3. PERENCANAAN DAN IMPLEMENTASI SISTEM

Tugas akhir ini bertujuan mencari hubungan dan pengaruh parameter yang mempengaruhi penyimpangan impedansi di PT. X dengan menggunakan *machine learning* yang diimplementasikan pada sebuah aplikasi berbasis website (*streamlit*). Untuk merealisasikan, tugas akhir ini dikerjakan secara berkelompok di mana penulis akan menggunakan algoritma asosiasi - ECLAT untuk mencari hubungan parameter dengan penyimpangan impedansi dan algoritma *explainable AI* – SHAP untuk mencari pengaruh parameter dengan penyimpangan impedansi. Tugas akhir akan dibuat untuk menghasilkan rekomendasi modifikasi parameter untuk mendapatkan nilai impedansi yang diinginkan pada sebuah aplikasi berbasis website (*streamlit*). Aplikasi tersebut akan menerima *input data*, memprediksi nilai impedansi dan menghasilkan rekomendasi modifikasi parameter serta nilai impedansi setelah parameter dimodifikasi. Selain itu, aplikasi tersebut memiliki *tab* untuk menampilkan hasil pola asosiasi dan parameter yang paling berpengaruh.

3.1 Persiapan Data

Pada tahapan ini, persiapan data mencakup pengumpulan dataset ukuran koil transformator dari departemen Produksi dan dataset impedansi dari departemen QIT (*Quality Inspection & Testing*) di PT. X. Dataset ini dibagi menjadi 3 bagian, yaitu koil LV, koil HV, dan impedansi. Dataset pertama didapatkan ketika pada awal tahun 2024, sementara dataset kedua pada April 2024. Jumlah data transformator adalah 945 data. Kemudian, dilanjutkan dengan proses pembersihan data, integrasi data, dan eksplorasi data sebelum dilakukan analisis lebih lanjut. Seluruh tahapan ini dilakukan pada aplikasi *Excel* sebelum masuk ke *Kaggle.* Output yang dihasilkan pada tahap ini adalah *final dataset* yang akan digunakan dalam proses asosiasi dan XAI.

3.1.1 *Data Cleaning*

Data cleaning adalah proses dalam analisis data yang melibatkan identifikasi, koreksi, dan penghapusan kesalahan atau inkonsistensi dalam *dataset*. Tujuannya adalah memastikan bahwa data yang digunakan dalam analisis atau pemodelan akurat, lengkap, dan konsisten. Langkah-langkah dalam *data cleaning* mencakup identifikasi kesalahan seperti nilai yang hilang, nilai non numerik, duplikat, diikuti dengan koreksi kesalahan dan penghapusan daya yang tidak

relevan. Normalisasi data juga diperlukan untuk menormalkan format data agar seragam dan konsisten. Proses ini juga memverifikasi konsistensi data di seluruh *dataset*.

Tabel 3.1

Data Panjang Lebar Tinggi Koil Transformator

JO	PID LV	LID LV	TID LV	POD LV	LOD LV	TOD LV	PID HV	LID HV	TID HV	POD HV	LOD HV	TOD HV
Coil 1	269	213	750	438	325	752	438	329	752	584	452	752
Coil 2	269	215	750	435	325	752	437	327	752	584	443	750
Coil 3	212	163	350	340	260	352	340	260	353	462	341	353
Coil 4	213	163	350	338	257	352	434	334	352	546	341	353
Coil 5	213	163	350	338	257	352	340	260	352	460	343	353
Coil 6	301	223	380	433	306	380	435	307	380	570	415	380
Coil 7	301	223	380	433	306	380	433	310	380	570	405	380
Coil 8	303	223	380	432	310	382	432	310	382	562	405	382
Coil 9	303	224	382	432	310	382	435	310	382	579	400	384
Coil 10	251	183	310	364	254	312	364	254	312	472	324	313
Coil 11	251	183	310	364	255	312	367	256	313	480	324	313
Coil 12	251	184	310	367	256	312	467	318	312	470	326	312
Coil 13	252	184	310	366	257	312	467	318	312	469	331	312
Coil 14	283	224	350	424	312	352	426	312	352	548	410	352
Coil 15	330	253	348	473	349	350	473	349	352	649	417	352
Coil 16	330	253	350	470	348	352	477	354	352	630	457	347
Coil 17	330	253	350	470	348	352	477	354	352	630	457	347
Coil 18	189	143	315	285	210	317	376	273	317	376	268	318
Coil 19	275	203	420	396	290	420	537	377	422	527	378	422
Coil 20	275	203	420	397	290	420	537	377	422	532	379	422

Dataset ini berisi 945 data dengan total 12 parameter, antara lain IDL LV (*Inner* Diameter *Length Low Voltage*), IDW LV (*Inner* Diameter *Width Low Voltage*), IDH (*Inner* Diameter *Height Low Voltage*), ODL LV (*Outer* Diameter *Length Low Voltage*), ODW LV (*Outer* Diameter *Width Low Voltage*), ODH LV (*Outer* Diameter *Height Low Voltage*), IDL HV (*Inner* Diameter *Length High Voltage*), IDW HV (*Inner* Diameter *Width High Voltage*), IDH HV (*Inner* Diameter *Height High Voltage*), ODL HV (*Outer* Diameter *Length High Voltage*), ODE HV (*Outer* Diameter *Width High Voltage*), ODH HV (*Outer* Diameter *Height High Voltage*).

Tabel 3.2

Data Impedansi Transformator

JO/SN	Tahun No Seri	Phasa	Capacity (KVA)	Frek (HZ)	Voltage		Vector Group	Date of test	Tank Number	% Imp. at(75°C)
					HV (Volt)	LV (Volt)				
Coil	1	3	800	50	11000	433	Dyn11	23-Dec-22	TAP 2	**4.61**
Coil	1	3	630	50	20000	400	YNyn0	3-Jan-23	TAP 3	**4.22**
Coil	1	3	1000	50	6600	400	Dyn11	5-Jan-23	TAP 3	**4.79**
Coil	1	3	1600	50	6600	400	Dyn11	5-Jan-23	TAP 3	**6.05**
Coil	1	3	3500	50	11000	730	Dd0	5-Jan-23	TAP 3	**5.63**
Coil	1	3	1000	50	20000	595	Dy5	6-Jan-23	TAP 3	**4.69**
Coil	1	3	2000	50	20000	400	Dyn5	7-Jan-23	TAP 3	**6.45**
Coil	1	3	100	50	22000	433	Dyn11	7-Jan-23	TAP 3	**4.00**
Coil	1	3	2500	50	20000	400	Dyn5	7-Jan-23	TAP 3	**7.26**
Coil	1	3	100	50	22000	433	Dyn11	7-Jan-23	TAP 3	**3.91**
Coil	1	3	630	50	20000	400	YNyn0	9-Jan-23	TAP 3	**3.90**
Coil	1	3	2000	50	20000	400	Dyn5	9-Jan-23	TAP 3	**7.52**
Coil	1	3	630	50	20000	400	YNyn0	9-Jan-23	TAP 3	**3.88**
Coil	1	3	2000	50	20000	400	Dyn5	10-Jan-23	TAP 3	**7.54**

3.1.2 *Data Integration*

Data *integration* adalah proses menggabungkan data dari berbagai sumber yang berbeda menjadi satu *dataset* yang terpadu. Tujuannya adalah untuk menciptakan gambaran yang lebih lengkap dan komprehensif tentang suatu subjek yang dianalisis. Proses data *integration* melibatkan pemahaman terhadap struktur dan format data dari setiap sumber, transformasi data ke format yang seragam dan konsisten, serta penggabungan data tersebut menjadi satu *dataset* yang terpadu. Data *integration* yang dilakukan adalah menggabungkan parameter: JO, PID LV, LID LV, TID LV, POD LV, LOD LV, TOD LV, PID HV, LID HV, TID HV, POD HV, LOD HV, TOD HV, Imp. Data *integration* ini sangat penting karena perlu memastikan bahwa JO transformer sama, sehingga tidak terjadi pertukaran nilai impedansi dengan hasil Tabel 3.3.

Tabel 3.3

Data Panjang, Lebar, Tinggi dan Impedansi Transformator

Coil	PID LV	LID LV	TID LV	POD LV	LOD LV	TOD LV	PID HV	LID HV	TID HV	POD HV	LOD HV	TOD HV	IMP
Coil 1	187	152	220	197	157	220	262	211	288	262	232	220	3,26
Coil 2	152	133	320	254	206	320	255	210	320	348	280	323	3,49
Coil 3	133	253	343	325	266	347	416	319	347	422	326	347	3,55
Coil 4	104	103	286	150	132	286	168	147	286	283	235	286	3,55
Coil 5	226	183	345	330	257	347	422	319	347	425	339	347	3,59
Coil 6	236	183	340	333	256	340	335	325	342	430	338	343	3,61
Coil 7	226	183	345	331	258	345	331	258	347	415	324	347	3,62
Coil 8	210	163	550	319	245	552	325	246	552	423	318	551	3,62
Coil 9	152	134	320	255	212	322	257	214	320	342	277	322	3,63
Coil 10	217	163	340	324	233	342	326	235	342	407	291	344	3,63
Coil 11	229	185	345	326	250	347	422	319	347	426	328	347	3,64
Coil 12	196	148	315	285	210	317	285	210	315	379	268	317	3,64
Coil 13	217	163	340	324	233	342	324	234	342	407	294	344	3,64
Coil 14	228	183	345	329	249	347	330	254	343	438	334	347	3,65
Coil 15	210	163	550	326	245	550	414	247	552	430	319	552	3,66
Coil 16	226	183	345	328	254	347	330	319	347	420	337	347	3,67
Coil 17	210	163	550	319	243	550	326	246	552	423	317	552	3,68
Coil 18	189	143	315	285	210	317	376	212	317	376	268	318	3,68
Coil 19	162	152	265	264	220	267	265	220	267	356	280	267	3,68
Coil 20	226	183	345	331	258	345	422	319	347	430	329	347	3,68

3.1.3 *Data Exploration*

Data panjang, lebar, dan tinggi *inner*, *outer* diameter merupakan suatu data numerik, sementara pada proses asosiasi memerlukan data berupa kategorial. Sehingga pengubahan data numerik menjadi data kategorial dapat dilakukan dengan beberapa cara, yaitu *fix-width binning, Equal depth-width binning.* Berikut merupakan implementasi dari ketiga cara di atas.

3.1.3.1 *Fix-Width Binning*

Fix-Width Binning digunakan untuk mengubah data numerik pada rentang/*range* nilai yang konsisten, seperti 1-10, 11-20, 21-29. Di mana setiap rentang akan dimasukkan dalam kelompok tertentu seperti 1-10: PID LV0, 11-20: PID LV1, 21-29: PID LV2, dan seterusnya. *Fix-width binning* digunakan untuk mengubah data numerik panjang, lebar, dan tinggi trafo dengan rentang yang tetap. Berikut merupakan kode program yang dijalankan pada aplikasi *Microsoft excel*.

```
Function TRANSCONV(dimn, num)
out = True
For i = 1 To 100
   If num < 10 * i And out = True Then
      i = i - 1
      TRANSCONV = dimn & i
      out = False
   End If
Next i

End Function
```

Gambar 3.1 Kode Program *Fix-Width Binning* pada PLT Transformator

Fungsi ‘TRANSCONV’ dirancang untuk mengubah data panjang, lebar, tinggi transformer yang berupa numerik menjadi kategorial dalam Microsoft Excel. Parameter 'dimn' digunakan untuk menyediakan label atau kategori untuk data yang akan diubah, sementara parameter 'num' mungkin mewakili nilai numerik yang akan dikonversi. Fungsi ini bekerja dengan cara membagi rentang nilai numerik menjadi kategori berdasarkan interval tertentu, yaitu jika nilai 'num' kurang dari 10 kali nilai iterasi 'i', fungsi akan mengembalikan label kategori yang sesuai dengan interval tersebut. Misalnya, jika 'num' adalah 7, maka fungsi akan mengembalikan label bernilai 1 karena 7 kurang dari 10*1 (i = 1).

```
Function IMPCONV(dimn, num)
out = True
For i = 1 To 20
   If num >= -i And num < 0 And out = True Then
      i = -i
      IMPCONV = dimn & i
      out = False
   End If

   If num <= i And num > 0 And out = True Then
      i = i
      IMPCONV = dimn & i
      out = False
   End If
Next i

End Function
```

Gambar 3.2 Kode Program *Fix-Width Binning* pada Impedansi Transformator 01

Fungsi IMPCONV dirancang untuk mengubah data impedansi transformer yang berupa numerik menjadi kategorial dalam Microsoft Excel. Parameter *'dimn'* digunakan untuk menyediakan label atau kategori untuk data yang akan diubah, sementara parameter *'num'* mungkin mewakili nilai numerik yang akan dikonversi. Fungsi ini bekerja dengan cara membagi rentang nilai numerik menjadi kategori berdasarkan interval 3 hingga 11.

3.2 *Eclat Algorithm – Association Machine Learning*

```
import numpy as np
import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder
```

Gambar 3.3 Kode Program *Import Library*

Pada tahap ini kode algoritma ECLAT dijabarkan dan diimplementasikan pada *platform Kaggle*. Hasil akhirnya adalah aturan pola hubungan data jika-maka yang ditampung pada sebuah *file Excel.* Kode di atas merupakan implementasi algoritma *Association Rule Mining* menggunakan Python, yang bertujuan untuk menemukan asosiasi atau hubungan antara item-item dalam *dataset* transaksi. Pertama, *import library* yang diperlukan, termasuk *NumPy* untuk operasi numerik, Pandas untuk manipulasi data dalam format *DataFrame*, dan modul dari paket *mlxtend* yang disediakan untuk analisis data mining. Pada baris berikutnya, *TransactionEncoder* dari modul *preprocessing* digunakan untuk mengubah data transaksi ke dalam format yang sesuai dengan algoritma Apriori. Kemudian, algoritma Apriori dijalankan pada data yang telah diformat menggunakan fungsi apriori dari *mlxtend.frequent_patterns*. Fungsi ini akan menghasilkan himpunan itemset yang sering muncul bersama-sama, yang selanjutnya digunakan untuk mengekstrak aturan asosiasi menggunakan fungsi *association_rules*.

```
data1 = pd.read_excel('/kaggle/input/rekap-coil-dataset/rekap_coil_v4.xlsx',
'association_fixwidth_imp')
arr1 = data1.to_numpy() #association_fixwidth_imp


data = arr1
```

Gambar 3.4 Kode Program Membaca *Dataset*

Kode program tersebut membaca satu file Excel, 'rekap_coil_v4.xlsx', yang memiliki beberapa *sheet* yang berbeda, yaitu '*association_fixwidth*'. Data dari kedua *sheet* tersebut diubah menjadi *array NumPy* menggunakan fungsi *'to_numpy()'*. *Array* pertama disimpan dalam variabel '*arr1*' yang merepresentasikan data dari *sheet* '*association_fixwidth*'.

```
te = TransactionEncoder()
te_ary = te.fit(data).transform(data)
df = pd.DataFrame(te_ary, columns=te.columns_)
df
```

Gambar 3.5 Kode Program *TransactionEncoder*

Dalam analisis aturan asosiasi, penggunaan *TransactionEncoder* dari modul *mlxtend.preprocessing* berperan mengubah data transaksi menjadi format yang dapat digunakan untuk pemodelan. Langkah pertama adalah membuat objek *TransactionEncoder*, yang kemudian digunakan untuk menyesuaikan dan mengubah daftar transaksi menjadi sebuah *array NumPy*. Dengan nilai 0 dan 1 yang mengindikasikan kehadiran atau ketiadaan suatu item dalam *dataset*, dapat dilakukan penelitian tentang asosiasi antar-item yang dapat mengungkapkan informasi penting tentang preferensi hubungan antara parameter yang efektif.

```
frequent_itemsets = apriori(df, min_support=0.01, use_colnames=True)
frequent_itemsets
```

Gambar 3.6 Kode Program *Apriori – Eclat Apply*

Fungsi *apriori* dari *library*, *mlxtend*, digunakan untuk mengidentifikasi kumpulan item yang sering muncul (*frequent itemsets*) dari *DataFrame* yang telah dihasilkan sebelumnya. Pada awalnya dilakukan penentuan parameter *min_support*=0.01 untuk menetapkan ambang batas minimum dukungan yang diperlukan untuk sebuah *itemset* dianggap sering muncul, dan *use_colnames=True* untuk menggunakan nama kolom dari *DataFrame* sebagai nama item dalam output. Hasil dari langkah ini akan menghasilkan kumpulan item yang sering muncul beserta nilai dukungan (*support*) dari setiap *itemset*.

```
rules = association_rules(frequent_itemsets, metric ="confidence", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])
rules = rules[['antecedents', 'consequents','support','confidence','lift']]
rules.to_excel("eclat_rules.xlsx")


rules
```

Gambar 3.7 Kode Implementasi Aturan Asosiasi

Fungsi *'association_rules'* dari *library mlxtend* digunakan untuk membuat aturan asosiasi dari kumpulan item yang sering muncul yang telah diidentifikasi sebelumnya. Parameter *metric="confidence"* digunakan untuk mengukur kekuatan asosiasi antara *itemset* dengan

metrik *"confidence",* dan "*min_threshold*=0.01" untuk menetapkan ambang batas minimum nilai *confidence* yang diperlukan untuk sebuah aturan asosiasi.

Setelah itu, mengurutkan aturan-aturan asosiasi berdasarkan nilai confidence dan lift secara menurun. Kemudian, memilih kolom-kolom yang ingin disimpan dalam *DataFrame* hasil, yaitu *'antecedents', 'consequents', 'support', 'confidence'*, dan *'lift'*. Hasil dari langkah ini adalah *DataFrame* yang berisi aturan-aturan asosiasi yang telah diurutkan dan difilter, kemudian disimpan dalam file *Excel* dengan nama "eclat_rules.xlsx".

3.3 *SHAP Algorithm – XAI Machine Learning*

```
shap.initjs()
```

Gambar 3.8 Inisiasi SHAP

Pada tahap ini kode algoritma SHAP dijabarkan dan diimplementasikan pada *platform Kaggle* dan *Streamlit*. Hasil akhirnya adalah SHAPLEY *values* dan sebuah plot untuk menunjukkan ringkasan dari SHAPLEY *values.* Kode *shap.initjs*() adalah bagian dari pustaka SHAP (*SHapley Additive exPlanations*). SHAP adalah sebuah pustaka Python yang digunakan untuk menjelaskan prediksi model machine learning dengan cara menghitung kontribusi masing-masing parameter terhadap prediksi. SHAP menggunakan konsep teori permainan *Shapley values* untuk menghitung kontribusi setiap parameter terhadap hasil prediksi.

```
def create_shapval(explainer, test_data):
   shap_values = pd.DataFrame(explainer.shap_values(test_data), columns=[str(col+' Shap') for col in test_data.columns])
   shap_values.index = test_data.index

#    shap_data = pd.concat([test_data, shap_values], axis=1)
   explanation = shap.Explanation(shap_values.values, data=test_data, base_values=Y.mean(), feature_names=test_data.columns)
   print("Explanation Length :", len(explanation))

   return shap_values, explanation
```

Gambar 3.9 Fungsi *create_shapval*

Fungsi '*create_shapval*' menciptakan nilai SHAP (*Shapley Additive exPlanations*) dan objek penjelasan terkait dengan model *machine learning* yang telah dijelaskan oleh *explainer* menggunakan data uji *test data*. Pada awalnya, fungsi ini menghasilkan nilai SHAP untuk setiap sampel dalam *test data* menggunakan objek *explainer*. Kemudian, nilai-nilai ini disimpan dalam *DataFrame Pandas* dengan nama kolom yang dihasilkan berdasarkan nama kolom dari *test_data*.

Objek penjelasan (*explanation*) juga dibuat menggunakan pustaka SHAP. Objek ini berisi nilai-nilai SHAP yang dihasilkan sebelumnya, data uji, nilai dasar (*base values*), dan nama parameter dari *test_data*. Pada akhirnya, panjang penjelasan (*explanation*) dicetak, dan fungsi mengembalikan *DataFrame* yang berisi nilai-nilai SHAP dan objek penjelasan tersebut.

```
# XGBoost
xgb_explainer = shap.TreeExplainer(xgb)
xgb_shap, xgb_explanation = create_shapval(xgb_explainer, x_test)
```

Gambar 3.10 Objek *xgb_explainer*

Objek *xgb_explainer*, yang diinisialisasi dengan *TreeExplainer* untuk model *XGBoost* (*xgb*). Objek *xgb_explainer* digunakan sebagai argumen untuk memanggil fungsi *create_shapval* bersama dengan data uji *x_test*. Fungsi *create_shapval* menghasilkan dua keluaran: *xgb_shap*, yang berisi nilai-nilai SHAP untuk setiap sampel dalam *x_test*, dan *xgb_explanation*, yang merupakan objek penjelasan yang menyimpan nilai-nilai SHAP serta informasi lain yang diperlukan seperti data uji, nilai dasar, dan nama parameter.

Dengan cara ini, menghasilkan nilai-nilai SHAP dan objek penjelasan untuk model *XGBoost* (*xgb*) menggunakan data uji (*x_test*). Nilai-nilai SHAP digunakan untuk menganalisis kontribusi setiap parameter terhadap prediksi model *XGBoost*.

```
def shap_plot(plot_type, shap_val, feature_data, explainer=None, explanation=None, index1=None, index2=None, show=True):

    def safe(to_check):
        if to_check is not None:
            return True
        else:
            print("ERROR: Use the right argument on", to_check)
            return False

    if plot_type == 'summary':
        shap.summary_plot(shap_val, features=feature_data, feature_names=feature_data.columns, show=show)
    elif plot_type == 'decision':
        return False if not safe(explainer) else shap.decision_plot(explainer.expected_value, shap_val, features=feature_data)
    elif plot_type == 'dependence':
        return False if not safe(index1) and safe(index2) else shap.dependence_plot(index1, interaction_index=index2, shap_values=shap_val, features=feature_data)
    elif plot_type == 'bar':
        shap.bar_plot(shap_val, feature_data.values[index1], feature_data.columns)
    elif plot_type == 'waterfall':
        shap.plots.waterfall(explanation, max_display=len(feature_data.columns)) # max_display=data.columns
```

Gambar 3.11 Fungsi *shap_plot*

Fungsi *shap_plot* ini digunakan untuk memvisualisasikan nilai-nilai SHAP dengan berbagai jenis plot tergantung pada argumen *plot_type*. Berikut adalah penjelasan singkat tentang apa yang dilakukan oleh masing-masing tipe plot:

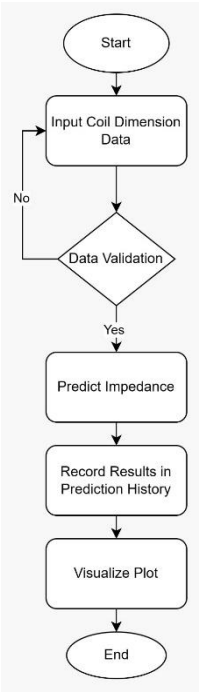
- *summary*: Ini menghasilkan plot ringkasan nilai-nilai SHAP, yang menunjukkan dampak relatif setiap parameter terhadap prediksi model secara keseluruhan.
- *decision*: Ini menunjukkan plot keputusan, yang menyoroti kontribusi masing-masing parameter pada keputusan individu tertentu.

- *dependence*: Ini menampilkan plot ketergantungan, yang menunjukkan bagaimana nilai-nilai SHAP dari satu parameter bervariasi sehubungan dengan nilai parameter lainnya.
- *Bar*: Ini menampilkan plot batang sederhana, yang menunjukkan tingkat pentingnya setiap parameter dalam membuat prediksi model. Hal ini serupa dengan plot ringkasan, tetapi dengan tampilan yang berbeda.
- *Waterfall*: Plot ini memberikan gambaran langkah demi langkah tentang bagaimana setiap parameter mempengaruhi prediksi model dari nilai *baseline*. Ini membantu untuk memahami kontribusi relatif dari setiap parameter pada prediksi model.

Fungsi ini menerima argumen tambahan seperti *shap_val* (nilai-nilai SHAP), *feature_data* (parameter data), *explainer* (*objek explainer*), *explanation* (*objek penjelasan*), dan *dep_index* (indeks interaksi untuk plot ketergantungan).

Fungsi safe merupakan fungsi bantuan yang memastikan bahwa argumen yang dibutuhkan untuk jenis plot tertentu telah disediakan dengan benar. Jika argumen tidak diberikan dengan benar, fungsi ini akan mencetak pesan kesalahan.

3.4 Streamlit *Coil Impedance Prediction*



Gambar 3.12 *Flow Chart Streamlit Application*

Aplikasi didesain untuk menerima input data dimensi trafo dan melakukan validasi data yang dimasukkan sesuai dengan beberapa aturan seperti *Outer Diameter Length LV < Inner Diameter Length HV.* Kemudian aplikasi akan memprediksi nilai impedansi dan melakukan pendataan hasil prediksi pada *google sheet*, kemudian menampilkan pengaruh parameter terhadap penyimpangan impedansi. Parameter lain juga dikembangkan untuk melihat relevansi penyesuaian parameter terhadap penyimpangan impedansi. *Prediction suggestion* dan *prediction advance* didesain agar dapat memberikan saran modifikasi ukuran parameter dimensi trafo. Program tersebut akan melakukan iterasi untuk mendapatkan hasil impedansi terbaik.

```
# python==3.11.5
streamlit
pandas
numpy
matplotlib
shap
st-gsheets-connection
xgboost
ipython

# pickle 5 have been installed within python
# time have been included within python
# python version recommendation 3.11.5 more than that cannot
# on streamlit dont need python==3.11.5
```

Gambar 3.13 Inisiasi *Library* pada Streamlit

Gambar 3.13 adalah isi dari *file requirements.txt* pada GitHub. Aplikasi ini menggunakan Python versi 3.11.5 dengan beberapa *library* penting untuk analisis data dan *training* mesin. *Library* utama yang digunakan adalah Streamlit (Snowflake Inc., 2024) untuk pembuatan aplikasi web interaktif, Pandas untuk manipulasi dan analisis data, NumPy untuk komputasi numerik, Matplotlib untuk visualisasi data, SHAP untuk interpretasi model pembelajaran mesin, '*st-gsheets-connection'* untuk menghubungkan dan mengelola data dari Google Sheets, serta *XGBoost* untuk algoritma *training machine learning* berbasis *gradient boosting*, sementara *Ipython* digunakan untuk lingkungan interaktif yang mendukung

eksperimen dan pengembangan. *Library Pickle5* sudah terinstal untuk serialisasi objek, dan *Time* telah disertakan untuk manajemen waktu dalam aplikasi. Sementara untuk versi *Python* pada penggunaan *Streamlit* tidak memerlukan spesifikasi tertentu.

```
conn = st.connection("gsheets", type=GSheetsConnection)

dataset_impedance = conn.read(
      worksheet="dataset_impedance",
      ttl=0,
      usecols=[i for i in range(0, 12)]
   ).dropna()

with open('models/xgb.pkl', 'rb') as file:
   modelImpedance = pickle.load(file)

# Load Dataset Impedance
   dataset = pd.DataFrame(dataset_impedance)
   dataset_impedance = dataset[modelImpedance.feature_names_in_]
```

Gambar 3.14 Persiapan Koneksi, *Google Sheet*, dan Model *Machine Learning*

Kode di atas menjelaskan proses membaca data dari Google Sheets, memuat model yang sudah disimpan, dan menyiapkan dataset untuk prediksi. Pertama, koneksi ke Google Sheets dibuat menggunakan *Streamlit* dengan tipe koneksi *GSheetsConnection*. Kemudian, data dari worksheet bernama "dataset_*impedance*" dibaca dengan mengambil kolom dari indeks 0 hingga 11, dan baris yang memiliki nilai kosong dihapus menggunakan '*.dropna*()'. Setelah itu, model *XGBoost* yang telah disimpan sebelumnya dalam file xgb.pkl dimuat menggunakan *pickle*. Terakhir, dataset diubah menjadi objek *DataFrame* dan disesuaikan agar hanya mencakup kolom-kolom yang digunakan oleh model untuk melakukan prediksi, yang ditentukan oleh atribut '*feature*_names_in_' dari model tersebut.

3.4.1 Prediction x SHAP

```
if submitted:
    status = st.status('Processing...', expanded=True)

    # Load Dataset Impedance
    dataset = pd.DataFrame(dataset_impedance)
    dataset_impedance = dataset[modelImpedance.feature_names_in_]

    # ! INPUT DATA
    input_data = pd.DataFrame(params, index=[0])
    test_data_imp = input_data[modelImpedance.feature_names_in_]
    st.subheader('Input Data')
    st.dataframe(test_data_imp)

    # ! PREDICT IMPEDANCE
    with status:
        st.write('Calculating Initial Impedance')
    impedance = predictImpedance(test_data_imp)
    st.subheader('Result')
    st.write(f'The predicted impedance is {impedance:.2f} ohms')

    plot_on = 0
    # ! SHAP VISUALIZATION
    with status:
        st.write('Calculating Initial SHAP Values.')
    st.subheader('SHAP Visualization Impedance')
    shap_values = visualize(modelImpedance, test_data_imp, plot_on, dataset_impedance)
```

Gambar 3.15 *Input* Data dan Prediksi Impedansi

Prediction x SHAP merupakan salah satu fitur yang dibuat oleh penulis dimana menghasilkan prediksi impedansi dan SHAP *values* atau pengaruh masing-masing parameter yang berkontribusi pada nilai impedansi yang diprediksi. Ketika pengguna mengirimkan data melalui aplikasi *Streamlit* (ditandai dengan *variabel submitted*), status proses ditampilkan

dengan pesan "*Processing*...". Data dari *Google Sheets* yang telah dibaca sebelumnya diubah menjadi *DataFrame* dan disesuaikan agar hanya mencakup kolom-kolom yang dibutuhkan oleh model *XGBoost*. Data input dari pengguna kemudian diubah menjadi *DataFrame* dan dipilih kolom-kolom yang sesuai dengan yang digunakan oleh model. Data input ini ditampilkan pada antarmuka pengguna. Selanjutnya, proses prediksi dilakukan dengan menghitung nilai impedansi menggunakan fungsi '*predictImpedance*' dan hasil prediksi ditampilkan dengan format yang rapi. Kemudian, untuk memberikan interpretasi terhadap prediksi, nilai SHAP (*SHapley Additive exPlanations*) dihitung dan divisualisasikan menggunakan fungsi '*visualize*'. Seluruh proses tersebut dijalankan sambil memperbarui status di antarmuka *Streamlit* untuk memberikan informasi kepada pengguna tentang tahapan yang sedang berlangsung.

```
# List of columns to modify
    columns = test_data_imp.columns.tolist()


    st.write("---")
    # Modify each column
    # column string
    for column in columns:
        with status:
            st.write(f'Calculating {column}: {input_data.iloc[0, 12]:.2f}')
        modify_and_restore(test_data_imp, column, input_data.iloc[0, 12], impedance)

    st.subheader("Summary Table")
    st.dataframe(summary.style.highlight_max(axis=0))

    with status:
        status.update(label="Process Complete!", state="complete", expanded=False)
        time.sleep(1)
```

Gambar 3.16 Lanjutan Prediksi Impedansi dan Modifikasi

Setelah prediksi awal impedansi dilakukan, aplikasi *Streamlit* akan memodifikasi setiap kolom dalam data input pengguna (*test_data_imp*). Pertama, daftar kolom yang akan dimodifikasi diambil dan disimpan dalam variabel columns. Untuk setiap kolom dalam daftar

tersebut, aplikasi memperbarui status dengan menampilkan pesan yang menyatakan kolom yang sedang dihitung beserta nilai yang terkait (diperoleh dari *input*_data). Fungsi '*modify_and_restore'* kemudian dipanggil untuk setiap kolom, yang bertugas melakukan modifikasi pada data input dan menghitung ulang nilai impedansi berdasarkan perubahan tersebut. Setelah semua kolom dimodifikasi dan perhitungan ulang selesai, aplikasi menampilkan tabel ringkasan (*summary*) dengan highlight pada nilai maksimum dalam setiap baris. Terakhir, status diperbarui untuk menunjukkan bahwa proses telah selesai dengan pesan "*Process Complete*!" dan status diperluas kembali selama satu detik sebelum berakhir.

```
def visualize(model, test_data, plot_on, dataset=None):

    @st.cache_resource
    def init_shap_js():
        shap.initjs()

    # ! SHAP PREPARATION
    init_shap_js()
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(test_data)
    shap_values_dataset = explainer.shap_values(dataset)

    shap_values_df = pd.DataFrame(shap_values, columns=[str(col+' Shap') for col in
test_data.columns])
    shap_values_df.index = test_data.index
    st.write(shap_values_df)
    explanation = shap.Explanation(shap_values, data=test_data,
feature_names=test_data.columns)
```

Gambar 3.17 Fungsi Visualisasi SHAP

Fungsi '*visualize'* dirancang untuk menghitung dan menampilkan nilai SHAP (*Shapley Additive exPlanations*) dalam aplikasi *Streamlit*. Pada bagian pertama, terdapat dekorator '*@st.cache_resource'* yang digunakan untuk menyimpan hasil dari fungsi '*init_shap_js'* agar tidak dihitung ulang setiap kali halaman Streamlit di-refresh. Fungsi '*init_shap_js'*

menginisialisasi *JavaScrip*t SHAP yang diperlukan untuk menampilkan plot SHAP interaktif. Sebuah *TreeExplainer* dari pustaka SHAP dibuat menggunakan model yang diberikan, dan nilai-nilai SHAP untuk data pengujian (*test_data*) dan dataset (jika diberikan) dihitung. Nilai-nilai SHAP ini kemudian diubah menjadi *DataFrame* dengan kolom yang dinamai sesuai dengan parameter-parameter data ditambah dengan akhiran "*Shap*". DataFrame nilai-nilai SHAP ditampilkan di aplikasi *Streamlit*.

```
    # ! SHAP PLOT
    if(plot_on == 1):
        with st.popover("Open Shap Plot"):
            shap_plot(plot_on, shap_values_dataset, dataset, shap_values, test_data,
explainer=explainer, explanation=explanation, show=False)
    else:
        shap_plot(plot_on, shap_values_dataset, dataset, shap_values, test_data,
explainer=explainer, explanation=explanation, show=False)


    return shap_values_df
```

Gambar 3.18 Fungsi Visualisasi SHAP Lanjutan

Selanjutnya, sebuah objek Explanation dari SHAP dibuat untuk data pengujian, menyertakan nilai-nilai SHAP, data asli, dan nama-nama parameter. Jika *flag plot*_on disetel ke 1, plot SHAP akan ditampilkan dalam popover *Streamlit*; jika tidak, plot akan ditampilkan secara langsung dengan memanggil fungsi *shap_plot*. Akhirnya, *DataFrame* nilai-nilai SHAP dikembalikan dari fungsi *visualize*, memungkinkan penggunaan lebih lanjut dalam aplikasi.

```
def shap_plot(plot_on, shap_val_dataset, dataset_data, shap_val, feature_data,
explainer=None, explanation=None, show=True):

   if(plot_on == 1):
      bar, waterfall = st.tabs(['Bar', 'Waterfall'])
   else:
      summary, bar, waterfall = st.tabs(['Summary', 'Bar', 'Waterfall'])

   # Summary from Dataset
   if(plot_on == 0):
      with summary:
         st.header('Summary Plot')
         plt.figure(figsize=(10, 8))

         shap.summary_plot(shap_val_dataset, features=dataset_data,
feature_names=dataset_data.columns, show=show)

         plt.tight_layout()
         st.pyplot(plt.gcf())
         plt.clf()  # Membersihkan plot setelah ditampilkan
         plt.close()  # Menutup plot agar tidak ada sisa plot)
```

Gambar 3.19 Fungsi *shap_plot*

Fungsi '*shap_plot*' bertanggung jawab untuk menghasilkan dan menampilkan plot SHAP berdasarkan nilai-nilai SHAP yang telah dihitung. Fungsi ini menerima beberapa parameter, termasuk indikator *plot_on* yang menentukan apakah hanya beberapa tab plot yang ditampilkan atau semuanya, nilai-nilai SHAP untuk dataset penuh (*shap_val_dataset*), data asli (*dataset_data*), nilai-nilai SHAP untuk parameter data tertentu (*shap_val*), parameter data (*feature_data*), serta objek *explainer* dan *explanation* dari SHAP yang bersifat opsional. Jika *plot_on* disetel ke 1, hanya tab *'Bar'* dan *'Waterfall'* yang dibuat. Jika tidak, tab *'Summary'*, *'Bar'*, dan *'Waterfall'* semuanya dibuat.

Pada tab *summary*, jika *plot_on* disetel ke 0, sebuah plot ringkasan SHAP ditampilkan. Dalam konteks ini, fungsi menggunakan '*shap.summary*'_plot untuk menghasilkan plot yang menunjukkan pentingnya parameter-parameter berdasarkan nilai-nilai SHAP dalam dataset. Plot ini dibuat dalam sebuah figure dengan ukuran 10x8 inci. Fungsi '*plt.tight_layout()*' digunakan untuk memastikan tata letak plot rapi, dan '*st.pyplot(plt.gcf())*' menampilkan plot dalam aplikasi *Streamlit*. Setelah plot ditampilkan, fungsi '*plt.clf()*' dan '*plt.close()*' dipanggil untuk membersihkan dan menutup plot, mencegah adanya plot sisa yang bisa mengganggu plot berikutnya. Selanjutnya akan ditampilkan juga plot dalam bentuk bar dan waterfall.

```
def modify_and_restore(df, column, adjusted, ori_imp):

    st.subheader(f"{column} {adjusted}")

    original_value = df.at[0, column]
    df.at[0, column] = original_value + adjusted
    st.dataframe(df)

    # ! PREDICT IMPEDANCE
    impedance = predictImpedance(df)
    st.write(f'The predicted impedance is {impedance:.2f} ohms')

    plot_on = 1
    shap_values = visualize(modelImpedance, df, plot_on, dataset_impedance)
    save = pd.DataFrame(shap_values, index=[0])
    save['ADJUSTED'] = [column + str(adjusted)]
    save['Difference'] = [ori_imp - impedance]
    save['Original Impedance'] = [ori_imp]
    save['Impedance'] = impedance

    global summary
    summary = pd.concat([summary, save], ignore_index=True)

    # ! INSERT DATA TO DB
    insert_data_adjusted(save)

    df.at[0, column] = original_value
```

Gambar 3.20 Fungsi *modify_and_restore*

Fungsi '*modify_and_restore*' menerima empat parameter: *df* (*DataFrame* yang berisi *data input*), *column* (nama kolom yang akan dimodifikasi), *adjusted* (nilai penyesuaian yang akan ditambahkan ke kolom), dan *ori_imp* (nilai impedansi asli sebelum penyesuaian). Pertama, sebuah *subheader* dengan nama kolom dan nilai penyesuaian ditampilkan di aplikasi *Streamlit*. Nilai asli dari kolom yang akan dimodifikasi disimpan dalam variabel *original_value*, kemudian nilai kolom tersebut diperbarui dengan menambahkan nilai penyesuaian. *DataFrame* yang telah diperbarui ini ditampilkan di aplikasi Streamlit.

Selanjutnya, prediksi impedansi baru dihitung dengan memanggil fungsi *predictImpedance* menggunakan *DataFrame* yang telah dimodifikasi, dan hasil prediksi ditampilkan. Visualisasi SHAP untuk *DataFrame* yang telah dimodifikasi dilakukan dengan memanggil fungsi visualize, di mana nilai plot_*on* disetel ke 1 untuk menampilkan plot. Nilai-nilai SHAP yang dihasilkan disimpan dalam *DataFrame* save, bersama dengan informasi tentang

penyesuaian yang dilakukan, perbedaan antara impedansi asli dan baru, serta nilai-nilai impedansi tersebut.

DataFrame '*save*' kemudian digabungkan dengan *DataFrame* summary global untuk menyimpan hasil penyesuaian ini. Hasil penyesuaian ini juga dimasukkan ke dalam basis data dengan memanggil fungsi *insert_data_adjusted*. Terakhir, nilai kolom yang telah dimodifikasi dikembalikan ke nilai aslinya untuk memastikan bahwa *DataFrame* kembali ke keadaan semula sebelum modifikasi dilakukan, memungkinkan penyesuaian berikutnya dilakukan dengan basis data yang konsisten.

3.4.2 Prediction Suggestion

```
# List of columns to modify
   columns = test_data_imp.columns.tolist()

   for i in range (-10, 11):
      with status:
         st.write(f'Calculating Range: {i:.2f}')

      # Modify each column
      for column in columns:
         modify_testing(test_data_imp, column, i, impedance)

   save_imp = pd.DataFrame(suggestion_imp)
   save_tol = pd.DataFrame(suggestion_tol)

   st.subheader("Suggestion Impedance")
   st.dataframe(save_imp.style.highlight_max(axis=0))
```

Gambar 3.21 *Prediction Suggestion Column*

Predisction suggestion akan memodifikasi parameter dan memprediksi nilai impedansi dari parameter-parameter yang dimodifikasi dengan aturan tertentu. Awalnya fungsi akan menghitung impedansi seperti pada fungsi sebelumnya hanya saja setelah itu, daftar kolom akan dimodifikasi dari *test_data_imp* dan disimpan dalam variabel columns. Selanjutnya, untuk setiap nilai dalam rentang dari -10 hingga 10, sebuah loop dijalankan untuk melakukan modifikasi dan perhitungan prediksi. Di dalam loop ini, status ditampilkan di aplikasi *Streamlit* dengan pesan yang menunjukkan nilai rentang saat ini yang sedang dihitung.

Untuk setiap nilai dalam rentang tersebut, setiap kolom dalam daftar columns dimodifikasi satu per satu dengan memanggil fungsi *modify_testing*. Fungsi ini bertugas untuk

melakukan penyesuaian pada kolom yang ditentukan, menghitung ulang prediksi impedansi berdasarkan nilai yang disesuaikan, dan menyimpan hasilnya. Nilai i yang digunakan dalam loop sebagai nilai penyesuaian ditambahkan atau dikurangi dari nilai asli kolom, dan prediksi impedansi dihitung untuk setiap penyesuaian.

Setelah *loop* selesai dan semua modifikasi serta perhitungan prediksi dilakukan, hasil modifikasi disimpan dalam *DataFrame save_imp* untuk sugesti impedansi dan '*save_tol'* untuk sugesti toleransi (jika ada). *DataFrame save_imp* kemudian ditampilkan dalam aplikasi *Streamlit* dengan *subheader* "*Suggestion Impedance*", menggunakan metode '*style. highlight_max(axis=0)*' untuk menyoroti nilai maksimum di setiap baris, membantu pengguna untuk dengan mudah mengidentifikasi hasil prediksi yang paling signifikan.

```
suggestion_imp = []

def modify_testing(df, column, adjusted, ori_imp):
    original_value = df.at[0, column]
    df.at[0, column] = original_value + adjusted

    # ! PREDICT IMPEDANCE
    impedance = predictImpedance(df)


    if (ori_imp > 0):
        if ((ori_imp - impedance) > 0):
            suggestion_imp.append({
                'Adjusted Column' : column,
                'Adjusted Value' : adjusted,
                'Adjusted' : f"{column}{adjusted}",
                'Difference Impedance' : ori_imp - impedance,
                'Original Impedance' : ori_imp,
                'Adjusted Impedance' : impedance,
            })
    else:
        if ((impedance - ori_imp) > 0):
            suggestion_imp.append({
                'Adjusted Column' : column,
                'Adjusted Value' : adjusted,
                'Adjusted' : f"{column}{adjusted}",
                'Difference Impedance' : impedance - ori_imp,
                'Original Impedance' : ori_imp,
                'Adjusted Impedance' : impedance,
            })

    df.at[0, column] = original_value

    return
```

Gambar 3.22 Fungsi *modify_testing*

Fungsi *modify_testing* menerima empat parameter: *df* (*DataFrame* yang berisi data input), *column* (nama kolom yang akan dimodifikasi), *adjusted* (nilai penyesuaian yang akan ditambahkan ke kolom), dan *ori_imp* (nilai impedansi asli sebelum penyesuaian). Pertama, nilai asli dari kolom yang akan dimodifikasi disimpan dalam variabel *original_value*, kemudian nilai kolom tersebut diperbarui dengan menambahkan nilai penyesuaian.

Selanjutnya, fungsi '*predictImpedance*' dipanggil untuk menghitung prediksi impedansi baru berdasarkan *DataFrame* yang telah dimodifikasi. Jika *ori_imp* lebih besar dari 0 dan perbedaan antara impedansi asli dan baru (impedansi asli dikurangi impedansi baru) lebih

besar dari 0, atau jika *ori_imp* kurang dari atau sama dengan 0 dan perbedaan antara impedansi baru dan asli (impedansi baru dikurangi impedansi asli) lebih besar dari 0, maka hasil penyesuaian tersebut disimpan dalam daftar *suggestion_imp*. Informasi yang disimpan meliputi kolom yang disesuaikan, nilai penyesuaian, deskripsi penyesuaian, perbedaan impedansi, impedansi asli, dan impedansi setelah penyesuaian.

Setelah menyimpan hasil penyesuaian, nilai kolom yang telah dimodifikasi dikembalikan ke nilai aslinya untuk memastikan bahwa *DataFrame* kembali ke keadaan semula sebelum modifikasi dilakukan. Fungsi ini memastikan bahwa setiap penyesuaian dilakukan secara independen dan tidak mempengaruhi penyesuaian berikutnya.

Daftar *suggestion_imp* diinisialisasi di luar fungsi, sehingga dapat digunakan untuk mengumpulkan semua saran hasil penyesuaian dari berbagai panggilan fungsi *modify_testing* dalam *loop*. Hasil akhir dari penyesuaian ini kemudian dapat ditampilkan dan dianalisis lebih lanjut di aplikasi *Streamlit*.

3.4.3 Prediction Advance

```
# Define column names and ranges
   # List of columns to modify
   columns = test_data_imp.columns.tolist()

   # List of checkbox states
   checkboxes = [A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11]

   columns = columns_checker(columns)
   ranges = range(input_data.iloc[0, 12], input_data.iloc[0, 13] + 1)

   columns_edited = []
   x = 0
   # Call the function
   modify_and_test_v1(test_data_imp, columns,
            ranges, impedance, x, columns_edited)

   modify_and_test_v3(test_data_imp, columns,
            ranges, impedance, x, columns_edited)

   num_columns = len(columns)
   # Call the function
   modify_and_test_v2(test_data_imp, columns,
            ranges, impedance, num_columns - 1, columns_edited)

   suggestion=suggestion.drop_duplicates(subset=['Inner Diameter Length LV', 'Inner
Diameter Width LV', 'Inner Diameter Height LV',
                        'Outer Diameter Length LV', 'Outer Diameter Width LV', 'Outer
Diameter Height LV',
                        'Inner Diameter Length HV', 'Inner Diameter Width HV', 'Inner
Diameter Height HV',
                        'Outer Diameter Length HV', 'Outer Diameter Width HV', 'Outer
Diameter Height HV'], keep='last')

   st.dataframe(suggestion)

   insert_data(suggestion)
```

Gambar 3.23 Kode Program *Prediction Advance*

Prediction advance merupakan fitur yang sudah dikembangkan dari *prediction suggestion* dimana dapat memodifikasi parameter dengan tingkat variasi lebih tinggi sehingga dapat menghasilkan saran perbaikan lebih banyak. Pertama, daftar kolom yang akan dimodifikasi diambil dari *test_data_imp* dan disimpan dalam *variabel columns*. Daftar status *checkbox* juga disiapkan, dengan masing-masing *checkbox* menunjukkan apakah kolom terkait

akan dimodifikasi atau tidak. Fungsi *columns_checker* dipanggil untuk memfilter kolom berdasarkan status *checkbox* yang dipilih, memastikan hanya kolom yang diinginkan yang akan dimodifikasi.

Selanjutnya, rentang nilai yang akan digunakan untuk modifikasi diambil dari dua nilai dalam *input_data* (nilai pada indeks 12 dan 13), menghasilkan rentang nilai yang akan digunakan untuk penyesuaian kolom. Variabel *columns_edited* diinisialisasi sebagai daftar kosong untuk melacak kolom yang telah dimodifikasi, dan *variabel x* diinisialisasi ke nol untuk digunakan sebagai penghitung atau indeks.

Tiga fungsi berbeda (*modify_and_test_v1, modify_and_test_v2, dan modify_and_test_v3*) dipanggil untuk melakukan penyesuaian dan pengujian prediksi impedansi berdasarkan kolom dan rentang nilai yang telah ditentukan. Setiap fungsi ini menerima *DataFrame*, kolom yang akan dimodifikasi, rentang nilai penyesuaian, nilai impedansi asli, serta variabel lain yang relevan untuk pelacakan dan pengeditan kolom.

Setelah semua modifikasi dan perhitungan prediksi selesai, *DataFrame suggestion* difilter untuk menghapus duplikasi, mempertahankan hanya baris terakhir untuk setiap kombinasi unik dari dimensi terkait impedansi. Hasil akhir dari penyesuaian ini ditampilkan dalam aplikasi *Streamlit* menggunakan *st.dataframe*(*suggestion*).

Akhirnya, hasil penyesuaian disimpan ke dalam database dengan memanggil fungsi *insert_data*(*suggestion*), memastikan bahwa data yang dihasilkan dari proses modifikasi ini tersimpan untuk analisis lebih lanjut atau penggunaan di masa mendatang.

```
def calculating(df, ori_imp, columns_adjusted):

    impedance = predictImpedance(df)

    global suggestion

    if (ori_imp > 0):
        if ((ori_imp - impedance) > 0):
            save = pd.DataFrame(df, index=[0])
            save['Original Impedance'] = ori_imp
            save['Predicted Impedance'] = impedance
            save['Difference'] = [ori_imp - impedance]
            save['Columns Adjusted'] = [columns_adjusted]

            suggestion = pd.concat([suggestion, save], ignore_index=True)
    else:
        if ((impedance - ori_imp) > 0):
            save = pd.DataFrame(df, index=[0])
            save['Original Impedance'] = ori_imp
            save['Predicted Impedance'] = impedance
            save['Difference'] = [impedance - ori_imp]
            save['Columns Adjusted'] = [columns_adjusted]

            suggestion = pd.concat([suggestion, save], ignore_index=True)


    return df
```

Gambar 3.24 Fungsi *Calculating*

Fungsi *calculating* menerima tiga parameter: *df* (*DataFrame* yang berisi *data input* yang telah dimodifikasi), *ori_imp* (nilai impedansi asli sebelum penyesuaian), dan *columns_adjusted* (daftar kolom yang telah disesuaikan). Pertama, prediksi impedansi baru dihitung dengan memanggil fungsi '*predictImpedance*' menggunakan *DataFrame* yang telah dimodifikasi.

Kemudian, hasil prediksi impedansi dibandingkan dengan nilai asli impedansi. Jika nilai *ori_imp* lebih besar dari 0 dan perbedaan antara impedansi asli dan baru (impedansi asli dikurangi impedansi baru) lebih besar dari 0, atau jika '*ori_imp'* kurang dari atau sama dengan 0 dan perbedaan antara impedansi baru dan asli (impedansi baru dikurangi impedansi asli) lebih besar dari 0, maka hasil penyesuaian tersebut dianggap relevan dan disimpan dalam *DataFrame suggestion*.

```
def modify_and_test_v1(df, columns, value_range, impedance, x, columns_edited):
    num_columns = len(columns)

    for i in value_range:
        if(columns[x] == columns[0]):
            with status:
                st.write(f'Calculating Layer {x}: {columns[x]} {i:.2f}')

        columns_original = columns_edited
        columns_edited.append(columns[x]+str(i))

        original_value1 = df.at[0, columns[x]]
        df.at[0, columns[x]] = df.at[0, columns[x]] + i
        calculating(df, impedance, columns_edited)

        x += 1

        if x < num_columns:
            modify_and_test_v1(df, columns, value_range, impedance, x, columns_edited)

        x -= 1

        reset_value(df, columns[x], original_value1)
        calculating(df, impedance, columns_edited)

        columns_edited = columns_original
```

Gambar 3.25 Fungsi Modifikasi Data *Input*

Fungsi *modify_and_test_v1* menerima beberapa parameter: *df (DataFrame* yang berisi *data input*), *columns* (daftar kolom yang akan dimodifikasi), *value_range* (rentang nilai penyesuaian), *impedance* (nilai impedansi asli sebelum penyesuaian), x (penghitung yang menunjukkan iterasi saat ini), dan *columns_edited* (daftar kolom yang telah disesuaikan).

Pertama, jumlah kolom yang akan dimodifikasi dihitung dan disimpan dalam variabel *num_columns*. Selanjutnya, untuk setiap nilai dalam rentang yang diberikan, sebuah pernyataan kondisional digunakan untuk menampilkan status iterasi saat ini di aplikasi *Streamlit* menggunakan status. Pada iterasi pertama untuk setiap *layer*, status ditampilkan yang menyertakan nama kolom dan nilai penyesuaian saat ini.

Variabel *columns_original* digunakan untuk menyimpan salinan daftar *columns_edited* saat ini sebelum modifikasi dilakukan, untuk digunakan kembali nanti setelah selesai menguji semua nilai dalam rentang tertentu. Kolom yang telah disesuaikan dengan nilai penyesuaian saat ini ditambahkan ke dalam *columns_edited*.

Nilai asli dari kolom yang akan dimodifikasi disimpan dalam *original_value1*, kemudian nilai kolom tersebut diperbarui dengan menambahkan nilai penyesuaian. Fungsi *calculating* kemudian dipanggil untuk menghitung ulang prediksi impedansi berdasarkan *DataFrame* yang telah dimodifikasi, dan hasil perhitungan disimpan dalam *DataFrame suggestion*.

Variabel x yang digunakan sebagai penghitung iterasi diperbarui, kemudian fungsi *modify_and_test_v1* dipanggil secara rekursif dengan nilai x yang baru. Proses ini diulangi untuk setiap kolom yang akan dimodifikasi, dengan nilai x yang bertambah setiap kali. Setelah semua iterasi selesai dilakukan untuk satu nilai rentang, nilai asli kolom yang telah dimodifikasi dikembalikan ke nilai semula, dan hasil perhitungan ulang prediksi impedansi dilakukan. Daftar *columns_edited* juga dikembalikan ke nilai semula untuk memulai iterasi berikutnya dengan kolom yang benar-benar baru.