

## BAB 4

### HASIL DAN PEMBAHASAN

#### 4.1. *Environment Testing*

Pengujian aplikasi dilakukan menggunakan perangkat lunak *Visual Studio Code* yang dimiliki oleh *Microsoft*. Perangkat lunak ini mendukung pengembangan program menggunakan bahasa pemrograman *python* versi 3.1.1.3 melalui kumpulan *extension* yang tersedia pada perangkat lunak. Kumpulan *extension* yang digunakan antara lain *Python*, *Jupyter*, *Jupyter Slideshow*, *Jupyter Notebook*, *Jupyter Keymap*, *Jupyter Cell Tags*, *Jupyter*. Perangkat lunak ini dijalankan pada komputer yang memiliki spesifikasi sebagai berikut:

- A. Ryzen 5600G 6-Core 3.9 GHz
- B. 16 GB RAM
- C. 480 GB SSD

#### 4.2. Hasil Penelitian

Pada penelitian ini, dilakukan beberapa tahapan untuk mendapatkan hasil dari penelitian. Proses ini dimulai dari *Data Preparation* yang terdiri dari pengumpulan data, pelabelan data, pra pemrosesan teks hingga transformasi teks. Kemudian, dilanjutkan dengan tahap *data mining* yaitu pembangunan model yang dilanjutkan dengan tahap *pattern/knowledge evaluation* dengan evaluasi dari pembangunan model. Terakhir tahap *knowledge presentation* dengan menunjukkan visualisasi hasil evaluasi pembangunan model.

##### 4.2.1. *Data Preparation*

###### 4.2.1.1. Penarikan dan Pengumpulan data

Dalam tahap pengumpulan data, dilakukan *data scraping* pada *website* toko aplikasi Google Play Store. Pengambilan data dilakukan menggunakan *library google\_play\_scraper* yang dapat dipasang untuk kemudian digunakan pada bahasa pemrograman *python*. Data yang diambil merupakan data dari pengguna aplikasi Twitter yang menceritakan pengalaman mereka dalam menggunakan Twitter pada kolom ulasan Google

Play Store. Data yang dikumpulkan merupakan ulasan sebanyak 136354 *record* yang ditulis sejak tahun 2020 hingga tahun 2023.

*Scraping* dilakukan menggunakan function `reviews_all` pada library `google_play_scraper`. *Function* ini dapat melakukan pengambilan data ulasan aplikasi Twitter menggunakan beberapa parameter untuk membatasi data yang akan diambil, Parameter-parameter tersebut diantaranya *id*, *lang*, *country*, dan *sort*.

#### 4.2.1.1.1. *Id*

*Id* sebagai parameter merujuk kepada *id* dari aplikasi yang akan dicari data ulasannya, dalam penelitian ini, digunakan *id* dari aplikasi Twitter yakni `com.twitter.android`.

#### 4.2.1.1.2. *Lang*

*Lang* sebagai parameter merujuk kepada bahasa dari ulasan yang ditulis. Dalam penelitian ini dibatasi untuk hanya mengambil ulasan yang ditulis dalam bahasa Inggris.

#### 4.2.1.1.3. *Country*

*Country* sebagai parameter merujuk kepada negara dimana ulasan ditulis. Dalam penelitian ini, dibatasi untuk hanya mengambil ulasan yang ditulis di negara Indonesia.

#### 4.2.1.1.4. *Sort*

*Sort* sebagai parameter merujuk kepada penyortiran dalam pengambilan ulasan. Dalam penelitian ini, disortir ulasan yang diambil berdasarkan tanggal, yakni tanggal terbaru. Berikut Gambar 4.1 yang merupakan kode dalam melakukan *scraping data*.

```
dataset = reviews_all(  
    'com.twitter.android'  
    ,sleep_milliseconds=0  
    ,lang = 'en'  
    ,country = 'id'  
    ,sort = Sort.NEWEST  
)
```

Gambar 4.1 Kode *scraping* data

Selanjutnya dilakukan pengubahan data ulasan yang berbentuk JSON menjadi *dataframe* pandas. Setelah data ulasan berbentuk *dataframe* pandas, ditambahkan kolom 'year' untuk mengelompokkan ulasan berdasarkan tahun penulisan. Ulasan yang dibatasi adalah ulasan yang tidak lebih tua dari tahun 2020 hingga bulan Juni 2023. Untuk membatasinya, digunakan *function* query yang dapat menjalankan perintah query pada *dataframe* pandas. Berikut Gambar 4.2 sebagai contoh kode dalam membatasi tahun dan Gambar 4.3 sebagai contoh data yang dikumpulkan

```
#transform reviews into pandas dataframe
dataset = pd.DataFrame(np.array(dataset),columns=['review'])
dataset = dataset.join(pd.DataFrame(dataset.pop('review').tolist()))

#mendapatkan tahun dari dataset
dataset['year'] = pd.DatetimeIndex(dataset['at']).year

#mengambil hanya tahun terbaru hingga 2023
dataset = dataset.query("year >= 2020")
```

Gambar 4.2 Kode pembatasan tahun yang diambil

	reviewId	content	score	year
0	b0bd418f-d86e-4474-8ef8-c706ece6f82c	When i scroll the timeline, it suddenly refres...	2	2023
2	a5ac2c4b-4dae-47eb-bed5-38e1104ed5f7	Navigating the apps is very confusing with the...	1	2023
3	e9380068-9b4d-4fdf-a9b6-3b06a80839a2	Unreliable. Too often it takes forever for a T...	2	2023
4	459e6307-2afc-4e4a-b67e-12fa21208352	I can't get on the app. Haven't used it for a ...	1	2023
5	f0941d1c-dca0-426e-8b80-0bcf7e67e377	Great socialisation poor function. I've used t...	2	2023

Gambar 4.3 Contoh data yang dikumpulkan

Setelah pembatasan tahun, kemudian dilakukan pembatasan kolom data apa saja yang akan disimpan untuk digunakan pada tahap-tahap selanjutnya. Kolom-kolom yang digunakan untuk disimpan yaitu kolom 'reviewId', 'content', 'score', 'year'. Kolom 'reviewId' yang berisikan id unik setiap ulasan dan kolom 'year' yang berisikan tahun dimana ulasan itu ditulis hanya akan digunakan sebagai identitas data, sedangkan kolom 'content' yang berisikan ulasan akan menjadi kolom yang digunakan untuk *text mining* dengan kolom 'score' yang digunakan sebagai pelabelan data.

Terakhir, data yang sudah dikumpulkan, disimpan kedalam bentuk *file comma-separated values* (CSV). File ini menyimpan data tabel menjadi barisan biasa yang dipisahkan dengan simbol koma. Untuk melakukan penyimpanan ini, digunakan *function* `to_csv` yang mengubah data yang semula berbentuk *dataframe* menjadi CSV. Tujuan penyimpanan ini untuk mencadangkan data supaya data tidak hilang ketika komputer mengalami kendala selama penelitian.

#### 4.2.1.2. Pelabelan data

Selanjutnya, dilakukan pelabelan data sesuai dengan ketentuan apakah ulasan tersebut berisikan sentimen positif atau tidak. Bagi ulasan yang memiliki sentimen positif akan diberi label 1, sedangkan bagi ulasan yang memiliki sentimen negatif akan diberi label 0. Sentimen positif dari ulasan dilihat dari penilaian bintang ulasan. Ulasan yang memberi penilaian bintang 4 dan 5 pada aplikasi dianggap sebagai ulasan yang memiliki sentimen positif, sedangkan ulasan yang memberi penilaian bintang 1 dan bintang 2 pada aplikasi dianggap sebagai ulasan yang memiliki sentimen negatif. Ulasan yang memberikan bintang 3 dianggap sebagai penilaian netral. Pada bintang penelitian ini ulasan bintang 3 dianggap tidak informatif (Fransiska, et al., 2020). Pada Tabel 4.1 berikut merupakan contoh ulasan yang memiliki sentimen positif dan negatif secara berurut.

Tabel 4.1 Sentimen positif dan negatif

Ulasan dengan sentimen positif	Ulasan dengan sentimen negatif
<i>It's nice app ☐ I am very happy ☐</i>	<i>The new update is TERRIBLE!</i>
<i>The app is really good. Works great all of the time.</i>	<i>Very bad app.I CAN'T LOG IN OR CREATE MY ACCOUNT</i>
<i>I really enjoy using the app.</i>	<i>The new video player is utterly bad.</i>

Berikut Gambar 4.4 yang merupakan kode dalam melakukan pelabelan data.



```

Pelabelan Data

df = df[df['score'] != 3]
df['positive_sentiment'] = np.where(df['score'] > 3, 1, 0)
✓ 0.0s
+ Code + Markdown

```

Gambar 4.4 Kode untuk melakukan pelabelan data

Dalam pelabelan ini, dibuat satu kolom bernama ‘*positive\_sentiment*’ yang memiliki fungsi untuk menampung nilai biner apakah ulasan pada suatu row memiliki sentimen positif atau negatif. Hasil dari pelabelan ini dapat dilihat pada Gambar 4.5 berikut.

	reviewId	content	score	year	positive_sentiment
0	b0bd418f-d86e-4474-8ef8-c706ece6f82c	When i scroll the timeline, it suddenly refres...	2	2023	0
2	a5ac2c4b-4dae-47eb-bed5-38e1104ed5f7	Navigating the apps is very confusing with the...	1	2023	0
3	e9380068-9b4d-4fdf-a9b6-3b06a80839a2	Unreliable. Too often it takes forever for a T...	2	2023	0
4	459e6307-2afc-4e4a-b67e-12fa21208352	I can't get on the app. Haven't used it for a ...	1	2023	0
5	f0941d1c-dca0-426e-8b80-0bcf7e67e377	Great socialisation poor function. I've used t...	2	2023	0

Gambar 4.5 Hasil pelabelan data

#### 4.2.1.3. Pra pemrosesan teks

Selanjutnya, data masuk ke tahap pra pemrosesan, dimana akan dilakukan beberapa tahapan untuk membuat data siap untuk di transformasi. Data yang akan digunakan untuk di proses yaitu data ulasan dari kolom ‘*content*’. Tahapan-tahapan ini yaitu pembersihan data, *case folding*, *Tokenization*, *Stemming*, *Lemmatization*, dan *Stopword Removal*. Dalam mengerjakan tahap ini, dimanfaatkan *library* NLTK (*Natural Language Toolkit*) untuk melakukan pra pemrosesan teks, seperti pembersihan data, *stemming*, *lemmatization*, hingga penghapusan *stopwords*.

#### 4.2.1.3.1. Pembersihan data

Dalam melakukan pembersihan data, diperiksa apakah terdapat data yang bernilai kosong atau tidak konsisten. Hasil pemeriksaan menunjukkan bahwa tidak ditemukan data yang bernilai kosong. Berikut Gambar 4.6 menunjukkan pemeriksaan data bernilai kosong.

```
df.isna().sum()

reviewId      0
content       0
score         0
year          0
positive_sentiment  0
dtype: int64
```

Gambar 4.6. Pemeriksaan data bernilai kosong

Selain itu, juga dilakukan pembersihan dengan menghapus karakter-karakter pada teks yang bukan merupakan huruf seperti tanda seru, tanda tanya, tanda pagar, *tag* HTML, serta *link* URL. Tujuan dari tahap ini adalah untuk memastikan bahwa selama proses *text mining* dapat berjalan dengan menganalisa karakter berupa huruf-huruf. Berikut Gambar 4.7 menunjukkan proses pembersihan data dan Gambar 4.8 menunjukkan hasil pembersihan data.

```
def pembersihan(text):

    text = re.sub(r'^a-zA-Z\s', '', text)      #pen
    text = re.sub(r'http\S+', ' ', text)      #
    text = ''.join([each for each in text if each])
    return text

✓ 0.0s

df['content'] = df['content'].apply(pembersihan)
df['content'].head()

✓ 0.9s
```

Gambar 4.7 Proses pembersihan data

```
0   When i scroll the timeline it suddenly refresh...
2   Navigating the apps is very confusing with the...
3   Unreliable Too often it takes forever for a Tw...
4   I cant get on the app Havent used it for a whi...
5   Great socialisation poor function Ive used thi...
Name: content, dtype: object
```

Gambar 4.8 Hasil pembersihan data

#### 4.2.1.3.2. *Case folding*

Selanjutnya, setelah data dibersihkan dilakukan *case folding* untuk mengubah huruf kapital menjadi huruf kecil. Untuk mengkonversi menjadi huruf kecil, dilakukan *cast* untuk mengubah tipe data dari kolom 'content' menjadi *string* sehingga dapat diaplikasikan *function lower()*. Tahap ini dilakukan untuk membuat data menjadi lebih konsisten. Berikut Gambar 4.9 menunjukkan proses *case folding* beserta Gambar 4.10 menunjukkan hasil dari proses *case folding*.

2. Case Folding

```
df['content'] = df['content'].str.lower()
df['content'].head()
```

✓ 0.0s

Gambar 4.9 Proses *case folding*

```
0   when i scroll the timeline it suddenly refresh...
2   navigating the apps is very confusing with the...
3   unreliable too often it takes forever for a tw...
4   i cant get on the app havent used it for a whi...
5   great socialisation poor function ive used thi...
Name: content, dtype: object
```

Gambar 4.10 Hasil dari proses *case folding*

#### 4.2.1.3.3. *Tokenization*

Setelah data menjadi huruf kecil semua, dilakukan proses *tokenization*. Proses ini membagi satu kalimat menjadi beberapa kata menggunakan *function .strip()* yang kemudian dikumpulkan menjadi sebuah *list* yang dibatasi dengan spasi menggunakan *function .split()*. *List* yang

berisikan kata-kata ini disebut sebagai token. Berikut Gambar 4.11 menunjukkan proses *tokenization* beserta Gambar 4.12 menunjukkan hasil dari proses *tokenization*.

```
def tokenization(text):
    text = text.strip()
    text = text.split(' ')
    return text

✓ 0.0s

df['content'] = df['content'].apply(tokenization)
df['content'].head()

✓ 0.4s
```

Gambar 4.11 Proses *tokenization*

```
0    [when, i, scroll, the, timeline, it, suddenly,...
2    [navigating, the, apps, is, very, confusing, w...
3    [unreliable, too, often, it, takes, forever, f...
4    [i, cant, get, on, the, app, havent, used, it,...
5    [great, socialisation, poor, function, ive, us...
Name: content, dtype: object
```

Gambar 4.12 Hasil dari proses *tokenization*

#### 4.2.1.3.4. *Stemming*

Setelah dipisah-pisah menjadi kumpulan kata, pada proses ini kata-kata yang memiliki imbuhan dapat dipotong menjadi kata dasar. Pada tahap ini, diimplementasikan *function PorterStemmer* dari *library NLTK*. *Function* ini melakukan pemotongan token-token yang memiliki kata imbuhan. Berikut Gambar 4.13 menunjukkan proses *stemming* beserta Gambar 4.14 menunjukkan hasil dari proses *stemming*.



```
# stemming
porter_stemmer = nltk.PorterStemmer()

def stemming(text):
    text = [porter_stemmer.stem(each) for each in text]
    return text

✓ 0.0s

df['content'] = df['content'].apply(stemming)
df['content'].head()

✓ 21.9s
```

Gambar 4.13 Proses stemming.

```
0    [when, i, scroll, the, timelin, it, suddenli, ...
2    [navig, the, app, is, veri, confus, with, the,...
3    [unreli, too, often, it, take, forev, for, a, ...
4    [i, cant, get, on, the, app, havent, use, it, ...
5    [great, socialis, poor, function, ive, use, th...
Name: content, dtype: object
```

Gambar 4.14 Hasil dari proses *stemming*.

#### 4.2.1.3.5. *Lemmatization*

Pemotongan kata yang berlebihan pada tahap *stemming* menimbulkan masalah pada kata-kata yang berubah bentuk ketika diberi imbuhan. Untuk penyelesaian masalah tersebut, kata-kata yang telah melewati proses *stemming* harus dilakukan *lemmatization*. Pada penelitian ini, digunakan *function* WordNetLemmatizer dari *library* NLTK yang dapat mengembalikan Berikut Gambar 4.15 yang menunjukkan proses *lemmatization* beserta Gambar 4.16 yang menunjukkan hasil dari proses *lemmatization*.

```

lemma = nltk.WordNetLemmatizer()
def lemmatization(text):
    text = [lemma.lemmatize(each) for each in text]
    return text
✓ 0.0s

df['content'] = df['content'].apply(lemmatization)
df['content'].head()
✓ 6.5s

```

Gambar 4.15 Proses *lemmatization*.

```

0    [when, i, scroll, the, timelin, it, suddenli, ...
2    [navig, the, app, is, veri, confu, with, the, ...
3    [unr, too, often, it, take, forev, for, a, twe...
4    [i, cant, get, on, the, app, havent, use, it, ...
5    [great, sociali, poor, function, ive, use, thi...
Name: content, dtype: object

```

Gambar 4.16 Hasil dari *lemmatization*.

#### 4.2.1.3.6. *Stopword Removal*

Setelah data dari satu kalimat dipecah menjadi beberapa kata melalui tokenisasi, yang kemudian dikembalikan menjadi bentuk asal dari kata tersebut. Dilakukan deteksi dan penghapusan *stopword*. *Stopword* adalah kata-kata yang tidak memiliki makna yang dapat mengganggu pembuatan model. Untuk melakukan penghapusan *stopword*, digunakan *library stopwords* dari NLTK yang berisikan kata-kata *stopword* dalam bahasa Inggris. Berikut Gambar 4.17 menunjukkan proses penghapusan *stopword* beserta Gambar 4.18 menunjukkan hasil dari proses penghapusan *stopword*.

```
def remove_stopwords(text):
    words = []
    for word in text:
        if word not in stopwords.words('english'):
            words.append(word)
    return words

✓ 0.0s

df['content'] = df['content'].apply(remove_stopwords)
df['content'].head()

✓ 9m 55.7s
```

Gambar 4.17 Proses penghapusan *stopwords*

```
0    [scroll, timelin, suddenli, refresh, realli, a...
2    [navig, app, veri, confu, constantli, chang, u...
3    [unr, often, take, forev, tweet, send, crash, ...
4    [cant, get, app, havent, use, ive, chang, pass...
5    [great, sociali, poor, function, ive, use, thi...
Name: content, dtype: object
```

Gambar 4.18 Hasil dari proses penghapusan *stopwords*.

#### 4.2.1.4. Transformasi teks

Setelah data melalui tahap pra pemrosesan teks, data selanjutnya ditransformasi menjadi data numerik. Namun, sebelumnya dibagi terlebih dahulu menjadi data latih dan data uji yang akan ditransformasi. Untuk pembagian data latih dan uji, digunakan *library model selection* dari *sklearn* yang didalamnya terdapat *function train\_test\_split*. *Function* ini berguna untuk membagi data yang akan dijadikan data latih dan data uji secara acak berdasarkan jumlah data. Pembagian data ini menghasilkan 102265 data yang akan dilatih dan 34089 data yang akan menjadi data uji.

Data latih dan data uji dari kolom '*content*' ditransformasikan dengan menggunakan penghitungan bobot menggunakan metode TF-IDF. Tahap ini menghitung frekuensi kemunculan suatu kata sekaligus *inverse document* dari

setiap kata pada dataset (Darren, et al., 2022). Berikut Gambar 4.19 menunjukkan proses pembagian data dan transformasi teks.

```
X_train, X_test, Y_train, Y_test = train_test_split(df['content'],
                                                    df['positive_sentiment'],
                                                    random_state=1)

vect = TfidfVectorizer().fit(X_train)

X_train_vectorised = vect.transform(X_train)
X_test_vectorised = vect.transform(X_test)
```

Gambar 4.19 Pembagian data dan transformasi teks.

#### 4.2.2. Data Mining

##### 4.2.2.1. Pembangunan model

Data yang sudah berbentuk angka siap untuk dipasangkan dengan algoritma *naïve bayes* dan *decision tree* untuk dibangun model prediksi dari data tersebut.

##### 4.2.2.2. Algoritma Naïve Bayes

Setelah ditentukan data latih dan data uji, diaplikasikan algoritma *Naïve Bayes* pada data latih untuk pengembangan model. Pada penelitian ini, digunakan MultinomialNB yang merupakan algoritma yang didasari oleh penghitungan *Naïve Bayes* yang lazim digunakan untuk melakukan analisis sentimen. Berikut Gambar 4.20 menunjukkan proses pembangunan model menggunakan algoritma *naïve bayes*.

```
#Multinomial NB
from sklearn.naive_bayes import MultinomialNB
MNB = MultinomialNB()
MNB = MNB.fit(X_train_vectorised, Y_train)

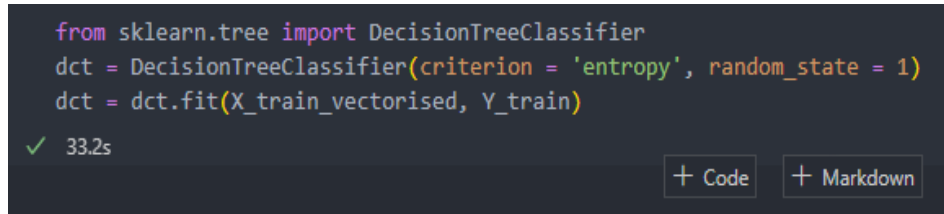
✓ 0.0s
```

Gambar 4.20 Proses pembangunan model menggunakan algoritma *Naïve Bayes*

##### 4.2.2.3. Algoritma Decision Tree

Selanjutnya, menggunakan data latih dan data uji yang sama, diaplikasikan algoritma *decision tree* untuk mengembangkan model. Hal ini

ditujukan untuk melihat dan membandingkan performa antara algoritma *Naïve Bayes* dan *Decision Tree* dalam membangun model klasifikasi analisis sentimen. Algoritma yang digunakan untuk membuat model klasifikasi yaitu *DecisionTreeClassifier*. Berikut Gambar 4.13 menunjukkan proses pembangunan model algoritma *decision tree*.



```
from sklearn.tree import DecisionTreeClassifier
dct = DecisionTreeClassifier(criterion = 'entropy', random_state = 1)
dct = dct.fit(X_train_vectorised, Y_train)
```

✓ 33.2s

+ Code + Markdown

Gambar 4.21 Proses pembangunan model menggunakan algoritma *Decision Tree*

### 4.2.3. Knowledge Evaluation

#### 4.2.3.1. Evaluasi

Setelah model dibuat, dilakukan evaluasi untuk mengetahui dan membandingkan performa dari algoritma yang di aplikasikan dalam pembuatan model klasifikasi. Pada tahap ini, dibuat *confusion matrix*, untuk mencari nilai akurasi, presisi, *recall*, dan *F1-score* yang berguna untuk mencari *True Positive Rate* dan *False Positive Rate* dalam menghitung luas area dibawah kurva ROC (*Receiving Operating Characteristic*) atau dikenal juga sebagai AUC-ROC. Dalam melakukan evaluasi, digunakan *library sklearn.metrics* untuk menghitung nilai-nilai yang dijadikan bahan evaluasi dari pembangunan model klasifikasi.

#### 4.2.3.2. Confusion Matrix

Evaluasi menggunakan *confusion matrix* dapat menampilkan hasil prediksi dari algoritma. Untuk membentuk *confusion matrix*, digunakan *function confusion\_matrix* dari *library sklearn.metrics* yaitu *confusion\_matrix*. Hasil yang didapatkan yaitu nilai *True Positive*, *False Positive*, *True Negative* dan *False Negative* dari prediksi model yang telah dibuat terhadap data uji.

#### 4.2.3.2.1. Algoritma Naïve Bayes

Berikut Gambar 4.22 yang menunjukkan proses pembuatan *confusion matrix* dari hasil pembangunan model menggunakan algoritma *Naïve Bayes*.

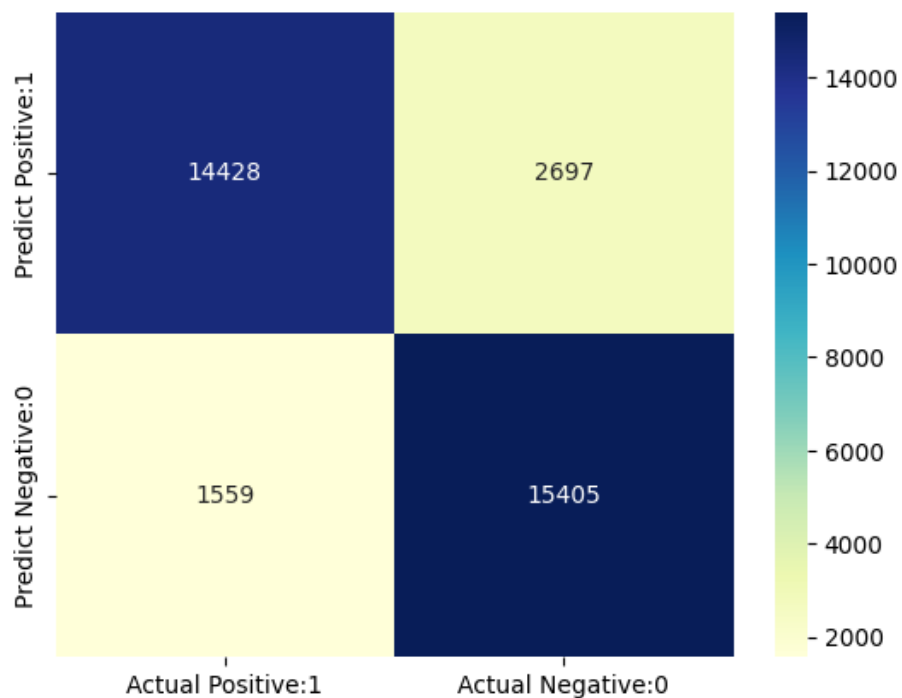
```
conf_matrix_NB = confusion_matrix(predicted_NB, Y_test)

cm_matrix = pd.DataFrame(
    data=conf_matrix_NB,
    columns=['Actual Positive:1', 'Actual Negative:0'],
    index=['Predict Positive:1', 'Predict Negative:0']
)

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Gambar 4.22 Penghitungan nilai *confusion matrix* pada algoritma *Naïve Bayes*

Selanjutnya, digunakan *library seaborn* untuk memvisualisasikan *confusion matrix* dari Berikut Gambar 4.23 yang menunjukkan visualisasi hasil *confusion matrix* dari algoritma *Naïve Bayes* dalam pembangunan model klasifikasi.



Gambar 4.23 *Confusion matrix* algoritma *Naïve Bayes*

Pada Gambar 4.23 dijelaskan bahwa algoritma *Naïve Bayes* menghasilkan nilai *True Positive* (TP) sebanyak 14428 data, nilai *False Positive* (FP) sebanyak 2697 data, nilai *False Negative* sebanyak 1559 data, dan nilai *True Negative* sebanyak 15405 data.

#### 4.2.3.2.2. Algoritma Decision Tree

Pada algoritma *Decision Tree*, juga dicari nilai *confusion matrix*, hal ini sesuai dengan tujuan penulisan yaitu untuk melakukan perbandingan antara kedua algoritma. Berikut Gambar 4.24 yang menunjukkan proses penghitungan *confusion matrix* dari hasil pembangunan model menggunakan algoritma *Decision Tree Classifier*.

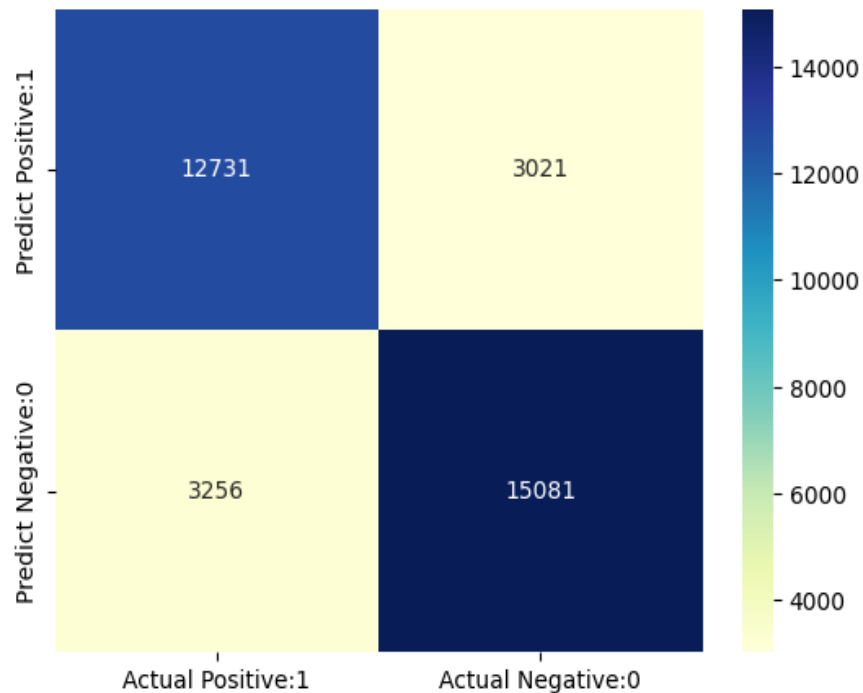
```
conf_matrix_DT = confusion_matrix(predicted_DCT, Y_test)

cm_matrix = pd.DataFrame(
    data=conf_matrix_DT,
    columns=['Actual Positive:1', 'Actual Negative:0'],
    index=['Predict Positive:1', 'Predict Negative:0']
)

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Gambar 4.24 Penghituna nilai *confusion matrix* pada algoritma  
*Decision Tree*

Selanjutnya dilakukan visualisasi *confusion matrix* menggunakan *library* seaborn. Berikut Gambar 4.25 yang menunjukkan hasil *confusion matrix* dari algoritma *Decision Tree* dalam pembangunan model klasifikasi.

Gambar 4.25 *Confusion matrix* algoritma *Decision Tree*

#### 4.2.3.3. Nilai akurasi

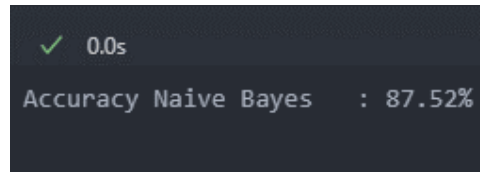
##### 4.2.3.3.1. Algoritma *Naïve Bayes*

Dalam menghitung nilai akurasi pada algoritma *Naïve Bayes*, digunakan *function accuracy\_score* dari *library sklearn.metrics*. Dalam menghitung nilai akurasi dipasangkan model klasifikasi yang sudah dibangun dengan label yang menjadi data uji. Berikut Gambar 4.26 menunjukkan penghitungan nilai akurasi beserta Gambar 4.27 berupa nilainya.

```
predicted_NB = MNB.predict(X_test_vectorised)
accuracy_NB = metrics.accuracy_score(predicted_NB, Y_test)
print("Accuracy Naïve Bayes : ",
      str(
        '{:04.2f}'.format(accuracy_NB*100)
      ) + '%')
)
```

Gambar 4.26 Proses penghitungan nilai akurasi algoritma *Naïve Bayes*



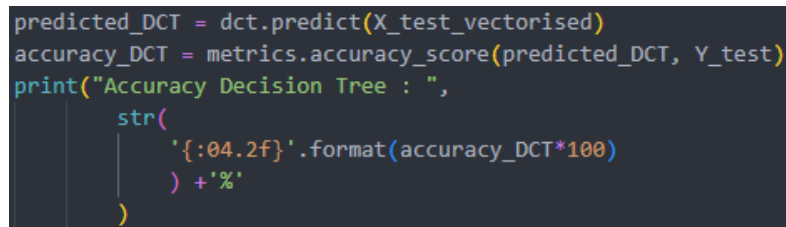


```
✓ 0.0s
Accuracy Naive Bayes : 87.52%
```

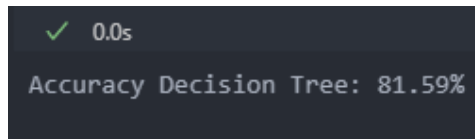
Gambar 4.27 Nilai akurasi algoritma *Naïve Bayes*

#### 4.2.3.3.2. Algoritma *Decision Tree*

Pada algoritma *Decision Tree*, dihitung nilai akurasi menggunakan *function accuracy\_score* dari *library sklearn.metrics*. Dalam menghitung nilai akurasi dipasangkan model klasifikasi yang sudah dibangun dengan label yang menjadi data uji. Berikut Gambar 4.28 menunjukkan penghitungan nilai akurasi beserta Gambar 4.29 berupa nilainya.



```
predicted_DCT = dct.predict(X_test_vectorised)
accuracy_DCT = metrics.accuracy_score(predicted_DCT, Y_test)
print("Accuracy Decision Tree : ",
      str(
        '{:04.2f}'.format(accuracy_DCT*100)
      ) + '%')
```

Gambar 4.28 Proses penghitungan nilai akurasi algoritma *Decision Tree*


```
✓ 0.0s
Accuracy Decision Tree: 81.59%
```

Gambar 4.29 Nilai akurasi algoritma *Decision Tree*

#### 4.2.3.4. Nilai presisi

##### 4.2.3.4.1. Algoritma *Naïve Bayes*

Dalam menghitung nilai presisi pada algoritma *Naïve Bayes*, digunakan *function precision\_score* dari *library sklearn.metrics*. Dalam menghitung nilai presisi dipasangkan model klasifikasi yang sudah dibangun dengan label yang menjadi data uji. Berikut Gambar 4.30 menunjukkan penghitungan nilai presisi beserta Gambar 4.31 berupa nilainya.

```

predicted_NB = MNB.predict(X_test_vectorised)
precision_NB = metrics.precision_score(predicted_NB, Y_test)
print("Precision Naive Bayes : ",
      str(
        '{:04.2f}'.format(precision_NB*100)
      ) + '%')

```

Gambar 4.30 Proses penghitungan nilai presisi algoritma *Naive Bayes*

```

✓ 0.0s
Precision Naive Bayes : 85.10%

```

Gambar 4.31 Nilai presisi algoritma *Naive Bayes*

#### 4.2.3.4.2. Algoritma *Decision Tree*

Dalam menghitung nilai presisi pada algoritma *Decision Tree*, digunakan *function precision\_score* dari *library sklearn.metrics*. Dalam menghitung nilai presisi dipasangkan model klasifikasi yang sudah dibangun dengan label yang menjadi data uji. Berikut Gambar 4.32 menunjukkan penghitungan nilai presisi beserta Gambar 4.33 berupa nilainya.

```

predicted_DCT = dct.predict(X_test_vectorised)
precision_DCT = metrics.precision_score(predicted_DCT, Y_test)
print("Precision Decision Tree : ",
      str(
        '{:04.2f}'.format(precision_DCT*100)
      ) + '%')

```

Gambar 4.32 Proses penghitungan nilai presisi algoritma *Decision Tree*

```

✓ 0.0s
Precision Decision Tree : 83.31%

```

Gambar 4.33 Nilai presisi algoritma *Decision Tree*

#### 4.2.3.5. Nilai *recall*

##### 4.2.3.5.1. Algoritma *Naïve Bayes*

Dalam menghitung nilai *recall* pada algoritma *Naïve Bayes*, digunakan *function recall\_score* dari *library sklearn.metrics*. Dalam menghitung nilai *recall* dipasangkan model klasifikasi yang sudah dibangun dengan label yang menjadi data uji. Berikut Gambar 4.34 menunjukkan penghitungan nilai presisi beserta Gambar 4.35 berupa nilainya.

```
predicted_NB = MNB.predict(X_test_vectorised)
recall_NB = metrics.recall_score(predicted_NB, Y_test)
print("Recall Naive Bayes : ",
      str(
        '{:04.2f}'.format(recall_NB*100)
      ) + '%')
```

Gambar 4.34 Proses penghitungan nilai *recall* algoritma *Naïve Bayes*

```
✓ 0.0s
Recall Naive Bayes: 90.81%
```

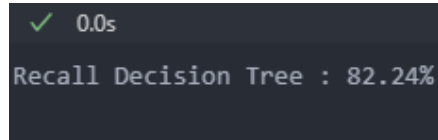
Gambar 4.35 Nilai *recall* algoritma *Naïve Bayes*

##### 4.2.3.5.2. Algoritma *Decision Tree*

Dalam menghitung nilai *recall* pada algoritma *Decision Tree*, digunakan *function recall\_score* dari *library sklearn.metrics*. Dalam menghitung nilai *recall* dipasangkan model klasifikasi yang sudah dibangun dengan label yang menjadi data uji. Berikut Gambar 4.36 menunjukkan penghitungan nilai presisi beserta Gambar 4.37 berupa nilainya.

```
predicted_DCT = dct.predict(X_test_vectorised)
recall_DCT = metrics.recall_score(predicted_DCT, Y_test)
print("Recall Decision Tree : ",
      str(
        '{:04.2f}'.format(recall_DCT*100)
      )
    )
```

Gambar 4.36 Proses penghitungan nilai *recall* algoritma *Decision Tree*



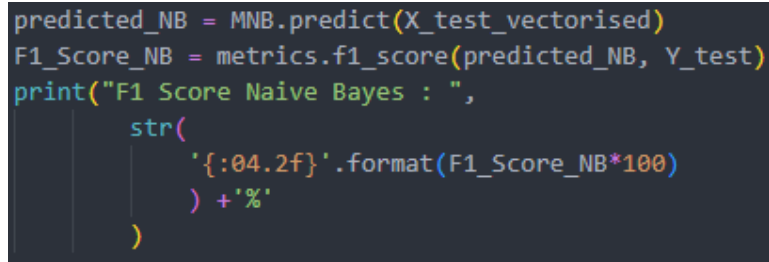
```
✓ 0.0s
Recall Decision Tree : 82.24%
```

Gambar 4.37 Nilai *recall* algoritma *Naïve Bayes*

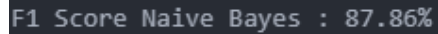
#### 4.2.3.6. Nilai *F1-Score*

##### 4.2.3.6.1. Algoritma *Naïve Bayes*

Penghitungan nilai *F1-Score* pada algoritma *Naïve Bayes*, digunakan *function f1\_score* dari *library sklearn.metrics*. Dalam menghitung nilai *f1-score* dipasangkan model klasifikasi yang sudah dibangun dengan label yang menjadi data uji. Berikut Gambar 4.38 menunjukkan penghitungan nilai presisi beserta Gambar 4.39 berupa nilainya.



```
predicted_NB = MNB.predict(X_test_vectorised)
F1_Score_NB = metrics.f1_score(predicted_NB, Y_test)
print("F1 Score Naïve Bayes : ",
      str(
          '{:04.2f}'.format(F1_Score_NB*100)
          ) + '%')
```

Gambar 4.38 Proses penghitungan nilai *f1-score* algoritma *Naïve Bayes*


```
F1 Score Naïve Bayes : 87.86%
```

Gambar 4.39 Nilai *f1-score* algoritma *Naïve Bayes*

##### 4.2.3.6.2. Algoritma *Decision Tree*

Penghitungan nilai *F1-Score* pada algoritma *Decision Tree*, digunakan *function f1\_score* dari *library sklearn.metrics*. Dalam menghitung nilai *f1-score* dipasangkan model klasifikasi yang sudah dibangun dengan label yang menjadi data uji. Berikut Gambar 4.40 menunjukkan penghitungan nilai presisi beserta Gambar 4.41 berupa nilainya.

```

predicted_DCT = dct.predict(X_test_vectorised)
F1_Score_DCT = metrics.f1_score(predicted_DCT, Y_test)
print("F1 Score Decision Tree : ",
      str(
        '{:04.2f}'.format(F1_Score_DCT*100)
      ) + '%')

```

Gambar 4.40 Proses penghitungan nilai *f1-score* algoritma *Decision Tree*

```

✓ 0.0s
F1 Score Decision Tree: 82.77%

```

Gambar 4.41 Nilai *f1-score* algoritma *Decision Tree*

#### 4.2.3.7. Luas area dibawah kurva ROC (*Receiver Operating Characteristic*)

##### 4.2.3.7.1. Algoritma *Naïve Bayes*

Penghitungan luas AUC-ROC pada algoritma *Naïve Bayes*, digunakan *function roc\_auc\_score* dari *library sklearn.metrics*. Dalam menghitung nilai *AUC-ROC* dipasangkan model klasifikasi yang sudah dibangun dengan label yang menjadi data uji. Berikut Gambar 4.42 menunjukkan penghitungan nilai presisi beserta Gambar 4.43 berupa nilainya.

```

predicted_NB = MNB.predict(X_test_vectorised)
ROC_AUC_NB = metrics.roc_auc_score(predicted_NB, Y_test)
print("ROC_AUC Naive Bayes : ",
      str(
        '{:04.2f}'.format(ROC_AUC_NB*100)
      )
)

```

Gambar 4.42 Proses penghitungan nilai AUC-ROC algoritma *Naïve Bayes*

```

✓ 0.0s
ROC_AUC Naive Bayes : 87.53

```

Gambar 4.43 Luas AUC-ROC algoritma *Naïve Bayes*

#### 4.2.3.7.2. Algoritma *Decision Tree*

Penghitungan luas AUC-ROC pada algoritma *Decision Tree*, digunakan *function roc\_auc\_score* dari *library sklearn.metrics*. Dalam menghitung nilai AUC-ROC dipasangkan model klasifikasi yang sudah dibangun dengan label yang menjadi data uji. Berikut Gambar 4.44 menunjukkan penghitungan nilai presisi beserta Gambar 4.45 berupa nilainya.

```
predicted_DCT = dct.predict(X_test_vectorised)
ROC_AUC_DCT = metrics.roc_auc_score(predicted_DCT, Y_test)
print("ROC_AUC Decision Tree : ",
      str(
        '{:04.2f}'.format(ROC_AUC_DCT*100)
      )
    )
```

Gambar 4.44 Proses penghitungan nilai AUC-ROC algoritma *Decision Tree*

```
✓ 0.0s
ROC_AUC Decision Tree: 81.53
```

Gambar 4.45 Nilai AUC-ROC algoritma *Decision Tree*

#### 4.2.4. Knowledge Presentation

##### 4.2.4.1. Visualisasi

Terakhir, dilakukan penggambaran visualisasi menggunakan *library matplotlib* untuk menjelaskan analisis akurasi presisi, *recall*, *f1-score*, serta nilai AUC-ROC dari algoritma *Naïve Bayes* dan *Decision Tree*. Untuk melakukan visualisasi pertama-tama, dilakukan impor *library matplotlib* untuk mendapatkan *library matplotlib* pada lingkungan penelitian. Gambar 4.46 menunjukkan impor *library*.

```
#impor library visualisasi
import matplotlib.pyplot as plt
```

Gambar 4.46 Proses impor *library*

Lalu, untuk menjelaskan analisis akurasi, presisi, *recall*, *f1-score* serta nilai AUC-ROC dari masing-masing algoritma, digunakan *function bar* dengan parameter X dan Y dimana X berisikan nama algoritma yang diuji, dan Y berisikan nilai evaluasi yang diuji. *Function bar* ini menghasilkan diagram batang yang mampu menjelaskan analisis akurasi, presisi, *recall*, *f1-score*, dan nilai AUC-ROC. Kemudian digunakan *function xlabel* untuk memberi label pada sumbu X dan *ylabel* untuk memberi label pada sumbu y. Ditampilkan nilai dari masing-masing hasil perhitungan pada diagram batang.

#### 4.2.4.2. Akurasi

Berikut Gambar 4.47 menjelaskan pembuatan diagram batang yang menggambarkan nilai akurasi antara algoritma *Naïve Bayes* dan algoritma *Decision Tree*, beserta Gambar 4.48 yang merupakan hasil.

```
X=["Naïve Bayes","Decision Tree"]
Y=[round(accuracy_NB, 2), round(accuracy_DCT, 2)]
blue = '#1f77b4'
red = '#bf3264'

plt.rcParams['figure.figsize'] = (4,5)

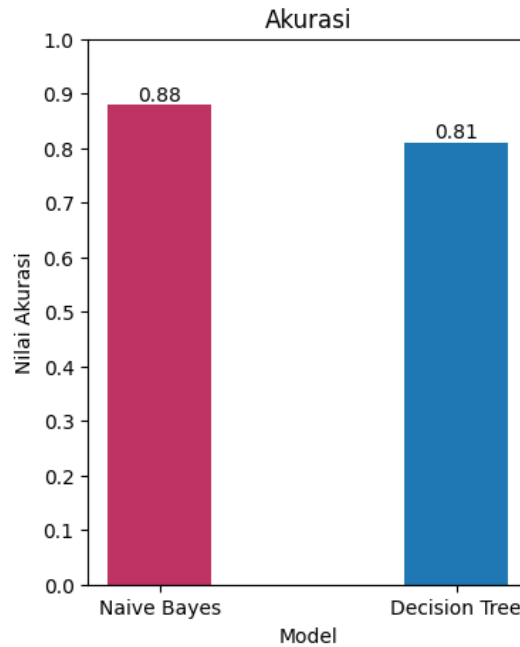
plt.bar(X,Y,width=0.35, color=(red, blue))
plt.yticks(np.arange(0, 1.1, 0.1))

plt.xlabel("Model")
plt.ylabel("Nilai Akurasi")
plt.title("Akurasi")

for i in range (len(X)):
    plt.text(i, Y[i], Y[i], ha='center', va='bottom')

plt.show()
```

Gambar 4.47 proses pembuatan visualisasi perbandingan nilai akurasi



Gambar 4.48 Hasil visualisasi perbandingan nilai akurasi

#### 4.2.4.3. Presisi

Berikut Gambar 4.49 menjelaskan pembuatan diagram batang yang menggambarkan nilai presisi antara algoritma *Naïve Bayes* dan algoritma *Decision Tree*, beserta Gambar 4.50 yang merupakan hasil.

```
X=["Naïve Bayes","Decision Tree"]
Y=[round(precision_NB, 2), round(precision_DCT, 2)]
blue = '#1f77b4'
red = '#bf3264'

plt.rcParams['figure.figsize'] = (4,5)

plt.bar(X,Y,width=0.35, color=(red, blue))
plt.yticks(np.arange(0, 1.1, 0.1))

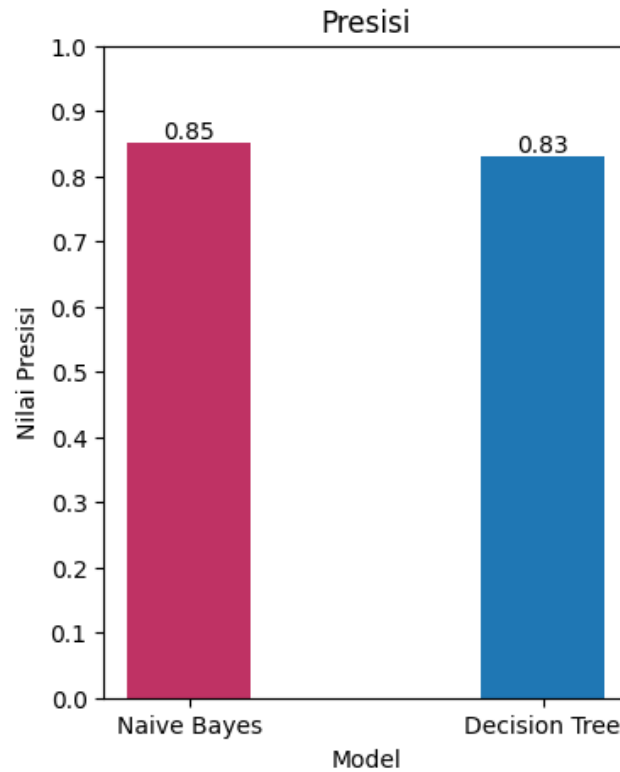
plt.xlabel("Model")
plt.ylabel("Nilai Presisi")
plt.title("Presisi")

for i in range (len(X)):
    plt.text(i, Y[i], Y[i], ha='center', va='bottom')

plt.show()
```

Gambar 4.49 Proses pembuatan visualisasi perbandingan nilai presisi





Gambar 4.50 Hasil visualisasi perbandingan nilai presisi

#### 4.2.4.4. *Recall*

Berikut Gambar 4.51 menjelaskan pembuatan diagram batang yang menggambarkan nilai *recall* antara algoritma *Naïve Bayes* dan algoritma *Decision Tree*, beserta Gambar 4.52 yang merupakan hasil.

```
X=["Naive Bayes","Decision Tree"]
Y=[round(recall_NB, 2), round(recall_DCT, 2)]
blue = '#1f77b4'
red = '#bf3264'

plt.rcParams['figure.figsize'] = (4,5)

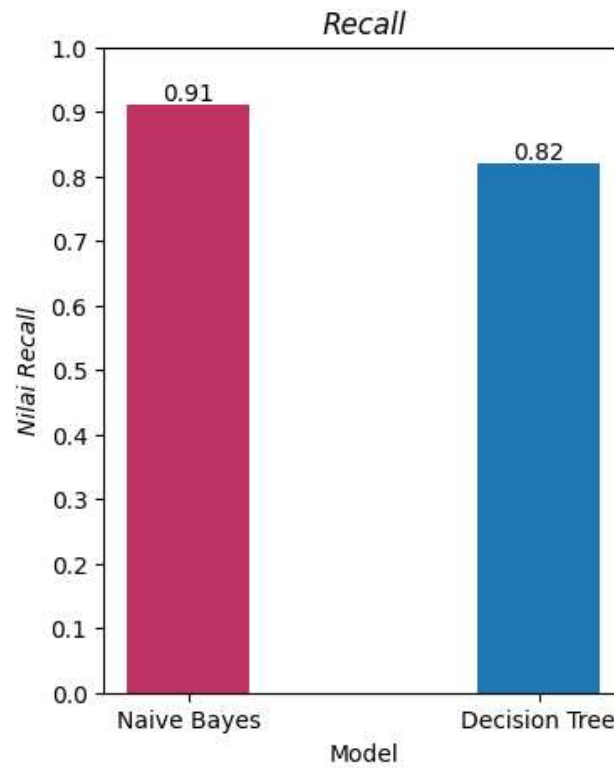
plt.bar(X,Y,width=0.35, color=(red, blue))
plt.yticks(np.arange(0, 1.1, 0.1))

plt.xlabel("Model")
plt.ylabel("Nilai Recall", style='italic')
plt.title("Recall", style='italic')

for i in range (len(X)):
    plt.text(i, Y[i], Y[i], ha='center', va='bottom')

plt.show()
```

Gambar 4.51 Proses pembuatan visualisasi perbandingan nilai *recall*



Gambar 4.52 Hasil visualisasi perbandingan nilai *recall*

#### 4.2.4.5. *F1-Score*

Berikut Gambar 4.53 menjelaskan pembuatan diagram batang yang menggambarkan nilai *f1-score* antara algoritma *Naïve Bayes* dan algoritma *Decision Tree*, beserta Gambar 4.54 yang merupakan hasil.

```

X=["Naive Bayes","Decision Tree"]
Y=[round(F1_Score_NB, 2), round(F1_Score_DCT, 2)]
blue = '#1f77b4'
red = '#bf3264'

plt.rcParams['figure.figsize'] = (4,5)

plt.bar(X,Y,width=0.35, color=(red, blue))
plt.yticks(np.arange(0, 1.1, 0.1))

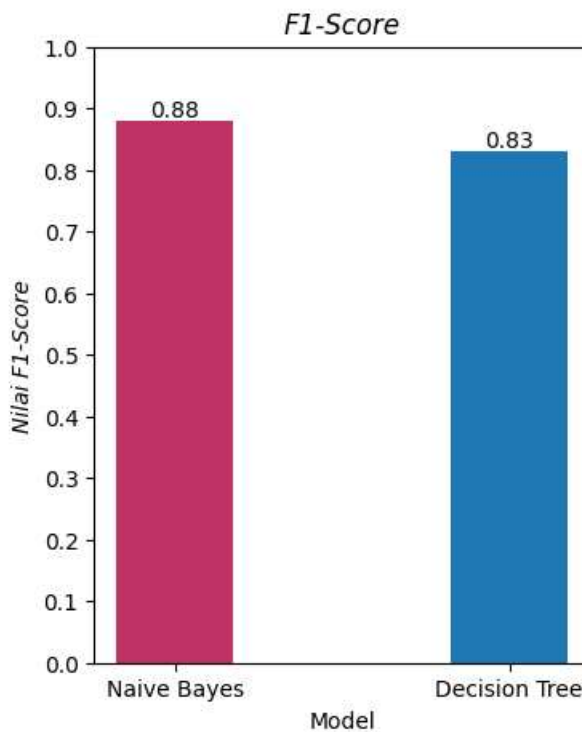
plt.xlabel("Model")
plt.ylabel("Nilai F1-Score", style='italic')
plt.title("F1-Score", style='italic')

for i in range (len(X)):
    plt.text(i, Y[i], Y[i], ha='center', va='bottom')

plt.show()

```

Gambar 4.53 Proses pembuatan visualisasi perbandingan nilai *f1-score*



Gambar 4.54 Hasil visualisasi perbandingan nilai *f1-score*

#### 4.2.4.6. Nilai AUC-ROC

Berikut Gambar 4.55 menjelaskan pembuatan diagram batang yang menggambarkan nilai AUC-ROC antara algoritma *Naïve Bayes* dan algoritma *Decision Tree*, beserta Gambar 4.56 yang merupakan hasil.

```

X=["Naive Bayes","Decision Tree"]
Y=[round(ROC_AUC_NB, 2), round(ROC_AUC_DCT, 2)]

blue = '#1f77b4'
red = '#bf3264'

plt.rcParams['figure.figsize'] = (4,5)

plt.bar(X,Y,width=0.35, color=(red, blue))
plt.yticks(np.arange(0, 1.1, 0.1))

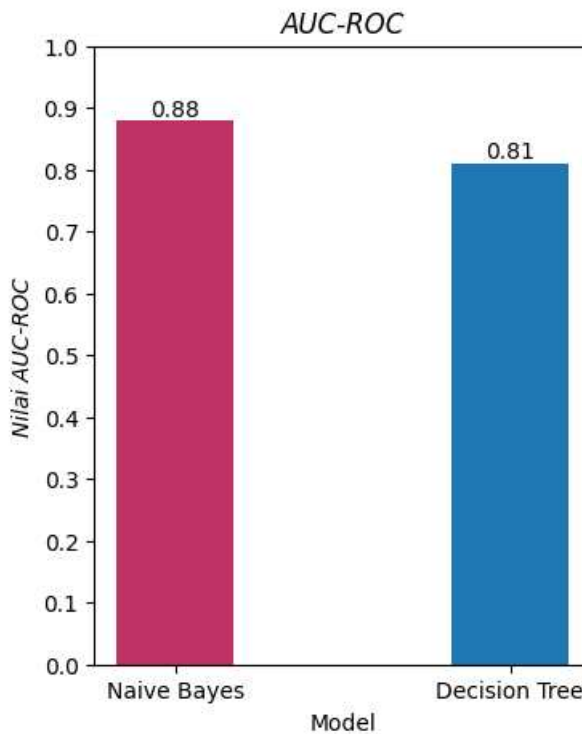
plt.xlabel("Model")
plt.ylabel("Nilai AUC-ROC", style='italic')
plt.title("AUC-ROC", style='italic')

for i in range (len(X)):
    plt.text(i, Y[i], Y[i], ha='center', va='bottom')

plt.show()

```

Gambar 4.55 Proses pembuatan visualisasi perbandingan nilai AUC-ROC



Gambar 4.56 Hasil visualisasi perbandingan nilai AUC-ROC

#### 4.2.4.7. Perbandingan Keseluruhan Penilaian

Berikut Gambar 4.57 menjelaskan pembuatan diagram batang yang menggambarkan perbandingan keseluruhan nilai antara algoritma *Naïve Bayes* dan algoritma *Decision Tree*, beserta Gambar 4.58 yang merupakan hasil.

```
X1=["Akurasi NB", "Presisi NB",
    "Recall NB", "F1-Score NB", "AUC-ROC NB"
]

X2=["Akurasi DT", "Presisi DT",
    "Recall DT", "F1-Score DT", "AUC-ROC DT"
]

Y1=[round(accuracy_NB, 2), round(precision_NB, 2),
    round(recall_NB, 2), round(F1_Score_NB, 2),
    round(ROC_AUC_NB, 2)
]
Y2=[round(accuracy_DCT, 2), round(precision_DCT, 2),
    round(recall_DCT, 2), round(F1_Score_DCT, 2),
    round(ROC_AUC_DCT, 2)
]

blue = '#1f77b4'
red = '#bf3264'

plt.rcParams['figure.figsize'] = (13,6)
plt.bar(X1, Y1,width=0.5 , color=(red))
plt.bar(X2, Y2,width=0.5 , color=(blue))
plt.yticks(np.arange(0, 1.1, 0.1))

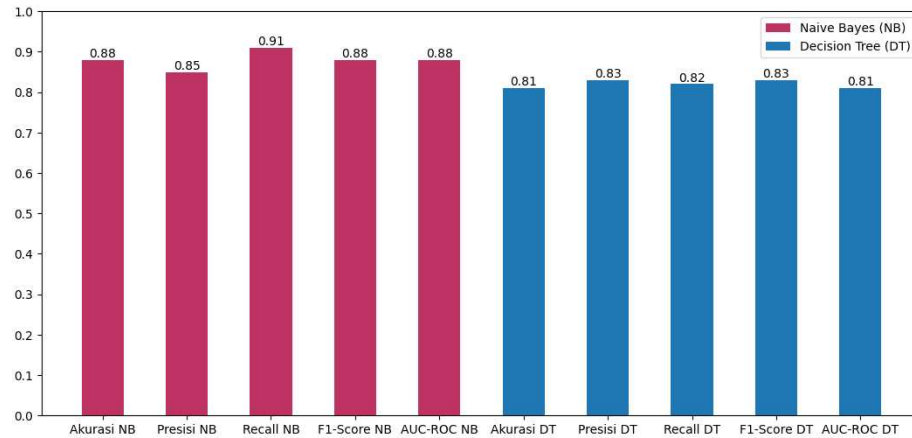
# value tiap bar naive bayes
for i in range (len(X1)):
    plt.text(i, Y1[i], Y1[i], ha='center', va='bottom')

# value tiap bar Decision Tree
for u in range (len(X2)):
    plt.text(u+5, Y2[u], Y2[u], ha='center', va='bottom')

plt.legend(["Naive Bayes (NB)", "Decision Tree (DT)"])

plt.show()
```

Gambar 4.57 Proses pembuatan visualisasi perbandingan seluruh nilai



Gambar 4.58 Hasil perbandingan keseluruhan nilai

#### 4.3. Evaluasi Sistem

Hasil penelitian ini menunjukkan perbandingan nilai evaluasi performa dari algoritma *Naïve Bayes* dan *Decision Tree*. Nilai evaluasi yang dibandingkan yakni nilai akurasi, presisi, *recall*, *f1-score*, dan luas area dibawah kurva ROC (Receiver Operating Characteristics) atau dikenal juga sebagai AUC-ROC. Perbandingan nilai-nilai evaluasi dari masing-masing algoritma dapat dilihat pada Tabel 4.2.

Tabel 4.2 Perbandingan nilai evaluasi *Naïve Bayes* dan *Decision Tree*

Penilaian	Naïve Bayes	Decision Tree
<b>Akurasi</b>	87.52%	81.59%
<b>Presisi</b>	90.81%	83.31%
<b>Recall</b>	85.10%	82.24%
<b>F1-Score</b>	87.86%	82.77%
<b>AUC-ROC</b>	0.88	0.81

Berdasarkan Tabel 4.2, algoritma *Naïve Bayes* memiliki tingkat akurasi tertinggi dalam melakukan analisis sentimen dibandingkan dengan algoritma *Decision Tree*. Algoritma *Naïve Bayes* mampu mencapai tingkat akurasi 87.52% dibandingkan dengan algoritma *Decision Tree* yang hanya mampu mencapai tingkat akurasi 81.59%. Pada nilai presisi, algoritma *Naïve Bayes* memiliki tingkat presisi 90.81%. Hal ini sangat jauh jika dibandingkan dengan algoritma *Decision Tree* yang hanya memiliki tingkat presisi di angka 83.31%. Dari segi

*recall*, algoritma *Naïve Bayes* juga unggul yang mencapai angka 85.10% dibandingkan dengan algoritma *Decision Tree* yang hanya mencapai angka 82.24%. Nilai *f1-score* algoritma *Naïve Bayes* mencapai angka 87.86% dengan nilai AUC-ROC (*area under curve-Receiver Operating Characteristic*) yang mencapai angka 0.8753. Berbeda dengan algoritma *Decision Tree* yang hanya memiliki nilai *f1-score* sebesar 82.77% dengan nilai AUC-ROC sebesar 0.8153.

Secara keseluruhan, penelitian ini menunjukkan bahwa algoritma *Naïve Bayes* lebih unggul dibandingkan algoritma *Decision Tree* dalam melakukan klasifikasi untuk analisis sentimen terhadap ulasan aplikasi media sosial di Google Play Store.

#### 4.3.1. Evaluasi dengan penghitungan manual

Penghitungan manual dari algoritma *Naïve Bayes* dan *Decision Tree* dilakukan pada bab 3. Penghitungan ini menghasilkan nilai akurasi, presisi, *recall*, *f1-score*, serta AUC-ROC yang jika dibandingkan dengan hasil evaluasi dari penghitungan program seperti Tabel 4.3 berikut.

Tabel 4.3 Hasil evaluasi penghitungan manual

Jenis Penilaian	Hasil Penghitungan Program		Hasil Penghitungan Manual	
	<i>Naïve Bayes</i>	<i>Decision Tree</i>	<i>Naïve Bayes</i>	<i>Decision Tree</i>
Akurasi	87.52%	81.59%	80%	70%
Presisi	90.81%	83.31%	83.4%	71.4%
Recall	85.10%	82.24%	83.4%	83.4%
F1-Score	87.86%	82.77%	83.4%	76.9%
AUC-ROC	0.8753	0.8153	0.625	0.525

Berdasarkan Tabel 4.3, dapat diketahui bahwa terdapat perbedaan nilai akurasi algoritma *Naïve Bayes* yaitu pada penghitungan manual berada di angka 80%, sedangkan penghitungan menggunakan program berada di angka 87.52%. Selain itu perbedaan angka juga dapat dilihat pada nilai akurasi algoritma *Decision Tree* yang pada penghitungan manual memiliki angka 70%, sedangkan pada penghitungan program memiliki angka 81.59%. Adanya perbedaan ini, disebabkan oleh jumlah data uji yang berbeda. Pada penghitungan program, digunakan sejumlah 34089, sedangkan pada penghitungan manual digunakan 10 sampel dari masing masing algoritma. Oleh karena itu, nilai akurasi, baik menggunakan algoritma *Naïve Bayes* maupun algoritma *Decision Tree* keduanya menghasilkan tingkat akurasi di

angka bulat. Dalam penghitungan manual, algoritma *Naïve Bayes* hanya mampu tepat memprediksi 8 data dari 10 data yang diuji, sedangkan algoritma *Decision Tree* hanya mampu tepat memprediksi 7 data dari 10 data yang diuji.

Perbedaan pada angka presisi juga terjadi antara penghitungan manual dengan penghitungan program. Pada penghitungan manual, didapatkan nilai presisi pada algoritma *Naïve Bayes* sebesar 83.4%, dan nilai presisi pada algoritma *Decision Tree* sebesar 71.4%. Nilai-nilai ini, berbeda dengan nilai yang didapatkan dari penghitungan program, dimana algoritma *Naïve Bayes* memiliki nilai presisi sebesar 90.81% dan algoritma *Decision Tree* memiliki nilai presisi sebesar 83.31%. Perbedaan ini terjadi dikarenakan perbedaan perbandingan nilai *True Positive* (TP) dan *False Positive* (FP) pada penghitungan program dan penghitungan manual. Pada penghitungan program, perbandingan nilai TP dan FP algoritma *Naïve Bayes* adalah (14425:2697) dan perbandingan nilai TP dan FP algoritma *Decision Tree* adalah (12677:3031). Sedangkan pada penghitungan manual perbandingan nilai TP dan FP algoritma *Naïve Bayes* adalah (5:1), dan perbandingan nilai TP dan FP algoritma *Decision Tree* adalah (5:2).

Pada angka *recall* ditemukan perbedaan yang tidak terlalu jauh antara penghitungan program dengan penghitungan manual. Nilai *recall* pada penghitungan manual dari algoritma *Naïve Bayes*, yaitu 83.4% dan algoritma *Decision Tree* berada pada angka yang sama yaitu 83.4%. Pada penghitungan program, algoritma *Naïve Bayes* memiliki nilai *recall* sebesar 85.1% dan algoritma *Decision Tree* memiliki nilai *recall* sebesar 82.24%. Perbedaan yang tidak terlalu jauh, didukung oleh perbandingan nilai *True Positive* (TP) dengan *False Negative* (FN) pada penghitungan manual, baik algoritma *Naïve Bayes* maupun *Decision Tree*, yakni (5:1). Pada penghitungan program algoritma *Naïve Bayes* memiliki perbandingan TP dan FN sebesar (14425:1526) dan algoritma *Decision Tree* memiliki perbandingan TP dan FN sebesar (12677:3310).

Selanjutnya pada penilaian *f1-score* ditemukan perbedaan antara nilai *f1-score* yang dihitung menggunakan penghitungan manual dengan penghitungan program. Pada penghitungan manual, algoritma *Naïve Bayes* mampu mencapai angka 83.4% dan algoritma *Decision Tree* mampu



mencapai 76.9%. Pada penghitungan program, nilai *f1-score* dari algoritma *Naïve Bayes* yakni 87.86% dan dari algoritma *Decision Tree* sebesar 82.77%. Perbedaan ini terjadi dikarenakan *f1-score* merupakan nilai rerata harmonis dari nilai *recall* dan nilai presisi. Baik algoritma *Naïve Bayes* maupun *Decision Tree* pada penghitungan program dan penghitungan manual memiliki nilai *recall* dan presisi yang berbeda.

Terakhir, yakni luas area dibawah kurva ROC atau AUC-ROC pada penghitungan manual dan penghitungan program terdapat perbedaan. Pada penghitungan manual algoritma *Naïve Bayes* dan *Decision Tree* memiliki *threshold* atau batasan sebanyak 4 batasan yang ditentukan. Algoritma *Naïve Bayes* memiliki nilai AUC-ROC sebesar 0.625, dan algoritma *Decision Tree* memiliki nilai AUC-ROC sebesar 0.525. Sedangkan pada penghitungan program, batasan tidak hanya 4 namun berjumlah acak. Dalam penghitungan program, nilai AUC-ROC dari algoritma *Naïve Bayes* sebesar 0.8753 dan nilai AUC-ROC dari algoritma *Decision Tree* sebesar 0.8153.

Secara keseluruhan, terdapat perbedaan nilai-nilai akurasi, presisi, *recall*, *f1-score*, dan AUC ROC baik pada penghitungan program maupun penghitungan manual. Perbedaan nilai-nilai ini diakibatkan oleh perbedaan jumlah data uji yang mengakibatkan terjadinya perbedaan nilai TP, FP, dan FN, serta perbedaan *threshold* dari penghitungan AUC-ROC. Secara keseluruhan algoritma *Naïve Bayes* lebih unggul dibandingkan algoritma *Decision Tree*, baik pada penghitungan manual, maupun penghitungan program.

#### 4.3.2. Evaluasi dengan penelitian lainnya

Mirip dengan penelitian (Rasi Nuraeni, et al., 2021) yang melakukan perbandingan algoritma klasifikasi menggunakan *Naïve Bayes Classifier* dan algoritma *Decision Tree* dalam menganalisis sistem klasifikasi judul skripsi.

Tabel 4.4 Perbandingan hasil evaluasi penelitian ini dengan penelitian (Nuraeni, R. et al, 2021).

Jenis Penilaian	Hasil Penelitian Penulis		Hasil Penelitian (Nuraeni, R. et al, 2021)	
	<i>Naïve Bayes</i>	<i>Decision Tree</i>	<i>Naïve Bayes</i>	<i>Decision Tree</i>
Akurasi	87.52%	81.59%	83.33%	60.33%
Presisi	90.81%	83.31%	54.81%	20%
Recall	85.10%	82.24%	59.93%	33%

Penelitian (Nuraeni, R., et al., 2021) mendapatkan nilai evaluasi pada algoritma *Naïve Bayes* yakni nilai akurasi sebesar 83%, nilai presisi sebesar 54%, dan nilai *Recall* sebesar 59%. Nilai-nilai tersebut lebih besar dibandingkan dengan nilai evaluasi pada algoritma *Decision Tree* yakni nilai akurasi 60%, nilai presisi 20%, dan nilai *recall* yang hanya mencapai 33%.

Penelitian yang dilakukan (Isnain, et al., 2022) mendapatkan nilai evaluasi menggunakan algoritma *Naïve Bayes* memiliki nilai akurasi 82%, dengan nilai presisi 82%, nilai *recall* sebesar 82% dan nilai *F1-score* sebesar 82%. (Isnain, et al., 2022) mendapatkan hasil yang lebih tinggi menggunakan algoritma LSTM, yang menghasilkan nilai akurasi 83%, presisi 83%, nilai *recall* 83%, dan nilai *f1-score* 83%.

Pada penelitian (Hozairi, et al., 2020), dengan menggunakan algoritma *Naïve Bayes*, dihasilkan nilai akurasi yang tinggi yaitu 89%, dengan nilai presisi yang juga tinggi yakni 88%, nilai *recall* yang mencapai 96%, serta nilai *f1-score* 91% dan nilai AUC-ROC yang berada di 0.562. Dengan menggunakan algoritma *Decision Tree* didapatkan nilai akurasi yang cukup rendah yaitu 74% dengan nilai presisi yang berada di angka 85%, nilai *recall* yang berada di angka 85%, serta nilai *f1-score* yang berada di angka 85% dan nilai AUC-ROC berada di angka 0.045.

Tabel 4.5 Perbandingan hasil evaluasi penelitian ini dengan penelitian (Hozairi, Anwari, Alim S., 2020).

Jenis Penilaian	Hasil Penelitian Penulis		Hasil Penelitian (Hozari, et al., 2020)	
	<i>Naïve Bayes</i>	<i>Decision Tree</i>	<i>Naïve Bayes</i>	<i>Decision Tree</i>
<b>Akurasi</b>	87.52%	81.59%	89%	74%
<b>Presisi</b>	90.81%	83.31%	88%	85%
<b>Recall</b>	85.10%	82.24%	96%	85%
<b>F1-Score</b>	87.86%	82.77%	91%	85%
<b>AUC-ROC</b>	0.8753	0.8153	0.987	0.769

Baik penelitian (Hozari, et al., 2020), maupun penelitian (Nuraeni, R., et al., 2021) selaras dengan penelitian ini dimana nilai akurasi, presisi, *recall*, *f1-score* dan nilai AUC-ROC dari algoritma *Naïve Bayes* lebih tinggi dibandingkan dengan algoritma *Decision Tree*.