LAMPIRAN

Lampiran 01. Informasi Pelabel Data

INFORMASI PELABEL DATA

Pelabel 1

Nama	Ni Luh Putu Apriantini, S.Pd., M.Pd.
Gelar Pendidikan	Sarjana Pendidikan Bahasa dan Sastra Indonesia
Perguruan Tinggi	Universitas Mahasaraswati Denpasar
Lulus Tahun	2011

Pelabel 2

Nama	I Gede Rai Widana, S.Pd.
Gelar Pendidikan	Sarjana Pendidikan Bahasa dan Sastra Indonesia
Perguruan Tinggi	Universitas Pendidikan Ganesha
Lulus Tahun	2024

Lampiran 02. *Source Code* Penelitian

```
# Data crawling
filename = 'ttshop_dibukaKembali.csv'
search_keyword = 'tiktok shop until:2024-01-12 since:2023-12-12
lang:id -filter:links -filter:replies'

limit = 20000

!npx --yes tweet-harvest@latest -o "{filename}" -s
"{search_keyword}" -l {limit} –token {twitter_auth_token}
```

L1. *Source Code Data crawling*

```
# Penghapusan Missing Value
df.isnull().sum()
df = df.dropna()
```

L2. *Source Code* Penghapusan *Missing Value*

```
# Data Cleaning
# Penghapusan link atau tautan
def clean_link(text):
    link_pattern = r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-
_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+'

    without_link = re.sub(link_pattern, '', text)
    return without_link

df['full_text'] = df['full_text'].apply(clean_link)

# Penghapusan Simbol
def clean_tweets(text):
  text = re.sub(r'@[A-Za-z0-9_]+', '', text)
  text = re.sub(r'#\w+', '', text)
  text = re.sub(r'RT[\s]+', '', text)
  text = re.sub(r'https ?: //\S+', '', text)

  text = re.sub(r'[^A-Za-z0-9 ]', '', text)
  text = re.sub(r'\s+', ' ', text). strip()
  text = re.sub(r'\d+', ' ', text). strip()

  return text

df['full_text'] = df['full_text'].apply(clean_tweets)
df_clean1 = df
df_clean1
```

L3. *Source Code Data Cleaning*

```
# Casefolding
df_clean1['full_text'] = df_clean1['full_text'].str.lower()
df_Casefolding = df_clean1
```

L4. *Source Code Casefolding*

```
#Penghapusan Data Duplikat
df_removeDuplicates =
df_Casefolding.drop_duplicates(subset=['full_text'])
df_removeDuplicates.duplicated().sum()
```

L5. *Source Code* Penghapusan Data Duplikat

```
#Normalisasi
def add_spaces(text):
    return f" {text} "

def normalization(str_text):
    for i in norm:
        str_text = str_text.replace(i, norm[i])
    return str_text

df_normalize['full_text'] =
df_normalize['full_text'].apply(add_spaces)

df_normalize["full_text"] =
df_normalize["full_text"].apply(lambda x: normalization(x))

df_normalize['full_text'] =
df_normalize['full_text'].str.strip()

df_normalize
```

L6. *Source Code* Normalisasi

```
#Penghapusan Stopwords
stop_words = StopWordRemoverFactory().get_stop_words()

if 'tidak' in stop_words:
    stop_words.remove('tidak')

new_array = ArrayDictionary(stop_words)
stop_words_remover_new = StopWordRemover(new_array)

def stopword(str_text):
  if isinstance(str_text, str):
    str_text = stop_words_remover_new.remove(str_text)
  return str_text
```

```
df_Stopwords['full_text'] =
df_Stopwords['full_text'].apply(stopword)
df_Stopwords
```

L7. *Source Code* Penghapusan *Stopwords*

```
#Tokenisasi
df_tokenize = df_Stopwords

df_tokenize ['full_text'] = df_tokenize
['full_text'].apply(lambda x: x.split())
df_tokenize
```

L8. *Source Code* Tokenisasi

```
#Stemming
def Stemming(text_cleaning):
  factory = StemmerFactory()
  stemmer = factory.create_stemmer()
  do = []
  for w in text_cleaning:
    dt = stemmer.stem(w)
    do.append(dt)
  d_clean = []
  d_clean =" ".join(do)
  print(d_clean)
  return d_clean

df_Stemming['full_text'] =
df_Stemming['full_text'].apply(Stemming)
```

L9. *Source Code Stemming*

```
#Labeling Menggunakan Pretrained Model
from transformers import pipeline, AutoTokenizer,
AutoModelForSequenceClassification

pretrained = "mdhugol/indonesia-bert-sentiment-classification"
model =
AutoModelForSequenceClassification.from_pretrained(pretrained)
tokenizer = AutoTokenizer.from_pretrained(pretrained)

sentiment_analysis = pipeline("sentiment-analysis", model=model,
tokenizer=tokenizer)

def label_text(text):
    results = sentiment_analysis(text)
    label_index = {'LABEL_0': 'Positive', 'LABEL_1': 'Neutral',
'LABEL_2': 'Negative'}
```

```
    labels = []
    for result in results:
        labels.append(label_index[result['label']])
    return labels
df['label'] = df['full_text'].apply(label_text)
```

L10. *Source Code Labeling* dengan *Pretrained Model*

```
#Ekstraksi Fitur
# Mengonversi label teks menjadi numerik
label_encoder = LabelEncoder()
df['label_numerical'] = label_encoder.fit_transform(df['label'])

# Menggunakan TfidfVectorizer dengan Unigram
tfidf = TfidfVectorizer(ngram_range=(1, 1))  max_features=5000
X_tfidf = tfidf.fit_transform(df_labelNumerical['full_text'])
y = df_labelNumerical['label_numerical']
```

L11. *Source Code* Ekstraksi Fitur

```
# Pembagian Data
X_dense = X_tfidf.todense()
X = np.asarray(X_dense)

with open(file_path + 'model/dataset.pkl', 'wb') as f:
    pickle.dump((X, y), f)

# Split data dengan rasio 80:20
X_train_80, X_test_20, y_train_80, y_test_20 = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=72,
    shuffle=True
)

# Split data dengan rasio 90:10
X_train_90, X_test_10, y_train_90, y_test_10 = train_test_split(
    X,
    y,
    test_size=0.1,
    random_state=72,
    shuffle=True
)
```

L12. *Source Code* Pembagian Data

```
# Oversampling dengan ADASYN
ada = ADASYN()
X_train_ADASYN_80, y_train_ADASYN_80 =
ada.fit_resample(X_train_80, y_train_80)
X_train_ADASYN_90, y_train_ADASYN_90 =
ada.fit_resample(X_train_90, y_train_90)
```

L13. *Source Code Oversampling* dengan ADASYN

```
# Fungsi Klasifikasi
DEFAULT_N_SPLIT = 10

def evaluation_Training(model, X_train, y_train, split_desc):
    # Pelatihan model
    model.fit(X_train, y_train)
    print(f"\nTraining Cross Validation Report ({split_desc}):")

# Cross Validation
def cross_val_Training(model, X_train, y_train,
n_splits=DEFAULT_N_SPLIT, use_ADASYN=False):
    kf = KFold(n_splits=n_splits, shuffle=True)

    X_train = np.array(X_train)
    y_train = np.array(y_train)

    fold_accuracy = []

    for fold, (train_index, val_index) in
enumerate(kf.split(X_train)):
        X_fold_train, X_fold_val = X_train[train_index],
X_train[val_index]
        y_fold_train, y_fold_val = y_train[train_index],
y_train[val_index]

        if use_ADASYN:
            adasyn = ADASYN()
            X_fold_train, y_fold_train =
adasyn.fit_resample(X_fold_train, y_fold_train)

        cloned_model = clone(model)
        cloned_model.fit(X_fold_train, y_fold_train)

        y_fold_val_pred = model.Predict(X_fold_val)
        accuracy = accuracy_score(y_fold_val, y_fold_val_pred)
        fold_accuracy.append(accuracy)
        print(f"Accuracy (Fold {fold + 1}): {accuracy:.4f}")

    mean_accuracy = np.mean(fold_accuracy)
    std_accuracy = np.std(fold_accuracy)
    print(f"\nMean Accuracy: {mean_accuracy:.4f}")
    print(f"Standard Deviation of Accuracy: {std_accuracy:.4f}")

def evaluation_Testing(model, X_test, y_test, split_desc):
```

```
    # Prediksi dengan data uji
    y_test_pred = model.Predict(X_test)

    # Laporan klasifikasi pengujian
    print(f"\nTesting Classification Report ({split_desc}):")
    print(classification_report(y_test, y_test_pred, digits=4))

    # Confusion Matrix
    cm_test = confusion_matrix(y_test, y_test_pred)
    print(f"\nConfusion Matrix ({split_desc}):")
    print(cm_test)
```

L14. *Source Code* Fungsi Klasifikasi

```
# Training SVM Linear
# Inisialisasi model SVM Linear
params_linear = {
    'C': 1,
    'kernel': 'linear'
}
svm_linear = SVC(**params_linear)

# Model tanpa ADASYN
# Linear OvO 80:20
print("\nModel L1 (Linear, OvO, Split 80:20):")
l1 = OneVsOneClassifier(svm_linear)
evaluation_Training(l1, X_train_80, y_train_80, "L1 (Linear,
OvO, 80:20)")
cross_val_Training(l1, X_train_80, y_train_80, use_ADASYN=False)

# Linear OvO 90:10
print("\nModel L2 (Linear, OvO, Split 90:10):")
l2 = OneVsOneClassifier(svm_linear)
evaluation_Training(l2, X_train_90, y_train_90, "L2 (Linear,
OvO, 90:10)")
cross_val_Training(l2, X_train_90, y_train_90, use_ADASYN=False)

# Linear OvR 80:20
print("\nModel L3 (Linear, OvR, Split 80:20):")
l3 = OneVsRestClassifier(svm_linear)
evaluation_Training(l3, X_train_80, y_train_80, "L3 (Linear,
OvR, 80:20)")
cross_val_Training(l3, X_train_80, y_train_80, use_ADASYN=False)

# Linear OvR 90:10
print("\nModel L4 (Linear, OvR, Split 90:10):")
l4 = OneVsRestClassifier(svm_linear)
evaluation_Training(l4, X_train_90, y_train_90, "L4 (Linear,
OvR, 90:10)")
cross_val_Training(l4, X_train_90, y_train_90, use_ADASYN=False)

# Model dengan ADASYN
```

```
svm_linear_ADASYN = SVC(**params_linear)

# Linear + ADASYN, OvO, 80:20
print("\nModel LA1 (Linear + ADASYN, OvO, Split 80:20):")
la1 = OneVsOneClassifier(svm_linear_ADASYN)
evaluation_Training(la1, X_train_ADASYN_80, y_train_ADASYN_80,
"LA1 (Linear + ADASYN, OvO, 80:20)")
cross_val_Training(la1, X_train_80, y_train_80, use_ADASYN=True)

# Linear + ADASYN, OvO, 90:10
print("\nModel LA2 (Linear + ADASYN, OvO, Split 90:10):")
la2 = OneVsOneClassifier(svm_linear_ADASYN)
evaluation_Training(la2, X_train_ADASYN_90, y_train_ADASYN_90,
"LA2 (Linear + ADASYN, OvO, 90:10)")
cross_val_Training(la2, X_train_90, y_train_90, use_ADASYN=True)

# Linear + ADASYN, OvR, 80:20
print("\nModel LA3 (Linear + ADASYN, OvR, Split 80:20):")
la3 = OneVsRestClassifier(svm_linear_ADASYN)
evaluation_Training(la3, X_train_ADASYN_80, y_train_ADASYN_80,
"LA3 (Linear + ADASYN, OvR, 80:20)")
cross_val_Training(la3, X_train_80, y_train_80, use_ADASYN=True)

# Linear + ADASYN, OvR, 90:10
print("\nModel LA4 (Linear + ADASYN, OvR, Split 90:10):")
la4 = OneVsRestClassifier(svm_linear_ADASYN)
evaluation_Training(la4, X_train_ADASYN_90, y_train_ADASYN_90,
"LA4 (Linear + ADASYN, OvR, 90:10)")
cross_val_Training(la4, X_train_90, y_train_90, use_ADASYN=True)
```

L15. *Source Code Training* SVM *Linear*

```
# Training SVM RBF
# Inisialisasi model SVM RBF
params_RBF = {
    'C' : 10,
    'gamma' : 'scale',
    'kernel' : 'RBF'
}
svm_RBF = SVC(**params_RBF)

# Model tanpa ADASYN
# RBF OvO 80:20
print("\nModel R1 (RBF, OvO, Split 80:20):")
r1 = OneVsOneClassifier(svm_RBF)
evaluation_Training(r1, X_train_80, y_train_80, "R1 (RBF, OvO,
80:20)")
cross_val_Training(r1, X_train_80, y_train_80, use_ADASYN=False)

# RBF OvO 90:10
print("\nModel R2 (RBF, OvO, Split 90:10):")
r2 = OneVsOneClassifier(svm_RBF)
```

```
evaluation_Training(r2, X_train_90, y_train_90, "R2 (RBF, OvO,
90:10)")
cross_val_Training(r2, X_train_90, y_train_90, use_ADASYN=False)

# RBF OvR 80:20
print("\nModel R3 (RBF, OvR, Split 80:20):")
r3 = OneVsRestClassifier(svm_RBF)
evaluation_Training(r3, X_train_80, y_train_80, "R3 (RBF, OvR,
80:20)")
cross_val_Training(r3, X_train_80, y_train_80, use_ADASYN=False)

# RBF OvR 90:10
print("\nModel R4 (RBF, OvR, Split 90:10):")
r4 = OneVsRestClassifier(svm_RBF)
evaluation_Training(r4, X_train_90, y_train_90, "R4 (RBF, OvR,
90:10)")
cross_val_Training(r4, X_train_90, y_train_90, use_ADASYN=False)

# Model dengan RBF + ADASYN
svm_RBF_ADASYN  = SVC(**params_RBF)

# RBF + ADASYN, OvO, 80:20
print("\nModel RA1 (RBF + ADASYN, OvO, Split 80:20):")
ra1 = OneVsOneClassifier(svm_RBF_ADASYN)
evaluation_Training(ra1, X_train_ADASYN_80, y_train_ADASYN_80,
"RA1 (RBF + ADASYN, OvO, 80:20)")
cross_val_Training(ra1, X_train_80, y_train_80, use_ADASYN=True)

# RBF + ADASYN, OvO, 90:10
print("\nModel RA2 (RBF + ADASYN, OvO, Split 90:10):")
ra2 = OneVsOneClassifier(svm_RBF_ADASYN)
evaluation_Training(ra2, X_train_ADASYN_90, y_train_ADASYN_90,
"RA2 (RBF + ADASYN, OvO, 90:10)")
cross_val_Training(ra2, X_train_90, y_train_90, use_ADASYN=True)

# RBF + ADASYN, OvR, 80:20
print("\nModel RA3 (RBF + ADASYN, OvR, Split 80:20):")
ra3 = OneVsRestClassifier(svm_RBF_ADASYN)
evaluation_Training(ra3, X_train_ADASYN_80, y_train_ADASYN_80,
"RA3 (RBF + ADASYN, OvR, 80:20)")
cross_val_Training(ra3, X_train_80, y_train_80, use_ADASYN=True)

# RBF + ADASYN, OvR, 90:10
print("\nModel RA4 (RBF + ADASYN, OvR, Split 90:10):")
ra4 = OneVsRestClassifier(svm_RBF_ADASYN)
evaluation_Training(ra4, X_train_ADASYN_90, y_train_ADASYN_90,
"RA4 (RBF + ADASYN, OvR, 90:10)")
cross_val_Training(ra4, X_train_90, y_train_90, use_ADASYN=True)
```

L16. *Source Code Training* SVM *RBF*

```
# Training SVM Polynomial
# Inisialisasi model SVM Polynomial
params_poly = {
    'C': 1,
    'coef0': 0.5,
    'degree': 3,
    'gamma': 1,
    'kernel': 'poly'
}
svm_poly = SVC(**params_poly)

# Model tanpa ADASYN
# Poly OvO 80:20
print("\nModel P1 (Polynomial, OvO, Split 80:20):")
p1 = OneVsOneClassifier(svm_poly)
evaluation_Training(p1, X_train_80, y_train_80, "P1 (Polynomial,
OvO, 80:20)")
cross_val_Training(p1, X_train_80, y_train_80, use_ADASYN=False)

# Poly OvO 90:10
print("\nModel P2 (Polynomial, OvO, Split 90:10):")
p2 = OneVsOneClassifier(svm_poly)
evaluation_Training(p2, X_train_90, y_train_90, "P2 (Polynomial,
OvO, 90:10)")
cross_val_Training(p2, X_train_90, y_train_90, use_ADASYN=False)

# Poly OvR 80:20
print("\nModel P3 (Polynomial, OvR, Split 80:20):")
p3 = OneVsRestClassifier(svm_poly)
evaluation_Training(p3, X_train_80, y_train_80, "P3 (Polynomial,
OvR, 80:20)")
cross_val_Training(p3, X_train_80, y_train_80, use_ADASYN=False)

# Poly OvR 90:10
print("\nModel P4 (Polynomial, OvR, Split 90:10):")
p4 = OneVsRestClassifier(svm_poly)
evaluation_Training(p4, X_train_90, y_train_90, "P4 (Polynomial,
OvR, 90:10)")
cross_val_Training(p4, X_train_90, y_train_90, use_ADASYN=False)


# Model dengan ADASYN
svm_poly_ADASYN = SVC(**params_poly)

# Poly + ADASYN, OvO, 80:20
print("\nModel PA1 (Polynomial + ADASYN, OvO, Split 80:20):")
pa1 = OneVsOneClassifier(svm_poly_ADASYN)
evaluation_Training(pa1, X_train_ADASYN_80, y_train_ADASYN_80,
"PA1 (Polynomial + ADASYN, OvO, 80:20)")
cross_val_Training(pa1, X_train_80, y_train_80, use_ADASYN=True)

# Poly + ADASYN, OvO, 90:10
print("\nModel PA2 (Polynomial + ADASYN, OvO, Split 90:10):")
pa2 = OneVsOneClassifier(svm_poly_ADASYN)
```

```
evaluation_Training(pa2, X_train_ADASYN_90, y_train_ADASYN_90,
"PA2 (Polynomial + ADASYN, OvO, 90:10)")
cross_val_Training(pa2, X_train_90, y_train_90, use_ADASYN=True)

# Poly + ADASYN, OvR, 80:20
print("\nModel PA3 (Polynomial + ADASYN, OvR, Split 80:20):")
pa3 = OneVsRestClassifier(svm_poly_ADASYN)
evaluation_Training(pa3, X_train_ADASYN_80, y_train_ADASYN_80,
"PA3 (Polynomial + ADASYN, OvR, 80:20)")
cross_val_Training(pa3, X_train_80, y_train_80, use_ADASYN=True)

# Poly + ADASYN, OvR, 90:10
print("\nModel PA4 (Polynomial + ADASYN, OvR, Split 90:10):")
pa4 = OneVsRestClassifier(svm_poly_ADASYN)
evaluation_Training(pa4, X_train_ADASYN_90, y_train_ADASYN_90,
"PA4 (Polynomial + ADASYN, OvR, 90:10)")
cross_val_Training(pa4, X_train_90, y_train_90, use_ADASYN=True)
```

L17. *Source Code Training* SVM *Polynomial*

```
# Training SVM Sigmoid
# Inisialisasi model SVM Sigmoid
params_sigmoid = {
    'C': 10,
    'coef0': 1,
    'gamma': 'scale',
    'kernel': 'sigmoid'
}
svm_sigmoid = SVC(**params_sigmoid)

# Model tanpa ADASYN
# Sigmoid OvO 80:20
print("\nModel S1 (Sigmoid, OvO, Split 80:20):")
s1 = OneVsOneClassifier(svm_sigmoid)
evaluation_Training(s1, X_train_80, y_train_80, "S1 (Sigmoid,
OvO, 80:20)")
cross_val_Training(s1, X_train_80, y_train_80, use_ADASYN=False)

# Sigmoid OvO 90:10
print("\nModel S2 (Sigmoid, OvO, Split 90:10):")
s2 = OneVsOneClassifier(svm_sigmoid)
evaluation_Training(s2, X_train_90, y_train_90, "S2 (Sigmoid,
OvO, 90:10)")
cross_val_Training(s2, X_train_90, y_train_90, use_ADASYN=False)

# Sigmoid OvR 80:20
print("\nModel S3 (Sigmoid, OvR, Split 80:20):")
s3 = OneVsRestClassifier(svm_sigmoid)
evaluation_Training(s3, X_train_80, y_train_80, "S3 (Sigmoid,
OvR, 80:20)")
cross_val_Training(s3, X_train_80, y_train_80, use_ADASYN=False)

# Sigmoid OvR 90:10
```

```
print("\nModel S4 (Sigmoid, OvR, Split 90:10):")
s4 = OneVsRestClassifier(svm_sigmoid)
evaluation_Training(s4, X_train_90, y_train_90, "S4 (Sigmoid,
OvR, 90:10)")
cross_val_Training(s4, X_train_90, y_train_90, use_ADASYN=False)


# Model dengan ADASYN
svm_sigmoid_ADASYN  = SVC(**params_sigmoid)

# Sigmoid + ADASYN, OvO, 80:20
print("\nModel SA1 (Sigmoid + ADASYN, OvO, Split 80:20):")
sa1 = OneVsOneClassifier(svm_sigmoid_ADASYN)
evaluation_Training(sa1, X_train_ADASYN_80, y_train_ADASYN_80,
"SA1 (Sigmoid + ADASYN, OvO, 80:20)")
cross_val_Training(sa1, X_train_80, y_train_80, use_ADASYN=True)

# Sigmoid + ADASYN, OvO, 90:10
print("\nModel SA2 (Sigmoid + ADASYN, OvO, Split 90:10):")
sa2 = OneVsOneClassifier(svm_sigmoid_ADASYN)
evaluation_Training(sa2, X_train_ADASYN_90, y_train_ADASYN_90,
"SA2 (Sigmoid + ADASYN, OvO, 90:10)")
cross_val_Training(sa2, X_train_90, y_train_90, use_ADASYN=True)

# Sigmoid + ADASYN, OvR, 80:20
print("\nModel SA3 (Sigmoid + ADASYN, OvR, Split 80:20):")
sa3 = OneVsRestClassifier(svm_sigmoid_ADASYN)
evaluation_Training(sa3, X_train_ADASYN_80, y_train_ADASYN_80,
"SA3 (Sigmoid + ADASYN, OvR, 80:20)")
cross_val_Training(sa3, X_train_80, y_train_80, use_ADASYN=True)

# Sigmoid + ADASYN, OvR, 90:10
print("\nModel SA4 (Sigmoid + ADASYN, OvR, Split 90:10):")
sa4 = OneVsRestClassifier(svm_sigmoid_ADASYN)
evaluation_Training(sa4, X_train_ADASYN_90, y_train_ADASYN_90,
"SA4 (Sigmoid + ADASYN, OvR, 90:10)")
cross_val_Training(sa4, X_train_90, y_train_90, use_ADASYN=True)
```

L18. *Source Code Training* SVM *Sigmoid*

```
# Testing SVM Linear
# Model tanpa ADASYN
# Linear OvO 80:20
print("\nModel L1 (Linear, OvO, Split 80:20):")
evaluation_Testing(l1, X_test_20, y_test_20, "L1 (Linear, OvO,
80:20)")

# Linear OvO 90:10
print("\nModel L2 (Linear, OvO, Split 90:10):")
evaluation_Testing(l2, X_test_10, y_test_10, "L2 (Linear, OvO,
90:10)")

# Linear OvR 80:20
```

```
print("\nModel L3 (Linear, OvR, Split 80:20):")
evaluation_Testing(l3, X_test_20, y_test_20, "L3 (Linear, OvR,
80:20)")

# Linear OvR 90:10
print("\nModel L4 (Linear, OvR, Split 90:10):")
evaluation_Testing(l4, X_test_10, y_test_10, "L4 (Linear, OvR,
90:10)")

# Model dengan ADASYN
# Linear + ADASYN, OvO, 80:20
print("\nModel LA1 (Linear + ADASYN, OvO, Split 80:20):")
evaluation_Testing(la1, X_test_20, y_test_20, "LA1 (Linear +
ADASYN, OvO, 80:20)")

# Linear + ADASYN, OvO, 90:10
print("\nModel LA2 (Linear + ADASYN, OvO, Split 90:10):")
evaluation_Testing(la2, X_test_10, y_test_10, "LA2 (Linear +
ADASYN, OvO, 90:10)")

# Linear + ADASYN, OvR, 80:20
print("\nModel LA3 (Linear + ADASYN, OvR, Split 80:20):")
evaluation_Testing(la3, X_test_20, y_test_20, "LA3 (Linear +
ADASYN, OvR, 80:20)")

# Linear + ADASYN, OvR, 90:10
print("\nModel LA4 (Linear + ADASYN, OvR, Split 90:10):")
evaluation_Testing(la4, X_test_10, y_test_10, "LA4 (Linear +
ADASYN, OvR, 90:10)")
```

L19. *Source Code Testing* SVM *Linear*

```
# Testing SVM RBF
# Model tanpa ADASYN
# RBF OvO 80:20
print("\nModel R1 (RBF, OvO, Split 80:20):")
evaluation_Testing(r1, X_test_20, y_test_20, "R1 (RBF, OvO,
80:20)")

# RBF OvO 90:10
print("\nModel R2 (RBF, OvO, Split 90:10):")
evaluation_Testing(r2, X_test_10, y_test_10, "R2 (RBF, OvO,
90:10)")

# RBF OvR 80:20
print("\nModel R3 (RBF, OvR, Split 80:20):")
evaluation_Testing(r3, X_test_20, y_test_20, "R3 (RBF, OvR,
80:20)")

# RBF OvR 90:10
print("\nModel R4 (RBF, OvR, Split 90:10):")
evaluation_Testing(r4, X_test_10, y_test_10, "R4 (RBF, OvR,
90:10)")
```

```
# Model dengan ADASYN
# RBF + ADASYN, OvO, 80:20
print("\nModel RA1 (RBF + ADASYN, OvO, Split 80:20):")
evaluation_Testing(ra1, X_test_20, y_test_20, "RA1 (RBF +
ADASYN, OvO, 80:20)")

# RBF + ADASYN, OvO, 90:10
print("\nModel RA2 (RBF + ADASYN, OvO, Split 90:10):")
evaluation_Testing(ra2, X_test_10, y_test_10, "RA2 (RBF +
ADASYN, OvO, 90:10)")

# RBF + ADASYN, OvR, 80:20
print("\nModel RA3 (RBF + ADASYN, OvR, Split 80:20):")
evaluation_Testing(ra3, X_test_20, y_test_20, "RA3 (RBF +
ADASYN, OvR, 80:20)")

# RBF + ADASYN, OvR, 90:10
print("\nModel RA4 (RBF + ADASYN, OvR, Split 90:10):")
evaluation_Testing(ra4, X_test_10, y_test_10, "RA4 (RBF +
ADASYN, OvR, 90:10)")
```

L20. *Source Code Testing* SVM *RBF*

```
# Testing SVM Polynomial
# Model tanpa ADASYN
# Poly OvO 80:20
print("\nModel P1 (Polynomial, OvO, Split 80:20):")
evaluation_Testing(p1, X_test_20, y_test_20, "P1 (Polynomial,
OvO, 80:20)")

# Poly OvO 90:10
print("\nModel P2 (Polynomial, OvO, Split 90:10):")
evaluation_Testing(p2, X_test_10, y_test_10, "P2 (Polynomial,
OvO, 90:10)")

# Poly OvR 80:20
print("\nModel P3 (Polynomial, OvR, Split 80:20):")
evaluation_Testing(p3, X_test_20, y_test_20, "P3 (Polynomial,
OvR, 80:20)")

# Poly OvR 90:10
print("\nModel P4 (Polynomial, OvR, Split 90:10):")
evaluation_Testing(p4, X_test_10, y_test_10, "P4 (Polynomial,
OvR, 90:10)")

# Model dengan ADASYN
# Poly + ADASYN, OvO, 80:20
print("\nModel PA1 (Polynomial + ADASYN, OvO, Split 80:20):")
evaluation_Testing(pa1, X_test_20, y_test_20, "PA1 (Polynomial +
ADASYN, OvO, 80:20)")

# Poly + ADASYN, OvO, 90:10
```

```
print("\nModel PA2 (Polynomial + ADASYN, OvO, Split 90:10):")
evaluation_Testing(pa2, X_test_10, y_test_10, "PA2 (Polynomial +
ADASYN, OvO, 90:10)")

# Poly + ADASYN, OvR, 80:20
print("\nModel PA3 (Polynomial + ADASYN, OvR, Split 80:20):")
evaluation_Testing(pa3, X_test_20, y_test_20, "PA3 (Polynomial +
ADASYN, OvR, 80:20)")

# Poly + ADASYN, OvR, 90:10
print("\nModel PA4 (Polynomial + ADASYN, OvR, Split 90:10):")
evaluation_Testing(pa4, X_test_10, y_test_10, "PA4 (Polynomial +
ADASYN, OvR, 90:10)")
```

L21. *Source Code Testing* SVM *Polynomial*

```
# Testing SVM Sigmoid
# Model tanpa ADASYN
# Sigmoid OvO 80:20
print("\nModel S1 (Sigmoid, OvO, Split 80:20):")
evaluation_Testing(s1, X_test_20, y_test_20, "S1 (Sigmoid, OvO,
80:20)")

# Sigmoid OvO 90:10
print("\nModel S2 (Sigmoid, OvO, Split 90:10):")
evaluation_Testing(s2, X_test_10, y_test_10, "S2 (Sigmoid, OvO,
90:10)")

# Sigmoid OvR 80:20
print("\nModel S3 (Sigmoid, OvR, Split 80:20):")
evaluation_Testing(s3, X_test_20, y_test_20, "S3 (Sigmoid, OvR,
80:20)")

# Sigmoid OvR 90:10
print("\nModel S4 (Sigmoid, OvR, Split 90:10):")
evaluation_Testing(s4, X_test_10, y_test_10, "S4 (Sigmoid, OvR,
90:10)")

# Model dengan ADASYN
# Sigmoid + ADASYN, OvO, 80:20
print("\nModel SA1 (Sigmoid + ADASYN, OvO, Split 80:20):")
evaluation_Testing(sa1, X_test_20, y_test_20, "SA1 (Sigmoid +
ADASYN, OvO, 80:20)")

# Sigmoid + ADASYN, OvO, 90:10
print("\nModel SA2 (Sigmoid + ADASYN, OvO, Split 90:10):")
evaluation_Testing(sa2, X_test_10, y_test_10, "SA2 (Sigmoid +
ADASYN, OvO, 90:10)")

# Sigmoid + ADASYN, OvR, 80:20
print("\nModel SA3 (Sigmoid + ADASYN, OvR, Split 80:20):")
evaluation_Testing(sa3, X_test_20, y_test_20, "SA3 (Sigmoid +
ADASYN, OvR, 80:20)")
```

```
# Sigmoid + ADASYN, OvR, 90:10
print("\nModel SA4 (Sigmoid + ADASYN, OvR, Split 90:10):")
evaluation_Testing(sa4, X_test_10, y_test_10, "SA4 (Sigmoid +
ADASYN, OvR, 90:10)")
```

L22. *Source Code Testing* SVM *Sigmoid*

RIWAYAT HIDUP

I Gede Krishna Adi atau yang lebih akrab dipanggil Krishna lahir di Denpasar pada tanggal 20 Juni 2002. Penulis lahir dari pasangan suami istri Bapak I Made Sadayatana dan Ibu Ni Nyoman Ayu Netrawati. Penulis merupakan pria berkebangsaan Indonesia dan beragama Hindu. Penulis saat ini beralamat di Jalan Gelogor Indah IA, Gang Damai, No. 9, Banjar Gelogor Carik, Desa Pemogan, Kecamatan Denpasar Selatan, Kota Denpasar, Provinsi Bali. Penulis menyelesaikan pendidikan dasar di SD 13 Pedungan dan lulus pada tahun 2014. Kemudian penulis melanjutkan pendidikan menengah pertama di SMP Ganesha Denpasar dan lulus pada tahun 2017. Selanjutnya penulis melanjutkan pendidikan menengah atas di SMA Negeri 5 Denpasar dan lulus pada tahun 2020. Setelah menempuh pendidikan formal selama 12 tahun, penulis melanjutkan pendidikan tinggi di Universitas Pendidikan Ganesha pada tahun 2020, dengan mengambil Program Studi S1 Ilmu Komputer, Jurusan Teknik Informatika. Pada akhir semester 10 tahun 2025 penulis telah menyelesaikan Tugas Akhir yang berjudul “Analisis Sentimen Pengguna Twitter Pada Pembukaan Kembali Tiktok Shop Di Indonesia: Implementasi SVM Dengan ADASYN”. Selanjutnya, sampai dengan penulisan skripsi ini, penulis masih terdaftar sebagai mahasiswa Program S1 Ilmu Komputer di Universitas Pendidikan Ganesha.