BAB IV
ANALISIS DAN HASIL

4.1 Tahap Persiapan Dataset

Pada penelitian ini menggunakan dataset yang sama untuk dua buah metode. Persiapan dataset dilakukan untuk memastikan setiap atribut data sudah sesuai dan tidak terdapat data yang nul atau kosong. Proses *cleaned* data juga dilakukan untuk memastikan dataset sudah benar-benar siap digunakan. Selain proses pembersihan data, proses transformasi data juga dilakukan pada penelitian ini yaitu menggubah format dari excel (.xls) menjadi *CommaSeparated Values* (.csv). Berikut ini merupakan dataset yang digunakan pada penelitian dapat dilihat pada Gambar 4.1 berikut.

	id	name	email	gender	username	phone	city	avatar	university	skill_1	skill_2	skill_3	skill_4	birth	mbti	age	password	job_interest
2	AAJ603	Chris Wes	cwestphal	Male	cwestphal	+62-349-5	Panungga	https://ro	Université	PHP	Visual Des	Google Cl	Flutter De	04/08/195	ESTJ	27	XvKKjs	.NET Developer
3	HNA156	Koral Nilg	knilges1@	Female	knilges1	+86-144-2	Baixi	https://ro	Thierry Gr	Node.js	Informatic	Salesforce	React Nati	12/01/20(	ENFJ	22	RQcbkY	Advance Data Analyst
4	FND122	Biron Mav	bmavin2@	Male	bmavin2	+358-639-	Hankasal	https://ro	South Dak	C#	Wireframi	Google Cl	Flutter De	17/04/195	ESFJ	33	lviTKkjrhR	AI Researcher
5	FMD294	Rurik Iwar	riwanowic	Male	riwanowic	+62-586-8	Pangkalan	https://ro	Ruprecht-	SQL	Usability T	DigitalOce	Android D	13/10/20(	ENTP	22	vyOIKKEKh	android architect
6	XVU891	Bobbee Fc	bfoord4@	Female	bfoord4	+7-251-50	Omsk	https://ro	Mongoliar	Node.js	Wireframi	AWS	Flutter De	28/11/20(	ESFP	18	YAWvBxRq	Android Developer
7	ZHP524	Lawrence	lheadan5(	Male	lheadan5	+86-266-6	Xarag	https://ro	Al-Buraim	SQL	Informatic	Google Cl	Flutter De	28/02/20(	ESFJ	20	GEIHI8LS	Android Engineer
8	FLG242	Sibley Ber	sberns6@	Female	sberns6	+63-341-7	Mambura	https://ro	Ecole Nati	Python	Front-end	IBM Cloud	Flutter De	06/08/195	ESFJ	32	dLNo3d	Android Technical deve
9	CAH089	Barbe Cat	bcato7@n	Female	bcato7	+62-592-9	Kalapagad	https://ro	Universida	Kotlin	Wireframi	Oracle Clo	Android D	12/02/195	ENFJ	27	4NGhGC	API Developer Specialist
10	YFX519	Demeter C	doutright	Female	doutright	+420-726-	Libě́šice	https://ro	Shonan In	Node.js	Front-end	Google Cl	React Nati	12/02/20(	ISFJ	22	UzaAba1Z	Application Integration
11	CMX837	Harley Ho	hhowsder	Male	hhowsder	+62-533-2	Pondokka	https://ro	Universida	Scala	Front-end	AWS	Ionic Deve	14/10/195	ESTJ	24	ShmNPI2	Application Security Spe
12	PQB446	Tadio Mac	tmadgewi	Male	tmadgewi	+86-278-2	Jiale	https://ro	Logistics E	C#	User Rese	Red Hat Cl	Flutter De	23/11/195	ENFJ	31	jlHP8nXD	Artificial Intelligence En
13	QSR330	Jannel Alls	jallsobroo	Female	jallsobroo	+33-754-3	Gif-sur-Yv	https://ro	Internatio	Java	Front-end	AWS	Android D	30/01/195	ENTJ	27	91x3YmlvI	Back end developer
14	AAC997	Etti Callar	ecallarc@l	Female	ecallarc	+1-309-67	Thetford-I	https://ro	Augustana	Scala	Interactio	IBM Cloud	Ionic Deve	17/08/195	ENTJ	24	HuM1ITRa	Big Data Specialist
15	KYV605	Keith McG	kmcgeane	Male	kmcgeane	+86-434-4	Erniusuok	https://ro	Mutesa 1	Java	Front-end	VMware C	Flutter De	20/09/195	ISFP	26	sMcWFvw	Blockchain Developer

Gambar 4.1 Dataset yang digunakan

Pada Gambar 4.1 memuat dataset yang berisi atribut data seperti ID *user, Name, Email, Gender, Username, Phone, City, Avatar, University, Skill1, Skill 2, Skill 3, Skill 4, Birth, Mbti, Age, Password, Job interest*. Dataset di atas dibuat menggunakan *website* Mockaroo. Mockaroo ialah alat daring gratis yang digunakan untuk menghasilkan kumpulan data CSV, JSON, SQL, dan Excel secara kustom untuk melakukan pengujian.

Pada metode USE proses *input* data akan dilanjukan dengan proses *preprocessing* data. Berbeda dengan metode TFRS. Pada metode TFRS setelah

melakukan *preprocessing* data dilanjutkan dengan *spliting* data. *Dataset* dibagi menjadi dua bagian utama yaitu data pelatihan (*training set*) sebesar 80% dan data pengujian (*test set*) sebesar 20%. Berikut ini menunjukkan dataset *job interest* yang digunakan pada penelitian dapat dilihat pada Gambar 4.2 sebagai berikut.

1	id_job_interest
2	xc1,.NET Developer
3	xc2,Advance Data Analyst
4	xc3,AI Researcher
5	xc4,android architect
6	xc5,Android Developer
7	xc6,Android Engineer
8	xc7,Android Technical developer
9	xc8,API Developer Specialist
10	xc9,Application Integration Specialist
11	xc10,Application Security Specialist
12	xc11,Artificial Intelligence Engineer
13	xc12,Back end developer
14	xc13,Big Data Specialist
15	xc14,Blockchain Developer

Gambar 4.2 Dataset *Job Interest*

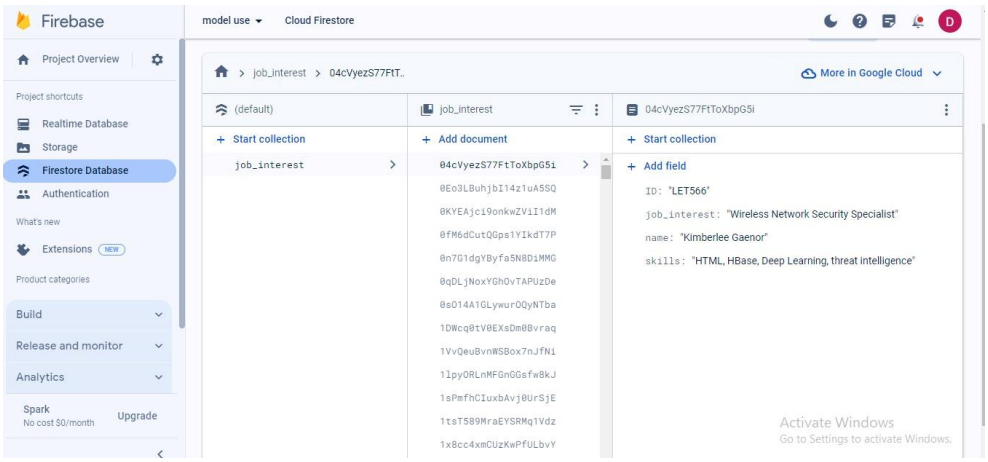
Pada Gambar 4.2 penelitian ini menggunakan dua buah parameter yaitu *skill* dan *job interest*. Dataset *job interest* yang digunakan pada penelitian ini dibuat menggunakan *website* yang sama seperti pembuatan dataset yaitu, Mockaroo. Namun, dataset *job interest* yang digunakan hanya berisi dua buah atribut yaitu ID dan *Job Interest*. Sehingga, ID yang terdapat pada dataset dan data *test* menjadi *primary key* pada penelitian ini.

Pada penelitian ini selain menggunakan *website* Mockaroo untuk meng-*generate* dataset penggunaan Microsoft Excel juga digunakan dalam melakukan persiapaan dataset. Proses tersebut merupakan proses perubahan format yang sesuai kebutuhan untuk memudahkan proses *modelling* sistem rekomendasi baik dalam metode USE dan metode TFRS. Microsoft Excel dipilih dalam melakukan persiapan dataset karena memudahkan pengembang dalam mengedit dan memanipulasi data dengan lebih terstruktur.

4.1.1 Analisis Penggunaan *Firebase*

Firebase digunakan sebagai *Cloud* dalam sistem rekomendasi menggunakan metode USE. *Firebase* dipilih karena kapasitas penyimpanan yang besar dan memudahkan dalam melakukan sinkronisasi database serta fitur *Cloud firestore* yang sangat membantu dalam pembuatan aplikasi Collabolio. Fitur tersebut

mendukung adanya sinkronisasi *offline* yang penting dilakukan dalam memastikan ketersediaan rekomendasi tanpa koneksi internet sekalipun. Tampilan *firebase* dapat dilihat pada Gambar 4.3 berikut.

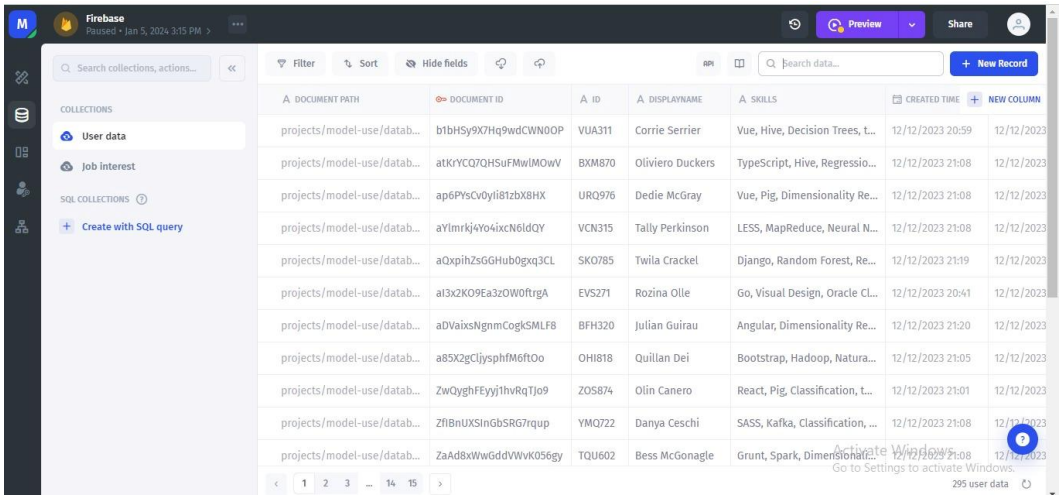


Gambar 4.3 Tampilan *Firebase*

Pada Gambar 4.3 merupakan tampilan *firebase* yang sudah berisikan dataset *user* atau pengguna yang digunakan pada metode USE. Memasukkan data ke *Firebase* merupakan langkah penting dalam membangun aplikasi. Memasukkan data yang digunakan guna menyimpan informasi pengguna, atribut-atribut yang digunakan, dan data pendukung yang diperlukan dalam interaksi antar pengguna.

Proses *input* atau memasukkan dataset bisa dilakukan dengan berbagai cara. Salah satu cara ialah penginputan secara manual dan menggunakan *tools* eksternal seperti *webiste* JetAdmin. *Website* JetAdmin merupakan platform pengembangan yang membantu proses pembuatan panel administrasi yang responsif. JetAdmin menawarkan beberapa fitur utama yaitu menyediakan *template* yang siap digunakan membuat para pengembang memungkinkan membangun panel admin dengan cepat. Selain itu, JetAdmin juga terintegrasi dengan berbagai *database* seperti, PostgreSQL, MySQL, MongoDB, dan lain-lain. JetAdmin menyediakan sistem autentifikasi dan otorisasi. Visualisasi data juga dapat dilakukan menggunakan JetAdmin mulai dari pembuatan grafik, diagram, dan laporan statik. Fitur menarik dari JetAdmin yaitu dapat membuat *endpoint* API secara kustom dengan mudah.

Dengan tampilan panel admin yang responsif memungkinkan JetAdmin dapat diakses melalui perangkat *mobile*. Tampilan JetAdmin dapat dilihat pada Gambar 4.4 berikut.



Gambar 4.4 Tampilan JetAdmin

Berdasarkan Gambar 4.4 tampilan JetAdmin yang digunakan untuk membantu dalam proses *input* data. Pada penelitian ini menggunakan *Firebase* SDK dalam membangun aplikasi Collabolio. Cara menggunakan *Firebase* SDK ialah menyiapkan proyek *firebase* dan mendapatkan file konfigurasi SDK. Lalu, menambahkan SDK *firebase* ke proyek yang digunakan dan melakukan inisialisasi *firebase* SDK dengan menggunakan konfigurasi proyek. Dengan kata lain yaitu menghubungkan *Firebase* menggunakan *Firebase* Admin SDK dan menginisialisasi koneksi ke *Firestore*.

```
import firebase_admin
from firebase_admin import credentials, firestore
import pandas as pd
import numpy as np
# Connect to firebase
cred = credentials.Certificate("/content/model-use-firebase-adminsdk-ojn3p-97a8353f7a.json")
firebase_admin.initialize_app(cred)
```

```
db = firestore.client()
```

Listing code di atas digunakan untuk mengkoneksikan Firebase. Pada baris `credentials.Certificate` merupakan kredensial untuk mengauntentifikasi aplikasi dengan *Firebase*. *File* JSON dijadikan sumber kredensial. File tersebut didapatkan saat mengunduh kredensial pada *Firebase*. Pada *listing code* ini `firebase_admin.initialize_app(cred)` bermaksud untuk menggunakan kredensial yang sudah dibuat sebelumnya. Fungsi `initialize_app` digunakan untuk melakukan inisialisasi aplikasi *Firebase* menggunakan kredensial yang valid. Pada baris terakhir `db = firestore.client()` merupakan fungsi yang mewakili koneksi ke *Firestore*. Fungsi ini digunakan untuk melakukan koneksi antara *Firestore* dengan proyek *Firebase* yang sudah diinisialisasi sebelumnya. Fungsi `db` mengakses dan mengelola data di *Firestore*. Penggunaan *listing code* di atas digunakan untuk menghubungkan *Firebase* dengan proyek menggunakan *Firebase* Admin SDK dan dapat menggunakan *Firestore* untuk mengelola data sesuai kebutuhan.

4.2 Analisis Model *Universal Sentence Encoder* (USE)

Universal Sentence Encoder atau USE merupakan sebuah model yang dapat digunakan untuk mengkodekan sebuah kalimat ke dalam vektor *embedding* yang berguna untuk memproses bahasa alami (NLP). Model yang menggunakan USE mempelajari makna dan sifat semantik kalimat dari data yang sangat besar. Model USE dirancang guna memaksimalkan kinerja *transfer learning* ke tugas-tugas NLP [37].

Penggunaan algoritma USE pada penelitian ini dikarenakan model USE merupakan model yang mampu menghasilkan representasi vektor numerik dari teks secara general. Selain itu, keunggulan USE ialah kemampuan dalam mengkodekan teks dengan panjang dan kompleksitas yang berbeda menjadi vektor numerik dengan dimensi yang sama. Hal ini menunjukkan sebuah perbandingan dan pencocokan teks yang lebih efisien dan akurat. Alasan penggunaan metode USE selajutnya yaitu kemampuan *transfer learning* yang baik. Hal ini dapat dilihat saat

model USE sudah dilatih menggunakan teknik *transfer learning* yang membuat model USE tidak perlu dilatih kembali. penggunaan model USE yang sudah dilatih sebelumnya dapat digunakan dalam data yang besar dan kompleks.

Efisiensi komputasi juga merupakan salah satu alasan penggunaan model USE pada penelitian ini. Hal ini pengaruhi dengan penggunaan *Tensorflow* Hub yang membantu dalam pemrosesan dan pengunduhan file proyek. Oleh karena itu, penggunaan *Tensorflow* Hub membantu model USE yang dijalankan menjadi lebih maksimal dan optimal dalam melakukan pemrosesan. USE dirancang untuk menghasilkan representasi teks secara general. Pada penelitian ini, model USE mampu untuk mengkodekan cerita pengguna yang panjang dan tingkat kompleksitas yang berbeda.

Pada penelitian ini menggunakan *dataframe* pandas dan operasi vektorisasi. Penggunaan *dataframe* pandas dapat dilihat pada *listing code* dimana *pandas* memungkinkan untuk melakukan manipulasi data yang efisien dan penggunaan vektorisasi dapat membantu proses komputasi atau pengolahan data secara bersama-sama atau serentak.

```
# Define a function to find the top N most similar users to
a given user
def    find_top_similar_users(current_user_ID,    user_data,
user_story, embed, n):
    # Check if current user not found
              if      user_data.loc[user_data['ID']      ==
current_user_ID].empty:
        return "Current user not found!"
    # Get the current user's data and story
       current_user   =   user_data.loc[user_data['ID']   ==
current_user_ID]
           current_user_story    =    f"I    have    Skill
{current_user['skills'].values.item()} , and I'm Interested
in {current_user['job_interest'].values.item()}"
    # Encode the current user story into a vector
    current_user_vector = embed([current_user_story])
```

```
    # Encode all other user stories into vectors and store
them in a matrix along with the user id
    other_user_vectors = []
    other_user_ID = []
    for user in user_story:
        vector = embed([user["story"]])
        other_user_vectors.append(vector)
        other_user_ID.append(user["ID"])
    other_user_matrix = np.array(other_user_vectors)
    print(vector.shape)
    # Calculate the similarity scores between the current
user vector and all other user vectors in the matrix
        similarity_scores   =   tf.matmul(other_user_matrix,
tf.transpose(current_user_vector))
    # Get the top N most similar users and their scores
                            most_similar_users              =
np.argsort(similarity_scores.numpy().reshape(-1))[::-1][:n]
      most_similar_user_ID  =  [other_user_ID[i]  for  i  in
most_similar_users]
                        most_similar_user_scores            =
similarity_scores.numpy().reshape(-1)[most_similar_users]
    # Convert the similarity scores to float64
                        most_similar_user_scores            =
most_similar_user_scores.astype(np.float64)
    # Create a list of dictionaries containing the user ID
and similarity score for each of the top N most similar users
    similar_users = []
    for i in range(1, n):
           similar_user  =  {"ID":  most_similar_user_ID[i],
"similarity_score": most_similar_user_scores[i]}
        similar_users.append(similar_user)

    return similar_users
```

Proses di atas merupakan proses rekomendasi yang dilakukan pada model USE ini menggunakan fungsi `find_top_similar_users` yang bertujuan untuk mencari N dari pengguna yang memiliki kesamaan teratas dengan pengguna lain. Parameter N yang dimaksud ialah parameter yang menentukan jumlah pengguna teratas yang akan dicari. Dalam konteks ini, variabel N merupakan bilangan bulat yang akan memperlihatkan berapa banyak pengguna yang memiliki kesamaan teratas.

Langkah pertama yang dilakukan untuk proses rekomendasi ialah memeriksa keberadaan *user* atau pengguna. Fungsi yang digunakan adalah metode `loc` pada *dataframe* `user_data` guna memeriksa ID para pengguna saat ID pengguna tidak ditemukan fungsi secara langsung akan memberikan pesan berisikan "*Current user not found*!". Selanjutnya, *user* atau pengguna yang ditemukan dalam data akan mendapatkan data *user* dan cerita *user* berdasarkan ID *user* atau pengguna. Fungsi yang digunakan masih sama seperti saat memeriksa ID *user* dengan fungsi `loc`. Pembuatan variabel `current_user_story` berisikan penggabungan parameter *skill* dan *job interest* dalam sebuah cerita.

Cerita *user* diubah menjadi sebuah vektor dengan menggunakan fungsi `embed`. Fungsi `embed` merupakan fungsi yang menerima teks sebagai *input* dan menghasilkan representasi vektor cerita *user*. Vektor cerita *user* akan disimpan pada daftar `other_user_vektors` dan ID *user* akan disimpan pada variabel `other_user_ID`.

```
 # Encode all other user stories into vectors and store them
in a matrix along with the user id
    other_user_vectors = []
    other_user_ID = []
    for user in user_story:
        vector = embed([user["story"]])
        other_user_vectors.append(vector)
        other_user_ID.append(user["ID"])
    other_user_matrix = np.array(other_user_vectors)
    print(vector.shape)
```

Pada *listing code* di atas digunakan untuk menentukan dimensi yang digunakan matriks akan disiapkan dengan memanfaatkan `np.zeros` untuk membuat matriks kosong dengan dimensi yang diinginkan. Pada penelitian ini ukuran matriks didapatkan dari (jumlah_pengguna, dimensi_vektor), jumlah pengguna merupakan jumlah total pengguna lain dalam data dan dimensi vektor merupakan dimensi vektor cerita *user*. Pada penelitian ini menggunakan dimensi vektor sebesar 512. Dimensi vektor 512 meningkatkan skalabilitas sehingga dapat menyelesaikan segala macam tugas. Vektor 512 memberikan performa yang baik dalam tugas pemprosesan bahasa alami. Dalam pemprosesan bahasa alami, dimensi vektor yang lebih besar akan membantu model mengenali pola-pola kompleks dan lebih baik dalam melakukan pemahaman tentang makna dan hubungan antar kata. Dimensi vektor yang lebih rendah juga akan memberikan hasil yang baik tergantung dengan dataset yang dihadapi.

Vektor cerita akan disimpan dalam matriks dengan menggunakan `np.array` dan fungsi `np.vstack`. Alasan menggunakan dua buah fungsi tersebut karena pustaka *NumPy* mampu membuat array dari objek yang diberikan. `np.array` akan dipakai dalam pembuatan array dari hasil hasil *embedding*. Selain itu, `np.array` akan melakukan konversi hasil *embedding* dalam bentuk array yang dapat diolah menggunakan operasi *NumPy*. Penggunaan fungsi `np.vstack` akan menggabungkan array numpy yang mewakili *embedding* dari cerita *user* menjadi satu array numpy tunggal. Hal ini akan menghasilkan vektor cerita *user* dengan satu matriks yang lebih besar dari sebelumnya. Dengan melakukan iterasi cerita vektor dan menyimpan vektor cerita *user* lainnya dalam bentuk matriks memudahkan dalam perhitungan skor kesamaan antar vektor pengguna.

Mencari skor kesamaan atau *similar score* pada penelitian menggunakan metode USE memanfaatkan fungsi `tf.transpose` untuk melakukan operasi transposisi. Selain menggunakan operasi transposisi juga menggunakan fungsi `tf.matmul` untuk operasi perkalian matriks. Pada penelitian ini matriks yang dikalikan ialah matriks `other_user_matrix` dengan matriks transposisi `current_user_vevtor`. Hasil dari operasi perhitungan yang dilakukan

merupakan matriks yang memiliki dimensi (m,1) dimana M adalah jumlah *user* lain dalam `other_user_matrix`.

Pada *listing code* `most_similar_users = np.argsort(similarity_scores.numpy().reshape(-1))[::-1][:n]` terdapat fungsi `np.argsort` yang berfungsi untuk mengembalikan indeks dari elemen dalam *array* dari yang paling kecil hingga paling besar. `most_similar_user_ID = [other_user_ID[i] for i in most_similar_users]` akan membuat *list comprehension* untuk membuat *list* yang berisi ID *user* yang sesuai dengan indeks N *user* yang paling mirip dengan *user* saat ini.

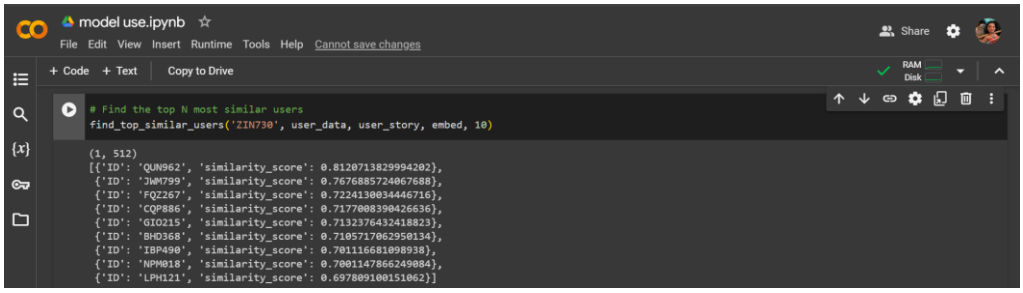
Setelah mendapatkan skor kesamaan yang bertipe data *NumPy* array diubah menjadi tipe data float64. Pengubahan ini dilakukan karena pada umumnya skor kesamaan bertipe bilangan real atau bilangan pecahan. Dalam *Tensorflow* memiliki tipe data *default* float32. Namun, saat pengubahan terjadi yang memanfaatkan *NumPy* yang memiliki tipe data *default* adalah float64. Selain itu, hal ini diperlukan guna memvalidasi kompatibilitas tipe data saat membuat daftar *dictionary* pengguna atau *user* teratas.

```
# Convert the similarity scores to float64
                        most_similar_user_scores          =
most_similar_user_scores.astype(np.float64)

    # Create a list of dictionaries containing the user ID
and similarity score for each of the top N most similar users
    similar_users = []
    for i in range(1, n):
            similar_user  =  {"ID":  most_similar_user_ID[i],
"similarity_score": most_similar_user_scores[i]}
        similar_users.append(similar_user)

    return similar_users
```

Pada proses pembuatan *dictionary user* teratas yang akan berisikan setiap pengguna teratas yang memiliki kemiripan yang tinggi terhadap pengguna saat ini. *Dictionary* akan memiliki dua buah kunci yaitu "*user_id*" dan "*score*" yang mewakili ID *user* dan skor kesamaan *user*. Penggunaan *loop* `for` dan `zip()` untuk menggabungkan daftar ID *user* dan daftar skor kesamaan menjadi satu objek yang memudahkan proses iterasi secara bersamaan. Pembuatan *dictionary* `user_data` akan ditambahkan pada daftar `most_similar_users_data` yang menggunakan metode `append()`. Hal ini akan memunculkan pengguna teratas yang paling mirip dengan pengguna saat ini. Hasil akhir yang didapatkan berupa daftar *dictionary* yang berisikan ID *user* dan skor kesamaan yang dapat dilihat sebagai berikut.



Gambar 4.5 Hasil Akhir Rekomendasi

Pada Gambar 4.5 memperlihatkan *listing code* `find_top_similar_users('ZIN730', user_data, user_story, embed, 10)` akan memunculkan 10 pengguna dengan kesamaan tearatas dengan *user* yang memiliki ID "ZIN730" dan hasil yang dapat dilihat akan berupa daftar *dictionary* yang berisi ID pengguna dan skor kesamaan antar pengguna.

4.2.1 *Analisis Similar Score*

Similar score atau skor kesamaan salah satu langkah dalam membangun sistem rekomendasi yang menghasilkan skor kesamaan atau kemiripan antar *item* yang terdapat pada dataset. *Similar score* berfungsi sebagai pembanding preferensi pengguna dengan *item* yang ada pada dataset sehingga akan menghasilkan skor kemiripan tertinggi dapat direkomendasikan kepada pengguna. *Similar score* pada

penelitian ini dihitung menggunakan *dot product* untuk mengukur sejauh mana dua vektor berada dalam arah yang sama. Operasi perkalian matriks yang dilakukan antara `other_user_matrix` dan *transpose* dari `current_user_vector` menggunakan fungsi `tf.matmul()` yang akan menghasilkan *similar score* pada variabel `similarity_scores`. Pada *dot product* akan menghitung *similarity score* ketika vektor representasi *user* sudah dilakukan normalisasi sehingga hasil *dot product* dapat dipengaruhi oleh panjang vektor. Berikut merupakan hasil rekomendasi yang dapat dilihat di bawah ini.

Tabel 4. 1 Hasil ID *User* dengan *Similar Score*

ID *User*	***Similarity Score***
QUN962	0.81
JWM799	0.76
FQZ267	0.72
CQP886	0.71
GIO215	0.71
BHD368	0.71
IBP490	0.70
NPM018	0.70
LPH121	0.69

Pada Tabel 4.1 Hasil *similar score* berada pada rentang nilai tertentu seperti 0 dan 1. Nilai *similar score* yang mendekati 1 menunjukkan tingkat kesamaan yang tinggi dan sebaliknya jika nilai mendakati 0 menunjukkan tingkat kesamaan yang rendah. *Similar score* dapat menunjukkan kualitas representasi yang dihasilkan oleh model *embed*. Jika *similar score* antar *user* memiliki nilai yang tinggi maka menunjukkan bahwa model *embed* berhasil dalam menangkap kesamaan atau keterkaitan antar pengguna. Namun, jika *similar score* rendah maka menunjukkan model *embed* perlu diperbaiki.

Melihat *similar score* yang dihasilkan antar *user* terdapat faktor-faktor kontribusi yang berpengaruh. Faktor-faktor yang digunakan pada penelitian ini ialah *skills* dan *job interest*. Faktor tersebut akan memperlihatkan keterampilan yang mirip atau minat kerja yang serupa memiliki *similar score* yang lebih tinggi. *Similar score* membantu dalam proses mengidentifikasikan pengguna yang memiliki *similar score* rendah. *User* dengan *similar score* yang rendah dapat dianggap sebagai *outlier* atau pengguna yang memiliki preferensi *skills* maupun *job interest* yang berbeda secara signifikan. Analisis *outlier* membantu dalam pemahaman lebih lanjut tentang variasi dalam data yang digunakan.

4.3 Analisis Model *Tensorflow Recommendation* (TFRS)

Tensorflow Recommendation atau yang biasa disebut TFRS merupakan sebuah pustaka atau *library* untuk membantu sistem rekomendasi menggunakan *Tensorflow*. Penggunaan TFRS dirancang khusus untuk memudahkan dalam proses membangun sebuah model rekomendasi. TFRS menyediakan komponen yang membantu sistem rekomendasi, seperti pemprosesan data, pembuatan fitur, pemodelan, hingga evaluasi [38].

Pada penelitian ini TFRS digunakan karena memungkinkan penggunaan fitur dan pemrosesan paralel. TFRS juga menyediakan perpustakaan yang bagus dalam membangun model sistem rekomendasi. TFRS dapat memanfaatkan kemampuan *Tensorflow* dalam mengelola dan memproses data dalam skala yang besar, serta mampu mengoptimalkan kinerja sistem rekomendasi. Pada metode TFRS dapat menyesuaikan fungsi kerugiaan (*loss function*), menentukan matriks evaluasi, dan menentukan arsitektur model yang lebih baik. Dengan adanya dukungan tersebut membuat pengguna merasakan fleksibelitas yang tinggi.

Pada penelitian ini menggunakan *Tensorflow Recommenders*. Selain itu, menggunakan modul dan pustaka antara lain, *Pandas*, *NumPy*, dan lain-lain. Pembuatan kelas *Config* diperlukan karena untuk menyimpan konfigurasi yang berhubungan dengan model rekomendasi. Pada kelas *Config* berisikan dimensi *embedding* dan *path* untuk penyimpanan model. Pada Gambar 4.1 dan 4.2 merupakan dataset yang digunakan pada metode TFRS.

Proses *Preprocessing* dilakukan untuk memastikan tipe data yang digunakan sudah tepat. Berikut proses *preprocessing* yang dilakukan pada metode TFRS.

```
df['skills'] = df[['skill_1', 'skill_2', 'skill_3',
'skill_4']].apply(lambda x: ', '.join(x), axis=1)
df = df.drop(['skill_1', 'skill_2', 'skill_3', 'skill_4'],
axis=1)

df[['id', 'skills', 'job_interest']] = df[['id', 'skills',
'job_interest']].astype(str)
df.loc[:, ['id', 'skills', 'job_interest']] = df[['id',
'skills', 'job_interest']].astype(str)
```

Pada proses *preprocessing* dilakukan dengan menggabungkan dan mengubah format data. Proses penggabungan kolom *skill* menjadi satu buah kolom dan menghapus kolom *skill* yang tidak diperlukan serta mengubah format data kolom menjadi *string*. Pada proses ini terdapat fungsi *lambda* yang berguna untuk meggabungkan nilai dari setiap baris menjadi satu *string* dan fungsi *drop* digunakan untuk menghapus kolom.

Proses selanjutnya ialah mempersiapkan data dengan menggabungkan nilai-nilai yang terdapat pada kolom. kemudian akan mengonversi kolom-kolom tersebut menjadi *array NumPy*. Dataset *interaction* merupakan data interaksi antara pengguna dan *item*, sedangkan data *items* berisikan informasi tentang *item* yang digunakan. Data *users* memtakan setiap *item* yang ada pada dataset *interaction* menjadi sebuah *dictionary* yang berisikan informasi pengguna. dataset *job_item* berisikan nilai *job_interest* dari dataset *items*.

Proses pembagian data *train* dan data *test* atau *splitting* data dilakukan dengan mengatur *seed* untuk memastikan hasil pengacakan yang konsisten. Dataset *users* diacak dan diambil 400 pertama untuk dilatih, sementara 100 berikutnya diambil untuk pengujian. Dengan kata lain perbandingan data *train* dengan data *test* sebesar 4:1. Memanfaatkan fungsi `shuffle()` dan `take()` serta `skip()` dataset

diacak dan dibagi secara acak dengan proporsi yang telah ditentukan. *Splitting* data dengan perbandingan 4:1 (80:20) dikarenakan banyak data untuk pengujian (20%) akan memperoleh evaluasi model yang lebih kredibel. Selain itu, perbandingan 4:1 mengurangi *overfitting* karena 80% data pelatihan akan dilatih dengan baik dan memiliki kemampuan yang baik dalam mempelajari pola.

Proses membangun model TFRS menggunakan dimensi 32 yang akan digunakan pada model. Penggunaan *embedding* berdimensi 32 digunakan untuk mengurangi dimensi fitur dan menyedehanakan representasi *item* dan pengguna dengan menggunakan dimensi yang rendah dapat membantu dalam mengurangi dimensi parameter model dan menghindari *overfitting*. Pada Gambar 4.12 akan memperlihatkan proses pembuatan model sistem rekomendasi menggunakan metode TFRS.

```
embedding_dimension = 32
job_model = tf.keras.Sequential([
            tf.keras.layers.experimental.preprocessing.Strin
gLookup(
                                vocabulary=unique_item_titles,
mask_token=None),
             tf.keras.layers.Embedding(len(unique_item_title
s) + 1, embedding_dimension)
                                  ])

user_model = tf.keras.Sequential([
             tf.keras.layers.experimental.preprocessing.Stri
ngLookup(
                                   vocabulary=unique_user_ids,
mask_token=None),
# We add an additional embedding to account for unknown
tokens.
              tf.keras.layers.Embedding(len(unique_user_ids)
+ 1, embedding_dimension)
                                  ])
```

Pada proses diatas memperlihatkan model TFRS terdiri dari dua lapisan, lapisan pertama `tf.keras.layers.experimental.preprocessing.StringLookup` digunakan untuk mengubah nilai *string* menjadi representasi numerik. Lapisan kedua terdiri dari `tf.keras.layers.Embedding` untuk memetakan nilai numerik ke dalam ruang *embedding*. Kedua *layers* ini memudahkan dalam pemroses data kategorikal seperti *jobs* dan ID *user* dalam model. Proses selanjutnya dapat dilihat pada *listing code* berikut.

```
class JobLensModel(tfrs.Model):

  def __init__(self, user_model, jobs_model):
    super().__init__()
    self.jobs_model: tf.keras.Model = jobs_model
    self.user_model: tf.keras.Model = user_model
    self.task: tf.keras.layers.Layer = task

   def compute_loss(self, features: Dict[Text, tf.Tensor],
training=False):
    # Define how the loss is computed.

    user_embeddings = self.user_model(features["id"])
                                jobs_embeddings              =
self.jobs_model(features["job_interest"])

    return self.task(user_embeddings, jobs_embeddings)
```

Pada proses diatas memperlihatkan penggunaan matriks *Factorized Top* K untuk evaluasi model dan mengukur kinerja rekomendasi dengan memeriksa peringkat *item* yang direkomendasikan berdasarkan faktorisasi embedding. Pembuatan kelas *JobLensModel* yang merupakan turunan dari `tfrs.Model.Kelas` bertugas menggabungkan model pengguna (*user*_model),

model pekerjaan (*job*_model), dan tugas (*task*). Tugas (*task*) yang berasal dari `tfrs.tasks.Retrieval` untuk menentukan tugas utama model yaitu mengambil *item* yang relevan untuk setiap pengguna. Metode *compute_loss* digunakan untuk menghitung nilai kerugian (*loss*) pada model.

Proses selanjutnya ialah melatih model yang sudah dibuat sebelumnya. Model di random *seed* karena memastikan hasil acak yang diberikan model tetap konsisten. Model TFRS pada penelitian ini memanfaatkan Adagrad dan *learning rate* sebesar 0,1. Adagrad ialah algoritma pengoptimal yang mengadaptasi *learning rate* sebagai parameter model berdasarkan gradiennya. Adagrad dipilih karena memiliki kococokan dengan tugas yang diberikan pada model. *learning rate* bertugas mengontrol dalam hal pembobotan dan bias model selama pelatihan data. Jika *learning rate* terlalu kecil, maka model membutuhkan waktu yang lama untuk mencapai konvergensi atau terjebak pada titik minimum. Jika *learning rate* terlalu besar model mengalami kesulitan dalam mencapai konvergensi dan melompati titik minimum. Pemilihan learning rate 0,1 berdasarkan data dan kompleksitas model. Nilai ini didasarkan pada evaluasi dan penelitian terdahulu dengan model dan dataset yang sama. Proses pengacakan data pelatihan memisahkan menjadi *batch-batch* dan menyimpan data di *cache* guna membantu proses penyiapan data dengan efisien saat proses pelatihan dan pengujian model. Pengacakan yang dilakukan untuk memperkenalkan variasi dari setiap *epoch*. Ukuran *batch* yang dipakai sebesar 8192 yang mengartikan bahwa data pelatihan diproses dalam batch sebesar 8192 sampel pada setiap proses iterasi pelatihan.

Caching data dilakukan karena dapat mengurangi *overhead* pemprosesan data yang berulang dan proses pelatihan serta evaluasi model menjadi lebih cepat dan efisien. *Epoch* merujuk pada pelatihan model pada satu kali iterasi, pada penelitian ini melakukan 10 *epoch*. Pada setiap *epoch* model diperbarui berdasarkan nilai *loss* dan matriks yang dihitung pada data pelatihan. Tujuan adanya data pengujian untuk mengawasi model dan menghindari *overfitting*. Proses berlanjut dengan mengevaluasi model dengan data pengujian untuk memperoleh matriks evaluasi seperti *loss* dan matriks *top*-k. Hasil rekomendasi dapat di lihat pada Gambar 4.6

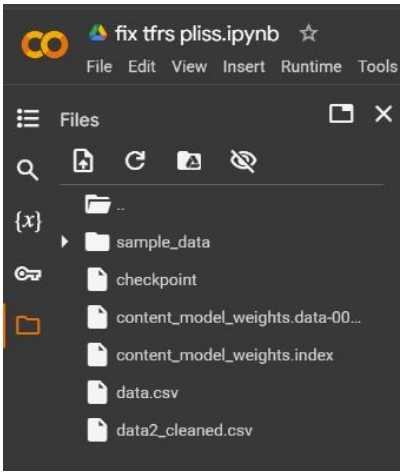
Gambar 4.6 Hasil Rekomendasi Metode TFRS

Pada Gambar 4.6 menunjukkan bahwa penggunaan *BruteForce* yang sederhana untuk mendapatkan rekomendasi dari dataset yang diberikan. Hasil rekomendasi dapat dilihat pada gambar di atas disimpan dalam variabel sehingga jika ingin melihat hasilnya hanya perlu menggunakan indeks berdasarkan user ID yang diberikan. *index*() mendapatkan rekomendasi berdasarkan ID pengguna. Selain itu, terdapat *loop for* yang melakukan iterasi melalui indeks dari list ID.

Hasil rekomendasi disimpan pada *path* yang sudah dideklarasikan dapat dilihat pada *listing code* berikut.

```
model.save_weights('content_model_weights',
save_format='tf')
loaded_model = JobLensModel(user_model, job_model)
loaded_model.load_weights('content_model_weights')
```

Pada proses diatas menggunakan *code* berguna untuk menyimpan bobot (*weights*) dari `model.save_weights('content_model_weights', save_format='tf')`model ke dalam *file*. `Save_format='tf'` untuk menyimpan model bobot dalam bentuk format *Tensorflow*. Setelah model disimpan terdapat *code* `loaded_model=JobLensModel(user_model, job_model)` dan `loaded_model.load_weights('content_model_weights', save_format='tf')` digunakan untuk memuat bobot model yang telah disimpan.



Gambar 4.7 Lokasi File TFRS

Pada Gambar 4.7 dapat dilihat hasil rekomedasi disimpan dalam sebuah *path* yang sudah dibuat. *File* yang disimpan tidak hanya *file* model saja tetapi terdapat *file* yang berisikan bobot model dan bias model. hal ini dilakukan karena menjadi langkah penting dalam proses pengembangan model tanpa perlu melatih atau mengulangi proses latihan dari awal. Bobot model dan bias model memiliki ukuran tergantung dengan tingkat kompleksitas model dan jumlah parameter. Penyimpanan bobot model dan bias model dilakukan untuk memudahkan dalam pemeliharaan kode dan *debugging*. Penyimpanan bobot dan bias model berhubungan dengan *transfer learning*, *transfer learning* ialah teknik menggunakan model yang sudah dilatih untuk mengerjakan tugas yang baru. Sehingga, model dapat dilatih pada dataset yang lebih besar dan kompleks dapat dilanjut melatih model dengan dataset yang lebih kecil.

4.3.1 Analisis *Loss Function*

Loss Function merupakan sebuah fungsi kerugian yang dimana terdapat matriks untuk mengukur seberapa baik model *machine learning* (ML) yang digunakan guna meningkatkan keakuratan. Pada sistem rekomendasi *loss function* digunakan untuk mengukur sejauh mana prediksi sistem rekomendasi yang cocok dengan preferensi atau perilaku pengguna [45]. tujuan utama *loss function* ialah mengoptimalkan model rekomendasi agar memberikan rekomendasi yang paling

relevan dan sesuai dengan kebutuhan pengguna. Pada penetian ini melihatkan hasil matriks akurasi dan loss yang diukur selama evaluasi model. Hasil *Loss function* dapat dilihat sebagai berikut.

```
25/25  [==============================]  -  1s  39ms/step  -
factorized_top_k/top_1_categorical_accuracy:  0.0000e+00  -
factorized_top_k/top_5_categorical_accuracy:    0.0200    -
factorized_top_k/top_10_categorical_accuracy:   0.0600    -
factorized_top_k/top_50_categorical_accuracy:   0.5200    -
factorized_top_k/top_100_categorical_accuracy:   1.0000   -
loss: 5.5447 - regularization_loss: 0.0000e+00 - total_loss:
5.5447
{'factorized_top_k/top_1_categorical_accuracy': 0.0,
 'factorized_top_k/top_5_categorical_accuracy':
0.019999999552965164,
 'factorized_top_k/top_10_categorical_accuracy':
0.05999999865889549,
 'factorized_top_k/top_50_categorical_accuracy':
0.5199999809265137,
 'factorized_top_k/top_100_categorical_accuracy': 1.0,
 'loss': 5.559291839599609,
 'regularization_loss': 0,
 'total_loss': 5.559291839599609}
```

Pada hasil diatas menunjukkan hasil kategori teratas mulai dari *top*-1 *categorical accuracy* hingga *top*-100 *categorical accuracy*. *Top*-1 *categorical accuracy* menunjukkan hasil 0.0 yang bermaksud seberapa sering *item* yang benar muncul sebagai prediksi teratas dalam daftar rekomendasi. Nilai 0.0 memiliki arti tidak ada prediksi yang benar dalam kategori teratas. Dilanjutkan dengan *top*-5 *categorical accuracy* yaitu seberapa sering *item* yang benar muncul diantara 5 prediksi teratas dalam daftar rekomendasi. Nilai *top*-5 *categorical accuracy* sebesar 0.05 yang menunjukkan bahwa item yang benar muncul dalam 5 prediksi tearatas sebanyak 5%. Pada *top*-50 *categorical accuracy* mendapatkan nilai akurasi sebesar

0.55 yang menunjukkan bahwa *item* yang benar muncul dalam 50 prediksi teratas sebanyak 55%.

Matriks mengukur kategori teratas 100 atau *top*-100 *categorical accuracy* yaitu seberapa sering *item* yang benar muncul diantara 100 prediksi teraratas dalam daftar rekomendasi. Nilai akurasi ini sebesar 1.0 yang menunjukkan *item* yang benera selalu muncul diantara 100 prediksi teratas. Nilai *top*-1 *categorical accuracy* yang kecil atau sangat rendah terdapat beberapa faktor yaitu terdapat *noise* atau kebisingan pada model dan keterbatasan representasi. Hal ini dapat ditanggulangi dengan memperbaiki kualitas data dengan melakukan *cleaned* data. Selain itu, bisa menggunakan augmentasi data dengan meningkatkan keberagaman data pelatihan dan membuat model yang lebih kompleks.

Nilai *regularization_loss* menunjukkan nilai *loss* yang terkait dengan metode regulasi yang digunakan pada model. Pada hasil *regularization_loss* model TFRS menunjukkan nilai sebesar 0. Hal ini terjadi karena tidak ada *loss* yang terjadi akibat regularisasi. Regularisasi merupakan teknik untuk mengendalikan kompleksitas model dan mencegah *overfitting*. Ketika model memiliki performa yang baik tanpa *overfitting* nilai *regularization_loss* yang rendah dapat dianggap normal.

Total *loss* merupakan matriks yang mengukur sejauh mana prediksi model berada dari nilai yang sebenarnya. Semakin rendah nilai total *loss* maka semakin baik model dalam menghasilkan prediksi yang akurat, begitupun sebaliknya. Pada penelitian ini nilai total *loss* yang didapatkan sebesar 5.50. Maka dapat dibilang model yang dibuat menunjukkan tingkat prediksi yang cukup baik meninjau hasil *loss* yang kecil.

4.4 Analisis Pengembangan Model dan Tingkat Akurasi

Pada penelitian ini menggunakan 2 metode yaitu *Universal Sentence Encode*r (USE) dan *Tensorflow Recommendation* (TFRS). Aplikasi Collabolio hadir sebagai platform sosial-kolaboratif yang menggunakan algoritma berbasis perjodohan, yang biasanya digunakan pada aplikasi kencan, untuk kreasi portofolio diciptakan untuk membantu pengguna menemukan rekan kerja yang tepat untuk

berkolaborasi dalam proyek-proyek kehidupan nyata yang bertujuan untuk mengatasi masalah pengembangan yang kurang berpengalaman terutama dalam kerja sama tim. Berikut ini merupakan Tabel 4.1 yang berisikan hasil pengembangan dan hasil rekomendasi pada penelitian.

Tabel 4. 2 Hasil Pengembangan dan Tingkat Akurasi

No	**Parameter**	***Universal Sentence Encoder* (USE)**	***Tensorflow Recommendation* (TFRS)**
1.	Dataset	*Jobs Interest* dan *Skill*s	*Jobs Interest* dan *Skill*s
2.	Matriks Evaluasi	*Similar Score*	*Loss Fuction*
3.	Penyimpanan Dataset Penelitian	*Firebase*	Google Colab dalam bentuk file .csv
4.	Hasil Rekomendasi	*User* ID	*Jobs*

Pada Tabel 4.1 menjelaskan hasil rekomendasi menggunakan USE akan memunculkan *user* atau pengguna mana saja yang memiliki *background* yang sama dengan *user* atau pengguna lainnya dengan melihat *skill*s dan *job interest* dari masing-masing *user* atau pengguna. Hasil rekomendasi tersebut diperoleh dari nilai *similar score* atau skor kesamaan. *Similar score* dapat ditingkat tergantung dengan metode yang digunakan, berdasarkan latar belakang penelitian USE mampu memberikan peforma yang baik dalam melakukan sistem rekomendasi. Penyimpanan dataset pada metode USE memanfatkan platform *Firebase* sebagai *Cloud* peyimpanan memudahkan dalam penyimpanan dan memproses data.

Model *Tensorflow Recommendation* merupakan pengembangan dari model aplikasi Collabolio. Pada penelitian ini selain mendapatkan teman jejaring sosial yang memiliki *background* yang sama tetapi juga dapat merekomendasikan pekerjaan yang cocok dengan *background* yang dimiliki user atau pengguna. TFRS mampu memberikan hasil rekomendasi yang diharapkan yaitu rekomendasi pekerjaan yang cocok dengan *user* atau pengguna. Rekomendasi tesebut didapatkan dari melihat *background user* mulai dari *skill*s yang dimiliki hingga jenis pekerjaan

yang dimiliki. Peningkatan akurasi pada metode TFRS yang sudah dilakukan ialah penyesuaian *learning rate* pada *optimizer*. Adapun beberapa hal yang harus ditingkatkan pada tahap pengembangan model dan peningkatan akurasi yaitu peningkatkan kualitas data yang lebih beragam. Dataset akan dihubungkan dengan Google Colab agar memudahkan dalam proses membangun model.