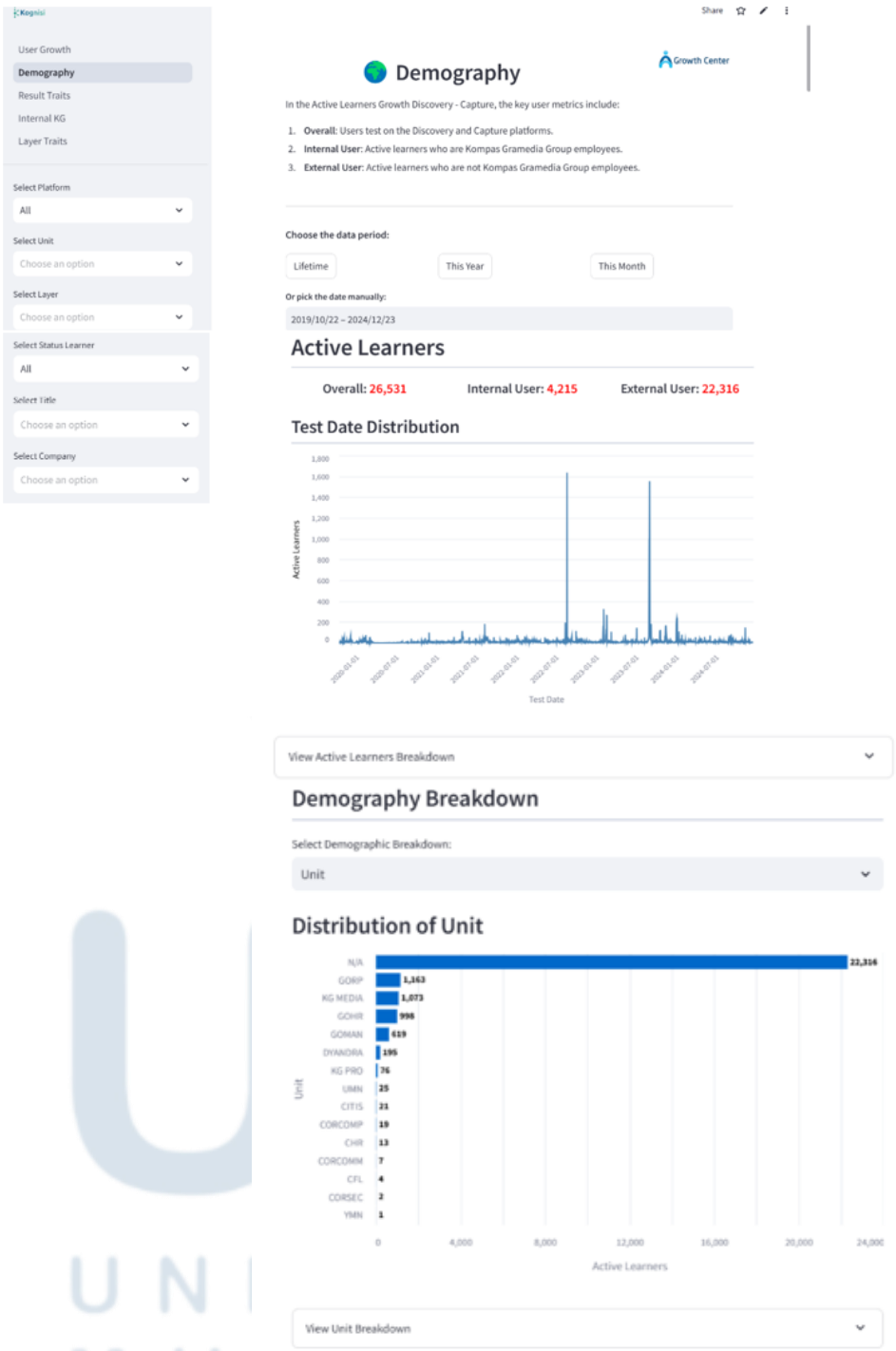
```python
if selected_layer:
    filtered_df = filtered_df[filtered_df['layer'].isin(selected_layer)]

if selected_title:
    filtered_df = filtered_df[filtered_df['title'].isin(selected_title)]

if selected_company:
    filtered_df = filtered_df[filtered_df['Company'].isin(selected_company)]

# Active Users section
st.header('Active Learners', divider='gray')
total_count = filtered_df['email'].nunique()
internal_count = filtered_df[filtered_df['status_learner'] == 'Internal']['email'].nunique()
external_count = filtered_df[filtered_df['status_learner'] == 'External']['email'].nunique()

# Display metrics columns
col1, col2, col3 = st.columns(3)
with col1:
    st.markdown(f"<p style='font-size: 20px; text-align: center;'><strong>Overall: <span style='color: red;'>{total_count:,}</span></strong></p>",
                unsafe_allow_html=True)
with col2:
    st.markdown(f"<p style='font-size: 20px; text-align: center;'><strong>Internal User: <span style='color: red;'>{internal_count:,}</span></strong></p>",
                unsafe_allow_html=True)
with col3:
    st.markdown(f"<p style='font-size: 20px; text-align: center;'><strong>External User: <span style='color: red;'>{external_count:,}</span></strong></p>",
                unsafe_allow_html=True)

# Register User Distribution
st.subheader('Test Date Distribution', divider='gray')

# Count unique active learners by test date
active_learners_counts = (filtered_df
                          .groupby('last_updated')
                          .agg(active_learners=('email', 'nunique'))
                          .reset_index())

# Create a line chart for active learners
line_chart = alt.Chart(active_learners_counts).mark_line(
    stroke='steelblue',  # Change the line color
    strokeWidth=2  # Increase line width for better visibility
).encode(
    x=alt.X('last_updated:T', title='Test Date',
            axis=alt.Axis(format='%Y-%m-%d', labelAngle=-45)),  # Tilt x-axis labels
    y=alt.Y('active_learners:Q', title='Active Learners', axis=alt.Axis(titleColor='black')),
    tooltip=[
        alt.Tooltip('last_updated:T', title='Test Date', format='%Y-%m-%d'),
        alt.Tooltip('active_learners:Q', title='Active Learners')
    ]
).properties(
    width=600,
    height=400
)

# Display the chart
st.altair_chart(line_chart, use_container_width=True)


# Active learners data table
with st.expander("View Active Learners Breakdown"):
    st.dataframe(active_learners_counts)

# Demography Breakdown
st.subheader('Demography Breakdown', divider='gray')
breakdown_options = ['Unit', 'Layer', 'Gender', 'Company', 'Generation', 'Institution', 'Last Education']
selected_breakdown = st.selectbox('Select Demographic Breakdown:', breakdown_options)

# Helper function to create a bar chart with text
def create_bar_chart_with_text(data, x_col, y_col, title):
```

```
    bar_chart = alt.Chart(data).mark_bar().encode(
        x=alt.X(f'{x_col}:Q', title='Active Learners'),
        y=alt.Y(f'{y_col}:O', title=title, sort='-x'),
        tooltip=[
            alt.Tooltip(f'{y_col}:O', title=title),
            alt.Tooltip(f'{x_col}:Q', title='Active Learners')
        ]
    ).properties(width=600, height=400)

    # Text layer for counts
    text = bar_chart.mark_text(
        align='left',
        fontWeight='bold',
        baseline='middle',
        dx=3  # Adjusts position of the text to fit nicely
    ).encode(
        text=alt.Text(f'{x_col}:Q', format=',')  # Display counts
    )

    return bar_chart + text

# Breakdown visualization based on selected demographic option
breakdown_mapping = {
    'Unit': 'unit',
    'Layer': 'layer',
    'Generation': 'generation',
    'Institution': 'institution',
    'Last Education': 'last_education',
    'Company': 'Company'
}

if selected_breakdown in breakdown_mapping:
    # Menghitung jumlah active learners berdasarkan breakdown yang dipilih
    counts = (
        filtered_df
        .groupby(['status_learner', breakdown_mapping[selected_breakdown]])
        .agg(active_learners=('email', 'nunique'))
        .reset_index()
    )

    # Penanganan untuk Institution, Last Education, dan Company
    if selected_breakdown in ['Institution', 'Last Education', 'Company']:
        # Hitung total active learners untuk setiap kategori
        total_counts = (
            counts.groupby(breakdown_mapping[selected_breakdown])
            .agg(total_active_learners=('active_learners', 'sum'))
            .reset_index()
        )
        # Gabungkan total dengan data utama
        counts = counts.merge(total_counts, on=breakdown_mapping[selected_breakdown])
        # Hitung persentase untuk 100% stacked bar chart
        counts['percentage'] = (counts['active_learners'] / counts['total_active_learners']) * 100

        # Batasi hanya 10 data teratas berdasarkan total_active_learners untuk Institution
        if selected_breakdown == 'Institution':
            counts = counts.sort_values(by='total_active_learners', ascending=False).head(10)
            st.subheader("Top 10 Institution")  # Subheader khusus untuk Institution
```

```
        else:
            st.subheader(f"Distribution of {selected_breakdown}")  # Subheader umum untuk Last Education dan Company

        # Visualisasi stacked bar chart
        stacked_bar_chart = alt.Chart(counts).mark_bar().encode(
            x=alt.X('percentage:Q', title='Active Learners (%)'),
            y=alt.Y(f'{breakdown_mapping[selected_breakdown]}:N', title=selected_breakdown, sort='-x'),
            color=alt.Color(
                'status_learner:N',
                title='User Type',
                scale=alt.Scale(domain=["Internal", "External"], range=['blue', 'orange'])
            ),
            tooltip=[
                alt.Tooltip(f'{breakdown_mapping[selected_breakdown]}:N', title=selected_breakdown),
                alt.Tooltip('active_learners:Q', title='Active Learners'),
                alt.Tooltip('status_learner:N', title='Status Learner')
            ]
        ).properties(width=600, height=400)

        st.altair_chart(stacked_bar_chart, use_container_width=True)

        # Menampilkan data dalam expander
        with st.expander(f"View {selected_breakdown} Breakdown"):
            st.dataframe(counts)

    # Penanganan untuk breakdown lain (selain Institution, Last Education, dan Company)
    else:
        # Tambahkan subheader untuk Unit, Layer, Generation
        st.subheader(f"Distribution of {selected_breakdown}")

        # Mengurutkan berdasarkan jumlah active learners dalam urutan menurun
        counts = counts.sort_values(by='active_learners', ascending=False)

        # Visualisasi bar chart
        st.altair_chart(
            create_bar_chart_with_text(counts, 'active_learners', breakdown_mapping[selected_breakdown], selected_breakdown),
            use_container_width=True
        )

        # Menampilkan data dalam expander
        with st.expander(f"View {selected_breakdown} Breakdown"):
            st.dataframe(counts)

# Penanganan khusus untuk Gender
elif selected_breakdown == 'Gender':
    # Menghitung jumlah active learners berdasarkan gender
    gender_counts = (
        filtered_df
        .groupby(['status_learner', 'gender'])
        .agg(active_learners=('email', 'nunique'))
        .reset_index()
        .sort_values(by='active_learners', ascending=False)
    )

    # Subheader untuk Gender Distribution
    st.subheader("Distribution of Gender")

    # Visualisasi pie chart untuk Gender
    pie_chart_gender = alt.Chart(gender_counts).mark_arc().encode(
        theta=alt.Theta('active_learners:Q', title='Active Learners'),
        color=alt.Color('gender:N', title='Gender'),
        tooltip=[
            alt.Tooltip('gender:N', title='Gender'),
            alt.Tooltip('active_learners:Q', title='Active Learners')
        ]
    ).properties(width=600, height=400)

    st.altair_chart(pie_chart_gender, use_container_width=True)

    # Menampilkan data dalam expander
    with st.expander("View Gender Breakdown"):
        st.dataframe(gender_counts)
```

Gambar 3.12 Syntax Page Demography

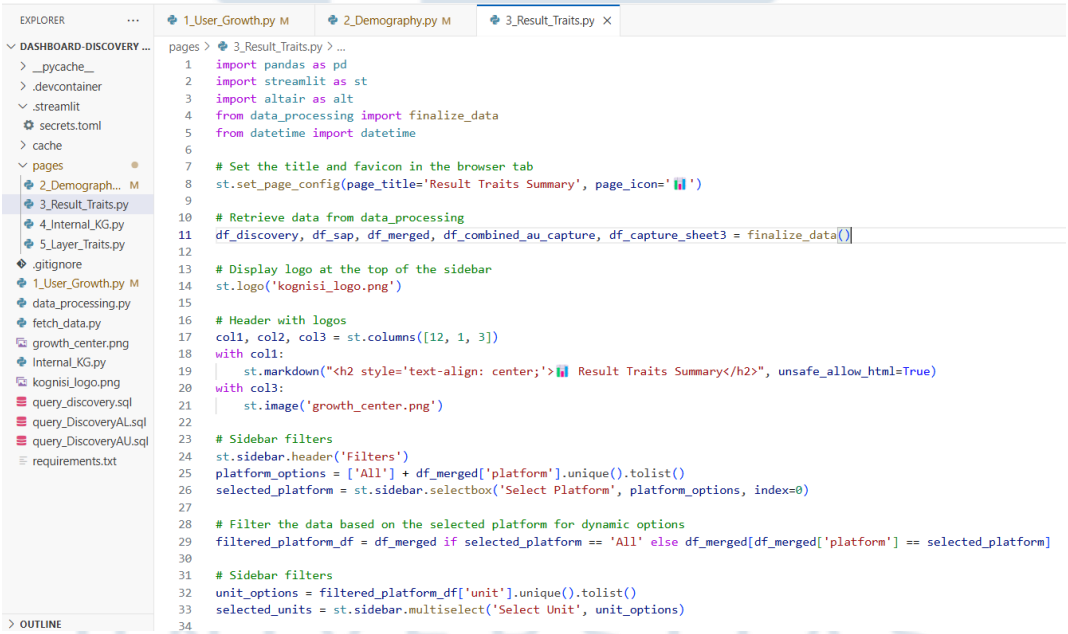
Setelah syntax tersebut dijalankan maka halaman demography pada dashboard streamlit terlihat seperti tampilan pada Gambar 3.13 Output Page Demography




Gambar 3.13 Output Page Demography

### 3.2.4 Page Result Traits

Page result traits merupakan halaman ketiga pada dashboard streamlt discovery yang dimana pada halaman ini dilakukan pada minggu ke – 8 sampai dengan minggu ke – 10 dengan membuat perancangan terlebih dahulu setelah perancangan selesai dilakukan maka akan dilanjutkan dengan implementasi yang dimana pada halaman result traits terdapat filter sesuai dengan yang diinginkan seperti life time, this year, this month, platform, unit, layer, status, title dan company. Pada dashboard streamlit discovery halaman result traits juga menampilkan visualisasi seperti bar chart platform distribution pada setiap unit yang ada pada perusahaan Kompas Gramedia Group serta yang nantinya dapat difilter sesuai keinginan stakeholder. Pada Gambar 3.14 Syntax Result Traits memperlihatkan codingan pembuatan halaman tersebut.



```python
layer_options = filtered_platform_df['layer'].unique().tolist()
selected_layers = st.sidebar.multiselect('Select Layer', layer_options)

status_options = ['All'] + filtered_platform_df['status_learner'].unique().tolist()
selected_status = st.sidebar.selectbox('Select Status Learner', status_options)

title_options = filtered_platform_df['title'].unique().tolist()
selected_titles = st.sidebar.multiselect('Select Title', title_options)

company_options = filtered_platform_df['Company'].unique().tolist()
selected_company = st.sidebar.multiselect('Select Company', company_options)

# Date filter setup
min_value, max_value = df_merged['last_updated'].min(), df_merged['last_updated'].max()

# Initialize session state for date filters
if 'from_date' not in st.session_state:
    st.session_state.from_date, st.session_state.to_date = min_value, max_value

# Shortcut buttons for date selection
st.write("**Choose the data period:**")
col1, col2, col3 = st.columns(3)
with col1:
    if st.button('Lifetime'):
        st.session_state.from_date, st.session_state.to_date = min_value, max_value
with col2:
    if st.button('This Year'):
        current_year = datetime.now().year
        st.session_state.from_date, st.session_state.to_date = datetime(current_year, 1, 1).date(), datetime.now().date()
with col3:
    if st.button('This Month'):
        current_year, current_month = datetime.now().year, datetime.now().month
        st.session_state.from_date, st.session_state.to_date = datetime(current_year, current_month, 1).date(), datetime.now().date()
# Ensure date selections are within valid range
st.session_state.from_date = max(st.session_state.from_date, min_value)
st.session_state.to_date = min(st.session_state.to_date, max_value)

# Manual date input
from_date, to_date = st.date_input(
    '**Or pick the date manually:**',
    value=(st.session_state.from_date, st.session_state.to_date),
    min_value=min_value,
    max_value=max_value
)

# Update session state with manual input
st.session_state.from_date, st.session_state.to_date = from_date, to_date

# Data filtering based on selected filters
filtered_df = df_merged[
    (df_merged['last_updated'] <= to_date) &
    (df_merged['last_updated'] >= from_date)
]

if selected_platform != 'All':
    filtered_df = filtered_df[filtered_df['platform'] == selected_platform]
if selected_status != 'All':
    filtered_df = filtered_df[filtered_df['status_learner'] == selected_status]
if selected_units:
    filtered_df = filtered_df[filtered_df['unit'].isin(selected_units)]
if selected_layers:
    filtered_df = filtered_df[filtered_df['layer'].isin(selected_layers)]
if selected_titles:
    filtered_df = filtered_df[filtered_df['title'].isin(selected_titles)]
if selected_company:
    filtered_df = filtered_df[filtered_df['Company'].isin(selected_company)]

# Display horizontal stacked bar chart of platform per unit if 'All' is selected for platform
if selected_platform == 'All':
    st.write("### Platform Distribution")

    # Group data to calculate counts and unique emails
    platform_unit_df = (
        filtered_df.groupby(['unit', 'platform'])
        .agg(count=('email', 'size'), Active_Learners=('email', 'nunique'))
        .reset_index()
    )

    # Calculate total counts per unit
    platform_unit_df['total'] = platform_unit_df.groupby('unit')['count'].transform('sum')
    platform_unit_df['percent'] = platform_unit_df['count'] / platform_unit_df['total'] * 100

```

```python
    # Sort units by total count of all platforms
    sorted_units = (
        platform_unit_df.groupby('unit')['count']
        .sum()
        .sort_values(ascending=False)
        .index.tolist()
    )

    # Create the 100% stacked horizontal bar chart with count labels
    platform_unit_chart = (
        alt.Chart(platform_unit_df)
        .mark_bar()
        .encode(
            y=alt.Y('unit:N', sort=sorted_units, title='Unit'),
            x=alt.X('percent:Q', stack="normalize", title='Percentage (%)'),
            color=alt.Color('platform:N', title='Platform'),
            tooltip=['platform', 'unit', 'Active_Learners']
        )
        .properties(width=600, height=400)
    )

    # Combine the bar chart and the text labels
    st.altair_chart(platform_unit_chart, use_container_width=True)

    # Data download for Growth Inventory
    with st.expander("Data Distribution Platform"):
        st.write(platform_unit_df)
        csv = platform_unit_df.to_csv(index=False).encode('utf-8')
        st.download_button("Download Data", data=csv, file_name="Distribution_Platform.csv", mime="text/csv",
                           help='Click here to download the data as a CSV file')


# Active Users section for Discovery
if selected_platform == 'Discovery':
    st.header('Active Learners - Discovery', divider='gray')
    total_count = filtered_df['email'].nunique()
    internal_count = filtered_df[filtered_df['status_learner'] == 'Internal']['Customer ID'].nunique()
    external_count = filtered_df[filtered_df['status_learner'] == 'External']['Customer ID'].nunique()

    # Display metrics columns
    col1, col2, col3 = st.columns(3)
    with col1:
        st.markdown(f"<p style='font-size: 20px; text-align: center;'><strong>Overall: <span style='color: red;'>{total_count:,}</span></strong></p>",
                    unsafe_allow_html=True)
    with col2:
        st.markdown(f"<p style='font-size: 20px; text-align: center;'><strong>Internal User: <span style='color: red;'>{internal_count:,}</span></strong></p>",
                    unsafe allow html=True)
    with col3:
        st.markdown(f"<p style='font-size: 20px; text-align: center;'><strong>External User: <span style='color: red;'>{external_count:,}</span></strong></p>",
                    unsafe_allow_html=True)

    # Active learners by bundle_name
    bundle_names = ['GI', 'LEAN', 'ELITE', 'Genuine', 'Astaka']
    if 'title' in filtered_df.columns:
        # Group by bundle_name and count unique Customer IDs
        df_active_learners = filtered_df.groupby(['Customer ID', 'title']).size().reset_index(name='test_count')
        bundle_counts = {bundle: df_active_learners[df_active_learners['title'] == bundle]['Customer ID'].nunique() for bundle in bundle_names}

        # Display active learners counts
        st.markdown("<h3>ACTIVE LEARNERS by Bundle</h3>", unsafe_allow_html=True)
        col1, col2, col3, col4, col5 = st.columns(5)
        for i, bundle in enumerate(bundle_names):
            with eval(f'col{i + 1}'):
                st.markdown(f"<p style='font-size: 20px; text-align: center;'><strong>{bundle}: <span style='color: red;'>{bundle_counts.get(bundle, 0):,}
                            </span></strong></p>", unsafe_allow_html=True)

        # Create bar chart for top bundles
        bundle_chart_data = pd.DataFrame.from_dict(bundle_counts, orient='index', columns=['count']).reset_index()
        bundle_chart_data.columns = ['Bundle', 'Active Learners']

        # Create the bar chart using Altair
        # Stacked bar chart for Growth Inventory
        st.subheader("Top Test Discovery")
        bundle_bar_chart = (
            alt.Chart(bundle_chart_data)
            .mark_bar()
            .encode(
                x=alt.X('Active Learners:Q', title='Active Learners Count'),
                y=alt.Y('Bundle:N', title='Bundle', sort='-x'),
                color=alt.Color('Bundle:N', title='Bundle'),
                tooltip=['Bundle:N', 'Active Learners:Q']
            )
            .properties(width=600, height=400)
        )

        # Display the chart
        st.altair_chart(bundle_bar_chart, use_container_width=True)

        # Data download for Top Test Discovery
        with st.expander("Top Test Discovery"):
            st.write(bundle_chart_data)
            csv = bundle_chart_data.to_csv(index=False).encode('utf-8')
            st.download_button("Download Data", data=csv, file_name="Top_Test_Discovery.csv", mime="text/csv", help='Click here to download the data as a CSV file')
```

```python
# Active learners data for Bundle GI
gi_active_learners = filtered_df[filtered_df['title'] == 'GI']

# Get highest scores
highest_scores = gi_active_learners.loc[gi_active_learners.groupby(['email', 'last_updated', 'Test Name'])['total_score'].idxmax()]
gi_active_learners_data = highest_scores[['name', 'email', 'Customer ID', 'title', 'last_updated', 'Test Name', 'total_score', 'final_result']]

# Stacked bar chart for Growth Inventory
st.subheader("Growth Inventory")
gi_filtered = filtered_df[filtered_df['title'] == 'GI']
gi_distribution = gi_filtered.groupby(['Test Name', 'typology']).agg({'Customer ID': 'nunique'}).reset_index()
gi_distribution.columns = ['Test Name', 'typology', 'Active Users']

# Calculate percentages
total_active_users_per_test = gi_distribution.groupby('Test Name')['Active Users'].transform('sum')
gi_distribution['Percentage'] = (gi_distribution['Active Users'] / total_active_users_per_test * 100).round(2)

# Plot chart
chart = alt.Chart(gi_distribution).mark_bar().encode(
    x='Test Name',
    y='Active Users',
    color='typology',
    tooltip=[
        alt.Tooltip('Test Name:N', title='Test Name'),
        alt.Tooltip('typology:N', title='Typology'),
        alt.Tooltip('Active Users:Q', title='Active Learners'),
        alt.Tooltip('Percentage:Q', title='Percentage', format='.1f')  # Format percentage with one decimal place
    ]
).properties(
    width=600,
    height=400
).configure_mark(
    opacity=0.8
).configure_axis(
    labelFontSize=12,
    titleFontSize=14
).configure_title(
    fontSize=16
)

st.altair_chart(chart, use_container_width=True)

# Data download for Growth Inventory
with st.expander("Data Growth Inventory"):
    st.write(gi_distribution)
    csv = gi_distribution.to_csv(index=False).encode('utf-8')
    st.download_button("Download Data", data=csv, file_name="Growth_Inventory.csv", mime="text/csv", help='Click here to download the data as a CSV file')


# Repeat for LEAN
lean_active_learners = filtered_df[filtered_df['title'] == 'LEAN']

# Get highest scores for LEAN
highest_scores_lean = lean_active_learners.loc[lean_active_learners.groupby(['email', 'last_updated', 'Test Name'])['total_score'].idxmax()]
lean_active_learners_data = highest_scores_lean[['name', 'email', 'Customer ID', 'title', 'last_updated', 'Test Name', 'total_score', 'final_result']]

# Stacked bar chart for LEAN
st.subheader("LEAN")
lean_filtered = filtered_df[filtered_df['title'] == 'LEAN']
lean_distribution = lean_filtered.groupby(['Test Name', 'typology']).agg({'Customer ID': 'nunique'}).reset_index()
lean_distribution.columns = ['Test Name', 'typology', 'Active Users']

# Calculate percentages for LEAN
total_active_users_per_test_lean = lean_distribution.groupby('Test Name')['Active Users'].transform('sum')
lean_distribution['Percentage'] = (lean_distribution['Active Users'] / total_active_users_per_test_lean * 100).round(2)

# Plot chart for LEAN
chart_lean = alt.Chart(lean_distribution).mark_bar().encode(
    x='Test Name',
    y='Active Users',
    color='typology',
    tooltip=[
        alt.Tooltip('Test Name:N', title='Test Name'),
        alt.Tooltip('typology:N', title='Typology'),
        alt.Tooltip('Active Users:Q', title='Active Learners'),
        alt.Tooltip('Percentage:Q', title='Percentage', format='.1f')  # Format percentage with one decimal place
    ]
).properties(
    width=600,
    height=400
).configure_mark(
    opacity=0.8
).configure_axis(
    labelFontSize=12,
    titleFontSize=14
).configure_title(
    fontSize=16
)

st.altair_chart(chart_lean, use_container_width=True)

# Data download for LEAN
with st.expander("Data LEAN"):
    st.write(lean_distribution)
    csv_lean = lean_distribution.to_csv(index=False).encode('utf-8')
    st.download_button("Download Data", data=csv_lean, file_name="LEAN.csv", mime="text/csv", help='Click here to download the data as a CSV file')
```

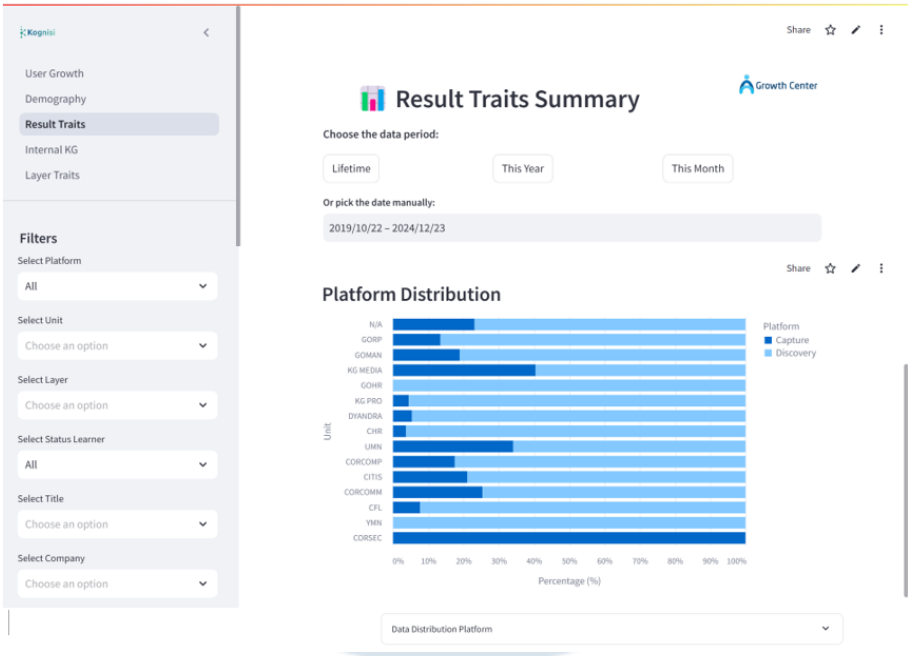
```
    # Display final result distribution
    st.subheader("FINAL RESULT LEAN")
    lean_active_learners = filtered_df[filtered_df['title'] == 'LEAN']

    # Get highest scores for LEAN
    highest_scores_lean = lean_active_learners.loc[lean_active_learners.groupby(['email', 'last_updated', 'Test Name'])['total_score'].idxmax()]
    lean_active_learners_data = highest_scores_lean[['name', 'email', 'Customer ID', 'title', 'last_updated', 'Test Name', 'total_score', 'final_result']]

    # Get unique combinations of Customer ID and Test Date
    unique_combinations = lean_active_learners_data[['Customer ID', 'last_updated']].drop_duplicates()

    # Merge back to get final_result
    unique_results = unique_combinations.merge(lean_active_learners_data[['Customer ID', 'last_updated', 'final_result']],
                                               on=['Customer ID', 'last_updated'],
                                               how='left').drop_duplicates()

    # Aggregate final_result for pie chart
    final_result_counts = unique_results['final_result'].value_counts().reset_index()
    final_result_counts.columns = ['Final Result', 'Count']

    # Calculate percentage
    final_result_counts['Percentage'] = (final_result_counts['Count'] / final_result_counts['Count'].sum()) * 100

    # Plot pie chart for final results
    pie_chart = alt.Chart(final_result_counts).mark_arc().encode(
        theta='Count:Q',
        color='Final Result:N',
        tooltip=[
            alt.Tooltip('Final Result:N', title='Final Result'),
            alt.Tooltip('Count:Q', title='Active Learners'),
            alt.Tooltip('Percentage:Q', title='Percentage', format='.1f')  # Format percentage with one decimal place
        ]
    ).properties(
        width=400,
        height=400
    )

    st.altair_chart(pie_chart, use_container_width=True)

    # Expander for final result data
    with st.expander("View Final Result Data"):
        st.write(final_result_counts)
        csv_final = final_result_counts.to_csv(index=False).encode('utf-8')  # Convert to CSV
        st.download_button(
            label="Download Final Result Data",
            data=csv_final,
            file_name="Final_Results_LEAN.csv",
            mime="text/csv",
            help='Click here to download the final result data as a CSV file'
    )

    # Repeat for ELITE
    elite_active_learners = filtered_df[filtered_df['title'] == 'ELITE']

    # Get highest scores for ELITE
    highest_scores_elite = elite_active_learners.loc[elite_active_learners.groupby(['email', 'last_updated', 'Test Name'])['total_score'].idxmax()]
    elite_active_learners_data = highest_scores_elite[['name', 'email', 'Customer ID', 'title', 'last_updated', 'Test Name', 'total_score', 'final_result']]

    # Stacked bar chart for ELITE
    st.subheader("ELITE")
    elite_filtered = filtered_df[filtered_df['title'] == 'ELITE']
    elite_distribution = elite_filtered.groupby(['Test Name', 'typology']).agg({'Customer ID': 'nunique'}).reset_index()
    elite_distribution.columns = ['Test Name', 'typology', 'Active Users']

    # Calculate percentages for ELITE
    total_active_users_per_test_elite = elite_distribution.groupby('Test Name')['Active Users'].transform('sum')
    elite_distribution['Percentage'] = (elite_distribution['Active Users'] / total_active_users_per_test_elite * 100).round(2)

    # Plot chart for ELITE
    chart_elite = alt.Chart(elite_distribution).mark_bar().encode(
        x='Test Name',
        y='Active Users',
        color='typology',
        tooltip=[
            alt.Tooltip('Test Name:N', title='Test Name'),
            alt.Tooltip('typology:N', title='Typology'),
            alt.Tooltip('Active Users:Q', title='Active Learners'),
            alt.Tooltip('Percentage:Q', title='Percentage', format='.1f')  # Format percentage with one decimal place
        ]
    ).properties(
        width=600,
        height=400
    ).configure_mark(
        opacity=0.8
    ).configure_axis(
        labelFontSize=12,
        titleFontSize=14
    ).configure_title(
```

```
    fontSize=16
)

st.altair_chart(chart_elite, use_container_width=True)

# Data download for ELITE
with st.expander("Data ELITE"):
    st.write(elite_distribution)
    csv_elite = elite_distribution.to_csv(index=False).encode('utf-8')
    st.download_button("Download Data", data=csv_elite, file_name="ELITE.csv", mime="text/csv", help='Click here to download the data as a CSV file')

# Display final result distribution
st.subheader("FINAL RESULT ELITE")
elite_active_learners = filtered_df[filtered_df['title'] == 'ELITE']

# Get highest scores for ELITE
highest_scores_elite = elite_active_learners.loc[elite_active_learners.groupby(['email', 'last_updated', 'Test Name'])['total_score'].idxmax()]
elite_active_learners_data = highest_scores_elite[['name', 'email', 'Customer ID', 'title', 'last_updated', 'Test Name', 'total_score', 'final_result']]

# Get unique combinations of Customer ID and Test Date
unique_combinations = elite_active_learners_data[['Customer ID', 'last_updated']].drop_duplicates()

# Merge back to get final_result
unique_results = unique_combinations.merge(elite_active_learners_data[['Customer ID', 'last_updated', 'final_result']],
                                           on=['Customer ID', 'last_updated'],
                                           how='left').drop_duplicates()

# Aggregate final_result for pie chart
final_result_counts = unique_results['final_result'].value_counts().reset_index()
final_result_counts.columns = ['Final Result', 'Count']

# Calculate percentage
final_result_counts['Percentage'] = (final_result_counts['Count'] / final_result_counts['Count'].sum()) * 100

# Plot pie chart for final results
pie_chart = alt.Chart(final_result_counts).mark_arc().encode(
    theta='Count:Q',
    color='Final Result:N',
    tooltip=[
        alt.Tooltip('Final Result:N', title='Final Result'),
        alt.Tooltip('Count:Q', title='Active Learners'),
        alt.Tooltip('Percentage:Q', title='Percentage', format='.1f')  # Format percentage with one decimal place
    ]
).properties(
    width=400,
    height=400
)

st.altair_chart(pie_chart, use_container_width=True)

# Expander for final result data
with st.expander("View Final Result Data"):
    st.write(final_result_counts)
    csv_final = final_result_counts.to_csv(index=False).encode('utf-8')  # Convert to CSV
    st.download_button(
        label="Download Final Result Data",
        data=csv_final,
        file_name="Final_Results_ELITE.csv",
        mime="text/csv",
        help='Click here to download the final result data as a CSV file'
)

st.subheader("Genuine")
# Filter DataFrame untuk bundle_name yang spesifik
genuine_filtered = filtered_df[filtered_df['title'] == 'Genuine']

# Get highest scores
highest_scores = genuine_filtered.loc[genuine_filtered.groupby(['email', 'last_updated', 'Test Name'])['total_score'].idxmax()]
genuine_active_learners_data = highest_scores[['name', 'email', 'Customer ID', 'title', 'last_updated', 'Test Name', 'total_score', 'final_result']]

# Add a rank column from 1 to 9 based on total_score for each Customer ID and Test Date
genuine_active_learners_data['rank'] = genuine_active_learners_data.groupby(['Customer ID', 'last_updated'])['total_score'].rank(ascending=False, method='first')

# Filter ranks from 1 to 9
genuine_active_learners_data = genuine_active_learners_data[genuine_active_learners_data['rank'] <= 9]

# Create a select box for rank selection with a unique key
genuine_rank = st.selectbox("Select Top for Genuine", options=list(range(1, 10)), key='genuine_rank_select')

# Filter data based on the selected rank
filtered_rank_data = genuine_active_learners_data[genuine_active_learners_data['rank'] == genuine_rank]

# Count the number of unique users for each test name at the selected rank
user_count_by_test = filtered_rank_data.groupby('Test Name')['Customer ID'].nunique().reset_index()
user_count_by_test.columns = ['Test Name', 'Total Active Users']
```

```
        # Display the results
        st.write(f"Genuine Top {genuine_rank}")
        st.dataframe(user_count_by_test)

        st.subheader("Astaka")
        # Filter DataFrame untuk bundle_name yang spesifik
        astaka_filtered = filtered_df[filtered_df['title'] == 'Astaka']

        # Get highest scores
        highest_scores = astaka_filtered.loc[astaka_filtered.groupby(['email', 'last_updated', 'Test Name'])['total_score'].idxmax()]
        astaka_active_learners_data = highest_scores[['name', 'email', 'Customer ID', 'title', 'last_updated', 'Test Name', 'total_score', 'final_result']]

        # Add a rank column from 1 to 6 based on total_score for each Customer ID and Test Date
        astaka_active_learners_data['rank'] = astaka_active_learners_data.groupby(['Customer ID', 'last_updated'])['total_score'].
        rank(ascending=False, method='first').astype(int)

        # Filter ranks from 1 to 6
        astaka_active_learners_data = astaka_active_learners_data[astaka_active_learners_data['rank'] <= 6]

        # Create a select box for rank selection with a unique key
        astaka_rank = st.selectbox("Select Top for Astaka", options=list(range(1, 7)), key='astaka_rank_select')

        # Filter data based on the selected rank
        filtered_rank_data = astaka_active_learners_data[astaka_active_learners_data['rank'] == astaka_rank]

        # Count the number of unique users for each test name at the selected rank
        user_count_by_test = filtered_rank_data.groupby('Test Name')['Customer ID'].nunique().reset_index()
        user_count_by_test.columns = ['Test Name', 'Total Active Users']

        # Display the results
        st.write(f"Astaka Top {astaka_rank}")
        st.dataframe(user_count_by_test)
if selected_platform == 'Capture':
    st.header('Active Learners - Capture', divider='gray')
    total_count = filtered_df['email'].nunique()
    internal_count = filtered_df[filtered_df['status_learner'] == 'Internal']['email'].nunique()
    external_count = filtered_df[filtered_df['status_learner'] == 'External']['email'].nunique()

    # Display metrics columns
    col1, col2, col3 = st.columns(3)
    with col1:
        st.markdown(f"<p style='font-size: 20px; text-align: center;'><strong>Overall: <span style='color: red;'>{total_count:,}</span></strong></p>",
                    unsafe_allow_html=True)
    with col2:
        st.markdown(f"<p style='font-size: 20px; text-align: center;'><strong>Internal User: <span style='color: red;'>{internal_count:,}</span></strong></p>",
                    unsafe_allow_html=True)
    with col3:
        st.markdown(f"<p style='font-size: 20px; text-align: center;'><strong>External User: <span style='color: red;'>{external_count:,}</span></strong></p>",
                    unsafe_allow_html=True)

    # Count unique emails per title
    title_counts = filtered_df.groupby('title')['email'].nunique().reset_index()
    title_counts.columns = ['title', 'active_learners']

    # Sort the data from highest to lowest
    title_counts = title_counts.sort_values(by='active_learners', ascending=False)

    # Create the bar chart
    chart = alt.Chart(title_counts).mark_bar().encode(
        x=alt.X('active_learners:Q', title='Active Learners'),
        y=alt.Y('title:O', title='Title', sort=alt.EncodingSortField(field='active_learners', order='descending')),
        color=alt.Color('title:O', scale=alt.Scale(scheme='category10')),
        tooltip=['title', 'active_learners']
    ).properties(
        title='Active Learners by Title'
    )

    # Add text layer to display active learners count on top of bars
    text = chart.mark_text(
        align='left',
        baseline='middle',
        color='black',
        fontSize=12,
        fontWeight='bold',
        dy=-5 # Adjusts the vertical position of the text
    ).encode(
        text='active_learners:Q'
    )

    # Combine the bar chart and text layer
    final_chart = chart + text

    # Display the chart
    st.altair_chart(final_chart, use_container_width=True)

    # Data download for Top Test Capture
    with st.expander("Top Test Capture"):
        st.write(title_counts)  # Now includes unit information
        csv = title_counts.to_csv(index=False).encode('utf-8')
        st.download_button(
            "Download Data",
            data=csv,
            file_name="Top_Test_Capture.csv",
            mime="text/csv",
            help='Click here to download the data as a CSV file'
        )
```

Gambar 3.14 Syntax Page Result Traits

Setelah syntax tersebut dijalankan maka halaman result traits pada dashboard streamlit terlihat seperti tampilan pada Gambar 3.15 Output Page Result Traits.

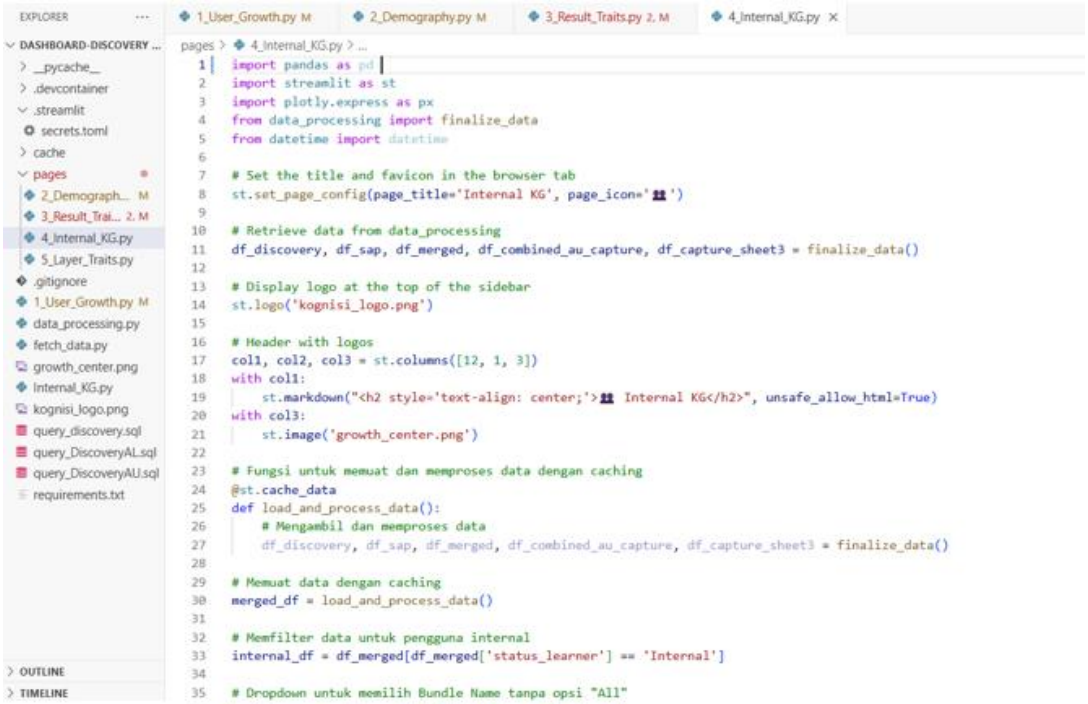



Gambar 3.15 Output Page Result Traits

### 3.2.5 Page Internal KG

Page internal KG merupakan halaman keempat pada dashboard streamlt discovery yang dimana pada halaman ini dilakukan pada minggu ke – 11 sampai dengan minggu ke – 14 dengan membuat perancangan terlebih dahulu setelah perancangan selesai dilakukan maka akan dilanjutkan dengan implementasi yang dimana pada halaman internal KG terdapat filter sesuai dengan yang diingin seperti bundle name dan test name. Pada dashboard streamlit discovery halaman internal KG juga menampilkan visualisasi seperti heatmap test taker per unit, pie chart overall result, bar chart result per unit, result per gender result per generation dan result per layer pada perusahaan Kompas Gramedia Group serta yang nantinya dapat difilter sesuai keinginan stakeholder. Pada

Gambar 3.16 Syntax Internal KG memperlihatkan codingan pembuatan halaman tersebut.



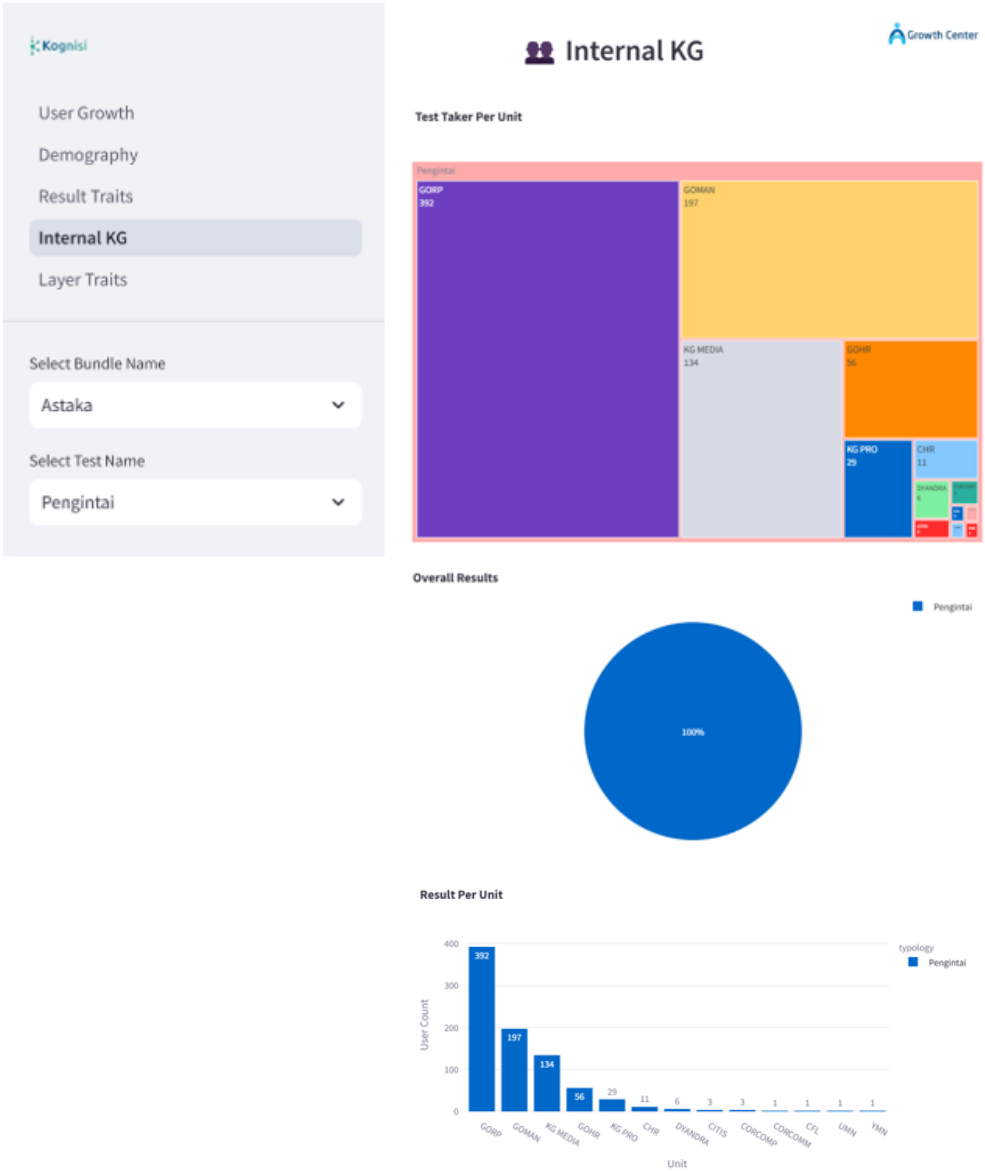
```python
import pandas as pd
import streamlit as st
import plotly.express as px
from data_processing import finalize_data
from datetime import datetime

# Set the title and favicon in the browser tab
st.set_page_config(page_title='Internal KG', page_icon='👥')

# Retrieve data from data_processing
df_discovery, df_sap, df_merged, df_combined_au_capture, df_capture_sheet3 = finalize_data()

# Display logo at the top of the sidebar
st.logo('kognisi_logo.png')

# Header with logos
col1, col2, col3 = st.columns([12, 1, 3])
with col1:
    st.markdown("<h2 style='text-align: center;'>👥 Internal KG</h2>", unsafe_allow_html=True)
with col3:
    st.image('growth_center.png')

# Fungsi untuk memuat dan memproses data dengan caching
@st.cache_data
def load_and_process_data():
    # Mengambil dan memproses data
    df_discovery, df_sap, df_merged, df_combined_au_capture, df_capture_sheet3 = finalize_data()

# Memuat data dengan caching
merged_df = load_and_process_data()

# Memfilter data untuk pengguna internal
internal_df = df_merged[df_merged['status_learner'] == 'Internal']

# Dropdown untuk memilih Bundle Name tanpa opsi "All"
bundle_names = internal_df['title'].dropna().unique()
selected_bundle = st.sidebar.selectbox("Select Bundle Name", bundle_names)

# Memfilter nama tes berdasarkan Bundle Name yang terkait dengan platform Discovery
test_name_options = internal_df[
    (internal_df['title'] == selected_bundle) &
    (internal_df['platform'] == 'Discovery')
]['Test Name'].dropna().unique()

if test_name_options.size == 0:
    st.sidebar.warning("No Test Names available for the selected Bundle Name in the Discovery platform.")
else:
    selected_test = st.sidebar.selectbox("Select Test Name", test_name_options)

# Menerapkan filter
internal_df = internal_df[internal_df['title'] == selected_bundle]
internal_df = internal_df[internal_df['Test Name'] == selected_test]

# Treemap untuk jumlah peserta tes per unit
treemap_data = internal_df.groupby(['Test Name', 'unit']).size().reset_index(name='Unit Count')
fig3 = px.treemap(
    treemap_data,
    title='Test Taker Per Unit',
    path=["Test Name", "unit"],
    values="Unit Count",
    hover_data=["Unit Count"],
    color="unit"
)
fig3.update_traces(textinfo='label+value', hovertemplate='<b>%{label}</b><br>User : %{value}')
fig3.update_layout(width=400, height=650)
st.plotly_chart(fig3, use_container_width=True)

# Diagram lingkaran untuk distribusi typology
typology_counts = internal_df['typology'].value_counts().reset_index()
typology_counts.columns = ['typology', 'User Count']
typology_counts = typology_counts.sort_values(by='User Count', ascending=False)
fig_typology = px.pie(
    typology_counts,
    names='typology',
    values='User Count',
    title='Overall Results'
```

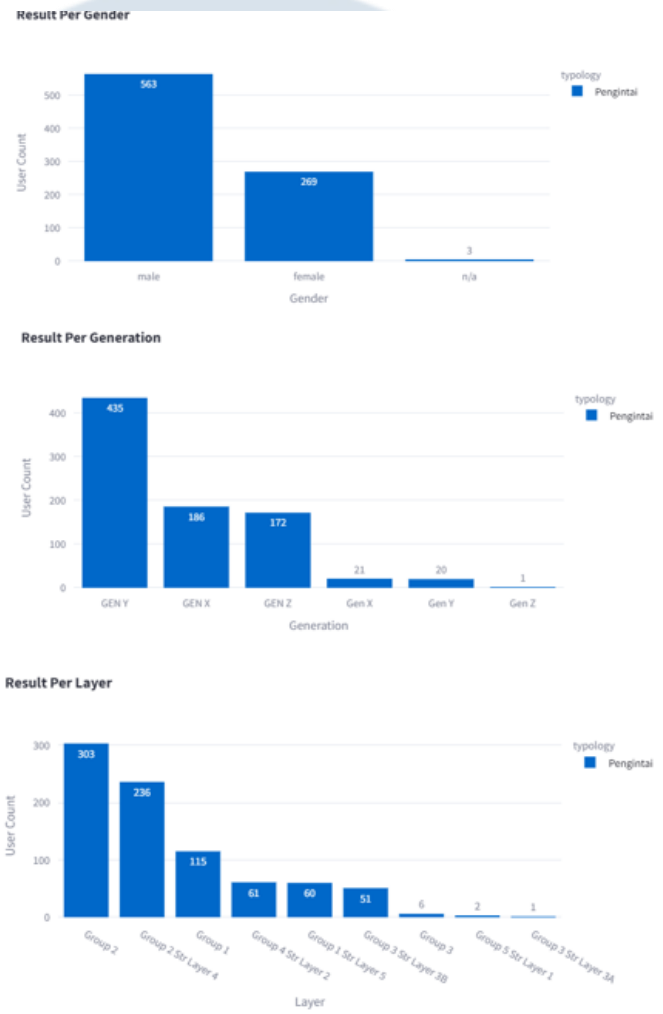
```
)
fig_typology.update_traces(
    hovertemplate='<b>%{label}</b><br>%{value}',
    textinfo='percent'
)
fig_typology.update_layout(
    legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="right", x=1)
)
st.plotly_chart(fig_typology, use_container_width=True)

# Diagram batang bertumpuk untuk hasil per unit
# Grouping by unit and typology, counting users
typology_counts = internal_df.groupby(['unit', 'typology']).size().reset_index(name='User Count')

# Sorting units by total user count
unit_total_counts = typology_counts.groupby('unit')['User Count'].sum().reset_index()
unit_total_counts = unit_total_counts.sort_values(by='User Count', ascending=False)
sorted_units = unit_total_counts['unit']

# Creating the stacked bar chart
fig_stacked = px.bar(
    typology_counts,
    x='unit',
    y='User Count',  # No percentage, using actual user count
    color='typology',
    title='Result Per Unit',
    labels={'User Count': 'User Count'},  # Update label to show actual counts
    text='User Count',
    height=400
)

# Updating layout to stack bars
fig_stacked.update_layout(
    barmode='stack',  # Stacked bars
    xaxis_title='Unit',
    yaxis_title='User Count',  # Updated y-axis label to reflect actual counts
    xaxis_categoryorder='array',
    xaxis_categoryarray=sorted_units,  # Sorting units by total count
    yaxis=dict(tickformat=',', title='User Count')  # Ensure y-axis shows raw counts
)

# Updating traces to show raw user count in the bars
fig_stacked.update_traces(texttemplate='%{text:.0f}')  # Display raw count in the bars
st.plotly_chart(fig_stacked, use_container_width=True)

# Diagram batang bertumpuk untuk gender berdasarkan count
gender_counts = internal_df.groupby(['gender', 'typology']).size().reset_index(name='User Count')
gender_total_counts = gender_counts.groupby('gender')['User Count'].sum().reset_index()
gender_total_counts = gender_total_counts.sort_values(by='User Count', ascending=False)
sorted_genders = gender_total_counts['gender']

fig_gender = px.bar(
    gender_counts,
    x='gender',
    y='User Count',
    color='typology',
    title='Result Per Gender',
    labels={'User Count': 'User Count'},
    text='User Count',
    height=400
)
fig_gender.update_layout(
    barmode='stack',
    xaxis_title='Gender',
    yaxis_title='User Count',
    xaxis_categoryorder='array',
    xaxis_categoryarray=sorted_genders,
    yaxis=dict(tickformat=',', title='User Count')
```

```
)
fig_gender.update_traces(texttemplate='%{text:.0f}')
st.plotly_chart(fig_gender, use_container_width=True)

# Diagram batang bertumpuk untuk generasi berdasarkan count
generation_counts = internal_df.groupby(['generation', 'typology']).size().reset_index(name='User Count')
generation_total_counts = generation_counts.groupby('generation')['User Count'].sum().reset_index()
generation_total_counts = generation_total_counts.sort_values(by='User Count', ascending=False)
sorted_generations = generation_total_counts['generation']

fig_generation = px.bar(
    generation_counts,
    x='generation',
    y='User Count',
    color='typology',
    title='Result Per Generation',
    labels={'User Count': 'User Count'},
    text='User Count',
    height=400
)
fig_generation.update_layout(
    barmode='stack',
    xaxis_title='Generation',
    yaxis_title='User Count',
    xaxis_categoryorder='array',
    xaxis_categoryarray=sorted_generations,
    yaxis=dict(tickformat=',', title='User Count')
)
fig_generation.update_traces(texttemplate='%{text:.0f}')
st.plotly_chart(fig_generation, use_container_width=True)

# Diagram batang bertumpuk untuk layer berdasarkan count
layer_counts = internal_df.groupby(['layer', 'typology']).size().reset_index(name='User Count')
layer_total_counts = layer_counts.groupby('layer')['User Count'].sum().reset_index()
layer_total_counts = layer_total_counts.sort_values(by='User Count', ascending=False)
sorted_layers = layer_total_counts['layer']

fig_layer = px.bar(
    layer_counts,
    x='layer',
    y='User Count',
    color='typology',
    title='Result Per Layer',
    labels={'User Count': 'User Count'},
    text='User Count',
    height=400
)
fig_layer.update_layout(
    barmode='stack',
    xaxis_title='Layer',
    yaxis_title='User Count',
    xaxis_categoryorder='array',
    xaxis_categoryarray=sorted_layers,
    yaxis=dict(tickformat=',', title='User Count')
)
fig_layer.update_traces(texttemplate='%{text:.0f}')
st.plotly_chart(fig_layer, use_container_width=True)
```

Gambar 3.16 Syntax Page Internal KG

Setelah syntax tersebut dijalankan maka halaman Internal KG pada dashboard streamlit terlihat seperti tampilan pada Gambar 3.17 Output Page Internal KG.



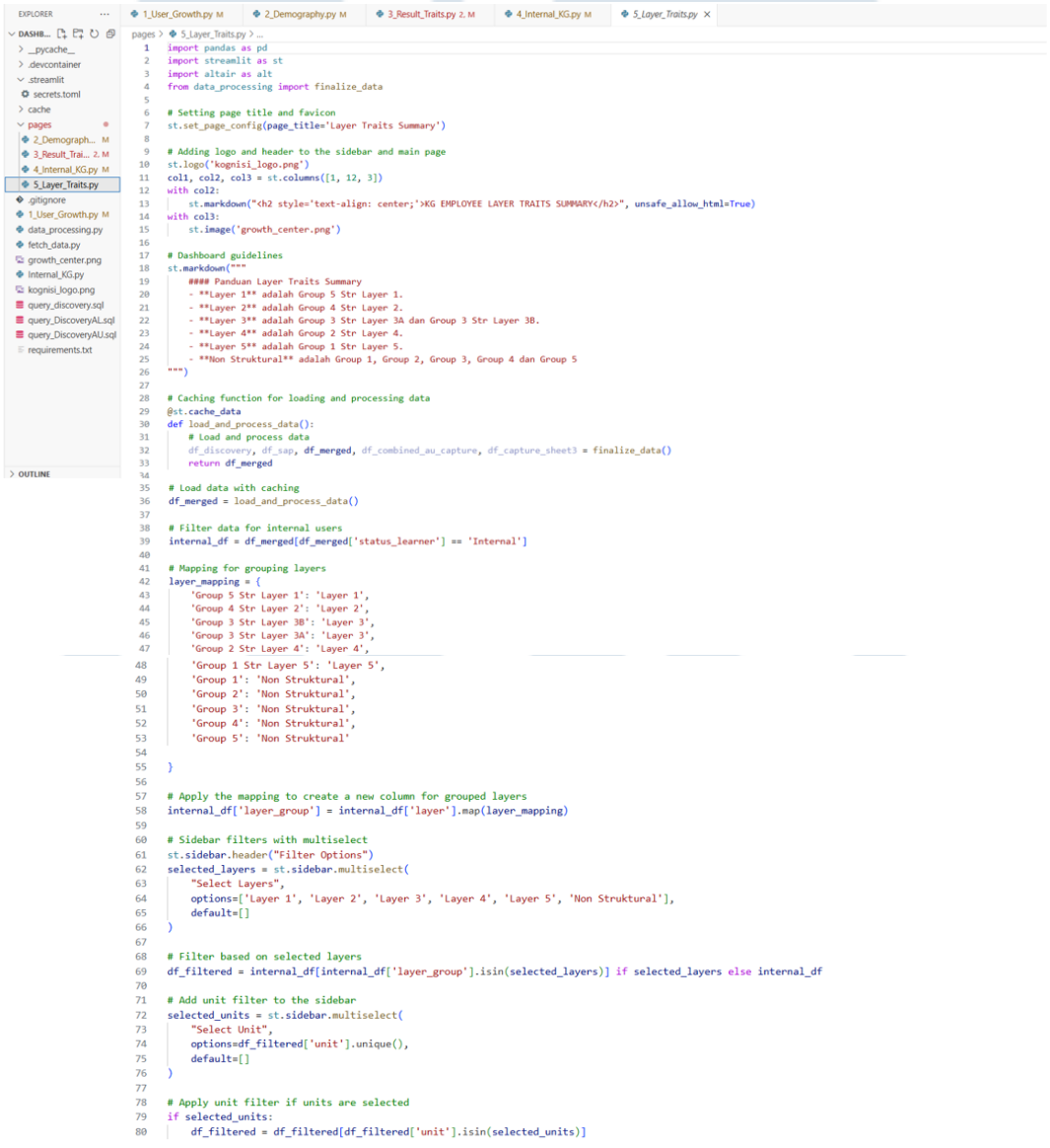



Gambar 3.17 Output Page Internal KG

### 3.2.6 Page layer Traits

Page layer traits merupakan halaman kelima pada dashboard streamlt discovery yang dimana pada halaman ini dilakukan pada minggu ke – 15 sampai dengan minggu ke – 19 dengan membuat perancangan terlebih dahulu setelah perancangan selesai dilakukan maka akan dilanjutkan dengan implementasi yang dimana pada halaman layer traits terdapat filter sesuai dengan yang diingin seperti layer, unit, dan bundle

name. Pada dashboard streamlit discovery halaman layer tarits juga menampilkan visualisasi seperti tabel summry setiap traits, bar chart Genuine dan bar chart Astaka pada perusahaan Kompas Gramedia Group serta yang nantinya dapat difilter sesuai keinginan stakeholder. Pada Gambar 3.18 Syntax Layer Traits memperlihatkan codingan pembuatan halaman tersebut.



```python
# Active learners by bundle
bundle_names = ['GI', 'LEAN', 'ELITE', 'Genuine', 'Astaka']

# Create filtered dataframe for active learners
if 'title' in df_filtered.columns:
    df_active_learners = df_filtered.groupby(['Customer ID', 'last_updated', 'title']).size().reset_index(name='test_count')

    # Count active learners per bundle
    bundle_counts = {bundle: df_active_learners[df_active_learners['title'] == bundle]['Customer ID'].nunique() for bundle in bundle_names}

    # Display active learners counts
    st.markdown("<h3>ACTIVE LEARNERS</h3>", unsafe_allow_html=True)
    col1, col2, col3, col4, col5 = st.columns(5)

    for i, bundle in enumerate(bundle_names):
        with eval(f'col{i + 1}'):
            st.markdown(f"<p style='font-size: 20px; text-align: center;'><strong>{bundle}: <span style='color: red;'>
                        {bundle_counts[bundle]:,}</span></strong></p>", unsafe_allow_html=True)


# 1. Get the latest test results for each email and Test Name
latest_test_results = df_filtered.loc[df_filtered.groupby(['email', 'Test Name'])['last_updated'].idxmax()]

# 2. Count participants based on bundle_name
participant_counts = df_filtered.groupby('title')['email'].nunique().reset_index()
participant_counts.columns = ['title', 'jumlah_partisipan']

# 3. Count users based on typology from the latest results
typology_user_counts = latest_test_results.groupby('typology')['email'].nunique().reset_index()
typology_user_counts.columns = ['typology', 'jumlah']

# 4. Get unique Test Name for each bundle_name
test_names = df_filtered[['title', 'Test Name']].drop_duplicates()

# 5. Get all unique typology results for each Test Name
typology_results = latest_test_results.groupby(['Test Name', 'typology'])['email'].nunique().reset_index()
typology_results.columns = ['Test Name', 'typology', 'jumlah']

# 6. Calculate percentages for each typology per Test Name
total_users_per_test = typology_results.groupby('Test Name')['jumlah'].transform('sum')
typology_results['persentase'] = (typology_results['jumlah'] / total_users_per_test * 100).round(2)

# 7. Combine results into one DataFrame
result_df = pd.merge(participant_counts, test_names, on='title', how='left')
result_df = pd.merge(result_df, typology_results, on='Test Name', how='left')
 # 8. Add rows for Overall ELITE based on filtered results
 final_results_elite = latest_test_results[latest_test_results['title'] == 'ELITE'].groupby('final_result')['email'].nunique().reset_index()
 overall_elite_rows = [{
     'title': 'ELITE',
     'jumlah_partisipan': participant_counts.loc[participant_counts['title'] == 'ELITE', 'jumlah_partisipan'].values[0],
     'Test Name': 'Overall ELITE',
     'typology': row['final_result'],
     'jumlah': row['email'],
     'persentase': (row['email'] / participant_counts.loc[participant_counts['title'] == 'ELITE', 'jumlah_partisipan'].values[0] * 100).round(2)
 } for _, row in final_results_elite.iterrows()]

 overall_elite_df = pd.DataFrame(overall_elite_rows)

 # 9. Add rows for Overall LEAN based on filtered results
 final_results_lean = latest_test_results[latest_test_results['title'] == 'LEAN'].groupby('final_result')['email'].nunique().reset_index()
 overall_lean_rows = [{
     'title': 'LEAN',
     'jumlah_partisipan': participant_counts.loc[participant_counts['title'] == 'LEAN', 'jumlah_partisipan'].values[0],
     'Test Name': 'Overall LEAN',
     'typology': row['final_result'],
     'jumlah': row['email'],
     'persentase': (row['email'] / participant_counts.loc[participant_counts['title'] == 'LEAN', 'jumlah_partisipan'].values[0] * 100).round(2)
 } for _, row in final_results_lean.iterrows()]

 overall_lean_df = pd.DataFrame(overall_lean_rows)

 # 10. Combine result_df with overall_elite_df and overall_lean_df
 result_df = pd.concat([result_df, overall_elite_df, overall_lean_df], ignore_index=True)

 # 11. Filter for desired bundles: GI, ELITE, LEAN
 filtered_bundles = ['GI', 'ELITE', 'LEAN']
 result_df = result_df[result_df['title'].isin(filtered_bundles)]
```

```python
# 12. Set categorical order and sort
result_df['title'] = pd.Categorical(result_df['title'], categories=filtered_bundles, ordered=True)
result_df = result_df.sort_values(['title', 'Test Name']).reset_index(drop=True)

# 13. Combine number of users with result_df without additional calculations
combined_result_df = pd.merge(result_df, typology_user_counts[['typology']], on='typology', how='left')

# 14. Create a custom sort order for 'Test Name'
def custom_sort_order(test_name):
    if test_name in ["Mindset", "Overall ELITE", "Overall LEAN"]:
        return 0
    return 1

combined_result_df['sort_order'] = combined_result_df['Test Name'].apply(custom_sort_order)

# Sort by bundle_name and custom sort order
combined_result_df = combined_result_df.sort_values(['title', 'sort_order', 'Test Name']).reset_index(drop=True)

# 15. Bundle filter
st.sidebar.header("Bundle Name Filter")
selected_bundle = st.sidebar.selectbox("Select Bundle Name", options=filtered_bundles, index=0)

# Filter combined_result_df based on selected bundle
if selected_bundle == 'LEAN':
    # Hapus baris dengan typology 'The Olympian' dan 'The Spectator' untuk bundle LEAN
    combined_result_df = combined_result_df[
        (combined_result_df['title'] == selected_bundle) &
        (~combined_result_df['typology'].isin(['The Olympian', 'The Spectator']))
    ]
elif selected_bundle == 'ELITE':
    # Hapus baris dengan typology 'Citizen' dan 'Governor' untuk bundle ELITE
    combined_result_df = combined_result_df[
    (combined_result_df['title'] == selected_bundle) &
    (~combined_result_df['typology'].isin(['Citizen', 'Governor']))
     ]
else:
     # Filter untuk bundle lain
     combined_result_df = combined_result_df[combined_result_df['title'] == selected_bundle]

# 16. Display results in Streamlit, excluding 'sort_order'
selected_layers_title = ', '.join(selected_layers) if selected_layers else 'All Layers'
selected_units_title = ', '.join(selected_units) if selected_units else 'All Units'
st.subheader(f"{selected_bundle} Traits Summary for {selected_layers_title} ({selected_units_title})")
st.dataframe(combined_result_df.drop(columns=['sort_order']))
# Menghitung total pengguna untuk setiap Test Name
total_users_per_test = combined_result_df.groupby('Test Name')['jumlah'].transform('sum')

# Menghitung persentase untuk setiap typology pada masing-masing Test Name
combined_result_df['persentase'] = (combined_result_df['jumlah'] / total_users_per_test * 100).round(2)

# Membuat stacked bar chart dengan warna otomatis untuk setiap nilai typology
stacked_bar_chart = alt.Chart(combined_result_df).mark_bar().encode(
    x=alt.X('persentase:Q', title='Persentase Active Learners'),
    y=alt.Y('Test Name:N', title='Test Name', sort='-x'),  # Menggunakan Test Name untuk sumbu Y
    color=alt.Color(
        'typology:N',
        title='Typology',
        scale=alt.Scale(scheme='category10')  # Palet warna otomatis untuk berbagai nilai typology
    ),
    tooltip=[
        alt.Tooltip('Test Name:N', title='Test Name'),
        alt.Tooltip('jumlah:Q', title='Jumlah'),
        alt.Tooltip('typology:N', title='Typology'),
        alt.Tooltip('persentase:Q', title='Persentase (%)')
    ]
).properties(
    title=f'Distribusi {selected_layers_title} Berdasarkan traits {selected_bundle} di {selected_units_title}',
    width=600,
    height=400
).configure_axis(
    labelFontSize=12,
    titleFontSize=14
)

# Menampilkan stacked bar chart di Streamlit
st.altair_chart(stacked_bar_chart, use_container_width=True)

# Filter for 'Genuine' bundle
```

```python
genuine_filtered = df_filtered[df_filtered['title'] == 'Genuine']

# Get the highest scores
highest_scores = genuine_filtered.loc[genuine_filtered.groupby(['email', 'last_updated', 'Test Name'])['total_score'].idxmax()]

# Select relevant columns for the genuine active learners data
genuine_active_learners_data = highest_scores[['name', 'email', 'Customer ID', 'title', 'last_updated', 'Test Name', 'total_score', 'final_result']]

# Add a rank column from 1 to 9 based on total_score for each Customer ID and Test Date
genuine_active_learners_data['rank'] = genuine_active_learners_data.groupby(['Customer ID', 'last_updated'])['total_score'].
rank(ascending=False, method='first').astype(int)

# Filter ranks from 1 to 9
genuine_active_learners_data = genuine_active_learners_data[genuine_active_learners_data['rank'] <= 9]

# Add a select box for filtering by rank
selected_rank = st.selectbox("Select Rank", options=range(1, 10), index=0)

# Filter the data based on the selected rank
filtered_data_by_rank = genuine_active_learners_data[genuine_active_learners_data['rank'] == selected_rank]

# Get the latest Test Date results for each email
latest_results = filtered_data_by_rank.loc[filtered_data_by_rank.groupby('email')['last_updated'].idxmax()]

# Calculate jumlah_partisipan (total unique email for the bundle)
total_participants = latest_results['email'].nunique()

st.subheader(f"Genuine TOP {selected_rank} Traits Summary for {selected_layers_title} ({selected_units_title})")

# Prepare the data for display
summary_data = (
    latest_results
    .groupby(['title', 'Test Name'])  # Group by bundle_name and Test Name
    .agg(
        jumlah_partisipan=('email', 'nunique'),  # Count of unique email based only on Test Name
        jumlah=('email', 'count')  # Count of total entries based on email per Test Name
    )
    .reset_index()
)

# Add the total_participants to the summary_data
summary_data['jumlah_partisipan'] = total_participants

# Calculate the percentage
summary_data['persentase'] = (summary_data['jumlah'] / summary_data['jumlah_partisipan'] * 100).round(2)

# Reorder the columns as specified
summary_data = summary_data[['title', 'jumlah_partisipan', 'Test Name', 'jumlah', 'persentase']]
st.dataframe(summary_data)

# Visualization: Simple bar chart for jumlah
bar_chart = alt.Chart(summary_data).mark_bar().encode(
    x=alt.X('Test Name:O', title='Test Name'),
    y=alt.Y('jumlah:Q', title='Jumlah'),
    tooltip=['Test Name', 'jumlah', 'persentase']
).properties(
    title=f'Distribusi Genuine TOP {selected_rank}'
)

# Display the bar chart
st.altair_chart(bar_chart, use_container_width=True)

# Filter for 'Astaka' bundle
astaka_filtered = df_filtered[df_filtered['title'] == 'Astaka']

# Get the highest scores
highest_scores = astaka_filtered.loc[astaka_filtered.groupby(['email', 'last_updated', 'Test Name'])['total_score'].idxmax()]

# Select relevant columns for the genuine active learners data
astaka_active_learners_data = highest_scores[['name', 'email', 'Customer ID', 'title', 'last_updated', 'Test Name', 'total_score', 'final_result']]

# Add a rank column from 1 to 6 based on total_score for each Customer ID and Test Date
astaka_active_learners_data['rank'] = astaka_active_learners_data.groupby(['Customer ID', 'last_updated'])['total_score'].
rank(ascending=False, method='first').astype(int)

# Filter ranks from 1 to 6
astaka_active_learners_data = astaka_active_learners_data[astaka_active_learners_data['rank'] <= 6]

# Add a select box for filtering by rank
selected_rank = st.selectbox("Select Rank", options=range(1, 7), index=0)

# Filter the data based on the selected rank
filtered_data_by_rank = astaka_active_learners_data[astaka_active_learners_data['rank'] == selected_rank]

# Get the latest Test Date results for each email
latest_results = filtered_data_by_rank.loc[filtered_data_by_rank.groupby('email')['last_updated'].idxmax()]
```

```python
latest_results = filtered_data_by_rank.loc[filtered_data_by_rank.groupby('email')['last_updated'].idxmax()]

# Calculate jumlah_partisipan (total unique email for the bundle)
total_participants = latest_results['email'].nunique()

# Prepare the data for display
summary_data = (
    latest_results
    .groupby(['title', 'Test Name'])  # Group by bundle_name and Test Name
    .agg(
        jumlah_partisipan=('email', 'nunique'),  # Count of unique email based only on Test Name
        jumlah=('email', 'count')  # Count of total entries based on email per Test Name
    )
    .reset_index()
)

# Add the total_participants to the summary_data
summary_data['jumlah_partisipan'] = total_participants

# Calculate the percentage
summary_data['persentase'] = (summary_data['jumlah'] / summary_data['jumlah_partisipan'] * 100).round(2)

# Reorder the columns as specified
summary_data = summary_data[['title', 'jumlah_partisipan', 'Test Name', 'jumlah', 'persentase']]

# Display the filtered data in Streamlit
st.subheader(f"Astaka TOP {selected_rank} Traits Summary for {selected_layers_title} ({selected_units_title})")
st.dataframe(summary_data)

# Visualization: Simple bar chart for jumlah
bar_chart = alt.Chart(summary_data).mark_bar().encode(
    x=alt.X('Test Name:O', title='Test Name'),
    y=alt.Y('jumlah:Q', title='Jumlah'),
    tooltip=['Test Name', 'jumlah', 'persentase']
).properties(
)

# Display the bar chart
st.altair_chart(bar_chart, use_container_width=True)

```

Gambar 3.18 Syntax Page Layer Traits

Setelah syntax tersebut dijalankan maka halaman layer traits pada dashboard streamlit terlihat seperti tampilan pada Gambar 3.19 Output Page Layer Traits.

**Bundle Name Filter**

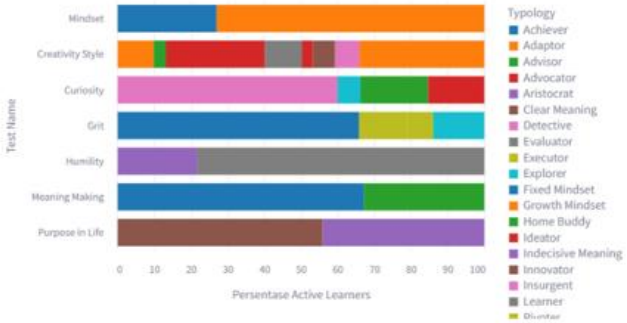
Select Bundle Name

GI

## GI Traits Summary for All Layers (All Units)

| | title | jumlah_partisipan | Test Name | typology | jumlah | persentase |
|---|---|---|---|---|---|---|
| 14 | GI | 954 | Grit | Achiever | 628 | 65.83 |
| 15 | GI | 954 | Grit | Executor | 124 | 13 |
| 16 | GI | 954 | Grit | Pivoter | 69 | 7.23 |
| 17 | GI | 954 | Grit | Planner | 133 | 13.94 |
| 18 | GI | 954 | Humility | Aristocrat | 207 | 21.7 |
| 19 | GI | 954 | Humility | Learner | 747 | 78.3 |
| 20 | GI | 954 | Meaning Making | Purpose Driven | 641 | 67.19 |
| 21 | GI | 954 | Meaning Making | Task Driven | 313 | 32.81 |
| 22 | GI | 954 | Purpose in Life | Clear Meaning | 533 | 55.87 |
| 23 | GI | 954 | Purpose in Life | Indecisive Meaning | 421 | 44.13 |

**Distribusi All Layers Berdasarkan traits GI di All Units**




Select Rank

1

## Genuine TOP 1 Traits Summary for All Layers (All Units)

| | title | jumlah_partisipan | Test Name | jumlah | persentase |
|---|---|---|---|---|---|
| 0 | Genuine | 726 | Auditory | 58 | 7.99 |
| 1 | Genuine | 726 | Interpersonal | 263 | 36.23 |
| 2 | Genuine | 726 | Intrapersonal | 111 | 15.29 |
| 3 | Genuine | 726 | Kinaesthetic | 29 | 3.99 |
| 4 | Genuine | 726 | Linguistic | 65 | 8.95 |
| 5 | Genuine | 726 | Mathematical & Logical | 88 | 12.12 |
| 6 | Genuine | 726 | Naturalistic | 29 | 3.99 |
| 7 | Genuine | 726 | Philosophical and Ethical | 20 | 2.75 |
| 8 | Genuine | 726 | Visual | 63 | 8.68 |


Gambar 3.19 Output Page Layer Traits

## 3.3 Kendala yang Ditemukan

Selama proses melaksanakan program pratik kerja magang, peserta magang tentunya menghadapi beberapa hambatan ataupun kesulitan baik dalam menjalankan tugas haruan hingga project dashboard streamlit discovery yang dikerjakan. Berikut ini beberapa kendala atau hambatan yang dihadapi peserta magang sebagai seorang internship People Data Analyst di perusahaan Kompas Gramedia Group yaitu sebagai berikut:

a. Waktu diskusi menjadi salah satu hambatan selama proses pratik kerja magang hal ini dikarena untuk dapat berdiskusi dengan mentor diperlukan penjadwalan melalui google kalender karena sistem kerja

pada perusahaan Kompas Gramedia Group yang hybrid dengan jadwal masuk kantor hanya 2 hari saja dalam 1 minggu sehingga peserta magang kesulitan untuk berdiskusi lebih detail dengan mentor baik itu untuk mendapatkan arahan maupun feedback.

b. Istilah – istilah pada data yang akan diolah sulit untuk dimengerti seperti group description ternyata adalah unit dari Kompas Gramedia Group masih banyak lagi istilah – istilah lainnya sehingga nantinya akan terjadi kesalahpahaman dalam penjelasan istilah pada data yang akan digunakan.
c. Internet perusahaan Kompas Gramedia Group yang sedikit kurang stabil sehingga kesulitan bagi seorang peserta magang untuk dapat menyelesaikan pekerjaan yang diberikan pada saat itu.

### 3.4 Solusi atas kendala yang ditemukan

Berdasarkan dari hambatan dan kesulitan yang dihadapi selama program pratik kerja magang pada perusahaan Kompas Gramedia Group terdapat solusi yang dimana mampu mengatasi seluruh hambatan yang dihadapi peserta magang sebagai seorang Internship People Data Analyst yaitu sebagai berikut :

a. Solusi terkait dengan waktu untuk dapat berdiskusi dengan mentor biasanya peserta magang akan melakukan chat terlebih dahulu melalui platform whatsapp untuk dapat menanyakan waktu kosong yang dimiliki mentor agar nantinya dapat di set pada google calendar.
b. Solusi terkait dengan istilah pada data peserta magang tentunya akan mengexplore terlebih dahulu seluruh informasi yang berkaitan dengan istilah tersebut dan kemudian disesuaikan kembali jika sudah biasanya akan dilakukan cross check kembali ke mentor dari setiap istilah tersebut agar tidak terjadi miss komunikasi.

c. Solusi terkait dengan internet perusahaan yang kurang stabil peserta magang akan menggunakan internet pribadi agar dapat menyelesaikan ataupun melanjutkan pekerjaan – pekerjaan.