Import Library

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import re
import string
import nltk
from nltk.sentiment.vader import  SentimentIntensityAnalyzer
from googleapiclient.discovery import build
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics  import confusion_matrix
from sklearn.metrics import classification_report,confusion_matrix
```

Scrapping Data

```
!pip install youtube-scraper
```

Import Paket

```
def video_comments(video_id):
    # empty list for storing reply
    replies = []

    # creating youtube resource object
    youtube = build('youtube', 'v3', developerKey=api_key)

    # retrieve youtube video results
    video_response = youtube.commentThreads().list(part='snippet,replies', videoId=video_ic

    # iterate video response
    while video_response:

        # extracting required info
        # from each result object
        for item in video_response['items']:

            # Extracting comments ()
            published = item['snippet']['topLevelComment']['snippet']['publishedAt']
            user = item['snippet']['topLevelComment']['snippet']['authorDisplayName']

            # Extracting comments
            comment = item['snippet']['topLevelComment']['snippet']['textDisplay']
            likeCount = item['snippet']['topLevelComment']['snippet']['likeCount']

            replies.append([published, user, comment, likeCount])

            # counting number of reply of comment
            replycount = item['snippet']['totalReplyCount']

            # if reply is there
            if replycount>0:
                # iterate through all reply
                for reply in item['replies']['comments']:

                    # Extract reply
                    published = reply['snippet']['publishedAt']
                    user = reply['snippet']['authorDisplayName']
                    repl = reply['snippet']['textDisplay']
                    likeCount = reply['snippet']['likeCount']

                    # Store reply is list
                    #replies.append(reply)
                    replies.append([published, user, repl, likeCount])

            # print comment with list of reply
            #print(comment, replies, end = '\n\n')

            # empty reply list
            #replies = []

        # Again repeat
        if 'nextPageToken' in video_response:
            video_response = youtube.commentThreads().list(
                    part = 'snippet,replies',
                    pageToken = video_response['nextPageToken'],
                    videoId = video_id
                ).execute()
        else:
            break
    #endwhile
    return replies


# api key
api_key = 'AIzaSyCx5pWDM6VDEHYiAZlRh35qhDW9gEwAr3s'
video_id = "UQtvZgKmGJo" #isikan dengan kode / ID video

# Call function
comments = video_comments(video_id)

comments
```

```
df = pd.DataFrame(comments, columns=['Waktu', 'Penulis', 'Komentar', 'Suka'])
df
```

```
len(df.index)
```

```
df[['Waktu', 'Penulis', 'Komentar', 'Suka']].head()
```

```
df_new = df[['Waktu', 'Penulis', 'Komentar', 'Suka']]
df_sorted= df_new.sort_values(by='Waktu', ascending=True)
df_sorted.head()
```

```
df_scrape = df_sorted[['Waktu', 'Penulis', 'Komentar', 'Suka']]
```

```
df_scrape.head()
```

```
df_scrape.to_csv("yt_data.csv", index = False)
```

Filtering

```
!pip install Sastrawi
```

```
!pip install vaderSentiment
```

```
df01 = pd.read_csv('yt_data.csv')
df01.head(5)
```

```
df01.shape
```

```
#menambah variabel "label"
Sentiment = []
for index, row in df01.iterrows():
    if row['Suka'] > 5:
        Sentiment.append('Positif')
    elif row['Suka'] >= 3 and row['Suka'] < 5:
        Sentiment.append('Netral')
    else:
        Sentiment.append('Negatif')

df01['Sentiment'] = Sentiment
df01.head()
```

```
df01.shape
```

```
#menghilangkan variabel yang tidak dipakai
df01_data = df01.copy()
df01_data = df01.drop(columns= ['Waktu', 'Penulis'])
df01_data.head()
```

```
df01_data.shape
```

```
x = df01_data.iloc[:, 0].values
y = df01_data.iloc[:, -1].values
```

```
#memecah data test 30% dari  keseluruhan data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state=0)
```

```
x_train
```

```
len(x_train)
```

```
len(x)
```

```
x_test
```

```
len(x_test)
```

```
nltk.download('punkt')
nltk.download('stopword')
nltk.download('wordnet') #stemming
```

```
df01_data.Komentar
```

```
#clearning
#remove url
df01_data['Komentar'] =  df01_data['Komentar'].str.replace('https\S', '', case=False) #toke

#ubah teks jadi huruf kecil
df01_data['Komentar'] =  df01_data['Komentar'].str.lower() #case folding

#remove mention
df01_data['Komentar'] =  df01_data['Komentar'].str.replace('@\S+', '', case=False) #tokeni

#remove hastag
df01_data['Komentar'] =  df01_data['Komentar'].str.replace('#\S+', '', case=False) #tokeni

#remove next character
df01_data['Komentar'] =  df01_data['Komentar'].str.replace("\'W+", '', case=False) #tokeni

#remove punctuation
df01_data['Komentar'] =  df01_data['Komentar'].str.replace('[^\w\s]', '', case=False) #toke

#remove number
df01_data['Komentar'] =  df01_data['Komentar'].str.replace(r'w*\d+\w*', '', case=False) #t

#remove spasi berlebih
df01_data['Komentar'] =  df01_data['Komentar'].str.replace('\s(2)', '', case=False) #token


df01_data.Komentar


#tokenizing (proses penguraian deskripsi dari kalimat jadi kata)
#testing
from nltk.tokenize import word_tokenize

x = df01_data.iloc[0]
print(nltk.word_tokenize(x['Komentar']))


def identify_token(row) :
  text = row['Komentar']
  tokens = nltk.word_tokenize(text)
  token_words = [w for w in tokens if w.isalpha()]
  return token_words

df01_data['Komentar'] = df01_data.apply(identify_token, axis = 1)
df01_data.Komentar


#stemming (kata dasar) (tahap cari root kata dari kata filtering)
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
stemming = PorterStemmer()
```

```
def stem_list(row) :
  text =  row ['Komentar']
  stem = [stemming.stem(word) for word in text]
  return(stem)

df01_data['Komentar'] = df01_data.apply(stem_list, axis=1)
df01_data.Komentar
```

```
#stopword (hapus kata tidak penting)
from nltk.corpus import  stopwords
import nltk
nltk.download('stopwords')
#from nltk.tokenize import word_tokenize
stops =  set (stopwords.words ('indonesian'))
```

```
df01_data.head()
```

```
df01_data['Sentiment'].value_counts()
```

Term Weighting

```
#menghitung vector

Komentar = df01['Komentar']

tfidfvectorizer = TfidfVectorizer(analyzer='word')
tfidf_wm = tfidfvectorizer.fit_transform(Komentar)
tfidf_tokens = tfidfvectorizer.get_feature_names_out()
df_tfidfvect = pd.DataFrame.sparse.from_spmatrix(tfidf_wm, index=Komentar.index, columns=t

print("\nTD-IDF Vectorizer\n")
print(df_tfidfvect)
```

```
#mengubah jadi vector term
tv =  TfidfVectorizer()
X_train = tv.fit_transform(x_train)
X_test = tv.transform(x_test)
```

```
X_test[1:1]
```

```
#menggunakan perhitungan tf idf
x = df01_data['Sentiment']
y = df01_data['Sentiment']

x_train, x_test, y_train, y_test  = train_test_split(x, y, test_size = 0.3, random_state=0
```

```
print("Matriks TF-IDF data pelatihan:")
print(X_train.toarray())
```

```
     Matriks TF-IDF data pelatihan:
     [[0. 0. 0. ... 0. 0. 0.]
      [0. 0. 0. ... 0. 0. 0.]
      [0. 0. 0. ... 0. 0. 0.]
      ...
      [0. 0. 0. ... 0. 0. 0.]
      [0. 0. 0. ... 0. 0. 0.]
      [0. 0. 0. ... 0. 0. 0.]]
```

```
print("Matriks TF-IDF data uji:")
print(X_test.toarray())
```

```
     Matriks TF-IDF data uji:
     [[0. 0. 0. ... 0. 0. 0.]
      [0. 0. 0. ... 0. 0. 0.]
      [0. 0. 0. ... 0. 0. 0.]
      ...
      [0. 0. 0. ... 0. 0. 0.]
      [0. 0. 0. ... 0. 0. 0.]
      [0. 0. 0. ... 0. 0. 0.]]
```

```
x_train
```

```
len(x_train)
```

```
x_test
```

```
len(x_test)
```

Menentukan Klasifikasi K-NN

```
# Pembuatan model
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train, y_train)
```

```
▾        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

```
# Prediksi menggunakan data test
y_pred = classifier.predict(X_test)
```

```
!pip install seaborn matplotlib
```

```
# Hitung akurasi
accuracy_score = metrics.accuracy_score(y_test, y_pred)
accuracy_score = round(accuracy_score * 100, 2)
print('Accuracy: ' + str(accuracy_score) + '%')
```

```
     Accuracy: 86.84%
```

```
#menghitung presisi
macro_precision = (metrics.precision_score(y_test, y_pred, average='macro'))
macro_precision = round(macro_precision * 100, 2)
print('Precision : ' + str(macro_precision) +'%')
```

```
     Precision : 45.23%
```

```
#menghitung recall
macro_recall = (metrics.recall_score(y_test, y_pred, average='macro'))
macro_recall = round(macro_recall * 100, 2)
print('Recall : '+str(macro_recall) +'%')
```

```
     Recall : 33.85%
```

```
#menghitung f1-score
macro_f1 = metrics.f1_score(y_test, y_pred, average='macro')
macro_f1_percentage = round(macro_f1 * 100, 2)
print('F1 : ' + str(macro_f1_percentage) +'%')
```

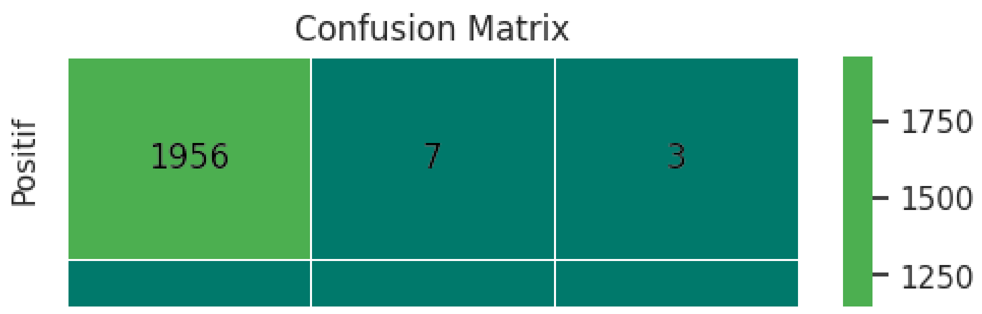
```
     F1 : 32.29%
```

```
# Menghitung confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
     Confusion Matrix:
     [[1956    7    3]
      [ 100    1    0]
      [ 183    4    2]]

     Classification Report:
                   precision    recall  f1-score   support

          Negatif       0.87      0.99      0.93      1966
           Netral       0.08      0.01      0.02       101
          Positif       0.40      0.01      0.02       189

         accuracy                           0.87      2256
        macro avg       0.45      0.34      0.32      2256
     weighted avg       0.80      0.87      0.81      2256
```

```
labels = ['Positif', 'Netral', 'Negatif']

conf_matrix = np.array([[1956, 7, 3],
                        [100, 1, 0],
                        [183, 4, 2]])

# Buat DataFrame dari confusion matrix
conf_matrix_df = pd.DataFrame(conf_matrix, index=labels, columns=labels)
```

```
custom_palette = sns.color_palette(["#00796B", "#4CAF50"])

# Plotting the heatmap
plt.figure(figsize=(6, 4))
sns.set(font_scale=1)
heatmap = sns.heatmap(conf_matrix_df, annot=True, fmt="d", cmap=custom_palette, linewidths=.

# Tampilkan tulisan pada heatmap
for i in range(len(labels)):
    for j in range(len(labels)):
        if conf_matrix_df.iloc[i, j] > 0:
            heatmap.text(j + 0.5, i + 0.5, str(conf_matrix_df.iloc[i, j]),
                         ha='center', va='center', color='black')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```


Confusion Matrix
Positif