4. IMPLEMENTASI SISTEM

Pada bab ini akan dibahas mengenai implementasi dari desain sistem yang telah dibuat pada bab sebelumnya. Selain itu, proses-proses seperti *data preprocessing* dan *data testing* juga akan diuraikan pada bab ini.

4.1 Implementasi Perangkat Lunak dan Perangkat Keras

Pemrosesan data dilakukan dengan bantuan aplikasi Microsoft Excel dan Microsoft Power BI. Microsoft Excel digunakan untuk mengelola data yang membutuhkan pola berulang atau semacamnya, misalnya saat menghitung inflasi kontinu dari inflasi dan saat menambahkan id untuk data. Microsoft Power BI digunakan untuk melakukan *data cleaning* seperti menghapus data yang tidak sesuai dengan ketentuan, melakukan beberapa transformasi data, dan melakukan *merge query* untuk mendapatkan data *foreign key* dari suatu data. Untuk pembuatan *database*, aplikasi pgAdmin untuk mengelola PosgreSQL digunakan. Data yang sudah disiapkan diubah ke dalam format csv dan di-*import* ke dalam pgAdmin untuk menjadi sebuah *database* yang utuh. Sementara itu, untuk keseluruhan sistem aplikasi *visual studio code* digunakan untuk mengedit sistem yang seluruhnya ditulis menggunakan bahasa pemrograman *python*. Dalam penggunaan *python* sendiri, akan ada beberapa *library* yang digunakan:

1. Pandas, digunakan untuk membaca file excel dan csv pada saat pengujian awal.
2. Numpy, digunakan untuk mengolah suatu data misalnya mencari rata-rata, mencari nilai absolut, dan lain-lain.
3. Sklearn, digunakan dalam membuat model regresi linear, membagi *data training* dan *data test*, mencari nilai MAE, menjalankan *grid search*, dan lain-lain.
4. Psycopg2, digunakan dalam menghubungkan *code python* dengan *database* yang ada pada pgAdmin
5. Streamlit, digunakan untuk membangun *user interface* menggunakan bahasa pemrograman *python*
6. Streamlit_option_menu, digunakan sebagai pelengkap dalam membangun *user interface*
7. Statsmodel, digunakan untuk melakukan prediksi menggunakan ARIMA
8. Matplotlib, digunakan untuk membuat grafik visualisasi hasil prediksi
9. Seaborn, digunakan untuk membuat visualisasi hasil *correlation matrix*

10. Pmdarima, digunakan untuk mencari nilai auto arima dari p, d, dan q model

Sementara itu, untuk perangkat keras yang digunakan dalam membuat dan menjalankan sistem ini memiliki spesifikasi sebagai berikut:

- *Processor* : Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
- RAM : 12 GB
- *Hard Disk* : 1 TB
- Sistem Operasi : Windows 11 - 64-bit operating system, x64-based processor

4.2 Langkah-langkah Implementasi

Pada Tabel 4.1 akan diuraikan langkah-langkah dalam melakukan implementasi sistem beserta keterangannya dan *tools* yang digunakan.

Tabel 4.1

Langkah-langkah Implementasi

No	Langkah	Keterangan	*Tools*
1	*Data cleaning*	- Melakukan penghapusan terhadap data yang kosong di kolom gaji - Melakukan penghapusan terhadap data yang gajinya lebih besar dari 20.000.000 - Melakukan penstandardan nama-nama kota dari *input user*	Microsoft Excel, Microsoft Power BI
2	*Data transformation*	- Menghitung nilai kontinu dari nilai inflasi per tahun - Melakukan *unique* untuk program kuliah mahasiswa dan menjadikan dataset baru - Melakukan *unique* untuk pilihan jawaban yang ada dan menjadikan dataset baru - Melakukan *merge* untuk tabel data_mahasiswa dan daerah - Melakukan *merge* untuk tabel data_mahasiswa dan program - Melakukan *merge* untuk tabel pertanyaan dan pilihan	Microsoft Excel, Microsoft Power BI, Visual Studio Code

		- Mengubah format data-data yang tidak sesuai agar disesuaikan untuk proses selanjutnya	
3	*Data normalization*	- Melakukan percobaan untuk menormalisasi data menggunakan *min-max* - Melakukan percobaan untuk menormalisasi data menggunakan *max absolute* - Melakukan percobaan untuk menormalisasi data menggunakan *z-score* - Membandingkan dan mengevaluasi hasil normalisasi	Visual Studio Code
4	Menentukan variabel-variabel prediksi	- Mencari nilai p, d, dan q terbaik untuk prediksi UMR menggunakan ARIMA - Mencari nilai p, d, dan q terbaik untuk prediksi inflasi menggunakan ARIMA - Mencari nilai *error* prediksi regresi linear untuk setiap jenis SKKK - Mencari nilai *error* prediksi regresi linear untuk program, prodi, dan fakultas yang ada - Mencari nilai *error* untuk prediksi menggunakan regularisasi normal, LASSO, Ridge, dan Elastic Net - Mencari nilai *error* untuk penggunaan filter Q1-Q11 dari setiap jurusan - Membandingkan dan mengevaluasi hasil-hasil yang telah didapatkan - Memasukkan data metode regresi linear terbaik ke dalam database	Visual Studio Code
5	Pembuatan database	- Membuat id setiap data secara berurutan	Microsoft Excel, pgAdmin

		- Mengubah semua format dataset ke dalam csv - Melakukan *import* database dari dataset-dataset yang telah tersedia	
6	Pembuatan program	- Membuat tampilan *user interface* dari halaman aplikasi prediksi dan admin - Membuat *code* untuk melakukan pemfilteran dataset yang akan digunakan - Membuat *code* untuk pengecekan variabel yang akan digunakan - Membuat *code* untuk mencari regularisasi regresi linear terbaik - Membuat *code* untuk melakukan prediksi menggunakan ARIMA untuk UMR dan inflasi - Membuat *code* untuk melakukan prediksi menggunakan regresi linear	Visual Studio Code

4.3 Hubungan antara Desain dan Implementasi

Berdasarkan desain yang telah dibuat sebelumnya, dibuat segmen-segmen program sebagai implementasi dari sistem yang ingin dibuat. Hubungan antara keduanya dapat dilihat pada Tabel 4.2.

Tabel 4.2

Hubungan Desain dan Implementasi

Proses	Desain	Kegiatan	Segmen program
Data tranformation	Gambar 3.1	Mendapatkan data *unique* untuk program di UKP	4.1
		Memproses data kota/kabupaten di Indonesia	4.2

		Memproses data UMR di Indonesia	4.3
Normalisasi	Gambar 3.1	Melakukan normalisasi menggunakan *min-max*	4.4
		Melakukan normalisasi menggunakan *max absolute*	4.5
		Melakukan normalisasi menggunakan *z-score*	4.6
Prediksi menggunakan ARIMA	Gambar 3.1	*Grid search* untuk mencari parameter data UMR	4.7
		Grid search untuk mencari parameter data inflasi	4.8
Prediksi menggunakan regresi linear	Gambar 3.1	Pengujian *error* program, prodi, dan fakultas	4.9
		Pengujian jenis-jenis SKKK	4.10
		Pengujian pertanyaan-pertanyaan opsional	4.11
		Pengujian *error* regresi linear biasa	4.12
		Pengujian *error* regresi linear Ridge	4.13
		Pengujian *error* regresi linear LASSO	4.14

		Pengujian *error* regresi linear Elastic Net	4.15
Input dan *output* data	Gambar 3.2, Gambar 3.3.	*User interface* aplikasi prediksi gaji	4.16
Mengambil, menyimpan, dan mengedit data dalam *database*	Gambar 3.2, Gambar 3.3	Koneksi sistem ke *database*	4.17
Cek *error rate* prediksi dengan UMR, inflasi, dengan keduanya, dan tanpa keduanya	Gambar 3.2	Pencarian kombinasi variabel terbaik	4.18
Cek metode terbaik	Gambar 3.2	Pencarian metode regularisasi regresi linear terbaik	4.19
Prediksi gaji menggunakan regresi linear	Gambar 3.2	Prediksi gaji mahasiswa	4.20
Memverifikasi apakah data valid atau tidak	Gambar 3.3	Aplikasi untuk verifikasi data admin	4.21

4.4 Implementasi *Data Preprocessing*

4.4.1 *Data Cleaning & Transformation*

Pada bagian ini, akan ditampilkan berbagai segmen program untuk proses *data cleaning & tranformation* sebelum data akan digunakan lebih lanjut. Segmen program 4.1 menampilkan *code* untuk mendapatkan data *unique* dari program yang ada di Universitas Kristen Petra beserta prodi dan fakultasnya. Data tersebut nantinya akan digunakan dalam pembuatan dataset program. Segmen program 4.2 dan segmen program 4.3 merupakan *code* untuk memproses format data agar sesuai dengan format yang dibutuhkan. Segmen

program 4.2 untuk data kota/kabupaten di Indonesia, sementara segmen program 4.3 untuk data UMR Indonesia.

Segmen Program 4.1 Mendapatkan Data *Unique* Program

```
import pandas as pd

file_path = 'D:\Skripsi\coding\Data Gaji.xlsx'
df_asli = pd.read_excel(file_path)
program = df_asli['Checked Program'].str.title().unique()
prodi = []
fakultas = []
for i in program:
     df = df_asli[(df_asli['Checked Program'].str.title() ==
i.title())]
    prd = df['Prodi'].str.title().unique()
    fkl = df['Fakultas'].str.title().unique()
    prodi.append(prd[0])
    fakultas.append(fkl[0])
print("===PROGRAM===")
for i in range (0, len(program)):
    print(program[i])
print("===PRODI===")
for i in range (0, len(prodi)):
    print(prodi[i])
print("===FAKULTAS===")
for i in range (0, len(fakultas)):
    print(fakultas[i])
```

Segmen Program 4.2 Memproses Data Kota/Kabupaten di Indonesia

```
list = input("String yang ingin disesuaikan:")
arr = list.split("///\t")
arr2 = []
for i in arr:
    temp = i.split("/\t")
    temp = [x for x in temp if x != ""]
    arr2.append(temp)
for i in arr2:
    for j in range (1, len(i)):
        print(i[j] +","+ i[0])
```

Segmen Program 4.3 Memproses Data UMR Indonesia

```
strg = input("String yang ingin disesuaikan:")
strg = strg.split(";")
c = 0
for i in strg:
    dt = i.split("Rp")
    dt[0] = dt[0].replace(":", "").strip().upper()
    dt[1] = dt[1].replace(".", "").strip()
```

```
    c += int(dt[1])
    print(dt[0]+","+dt[1]+",2024")
print(c/38)
```

4.4.2 *Data Normalization*

Data yang sudah bersih dan siap untuk dipakai, akan coba dilakukan normalisasi. Normalisasi dilakukan dengan 3 metode yaitu *min-max, max absolute,* dan *z-score*. Karena percobaan normalisasi dilakukan dengan mencoba berbagai kemungkinan jenis regresi linear, beberapa *code* dinonaktifkan menggunakan komentar untuk bisa digunakan sewaktu-waktu jika dibutuhkan. Segmen program 4.4 merupakan normalisasi menggunakan *min-max*, segmen program 4.5 menggunakan *max absolute*, dan segmen program 4.6 menggunakan *z-score*.

Segmen Program 4.4 Normalisasi Menggunakan *Min-Max*

```
import pandas as pd
from datetime import datetime
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import metrics
import numpy as np

data = pd.read_excel('D:\Skripsi\coding\Data Gaji.xlsx')
reg = pd.read_excel('D:\Skripsi\coding\Data Inflasi.xlsx', sheet_name="Inflasi Tahunan")
reg2 = pd.read_csv('D:\Skripsi\coding\Indonesian Salary by Region (1997-2022).csv')
current_date_time = datetime.now()
current_year = current_date_time.year
daerah_filter = "INDONESIA"
filtered_reg = reg2[reg2['REGION'] == daerah_filter]
reg['Inflasi_cont_normalized'] = (reg['Inflasi Cont'] - reg['Inflasi Cont'].min()) / (reg['Inflasi Cont'].max() - reg['Inflasi Cont'].min())
filtered_reg['Salary_normalized'] = (filtered_reg['SALARY'] - filtered_reg['SALARY'].min()) / (filtered_reg['SALARY'].max() - filtered_reg['SALARY'].min())
program = data['Checked Program'].str.lower().unique()
list_mape = []
for i in program:
      df = data[(data['Checked Program'].str.lower() == i.lower()) & (data['Gaji'] <= 20000000)]
    df = df.dropna(subset=['IPK', 'SKKK Total', 'Gaji'])
     df['IPK_normalized'] = (df['IPK'] - df['IPK'].min()) / (df['IPK'].max() - df['IPK'].min())
```

```
     df['SKKK_normalized'] = (df['SKKK Total'] - df['SKKK
Total'].min()) / (df['SKKK Total'].max() - df['SKKK
Total'].min())
    df['Gaji_normalized'] = (df['Gaji'] - df['Gaji'].min()) /
(df['Gaji'].max() - df['Gaji'].min())
     df = pd.merge(df, reg, how='left', left_on='Tahun',
right_on='Tahun')
        df = pd.merge(df, filtered_reg, how='left',
left_on='Tahun', right_on='YEAR')
    if (len(df) > 1):
                          X     =     df[['IPK_normalized',
'SKKK_normalized',                'Inflasi_cont_normalized',
'Salary_normalized']]
       y = df['Gaji_normalized']
       X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=42)
              X_train2, X_test2, y_train2, y_test2 =
train_test_split(X,        df['Gaji'],        test_size=0.2,
random_state=42)
        # Mencari alpha terbaik untuk ridge
        # try:
        #     param_grid = {'alpha': np.logspace(-4, 4, 50)}
        #     ridge_reg = Ridge()
       #    grid_search = GridSearchCV(ridge_reg, param_grid,
cv=5)
        #     grid_search.fit(X_train, y_train)
                    #               best_alpha_ridge    =
grid_search.best_params_['alpha']
        # except Exception as e:
        #     best_alpha_ridge = 1.0
        # Mencari alpha terbaik untuk lasso
        # try:
        #     param_grid = {'alpha': np.logspace(-4, 4, 50)}
        #     lasso_reg = Lasso()
       #    grid_search = GridSearchCV(lasso_reg, param_grid,
cv=5)
        #     grid_search.fit(X_train, y_train)
        # best_alpha_lasso = grid_search.best_params_['alpha']
        # except Exception as e:
        #     best_alpha_lasso = 1.0
        # Mencari alpha terbaik untuk elastic net
        try:
            param_grid = {'alpha': np.logspace(-4, 4, 50)}
            enet_reg = ElasticNet()
             grid_search = GridSearchCV(enet_reg, param_grid,
cv=5)
            grid_search.fit(X_train, y_train)
           best_alpha_enet = grid_search.best_params_['alpha']
        except Exception as e:
            best_alpha_enet = 1.0
        model = ElasticNet(alpha=best_alpha_enet)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
```

```
        real_data = ((df['Gaji'].max() - df['Gaji'].min()) *
y_pred) + df['Gaji'].min()
        # abs_percentage_error = np.abs((y_test - y_pred) /
y_test)
       # abs_percentage_error[np.isinf(abs_percentage_error)]
= np.nan
        # mape = np.nanmean(abs_percentage_error) * 100
       mape = np.mean(np.abs((y_test2 - real_data) / y_test2))
* 100
    else:
        mape = "Data tidak mencukupi"
    list_mape.append(mape)
for i in list_mape:
    print(i)
```

Segmen Program 4.5 Normalisasi Menggunakan *Max Absolute*

```
import pandas as pd
from datetime import datetime
from sklearn.linear_model import LinearRegression, Ridge,
Lasso, ElasticNet
from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn import metrics
import numpy as np
from sklearn.preprocessing import MaxAbsScaler

data = pd.read_excel('D:\Skripsi\coding\Data Gaji.xlsx')
reg = pd.read_excel('D:\Skripsi\coding\Data Inflasi.xlsx',
sheet_name="Inflasi Tahunan")
reg2 = pd.read_csv('D:\Skripsi\coding\Indonesian Salary by
Region (1997-2022).csv')
current_date_time = datetime.now()
current_year = current_date_time.year
daerah_filter = "INDONESIA"
filtered_reg = reg2[reg2['REGION'] == daerah_filter]
scaler = MaxAbsScaler()
reg[['Inflasi_cont_normalized']]                                =
scaler.fit_transform(reg[['Inflasi Cont']])
filtered_reg[['Salary_normalized']]                             =
scaler.fit_transform(filtered_reg[['SALARY']])
listMape = []
program = data['Checked Program'].str.lower().unique()
for i in program:
      df = data[(data['Checked Program'].str.lower() ==
i.lower()) & (data['Gaji'] <= 20000000)]
    df = df.dropna(subset=['IPK', 'SKKK Total', 'Gaji'])
             df[['IPK_normalized',      'SKKK_normalized',
'Gaji_normalized']] = scaler.fit_transform(df[['IPK', 'SKKK
Total', 'Gaji']])
    max_abs_value = df['Gaji'].abs().max()
     df = pd.merge(df, reg, how='left', left_on='Tahun',
right_on='Tahun')
```

```
        df   =   pd.merge(df,   filtered_reg,   how='left',
left_on='Tahun', right_on='YEAR')
    if (len(df) > 1):
            X  =  df[['IPK_normalized',  'SKKK_normalized',
'Inflasi_cont_normalized', 'Salary_normalized']]
        y = df['Gaji_normalized']
       X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=42)
              X_train2,   X_test2,   y_train2,   y_test2   =
train_test_split(X,        df['Gaji'],        test_size=0.2,
random_state=42)
        # Mencari alpha terbaik untuk ridge
        # try:
        #     param_grid = {'alpha': np.logspace(-4, 4, 50)}
        #     ridge_reg = Ridge()
       #     grid_search = GridSearchCV(ridge_reg, param_grid,
cv=5)
        #     grid_search.fit(X_train, y_train)
                       #                best_alpha_ridge     =
grid_search.best_params_['alpha']
        # except Exception as e:
        #     best_alpha_ridge = 1.0
        # Mencari alpha terbaik untuk lasso
        # try:
        #     param_grid = {'alpha': np.logspace(-4, 4, 50)}
        #     lasso_reg = Lasso()
       #     grid_search = GridSearchCV(lasso_reg, param_grid,
cv=5)
        #     grid_search.fit(X_train, y_train)
                       #                best_alpha_lasso     =
grid_search.best_params_['alpha']
        # except Exception as e:
        #     best_alpha_lasso = 1.0
        # Mencari alpha terbaik untuk elastic net
        try:
            param_grid = {'alpha': np.logspace(-4, 4, 50)}
            enet_reg = ElasticNet()
             grid_search = GridSearchCV(enet_reg, param_grid,
cv=5)
            grid_search.fit(X_train, y_train)
          best_alpha_enet = grid_search.best_params_['alpha']
        except Exception as e:
            best_alpha_enet = 1.0
        model = ElasticNet(alpha=best_alpha_enet)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        real_data = y_pred * max_abs_value
       mape = np.mean(np.abs((y_test2 - real_data) / y_test2))
* 100
    else:
        mape = "Data tidak mencukupi"
    listMape.append(mape)
for i in listMape:
```

```
    print(i)
```

Segmen Program 4.6 Normalisasi Menggunakan *Z-Score*

```
import pandas as pd
from datetime import datetime
from sklearn.linear_model import LinearRegression, Ridge,
Lasso, ElasticNet
from sklearn.model_selection import train_test_split,
GridSearchCV
import numpy as np
from sklearn.preprocessing import StandardScaler

data = pd.read_excel('D:\Skripsi\coding\Data Gaji.xlsx')
reg = pd.read_excel('D:\Skripsi\coding\Data Inflasi.xlsx',
sheet_name="Inflasi Tahunan")
reg2 = pd.read_csv('D:\Skripsi\coding\Indonesian Salary by
Region (1997-2022).csv')
current_date_time = datetime.now()
current_year = current_date_time.year
daerah_filter = "INDONESIA"
filtered_reg = reg2[reg2['REGION'] == daerah_filter]
mapeArr = []
scaler = StandardScaler()
reg[['Inflasi_cont_normalized']] =
scaler.fit_transform(reg[['Inflasi Cont']])
filtered_reg[['Salary_normalized']] =
scaler.fit_transform(filtered_reg[['SALARY']])
program = data['Checked Program'].str.lower().unique()
for i in program:
      df = data[(data['Checked Program'].str.lower() ==
i.lower()) & (data['Gaji'] <= 20000000)]
    df = df.dropna(subset=['IPK', 'SKKK Total', 'Gaji'])
             df[['IPK_normalized', 'SKKK_normalized',
'Gaji_normalized']] = scaler.fit_transform(df[['IPK', 'SKKK
Total', 'Gaji']])
    mean = np.mean(df['Gaji'])
    std = np.std(df['Gaji'])
     df = pd.merge(df, reg, how='left', left_on='Tahun',
right_on='Tahun')
        df = pd.merge(df, filtered_reg, how='left',
left_on='Tahun', right_on='YEAR')
    if (len(df) > 1):
            X = df[['IPK_normalized', 'SKKK_normalized',
'Salary_normalized', 'Inflasi_cont_normalized']]
        y = df['Gaji_normalized']
       X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.1, random_state=42)
             X_train2, X_test2, y_train2, y_test2 =
train_test_split(X,        df['Gaji'],        test_size=0.1,
random_state=42)
        # Mencari alpha terbaik untuk ridge
        # try:
```

```
        #     param_grid = {'alpha': np.logspace(-4, 4, 50)}
        #     ridge_reg = Ridge()
       #     grid_search = GridSearchCV(ridge_reg, param_grid,
cv=5)
        #     grid_search.fit(X_train, y_train)
                      #                best_alpha_ridge     =
grid_search.best_params_['alpha']
        # except Exception as e:
        #     best_alpha_ridge = 1.0
        # Mencari alpha terbaik untuk lasso
        # try:
        #     param_grid = {'alpha': np.logspace(-4, 4, 50)}
        #     lasso_reg = Lasso()
       #     grid_search = GridSearchCV(lasso_reg, param_grid,
cv=5)
        #     grid_search.fit(X_train, y_train)
                      #                best_alpha_lasso     =
grid_search.best_params_['alpha']
        # except Exception as e:
        #     best_alpha_lasso = 1.0
        # Mencari alpha terbaik untuk elastic net
        try:
            param_grid = {'alpha': np.logspace(-4, 4, 50)}
            enet_reg = ElasticNet()
             grid_search = GridSearchCV(enet_reg, param_grid,
cv=5)
            grid_search.fit(X_train, y_train)
           best_alpha_enet = grid_search.best_params_['alpha']
        except Exception as e:
            best_alpha_enet = 1.0
                model   =   ElasticNet(alpha=best_alpha_enet,
l1_ratio=0.5)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        real_data = y_pred * std + mean

       mape = np.mean(np.abs((y_test2 - real_data) / y_test2))
* 100
    else:
        mape = "Data tidak mencukupi"
    mapeArr.append(mape)
for i in mapeArr:
    print(i)
```

4.5 Implementasi Evaluasi dan Seleksi

4.5.1 Mencari Nilai p, d, dan q Terbaik Untuk ARIMA

Sebelum melakukan prediksi menggunakan ARIMA, parameter p, d, dan q perlu dicari terlebih dahulu. Untuk mencari p, d, dan q terbaik, dilakukan pengujian menggunakan *grid search* untuk parameter (0,0,0) hingga (7,7,7). Percobaan tersebut dilakukan dengan berbagai kombinasi *data training-data testing* baik untuk data UMR, maupun data inflasi.

Segmen program 4.7 merupakan *code* untuk melakukan *grid search* pada data UMR, sementara segmen program 4.8 pada data inflasi.

Segmen Program 4.7 *Grid Search* Untuk Mencari Parameter Data UMR Terbaik

```
import pandas as pd
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_error

# Membaca dataset
df = pd.read_csv('D:\Skripsi\coding\Indonesian Salary by
Region (1997-2022).csv', sep=',')
df.set_index('YEAR', inplace=True)
regions = df['REGION'].unique()
llres = {"tahun": [], "data": []}
for y in range (2010, 2023):
    lres = {"pdq": [], "mae": [], "mape": []}
    for p in range(0, 8):
        for d in range (0, 8):
            for q in range (0, 8):
                res = {"daerah": [], "mae": [], "mape": []}
                for reg in regions:
                    data = df[df['REGION'] == reg]
                  train_data = data[data.index <= y]['SALARY']
                   test_data = data[data.index > y]['SALARY']
                    try:
                        model = ARIMA(train_data, order=(p, d,
q))
                         model_fit = model.fit()
                                                forecast =
model_fit.forecast(steps=len(test_data))

                          mae = mean_absolute_error(test_data,
forecast)
                         test_data = test_data.reset_index()
                         forecast = forecast.reset_index()
                                                    mape =
np.mean(np.abs((test_data['SALARY'] -
forecast['predicted_mean'])/test_data['SALARY'])) * 100
                    except:
                        mae = -1
                        mape = -1

                    # Menyimpan hasil MAE ke dalam dictionary
                    res["daerah"].append(reg)
                    res['mae'].append(mae)
                    res['mape'].append(mape)

                maeTot = 0
                mapeTot = 0
                tot = 0
```

```
                for i in range (len(res['daerah'])):
                    if (res["mae"][i] != -1):
                        maeTot += res["mae"][i]
                        mapeTot += res['mape'][i]
                        tot+=1
                lres['pdq'].append(str(p)+str(d)+str(q))
                lres['mae'].append(maeTot/tot)
                lres['mape'].append(mapeTot/tot)
    llres['tahun'].append(y)
    llres['data'].append(lres)
for i in range (0, len(llres['tahun'])):
    print(llres['tahun'][i])
    for j in range (0, len(lres['pdq'])):
        print(llres['data'][i]['mape'][j])
```

Segmen Program 4.8 *Grid Search* Untuk Mencari Parameter Data Inflasi Terbaik

```
import pandas as pd
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_error

df  =  pd.read_excel('D:\Skripsi\coding\Data  Inflasi.xlsx',
sheet_name="Inflasi Tahunan")
df.set_index('Tahun', inplace=True)
llres = {"tahun": [], "data": []}
for y in range (2010, 2023):
    lres = {"pdq": [], "mae": [], "mape": []}
    for p in range(0, 8):
        for d in range (0, 8):
            for q in range (0, 8):
                data = df
                 train_data = data[data.index <= y]['Inflasi
Cont']
                  test_data = data[data.index > y]['Inflasi
Cont']
                try:
                    model = ARIMA(train_data, order=(p,d,q))
                    model_fit = model.fit()
                                                forecast   =
model_fit.forecast(steps=len(test_data))

                        mae = mean_absolute_error(test_data,
forecast)
                    test_data = test_data.reset_index()
                    forecast = forecast.reset_index()
                    mape = np.mean(np.abs((test_data['Inflasi
Cont']   -   forecast['predicted_mean'])/test_data['Inflasi
Cont'])) * 100
                except:
                    mae = -1
                    mape = -1
                lres['pdq'].append(str(p)+str(d)+str(q))
```

```
                lres['mae'].append(mae)
                lres['mape'].append(mape)
    llres['tahun'].append(y)
    llres['data'].append(lres)
for i in range (0, len(llres['tahun'])):
    print(llres['tahun'][i])
    for j in range (0, len(lres['pdq'])):
        print(llres['data'][i]['mape'][j])
```

4.5.2 Menguji Variabel yang Akan Digunakan Untuk Prediksi

Sebelum dilakukan prediksi, variabel-variabel akan diuji dan diamati untuk kemudian dipergunakan sesuai dengan hasil dari pengamatan. Beberapa pengujian yang dilakukan adalah pengujian *error* antara program, prodi, dan fakultas seperti tampak pada segmen program 4.9, pengujian jenis-jenis SKKK seperti tampak pada segmen program 4.10, dan pengujian pertanyaan-pertanyaan opsional seperti tampak pada segmen program 4.11.

Segmen Program 4.9 Pengujian *Error* Program, Prodi, dan Fakultas

```
import pandas as pd
from datetime import datetime
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import numpy as np

data = pd.read_excel('D:\Skripsi\coding\Data Gaji.xlsx')
reg  =  pd.read_excel('D:\Skripsi\coding\Data  Inflasi.xlsx',
sheet_name="Inflasi Tahunan")
reg2  =  pd.read_csv('D:\Skripsi\coding\Indonesian  Salary  by
Region (1997-2022).csv')
current_date_time = datetime.now()
current_year = current_date_time.year
daerah_filter = "INDONESIA"
filtered_reg = reg2[reg2['REGION'] == daerah_filter]
# Saat ingin mengubah ke program atau prodi data['Fakultas']
tinggal diubah
daftar = data['Fakultas'].str.lower().unique()
for i in daftar:
     df = data[(data['Fakultas'].str.lower() == i.lower()) &
(data['Gaji'] <= 20000000)]
    df = df.dropna(subset=['IPK', 'SKKK Total', 'Gaji'])
      df  =  pd.merge(df,  reg,  how='left',  left_on='Tahun',
right_on='Tahun')
        df   =   pd.merge(df,   filtered_reg,   how='left',
left_on='Tahun', right_on='YEAR')
    if (len(df) > 1):
        # Saat ingin menambah atau mengurangi variabel hanya
tinggal mengubah X
```

```
        X = df[['IPK', 'SKKK Total']]
        y = df['Gaji']
        X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.1, random_state=42)
        model = LinearRegression()
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        mape = np.mean(np.abs((y_test - y_pred) / y_test)) *
100
    else:
        mape = "Data tidak mencukupi"
    print(mape)
```

Segmen Program 4.10 Pengujian Jenis-Jenis SKKK

```
import pandas as pd
from datetime import datetime
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import numpy as np

data = pd.read_excel('D:\Skripsi\coding\Data Gaji.xlsx')
reg = pd.read_excel('D:\Skripsi\coding\Data Inflasi.xlsx',
sheet_name="Inflasi Tahunan")
reg2 = pd.read_csv('D:\Skripsi\coding\Indonesian Salary by
Region (1997-2022).csv')
current_date_time = datetime.now()
current_year = current_date_time.year
daerah_filter = "INDONESIA"
filtered_reg = reg2[reg2['REGION'] == daerah_filter]
program = data['Prodi'].str.lower().unique()
for i in program:
    df = data[(data['Prodi'].str.lower() == i.lower()) &
(data['Gaji'] <= 20000000)]
    df = df.dropna(subset=['IPK', 'SKKK Total', 'Gaji',
'ORPIM', 'PENALARAN', 'BAKMI', 'PENGMAS'])
    df = pd.merge(df, reg, how='left', left_on='Tahun',
right_on='Tahun')
    df = pd.merge(df, filtered_reg, how='left',
left_on='Tahun', right_on='YEAR')
    if (len(df) > 1):
        # Saat ingin mengubah jenis SKKK, X tinggal diubah
        X = df[['PENGMAS']]
        y = df['Gaji']
        X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=42)
        model = LinearRegression()
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        mape = np.mean(np.abs((y_test - y_pred) / y_test)) *
100
    else:
        mape = "Data tidak mencukupi"
```

```
    print(mape)
```

Segmen Program 4.11 Pengujian Pertanyaan-Pertanyaan Opsional

```
import pandas as pd
from datetime import datetime
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import numpy as np

data = pd.read_excel('D:\Skripsi\coding\Data Gaji.xlsx')
reg  =  pd.read_excel('D:\Skripsi\coding\Data  Inflasi.xlsx',
sheet_name="Inflasi Tahunan")
reg2  =  pd.read_csv('D:\Skripsi\coding\Indonesian  Salary  by
Region (1997-2022).csv')
current_date_time = datetime.now()
current_year = current_date_time.year
daerah_filter = "INDONESIA"
filtered_reg = reg2[reg2['REGION'] == daerah_filter]
program = data['Checked Program'].str.lower().unique()
data  =  pd.merge(data,  reg,  how='left',  left_on='Tahun',
right_on='Tahun')
data    =    pd.merge(data,    filtered_reg,    how='left',
left_on='Tahun', right_on='YEAR')
count = 4
for i in program:
    df   =   data[(data['Checked   Program'].str.lower()   ==
i.lower()) & (data['Gaji'] <= 20000000)]
    df   =   df.dropna(subset=['IPK',   'SKKK   Total','Gaji',
f'Q{count}'])
    totalMape = []
    jawaban = df[f'Q{count}'].unique()
    for j in jawaban:
        df2 = df[(df[f'Q{count}'] == j)]
        if (len(df2) > 1):
            X = df2[['IPK','SKKK Total', 'SALARY', 'Inflasi
Cont']]
            y = df2['Gaji']
            X_train,    X_test,    y_train,    y_test    =
train_test_split(X, y, test_size=0.2, random_state=42)
            model = LinearRegression()
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)
            mape = np.mean(np.abs((y_test - y_pred) / y_test))
* 100
            totalMape.append(mape)
        else:
            mae = "Data tidak mencukupi"
            mape = "Data tidak mencukupi"
    print(np.mean(totalMape))
```

4.5.3 Menguji Metode Regresi Linear yang Akan Digunakan

Selain dilakukan pengujian variabel, sebelum melakukan prediksi juga dilakukan pengujian regularisasi regresi linear. Pada skripsi ini, ada 4 metode regularisasi regresi linear yang digunakan yaitu regresi linear biasa, Ridge, LASSO, dan Elastic Net. *Code* untuk pengecekan *error* regresi linear biasa dapat dilihat pada segmen program 4.12, untuk Ridge pada segmen program 4.13, untuk LASSO pada segmen program 4.14, dan untuk Elastic Net pada segmen program 4.15.

Segmen Program 4.12 Pengujian *Error* Untuk Regresi Linear Biasa

```
import pandas as pd
import numpy as np
from connect import connect_to_database
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

connection = connect_to_database()
query = "SELECT * FROM data_mahasiswa d JOIN program p ON
d.program = p.id_program ORDER BY kode_unik"
data = pd.read_sql_query(query, connection)
query = "SELECT * FROM umr"
data_umr = pd.read_sql_query(query, connection)
daerah = "INDONESIA"
data_umr = data_umr[data_umr['daerah_umr'].str.lower() ==
daerah.lower()]
data = pd.merge(data, data_umr, how='left', left_on='tahun',
right_on='tahun_umr')
query = "SELECT * FROM inflasi"
data_inflasi = pd.read_sql_query(query, connection)
data = pd.merge(data, data_inflasi, how='left',
left_on='tahun', right_on='tahun_inflasi')
program = data['nama_program'].str.lower().unique()
hasil = []
for i in program:
    df = data[(data['nama_program'].str.lower() == i.lower())]
    if len(df) > 1:
        X = df[['ipk', 'skkk_total', 'nominal_umr',
'inflasi_kontinu']]
        y = df['gaji']
        X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=42)
        linear_reg = LinearRegression()
        linear_reg.fit(X_train, y_train)
        y_pred_linear_reg = linear_reg.predict(X_test)
        mape = np.mean(np.abs((y_test - y_pred_linear_reg) /
y_test)) * 100
        hasil.append(mape)
    else:
```

```
        hasil.append("Data tidak mencukupi")
for i in hasil:
    print(i)
```

Segmen Program 4.13 Pengujian *Error* Untuk Regresi Linear Ridge

```
import pandas as pd
import numpy as np
from connect import connect_to_database
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from    sklearn.model_selection    import    train_test_split,
GridSearchCV

connection = connect_to_database()
query = "SELECT * FROM data_mahasiswa d JOIN program p ON
d.program = p.id_program ORDER BY kode_unik"
data = pd.read_sql_query(query, connection)
query = "SELECT * FROM umr"
data_umr = pd.read_sql_query(query, connection)
daerah = "INDONESIA"
data_umr  =  data_umr[data_umr['daerah_umr'].str.lower()  ==
daerah.lower()]
data = pd.merge(data, data_umr, how='left', left_on='tahun',
right_on='tahun_umr')
query = "SELECT * FROM inflasi"
data_inflasi = pd.read_sql_query(query, connection)
data    =    pd.merge(data,    data_inflasi,    how='left',
left_on='tahun', right_on='tahun_inflasi')
program = data['nama_program'].str.lower().unique()
hasil = []
for i in program:
    df = data[(data['nama_program'].str.lower() == i.lower())]
    if len(df) > 1:
        X   =   df[['ipk',   'skkk_total',   'nominal_umr',
'inflasi_kontinu']]
        y = df['gaji']
        X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=42)
        try:
            param_grid = {'alpha': np.logspace(-4, 4, 50)}
            ridge_reg = Ridge()
            grid_search = GridSearchCV(ridge_reg, param_grid,
cv=5)
            grid_search.fit(X_train, y_train)
            best_alpha = grid_search.best_params_['alpha']
        except Exception as e:
            best_alpha = 1.0
        ridge_reg.fit(X_train, y_train)
        y_pred_ridge_reg = ridge_reg.predict(X_test)
        mape = np.mean(np.abs((y_test - y_pred_ridge_reg) /
y_test)) * 100
        hasil.append(mape)
```

```
    else:
        hasil.append("Data tidak mencukupi")
for i in hasil:
    print(i)
```

Segmen Program 4.14 Pengujian *Error* Untuk Regresi Linear LASSO

```
import pandas as pd
import numpy as np
from connect import connect_to_database
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from    sklearn.model_selection    import    train_test_split,
GridSearchCV

connection = connect_to_database()
query = "SELECT * FROM data_mahasiswa d JOIN program p ON
d.program = p.id_program ORDER BY kode_unik"
data = pd.read_sql_query(query, connection)
query = "SELECT * FROM umr"
data_umr = pd.read_sql_query(query, connection)
daerah = "INDONESIA"
data_umr  =  data_umr[data_umr['daerah_umr'].str.lower()  ==
daerah.lower()]
data = pd.merge(data, data_umr, how='left', left_on='tahun',
right_on='tahun_umr')
query = "SELECT * FROM inflasi"
data_inflasi = pd.read_sql_query(query, connection)
data    =    pd.merge(data,    data_inflasi,    how='left',
left_on='tahun', right_on='tahun_inflasi')
program = data['nama_program'].str.lower().unique()
hasil = []
for i in program:
    df = data[(data['nama_program'].str.lower() == i.lower())]
    if len(df) > 1:
        X   =   df[['ipk',   'skkk_total',   'nominal_umr',
'inflasi_kontinu']]
        y = df['gaji']
        X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=42)
        try:
            param_grid = {'alpha': np.logspace(-4, 4, 50)}
            ridge_reg = Lasso()
            grid_search = GridSearchCV(ridge_reg, param_grid,
cv=5)
            grid_search.fit(X_train, y_train)
            best_alpha = grid_search.best_params_['alpha']
        except Exception as e:
            best_alpha = 1.0
        lasso_reg = Lasso(alpha=best_alpha)
        lasso_reg.fit(X_train, y_train)
        y_pred_lasso_reg = lasso_reg.predict(X_test)
```

```
        mape = np.mean(np.abs((y_test - y_pred_lasso_reg) /
y_test)) * 100
        hasil.append(mape)
    else:
        hasil.append("Data tidak mencukupi")
for i in hasil:
    print(i)
```

Segmen Program 4.15 Pengujian *Error* Untuk Regresi Linear Elastic Net

```
import pandas as pd
import numpy as np
from connect import connect_to_database
from sklearn.model_selection import train_test_split
from sklearn.linear_model import ElasticNet
from    sklearn.model_selection    import    train_test_split,
GridSearchCV

connection = connect_to_database()
query = "SELECT * FROM data_mahasiswa d JOIN program p ON
d.program = p.id_program ORDER BY kode_unik"
data = pd.read_sql_query(query, connection)
query = "SELECT * FROM umr"
data_umr = pd.read_sql_query(query, connection)
daerah = "INDONESIA"
data_umr  =  data_umr[data_umr['daerah_umr'].str.lower()  ==
daerah.lower()]
data = pd.merge(data, data_umr, how='left', left_on='tahun',
right_on='tahun_umr')
query = "SELECT * FROM inflasi"
data_inflasi = pd.read_sql_query(query, connection)
data    =    pd.merge(data,    data_inflasi,    how='left',
left_on='tahun', right_on='tahun_inflasi')
program = data['nama_program'].str.lower().unique()
hasil = []
for i in program:
    df = data[(data['nama_program'].str.lower() == i.lower())]
    if len(df) > 1:
        X   =   df[['ipk',   'skkk_total',   'nominal_umr',
'inflasi_kontinu']]
        y = df['gaji']
        X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=42)
        try:
            param_grid = {'alpha': np.logspace(-4, 4, 50)}
            ridge_reg = ElasticNet()
            grid_search = GridSearchCV(ridge_reg, param_grid,
cv=5)
            grid_search.fit(X_train, y_train)
            best_alpha = grid_search.best_params_['alpha']
        except Exception as e:
            best_alpha = 1.0
```

```
        elastic_net_reg     =     ElasticNet(alpha=best_alpha,
l1_ratio=0.5)
        elastic_net_reg.fit(X_train, y_train)
        y_pred_elastic_net_reg                                  =
elastic_net_reg.predict(X_test)
        mape        =        np.mean(np.abs((y_test        -
y_pred_elastic_net_reg) / y_test)) * 100
        hasil.append(mape)
    else:
        hasil.append("Data tidak mencukupi")
for i in hasil:
    print(i)
```

4.6 Implementasi Aplikasi

4.6.1 Aplikasi Prediksi Gaji Mahasiswa

Dalam membuat aplikasi prediksi gaji, hal pertama yang perlu dibuat adalah *user interface* sebagai media penghubung *user* dan sistem. *Code* untuk *user interface* beserta beberapa fiturnya seperti input data baru, filter data yang dipilih, terdapat pada segmen program 4.16. Di tengahnya, koneksi antar *database* dan sistem perlu dibuat seperti tampak pada segmen program 4.17. Sementara itu, untuk proses prediksi dimulai dari pengecekan kombinasi variabel terbaik seperti tampak pada segmen program 4.18, kemudian dilanjutkan dengan mencari regularisasi regresi linear seperti tampak pada segmen program 4.19, dan yang terakhir adalah proses prediksi gaji itu sendiri seperti tampak pada segmen program 4.20.

Segmen Program 4.16 *User Interface* Aplikasi Prediksi Gaji Mahasiswa

```
import streamlit as st
import pandas as pd
from streamlit_option_menu import option_menu
import pyodbc
import pandas as pd
import matplotlib.pyplot as plt
from connect import connect_to_database
from datetime import datetime
from cekVariabelPrediksi import checkUMR
from prediksiGaji import prediksiGaji

connection = connect_to_database()
cursor = connection.cursor()
query = "SELECT * FROM pertanyaan"
df = pd.read_sql_query(query, connection)
query = "SELECT q4, program FROM data_mahasiswa WHERE
konfirmasi = 1"
daftar_perusahaan = pd.read_sql_query(query, connection)
query = "SELECT DISTINCT(provinsi) FROM daerah"
```

```
provinsi_list = pd.read_sql_query(query, connection)
query = "SELECT DISTINCT(nama_fakultas) FROM program"
fakultas_list = pd.read_sql_query(query, connection)
query = "SELECT * FROM data_mahasiswa d JOIN program p ON
d.program = p.id_program JOIN daerah da ON d.daerah =
da.id_daerah WHERE konfirmasi = 1"
data = pd.read_sql_query(query, connection)
query = "SELECT * FROM inflasi WHERE konfirmasi = 1 ORDER BY
tahun_inflasi"
data_inflasi = pd.read_sql_query(query, connection)
kode_program = {'Business Accounting': 'A', 'International
Business Accounting': 'B', 'Tax Accounting': 'C',
'Arsitektur': 'D', 'Bahasa Mandarin': 'E',
'Desain Interior': 'F', 'Desain Komunikasi Visual': 'G',
'International Program In Digital Media': 'H', 'Ilmu
Komunikasi': 'J', 'Business Information System': 'K',
'Informatika': 'L', 'Business Management': 'M', 'Creative
Tourism': 'N', 'Finance And Investment': 'O', 'Hotel
Management': 'P',
'International Business Management': 'Q', 'Marketing
Management': 'R', 'Pendidikan Guru - Sekolah Dasar': 'S',
'English For Business': 'T', 'English For Creative Industry':
'U',
'Teknik Elektro': 'V', 'Teknik Industri': 'X', 'Automotive':
'Y', 'Teknik Mesin': 'Z', 'Teknik Sipil': 'AA',
'International Business Engineering': 'W', 'Textile And
Fashion Design': 'I'}
urutanQ = {'Business Accounting': [1,7,10,6,5,11,9,2,8,3,4],
'International Business Accounting':
[11,6,2,8,3,7,10,4,9,1,5], 'Tax Accounting':
[8,2,1,6,7,4,5,10,3,11,9], 'Arsitektur':
[10,1,7,5,8,6,9,4,3,2,11], 'Bahasa Mandarin':
[6,3,7,9,8,1,2,5,11,10,4],
'Desain Interior': [1,7,3,5,8,2,11,6,9,10,4], 'Desain
Komunikasi Visual': [2,4,3,11,5,10,8,1,9,7,6], 'International
Program In Digital Media': [11,2,10,3,9,5,1,4,6,7,8], 'Ilmu
Komunikasi': [10,2,8,9,6,7,1,11,5,3,4], 'Business Information
System': [9,6,5,1,2,3,10,7,11,4,8],
'Informatika': [7,5,2,8,9,4,6,3,11,1,10], 'Business
Management': [2,6,1,7,3,10,5,11,9,8,4], 'Creative Tourism':
[1,4,2,10,3,6,11,8,7,9,5], 'Finance And Investment':
[2,11,8,10,9,6,5,7,3,4,1], 'Hotel Management':
[5,7,9,1,2,8,6,11,4,3,10],
'International Business Management':
[4,1,3,8,10,2,6,11,9,7,5], 'Marketing Management':
[1,11,2,10,3,9,6,5,8,7,4], 'Pendidikan Guru - Sekolah Dasar':
[9,10,7,8,2,6,5,3,11,1,4], 'English For Business':
[2,1,7,3,6,10,8,9,11,5,4], 'English For Creative Industry':
[4,1,6,11,10,7,9,3,2,8,5],
'Teknik Elektro': [3,7,1,11,6,8,4,5,10,9,2], 'Teknik
Industri': [5,2,7,10,9,6,8,3,1,11,4], 'Automotive':
[5,11,7,8,9,10,3,2,1,4,6], 'Teknik Mesin':
```

```
[8,11,9,3,10,2,5,1,7,6,4], 'Teknik Sipil':
[1,6,4,2,8,9,11,5,3,10,7],
'International Business Engineering':
[1,10,3,6,9,7,5,8,11,2,4], 'Textile And Fashion Design':
[8,11,2,9,1,6,7,10,3,4,5]}
urutanSKKK = {'Business Accounting': ['BAKMI', 'PENALARAN',
'ORPIM', 'PENGMAS'], 'International Business Accounting':
['ORPIM', 'PENGMAS', 'BAKMI', 'PENALARAN'], 'Tax Accounting':
['PENGMAS', 'ORPIM', 'BAKMI', 'PENALARAN'], 'Arsitektur':
['BAKMI', 'PENALARAN', 'ORPIM', 'PENGMAS'], 'Bahasa Mandarin':
['BAKMI', 'PENGMAS', 'ORPIM', 'PENALARAN'],
'Desain Interior': ['PENGMAS', 'BAKMI', 'PENALARAN', 'ORPIM'],
'Desain Komunikasi Visual': ['BAKMI', 'PENGMAS', 'PENALARAN',
'ORPIM'], 'International Program In Digital Media':
['PENALARAN', 'BAKMI', 'ORPIM', 'PENGMAS'], 'Ilmu Komunikasi':
['ORPIM', 'PENGMAS', 'PENALARAN', 'BAKMI'], 'Business
Information System': ['PENGMAS', 'PENALARAN', 'ORPIM',
'BAKMI'],
'Informatika': ['PENGMAS', 'PENALARAN', 'BAKMI', 'ORPIM'],
'Business Management': ['ORPIM', 'PENGMAS', 'BAKMI',
'PENALARAN'], 'Creative Tourism': ['BAKMI', 'PENGMAS',
'PENALARAN', 'ORPIM'], 'Finance And Investment': ['ORPIM',
'PENALARAN', 'PENGMAS', 'BAKMI'], 'Hotel Management':
['PENALARAN', 'ORPIM', 'BAKMI', 'PENGMAS'],
'International Business Management': ['ORPIM', 'PENGMAS',
'BAKMI', 'PENALARAN'], 'Marketing Management': ['PENGMAS',
'BAKMI', 'PENALARAN', 'ORPIM'], 'Pendidikan Guru - Sekolah
Dasar': ['PENGMAS', 'BAKMI', 'PENALARAN', 'ORPIM'], 'English
For Business': ['ORPIM', 'PENGMAS', 'BAKMI', 'PENALARAN'],
'English For Creative Industry': ['ORPIM', 'PENGMAS',
'PENALARAN', 'BAKMI'],
'Teknik Elektro': ['PENALARAN', 'ORPIM', 'BAKMI', 'PENGMAS'],
'Teknik Industri': ['PENGMAS', 'ORPIM', 'PENALARAN', 'BAKMI'],
'Automotive': ['ORPIM', 'BAKMI', 'PENGMAS', 'PENALARAN'],
'Teknik Mesin': ['BAKMI', 'PENGMAS', 'ORPIM', 'PENALARAN'],
'Teknik Sipil': ['PENALARAN', 'BAKMI', 'ORPIM', 'PENGMAS'],
'International Business Engineering': ['PENALARAN', 'PENGMAS',
'BAKMI', 'ORPIM'], 'Textile And Fashion Design': ['BAKMI',
'ORPIM', 'PENGMAS', 'PENALARAN']}

def get_kota(provinsi):
    query = "SELECT nama_kota_kabupaten FROM daerah WHERE
provinsi = '" + provinsi + "'"
    kota_list = pd.read_sql_query(query, connection)
    return kota_list['nama_kota_kabupaten']

def get_prodi(fakultas):
    query = "SELECT DISTINCT(nama_prodi) FROM program WHERE
lower(nama_fakultas) = lower('" + fakultas + "')"
    prodi_list = pd.read_sql_query(query, connection)
    return prodi_list['nama_prodi']

def get_program(prodi):
```

```
    query = "SELECT DISTINCT(nama_program) FROM program WHERE
lower(nama_prodi) = lower('" + prodi + "')"
    program_list = pd.read_sql_query(query, connection)
    return program_list['nama_program']

def get_umr(daerah):
    query = "SELECT * FROM umr WHERE upper(daerah_umr) =
upper('"+daerah+"') AND konfirmasi = 1 ORDER BY tahun_umr"
    data_umr = pd.read_sql_query(query, connection)
    return data_umr

def show_page_1():
    jawaban_pertanyaan = {}
    nrp = st.text_input("NRP:")
    thn = st.text_input("Tahun Lulus:")
    colFakultas, colProdi, colProgram = st.columns(3)

    with colFakultas:
        selected_fakultas     =     st.selectbox('Fakultas:',
fakultas_list['nama_fakultas'])
    with colProdi:
        selected_prodi        =        st.selectbox('Prodi:',
get_prodi(selected_fakultas))
    with colProgram:
        selected_program      =      st.selectbox('Program:',
get_program(selected_prodi))

    colIPK, colSKKK = st.columns(2)
    with colIPK:
        ipk = st.number_input("IPK:", step=0.1, format="%.1f")

    with colSKKK:
        skkk     =     st.number_input("SKKK:",     step=0.1,
format="%.1f")

    colOrpim,    colPenalaran,    colBakmi,    colPengmas    =
st.columns(4)
    with colOrpim:
        orpim    =    st.number_input("ORPIM:",    step=0.1,
format="%.1f")
    with colPenalaran:
        penalaran  =  st.number_input("Penalaran:",  step=0.1,
format="%.1f")
    with colBakmi:
        bakmi    =    st.number_input("BAKMI:",    step=0.1,
format="%.1f")
    with colPengmas:
        pengmas   =   st.number_input("PENGMAS:",   step=0.1,
format="%.1f")

    gaji = st.number_input("Gaji 1 Tahun Setelah Lulus:")

    colProv, colKota = st.columns(2)
```

```
    with colProv:
        selected_prov    =    st.selectbox('Provinsi    Tempat
Bekerja:', provinsi_list['provinsi'], index=5)
    with colKota:
        selected_kota=  st.selectbox('Kota  Tempat  Bekerja:',
get_kota(selected_prov))

    for  index,  value  in  enumerate(df['pertanyaan_alumni'],
start=1):
        query2 = "SELECT * FROM pilihan WHERE id_pertanyaan =
" + str(df['id_pertanyaan'][index-1])
        pilihan = pd.read_sql_query(query2, connection)
        if (len(pilihan["detail_pilihan"]) != 0) :
            pil = pilihan['detail_pilihan']
            user_input     =     st.selectbox(value,     pil,
key=f"pertanyaan_{index}")
        else:
            user_input         =         st.text_input(value,
key=f"pertanyaan_{index}")
        jawaban_pertanyaan[f"pertanyaan_{index}"]           =
user_input
    if st.button("Submit"):
        if (len(nrp) != 9 or thn == ""):
            st.error("Silakan isi NRP dan tahun kelulusan")
        else:
            query  =  f"SELECT  id_program  FROM  program  WHERE
upper(nama_program) = upper('{selected_program}')"
            dt_program = pd.read_sql_query(query, connection)
            query  =  f"SELECT  id_daerah  FROM  daerah  WHERE
upper(nama_kota_kabupaten) = upper('{selected_kota}')"
            dt_daerah = pd.read_sql_query(query, connection)
            query   =   f"SELECT   count(*)   as   jumlah   from
data_mahasiswa WHERE tahun = {thn}"
            no_urut = pd.read_sql_query(query, connection)
            kode_unik        =        str(thn[2:4])        +
kode_program[selected_program]                                +
str('{:04d}'.format(no_urut['jumlah'].iloc[0]+1))
            query   =   f"INSERT   INTO   data_mahasiswa(ipk,
kode_unik, skkk_total, skkk_orpim, skkk_penalaran, skkk_bakmi,
skkk_pengmas, gaji, tahun, q1, q2, q3, q4, q5, q6, q7, q8, q9,
q10,  q11,  program,  daerah,  konfirmasi)  VALUES  ({ipk},
'{kode_unik}',  {skkk},  {orpim},  {penalaran},  {bakmi},
{pengmas}, {gaji}, {thn}"
            for i in range (1,12):
                jwb = jawaban_pertanyaan[f"pertanyaan_{i}"]
                query += f", '{jwb}'"
            query += f", {dt_program['id_program'].iloc[0]},
{dt_daerah['id_daerah'].iloc[0]}, 0);"
            cursor.execute(query)
            connection.commit()
            st.write("*** Data berhasil disubmit ***")
def show_page_2():
    def plot_umr(selected_prov):
```

```
        plt.figure(figsize=(10, 6))
        data_umr = get_umr(selected_prov)
        plt.plot(data_umr['tahun_umr'],
data_umr['nominal_umr'], marker='o', color='green')
        plt.title(f'Grafik Upah Minimum Regional (UMR)
{selected_prov}')
        plt.xlabel('Tahun')
        plt.ylabel('UMR (IDR)')
        plt.xticks(data_umr['tahun_umr'], rotation=45)
        plt.grid(True)
        st.pyplot(plt)

    new_data = pd.Series(["INDONESIA"], index=[0])
    pl = pd.concat([new_data, provinsi_list['provinsi']])
    selected_prov = st.selectbox('Provinsi Tempat Bekerja:',
pl)
    plot_umr(selected_prov)
    thn = st.text_input("Tahun:")
    umr = st.number_input("UMR:")

    if st.button("Submit"):
        if (thn == ""):
            st.error("Silakan isi data tahun")
        else:
            query = f"INSERT INTO public.umr(daerah_umr,
nominal_umr, tahun_umr, konfirmasi) VALUES ('{selected_prov}',
{umr}, {thn}, 0);"
            cursor.execute(query)
            connection.commit()
            st.write("*** Data berhasil disubmit ***")

def show_page_3():
    plt.figure(figsize=(10, 6))
    plt.plot(data_inflasi['tahun_inflasi'],
data_inflasi['inflasi_kontinu'], marker='o', color='orange')
    plt.title('Grafik Inflasi Indonesia')
    plt.xlabel('Tahun')
    plt.ylabel('Inflasi (cont)')
    plt.xticks(data_inflasi['tahun_inflasi'], rotation=45)
    plt.grid(True)

    st.pyplot(plt)
    thn = st.text_input("Tahun:")
    inf = st.number_input("Inflasi:")
    if st.button("Submit"):
        if (thn == ""):
            st.error("Silakan isi data tahun")
        else:
            thn = int(thn)
            query = f"SELECT inflasi_kontinu FROM inflasi
WHERE tahun_inflasi = {thn-1} AND konfirmasi = 1"
            tahun_prev = pd.read_sql_query(query, connection)
            if (len(tahun_prev) <= 0):
```

```
                st.error(f"Data untuk tahun {thn-1} belum
tersedia, silakan menunggu atau input data pada tahun
tersebut.")
            else:
                data_prev                                    =
int(tahun_prev['inflasi_kontinu'].iloc[0])
                query = f"INSERT INTO inflasi(tahun_inflasi,
inflasi, inflasi_kontinu, konfirmasi) VALUES ({thn}, {inf},
{(100+inf)/100*data_prev}, 0);"
                cursor.execute(query)
                connection.commit()
                st.write("*** Data berhasil disubmit ***")

with st.sidebar :
    selected = option_menu (
            'Selamat   Datang',   ['Saya   Ingin   Melakukan
Prediksi','Saya Ingin Berkontribusi Data'], default_index=0
    )
st.markdown(
    """
    <style>
    .stButton>button {
        width: 100%;
    }
    </style>
    """,
    unsafe_allow_html=True
)
if(selected =='Saya Ingin Melakukan Prediksi') :
    jawaban_pertanyaan = {}
    st.title("Saya Ingin Melakukan Prediksi")

    tahun_sekarang = datetime.now().year
    tahun = [str(tahun) for tahun in range(tahun_sekarang,
tahun_sekarang + 4)]
    selected_tahun = st.selectbox('Tahun:', tahun)
    # INISIALISASI JAWABAN
    selected_prov = "-"
    selected_kota = "-"
    for i in range (1, 12):
        jawaban_pertanyaan[f"pertanyaan_{i}"] = "-"
    orpim = 0
    penalaran = 0
    bakmi = 0
    pengmas = 0

    colFakultas, colProdi, colProgram = st.columns(3)

    with colFakultas:
        selected_fakultas     =     st.selectbox('Fakultas:',
fakultas_list['nama_fakultas'])
    with colProdi:
        prodi = ["Informatika", "Option 2", "Option 3"]
```

```
        selected_prodi        =        st.selectbox('Prodi:',
get_prodi(selected_fakultas))

    with colProgram:
        program = ["Sistem Informasi Bisnis", "Option 2",
"Option 3"]
        selected_program      =      st.selectbox('Program:',
get_program(selected_prodi))

    colIPK, colSKKK = st.columns(2)

    with colIPK:
        ipk = st.number_input("IPK:", step=0.1, format="%.1f")

    with colSKKK:
        skkk     =     st.number_input("SKKK:",     step=0.1,
format="%.1f")

    # Create a session state to keep track of button click
state
    if 'button_clicked' not in st.session_state:
        st.session_state.button_clicked = False

    if st.button("Detail SKKK"):
        # Toggle button clicked state
        st.session_state.button_clicked         =         not
st.session_state.button_clicked

    if st.session_state.button_clicked:
        columns                                              =
st.columns(len(urutanSKKK[selected_program]))
        col_map    =    dict(zip(urutanSKKK[selected_program],
columns))
        colOrpim = col_map['ORPIM']
        colPenalaran = col_map['PENALARAN']
        colPengmas = col_map['PENGMAS']
        colBakmi = col_map['BAKMI']
        with colOrpim:
            orpim   =   st.number_input("ORPIM:",   step=0.1,
format="%.1f")
        with colPenalaran:
            penalaran     =     st.number_input("Penalaran:",
step=0.1, format="%.1f")
        with colBakmi:
            bakmi   =   st.number_input("BAKMI:",   step=0.1,
format="%.1f")
        with colPengmas:
            pengmas  =  st.number_input("PENGMAS:",  step=0.1,
format="%.1f")

    # Create a session state to keep track of button click
state
    if 'button_clicked2' not in st.session_state:
```

```
        st.session_state.button_clicked2 = False

    if st.button("Detail Pekerjaan"):
        # Toggle button clicked state
        st.session_state.button_clicked2        =        not
st.session_state.button_clicked2

    if st.session_state.button_clicked2:
        colProv, colKota = st.columns(2)
        with colProv:
            pl = provinsi_list['provinsi']
            new_data = pd.Series(["-"], index=[0])
            pl = pd.concat([new_data, pl])
            selected_prov  =  st.selectbox('Provinsi  Tempat
Ingin Bekerja:', pl)
        with colKota:
            kl = get_kota(selected_prov)
            new_data = pd.Series(["-"], index=[0])
            kl = pd.concat([new_data, kl])
            selected_kota=  st.selectbox('Kota  Tempat  Ingin
Bekerja:', kl)
        for index in urutanQ[selected_program]:
            query2   =   "SELECT   *   FROM   pilihan   WHERE
id_pertanyaan = " + str(df['id_pertanyaan'][index-1])
            pilihan = pd.read_sql_query(query2, connection)
            new_data = pd.Series(["-"], index=[0])
            if (index != 4) :
                pil = pilihan['detail_pilihan']
                pil = pd.concat([new_data, pil])
                user_input                                    =
st.selectbox(df['pertanyaan_mhs'][index-1],              pil,
key=f"pertanyaan_{index}")
            else:
                query2 = "SELECT id_program FROM program WHERE
nama_program = '" + selected_program + "'"
                kd_program    =    pd.read_sql_query(query2,
connection)
                pil                                           =
daftar_perusahaan[(daftar_perusahaan['program']            ==
kd_program['id_program'].iloc[0])]
                pil = pil['q4']
                pil = pd.concat([new_data, pil])
                user_input                                    =
st.selectbox(df['pertanyaan_mhs'][index-1],     pil.unique(),
key=f"pertanyaan_{index}")
            jawaban_pertanyaan[f"pertanyaan_{index}"]       =
user_input
    checkbox_cek = st.checkbox('Gunakan Metode Terbaik (akan
memakan waktu lebih lama)')


    if st.button("Submit"):
        # Proses pemfilteran data berdasarkan jurusan
```

```
        data_filtered                                          =
data[(data['nama_program'].str.lower()                        ==
selected_program.lower())]
        # Jika difilter berdasarkan program membuat data tidak
mencukupi
        if (len(data_filtered['id_data_gaji']) < 2):
            data_filtered                                      =
data[(data['nama_prodi'].str.lower()                          ==
selected_prodi.lower())]
            # Jika difilter berdasarkan prodi masih membuat
data tidak mencukupi
            if (len(data_filtered['id_data_gaji']) < 2):
                data_filtered                                  =
data[(data['nama_fakultas'].str.lower()                       ==
selected_fakultas.lower())]
                if (len(data_filtered['id_data_gaji']) < 2):
                    st.write("Data tidak mencukupi")
                    st.stop()

        # Proses pemfilteran data berdasarkan kota (opsional)
        if (selected_prov != "-"):
            data_filtered_daerah                               =
data_filtered[(data_filtered['provinsi'].str.lower()          ==
selected_prov.lower())]
            if  (len(data_filtered_daerah['id_data_gaji'])  >=
2):
                data_filtered = data_filtered_daerah
                if (selected_kota != "-"):
                    data_filtered_daerah                       =
data_filtered[(data_filtered['nama_kota_kabupaten'].str.lower
() == selected_kota.lower())]
                    if
(len(data_filtered_daerah['id_data_gaji']) >= 2):
                        data_filtered = data_filtered_daerah
                    else:
                        st.write("---       Filter       tidak
menggunakan Data Kota/Kabupaten")
            else:
                st.write("--- Filter tidak menggunakan Data
Provinsi")
                st.write("--- Filter tidak menggunakan Data
Kota/Kabupaten")

        #   Proses   pemfilteran   data   berdasarkan   Q1-Q11
(opsional)
        for i in urutanQ[selected_program]:
            jwb = jawaban_pertanyaan[f"pertanyaan_{i}"]
            if (jwb != "-"):
                data_filtered_pertanyaan                       =
data_filtered[(data_filtered[f"q{i}"] == jwb)]
                if
(len(data_filtered_pertanyaan['id_data_gaji']) >= 2):
                    data_filtered = data_filtered_pertanyaan
```

```
                else:
                    st.write(f"--- Filter tidak menggunakan
Data Pertanyaan Q{i}")

        # st.write(data_filtered)
        hasilMape = checkUMR(data_filtered, selected_prov)
        minMape= min(hasilMape)
        kode = 0
        if (hasilMape[0] == minMape):
            kode = 1
        elif (hasilMape[1] == minMape):
            kode = 2
        elif (hasilMape[2] == minMape):
            kode = 3
        elif (hasilMape[3] == minMape):
            kode = 4
        daftar_tampilan = ['ipk', 'skkk_total', 'gaji']
        data_prediksi = [ipk, skkk]
        if (orpim != 0):
            daftar_tampilan.append('skkk_orpim')
            data_prediksi.append(orpim)
        if (penalaran != 0):
            daftar_tampilan.append('skkk_penalaran')
            data_prediksi.append(penalaran)
        if (bakmi != 0):
            daftar_tampilan.append('skkk_bakmi')
            data_prediksi.append(bakmi)
        if (pengmas != 0):
            daftar_tampilan.append('skkk_pengmas')
            data_prediksi.append(pengmas)
        prediksiGaji(data_filtered,           selected_tahun,
selected_prov,    kode,    daftar_tampilan,    data_prediksi,
checkbox_cek)
        st.write("""
        ## Petunjuk
        1. Untuk meningkatkan keakuratan, silakan coba isi
lebih banyak kolom input
        2. Input detail pekerjaan dan detail SKKK telah
diurutkan berdasarkan pengaruh terbaik""")
elif(selected =='Saya Ingin Berkontribusi Data') :
    jawaban_pertanyaan = {}
    st.title("Saya Ingin Berkontribusi Data")
    st.write()
    st.write("Data Apa yang Ingin Anda Kontribusikan?")
    page = st.session_state.get("page", "Gaji")
    colGaji, colUMR, colInflasi = st.columns(3)
    with colGaji:
        # Tombol untuk menavigasi ke Halaman 1
        if st.button("Gaji"):
            page = "Halaman Gaji"
    with colUMR:
        # Tombol untuk menavigasi ke Halaman 2
        if st.button("UMR"):
```

```
            page = "Halaman UMR"
    with colInflasi:
        # Tombol untuk menavigasi ke Halaman 3
        if st.button("Inflasi"):
            page = "Halaman Inflasi"

    # Memperbarui variabel state
    st.session_state["page"] = page

    # Menampilkan konten berdasarkan halaman yang sedang aktif
    if page == "Halaman Gaji":
        show_page_1()
    elif page == "Halaman UMR":
        show_page_2()
    elif page == "Halaman Inflasi":
        show_page_3()
```

Segmen Program 4.17 Koneksi Sistem dan *Database*

```
import psycopg2

def connect_to_database():
    try:
        # Connect to your PostgreSQL database
        connection = psycopg2.connect(
            dbname="sistem_prediksiGaji",
            user="postgres",
            password="caroline55",
            host="localhost",
            port="5432"
        )
        return connection
    except psycopg2.Error as e:
        print("Error connecting to PostgreSQL database:", e)
        return None
```

Segmen Program 4.18 Pencarian Kombinasi Variabel Terbaik

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import numpy as np
from connect import connect_to_database

def checkUMR (data, daerah):
    hasilMape = []
    connection = connect_to_database()
    query = "SELECT * FROM umr ORDER BY tahun_umr"
    data_umr = pd.read_sql_query(query, connection)
    if (daerah == "-"):
        daerah = "INDONESIA"
```

```
    data_umr = data_umr[data_umr['daerah_umr'].str.lower() ==
daerah.lower()]
    data    =    pd.merge(data,    data_umr,    how='left',
left_on='tahun', right_on='tahun_umr')
    query = "SELECT * FROM inflasi ORDER BY tahun_inflasi"
    data_inflasi = pd.read_sql_query(query, connection)
    data   =   pd.merge(data,   data_inflasi,   how='left',
left_on='tahun', right_on='tahun_inflasi')

    # Tidak Menggunakan UMR dan Inflasi
    X = data[['ipk', 'skkk_total']]
    y = data['gaji']
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
    hasilMape.append(mape)

    # Menggunakan UMR
    X = data[['ipk', 'skkk_total', 'nominal_umr']]
    y = data['gaji']
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
    hasilMape.append(mape)

    # Menggunakan Inflasi
    X = data[['ipk', 'skkk_total', 'inflasi_kontinu']]
    y = data['gaji']
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
    hasilMape.append(mape)

    # Menggunakan UMR dan Inflasi
    X    =    data[['ipk',    'skkk_total',    'nominal_umr',
'inflasi_kontinu']]
    y = data['gaji']
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
    hasilMape.append(mape)
```

```
    return hasilMape
```

Segmen Program 4.19 Pencarian Regularisasi Regresi Linear Terbaik

```
import numpy as np
from sklearn.linear_model import LinearRegression, Ridge,
Lasso, ElasticNet
from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.preprocessing import StandardScaler

def cekMetodeTerbaik(df, daftar_variabel):
    scaler = StandardScaler()
    daftar_variabel_new = []
    for i in daftar_variabel:
        daftar_variabel_new.append(i+"_transformed")
    df[daftar_variabel_new] =
scaler.fit_transform(df[daftar_variabel])
    mean = np.mean(df['gaji'])
    std = np.std(df['gaji'])
    daftar_variabel.remove('gaji')
    daftar_variabel_new.remove('gaji_transformed')
    X = df[daftar_variabel]
    y = df['gaji']
    X_transformed = df[daftar_variabel_new]
    y_transformed = df['gaji_transformed']
    hasil = []
    # Split the data into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
    X_train2, X_test2, y_train2, y_test2 =
train_test_split(X_transformed, y_transformed, test_size=0.2,
random_state=42)

    # Mencari alpha terbaik untuk ridge
    try:
        param_grid = {'alpha': np.logspace(-4, 4, 50)}
        ridge_reg = Ridge()
        grid_search = GridSearchCV(ridge_reg, param_grid,
cv=5)
        grid_search.fit(X_train2, y_train2)
        best_alpha_ridge = grid_search.best_params_['alpha']
    except Exception as e:
        best_alpha_ridge = 1.0

    # Mencari alpha terbaik untuk lasso
    try:
        param_grid = {'alpha': np.logspace(-4, 4, 50)}
        lasso_reg = Lasso()
        grid_search = GridSearchCV(lasso_reg, param_grid,
cv=5)
        grid_search.fit(X_train2, y_train2)
        best_alpha_lasso = grid_search.best_params_['alpha']
```

```
    except Exception as e:
        best_alpha_lasso = 1.0

    # Mencari alpha terbaik untuk elastic net
    try:
        param_grid = {'alpha': np.logspace(-4, 4, 50)}
        enet_reg = ElasticNet()
        grid_search = GridSearchCV(enet_reg, param_grid, cv=5)
        grid_search.fit(X_train2, y_train2)
        best_alpha_enet = grid_search.best_params_['alpha']
    except Exception as e:
        best_alpha_enet = 1.0

    # Define the models
    linear_reg = LinearRegression()
    ridge_reg = Ridge(alpha=best_alpha_ridge)
    lasso_reg = Lasso(alpha=best_alpha_lasso)
    elastic_net_reg = ElasticNet(alpha=best_alpha_enet)

    # Fit the models
    linear_reg.fit(X_train, y_train)
    ridge_reg.fit(X_train2, y_train2)
    lasso_reg.fit(X_train2, y_train2)
    elastic_net_reg.fit(X_train2, y_train2)

    # Make predictions
    y_pred_linear_reg = linear_reg.predict(X_test)
    y_pred_ridge_reg = ridge_reg.predict(X_test2)
    y_pred_lasso_reg = lasso_reg.predict(X_test2)
    y_pred_elastic_net_reg = elastic_net_reg.predict(X_test2)

    # Evaluate models
    def evaluate_model(y_test, y_pred):
        mape = np.mean(np.abs((y_test - y_pred) / y_test)) *
100
        return mape

    def evaluate_model_trans(y_test, y_pred):
        real_data = y_pred * std + mean
        mape = np.mean(np.abs((y_test - real_data) / y_test))
* 100
        return mape

    hasil.append(evaluate_model(y_test, y_pred_linear_reg))
    hasil.append(evaluate_model_trans(y_test,
y_pred_ridge_reg))
    hasil.append(evaluate_model_trans(y_test,
y_pred_lasso_reg))
    hasil.append(evaluate_model_trans(y_test,
y_pred_elastic_net_reg))
    hasil_min = min(hasil)
    if (hasil[0] == hasil_min):
        return(1)
```

```
    elif (hasil[1] == hasil_min):
        return(2)
    elif (hasil[2] == hasil_min):
        return(3)
    elif (hasil[3] == hasil_min):
        return(4)
```

Segmen Program 4.20 Prediksi Gaji Mahasiswa

```
import streamlit as st
import pandas as pd
from sklearn.linear_model import LinearRegression, Ridge,
Lasso, ElasticNet
import numpy as np
from connect import connect_to_database
from cekMetodeRegresi import cekMetodeTerbaik
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
from sklearn import metrics
from datetime import datetime
import seaborn as sns
from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.preprocessing import StandardScaler

def prediksiGaji (data, tahun, daerah, kode, daftar_tampilan,
data_prediksi, cek_metode):
    connection = connect_to_database()
    tahun_sekarang = datetime.now().year
    range_prediksi = int(tahun)+1-tahun_sekarang
    forecast_index = range(tahun_sekarang, int(tahun) + 1)
    if (kode != 1):
        # Prediksi Inflasi
        if (kode == 3 or kode == 4):
            query = "SELECT * FROM inflasi WHERE tahun_inflasi
< "+tahun+" ORDER BY tahun_inflasi"
            data_inflasi = pd.read_sql_query(query,
connection)
            data = pd.merge(data, data_inflasi, how='left',
left_on='tahun', right_on='tahun_inflasi')
            data_inflasi.set_index('tahun_inflasi',
inplace=True)
            arima_order = (1, 1, 1)
            model = ARIMA(data_inflasi['inflasi_kontinu'],
order=arima_order)
            model_fit = model.fit()
            forecast =
model_fit.forecast(steps=range_prediksi)
            forecast_inflasi =
forecast.to_frame().set_index(pd.Index(forecast_index))
            daftar_tampilan.append('inflasi_kontinu')
        # Prediksi UMR
        if (kode == 2 or kode == 4):
```

```
            if (daerah == "-"):
                daerah = "INDONESIA"
            query = "SELECT * FROM umr WHERE UPPER(daerah_umr)
= UPPER('" + daerah + "') AND tahun_umr < "+tahun+" ORDER BY
tahun_umr"
            data_umr = pd.read_sql_query(query, connection)
            data  =  pd.merge(data,  data_umr,  how='left',
left_on='tahun', right_on='tahun_umr')
            data_umr.set_index('tahun_umr', inplace=True)
            arima_order = (0, 2, 1)
            model      =      ARIMA(data_umr['nominal_umr'],
order=arima_order)
            model_fit = model.fit()
            forecast                                        =
model_fit.forecast(steps=range_prediksi)
            forecast_umr                                    =
forecast.to_frame().set_index(pd.Index(forecast_index))
            daftar_tampilan.append('nominal_umr')

    df = data[daftar_tampilan]
    corr_matrix = df.corr()
    tes = False
    try:
        cmatrix_inflasi       =       corr_matrix.loc['gaji',
'inflasi_kontinu']
    except Exception as e:
        cmatrix_inflasi = -1
    try:
        cmatrix_umr = corr_matrix.loc['gaji', 'nominal_umr']
    except Exception as e:
        cmatrix_umr = -1

    if (cmatrix_inflasi > 0):
        plt.figure(figsize=(12, 6))
        fig, ax1 = plt.subplots()
        ax2 = ax1.twinx()
        ax1.set_ylabel('Inflasi (cont)', color='green')
        ax1.set_xlabel('Tahun')
        ax1.plot(data_inflasi['inflasi_kontinu'],
label='Inflasi', color='green', marker='.')
        ax1.plot(forecast_inflasi,  label='Forecast  Inflasi',
color='blue', marker='.', linestyle='--')
        data_prediksi.append(forecast_inflasi.iloc[-
1].item())
        tes = True
    else:
        if 'inflasi_kontinu' in daftar_tampilan:
            daftar_tampilan.remove('inflasi_kontinu')

    if (cmatrix_umr > 0):
        if (tes == False):
            plt.figure(figsize=(12, 6))
            fig, ax1 = plt.subplots()
```

```
            ax2 = ax1.twinx()
            tes = True
        ax2.set_xlabel('Tahun')
        ax2.set_ylabel('UMR', color='red')
        ax2.plot(data_umr['nominal_umr'],        label='UMR',
color='red', marker='.')
        ax2.plot(forecast_umr,      label='Forecast      UMR',
color='magenta',marker='.', linestyle='--')
        data_prediksi.append(forecast_umr.iloc[-1].item())
    else:
        if 'nominal_umr' in daftar_tampilan:
            daftar_tampilan.remove('nominal_umr')

    if (cek_metode == False):
        id_program = data['program'].iloc[0]
        query = "SELECT * FROM program WHERE id_program =
"+str(id_program)
        detail_program = pd.read_sql_query(query, connection)
        metode = detail_program['metode_regresi'].iloc[0]
        if (metode == "Linear"):
            metode = 1
        elif (metode == "Ridge"):
            metode = 2
        elif (metode == "Lasso"):
            metode = 3
        elif (metode == "Elastic Net"):
            metode = 4
    else:
        metode = cekMetodeTerbaik(data, daftar_tampilan)
        daftar_tampilan.append('gaji')

    # Pengecekan metode masih kurang karena kurangnya data
    # Jika ada UMR data sudah pasti butuh normalisasi, jadi
dialihkan menggunakan LASSO yang mean errornya terkecil
    if 'nominal_umr' in daftar_tampilan and metode == 1:
        metode = 3

    if (metode != 1):
        scaler = StandardScaler()
        daftar_variabel_new = []
        for i in daftar_tampilan:
            daftar_variabel_new.append(i+"_transformed")
        data[daftar_variabel_new]                                =
scaler.fit_transform(data[daftar_tampilan])
        mean = np.mean(data['gaji'])
        std = np.std(data['gaji'])
        daftar_tampilan.remove('gaji')
        daftar_variabel_new.remove('gaji_transformed')
        data_prediksi_new = []
        for i in range (len(data_prediksi)):
            data_prediksi_new.append((data_prediksi[i]-
np.mean(data[daftar_tampilan[i]]))/np.std(data[daftar_tampila
n[i]]))
```

```
        X_transformed = data[daftar_variabel_new]
        y_transformed = data['gaji_transformed']
        X_train2,     X_test2,     y_train2,     y_test2     =
train_test_split(X_transformed, y_transformed, test_size=0.2,
random_state=42)
    else:
        daftar_tampilan.remove('gaji')

    X = data[daftar_tampilan]
    y = data['gaji']
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    metodeUsed = ""
    if (metode == 1):
        model = LinearRegression()
        model_cek = LinearRegression()
        metodeUsed = "Linear Regression"
    elif (metode == 2):
        try:
            param_grid = {'alpha': np.logspace(-4, 4, 50)}
            ridge_reg = Ridge()
            grid_search = GridSearchCV(ridge_reg, param_grid,
cv=5)
            grid_search.fit(X_transformed, y_transformed)
            best_alpha_ridge                                    =
grid_search.best_params_['alpha']
        except Exception as e:
            best_alpha_ridge = 1.0
        model = Ridge(alpha=best_alpha_ridge)
        model_cek = Ridge(alpha=best_alpha_ridge)
        metodeUsed = "Ridge Regression"
    elif (metode == 3):
        try:
            param_grid = {'alpha': np.logspace(-4, 4, 50)}
            lasso_reg = Lasso()
            grid_search = GridSearchCV(lasso_reg, param_grid,
cv=5)
            grid_search.fit(X_transformed, y_transformed)
            best_alpha_lasso                                    =
grid_search.best_params_['alpha']
        except Exception as e:
            best_alpha_lasso = 1.0
        model = Lasso(alpha=best_alpha_lasso)
        model_cek = Lasso(alpha=best_alpha_lasso)
        metodeUsed = "Lasso Regression"
    elif (metode == 4):
        try:
            param_grid = {'alpha': np.logspace(-4, 4, 50)}
            enet_reg = ElasticNet()
            grid_search = GridSearchCV(enet_reg, param_grid,
cv=5)
            grid_search.fit(X_transformed, y_transformed)
```

```
            best_alpha_enet                                    =
grid_search.best_params_['alpha']
        except Exception as e:
            best_alpha_enet = 1.0
        model = ElasticNet(alpha=best_alpha_enet)
        model_cek = ElasticNet(alpha=best_alpha_enet)
        metodeUsed = "Elastic Net Regression"

    if (metode != 1):
        model.fit(X_transformed, y_transformed)
        y_pred = model.predict([data_prediksi_new])
        hasil = y_pred[0] * std + mean
        model_cek.fit(X_train2, y_train2)
        y_pred_cek_n = model_cek.predict(X_test2)
        y_pred_cek = y_pred_cek_n * std + mean
    else:
        model.fit(X, y)
        y_pred = model.predict([data_prediksi])
        hasil = y_pred[0]
        model_cek.fit(X_train, y_train)
        y_pred_cek = model_cek.predict(X_test)

    mae = metrics.mean_absolute_error(y_test, y_pred_cek)
    mape = np.mean(np.abs((y_test - y_pred_cek) / y_test)) *
100
    st.write(f"**Nilai MAE:** {mae}")
    st.write(f"**Nilai MAPE:** {mape}")
    st.write(f"**Jumlah Data Training:** {len(data)}")

    if (tes):
        fig.tight_layout()
        plt.title('ARIMA Forecast')
        lines, labels = ax1.get_legend_handles_labels()
        lines2, labels2 = ax2.get_legend_handles_labels()
        plt.legend(lines   +   lines2,   labels   +   labels2,
loc='upper left')
        plt.show()
        st.pyplot(plt)

    daftar_tampilan.append('gaji')
    df = data[daftar_tampilan]
    corr_matrix = df.corr()
    plt.figure(figsize=(8, 6))
    sns.heatmap(corr_matrix,   annot=True,   cmap='coolwarm',
fmt=".2f")
    plt.title('Correlation Matrix')
    st.pyplot(plt)

    st.write(f"**Metode yang Digunakan:** {metodeUsed}")
    st.write(f"**Hasil Prediksi:** {hasil}")
    ipk_gaji = corr_matrix.loc['ipk', 'gaji']
    skkk_gaji = corr_matrix.loc['skkk_total', 'gaji']
    hub_ipk = cekHubungan(ipk_gaji)
```

```
    hub_skkk = cekHubungan(skkk_gaji)
    st.write(f"**Persentase        Hubungan        IPK-Gaji:**
{ipk_gaji*100}% ({hub_ipk})")
    st.write(f"**Persentase       Hubungan       SKKK-Gaji:**
{skkk_gaji*100}% ({hub_skkk})")


def cekHubungan(nilai):
    if (nilai <= 0):
        return("Tidak ada hubungan")
    elif (nilai <= 0.2):
        return("Sangat lemah")
    elif (nilai <= 0.4):
        return("Lemah")
    elif (nilai <= 0.6):
        return("Rata-rata")
    elif (nilai <= 0.8):
        return("Kuat")
    elif (nilai <= 1):
        return("Sangat kuat")
```

4.6.2 Aplikasi Verifikasi Data Baru Admin

Aplikasi untuk verifikasi data baru oleh admin dimulai dari *login user*, kemudian diikuti pilihan halaman-halaman data apa yang ingin diverifikasi. Pada setiap halaman, admin bisa melihat data yang sudah ada atau memverifikasi data baru atau menghapus data baru tersebut. Seluruh *code* untuk aplikasi admin ini tertuang dalam segmen program 4.21.

Segmen Program 4.21 Aplikasi Verifikasi Data Baru Admin

```
import streamlit as st
import pandas as pd
from streamlit_option_menu import option_menu
from connect import connect_to_database
import streamlit as st

username = "admin"
password = "password"

st.markdown(
    """
    <style>
    .stButton>button {
        width: 100%;
    }
    </style>
    """,
    unsafe_allow_html=True
)
```

```
# Initialize session state
if 'button_clicked' not in st.session_state:
    st.session_state.button_clicked = False

# Display button with dynamic label
if st.session_state.button_clicked:
    connection = connect_to_database()
    cursor = connection.cursor()
    query = 'SELECT * FROM data_mahasiswa WHERE konfirmasi =
1 ORDER BY id_data_gaji'
    df_gaji = pd.read_sql(query, connection)
    query = 'SELECT * FROM data_mahasiswa WHERE konfirmasi =
0 ORDER BY id_data_gaji'
    df_gaji_2 = pd.read_sql(query, connection)
    query = 'SELECT * FROM umr WHERE konfirmasi = 1 ORDER BY
id_umr'
    df_umr = pd.read_sql(query, connection)
    query = 'SELECT * FROM umr WHERE konfirmasi = 0 ORDER BY
id_umr'
    df_umr_2 = pd.read_sql(query, connection)
    query = 'SELECT * FROM inflasi WHERE konfirmasi = 1 ORDER
BY id_inflasi '
    df_inflasi = pd.read_sql(query, connection)
    query = 'SELECT * FROM inflasi WHERE konfirmasi = 0 ORDER
BY id_inflasi '
    df_inflasi_2 = pd.read_sql(query, connection)

    def verify_data(id, code):
        if (code == 1):
            tabel = "data_mahasiswa"
            kolom = "id_data_gaji"
        elif (code == 2):
            tabel = "umr"
            kolom = "id_umr"
        elif (code == 3):
            tabel = "inflasi"
            kolom = "id_inflasi"
        query = f"UPDATE {tabel} SET konfirmasi = 1 WHERE
{kolom} = {id}"
        cursor.execute(query)
        connection.commit()
    def hapus_data(id, code):
        if (code == 1):
            tabel = "data_mahasiswa"
            kolom = "id_data_gaji"
        elif (code == 2):
            tabel = "umr"
            kolom = "id_umr"
        elif (code == 3):
            tabel = "inflasi"
            kolom = "id_inflasi"
```

```
        query = f"DELETE FROM {tabel} WHERE {kolom} = {id}"
        cursor.execute(query)
        connection.commit()

    def cont():
        st.empty()
        with st.sidebar:
            selected = option_menu(
                'Verifikasi Data', ['Data Gaji', 'Data UMR',
'Data Inflasi', 'Keluar'], default_index=0
            )

        if selected == 'Data Gaji':
            st.title("Data Gaji")
            page = st.session_state.get("page", "Gaji")
            colLihatGaji, colVerifikasiGaji = st.columns(2)
            with colLihatGaji:
                if st.button("Lihat Data yang Sudah Ada"):
                    page = "Lihat Data Gaji"
                    st.session_state.page = "Lihat Data Gaji"
            with colVerifikasiGaji:
                if st.button("Verifikasi Data Baru"):
                    page = "Verifikasi Data Gaji"
                    st.session_state.page = "Verifikasi Data
Gaji"

            if page == "Lihat Data Gaji":
                st.table(df_gaji)
            elif page == "Verifikasi Data Gaji":
                for index, row in df_gaji_2.iterrows():
                    colVerifikasi, colHapus = st.columns(2)
                    with colVerifikasi:
                        if     st.button(f"Verifikasi     Data
{row['kode_unik']}"):
                            verify_data(row['id_data_gaji'],
1)
                            st.write(f"Data dengan kode_unik
{row['kode_unik']} telah diverifikasi.")
                    with colHapus:
                        if       st.button(f"Hapus       Data
{row['kode_unik']}"):
                            hapus_data(row['id_data_gaji'],
1)
                            st.write(f"Data dengan kode_unik
{row['kode_unik']} berhasil dihapus.")
                    st.table(row)


        elif selected == 'Data UMR':
            st.title("Data UMR")
            page = st.session_state.get("page", "Gaji")
            colLihatUmr, colVerifikasiUmr = st.columns(2)
            with colLihatUmr:
```

```
                if st.button("Lihat Data yang Sudah Ada"):
                    page = "Lihat Data UMR"
                    st.session_state.page = "Lihat Data UMR"
            with colVerifikasiUmr:
                if st.button("Verifikasi Data Baru"):
                    page = "Verifikasi Data UMR"
                    st.session_state.page = "Verifikasi Data
UMR"

            if page == "Lihat Data UMR":
                st.table(df_umr)

            elif page == "Verifikasi Data UMR":
                for index, row in df_umr_2.iterrows():
                    colVerifikasi, colHapus = st.columns(2)
                    with colVerifikasi:
                        if     st.button(f"Verifikasi     Data
{row['id_umr']}"):
                            verify_data(row['id_umr'], 2)
                            st.write(f"Data     dengan     id
{row['id_umr']} telah diverifikasi.")
                    with colHapus:
                        if       st.button(f"Hapus       Data
{row['id_umr']}"):
                            hapus_data(row['id_umr'], 2)
                            st.write(f"Data     dengan     id
{row['id_umr']} berhasil dihapus.")
                    st.table(row)

        elif selected == 'Data Inflasi':
            st.title("Data Inflasi")
            page = st.session_state.get("page", "Gaji")
            colLihatInf, colVerifikasiInf = st.columns(2)
            with colLihatInf:
                if st.button("Lihat Data yang Sudah Ada"):
                    page = "Lihat Data Inflasi"
                    st.session_state.page   =   "Lihat   Data
Inflasi"
            with colVerifikasiInf:
                if st.button("Verifikasi Data Baru"):
                    page = "Verifikasi Data Inflasi"
                    st.session_state.page = "Verifikasi Data
Inflasi"

            if page == "Lihat Data Inflasi":
                st.table(df_inflasi)

            elif page == "Verifikasi Data Inflasi":
                for index, row in df_inflasi_2.iterrows():
                    colVerifikasi, colHapus = st.columns(2)
                    with colVerifikasi:
                        if     st.button(f"Verifikasi     Data
{row['id_inflasi']}"):
```

```
                            verify_data(row['id_inflasi'], 3)
                            st.write(f"Data     dengan     id
{row['id_inflasi']} telah diverifikasi.")
                    with colHapus:
                        if       st.button(f"Hapus       Data
{row['id_inflasi']}"):
                            hapus_data(row['id_inflasi'], 3)
                            st.write(f"Data     dengan     id
{row['id_inflasi']} berhasil dihapus.")
                    st.table(row)
        elif selected == "Keluar":
            st.session_state.button_clicked = False
    cont()
else:
    st.title("Selamat Datang")
    user_input = st.text_input("Nama Pengguna")
    password_input     =     st.text_input("Kata     Sandi",
type="password")
    if st.button("Masuk"):
        if  (user_input  ==  username  and  password_input  ==
password):
            st.session_state.button_clicked = True
        else:
            st.error("Nama  pengguna  atau  kata  sandi  salah.
Silakan masukkan lagi.")
```