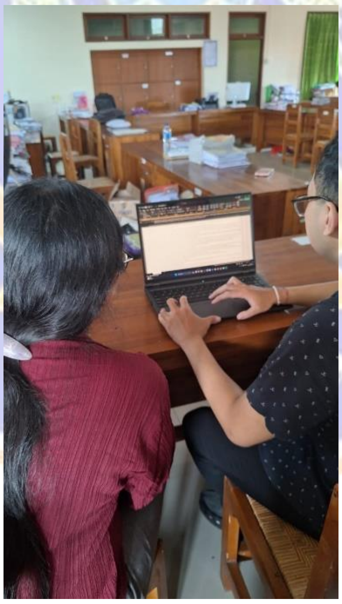
LAMPIRAN

Lampiran 1. Dokumentasi Kegiatan



Lampiran 2. Proses *Cleaning* dan *Case Folding* pada *Python*

```
def preprocess_text(kalimat):
    # Mengubah kalimat menjadi huruf kecil
    lower_case = kalimat.lower()

    # Menghapus angka dari kalimat
    hasil = re.sub(r"\d+", "", lower_case)
    # Menghapus tanda baca dari kalimat
    hasil = hasil.translate(str.maketrans("","",string.punctuation))

    # Menghapus spasi pada awal dan akhir kalimat
    hasil = hasil.strip()

    # Menghapus Username Twitter
    hasil = re.sub('@[^\s]+', ' ', hasil)

    # Menghapus https dan http
    hasil = re.sub(r"(?:\@|http?\://|https?\://|www)\S+", "", hasil)

    # Menghilangkan Tanda Baca
    hasil = hasil.translate(str.maketrans('', '', string.punctuation))

    # Mengganti karakter HTML dengan tanda petik
    hasil = re.sub('<.*?>', ' ', hasil)

    # Mempertimbangkan huruf dan angka
    hasil = re.sub('[^a-zA-Z0-9]', ' ', hasil)

    # Mengganti line baru dengan spasi
    hasil = re.sub("\n", " ", hasil)

    # Menghapus single char
    hasil = re.sub(r"\b[a-zA-Z]\b", " ", hasil)

    # Menghilangkan emoji
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emotikon wajah
                               u"\U0001F300-\U0001F5FF"  # simbol & benda
                               u"\U0001F680-\U0001F6FF"  # transportasi & simbol
peralatan
                               u"\U0001F1E0-\U0001F1FF"  # bendera negara
                               u"\U00002500-\U00002BEF"  # karakter CJK (Chinese,
Japanese, Korean)
                               u"\U00002702-\U000027B0"  # simbol & tanda
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               u"\U0001f926-\U0001f937"
                               u"\U00010000-\U0010ffff"
                               u"\u200d"
                               u"\u2640-\u2642"
                               u"\u2600-\u2B55"
                               u"\u23cf"
                               u"\u23e9"
                               u"\u231a"
                               u"\u3030"
                               u"\ufe0f"
                               "]+", flags=re.UNICODE)
    hasil = emoji_pattern.sub(r'', hasil)

    # Memisahkan dan menggabungkan kata
    hasil = ' '.join(hasil.split())

    return hasil

df['full_text'] = df['full_text'].astype(str)
df['text_clean'] = df['full_text'].apply(preprocess_text)
```

Lampiran 3. Proses Normalisasi Kata 'Alay' Menjadi Normal

```
kamus = pd.read_csv('kamusalay.csv', header=None)
kamus.columns = ['kata_asli', 'kata_normal']

def normalisasi_teks(teks, kamus):
    kata_asli = teks.split()
    kata_normal = []

    for kata in kata_asli:
        normal = kamus[kamus['kata_asli'] == kata]['kata_normal'].values
        if normal:
            kata_normal.append(normal[0])
        else:
            kata_normal.append(kata)

    return ' '.join(kata_normal)

df['normal'] = df['text_clean'].apply(normalisasi_teks, kamus=kamus)
```

Lampiran 4. Proses Tokenisasi Kalimat

```
import nltk
nltk.download('all')
from nltk.tokenize import word_tokenize

def tokenize_text(kalimat):
  tokens = nltk.tokenize.word_tokenize(kalimat)
  return tokens

df['token'] = df['normal'].apply(tokenize_text)
```

Lampiran 5. Proses Normalisasi Kata Tidak Baku Menjadi Normal

```
normalized_word = pd.read_csv("Normalisasi.csv", encoding='latin1')

normalized_word_dict={}
for index, row in normalized_word.iterrows():
    if row[0] not in normalized_word_dict:
        normalized_word_dict[row[0]] = row[1]

def normalized_term(document):
    return [normalized_word_dict[term] if term in normalized_word_dict else term
for term in document]

df['normal_1'] = df['token'].apply(normalized_term)
```

Lampiran 6. Proses *Stopword Removal*

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory import
StopWordRemoverFactory
factory = StopWordRemoverFactory()
stopwords = factory.get_stop_words()

def stopword_text(tokens):
    cleaned_tokens = []
    for token in tokens:
        if token not in stopwords:
            cleaned_tokens.append(token)
    return cleaned_tokens

df['stop'] = df['normal_1'].apply(stopword_text)
```

Lampiran 7. Proses *Stemming*

```
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
stem_factory = StemmerFactory()
stemmer = stem_factory.create_stemmer()

def stemming_text(tokens):
    hasil = [stemmer.stem(token) for token in tokens]
    return hasil

df['stemmed'] = df['stop'].apply(stemming_text)
df = df[df['stemmed'].map(lambda x: len(x) > 0)]
```

Lampiran 8. Proses Normalisasi Kata dengan Menyesuaikan KBBI

```
normalized_word = pd.read_csv("Normalisasi.csv", encoding='latin1')

normalized_word_dict={}
for index, row in normalized_word.iterrows():
    if row[0] not in normalized_word_dict:
        normalized_word_dict[row[0]] = row[1]

def normalized_term(document):
    return [normalized_word_dict[term] if term in normalized_word_dict else term
for term in document]

df['normal_1'] = df['token'].apply(normalized_term)
```

Lampiran 9. Proses *POS Tagging* Aspek Kinerja dengan CRF

```
def analyze_sentiment_based_on_aspects(text, ct):
    aspects = {
        "tidak beraspek": [],
        "penanganan kejahatan": ["jahat", "tangan", "penanganan", "kejahatan",
"tangkap", "serah", "periksa", "kasus", "tindak", "investigasi", "menangkap",
"kriminal", "pidana"],
        "kecepatan respon": ["cepat", "respon", "merespon", "langsung", "giat",
"gerak", "tuntas", "responsif", "sigap", "tanggap", "segera", "respons"],
        "interaksi terhadap masyarakat": ["interaksi", "ramah", "bantu",
"membantu", "masyarakat", "rakyat", "orang", "himbaunan", "selamat", "lapor",
"bimbing", "bimbingan", "komunikatif"],
    }

    text = text.lower()  # Konversi teks ke huruf kecil
    tokens = text.split()  # Tokenisasi teks

    # Melakukan POS tagging menggunakan CRF tagger
    tagged_text = ct.tag(tokens)
    aspect_sentiment = {aspect: 0 for aspect in aspects}

    for word, pos_tag in tagged_text:
        for aspect, keywords in aspects.items():
            if any(keyword in word for keyword in keywords):
                aspect_sentiment[aspect] += 1

    chosen_aspect = max(aspect_sentiment, key=aspect_sentiment.get)

    return chosen_aspect

# Load POS tagger model
ct = CRFTagger()
ct.set_model_file('all_indo_man_tag_corpus_model.crf.tagger')

# Menerapkan analisis sentimen berbasis aspek ke kolom 'teks' dalam DataFrame
data['sentiment_aspects'] = data['teks'].apply(lambda x:
analyze_sentiment_based_on_aspects(x, ct))

# Remove rows with no aspects
data = data[data['sentiment_aspects'].apply(lambda x: len(x) > 0)]

# Menghapus baris yang tidak memiliki aspek
data = data[data['sentiment_aspects'] != ""]

data.to_csv("data_aspect.csv", index=False)
```

Lampiran 10. Proses Pembagian *Dataset* Pemodelan Aspek Kinerja

```
# Membagi data menjadi data pelatihan dan data uji
X = data['teks']
y = data['sentiment_aspects']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Lampiran 11. Proses Pembagian *Dataset* Pemodelan Sentimen Berbasis Aspek

```
# Data dengan dua fitur: teks dan sentiment_aspects
X = data[['teks', 'sentiment_aspects']]
y = data['polarity']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Lampiran 12. Proses Ekstraksi Fitur pada Pemodelan Aspek Kinerja Menggunakan TF-IDF

```
# Inisialisasi TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
```

Lampiran 13. Proses Ekstraksi Fitur pada Pemodelan Sentimen Berbasis Aspek Menggunakan TF-IDF

```
# Inisialisasi TF-IDF Vectorizer untuk kolom 'teks'
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train['teks'])
X_test_tfidf = tfidf_vectorizer.transform(X_test['teks'])
```

Lampiran 14. Pemodelan Klasifikasi Aspek Kinerja

```
# Inisialisasi model SVM
svm_model = SVC(kernel='linear')

# Melakukan vektorisasi pada data teks pelatihan
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

# Melatih model SVM
svm_model.fit(X_train_tfidf, y_train)

# Melakukan vektorisasi pada data uji
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

Lampiran 15. Pemodelan Klasifikasi Sentimen Berbasis Aspek

```
# Inisialisasi model SVM
svm_model = SVC(kernel='linear')

# Melatih model SVM
svm_model.fit(X_train_tfidf, y_train)

from sklearn.model_selection import GridSearchCV

# Membuat parameter grid
param_grid = {
    'C': [0.1, 1, 10],  # Nilai C yang akan diuji
    'kernel': ['linear', 'rbf'],  # Jenis kernel yang akan diuji
}

# Inisialisasi model SVM
svm_model = SVC()

# Inisialisasi Grid Search dengan model dan parameter grid
grid_search = GridSearchCV(svm_model, param_grid, cv=5)

# Melakukan grid search
grid_search.fit(X_train_tfidf, y_train)

# Mendapatkan model terbaik
best_svm_sentiment = grid_search.best_estimator_

# Melakukan prediksi dengan model terbaik
y_pred_sentiment = best_svm_sentiment.predict(X_test_tfidf)
```

Lampiran 16. Proses Evaluasi Model Klasifikasi Aspek Kinerja

```
# Melakukan prediksi pada data uji
y_pred = svm_model.predict(X_test_tfidf)

# Menampilkan akurasi model
accuracy = accuracy_score(y_test, y_pred)
```

Lampiran 17. Pemodelan Evaluasi Model Klasifikasi Sentimen Berbasis Aspek

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

y_pred = y_pred_sentiment
model = best_svm_sentiment

# Evaluasi model sentimen terbaik
accuracy_sentiment = accuracy_score(y_test, y_pred_sentiment)

report_sentiment = classification_report(y_test, y_pred_sentiment,
zero_division=0)

# Menghitung confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=model.classes_)

# Menampilkan confusion matrix dengan angka
classes = model.classes_
plt.figure(figsize=(8, 6))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

# Menambahkan angka di setiap sel
for i in range(len(classes)):
    for j in range(len(classes)):
        plt.text(j, i, str(cm[i, j]), ha='center', va='center', color='black',
fontsize=12)

plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

print(f'Akurasi Model Sentimen (setelah tuning): {accuracy_sentiment:.2f}')
print(report_sentiment)
plt.show()
```