BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Dalam perusahaan Kompas Gramedia, khususnya di Group of Retail and Publishing pada Divisi SIT, mahasiswa diberikan kesempatan untuk bergabung dalam Divisi Data Analyst (DA) guna mendukung berbagai proyek yang berkaitan dengan database, machine learning, dan deep learning. Dalam divisi ini, terdapat dua peran utama, yaitu Data Scientist dan Data Engineer. Data Engineer bertanggung jawab atas pengelolaan infrastruktur database perusahaan, termasuk manipulasi, pembuatan, pembaruan, serta penghapusan tabel, database, dan data. Selain itu, mereka memastikan integritas database tetap terjaga serta mengelola hak akses bagi rekan kerja yang membutuhkan akses terhadap keseluruhan sistem database perusahaan.

Mahasiswa yang mengikuti program magang ini ditempatkan sebagai Data Scientist, yang memiliki tugas utama dalam pengambilan data dari database menggunakan SQL serta melakukan analisis terhadap berbagai proyek, salah satunya adalah pengembangan model deteksi kecurangan (fraud detection). Dalam proses ini, mereka bertanggung jawab untuk memilih kolom data yang paling relevan guna memastikan akurasi analisis yang dilakukan. Selain itu, mahasiswa juga harus mengevaluasi berbagai model yang sesuai dengan kebutuhan proyek agar dapat menghasilkan solusi yang optimal.

Struktur koordinasi dalam program magang ini melibatkan beberapa tingkatan. Pada tingkatan pertama, mahasiswa bekerja di bawah bimbingan seorang mentor yang bertugas memberikan arahan pekerjaan, membantu dalam menyelesaikan kendala yang dihadapi, serta memastikan bahwa hasil kerja telah memenuhi standar yang diharapkan. Mentor juga bertanggung jawab untuk mengevaluasi hasil pekerjaan, memberikan masukan yang diperlukan, serta menjamin kualitas analisis sebelum diteruskan ke tahap berikutnya. Di atas mentor, terdapat supervisor yang berperan sebagai manajer dalam struktur koordinasi ini. Setiap hasil pekerjaan yang telah diselesaikan oleh mahasiswa harus dipresentasikan kepada supervisor guna menilai perkembangan proyek serta menentukan langkah selanjutnya. Supervisor juga memiliki wewenang untuk memberikan persetujuan akhir (ACC) terhadap suatu proyek setelah melalui tahap

evaluasi menyeluruh guna memastikan bahwa proyek telah selesai atau jika masih diperlukan perbaikan lebih lanjut sebelum digunakan dalam implementasi yang lebih luas.

3.2 Tugas yang Dilakukan

Selama di Kompas Gramedia, salah satu tugas utama yang dilakukan adalah membuat klasifikasi *gender* menggunakan model IndoBERT. Pemilihan IndoBERT didasarkan pada hasil diskusi dengan mentor dan supervisor. Pergantian model ini juga diperkuat oleh hasil akurasi model sebelumnya, yaitu random forest, yang hanya mencapai 70%-72%. Meskipun standar koperasi Kompas Gramedia menyatakan bahwa akurasi di atas 70% sudah memadai, supervisor ingin melihat apakah penggunaan model IndoBERT dapat memberikan hasil yang lebih baik dibandingkan model sebelumnya.

Secara singkat, BERT pada dasarnya merupakan versi yang ditingkatkan dari Transformer dengan beberapa modifikasi penting, terutama sifatnya yang bidirectional, yang memungkinkan model untuk mempertimbangkan konteks dari token di masa lalu, sekarang, dan masa depan secara simultan saat membuat prediksi. Kemampuan ini meningkatkan pemahaman makna kontekstual dari dua arah sekaligus, menjadikannya unggul dalam memahami bahasa alami secara lebih mendalam [21]. Selain itu, BERT hanya menggunakan bagian encoder dari arsitektur Transformer, berbeda dengan Transformer asli yang menggunakan encoder-decoder untuk tugas many-to-many seperti penerjemahan. Karena hanya fokus pada pemahaman input, BERT lebih cocok untuk tugas many-to-one atau one-to-one, seperti klasifikasi atau ekstraksi informasi, bukan untuk menghasilkan kalimat. Struktur ini menjadikannya sangat efektif dalam berbagai aplikasi seperti analisis sentimen, deteksi spam, dan sistem tanya-jawab.

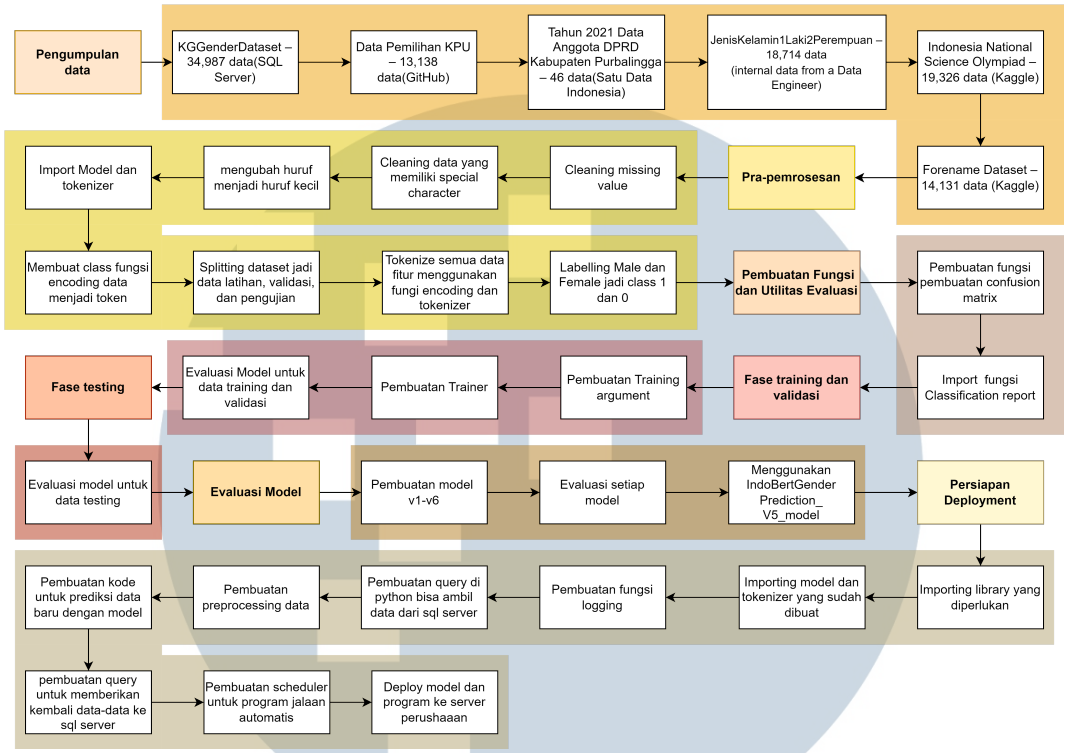
IndoBERT merupakan turunan dari arsitektur BERT yang secara khusus dikembangkan untuk memahami Bahasa Indonesia, dengan didasarkan pada proses pre-training yang menggunakan korpus teks berbahasa Indonesia. Tahap pre-training ini memungkinkan model untuk mempelajari representasi kontekstual dari token-token dalam Bahasa Indonesia, sehingga model memperoleh pemahaman yang mendalam terhadap struktur dan makna bahasa tersebut. Dengan demikian, saat IndoBERT digunakan dalam berbagai tugas pemrosesan bahasa alami, tidak diperlukan lagi proses embedding tambahan, yaitu proses yang memberikan representasi vektor terhadap token, karena model telah secara implisit memahami

relasi antar token melalui pelatihan sebelumnya [22]. Oleh karena itu, IndoBERT dapat langsung digunakan untuk berbagai aplikasi tanpa memerlukan tahap awal untuk menyesuaikan diri terhadap Bahasa Indonesia.

Terdapat dua versi IndoBERT yang saat ini tersedia, yakni IndoBERT yang dikembangkan oleh IndoLEM dan IndoBERT versi IndoNLU. Dalam proyek ini, digunakan IndoBERT dari IndoNLU, bukan hanya karena model ini telah digunakan secara luas oleh komunitas, tetapi juga karena performanya yang secara konsisten lebih unggul dibandingkan versi IndoLEM. Berdasarkan evaluasi kinerja, IndoBERT IndoNLU mencapai rata-rata *F1 score* sebesar 85,71 pada berbagai tugas, sementara IndoBERT IndoLEM hanya memperoleh skor 84,13, dan itu pun hanya dari satu jenis tugas [23][24]. Oleh karena itu, pemilihan IndoBERT IndoNLU dinilai lebih tepat untuk mendukung kebutuhan dan akurasi dalam proyek ini.

3.3 Uraian Pelaksanaan Magang

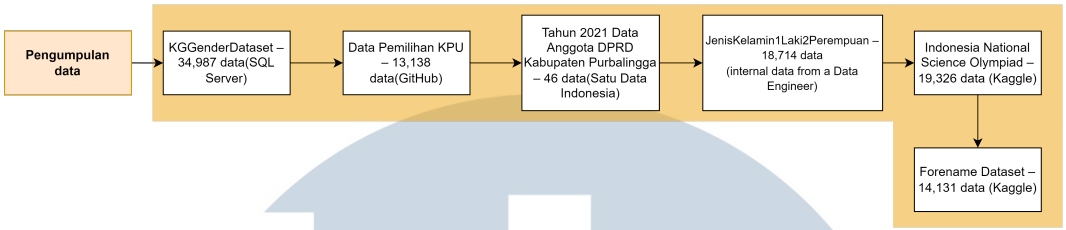
Proyek ini dimulai pada 3 febuari 2025 dan memiliki update paling terakhir pada bulan April 2025 setelah dipresentasikan kepada supervisor dan mentor dengan hasil dari Proyek tersebut. Dalam rangkaian waktu tersebut, ada beberapa langkah utama yang dilakukan untuk melancarkan pembuatan proyek ini: pengumpulan data, pra pemprosesan, pelatihan dan pengujian model, serta akhirnya, penerapan ke server produksi. Setiap langkah-langkah tersebut bisa terlihat dalam alur bagan pekerjaan yang bisa terlihat pada gambar 3.1.



Gambar 3.1. Bagan proses pembuatan model IndoBERT prediksi kelamin dari nama Indonesia

3.3.1 pengumpulan data

Setiap *task* memerlukan data dan fitur khusus agar model dapat bekerja secara optimal. Dalam makalah BERT, IndoLEM, dan IndoNLU, terlihat bahwa pengujian kinerja BERT dilakukan dengan menggunakan *dataset* khusus yang disesuaikan dengan masing-masing *task* yang ditargetkan [21] [24] [23]. Salah satu studi relevan lainnya adalah karya To, H. Q., Nguyen, K. V., Nguyen, N. L.-T., dan Nguyen, A. G.-T., yang memprediksi *gender* berdasarkan nama dalam bahasa Vietnam. Walaupun fokusnya bukan pada nama-nama Indonesia, pendekatan mereka dapat dijadikan referensi karena kesamaan struktur tugas menggunakan nama sebagai satu-satunya fitur untuk memprediksi *gender* [25]. Bagan pekerjaan untuk pengumpulan data bisa dilihat pada gambar 3.2.



Gambar 3.2. Bagan Kerja untuk pengumpulan data

	CustomerName	Gender
1	GERALL OWEN	M
2	Mardatomi Maselta	M
3	Stefanus Suyono	M
4	Mohamad Iwan Cahyadi	M
5	HKF Official	F
6	Rizky Bramantyo	M
7	Enya grace	F
8	nhia pram	F
9	Sintya Handayani	F
10	Aril Al Kautsar	M
11	ragel pranata	F
12	Rando lubis	M
13	Dwi Kurnia Asih	F
14	Meisyani fadila	F
15	Omega Purnomo	M

Gambar 3.3. Sebagian dataset untuk prediksi *gender* berdasarkan nama

Berdasarkan referensi tersebut, proyek ini mengadopsi struktur dataset yang sederhana dan terfokus, seperti yang diperlihatkan pada Gambar 3.3. Dataset hanya terdiri dari satu variabel independen, yaitu nama individu, dan satu variabel dependen, yaitu *gender*. Dengan hanya menggunakan dua kolom utama ini, model dapat dibangun secara efisien tanpa perlu menangani fitur tambahan yang dapat meningkatkan kompleksitas. Pendekatan ini bertujuan untuk mengoptimalkan klasifikasi dengan memanfaatkan struktur data yang minimalis namun informatif.

Sangat penting bahwa nama-nama yang digunakan dalam proyek ini merupakan nama-nama Indonesia, bukan nama-nama dalam bahasa Inggris atau dari negara lain. Nama-nama ini bisa saja sangat umum maupun jarang digunakan. Meskipun beberapa di antaranya berasal dari budaya asing, nama-nama tersebut sudah dikenal luas dan umum digunakan di Indonesia. Data yang digunakan dikumpulkan dari berbagai sumber seperti acara di Indonesia, basis data perusahaan, dan *dataset* yang secara spesifik dilabeli berisi nama-nama Indonesia.

Pada awalnya, data disimpan dalam basis data *SQL Server*. Kode berikut

digunakan untuk mengambil data yang dibutuhkan. Demi menjaga kerahasiaan, nama tabel dalam kueri telah diubah, meskipun konteks dan maknanya tetap sama:

```
SELECT CustomerName, MemberId
FROM Customer
WHERE
  MemberId IS NOT NULL
  AND LTRIM(RTRIM(MemberId)) <> '' -- Exclude empty strings
  AND CustomerName IS NOT NULL
  AND LTRIM(RTRIM(CustomerName)) <> '' -- Exclude empty
   strings
  AND Gender IS NOT NULL
  AND LTRIM(RTRIM(Gender)) <> ''
```

Kode 3.1: Potongan kode SQL untuk pengambilan data

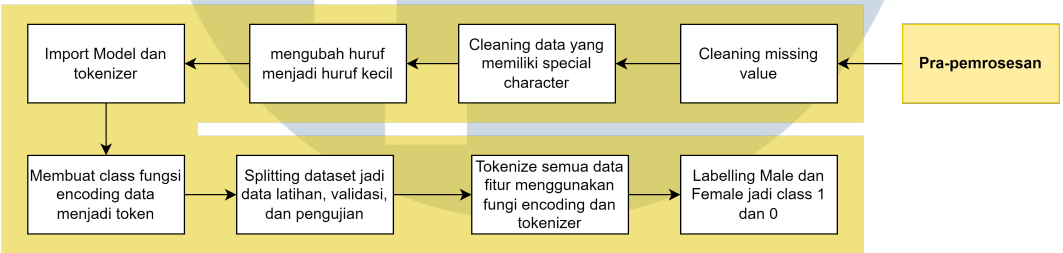
Kode 3.1 dapat disederhanakan untuk mengambil nama pelanggan yang merupakan anggota dan memiliki informasi *gender* yang tercatat dalam tabel *Customer*. Pada klausa *WHERE*, kondisi tersebut memastikan bahwa *Member ID* tidak bernilai *NULL* maupun string kosong. Pengecekan ini penting karena beberapa entri menggunakan string kosong, bukan *NULL*, untuk menunjukkan data yang hilang, dan keduanya perlu disaring agar integritas data tetap terjaga.

Namun, *dataset* ini saja tidak cukup untuk pelatihan model, sebab hanya menghasilkan total 34.987 data saja. Untuk memperkaya *dataset* dan memastikan informasi yang lebih beragam serta representatif, sumber data eksternal juga digunakan. Secara keseluruhan, sebanyak 100.342 nama beserta label *gender* dikumpulkan dari sumber-sumber berikut untuk memberikan fondasi yang lebih komprehensif bagi model. Distribusi dataset antara laki-laki :

1. Indonesia National Science Olympiad – 19.326 data (*Kaggle*)
2. Data Pemilihan KPU – 13.138 data (*GitHub*)
3. JenisKelamin1Laki2Perempuan – 18.714 data (data internal dari seorang *Data Engineer*)
4. *KGGenderDataset* – 34.987 data (*SQL Server*)
5. Data Anggota DPRD Kabupaten Purbalingga Tahun 2021 – 46 data (*Satu Data Indonesia*)
6. *Forename Dataset* – 14.131 data (*Kaggle*)

3.3.2 Pra-pemrosesan

Tahap pra-pemrosesan dapat dijelaskan sebagai proses untuk menghilangkan data yang bermasalah serta meningkatkan kualitas data, sehingga hasil prediksi dari suatu model menjadi lebih optimal. Beberapa langkah yang umum dilakukan dalam pra-pemrosesan mencakup evaluasi, penyaringan, manipulasi, dan pengkodean data agar dapat dipahami dan diproses dengan baik oleh model. Permasalahan umum seperti *outliers*, *missing value*, dan ketidakkonsistenan data harus diatasi untuk menjaga kualitas dataset [26]. Oleh karena itu, dalam proyek prediksi gender ini, akan diterapkan berbagai strategi *preprocessing* untuk memastikan proses prediksi berjalan lancar dan menghasilkan output yang optimal. Bagan kerja untuk pra-pemrosesan bisa dilihat pada gambar 3.4.



Gambar 3.4. Bagan Kerja untuk Pra-pemrosesan

```
Rows missing BOTH fields : 362
Rows missing EXACTLY ONE : 15
```

Gambar 3.5. Total nilai yang hilang dalam dataset, baik pada kolom gender, nama, atau keduanya

```
Rows with special characters in 'CustomerName': 449
```

Gambar 3.6. Total karakter khusus dalam nama pelanggan

Meskipun *dataset* mencakup total 100.342 entri, tidak semuanya bersih atau layak digunakan. Beberapa entri tidak lengkap, seperti tidak memiliki nama pelanggan atau label *gender* yang belum ditentukan. Seperti yang ditunjukkan pada Gambar 3.5, terdapat 15 entri yang kehilangan salah satu informasi (nama atau *gender*), dan 362 entri yang tidak memiliki keduanya. Untuk menjaga

keandalan dan validitas *dataset* dalam pelatihan model, salah satu strategi yang paling tepat adalah menghapus entri-entri yang memiliki *missing value*. Meskipun ada pendekatan lain, karena jumlah data yang bermasalah sangat kecil, penghapusan data tidak akan berdampak signifikan terhadap kualitas model. Sebaliknya, membiarkan data yang tidak lengkap dapat menimbulkan konsekuensi serius, seperti prediksi yang tidak akurat atau bahkan *overfitting* [27].

Masalah *missing value* menjadi semakin krusial karena proyek ini menggunakan IndoBERT yang tersedia melalui platform HuggingFace. Tokenizer dari HuggingFace memiliki ekspektasi bahwa seluruh input bertipe *string*. Apabila ada tipe data lain seperti *integer*, *float*, atau *NoneType*, maka proses tokenisasi akan gagal dan menimbulkan *ValueError* [28]. Meskipun *empty string* (”” atau ”) secara teknis dianggap bertipe *string*, penggunaannya tetap dihindari karena tidak mengandung informasi bermakna dan dapat mengurangi kualitas representasi model. Oleh karena itu, menjaga kebersihan data, terutama dari nilai kosong atau tidak valid, adalah langkah penting yang tidak bisa diabaikan.

Masalah lain yang ditemukan dalam *dataset* adalah adanya karakter khusus dalam beberapa nama, seperti &, *, dan $. Setelah diskusi dengan mentor dan supervisor, Karakter-karakter ini dapat mengganggu proses pengolahan data dan berdampak negatif terhadap performa model jika tidak ditangani dengan benar. Namun, penting untuk dicatat bahwa tidak semua karakter khusus harus dihapus. Beberapa karakter seperti titik (.), apostrof ('), tanda hubung (-), dan tanda kutip (") umumnya masih dapat diterima dan sering muncul dalam nama yang valid. Berdasarkan Gambar 3.6, terdapat 449 nama dalam *dataset* yang mengandung karakter khusus seperti ini.

Terakhir, dalam pemrosesan bahasa alami, adalah praktik umum untuk mengubah semua huruf kapital menjadi huruf kecil sebelum diproses lebih lanjut. Ini karena model biasanya membedakan antara huruf kapital dan huruf kecil—misalnya, ‘A’ dan ‘a’ dianggap sebagai karakter yang berbeda [29]. Meskipun IndoBERT mungkin memiliki mekanisme internal untuk menangani sensitivitas huruf kapital, mengubah seluruh teks menjadi huruf kecil tetap menjadi pendekatan yang lebih aman dan konsisten. Langkah ini membantu menjaga keseragaman dan mengurangi risiko kesalahan selama pelatihan.

Potongan kode di bawah ini menunjukkan bagaimana permasalahan ini ditangani dengan menyaring data yang bermasalah:

```
1 import pandas as pd
2 import re
```

```
3
4  # Load the dataset
5  df = pd.read_csv('KGGenderDataset.csv')
6
7  # Total rows before any cleaning
8  initial_count = df.shape[0]
9  print(f"Total rows before cleaning: {initial_count}\n")
10
11 # 1. Build masks for missing or empty values
12 name_missing = df['CustomerName'].isna() | (df['CustomerName'].
       astype(str).str.strip() == '')
13 gender_missing = df['Gender'].isna() | (df['Gender'].astype(str).
       str.strip() == '')
14
15 # a. Compute how many are missing both vs. exactly one
16 both_missing = name_missing & gender_missing
17 one_missing = name_missing ^ gender_missing
18
19 print(f"Rows missing BOTH fields : {both_missing.sum()}")
20 print(f"Rows missing EXACTLY ONE : {one_missing.sum()}\n")
21
22 # (Optional) See a few examples of each
23 print("Examples missing BOTH:")
24 print(df[both_missing].head(5), "\n")
25
26 print("Examples missing EXACTLY ONE:")
27 print(df[one_missing].head(5), "\n")
28
29 # 3. Now drop all rows that are missing anything
30 df = df.drop(df[both_missing | one_missing].index)
31
32 # Define the pattern for disallowed special characters
33 special_char_pattern = re.compile(r"[^A-Za-z0-9\s.\'-]")
34
35
36
37 # Identify rows with special characters in `CustomerName`
38 special_char_mask = df['CustomerName'].astype(str).str.contains(
       special_char_pattern)
39 print(f"Rows with special characters in `CustomerName`: {
       special_char_mask.sum()}\n")
40
41 # Remove those rows
```

```
df = df[~special_char_mask].copy()

# Convert 'CustomerName' to lowercase
df['CustomerName'] = df['CustomerName'].str.lower()

# Final counts
final_count = df.shape[0]
deleted_count = initial_count - final_count
print(f"Total rows after cleaning: {final_count}")
print(f"Total rows deleted during cleaning: {deleted_count}\n")

# Export the cleaned DataFrame
df.to_csv('KGGenderDataset_Cleaned.csv', index=False, encoding='
    utf-8')
```

Kode 3.2: Potongan kode untuk membersihkan dataset KGGenderDataset

Seperti yang ditunjukkan pada Kode 3.2, kode lengkap untuk melakukan pra-pemrosesan dataset disajikan. Berikut ini adalah penjelasan dari setiap langkah penting dalam proses tersebut:

1. **Library yang perlu di import**

```
import pandas as pd
import re
```

Kode 3.3: Potongan kode untuk library yang diperlukan

Berikut ini alasan kenapa perlu diimport library berikut seperti di kode 3.3:

(a) *pandas*

pustaka ini menyediakan struktur `DataFrame` yang sangat efisien untuk memuat, memanipulasi, dan menyimpan data tabular—misalnya, fungsi `pd.read_csv()` untuk membaca file CSV, metode `.shape` untuk memeriksa jumlah baris dan kolom, serta kemampuan melakukan filtering dan pembersihan data melalui teknik boolean indexing dan fungsi string seperti `.str.strip()` dan `.str.lower()`, sehingga proses seperti menghitung jumlah baris sebelum dan sesudah pembersihan, membangun masker untuk nilai hilang, dan mengekspor kembali `DataFrame` yang sudah dibersihkan dapat dilakukan dalam satu baris kode saja tanpa perlu menulis loop manual sehingga secara

signifikan meminimalkan boilerplate dan memudahkan pemeliharaan kode [30].

(b) *re*

modul ini diperlukan untuk membuat pola ekspresi reguler yang dapat mendeteksi dan menyaring karakter tidak diinginkan dalam teks—misalnya, dengan menggunakan `re.compile(r"[^A-Za-z0-9\s\.\'-]")`, kita dapat mendefinisikan karakter yang dianggap tidak valid pada kolom `CustomerName`, kemudian memanfaatkan `str.contains(special_char_pattern)` untuk membuat masker boolean yang menandai setiap baris yang mengandung simbol-simbol terlarang dan menghapusnya secara efisien, tanpa perlu mengecek karakter per karakter secara manual sehingga memudahkan pembersihan data teks [31].

2. **Mendefinisikan Dataset**

```
# Load the dataset
df = pd.read_csv('KGGenderDataset.csv')
```

Kode 3.4: Potongan kode untuk mendefinisikan Dataset

Dalam kode 3.4, Dataset diberikan ke dalam variabel bernama `df`. Untuk memudahkan, sepanjang laporan ini, dataset yang sebelumnya disebut `KGGenderDataset` akan direpresentasikan sebagai `df`.

3. **Menangani Data yang Hilang**

```
# 1. Build masks for missing or empty values
name_missing = df['CustomerName'].isna() | (df['CustomerName'
    ].astype(str).str.strip() == '')
gender_missing = df['Gender'].isna() | (df['Gender'].astype(
    str).str.strip() == '')

# a. Compute how many are missing both vs. exactly one
both_missing = name_missing & gender_missing
one_missing = name_missing ^ gender_missing

print(f"Rows missing BOTH fields : {both_missing.sum()}")
print(f"Rows missing EXACTLY ONE : {one_missing.sum()}\n")

```

```
# (Optional) See a few examples of each
print("Examples missing BOTH:")
print(df[both_missing].head(5), "\n")

print("Examples missing EXACTLY ONE:")
print(df[one_missing].head(5), "\n")

# 3. Now drop all rows that are missing anything
df = df.drop(df[both_missing | one_missing].index)
```

Kode 3.5: Potongan kode untuk menghilangkan data yang NULL

Dalam kode 3.5, Data yang hilang ditangani dengan menggunakan fungsi `.isna()` dan juga pemeriksaan terhadap string kosong (""). Baris-baris yang memiliki nama atau label gender yang kosong akan dihapus menggunakan metode `.drop()` dengan mengacu pada indeks baris yang memiliki nilai kosong tersebut. Langkah ini memastikan bahwa hanya data yang lengkap dan valid yang akan digunakan dalam analisis.

4. **Mengidentifikasi dan Menghapus Karakter Khusus**

```
# Define the pattern for disallowed special characters
special_char_pattern = re.compile(r"[^A-Za-z0-9\s.\'-]")



# Identify rows with special characters in 'CustomerName'
special_char_mask = df['CustomerName'].astype(str).str.
    contains(special_char_pattern)
print(f"Rows with special characters in 'CustomerName': {
    special_char_mask.sum()}\n")

# Remove those rows
df = df[~special_char_mask].copy()
```

Kode 3.6: Potongan kode untuk membersihkan data yang memiliki karakter khusus

Di kode 3.6, Karakter khusus dalam nama diidentifikasi menggunakan metode `.contains()` dari pustaka `re` (regular expressions). Metode ini akan mengembalikan nilai Boolean:

(a) `True` menunjukkan bahwa nama mengandung karakter khusus.

(b) `False` menunjukkan bahwa nama tidak mengandung karakter khusus.

Setelah karakter khusus diidentifikasi, baris-baris yang mengandung karakter tersebut akan dihapus menggunakan ekspresi `special_char_mask`. Simbol tilde (`~`) pada kode `df = df[~special_char_mask].copy()` berfungsi sebagai operator NOT, yang membalik nilai Boolean — sehingga hanya baris yang tidak memiliki karakter khusus yang akan dipertahankan dan di salin.

5. **Mengubah Menjadi Huruf Kecil**

```
# Convert 'CustomerName' to lowercase
df['CustomerName'] = df['CustomerName'].str.lower()
```

Kode 3.7: Potongan kode untuk mengubah semua huruf jadi huruf kecil

Seperti kode 3.7, untuk menghindari perbedaan perlakuan antara huruf besar dan kecil, semua huruf pada nama akan diubah menjadi huruf kecil menggunakan metode `.str.lower()`. Langkah ini memastikan bahwa nama seperti "Ali" dan "ALI" akan dianggap sama oleh model selama pelatihan dan analisis.

6. **Menyimpan Dataset yang Sudah Diproses**

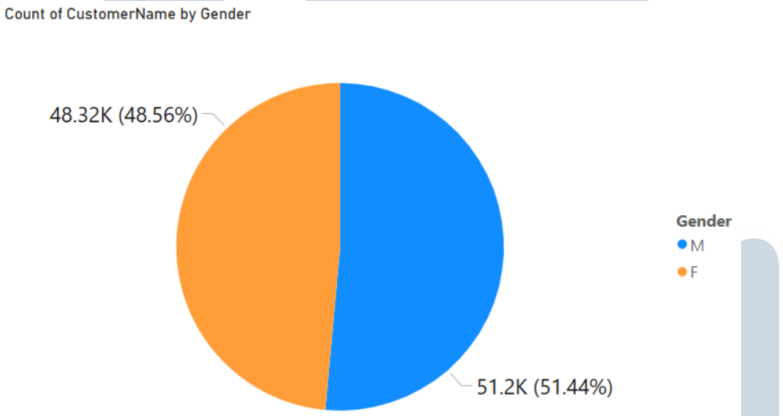
```
# Export the cleaned DataFrame
df.to_csv('KGGenderDataset_Cleaned.csv', index=False,
    encoding='utf-8')
```

Kode 3.8: Potongan kode untuk menyiman dataset yang sudah diproses

Setelah semua langkah pra-pemrosesan selesai dilakukan, mengikuti kode 3.8 dataset yang sudah dibersihkan dapat disimpan untuk digunakan di masa mendatang. Menyimpan data yang telah diproses sangat disarankan agar tidak perlu mengulangi proses ini setiap kali data digunakan kembali atau ketika model dilatih ulang.

	CustomerName	Gender
1	gerall owen	M
2	mardatomi maselta	M
3	stefanus suyono	M
4	mohamad iwan cahyadi	M
5	hkf official	F
6	rizky bramantyo	M
7	enya grace	F
8	nhia pram	F
9	sintya handayani	F
10	aril al kautsar	M
11	ragel pranata	F
12	rando lubis	M
13	dwi kurnia asih	F
14	meisyani fadila	F
15	omega purnomo	M

Gambar 3.7. Tabel dataset yang sudah bersih



Gambar 3.8. pie chart distribusi kelamin pada dataset yang sudah bersih

Seperti yang ditunjukkan pada Gambar 3.7, berikut adalah contoh tampilan dari dataset yang telah dibersihkan. Setelah melalui proses pra-pemrosesan, total data yang tersedia adalah sebanyak 99.516 entri. Berdasarkan Gambar 3.8, sebanyak 51,44% dari data (51.196 entri) merupakan nama laki-laki, sedangkan sisanya, yaitu 48,56% (48.320 entri), adalah nama perempuan. Meskipun terdapat data yang dihapus selama proses pembersihan, langkah ini penting dilakukan agar model tidak bingung oleh data yang tidak lengkap atau tidak konsisten, yang pada akhirnya dapat menurunkan tingkat akurasi model.

```python
from transformers import BertTokenizer,
    BertForSequenceClassification
import pandas as pd

# Load your dataset (replace 'your_file.csv' with your actual file
    )
df = pd.read_csv('KGGenderDataset_Cleaned.csv')

# 1. Count your classes
classes = df["Gender"].unique()  # e.g., ['F', 'M', 'U']
n_classes = len(classes)

print("Number of classes:", n_classes)

# 2. Load tokenizer and model
tokenizer = BertTokenizer.from_pretrained("indobenchmark/indobert-
    base-p1")
model = BertForSequenceClassification.from_pretrained("
    indobenchmark/indobert-base-p2", num_labels=n_classes)
```

Kode 3.9: Potongan kode untuk label M dan F menjadi class 1 dan 0

Selanjutnya, sangat penting untuk mengimpor model *IndoBERT* ke dalam proyek, seperti yang ditunjukkan pada Kode 3.9. Kita perlu mengimpor baik *tokenizer*nya maupun *model*-nya. *Tokenizer* bertugas untuk memecah setiap kata atau sub-kata menjadi *token* dan memberikan ID berdasarkan kosakata yang dibentuk selama proses *pre-training* [28]. Di sisi lain, *model* memuat bobot-bobot yang telah dilatih, seperti bobot *embedding*, bobot dalam *multi-head attention*, dan lainnya [32].

Untuk mendapatkan *tokenizer*, kita menggunakan `BertTokenizer`, sedangkan untuk *model*-nya, kita memakai `BertForSequenceClassification`. Karena proyek ini merupakan tugas klasifikasi biner (memprediksi *gender*: 'M' atau 'F'), maka parameter `num_labels` diatur ke 2, agar *IndoBERT* mengetahui bahwa hanya ada dua kelas yang perlu diprediksi. Kepala (*head*) *model* juga bisa diubah sesuai dengan jenis tugasnya, seperti untuk *question answering*, *masked language modeling*, dan tugas *NLP* lainnya.

Mengacu pada kode 3.9, terdapat dua versi *model IndoBERT* yang digunakan, yaitu `indobert-base-p1` dan `indobert-base-p2`. Keduanya merupakan *checkpoint* dari fase pelatihan yang berbeda, dengan `p2` sebagai versi yang lebih baru dan telah mengalami peningkatan performa. IndoBERT fase 2 atau

`indobert-base-p2` untuk melakukan prediksi dengan hasil yang lebih memandai. Tokenizer yang digunakan pada versi `p1` dan `p2` tetap sama, karena seperti dijelaskan dalam makalah IndoNLU [23], perbedaan antara kedua fase tersebut hanya terletak pada proses pelatihannya. Keduanya menggunakan *vocabulary* berukuran 30,522 yang dibangun dengan algoritma SentencePiece dan byte pair encoding (BPE).

```
# Prepare lists of texts and labels
texts = df['CustomerName'].tolist()
labels = df['Gender'].tolist()
```

Kode 3.10: Kode ini mengubah kolom data menjadi list Python dan menyimpannya ke dalam variabel baru bernama texts dan labels.

Mengacu pada Kode 3.10, kolom-kolom dalam *DataFrame* dikonversi menjadi *list* Python biasa. Langkah ini diperlukan karena kelas *Dataset* dan API *tokenizer* milik PyTorch mengharapkan input berupa urutan sederhana yang dapat diindeks, bukan objek *pandas* yang lebih kompleks. Dengan mengubah kolom *DataFrame* menjadi *list*, kita menghilangkan beban tambahan dan *metadata* dari *pandas*, sehingga struktur data menjadi lebih ringan. *List* ini juga memiliki panjang tetap dan mendukung pengindeksan berbasis *integer* dengan cepat, yang pada akhirnya membuat proses implementasi model menjadi lebih sederhana dan efisien.

```
# First split into train (80%) and test (20%)
train_texts, test_texts, train_labels, test_labels =
    train_test_split(
    texts, labels, test_size=0.2, random_state=42
)

# Then split the training set into train (60%) and validation
    (20%)
# Since train_texts is 80% of the original data, we take 25% of
    that to get 20%
train_texts, val_texts, train_labels, val_labels =
    train_test_split(
    train_texts, train_labels, test_size=0.25, random_state=42
)
```

Kode 3.11: Potongan kode untuk membagikan data menjadi data pelatihan

Terakhir, seperti yang ditunjukkan pada Gambar 3.11, dataset dibagi menjadi tiga bagian: data pelatihan, validasi, dan pengujian. Pembagian awal memisahkan dataset menjadi 80% untuk pelatihan dan 20% untuk pengujian.

Selanjutnya, data pelatihan dibagi kembali, dengan 60% tetap digunakan untuk pelatihan, sedangkan 20% sisanya digunakan untuk validasi [33]. Ini dilakukan untuk mencegah terjadinya overfitting jika keseluruhan dataset dipake sebagai data latih saja. Maka dengan itu, setiap bagian dari dataset ini memiliki fungsi spesifik dalam proses pelatihan model [34]:

1. **Training dataset** digunakan untuk melatih model agar mempelajari pola-pola yang ada dan menyesuaikan bobot internal berdasarkan data berlabel.
2. **Validation dataset** berfungsi sebagai uji coba selama pelatihan. Dataset ini membantu mengevaluasi performa model dan berguna untuk menyetel *hyperparameter* seperti *learning rate*, ukuran *batch*, dan jumlah *epoch*.
3. **Testing dataset** digunakan di akhir pelatihan untuk mengukur performa model terhadap data yang benar-benar belum pernah dilihat sebelumnya. Ini memberikan gambaran kesiapan model untuk digunakan dalam aplikasi dunia nyata.

```
import torch
from torch.utils.data import Dataset

class GenderDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_length=64):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length
        # Map string labels ("F", "M") to integers (0, 1)
        self.label2id = {"F": 0, "M": 1}

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = self.texts[idx]
        label_str = self.labels[idx]  # e.g., "F" or "M"
        label_int = self.label2id[label_str]  # convert to 0 or 1

        # Tokenize the text
        encoding = self.tokenizer(
            text,
```

```
            truncation=True,
            padding='max_length',
            max_length=self.max_length,
            return_tensors='pt'
        )

        # Remove the extra batch dimension
        encoding = {k: v.squeeze(0) for k, v in encoding.items()}

        # Add label to the encoding dictionary
        encoding['labels'] = torch.tensor(label_int, dtype=torch.
    long)

        return encoding
```

Kode 3.12: Potongan kode untuk mengubah kalimat jadi token dan label Laki-laki dan perempuan menjadi '1' dan '0'

Sebelum melakukan proses pelatihan dan pengujian, beberapa langkah persiapan perlu dilakukan untuk mempermudah proses. Seperti yang ditunjukkan pada Kode 3.12, langkah pertama adalah membuat kelas *tokenizer* khusus yang bertujuan untuk menyederhanakan dan mengotomatiskan proses tokenisasi. Hal ini sangat membantu, terutama dalam mengubah label *gender* dari 'F' dan 'M' menjadi bilangan bulat, di mana 'M' diubah menjadi 1 dan 'F' menjadi 0.

Berikut ini adalah penjelasan dari setiap baris kode yang ditampilkan:

1. `def __init__(self, texts, labels, tokenizer, max_length=64):`

```
import torch
from torch.utils.data import Dataset

class GenderDataset(Dataset):
    def __init__(self, texts, labels, tokenizer,
    max_length=64):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length
        # Map string labels ("F", "M") to integers (0,
    1)
        self.label2id = {"F": 0, "M": 1}
```

Kode 3.13: kode `def __init__` untuk class GenderDataset

Pada bagian ini [35], fungsi __init__ atau yang dikenal juga sebagai konstruktor—dibuat sebagai bagian dari pendekatan *Object-Oriented*

Programming (OOP). Fungsi ini memiliki peran penting dalam mendefinisikan struktur sebuah objek. Ketika objek dibuat dari sebuah kelas, fungsi __init__ akan dipanggil secara otomatis untuk menginisialisasi objek dengan variabel-variabel yang diperlukan. Dalam contoh yang diberikan, objek yang didefinisikan mencakup beberapa komponen: *texts*, *labels*, *tokenizer*, *max_length*, dan *label2id*. Komponen-komponen ini disimpan sebagai bagian dari status internal objek dan bisa diakses nantinya. Fungsi __init__ pada dasarnya menetapkan apa saja yang harus dimiliki objek, sehingga ketika objek dibuat (atau "dipanggil menjadi ada"), kode sudah tahu bagaimana cara menginisialisasi dan menyusun datanya.

2. `def __len__(self):`

```
import torch
from torch.utils.data import Dataset

def __len__(self):
        return len(self.texts)
```

Kode 3.14: kode `def __len__` untuk class GenderDataset

Kode 3.14, yang terdapat di dalam kelas, berfungsi untuk mengembalikan jumlah total contoh (data) yang ada di dalam *dataset*. Fungsi ini sangat penting dalam berbagai situasi praktis. Misalnya, fungsi ini memungkinkan pengguna untuk memantau perkembangan proses pelatihan. Selain itu, beberapa pustaka pihak ketiga sering memanggil `len(dataset)` untuk melakukan alokasi memori sebelumnya atau menampilkan statistik *dataset*. Fungsi ini juga penting bagi *DataLoader* karena membantu menentukan berapa banyak *batch* yang dapat dibuat dari *dataset* tersebut [36].

3. `def __getitem__(self, idx):`

```

 def __getitem__(self, idx):
        text = self.texts[idx]
        label_str = self.labels[idx]  # e.g., "F" or "M"
        label_int = self.label2id[label_str]  # convert
   to 0 or 1

        # Tokenize the text
        encoding = self.tokenizer(
            text,
            truncation=True,
            padding='max_length',
            max_length=self.max_length,
```

```
            return_tensors='pt'
        )

        # Remove the extra batch dimension
        encoding = {k: v.squeeze(0) for k, v in encoding
    .items()}

        # Add label to the encoding dictionary
        encoding['labels'] = torch.tensor(label_int,
    dtype=torch.long)

        return encoding
```

Kode 3.15: kode `def __len__` untuk class GenderDataset

Dalam kode 3.15, proses dimulai dengan mengambil satu contoh data mentah—yakni sebuah string teks dan label gender yang sesuai—dari *dataset*. Dengan menggunakan indeks `idx`, kita mengambil `self.texts[idx]` dan `self.labels[idx]` untuk memperoleh pasangan seperti kalimat "*He plays soccer*" dan label "M". Pada tahap ini, data masih dalam bentuk aslinya—teks biasa dan label yang mudah dibaca manusia—sehingga perlu dikonversi ke bentuk *tensor* numerik yang dibutuhkan oleh *BERT* dan *PyTorch*.

Pertama, label gender berupa string dikonversi ke bilangan bulat menggunakan *dictionary* `label2id` yang didefinisikan dalam `__init__`. Sebagai contoh, label "M" diubah menjadi 1, dan "F" menjadi 0. Konversi ini penting karena model *deep learning* hanya dapat bekerja dengan data numerik, dan label numerik diperlukan untuk menghitung *loss* selama pelatihan.

Selanjutnya, teks input di-tokenisasi menggunakan *tokenizer* IndoBERT dengan parameter seperti `truncation=True`, `padding='max_length'`, dan panjang maksimum tertentu (`max_length`). *Truncation* memastikan bahwa kalimat yang terlalu panjang dipotong agar sesuai dengan batas maksimum input model. *Padding* mengisi kalimat yang lebih pendek dengan token `[PAD]` khusus agar semua input dalam satu *batch* memiliki panjang yang sama. Konsistensi ini memungkinkan *DataLoader PyTorch* menggabungkannya dalam bentuk *tensor* berukuran `(batch_size, max_length)`.

Karena kita menggunakan `return_tensors='pt'`, *tokenizer* akan mengembalikan *tensor PyTorch* yang dibungkus dengan dimensi *batch* berukuran 1 (yaitu berbentuk `(1, max_length)`). Untuk menghapus dimensi ekstra ini, digunakan `.squeeze(0)`, yang menyederhanakan

bentuknya menjadi `(max_length)`. Tanpa langkah ini, BERT mungkin tidak dapat menerima input dengan bentuk yang tidak sesuai saat pelatihan.

Terakhir, *tensor* label numerik dimasukkan ke dalam *dictionary* yang sama di bawah `key "labels"`. *Dictionary* ini sekarang berisi semua yang dibutuhkan model: `input_ids`, `attention_mask`, dan `labels`. Saat digabungkan dalam bentuk *batch*, *input* akan berbentuk `(batch_size, max_length)` dan *label* berbentuk `(batch_size)`, sehingga siap untuk proses pelatihan dan perhitungan *loss*.

```

# Create dataset instances for each split
train_dataset = GenderDataset(train_texts, train_labels,
    tokenizer)
val_dataset = GenderDataset(val_texts, val_labels, tokenizer
    )
test_dataset = GenderDataset(test_texts, test_labels,
    tokenizer)
```

Kode 3.16: Kode yang membantu pelabelan untuk dataset pelatihan

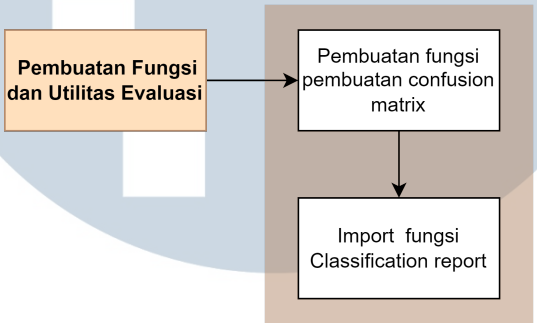
Seperti ditunjukkan pada Kode 3.16, setelah kelas `GenderDataset` didefinisikan, penggunaannya menjadi sangat mudah—cukup dengan membuat *instance* dari kelas tersebut menggunakan daftar teks dan label yang diambil dari sebuah *dataset*. Proses ini secara otomatis memanggil metode `__init__()` dari kelas tersebut, yang bertugas menyimpan data input dan konfigurasi *tokenizer*. Walaupun metode seperti `__len__()` dan `__getitem__()` tidak dipanggil secara langsung oleh pengguna, keduanya sangat penting dalam proses pelatihan.

Ketika pelatihan dimulai, kelas `Trainer` dari *Hugging Face* akan secara otomatis membuat `DataLoader` dari *PyTorch* untuk mengatur pemuatan data dalam bentuk *batch* ke dalam model [37]. `DataLoader` ini merupakan bagian dari utilitas inti *PyTorch* dan bertugas memanggil `__len__()` untuk mengetahui ukuran *dataset* serta jumlah *batch*, dan `__getitem__()` untuk mengambil, *tokenize*, dan memberi label pada setiap contoh pelatihan berdasarkan indeks [38]. Kedua metode ini wajib dimiliki oleh apa yang disebut *PyTorch* sebagai *map-style dataset*—yaitu *dataset* yang dapat diindeks dan efisien untuk digunakan jika seluruh data dapat dimuat ke dalam memori atau diakses melalui indeks secara cepat.

Namun, jika ukuran *dataset* terlalu besar untuk dimuat sepenuhnya ke dalam memori, maka pendekatan lain diperlukan, yaitu *iterable-style dataset*. *Dataset* jenis ini menggunakan metode `__iter__()` untuk menghasilkan satu sampel pada satu waktu, dan sangat berguna untuk pemrosesan data skala besar atau *streaming* yang tidak memungkinkan memuat data sekaligus [38].

3.3.3 Fungsi dan Utilitas Evaluasi

Evaluasi model sangat penting untuk mengetahui performa, kelemahan, dan keunggulan dari sebuah model. Tanpa adanya evaluasi, akan sulit menentukan apakah model tersebut sudah layak untuk digunakan atau belum. Oleh karena itu, sebuah fungsi Python akan dibuat yang berisi metrik dan alat evaluasi, sehingga hanya dengan sekali eksekusi, seluruh metrik yang dibutuhkan dapat dihasilkan. Karena tugas yang dikerjakan adalah klasifikasi *gender* berdasarkan nama, maka *confusion matrix* dapat digunakan untuk mengevaluasi kelayakan model [39]. Selain itu, fungsi `classification_report` juga akan digunakan untuk mengevaluasi model berdasarkan berbagai metrik yang tersedia [40]. Bagan kerja dari pembuatan fungsi ini dapat dilihat pada Gambar 3.9.



Gambar 3.9. Bagan kerja untuk fungsi dan utilitas evaluasi

```
def evaluate_and_print(dataset, dataset_name):
    print(f"=== Evaluation on {dataset_name} Set ===")
    eval_results = trainer.predict(dataset)
    preds = torch.argmax(torch.tensor(eval_results.predictions), dim=1).numpy()
    true_labels = eval_results.label_ids

    # Confusion Matrix
    cm = confusion_matrix(true_labels, preds)
    print("Confusion Matrix:")
    print(cm)
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["F", "M"], yticklabels=["F", "M"])
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title(f"Confusion Matrix - {dataset_name} Set")
    plt.show()

    # Classification Report
    print("Classification Report:")
    print(classification_report(true_labels, preds, target_names=["F", "M"]))
```

Kode 3.17: Kode untuk Eveluasi model IndoBERT

Evaluasi terhadap performa model sangat penting untuk mengetahui seberapa efektif model tersebut. Seperti yang ditunjukkan pada Kode 3.17, sebuah fungsi evaluasi kustom dibuat untuk mengukur performa model pada *dataset* pelatihan, validasi, dan pengujian. Fungsi ini membantu memberikan cara yang konsisten dan terstruktur dalam mengevaluasi performa model di berbagai pembagian data. Berikut ini penjelasan dari setiap potongan kode yang disusun dalam fungsi evaluasi tersebut:

1. **Evaluasi untuk testing dataset**

```
1
2 print(f"=== Evaluation on {dataset_name} Set ===")
3     eval_results = trainer.predict(dataset)
4     preds = torch.argmax(torch.tensor(eval_results.
    predictions), dim=1).numpy()
5     true_labels = eval_results.label_ids
```

Kode 3.18: Kode untuk persiapan evaluasi untuk testing dataset

Berikut adalah penjelasan untuk setiap baris kode yang digunakan dalam fungsi evaluasi:

(a) `eval_results = trainer.predict(dataset)`

Baris ini menjalankan model dalam mode evaluasi terhadap *dataset* yang diberikan. Fungsi ini mengembalikan objek `PredictionOutput` yang berisi:

i. `predictions`: Logit mentah dari setiap sampel (berbentuk (N, C), di mana N adalah jumlah sampel dan C adalah jumlah kelas).

ii. `label_ids`: Label sebenarnya untuk setiap sampel (berbentuk $(N,)$).

iii. `metrics`: Metrik evaluasi bawaan seperti *loss*.

(b) `preds = torch.argmax(torch.tensor(eval_results.predictions), dim=1).numpy()`

Baris ini mengubah logit mentah menjadi label kelas prediksi dengan cara:

i. Mengubah array `predictions` (berbentuk NumPy) menjadi *tensor* PyTorch.

ii. Menggunakan fungsi `argmax` pada dimensi kelas (`dim=1`) untuk memilih kelas dengan skor tertinggi pada setiap sampel.

iii. Mengubah hasilnya kembali menjadi array NumPy dengan bentuk $(N,)$.

(c) `true_labels = eval_results.label_ids`

Baris ini mengekstrak label sebenarnya dari `PredictionOutput`. Label ini sudah dalam format array NumPy dan akan digunakan untuk menghitung metrik evaluasi seperti akurasi, presisi, *recall*, atau skor F1.

2. **Bikin confusion Matrix**

```
1
2 cm = confusion_matrix(true_labels, preds)
3     print("Confusion Matrix:")
4     print(cm)
5     sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
    xticklabels=["F", "M"], yticklabels=["F", "M"])
6     plt.xlabel("Predicted Label")
7     plt.ylabel("True Label")
8     plt.title(f"Confusion Matrix - {dataset_name} Set")
9     plt.show()
```

Kode 3.19: Kode untuk persiapan evaluasi untuk testing dataset

Dalam tugas klasifikasi biner, salah satu cara paling sederhana dan informatif untuk mengevaluasi performa model adalah dengan menggunakan *confusion matrix* (matriks kebingungan). Matriks ini memberikan rincian hasil prediksi dalam empat kategori utama [41]:

(a) **True Positives (TP)**: Kasus di mana model berhasil memprediksi kelas positif dengan benar.

(b) **True Negatives (TN)**: Kasus di mana model berhasil memprediksi kelas negatif dengan benar.

(c) **False Positives (FP)**: Kasus negatif yang secara keliru diprediksi sebagai positif.

(d) **False Negatives (FN)**: Kasus positif yang secara keliru diprediksi sebagai negatif.

Sebagai contoh, jika kita menyatakan laki-laki sebagai kelas 1 (kelas ”positif”) dan perempuan sebagai kelas 0 (kelas ”negatif”):

(a) Untuk mengetahui seberapa baik model mengidentifikasi laki-laki, kita perhatikan jumlah TP. Kesalahan dalam memprediksi laki-laki akan muncul dalam jumlah FN.

(b) Untuk mengetahui seberapa baik model mengidentifikasi perempuan, kita lihat jumlah TN. Kesalahan dalam memprediksi perempuan akan muncul dalam jumlah FP.

Dengan cara ini, *confusion matrix* memudahkan kita untuk memahami jumlah prediksi yang benar maupun jenis kesalahan yang dilakukan model terhadap masing-masing kelas. Ini sangat berguna terutama ketika menangani ketidakseimbangan kelas atau kelemahan tertentu pada performa model.

3. **Bikin report klasifikasi**

```
1
2 print("Classification Report:")
3     print(classification_report(true_labels, preds,
    target_names=["F", "M"]))
```

Kode 3.20: Kode ini untuk buat report klasifikasi

Kode 3.20 menampilkan laporan performa model menggunakan `classification_report`, yang menyediakan berbagai metrik evaluasi utama untuk setiap kelas: akurasi, presisi, recall, dan F1-score. Selain itu, laporan ini juga mencakup *macro average* dan *weighted average* untuk memberikan gambaran umum tentang efektivitas model secara keseluruhan [42]:

(a) *Accuracy*

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

Akurasi mengukur seberapa sering model membuat prediksi yang benar, baik untuk kelas perempuan maupun laki-laki. Meskipun merupakan metrik yang umum digunakan, akurasi bisa menyesatkan jika distribusi data tidak seimbang.

(b) *Precision*

$$\text{Precision}_{\text{class}} = \frac{TP_{\text{class}}}{TP_{\text{class}} + FP_{\text{class}}} \tag{3.2}$$

Presisi menunjukkan seberapa andal prediksi model terhadap kelas tertentu. Presisi yang tinggi berarti model jarang salah mendeteksi kasus negatif sebagai positif. Dalam klasifikasi biner:

i. Untuk menghitung presisi kelas **perempuan**, anggap perempuan sebagai kelas positif.

ii. Untuk menghitung presisi kelas **laki-laki**, anggap laki-laki sebagai kelas positif.

(c) *Recall*

$$\text{Recall}_{\text{class}} = \frac{TP_{\text{class}}}{TP_{\text{class}} + FN_{\text{class}}} \tag{3.3}$$

Recall mengukur seberapa baik model menemukan semua kasus positif yang sebenarnya. Recall yang tinggi berarti model tidak banyak melewatkan kasus positif. Ini penting dalam konteks di mana kesalahan dalam mengabaikan kasus positif berdampak besar.

(d) *F1 Score*

$$\text{F1}_{\text{class}} = \frac{2 \times \text{Precision}_{\text{class}} \times \text{Recall}_{\text{class}}}{\text{Precision}_{\text{class}} + \text{Recall}_{\text{class}}} \tag{3.4}$$

F1 score adalah rata-rata harmonik antara precision dan recall. Skor ini berguna ketika data tidak seimbang, karena membantu menjaga keseimbangan antara keduanya. Trade-off umum antara precision dan recall adalah:

i. Precision tinggi $\Rightarrow$ model lebih selektif, risiko FN meningkat.

ii. Recall tinggi $\Rightarrow$ model lebih longgar, risiko FP meningkat.

Seperti sebelumnya, F1 score dihitung dengan menganggap satu kelas sebagai positif.

(e) *Macro Average*

$$\text{Macro Average} = \frac{1}{K} \sum_{i=1}^{K} M_i \tag{3.5}$$

Di mana M_i adalah nilai metrik (precision, recall, atau F1) untuk kelas ke-i, dan K adalah jumlah total kelas. Macro average mengambil rata-rata sederhana dari semua kelas tanpa mempertimbangkan ukuran kelas, sehingga cocok digunakan saat semua kelas dianggap sama penting.

(f) *Weighted Average*

$$\text{Weighted Average} = \frac{\sum_{i=1}^{K} \text{support}_i \cdot M_i}{\sum_{i=1}^{K} \text{support}_i} \tag{3.6}$$

Di mana support_i adalah jumlah data aktual di kelas ke-i, dan M_i adalah nilai metrik untuk kelas ke-i. Weighted average mempertimbangkan distribusi kelas, memberikan bobot lebih besar pada kelas yang memiliki lebih banyak sampel. Misalnya, jika terdapat 12.000 data perempuan dan 8.000 laki-laki, maka performa model terhadap perempuan akan lebih mempengaruhi rata-rata tertimbang.

Sebagai contoh perhitungan beberapa metrik evaluasi, misalnya *accuracy*, *precision*, dan *recall*, kita gunakan data berjumlah total 600 orang. Dalam data tersebut, terdapat 300 laki-laki dan 300 perempuan yang masing-masing mewakili dua kelas: kelas 1 untuk laki-laki dan kelas 0 untuk perempuan. Setelah dilakukan prediksi oleh model, dari 300 laki-laki, sebanyak 270 berhasil diprediksi dengan benar sebagai laki-laki, sedangkan 30 lainnya salah diklasifikasikan sebagai perempuan. Di sisi lain, dari 300 perempuan, model berhasil memprediksi 280 orang dengan benar sebagai perempuan, sementara 20 sisanya salah diklasifikasikan sebagai laki-laki. Berdasarkan hasil tersebut, kita dapat membentuk *confusion matrix* untuk menggambarkan kinerja model klasifikasi:

Tabel 3.1. Confusion Matrix dengan Penandaan Klasifikasi (Kelas 1: Laki-laki, Kelas 0: Perempuan)

	Prediksi 1 (Laki-laki)	**Prediksi 0 (Perempuan)**
Actual 1 (Laki-laki)	270 (TP)	30 (FN)
Actual 0 (Perempuan)	20 (FP)	280 (TN)

Berdasarkan Tabel 3.1, berikut ini disajikan perhitungan metrik evaluasi umum—seperti *accuracy*, *precision*, *recall*, dan *F1-score*—serta agregasi hasil dengan *macro average* dan *weighted average* untuk kedua kelas:

(a) *Akurasi*

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} = \frac{270+280}{270+280+20+30} = \frac{550}{600} = 0.917$$

(b) *Perhitungan untuk Kelas 1 (Laki-laki)*:

i. *Precision*

$$\text{Precision}_1 = \frac{TP_1}{TP_1 + FP_1} = \frac{270}{270+20} = \frac{270}{290} = 0.931$$

ii. *Recall*

$$\text{Recall}_1 = \frac{TP_1}{TP_1 + FN_1} = \frac{270}{270+30} = \frac{270}{300} = 0.900$$

iii. *F1-Score*

$$\text{F1}_1 = \frac{2 \times \text{Precision}_1 \times \text{Recall}_1}{\text{Precision}_1 + \text{Recall}_1} = \frac{2 \times 0.931 \times 0.900}{0.931 + 0.900} = 0.915$$

(c) *Perhitungan untuk Kelas 0 (Perempuan)*:

i. *Precision*

$$\text{Precision}_0 = \frac{TP_0}{TP_0 + FP_0} = \frac{280}{280+30} = \frac{280}{310} = 0.903$$

ii. *Recall*

$$\text{Recall}_0 = \frac{TP_0}{TP_0 + FN_0} = \frac{280}{280+20} = \frac{280}{300} = 0.933$$

iii. *F1-Score*

$$\text{F1}_0 = \frac{2 \times \text{Precision}_0 \times \text{Recall}_0}{\text{Precision}_0 + \text{Recall}_0} = \frac{2 \times 0.903 \times 0.933}{0.903 + 0.933} = 0.918$$

(d) *Macro Average*

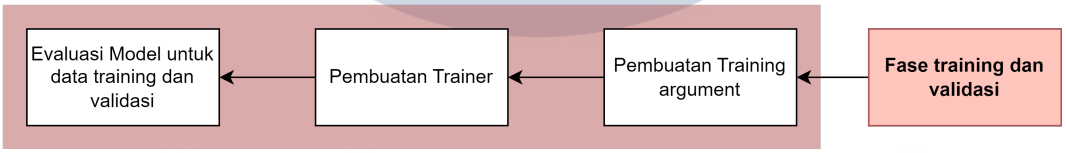
$$\text{Macro F1} = \frac{1}{2}(F1_1 + F1_0) = \frac{1}{2}(0.915 + 0.918) = 0.917$$

(e) *Weighted Average*

$$\text{Weighted F1} = \frac{300 \cdot 0.915 + 300 \cdot 0.918}{600} = \frac{274.5 + 275.4}{600} = \frac{549.9}{600} = 0.9165$$

3.3.4 Fase pelatihan dan validasi

Pada fase ini, model *IndoBERT* dilatih menggunakan *dataset* pelatihan dan dievaluasi terhadap dua subset data: pelatihan dan validasi. Proses pelatihan bertujuan untuk menyetel bobot-bobot internal model melalui mekanisme *fine-tuning*, sehingga model dapat menyesuaikan diri dengan tugas klasifikasi *gender* yang diberikan. Dalam tahap ini, model memperoleh akses terhadap fitur dan label dari data pelatihan, memungkinkan proses pembelajaran berlangsung secara eksplisit. Sebaliknya, data validasi digunakan untuk mengevaluasi kemampuan model dalam mengeneralisasi tanpa memberikan label secara langsung—model diminta untuk melakukan prediksi terhadap data yang belum pernah dilihat sebelumnya. Evaluasi pada data validasi berfungsi sebagai indikator kesiapan model sebelum memasuki fase pengujian akhir, serta membantu mendeteksi potensi *overfitting* atau *underfitting*. Selain itu, hasil dari validasi dapat dimanfaatkan untuk melakukan penyesuaian terhadap *hyperparameter* agar performa model tetap optimal [43]. Bagan kerja untuk fase pelatihan dan validasi bisa dilihat di gambar 3.10.



Gambar 3.10. Bagang kerja magang fase pelatihan dan validasi

```
1
2  # Define training arguments
3  training_args = TrainingArguments(
4      output_dir='./results',              # Output directory
5      num_train_epochs=3,                  # Adjust epochs as
       needed
6      per_device_train_batch_size=128,     # Adjust based on
       GPU/CPU
7      per_device_eval_batch_size=128,
8      evaluation_strategy="epoch",         # Evaluate at end of
       each epoch
9      save_total_limit=2,
10     learning_rate=2e-5,
11     weight_decay=0.01,
12     logging_dir='./logs'
13 )
14
15 # Create the Trainer
16 trainer = Trainer(
17     model=model,
18     args=training_args,
19     train_dataset=train_dataset,
```

```
    eval_dataset=val_dataset
)

# Start training
trainer.train()
```

Kode 3.21: Kode dibuat untuk fase training untuk model

Pada Kode 3.21, ditampilkan potongan kode yang menunjukkan bagaimana *model* diinisialisasi serta bagaimana proses pelatihan dilakukan. Berikut adalah penjelasan dari setiap potongan kode yang ditampilkan dalam gambar tersebut:

1. **Training argument**

```

# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',              # Output
    directory
    num_train_epochs=3,                  # Adjust epochs
    as needed
    per_device_train_batch_size=128,     # Adjust based
    on GPU/CPU
    per_device_eval_batch_size=128,
    evaluation_strategy="epoch",         # Evaluate at
    end of each epoch
    save_total_limit=2,
    learning_rate=2e-5,
    weight_decay=0.01,
    logging_dir='./logs'
)
```

Kode 3.22: Argument untuk memberikan kepada model untuk training

Dalam potongan kode 3.22, sejumlah *hyperparameter* untuk proses pelatihan diinisialisasi. Satu-satunya parameter yang diubah dari nilai *default* adalah *batch_size*, yang disesuaikan menjadi 128; sementara parameter lainnya tetap menggunakan nilai bawaan [37]. Berikut adalah penjelasan masing-masing *hyperparameter*:

(a) `output_dir='./results'`
Direktori tempat `Trainer` menyimpan *checkpoint* model, status *optimizer*, dan log pelatihan.

(b) `num_train_epochs=3`
Menentukan jumlah perulangan penuh terhadap data pelatihan. Ini mengontrol seberapa sering setiap sampel digunakan dalam proses pelatihan.

(c) `per_device_train_batch_size=128`
Jumlah sampel yang diproses secara bersamaan pada setiap GPU/CPU saat pelatihan. Ukuran *batch* efektif adalah nilai ini dikalikan dengan jumlah perangkat.

(d) `per_device_eval_batch_size=128`
Sama seperti di atas, tetapi digunakan untuk proses validasi. Nilai ini biasanya bisa lebih besar karena tidak perlu menyimpan *gradien*.

(e) `evaluation_strategy="epoch"`
Menentukan agar evaluasi dijalankan di akhir setiap *epoch*. Alternatif lainnya adalah `"no"` atau `"steps"` untuk jadwal evaluasi yang berbeda.

(f) `save_total_limit=2`
Menyimpan hanya dua *checkpoint* terbaru dan menghapus yang lama untuk menghemat ruang penyimpanan.

(g) `learning_rate=2e-5`
Menentukan langkah awal pembelajaran untuk *optimizer* AdamW. Nilai umum untuk *fine-tuning* berkisar antara `1e-5` hingga `5e-5`.

(h) `weight_decay=0.01`
Memberikan regularisasi L2 pada bobot (tidak termasuk *bias* dan `LayerNorm`) untuk membantu mengurangi *overfitting*.

(i) `logging_dir='./logs'`
Direktori tempat menyimpan log pelatihan dan evaluasi, yang bisa divisualisasikan menggunakan `TensorBoard` atau `Weights & Biases`.

2. **Kode Trainer**

```

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset
)
```

Kode 3.23: Kode untuk menginisiasi Trainer

Kode 3.23 adalah bagain yang menginisialisasi objek `Trainer`. Di dalam `Trainer`, terdapat beberapa atribut penting yang harus didefinisikan:

(a) `model=model`

Menentukan model Transformer yang sudah diinisialisasi sebelumnya (seperti model BERT untuk klasifikasi urutan) yang akan di-*fine-tune* selama proses pelatihan.

(b) `args=training_args`

Mengacu pada objek `TrainingArguments` yang berisi seluruh pengaturan *hyperparameter* penting—seperti ukuran *batch*, *learning rate*, jumlah *epoch* pelatihan, dan pengaturan penyimpanan *checkpoint*.

(c) `train_dataset=train_dataset`

Menyediakan instance `GenderDataset` untuk bagian pelatihan. Dataset ini akan digunakan oleh *DataLoader* internal PyTorch untuk mengambil dan membentuk *batch* dari sampel pelatihan.

(d) `eval_dataset=val_dataset`

Menyediakan instance `GenderDataset` untuk bagian validasi. Dataset ini digunakan untuk mengevaluasi kinerja model pada titik evaluasi yang telah ditentukan.

Terakhir, seperti yang ditunjukkan pada kode 3.21, proses pelatihan dapat dimulai dengan cukup memanggil `trainer.train()`. Setelah perintah ini dijalankan, pelatihan akan berlangsung secara otomatis, dan metrik seperti *loss* serta *accuracy* dapat dipantau selama proses tersebut berjalan. Waktu yang dibutuhkan untuk menyelesaikan pelatihan model ini berkisar antara 1 jam 30 menit hingga 2 jam 30 menit, tergantung pada beban komputasi dan didukung oleh spesifikasi perangkat keras yang telah disiapkan untuk memperlancar pelatihan.

```
1
2 evaluate_and_print(train_dataset, "Training")
3 evaluate_and_print(val_dataset, "Validation")
```

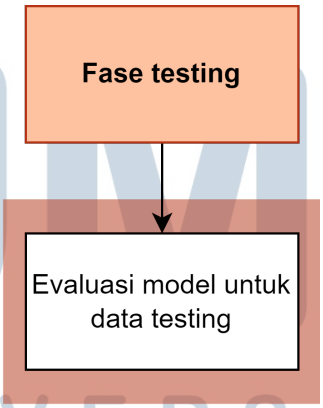
Kode 3.24: Kode untuk menjalankan evaluasi untuk training

Setelah proses pelatihan selesai dilakukan, evaluasi terhadap performa model pada data pelatihan dan validasi dapat dilakukan dengan memanggil fungsi `evaluate_and_print`, seperti ditunjukkan pada kode 3.24. Fungsi ini membutuhkan dua argumen utama, yaitu dataset yang akan dievaluasi dan judul yang relevan untuk membedakan hasil evaluasi, sehingga keluaran yang dihasilkan mudah diinterpretasikan. Selanjutnya, fungsi ini akan memanggil fungsi lain yang telah didefinisikan sebelumnya, yaitu `evaluate_and_print` seperti ditunjukkan pada kode 3.17. Fungsi tersebut secara otomatis akan menghasilkan laporan klasifikasi yang mencakup metrik akurasi, presisi, recall,

dan F1-score, serta menampilkan matriks kebingungan (*confusion matrix*) untuk dataset yang diberikan. Evaluasi ini penting sebagai bagian dari tugas khusus untuk memastikan bahwa model tidak hanya menghafal data pelatihan, tetapi juga mampu mengeneralisasi pola pada data validasi. Oleh karena itu, hasil evaluasi yang diperoleh melalui kedua fungsi ini disajikan dan dibahas secara lebih rinci pada anak subbagian 3.3.6 Evaluasi model di bagian 3.3.

3.3.5 Fase testing dan penyimpanan mode

Fase pengujian merupakan tahap evaluatif akhir yang dirancang untuk mengukur sejauh mana model mampu melakukan generalisasi terhadap data yang belum pernah dilihat sebelumnya. Berbeda dengan data pelatihan dan validasi yang digunakan selama proses pengembangan, data pengujian tidak terlibat dalam proses pembelajaran, sehingga memberikan indikator objektif terhadap kinerja model. Evaluasi pada fase ini sangat krusial karena mencerminkan kemampuan model dalam menangani data dunia nyata. Apabila model menunjukkan performa yang memadai pada data pengujian, maka dapat disimpulkan bahwa model tersebut memiliki potensi untuk digunakan secara operasional dalam konteks prediktif yang sesungguhnya [43]. Bagan kerja fase pengujian bisa dilihat pada gambar 3.11



Gambar 3.11. Bagang kerja magang fase testing

```
1
2 evaluate_and_print(test_dataset, "Test")
3
4 # --- Saving the Model and Tokenizer ---
5
6 # Save the fine-tuned model using Hugging Face's save_model
     method
7 trainer.save_model('./IndoBertGenderPrediction_V5_model')
8
```

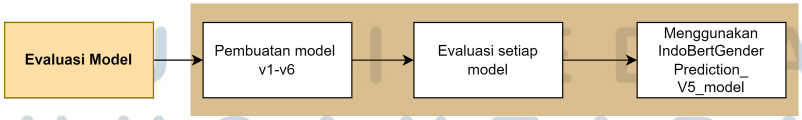
```
# Also save the tokenizer for consistency during inference
tokenizer.save_pretrained('./
    IndoBertGenderPrediction_V5_model')
```

Kode 3.25: Kode untuk menjalankan evaluasi untuk training

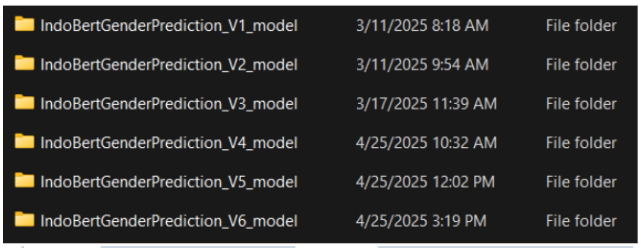
Seperti yang ditunjukkan pada Kode 3.25, fungsi `evaluate_and_print` dipanggil dengan argumen `test_dataset` untuk mengevaluasi kinerja model pada fase pengujian. Dataset ini sebelumnya telah disiapkan menggunakan kelas `GenderDataset`. Fungsi ini menghasilkan metrik evaluasi seperti akurasi, presisi, recall, F1-score, serta matriks kebingungan, yang seluruhnya merefleksikan kemampuan model dalam menggeneralisasi terhadap data baru yang akan diperlihatkan dan dijelaskan lebih lanjut di anak subbagian 3.3.6 Evaluasi model di subbagian 3.3. Setelah proses evaluasi selesai, model dan tokenizer disimpan ke dalam sebuah direktori, contoh `./IndoBertGenderPrediction_V5_model`, agar dapat digunakan kembali secara konsisten dalam proses inferensi berikutnya.

3.3.6 Evaluasi Model

Dengan pendekatan ini, proses pelatihan model dimulai dan setiap versi dievaluasi untuk menentukan model dengan performa terbaik. Secara keseluruhan, seperti yang ditunjukkan pada Gambar 3.13, enam model independen telah dilatih, tanpa satu pun yang merupakan kelanjutan atau hasil fine-tuning dari model sebelumnya. Strategi ini memastikan bahwa enam model yang dihasilkan benar-benar berbeda dan dapat dibandingkan secara objektif. Menurut IBM [44], pendekatan seperti ini efektif untuk mencegah *overfitting*, yaitu kondisi ketika model terlalu menyesuaikan diri dengan data pelatihan hingga gagal mengenali pola umum dan cenderung "menghafal" data. Pada akhirnya, model yang dipilih untuk digunakan adalah `IndoBertGenderPrediction_V5_model`, karena menunjukkan performa terbaik di antara seluruh versi yang diuji. Bagan kerja untuk evaluasi model bisa dilihat pada gambar 3.12.

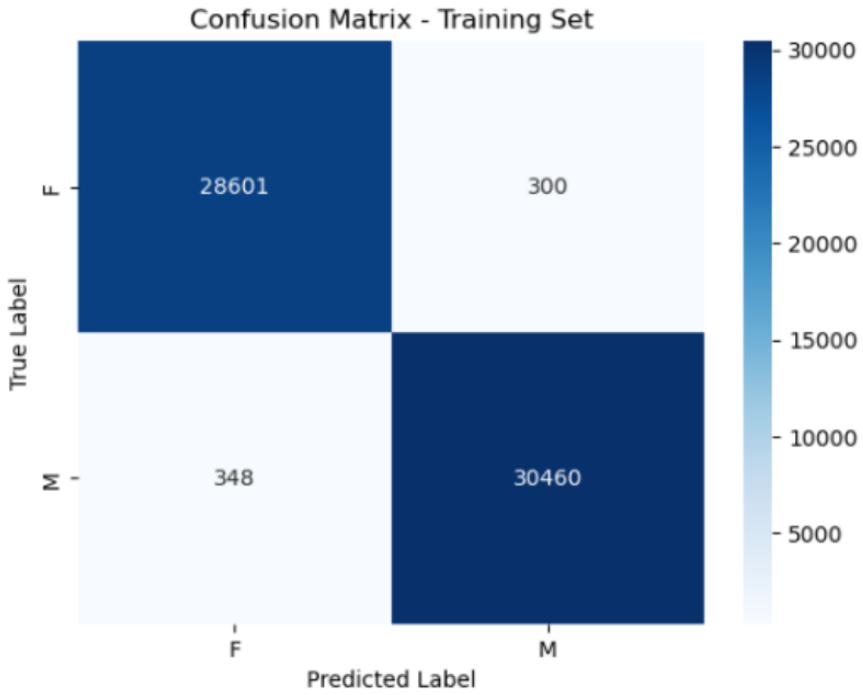


Gambar 3.12. Beberapa versi model yang telah dibuatkan



Gambar 3.13. Beberapa versi model yang telah dibuatkan

1. Evaluasi Training



Gambar 3.14. Hasil confusion matrix untuk Dataset pelatihan

Tabel 3.2. Classification Report buat dataset training

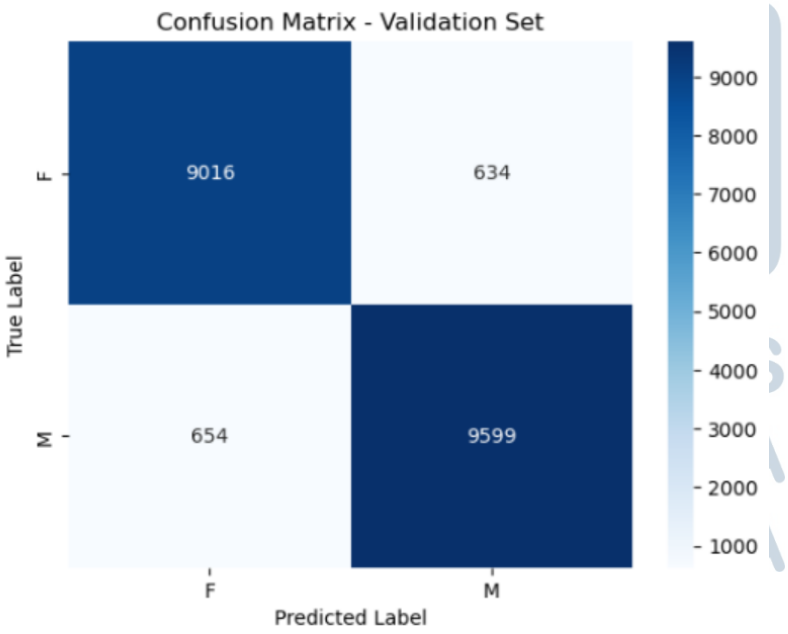
	Precision	Recall	F1-score	Support
F	0.99	0.99	0.99	28901
M	0.99	0.99	0.99	30808
Accuracy			0.99	59709
Macro avg	0.99	0.99	0.99	59709
Weighted avg	0.99	0.99	0.99	59709

Gambar 3.14 dan tabel 3.2 menampilkan hasil evaluasi model pada dataset pelatihan. Berdasarkan matriks kebingungan, model berhasil

mengklasifikasikan dengan benar sebanyak 28.601 dari 28.901 data perempuan, dengan hanya 300 data yang salah prediksi. Untuk data laki-laki, sebanyak 30.460 dari 30.808 berhasil diidentifikasi secara tepat, sementara 348 sisanya keliru diklasifikasikan sebagai perempuan. Secara keseluruhan, model menunjukkan performa yang sangat tinggi, dengan akurasi, presisi, recall, dan skor F1 yang masing-masing mencapai 99%. Hasil ini mengindikasikan bahwa model sangat andal dalam mengenali *gender* berdasarkan nama pada data pelatihan.

Namun demikian, akurasi yang sangat tinggi pada data pelatihan juga dapat menjadi indikasi adanya *overfitting*. Overfitting terjadi ketika model terlalu menyesuaikan diri dengan data pelatihan, hingga menghafal pola-pola spesifik alih-alih mempelajari generalisasi yang dapat diterapkan pada data baru. Dalam konteks dunia nyata, data cenderung mengandung noise dan variasi yang membuat akurasi tinggi pada data pelatihan menjadi kurang realistis. Oleh karena itu, meskipun performa pelatihan tampak sangat menjanjikan, kewaspadaan terhadap risiko overfitting tetap diperlukan untuk memastikan kemampuan generalisasi model terhadap data yang belum pernah dilihat sebelumnya.

2. Evaluasi Validasi



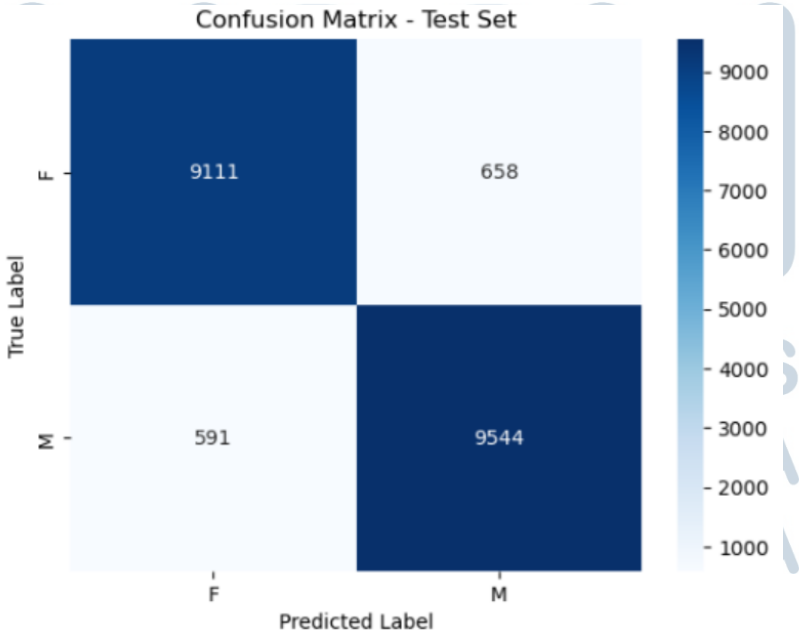
Gambar 3.15. Hasil confusion matrix untuk Dataset Validasi

Tabel 3.3. Classification Report buat dataset validasi

	Precision	Recall	F1-score	Support
F	0.93	0.93	0.93	9650
M	0.94	0.94	0.94	10253
Accuracy			0.94	19903
Macro avg	0.94	0.94	0.94	19903
Weighted avg	0.94	0.94	0.94	19903

Melihat Gambar 3.15 dan tabel 3.3, performa model pada data validasi menunjukkan sedikit penurunan dibandingkan dengan data pelatihan, namun hasilnya tetap berada pada tingkat yang sangat baik. Nilai metrik klasifikasi—seperti akurasi, presisi, recall, dan skor F1—berada dalam kisaran 93% hingga 94%, yang mencerminkan efektivitas model dalam melakukan generalisasi terhadap data yang belum pernah dilihat sebelumnya. Pada titik ini, tidak terdapat kebutuhan mendesak untuk melakukan penyesuaian terhadap hyperparameter. Satu-satunya alasan potensial untuk melakukan perubahan hyperparameter adalah demi mempercepat proses pelatihan, sehingga memungkinkan pengembangan lebih banyak versi model dalam waktu yang lebih efisien.

3. Evaluasi Testing



Gambar 3.16. Hasil confusion matrix untuk Dataset Pengujian

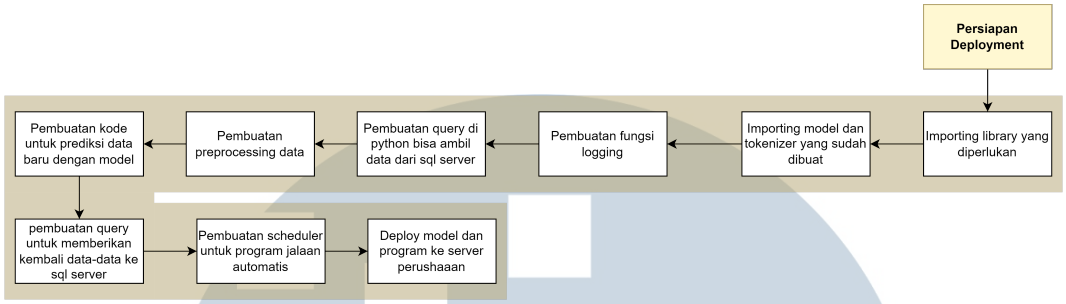
Tabel 3.4. Classification Report buat dataset pengujian

	Precision	Recall	F1-score	Support
F	0.94	0.93	0.94	9769
M	0.94	0.94	0.94	10135
Accuracy			0.94	19904
Macro avg	0.94	0.94	0.94	19904
Weighted avg	0.94	0.94	0.94	19904

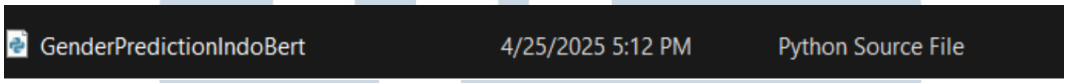
Pada Gambar 3.16 dan tabel 3.4, performa model menunjukkan sedikit peningkatan dibandingkan hasil validasi—terutama untuk kelas perempuan. Nilai presisi dan skor F1 untuk kelas perempuan keduanya mencapai 94%, sementara nilai recall tetap stabil di angka 93%, sama seperti pada data validasi sebelumnya. Secara keseluruhan, model menunjukkan performa yang kuat dan konsisten di seluruh kelas. Capaian akurasi sebesar 94% menandakan bahwa model telah terlatih dengan baik dan memiliki kemampuan generalisasi yang cukup kuat, sehingga kemungkinan besar telah siap untuk digunakan secara operasional dengan tingkat kepercayaan yang tinggi.

3.3.7 Penerapan server

Setelah memilih model dengan performa terbaik, sebuah program Python khusus dikembangkan untuk berfungsi sebagai penjadwal (*scheduler*), sebagaimana ditunjukkan pada Gambar 3.18. Meskipun proses prediksi dapat dijalankan secara manual, otomatisasi sangat penting untuk memastikan bahwa proses tersebut berjalan secara konsisten tanpa ketergantungan pada intervensi manusia. Hal ini menjadi krusial karena tugas terjadwal sering kali terlupakan atau terabaikan. Selain itu, otomatisasi membantu mengurangi potensi kesalahan manusia. Tujuan utama dari penjadwal ini adalah menjalankan model setiap hari pada pukul 23.00 WIB, yakni di luar jam kerja. Penjadwalan pada waktu ini dipilih agar proses tidak mengganggu operasional aktif, terutama jika waktu komputasi yang dibutuhkan cukup lama atau jika hasil prediksi perlu ditransfer kembali ke server SQL. Bagan kerja penerapan server bisa dilihat pada gambar 3.17.



Gambar 3.17. Bagan kerja untuk penerapan server



Gambar 3.18. File sumber Python yang berisi prediksi gender terjadwal.

Untuk mengotomatisasi seluruh alur kerja, proses pengambilan data serta pengiriman hasil prediksi harus dilakukan tanpa interaksi manual. Data akan diambil langsung dari server SQL, dan setelah proses inferensi selesai, hasil prediksi akan secara otomatis ditulis kembali ke server. Implementasi alur kerja ini memerlukan penambahan kode untuk mengelola integrasi data pada kedua sisi, baik saat pengambilan maupun saat penyimpanan hasil.

Di samping fungsi inti tersebut, beberapa penyempurnaan tambahan juga diperlukan agar sistem berjalan dengan stabil. Ini termasuk penanganan nilai yang hilang (*missing values*) pada data yang diambil guna mencegah kesalahan saat eksekusi, pemuatan model terlatih secara benar sebelum proses prediksi, serta penerapan sistem pencatatan log (*logging*). Logging sangat penting untuk memantau jalannya program, mendeteksi kesalahan secara dini, dan memberikan informasi yang dibutuhkan untuk merespons permasalahan dengan cepat. Berikut adalah kode yang telah dibuat berdasarkan kebutuhan tersebut:

1. **Importing Library**

```
import pyodbc
import time
import schedule
import pandas as pd
from datetime import datetime
import os
import torch
from transformers import BertForSequenceClassification,
    BertTokenizer
```

Kode 3.26: Library yang perlu di import untuk file Server

Beberapa pustaka Python digunakan sebagai dependensi utama dalam pengembangan program ini, seperti di kode 3.26. Pustaka seperti `pandas`, `torch`, `pyodbc`, dan `transformers` merupakan komponen yang sangat penting bagi fungsi inti program. Keempat pustaka ini tidak boleh dihapus atau digantikan, kecuali seluruh logika terkait yang bergantung padanya ditulis ulang secara menyeluruh. Misalnya, penghapusan `transformers` akan membuat model tidak dapat dimuat, sementara penghapusan `torch` akan menyebabkan proses inferensi gagal.

Sebaliknya, pustaka seperti `datetime`, `time`, dan `os` berperan penting dalam mendukung fungsi tambahan seperti penjadwalan tugas dan pencatatan log selama proses prediksi berlangsung. Pustaka-pustaka ini dapat digantikan oleh alternatif lain tergantung preferensi pengembang atau kebutuhan proyek tertentu. Misalnya, pustaka `pendulum` dapat digunakan sebagai pengganti `datetime` untuk manajemen waktu yang lebih fleksibel. Namun, apabila pustaka ini atau pustaka pengganti tidak disertakan, maka bagian program yang berkaitan dengan penjadwalan atau pencatatan kemungkinan besar tidak akan berjalan sebagaimana mestinya, yang dapat mengganggu alur kerja otomatisasi secara keseluruhan.

2. ***Importing* Model**

```
# Load your saved model and tokenizer
model_dir = './IndoBertGenderPrediction_V5_model'

model = BertForSequenceClassification.from_pretrained(
    model_dir)
tokenizer = BertTokenizer.from_pretrained(model_dir)
```

Kode 3.27: Import model dan tokenizer yang sudah dibuat

Pada saat proses penyimpanan model, sebuah folder akan dibuat secara otomatis yang berisi *tokenizer* dan bobot dari model yang telah dilatih. Nama folder ini dapat ditentukan atau disesuaikan oleh pengguna sesuai kebutuhan. Dengan mekanisme ini, model tidak perlu diunduh kembali dari situs web atau repositori daring seperti Hugging Face, karena seluruh komponen penting sudah tersedia secara lokal. Hal ini tidak hanya menghemat waktu, tetapi juga mendukung efisiensi kerja saat model digunakan kembali dalam proses inferensi atau integrasi sistem di masa mendatang, seperti yang dilakukan di kode 3.27.

3. **Ambil data di SQL server**

```sql
SELECT DISTINCT
  dc.MemberId,
  dc.CustomerName
FROM DimCustomer dc
LEFT JOIN GenderPredictionMember gp
  ON dc.MemberId = gp.MemberId
WHERE
  gp.MemberId IS NULL                              -- never
   predicted
  AND (dc.Gender IS NULL OR dc.Gender NOT IN ('M','F'))
  AND dc.MemberId IS NOT NULL
  AND LTRIM(RTRIM(dc.MemberId)) <> ''              -- not
   blank
  AND dc.CustomerName IS NOT NULL
  AND LTRIM(RTRIM(dc.CustomerName)) <> ''          -- not
   blank
  ORDER BY dc.MemberId;
```

Kode 3.28: Di program python menggunakan library pyodbc sql diatas dipakai untuk ambil data langsung dari sql server

Kode 3.28 dapat disimpulkan sebagai proses pengambilan *MemberId* dan *CustomerName* dari setiap entri dalam tabel `DimCustomer`, dengan ketentuan bahwa kedua field tersebut tidak boleh mengandung nilai `null`. Selain itu, data yang sudah pernah diproses—ditandai dengan keberadaan *MemberId* yang sama di dalam tabel `GenderPredictionMember`—akan dikecualikan dari pengambilan. Dengan demikian, hanya data anggota yang belum diprediksi *gender* yang akan diproses lebih lanjut.

Meskipun *MemberId* tidak secara langsung digunakan dalam proses klasifikasi *gender*, field ini tetap dilibatkan dalam proses pengembalian hasil prediksi ke server SQL. Hal ini dilakukan karena *MemberId* bertindak sebagai pengenal unik (*primary key*) yang penting untuk memastikan integritas data. Pengenal ini memungkinkan sistem untuk membedakan entri dengan nama yang sama atau mirip, serta menjaga keteraturan dan ketepatan pencatatan hasil prediksi dalam tabel `GenderPredictionMember`.

4. **pemrosesan**

```python
log_message("Cleaning data...")

GenderData = GenderData.dropna(subset=['CustomerName'])

log_message("Data successfully cleaned!")
```

Kode 3.29: Preprocessing data yang didapatkan dari SQL server

Sebagai tindakan pencegahan tambahan, mengikuti kode 3.29, dilakukan penghapusan nilai `NULL` pada kolom `CustomerName`. Langkah ini sangat

penting, terutama jika program dijalankan menggunakan dataset yang diimpor secara lokal, bukan langsung dari server SQL. Penyaringan ini dimaksudkan untuk mengatasi potensi hilangnya data nama pelanggan yang dapat terjadi akibat kesalahan dalam proses impor atau penyimpanan data. Karena `CustomerName` merupakan fitur utama dalam proses klasifikasi *gender*, keberadaan nilai ini sangat krusial bagi akurasi prediksi. Oleh karena itu, kode yang memfilter nilai `NULL` disertakan sebagai bentuk kehati-hatian demi memastikan keandalan hasil yang diperoleh dari model.

5. **Prediksi**

```
def predict_gender(text):
    text = str(text)  # Convert to string
    inputs = tokenizer(text, return_tensors="pt",
   truncation=True, padding=True)
    inputs = {k: v.to(device) for k, v in inputs.items()
   }
    with torch.no_grad():
        outputs = model(**inputs)
    predicted_label = torch.argmax(outputs.logits, dim
   =1).item()
    return "M" if predicted_label == 1 else "F"
```

Kode 3.30: Pembuatan fungsi prediksi di dalam program prediksi kelamin

Pada bagian kode 3.30, `tokenizer` yang telah diimpor digunakan untuk melakukan tokenisasi terhadap nama-nama baru yang akan diprediksi *gender*. Proses prediksi dilakukan dalam konteks `torch.no_grad()`, yang memastikan bahwa tidak ada pembaruan bobot selama inferensi berlangsung. Pendekatan ini memungkinkan model menghasilkan output secara efisien tanpa mengubah parameter internal.

Label hasil prediksi diperoleh dengan menerapkan fungsi `argmax` pada output `logits`, sesuai dengan rumus berikut:

$$\arg\max_{i\in\{0,1\}} [L_0, L_1] = \begin{cases} 1, & \text{jika } L_1 > L_0 \\ 0, & \text{jika } L_0 \geq L_1 \end{cases} \tag{3.7}$$

Di rumus 3.7, L_1 merupakan logit untuk kelas positif (Laki-laki), sedangkan L_0 merupakan logit untuk kelas negatif (Perempuan). Jika kedua nilai logit sama besar, maka `argmax` akan mengembalikan indeks pertama (yaitu L_0), sesuai dengan dokumentasi PyTorch, sehingga hasilnya adalah 0. Jika posisi L_1 dan L_0 ditukar, maka L_1 akan menjadi penentu hasil ketika nilai keduanya identik [45].

Dalam implementasi kode, `GenderData` adalah sebuah `DataFrame` yang digunakan untuk menyimpan hasil prediksi. Sebuah kolom baru dengan nama `Predicted_Gender` akan ditambahkan ke dalam `DataFrame` tersebut, di mana fungsi `predict_gender` diterapkan ke kolom `CustomerName` menggunakan metode `apply()`. Output dari fungsi ini kemudian disimpan dalam kolom `Predicted_Gender`.

6. **pengiriman data balik ke SQL server**

```
1
2 for row in data:
3     # Execute stored procedure for each row
4     cursor.execute("""
5         EXEC sp_GenderPredictionMember
6             @MemberId = ?,
7             @CustomerName = ?,
8             @Predicted_Gender = ?
9     """,
10    row['MemberId'],          # MyValueId
11    row['CustomerName'],      # Nama Customer
12    row['Predicted_Gender']   # Hasil Prediksi Gender
13    )
```

Kode 3.31: pengiriman data kembali ke SQL server dengan mengirim ke store procedure

Kode 3.31 menunjukkan cara mengirim hasil prediksi dari model ke server SQL. Dengan menggunakan pustaka `pyodbc`, kode tersebut memanggil prosedur tersimpan bernama `sp_GenderPredictionMember`, dan meneruskan daftar yang berisi `MemberId`, `CustomerName`, serta `Predicted_Gender`. Semua proses pemformatan dan penanganan data dilakukan di dalam prosedur tersimpan tersebut. Oleh karena itu, tugas utama dari bagian kode ini adalah memastikan bahwa data dikirim dengan aman dan benar ke prosedur, bukan langsung dimasukkan ke dalam tabel `GenderPredictionMember`.

7. ***Scheduler***

```
1 # Schedule the job to run at a specific time (e.g.,
     22:00 or 10:00 PM)
2 schedule.every().day.at("22:00").do(run_prediction)
3
4 print("The program Gender prediction has started. The
     process will start at 22:00!")
5
6 while True:
7     # Run any pending scheduled tasks
8     schedule.run_pending()
9     time.sleep(1)  # Small sleep to prevent a busy loop
```

Kode 3.32: kode yang buat program tersebut berjalan otomatis pada ham 22.00 WIB

Mengikuti kode 3.32, dengan menggunakan pustaka `schedule`, tugas dapat dijadwalkan dengan memanggil `schedule.every().day.at("22:00").do(run_prediction)`.

Pernyataan ini berarti bahwa fungsi `run_prediction` akan dijalankan secara otomatis setiap hari pada pukul 22.00 WIB. Untuk mengaktifkan penjadwalan ini, digunakan perulangan `while True` bersama dengan `schedule.run_pending()`. Struktur perulangan ini akan terus memeriksa apakah sudah waktunya menjalankan tugas, dan akan mengeksekusi fungsi yang dijadwalkan ketika waktunya tiba. Selain itu, `time.sleep(1)` digunakan untuk menjeda perulangan selama satu detik setiap kali pemeriksaan dilakukan. Penjeda ini bertujuan untuk mencegah beban berlebih pada sistem dan menghindari konflik dengan proses lain yang berjalan bersamaan.

8. ***Logging function***

```
def log_message(message):
    """
    Logs a message to both MasterLogGenderPrediction.log and MasterLogGenderPredictionBackup.log.
    """
    log_dir = "logs_IndoBert"
    os.makedirs(log_dir, exist_ok=True)  #     Ensure "logs" directory exists

    log_filename = os.path.join(log_dir, "MasterLogDataGenderPredictionIndoBert.log")
    backup_filename = os.path.join(log_dir, "MasterLogGenderPredictionBackupIndoBert.log")

    now = datetime.now()
    timestamp = now.strftime("%Y-%m-%d %H:%M:%S")  # Full timestamp (YYYY-MM-DD HH:MM:SS)
    log_entry = f"{timestamp} - {message}\n"

    try:
        #     Write to both log files
        with open(log_filename, "a") as main_log, open(backup_filename, "a") as backup_log:
            main_log.write(log_entry)
            backup_log.write(log_entry)
    except Exception as e:
        print(f"ERROR: Failed to write logs - {e}")

    print(log_entry, end="")  #     Also print to console for real-time feedback
```

Kode 3.33: Pembuatan sistem pencatat untuk menyimpan status program prediksi kelamin

Mengikuti kode 3.33, dengan menggunakan pustaka `os`, fungsi ini membuat file log khusus di dalam direktori `logs_IndoBert`. Jika direktori tersebut

belum tersedia di komputer lokal, maka program akan secara otomatis membuatnya. Demikian pula, apabila file log belum ada, file tersebut akan dibuat sesuai kebutuhan. Fungsi ini menulis entri log dalam format *timestamp - message*, contohnya: `2025-08-05 22:00:00 - Pesan yang ditampilkan dalam log`. Setiap entri log tidak hanya ditulis ke dalam file log, tetapi juga ditampilkan ke konsol, baik saat dijalankan melalui *command prompt* maupun di lingkungan seperti Jupyter Lab/Notebook. Output ganda ini memberikan umpan balik langsung kepada pengguna sekaligus menyimpan catatan permanen dari aktivitas program.

Penyimpanan log ke dalam file berfungsi sebagai langkah pengamanan penting, terutama jika terjadi kegagalan sistem atau program ditutup secara tiba-tiba. Jika informasi log hanya ditampilkan di konsol, maka catatan tersebut akan hilang. Sebaliknya, pencatatan ke dalam file memungkinkan dokumentasi status dan progres program secara berkelanjutan. Perlu dicatat bahwa blok `try-except` dalam fungsi ini hanya menangani kesalahan yang berkaitan dengan proses pencatatan log—misalnya kegagalan saat menulis ke file atau error dari sistem operasi. Kesalahan lain, seperti yang berasal dari model *machine learning*, harus ditangani secara terpisah di bagian program yang relevan.

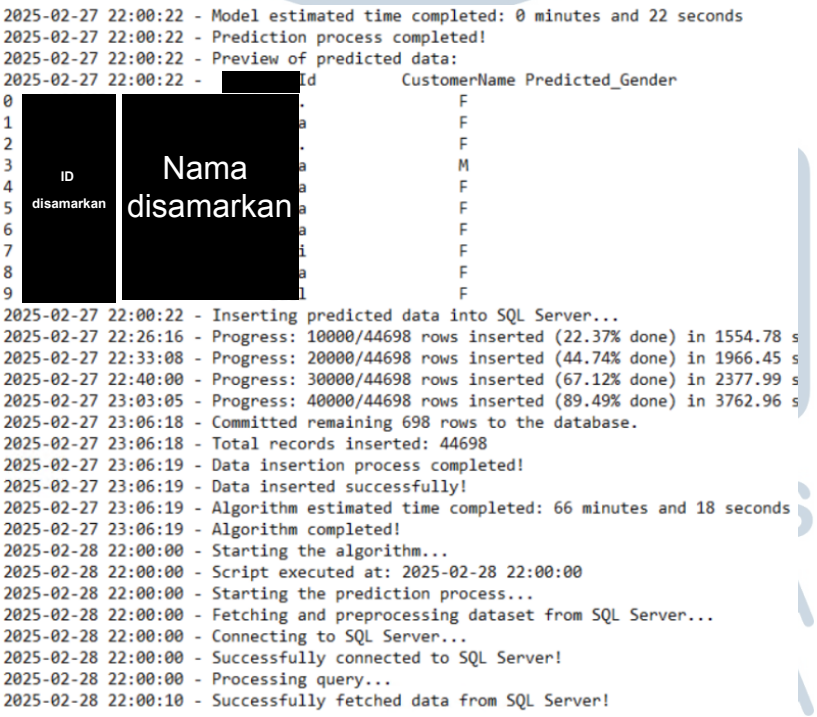
Meskipun kode implementasi lengkap tidak disertakan, komponen dan struktur utamanya telah dijelaskan dan dapat disesuaikan berdasarkan preferensi masing-masing pengguna. Elemen seperti nama fungsi, nama variabel, dan *identifier* pada `DataFrame` bersifat fleksibel dan tidak harus identik seperti yang ditampilkan dalam penjelasan. Namun, apabila pengguna memutuskan untuk mengganti pustaka inti (*core library*), maka sangat penting untuk menyesuaikan seluruh bagian kode agar selaras dengan sintaks dan perilaku pustaka baru tersebut. Selama logika inti tetap dipertahankan, program akhir seharusnya mampu menghasilkan keluaran yang serupa dan tetap andal.

Setelah seluruh kode dijelaskan, langkah terakhir yang perlu dilakukan adalah mengeksekusi program melalui `Anaconda Prompt` atau `Command Prompt`, dengan asumsi bahwa lingkungan `Python` dan pustaka-pustaka terkait telah diinstal dan dikenali oleh sistem. Perintah yang digunakan adalah sebagai berikut (nama file dapat disesuaikan bila berbeda):

```
python GenderPredictionIndoBert.py
```

Apabila semua konfigurasi telah dilakukan dengan benar, program akan berjalan tanpa menghasilkan kesalahan dan akan memberikan output sesuai yang diharapkan. Hal ini diperlihatkan pada gambar 3.19 dan gambar 3.20, di mana gambar 3.19 menampilkan hasil eksekusi program di `Anaconda Prompt`, sedangkan Gambar 3.20 memperlihatkan keluaran yang tercatat dalam file log.

Gambar 3.19. Keluaran di prompt saat memulai dari command prompt/anaconda prompt



Gambar 3.20. Keluaran di file log saat program dimulai

3.3.8 Pelaksanaan Magang

Pelaksanaan kerja magang diuraikan pada Tabel 3.5. Perlu diingat bahwa pekerjaan ini dijalankan secara berselang-seling dengan proyek lain yang juga harus diselesaikan. Selain itu, proses pembuatan laporan memerlukan waktu tambahan karena adanya komitmen untuk menyusun laporan yang sedetail mungkin. Artinya, setiap perhitungan, rumus, dan strategi dijelaskan secara rinci dalam laporan, bukan hanya disampaikan sebagai informasi sekilas. Dalam presentasi kepada supervisor dan mentor, model yang telah dibuat memang dijelaskan, tetapi disampaikan secara umum. Beberapa konsep utama seperti Transformer, Masked Language Modeling (MLM), dan Next Sentence Prediction (NSP) dijelaskan secara ringkas untuk memberikan pemahaman umum, tanpa membahas detail teknis seperti perhitungan matematis atau algoritma secara mendalam. Penjelasan ini disesuaikan agar mudah dipahami oleh audiens, cukup untuk memberikan konteks yang relevan. Selain itu, laporan kerja juga diterjemahkan ke dalam dua versi bahasa, yaitu Bahasa Indonesia dan Bahasa Inggris. Proses penerjemahan ini dibantu oleh model berbasis MLM seperti ChatGPT, dengan tetap dilakukan peninjauan manual untuk memastikan hasil terjemahan tetap masuk akal dan sesuai konteks. Penjelasan lebih lanjut mengenai pembaruan model IndoBERT yang disebabkan oleh kekurangan data dapat ditemukan pada bagian 3.4, khususnya pada daftar 1.

Tabel 3.5. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke -	**Pekerjaan yang dilakukan**
Februari	
1	Briefing Pekerjaan apa yang harus dilakukan, salah satunya pembuatan model klasifikasi *gender*. Mengumpulkan data untuk klasifikasi *gender* dan membangun model menggunakan Random Forest dan ANN, termasuk preprocessing dan fine-tuning untuk mengevaluasi performa terbaik.
2	Melanjutkan eksperimen dengan Random Forest, melakukan fine-tuning tambahan, dan menyiapkan kode untuk deployment ke server.
4	Presentasi hasil model Random Forest kepada mentor dan supervisor. Disarankan untuk mencoba IndoBERT karena hasil Random Forest kurang memadai.
Maret	

Minggu Ke -	Pekerjaan yang dilakukan
1	Mempelajari arsitektur dasar Transformer dan prinsip kerja IndoBERT secara umum, serta membangun kerangka awal model IndoBERT.
2	IndoBERT siap untuk dideploy ke server karena menunjukkan hasil yang lebih baik. Dilanjutkan dengan penulisan laporan kerja bagian Transformer.
3	Melanjutkan penulisan laporan bagian Transformer dengan menyertakan detail teknis.
4	Menyelesaikan bagian laporan tentang Transformer dan mempresentasikan hasil model IndoBERT kepada supervisor dan mentor.
April	
1–2	Menulis bagian laporan kerja yang menjelaskan BERT dan cara kerja IndoBERT.
3	Melanjutkan laporan dan memperbarui model IndoBERT karena ditemukan kekurangan data.
4	Presentasi pembaruan terhadap model IndoBERT.
Mei	
1	Menulis bagian laporan tentang implementasi model hingga proses deployment ke server.
2	Menyelesaikan bagian implementasi dan membuat dua versi laporan: Bahasa Indonesia dan Bahasa Inggris.
3	Menambahkan informasi yang kurang dan memeriksa kelengkapan isi laporan serta hasil terjemahannya.
4	Presentasi laporan akhir kepada supervisor dan mentor, menyatakan laporan siap digunakan sebagai referensi proyek selanjutnya.

3.4 Kendala dan Solusi yang Ditemukan

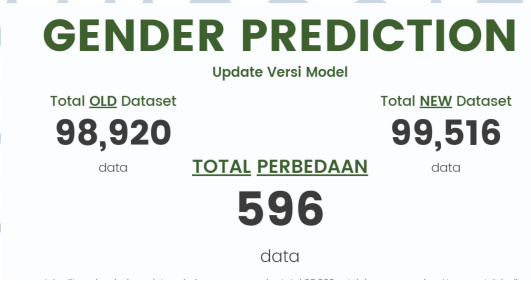
Bagian ini akan membahas kendala-kendala yang belum dijelaskan pada subbab sebelumnya dan bersifat tidak terduga selama pelaksanaan proyek. Sebelumnya, sebagian besar kendala minor seperti pencarian data tambahan karena keterbatasan data awal, pembuatan *scheduler* sebagai solusi dari masalah akses

server melalui *remote*, dan peralihan dari model *random forest* ke IndoBERT karena isu akurasi, telah dijelaskan di bagian lain. Namun, masih terdapat sejumlah kendala tambahan yang bersifat insidental dan memerlukan penanganan cepat karena tidak direncanakan sebelumnya. Kendala-kendala ini muncul tiba-tiba dan memberikan dampak langsung terhadap alur kerja proyek, sehingga perlu dicatat secara terpisah.

Beberapa kendala yang dimaksud antara lain adalah hilangnya sebagian data pada tahap *preprocessing*, yang mengharuskan tim melakukan audit ulang terhadap alur pengolahan data mentah. Selain itu, proses *training* memerlukan waktu jauh lebih lama dari estimasi awal, yang berdampak pada keterlambatan jadwal penyelesaian model. Permasalahan lainnya adalah *scheduler* yang tiba-tiba mengalami *error*, meskipun sebelumnya telah diuji dan berjalan normal. Ketiga kendala ini memerlukan solusi cepat dan adaptif karena terjadi di luar skenario pengujian awal dan mengganggu kestabilan sistem. Berikut adalah penjelasan lebih detail dengan setiap kendala:

1. **Terjadi kehilangan data pada tahap *preprocessing*.**

   Pada awalnya, versi IndoBERT 1 hingga 3 dikembangkan menggunakan dataset bernama `KGGenderDatasetClean.csv`. Dataset ini merupakan hasil dari proses *preprocessing* yang telah dilakukan pada proyek *random forest* sebelumnya dan digunakan sebagai data utama untuk pelatihan, validasi, dan pengujian. Namun, saat proses pembersihan kode untuk keperluan dokumentasi IndoBERT dilakukan, ditemukan adanya perbedaan sebanyak 596 entri antara dataset lama dan dataset yang dihasilkan oleh kode baru, sebagaimana ditunjukkan pada Gambar 3.21. Perbedaan ini muncul setelah eksekusi kode pembersihan terbaru seperti yang ditunjukkan pada Kode 3.2.



Gambar 3.21. Perbedaan data dari dataset lama dengan dataset baru adalah sebanyak 596 entri

Penyebab perbedaan jumlah data ini belum berhasil diidentifikasi secara pasti karena struktur kode *preprocessing* pada proyek *random forest* dan kode baru pada Kode 3.2 tidak berbeda secara signifikan. Selain itu, investigasi lebih dalam tidak menjadi prioritas karena fokus utama adalah menentukan versi kode mana yang memberikan hasil lebih valid. Setelah dilakukan pengecekan secara menyeluruh, kode dalam Kode 3.2 terbukti menghasilkan dataset yang lebih konsisten. Sebaliknya, kode lama menunjukkan perilaku yang tidak stabil karena hasil *preprocessing*-nya bervariasi setiap kali dijalankan. Mengingat program prediksi *gender* harus segera diselesaikan dan digunakan, maka prioritas diberikan pada penggunaan kode yang dapat memberikan hasil yang konsisten dan dapat direproduksi. Jika dilihat, model versi 5-6 menggunakan dataset baru untuk mendapatkan hasil yang diiinginkan.

2. **Proses *training* berlangsung sangat lama.**

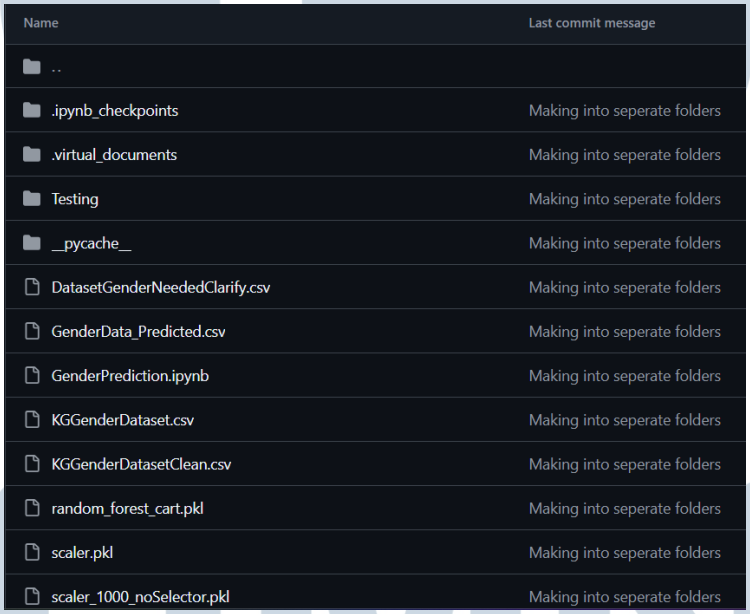
   Salah satu permasalahan utama dalam proses pelatihan model *deep learning* adalah lamanya waktu yang dibutuhkan untuk menyelesaikan fase *training*. Sebagai contoh, pelatihan model IndoBERT menggunakan konfigurasi standar dapat memakan waktu hingga 28–30 jam untuk setiap siklus pelatihan. Oleh karena itu, efisiensi menjadi faktor yang sangat krusial dalam proyek ini. Solusi utama untuk mengatasi permasalahan tersebut adalah dengan memanfaatkan GPU, khususnya GPU dari NVIDIA, sebagai perangkat untuk menjalankan proses pelatihan.

   Penggunaan GPU jauh lebih efektif dibandingkan CPU karena arsitektur GPU memiliki ribuan inti yang mendukung pemrosesan paralel, optimal untuk operasi matriks, serta mampu menangani *dataset* berukuran besar berkat *bandwidth* memori yang tinggi dan efisiensi pemrosesan paralel [46]. Selain itu, GPU dari NVIDIA secara khusus dirancang untuk mendukung proyek *deep learning*, dengan fitur seperti *tensor cores* yang mempercepat proses pelatihan untuk *dataset* berskala besar [47]. Dengan memanfaatkan GPU NVIDIA, waktu pelatihan dapat dikurangi secara signifikan, dari sebelumnya 28–30 jam menjadi hanya sekitar 1–2 jam saja per siklus.

3. **Folder data yang korupsi atau hilang**

   Pada tanggal 20 Mei 2025, SSD yang menyimpan proyek prediksi *gender*

mengalami kerusakan mendadak sehingga seluruh data di dalamnya hilang. Meskipun berbagai upaya telah dilakukan untuk memulihkan data tersebut, hasil akhirnya menunjukkan bahwa data-data itu tidak dapat dikembalikan. Untungnya, masih tersedia beberapa sumber cadangan yang dapat membantu memulihkan sebagian isi folder proyek, seperti repositori GitHub yang menyimpan hasil pengerjaan terdahulu—termasuk dataset dan kode model *random forest* seperti terlihat pada Gambar 3.22. Selain itu, tersedia juga laporan pengerjaan proyek yang memuat dokumentasi dan penjelasan mengenai kode yang sebelumnya telah dibuat, sebagaimana ditunjukkan pada Gambar 3.23.



Gambar 3.22. Pada folder klasifikasi gender terdapat proyek yang telah dikerjakan serta dataset.