Sentiment Analysis movie review

Tasneem Mohamed Ahmed Mohamed	*2022170105*
Bsmala Tarek Kamal Khalil ElBagoury	*2022170094*
Amira Mostafa Haroon AbdelWahaab	*2022170077*
Shahd Sherif Wagdy Khalifa	*2022170207*
Nora Ahmed Salem Ahmed	*2022170630*
Mennatullah Alaa Ahmed	*2021170627*

Download & Load data :

```
url = "http://www.cs.cornell.edu/people/pabo/movie-review-data/review_polarity.tar.gz"
output_csv = "data_set.csv"
print("Downloading dataset...")
response = requests.get(url)
if response.status_code != 200:
    raise Exception("Failed to download the dataset")
print("Extracting dataset...")
tar = tarfile.open(fileobj=BytesIO(response.content), mode="r:gz")
tar.extractall(path="movie_reviews")
tar.close()
reviews = []
labels = []
base_dir = "movie_reviews/txt_sentoken"
pos_dir = os.path.join(base_dir, "pos")
for filename in os.listdir(pos_dir):
    if filename.endswith(".txt"):
        with open(os.path.join(pos_dir, filename), "r", encoding="utf-8") as file:
            review = file.read().strip()
            reviews.append(review)
            labels.append(1)
neg_dir = os.path.join(base_dir, "neg")
for filename in os.listdir(neg_dir):
    if filename.endswith(".txt"):
        with open(os.path.join(neg_dir, filename), "r", encoding="utf-8") as file:
            review = file.read().strip()
            reviews.append(review)
            labels.append(0)
print("Creating CSV file...")
data = {"review": reviews, "sentiment": labels}
df = pd.DataFrame(data)
df.to_csv(output_csv, index=False, encoding="utf-8")
```

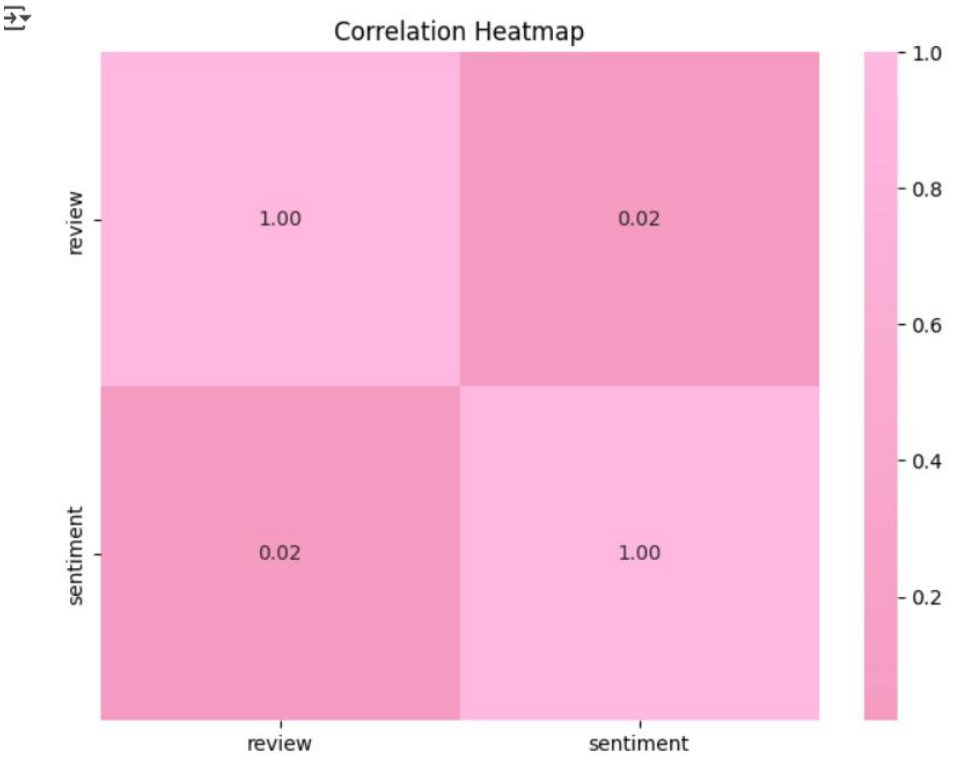
Read file :

```
dataf = pd.read_csv("/content/data_set.csv")
```

Correlation Matrix :

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
original_encodings = {}
categorical_columns = ['review']
label_encoder = LabelEncoder()
for column in categorical_columns:
    dataf[column] = label_encoder.fit_transform(dataf[column])
    original_encodings[column] = label_encoder

correlation_matrix = dataf.corr()
colors = ['#EC7FA2', '#FFB8E0']
cmap = LinearSegmentedColormap.from_list('custom_cmap', colors)
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap=cmap,
fmt=".2f", center=0)
plt.title('Correlation Heatmap')
plt.show()
```



Undo Encoding :

```
for column in categorical_columns:
    label_encoder = original_encodings[column]
    dataf[column] =
label_encoder.inverse_transform(dataf[column])
print(dataf.head())
```

“Preprcessing & Handling data”

1-convert into lowercase

```
dataf = dataf.apply(lambda x: x.str.lower() if x.dtype ==
"object" else x)
dataf.head()
```

2-Remove Duplicates :

```
def remove_word_duplicates(text):

    words = word_tokenize(text)
    seen = set()
    new_words = []
    for word in words:
        if word.lower() not in seen:
            seen.add(word.lower())
            new_words.append(word)
    return ' '.join(new_words)
```

```
df['review'] = df['review'].apply(remove_word_duplicates)
```

3-Remove punctuation :

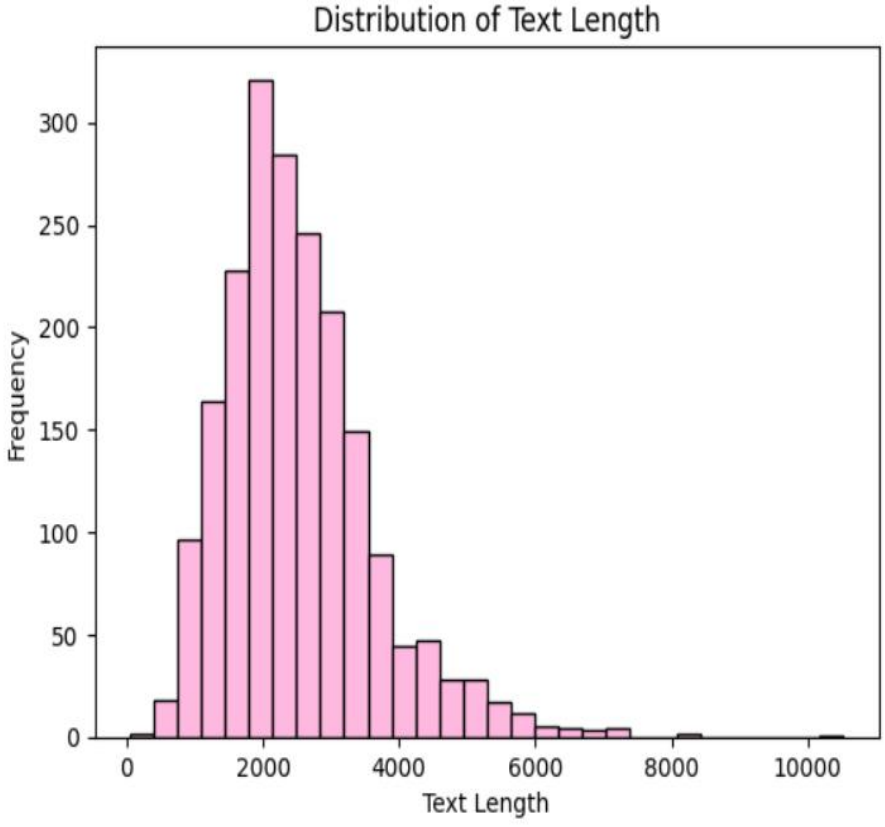
```
def remove_punctuation(text):
    return text.translate(str.maketrans('', '',
string.punctuation))
dataf = dataf.applymap(lambda x: remove_punctuation(str(x)) if
isinstance(x, str) else x)
dataf.head()
```

4- Remove Stopwords :

```
stop_words = set(stopwords.words('english'))
columns_to_remove_stopwords = ['review']
def remove_stopwords(text):
    return ' '.join([word for word in str(text).split() if
word.lower() not in stop_words])
for column in columns_to_remove_stopwords:
    dataf[column] = dataf[column].apply(remove_stopwords)
dataf.head()
```

Distribution of text length :

```
dataf['Text_Length'] = dataf['review'].apply(len)
print("\nDistribution of text length:")
print(dataf['Text_Length'].describe())
plt.hist(dataf['Text_Length'], bins=30, color='#FFB8E0',
edgecolor='black')
plt.xlabel('Text Length')
plt.ylabel('Frequency')
plt.title('Distribution of Text Length')
plt.show()
```



5- Normalization :

```
normalization_map = {
    'u': 'you',
    'ur': 'your',
    'r': 'are',
    'b4': 'before',
    'gr8': 'great',
    'gud': 'good',
    'pls': 'please',
    'plz': 'please',
    'thx': 'thanks',
    'l8r': 'later',
    'msg': 'message',
    'btw': 'by the way',
    'idk': 'i do not know',
    'imo': 'in my opinion'
}
def normalize_text(text):
    words = text.split()
    normalized_words = [normalization_map[word] if word in
normalization_map else word for word in words]
    normalized_text = ' '.join(normalized_words)
    return normalized_text
columns_to_normalize = ['review']
for column in columns_to_normalize:
    dataf[column] = dataf[column].apply(normalize_text)

dataf.head()
```

6- Lemmetaization :

```
dataf['review'] = dataf['review'].apply(lambda x:
nltk.word_tokenize(x))
stop_words = set(stopwords.words('english'))
dataf['review'] = dataf['review'].apply(lambda x: [word for
word in x if word not in stop_words])
lemmatizer = WordNetLemmatizer()
dataf['review'] = dataf['review'].apply(lambda x:
[lemmatizer.lemmatize(word) for word in x])
dataf['review'] = dataf['review'].apply(lambda x: ' '.join(x))
```

7- Stemmeing :

```
stemmer = PorterStemmer()
def stem_text(text):
    tokens = text.split()
    stemmed_tokens = [stemmer.stem(token) for token in tokens
if token not in stop_words]
    return ' '.join(stemmed_tokens)
dataf['review']= dataf['review'].apply(stem_text)
dataf.head()
```

8- Convert to string :

```
columns_to_convert = ['review']
for column in columns_to_convert:
    dataf[column] = dataf[column].astype(str)
dataf.head()
```

9- Frequency for each word :

```
all_joined_text = ' '.join(dataf['review'])
tokens = all_joined_text.split()
word_counts = Counter(tokens)
print("Most common words and their frequencies:")
for word, frequency in word_counts.most_common(20):
    print(f"{word}: {frequency}")
```

10- Split data & count vectorizer :

```
X_train, X_test, y_train, y_test =
train_test_split(dataf['review'], dataf['sentiment'],
test_size=0.2, random_state=0)
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

"Feature Extraction (TF-IDF)"

```
X = dataf['review']
y = dataf['sentiment']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
X_train = X_train.astype(str).tolist()
X_test = X_test.astype(str).tolist()
vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

"Apply ML Models"

1- SVM Model :

LinearSVC	*SVC_linear*	*SVC_RBF*
`param_grid': {'C': [0.01, 0.1, 1, 10, 100]}`	`param_grid': {'C': [0.01, 0.1, 1, 10, 100]}`	`param_grid': {'C': [0.01, 0.1, 1, 10, 100], 'gamma': ['scale', 'auto', 0.01, 0.1]}`
Train Accuracy : 0.98625 *Test Accuracy : 0.8775*	*Train Accuracy : 0.94812* *Test Accuracy : 0.85*	*Train Accuracy : 0.97062* *Test Accuracy : 0.86*

2- KNN Model :

Knn paramater grid :

```
knn_param_grid = {
    'clf__n_neighbors': [3, 5, 7, 9],
    'clf__weights': ['uniform', 'distance'],
    'clf__algorithm': ['auto', 'ball_tree', 'kd_tree',
'brute'],
    'clf__p': [1, 2]
}
```

Train Accuracy	Test Accuracy
1.0	0.72

3- Logistic Regression :

Train Accuracy	Test Accuracy
0.955625	0.8575

4- Naive Bayes :

Train Accuracy	Test Accuracy
0.92	0.8

5- Decision Tree :

Decision tree paramater grid :

```
dt_param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}
```

Train Accuracy	*Test Accuracy*
0.876875	*0.6725*

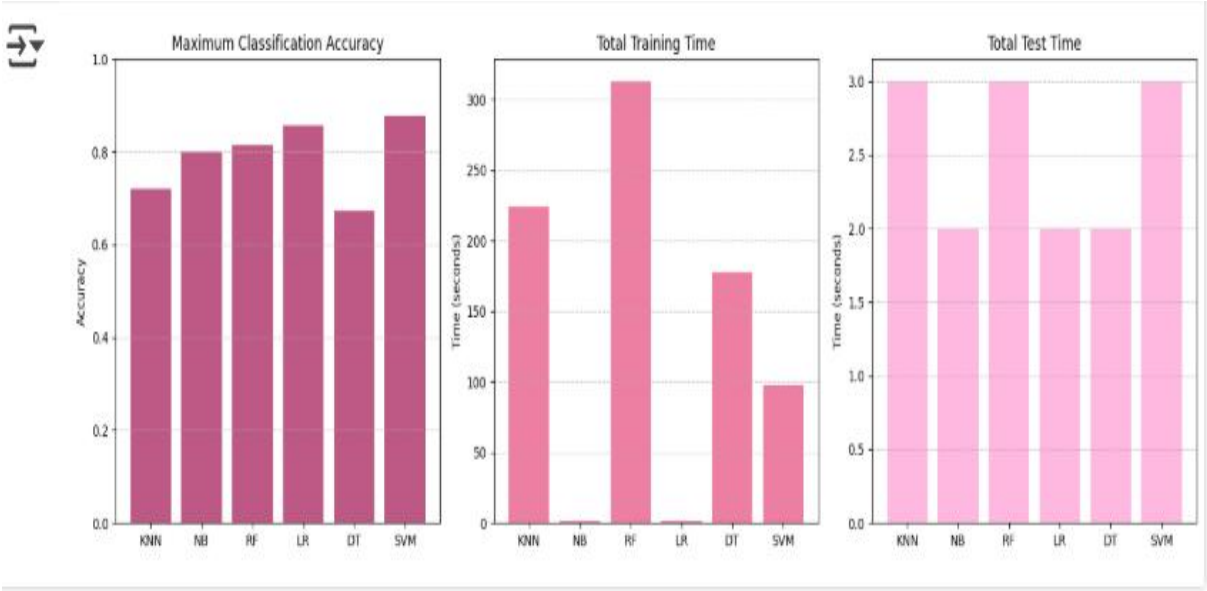
6 - Random Forest Model :

Random Forest paramater grid :

```
rf_param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'criterion': ['gini', 'entropy']
}
```

Train Accuracy	*Test Accuracy*
1.0	*0.815*

“Gragh models”



“Conclusion ”

The Best Model : SVM

```
SVM Model: with Kernel => Linear
 LinearSVC
 Best Parameters: {'C': 1}
 Best CV F1-Score: 0.8824663772734981
 Train Accuracy: 0.98625
 Test Accuracy: 0.8775
```