SENTIMENT ANALYSIS
LUXURY CARS REVIEWS

Mahmoud Al Hamad 1010767

Student Name(s)	Mahmoud Al Hamad
Consulted Tutor	Sini Rag Pulari
Subject	NLP
Github	https://github.com/Alhamad-Mahmoud/IT9002-NLP-Project

TABLE OF CONTENTS

PROBLEM STATEMENT DEFINITION

This Sentiment Analysis and Classification Project focuses on utilizing Natural Language Processing (NLP) techniques to analyze customer reviews. The primary objective is to predict whether customers recommend the purchased product based on the information in their review text.

Key challenges include extracting valuable information from the "Review" variable using text mining techniques and converting text files into numeric feature vectors for machine learning algorithms.

The project covers a comprehensive exploration of the dataset through detailed Exploratory Data Analysis (EDA) and visualization. It also involves building sentiment classification models using machine learning algorithms (Naive Bayes, Support Vector Machine) and deep learning algorithms.

MOTIVATION

The project's significance lies in its pivotal role within the automotive industry, where analyzing customer reviews can shape the trajectory of brand reputation and market competitiveness.

By predicting customer recommendations based on reviews, the project offers actionable insights, enabling manufacturers to refine product offerings and enhance service quality. This approach contributes to the advancement of Natural Language Processing (NLP) methodologies, showcasing the application of sophisticated techniques for sentiment analysis on extensive datasets.

Beyond the automotive domain, the societal impact is evident as the project fosters transparency in consumer decision-making, ensuring accurate sentiment predictions for informed choices. The outcomes of this endeavor hold potential to transform not only how manufacturers perceive customer satisfaction but also how consumers navigate a marketplace increasingly influenced by online reviews.

PROBLEM STATEMENT

The project addresses the challenge of effectively analyzing automotive customer reviews, aiming to predict recommendations based on Edmunds-Consumer Car Ratings. Extracting meaningful insights from textual reviews is crucial for understanding consumer sentiments and influencing purchasing decisions in the automotive industry.

PROPOSED SOLUTION

The solution leverages NLP techniques like text mining, employing machine learning (Naive Bayes, SVM) and deep learning algorithms. By combining these approaches, the project aims to build robust sentiment classification models that enhance the accuracy of predicting customer recommendations, providing valuable insights for manufacturers to improve their products.

IMPLEMENTATION and IMPACT

The solution's implementation involves in-depth Exploratory Data Analysis (EDA) and visualization, ensuring a comprehensive understanding of the dataset's features. This project not only contributes to NLP advancements but also has the potential to reshape feedback analysis in the automotive industry. The informed decision-making facilitated by this approach holds significance for both consumers and manufacturers, impacting the industry's feedback analysis processes.

1) LIBRARIES NEEDED IN THE PROJECT

various Python libraries for data analysis, visualization, and machine learning used in this report. Here's a brief overview of the major libraries used:

NumPy (np): Fundamental package for scientific computing with support for large, multidimensional arrays and matrices.

Pandas (pd): Provides data structures like DataFrame for efficient data manipulation and analysis.

Seaborn (sns) and Matplotlib (plt): Data visualization libraries for creating attractive and informative statistical graphics.

Re (re): Regular expression operations for string manipulation.

Plotly Express (px) and Graph Objects (go): Interactive plotting libraries for creating visually appealing plots and graphs.

SciPy Stats (stats): Library for scientific and technical computing, including statistical functions.

StatsModels (sm) and Formula API (smf): Provides classes and functions for statistical models' estimation.

Scikit-Learn (sklearn): A machine learning library that includes various classification and model selection tools.

NLTK (nltk): Natural Language Toolkit for working with human language data. It includes tokenization, stemming, lemmatization, and more.

WordCloud: Generates word clouds, visual representations of word frequency in a given text.

Yellowbrick: Visualization library that extends Scikit-Learn to provide visual diagnostic tools.

Cufflinks (cf): Connects Plotly with Pandas for interactive visualizations in offline mode.

Warnings (warnings): Provides control over warning messages in Python.

IPyWidgets (ipywidgets): Interactive HTML widgets for Jupyter notebooks.

Termcolor and Colorama: Libraries for adding color to terminal text for better readability.

```
# Importing necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import re
import plotly.express as px
import plotly.graph_objects as go
import scipy.stats as stats
import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, f1_score, recall_score
from sklearn.metrics import precision_recall_curve, average_precision_score
from yellowbrick.classifier import PrecisionRecallCurve

import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, PorterStemmer
from collections import Counter
from wordcloud import WordCloud

# Importing plotly and cufflinks
import cufflinks as cf
import plotly.offline
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
warnings.warn("this will not show")
```

```
# Importing plotly and cufflinks
import cufflinks as cf
import plotly.offline
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
warnings.warn("this will not show")

# Figure & Display options
%matplotlib inline
fig, ax = plt.subplots()
plt.rcParams["figure.figsize"] = (12, 8)
pd.set_option('max_colwidth', 200)
pd.set_option('display.max_rows', 1000)
pd.set_option('display.max_columns', 200)
pd.set_option('display.float_format', lambda x: '%.2f' % x)


import colorama
from colorama import Fore, Style
from termcolor import colored

import ipywidgets
from ipywidgets import interact
from IPython.display import Image, display
```

1.1 User Defined Function

I have defined some useful user defined functions.

```
###############################################################################

def analyze_missing_values(data_frame):
    missing_count = data_frame.isnull().sum().sort_values(ascending=False)
    missing_percentage = (data_frame.isnull().sum() / data_frame.isnull().count()).sort_values(ascending=False)
    missing_info = pd.concat([missing_count, missing_percentage], axis=1, keys=['Missing_Count', 'Missing_Percentage'])
    return missing_info[missing_info['Missing_Count'] > 0]

###############################################################################

def explore_data(data_frame):
    print(colored("Data Shape:", attrs=['bold']), data_frame.shape, '\n',
          colored('-' * 79, 'red', attrs=['bold']),
          colored("\nData Info:\n", attrs=['bold']), sep='')
    print(data_frame.info(), '\n',
          colored('-' * 79, 'red', attrs=['bold']), sep='')
    print(colored("Number of Uniques:\n", attrs=['bold']), data_frame.nunique(), '\n',
          colored('-' * 79, 'red', attrs=['bold']), sep='')
    print(colored("Missing Values Summary:\n", attrs=['bold']), analyze_missing_values(data_frame), '\n',
          colored('-' * 79, 'red', attrs=['bold']), sep='')
    print(colored("All Columns:", attrs=['bold']), list(data_frame.columns), '\n',
          colored('-' * 79, 'red', attrs=['bold']), sep='')

    data_frame.columns = data_frame.columns.str.lower().str.replace('&', '_').str.replace(' ', '_')

    print(colored("Columns after Rename:", attrs=['bold']), list(data_frame.columns), '\n',
          colored('-' * 79, 'red', attrs=['bold']), sep='')


def check_multicollinearity(data_frame):
    features = []
    collinear_features = []
    for col in data_frame.corr().columns:
        for i in data_frame.corr().index:
            if (abs(data_frame.corr()[col][i]) > .9 and abs(data_frame.corr()[col][i]) < 1):
                features.append(col)
                collinear_features.append(i)
                print(colored(f"Multicollinearity alert between {col} and {i}",
                              "red", attrs=['bold']), data_frame.shape, '\n',
                      colored('-' * 79, 'red', attrs=['bold']), sep='')
```

```python
def drop_selected_columns(data_frame, columns_to_drop):
    if columns_to_drop != []:
        data_frame.drop(columns_to_drop, axis=1, inplace=True)
        print(columns_to_drop, 'were dropped')
    else:
        print(colored('Checking for missing values and dropping related columns if necessary...', attrs=['bold']), '\n',
              colored('-' * 79, 'red', attrs=['bold']), sep='')

def drop_columns_with_nulls(data_frame, limit):
    print('Data Shape:', data_frame.shape)
    for col in data_frame.isnull().sum().index:
        if (data_frame.isnull().sum()[col] / data_frame.shape[0] * 100) > limit:
            print(data_frame.isnull().sum()[col], 'percent of', col, 'null and were dropped')
            data_frame.drop(col, axis=1, inplace=True)
            print('New shape:', data_frame.shape)
    print('New shape after missing value control:', data_frame.shape)

################################################################################

# To view summary information about a column

def summarize_column(column_name):
    print("Column Name     : ", column_name)
    print("--------------------------------")
    print("Percentage of Nulls   : ", "%", round(df[column_name].isnull().sum() / df.shape[0] * 100, 2))
    print("Number of Nulls       : ", df[column_name].isnull().sum())
    print("Number of Uniques     : ", df[column_name].nunique())
    print(df[column_name].value_counts(dropna=False))
```

2) DATA COLLECTION

2.1 Context

This is a dataset containing consumer's thought and the star rating of car manufacturer/model/type.

Why i select this dataset:

Currently, this dataset has data of 62 major brands. i only focus for my project on luxury car reviews within the dataset. By narrowing down the scope to luxury cars, i can delve deeper into understanding consumer sentiments, preferences, and experiences specific to high-end automotive brands. This targeted approach allows for more meaningful insights and analysis, particularly within the luxury car market segment.

Vehicles stand as a significant element of personal identity, reflecting one's preferences and lifestyle. The reviews provided by customers offer a glimpse into their experiences with different car models.

In this context, the basic goal of this project is to predict whether customers, especially assumed as man, recommend the car they purchased using the information in their *review*. Especially, it should be noted that the expectation in this project is to use only the "review" variable and neglect the other ones. Of course,

The data is a collection of 55165 Rows and 10 column variables. Each row includes a written comment as well as additional customer information. Also each row corresponds to a customer review, and includes the variables.

Dataset srouce :https://www.kaggle.com/datasets/ankkur13/edmundsconsumer-car-ratings-andreviews/data (https://www.kaggle.com/datasets/ankkur13/edmundsconsumer-car-ratings-andreviews/data)

2.2 About the Features

The dataset encompasses various features, including temporal aspects, author details, age distribution, vehicle make, model specifics, review titles, and ratings. These features offer a comprehensive view, enabling analyses ranging from sentiment trends to customer satisfaction and binary classification tasks based on vehicle recommendations.

**1) review_date (Categorical - Temporal): This column provides the timestamp of each review. Analyzing this data could reveal patterns related to when reviews are posted, potential seasonality, or trends over time.

**2) author_name (Categorical - Text): Contains the names of the individuals providing reviews. This information can be used to identify frequent reviewers, trends related to specific authors, or even sentiment analysis based on the author.

**3) age (Numerical - Continuous): Represents the age of the authors. Analyzing the distribution of ages can provide insights into the age groups that are more likely to review, and potential correlations between age and the given ratings

**4) vehicle_make (Categorical - Text): Specifies the make or brand of the vehicle being reviewed (e.g., Ferrari). This can be used to understand which vehicle brands are more popular or receive more reviews.

**5) vehicle_title (Categorical - Text): Provides information about the specific model or configuration of the vehicle. Analyzing this column can reveal which models are more frequently reviewed and the sentiments associated with them. **6) review_title: Categorical (Text) - Title given to the review.

**7) review_title (Categorical - Text): The titles given to the reviews. Analyzing these titles might give an overview of the reviewers' initial impressions or the main points of their reviews.

**8) rating (Numerical - Discrete): Numeric ratings assigned to the vehicles. Analyzing the distribution of ratings provides insights into the overall satisfaction of customers.

**9) recommended_ind (Categorical - Binary): Indicates whether the author recommends the vehicle (1) or not (0). This column is crucial for understanding customer satisfaction and can be used for binary classification tasks.

2.3 Target Variable

- In my project, the target variable is typically the sentiment label assigned to a piece of text "recommended_ind". Sentiment labels indicate the emotional tone expressed in the text it is type is binary (positive/negative)

3) ANALYSIS

3.1 Reading the Data

```
df0 = pd.read_csv('Luxury-Car-Review.csv')
df = df0.copy()
df.head(1)
```

	Unnamed: 0	review_date	author_name	age	vehicle_make	vehicle_title	review_title	review	rating	recommended_ind
0	0	on 04/28/17 08:08 AM (PDT)	Garrett Stites	33	Ferrari	2015 Ferrari 458 Italia Convertible Spider 2dr Convertible (4.5L 8cyl 7AM)	The best car around!	This car gets great gas mileage and is the best car you could buy.	5.00	1

4) EXPLORATORY DATA &VISUALIZATION

4.1 - A General Looking at the Data

Pandas profiling is an open source Python module with which we can quickly do an exploratory data analysis with just a few lines of code.

The function, explore_data performs an initial exploration of a DataFrame (df). Here's a breakdown of what the function does:

Display Shape and Info:

Prints the shape of the DataFrame. Prints information about the DataFrame using df.info().

Display Number of Uniques:

Prints the number of unique values for each column in the DataFrame.

Display Missing Values:

Prints information about missing values in the DataFrame using the missing_values function (which is assumed to be defined elsewhere).

Display All Columns:

Prints the names of all columns in the DataFrame.

Rename Columns:

Converts column names to lowercase. Replaces spaces with underscores. Replaces the ampersand (&) with an underscore. Display Columns after Rename:

Prints the names of all columns after the renaming process.

```
explore_data(df)
```

```
Data Shape:(55164, 10)
--------------------------------------------------------------------------------
Data Info:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55164 entries, 0 to 55163
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Unnamed: 0       55086 non-null  object 
 1   review_date      45127 non-null  object 
 2   author_name      42380 non-null  object 
 3   age              55164 non-null  int64  
 4   vehicle_make     55162 non-null  object 
 5   vehicle_title    42381 non-null  object 
 6   review_title     42379 non-null  object 
 7   review           42381 non-null  object 
 8   rating           39635 non-null  float64
 9   recommended_ind  55164 non-null  int64  
dtypes: float64(1), int64(2), object(7)
memory usage: 4.2+ MB
None
--------------------------------------------------------------------------------
Number of Uniques:
Unnamed: 0         20549
review_date        32048
author_name        34517
age                   77
vehicle_make          18
vehicle_title       4590
review_title       35325
review             42018
rating                33
recommended_ind        2
dtype: int64
--------------------------------------------------------------------------------
Missing Values Summary:
                Missing_Count  Missing_Percentage
rating                  15529                0.28
review_title            12785                0.23
author_name             12784                0.23
vehicle_title           12783                0.23
review                  12783                0.23
review_date             10037                0.18
Unnamed: 0                 78                0.00
vehicle_make                2                0.00
--------------------------------------------------------------------------------
All Columns:['Unnamed: 0', 'review_date', 'author_name', 'age', 'vehicle_make', 'vehicle_title', 'review_title', 'review', 'rating', 'recommended_ind']
--------------------------------------------------------------------------------
Columns after Rename:['unnamed:_0', 'review_date', 'author_name', 'age', 'vehicle_make', 'vehicle_title', 'review_title', 'review', 'rating', 'recommended_ind']
--------------------------------------------------------------------------------
```

```
df.head(1)
```

	unnamed:_0	review_date	author_name	age	vehicle_make	vehicle_title	review_title	review	rating	recommended_ind
0	0	on 04/28/17 08:08 AM (PDT)	Garrett Stites	33	Ferrari	2015 Ferrari 458 Italia Convertible Spider 2dr Convertible (4.5L 8cyl 7AM)	The best car around!	This car gets great gas mileage and is the best car you could buy.	5.00	1

```
df.shape
```

```
(55164, 10)
```

- Initially Data set Contains 55164 Rows and 10 colom
- I want to drop a column named "unnamed:_0" from my DataFram

```
df.drop("unnamed:_0", axis=1, inplace=True)
df.head(1)
```

	review_date	author_name	age	vehicle_make	vehicle_title	review_title	review	rating	recommended_ind
0	on 04/28/17 08:08 AM (PDT)	Garrett Stites	33	Ferrari	2015 Ferrari 458 Italia Convertible Spider 2dr Convertible (4.5L 8cyl 7AM)	The best car around!	This car gets great gas mileage and is the best car you could buy.	5.00	1

- Producing a visually appealing representation of the summary statistics with a background gradient

```
df.describe(percentiles=[.5]).T[['mean', 'std','min' ,'50%', 'count','max']]
```

	mean	std	min	50%	count	max
age	43.22	12.30	18.00	41.00	55164.00	99.00
rating	4.44	0.83	1.00	4.75	39635.00	5.00
recommended_ind	0.56	0.50	0.00	1.00	55164.00	1.00

- generating descriptive statistics of the object (categorical) columns in a DataFrame. It provides information such as the count, unique values, top value, and frequency for each categorical column.

```
df.describe(include='object').T
```

	count	unique	top	freq
review_date	45127	32048	5	937
author_name	42380	34517	John	143
vehicle_make	55162	18	BMW	10382
vehicle_title	42381	4590	2004 Audi A8 Sedan L quattro AWD 4dr Sedan (4.2L 8cyl 6A)	91
review_title	42379	35325	Great Car	316
review	42381	42018		267

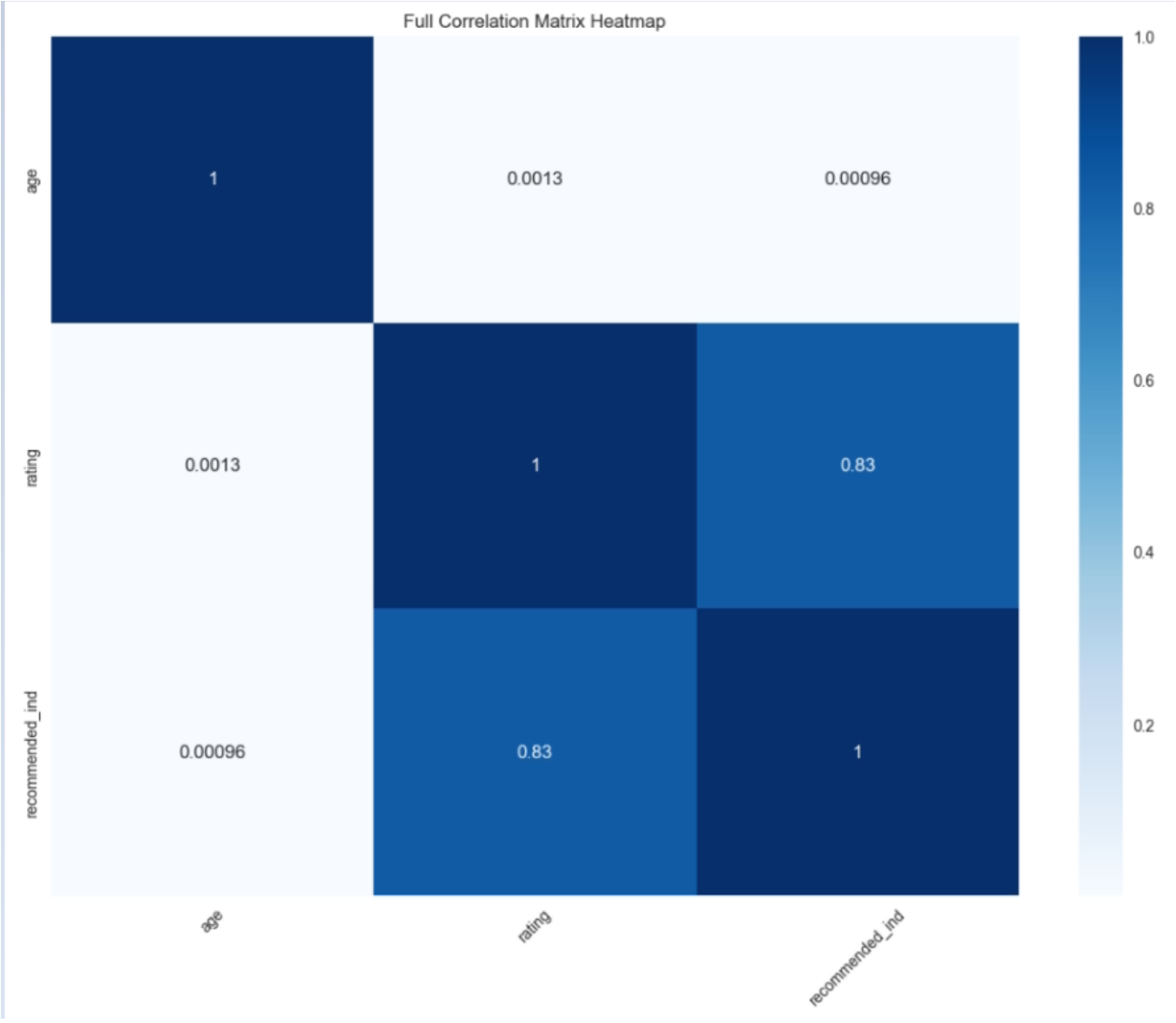
```python
# unique values for object features
for col in df.select_dtypes(include="object").columns:
    print(f"{col} feature has {df[col].nunique()} unique values.")
```

```
review_date feature has 32048 unique values.
author_name feature has 34517 unique values.
vehicle_make feature has 18 unique values.
vehicle_title feature has 4590 unique values.
review_title feature has 35325 unique values.
review feature has 42018 unique values.
```

```python
# unique values for numerical features
for col in df.select_dtypes(include=[np.number]).columns:
    print(f"{col} feature has {df[col].nunique()} unique values.")
```

```
age feature has 77 unique values.
rating feature has 33 unique values.
recommended_ind feature has 2 unique values.
```

```
plt.figure(figsize=(14, 10))

# Create a full correlation matrix
correlation_matrix = df.corr()

# Plot  a heatmap  blue color palette
sns.heatmap(correlation_matrix, annot=True, cmap='Blues')


plt.xticks(rotation=45)

plt.title('Full Correlation Matrix Heatmap')
plt.show()
```



- the correlation coefficients between different pairs of variables in a DataFrame. Here's how to interpret it:
- age vs. age: The correlation of a variable with itself is always 1. So, the correlation between 'age' and 'age' is 1.
- age vs. rating: The correlation coefficient between 'age' and 'rating' is 0.0013. This value is close to zero, indicating a very weak or negligible linear correlation between the age of the reviewers and the ratings they give.
- age vs. recommended_ind: The correlation coefficient between 'age' and
- 'recommended_ind' is 0.00096. Similar to the previous case, this value is close to zero, suggesting a very weak or negligible linear correlation between age and the recommendation indicator.

4.2 - The Examination of Target Variable

```
df.columns
```

```
Index(['review_date', 'author_name', 'age', 'vehicle_make', 'vehicle_title',
       'review_title', 'review', 'rating', 'recommended_ind'],
      dtype='object')
```

- Check Proportion of Target Class Variable:
- The target class variable is imbalanced, where "Recommended" values are more dominating then "Not Recommended".

```
df.head(2)
```

	review_date	author_name	age	vehicle_make	vehicle_title	review_title	review	rating	recommended_ind
0	on 04/28/17 08:08 AM (PDT)	Garrett Stites	33	Ferrari	2015 Ferrari 458 Italia Convertible Spider 2dr Convertible (4.5L 8cyl 7AM)	The best car around!	This car gets great gas mileage and is the best car you could buy.	5.00	1
1	on 11/19/11 16:47 PM (PST)	debu99	34	Ferrari	2006 Ferrari 612 Scaglietti Coupe F1 2dr Coupe (5.7L 12cyl 6AM)	keeps on beeing just great	Owning the 612 now over 3 years and using it in evey day traffic it just keeps on being great,The paint job is the only problem, technical it's fantastic and by far the best car i ever owned..	4.75	1

The Examination of "recommended_ind" Variable

- "recommended_ind" is a binary variable stating where the customer recommends the product where 1 is recommended, 0 is not recommended.This information provides a distribution of the binary values in the "recommended_ind" column. It seems that there are more occurrences of value 1 than value 0. In a binary classification context, this could imply that the dataset is somewhat imbalanced, with more instances of positive recommendations (1) than negative recommendations (0).

```
df["recommended_ind"].value_counts()
```

```
1    30963
0    24201
Name: recommended_ind, dtype: int64
```

```
summarize_column("recommended_ind")
```

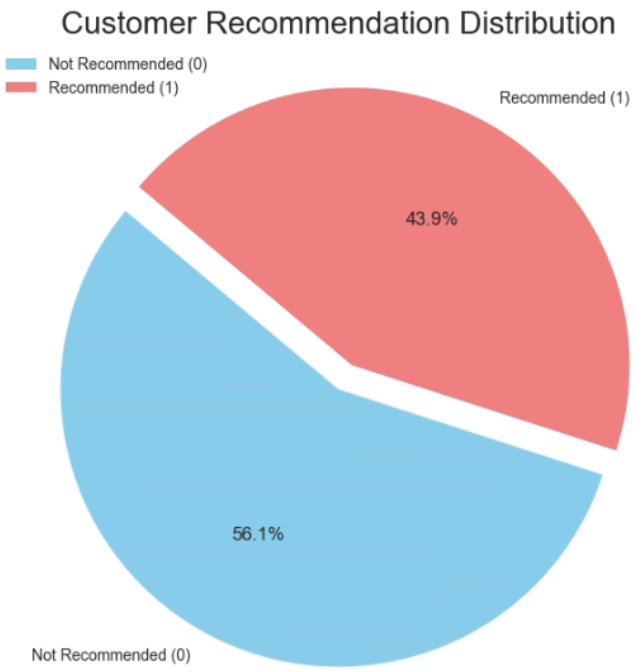
```
Column Name    :  recommended_ind
--------------------------------
Percentage of Nulls   :  % 0.0
Number of Nulls       :  0
Number of Uniques     :  2
1    30963
0    24201
Name: recommended_ind, dtype: int64
```

```
df["recommended_ind"].describe().T
```

```
count   55164.00
mean        0.56
std         0.50
min         0.00
25%         0.00
50%         1.00
75%         1.00
max         1.00
Name: recommended_ind, dtype: float64
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

# Use matplotlib bar for vertical bar chart with different colors
colors = ['skyblue', 'lightcoral']  # Add more colors if needed
bars = plt.bar(df.recommended_ind.value_counts().index, df.recommended_ind.value_counts().values, color=colors)

plt.title('Customer Recommendation Distribution', fontsize=30)
plt.xlabel("Recommendation Label", fontsize=24)
plt.ylabel("The Number of Recommendations", fontsize=24)

# Annotate the bars with their respective values
for index, value in enumerate(df.recommended_ind.value_counts().sort_values()):
    plt.text(index, value, f"{value}", ha="center", va="bottom", fontsize=13)

# Add legend with labels to the left
plt.legend(bars, ['Not Recommended (0)', 'Recommended (1)'], loc='upper left')

# Show the plot
plt.show()
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 8))  # Adjust the figure size if needed

# Use matplotlib pie chart with different colors
colors = ['skyblue', 'lightcoral']  # Add more colors if needed
explode = (0, 0.1)  # Explode the Recommended slice for emphasis

plt.pie(df.recommended_ind.value_counts().values, labels=['Not Recommended (0)', 'Recommended (1)'],
        autopct='%1.1f%%', startangle=140, colors=colors, explode=explode)

plt.title('Customer Recommendation Distribution', fontsize=20)

# Add legend
plt.legend(['Not Recommended (0)', 'Recommended (1)'], loc='upper left')

# Show the plot
plt.show()
```



4.3 - The Examination of Other Feature

- The Examination of "rating" Variable

"rating" is a positive ordinal integer variable representing the product score assigned by the customer, ranging from 1 (worst) to 5 (best).

```
df["rating"].value_counts().to_frame().T
```

	5.00	4.88	4.75	4.62	4.00	4.50	4.38	3.00	4.25	4.12	2.00	3.88	1.00	3.75	3.62	3.50	3.38	3.25	3.12	2.88	2.75	2.62	2.50	2.38	2.25	2.12
rating	12316	6720	4336	2657	2130	1821	1353	983	979	781	586	562	547	501	442	439	380	317	309	250	226	183	163	146	106	102

```
summarize_column("rating")
```

```
Column Name    :  rating
---------------------------------
Percentage of Nulls   :  % 28.15
Number of Nulls       :  15529
Number of Uniques     :  33
NaN     15529
5.00    12316
4.88     6720
4.75     4336
4.62     2657
4.00     2130
4.50     1821
4.38     1353
3.00      983
4.25      979
4.12      781
2.00      586
3.88      562
1.00      547
```

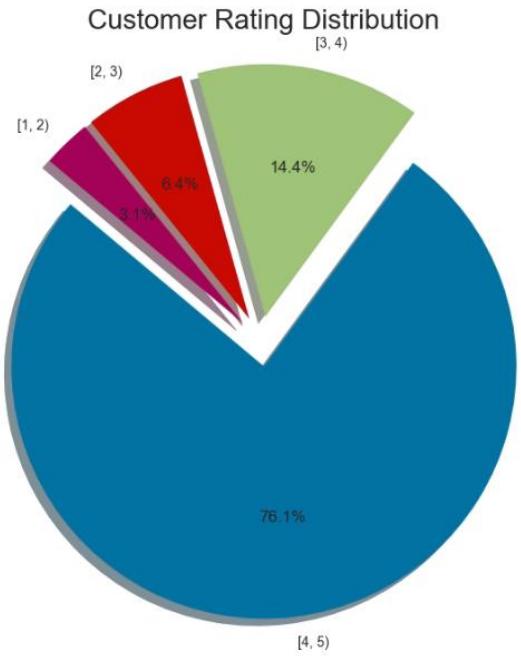
```
df["rating"].describe().T
```

```
count   39635.00
mean        4.44
std         0.83
min         1.00
25%         4.25
50%         4.75
75%         5.00
max         5.00
Name: rating, dtype: float64
```

```python
# Define custom rating ranges
bins = [1, 2, 3, 4, 5]

# Create a new column with rating ranges
df['rating_range'] = pd.cut(df['rating'], bins=bins, right=False)

# Count the occurrences in each rating range
rating_counts = df['rating_range'].value_counts().sort_index()

# Create a color map with shades of blue
colors = plt.cm.Blues(np.linspace(0.2, 1, len(rating_counts)))

# Create a bar chart using Matplotlib with custom colors
plt.figure(figsize=(10, 6))
rating_counts.plot(kind='bar', color=colors)

# Set plot title and labels
plt.title('Customer Rating Distribution', fontsize=16)
plt.xlabel("Rating Range", fontsize=14)
plt.ylabel("The Number of Ratings", fontsize=14)

# Rotate x-axis labels for rating ranges 1-2
plt.xticks(rotation=0)

# Add annotations on top of each bar
for index, value in enumerate(rating_counts):
    plt.text(index, value + 0.1, str(value), ha='center', va='bottom', fontsize=12)

# Show the plot
plt.show()
```

The distribution of customer ratings in the dataset indicates that a substantial majority, comprising 207,777 reviews, falls within the higher range of 4 to 5. This suggests a prevalent trend of positive feedback for the vehicles. Additionally, there are 3,933 reviews in the range of 3 to 4, 1,762 in the range of 2 to 3, and 847 in the lowest range of 1 to 2.

```
# Define custom rating ranges
bins = [1, 2, 3, 4, 5]

# Create a new column with rating ranges
df['rating_range'] = pd.cut(df['rating'], bins=bins, right=False)

# Count the occurrences of each rating range
rating_counts = df['rating_range'].value_counts()

# Plot pie chart
plt.figure(figsize=(8, 8))
explode = [0.1] * len(rating_counts)  # Explode all slices for emphasis

plt.pie(rating_counts, labels=rating_counts.index, explode=explode, autopct='%1.1f%%', shadow=True, startangle=140)
plt.title('Customer Rating Distribution', fontsize=20)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle

# Show the plot
plt.show()
```



The Examination of "age" Variable

"age" is a Positive Integer variable of the reviewer’s age.

```
df["age"].value_counts().to_frame().T
```

	39	35	36	34	38	37	41	33	46	48	42	32	44	40	43	31	47	53	45	29	49	56	52	28	26
age	2990	2184	2010	1868	1840	1831	1829	1707	1698	1504	1496	1483	1397	1388	1387	1335	1327	1279	1203	1199	1161	1096	1043	1010	963

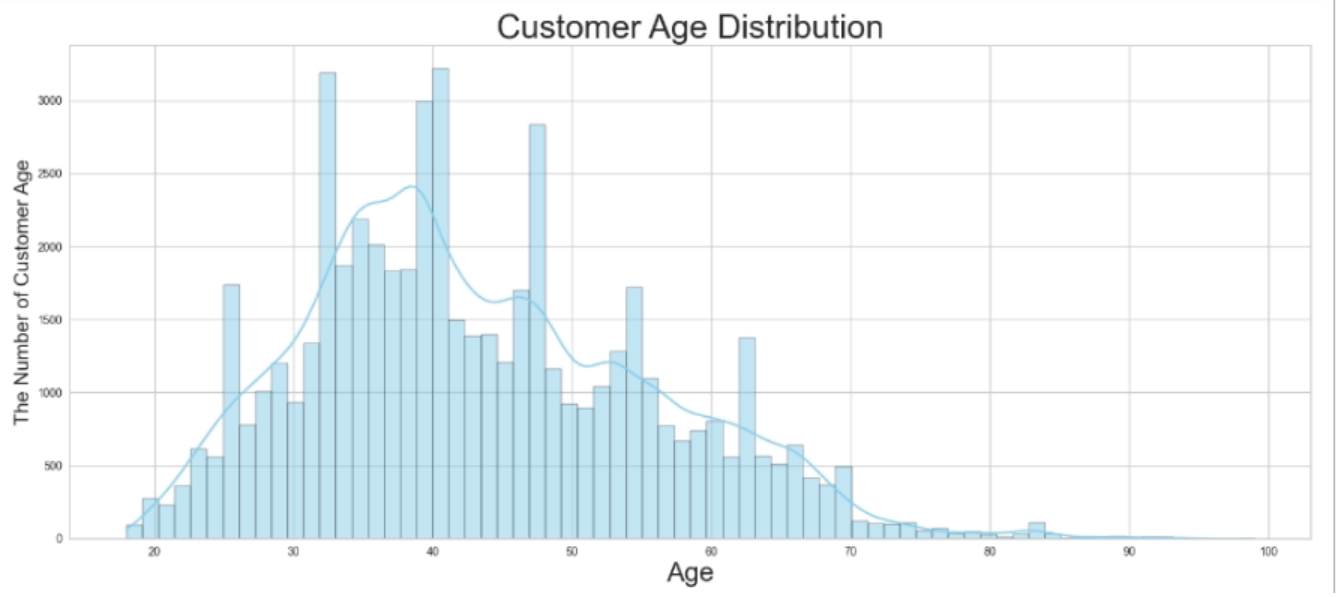
```
summarize_column("age")
```

```
Column Name    :  age
--------------------------------
Percentage of Nulls   :  % 0.0
Number of Nulls       :  0
Number of Uniques     :  77
39    2990
35    2184
36    2010
34    1868
38    1840
37    1831
41    1829
33    1707
46    1698
48    1504
42    1496
32    1483
44    1397
40    1388
```

```
df["age"].describe().T
```

```
count   55164.00
mean       43.22
std        12.30
min        18.00
25%        34.00
50%        41.00
75%        52.00
max        99.00
Name: age, dtype: float64
```

```
plt.figure(figsize=(20, 8))
plt.title('Customer Age Distribution', fontsize=30)
plt.xlabel("Age", fontsize=24)
plt.ylabel("The Number of Customer Age", fontsize=18)

# Use Seaborn to create a histogram with kde
sns.histplot(df, x='age', kde=True, bins=70, color='skyblue')

# Show the plot
plt.show()
```




```
df.columns
```

```
Index(['review_date', 'author_name', 'age', 'vehicle_make', 'vehicle_title',
       'review_title', 'review', 'rating', 'recommended_ind', 'rating_range'],
      dtype='object')
```

Here are some insights I can gather from the data:

Most Common Ages: The ages with the highest counts are 39, 35, 36, 34, and 38. These are the most common ages in the dataset.

Age Distribution: The data seems to have a relatively diverse distribution of ages, with a peak around the late 30s.

Decreasing Count with Age: As I move towards higher ages, the count tends to decrease, which is expected as the population generally decreases in older age groups.

Outliers: There is a notable drop in counts for ages beyond 70, indicating that there are fewer instances of individuals in those age groups in the dataset.

Sparse Age Groups: Some age groups, especially those in the late teens and early twenties, have lower counts, suggesting that the dataset might not be as representative for these age groups.

Age Group Variety: The dataset includes a variety of age groups, covering a broad range from 18 to 99.

- **The Examination of "vehicle_make" Variable.**

```
df["vehicle_make"].value_counts()
```

```
BMW             10382
Mercedes        10093
Toyota           8145
Audi             7663
Nissan           5270
Cadillac         5108
Jaguar           2477
Porsche          2429
Land Rover       2320
Mazerati          302
Ferrari           258
Bentley           174
Lamborghini       156
Tesla             140
Aston Martin      120
Alra Romeo         77
Rolls-Royce        39
Bugatti             9
Name: vehicle_make, dtype: int64
```

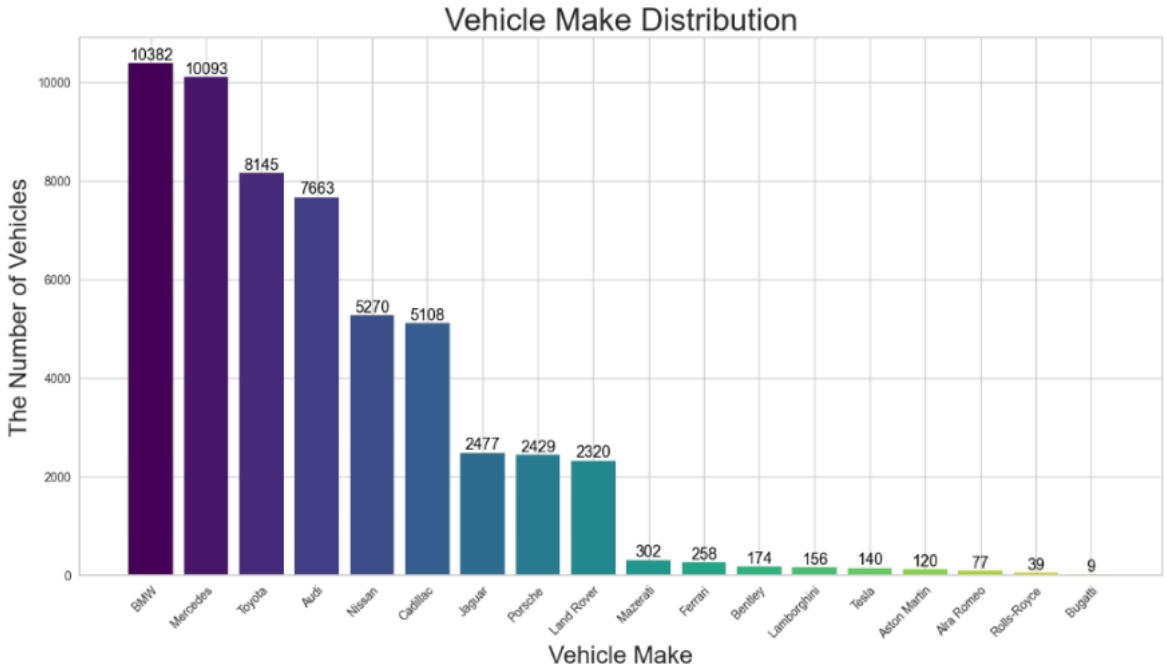
```
summarize_column("vehicle_make")
```

```
Column Name    :  vehicle_make
--------------------------------
Percentage of Nulls   :  % 0.0
Number of Nulls       :  2
Number of Uniques     :  18
BMW             10382
Mercedes        10093
Toyota           8145
Audi             7663
Nissan           5270
Cadillac         5108
Jaguar           2477
Porsche          2429
Land Rover       2320
Mazerati          302
Ferrari           258
Bentley           174
Lamborghini       156
Tesla             140
Aston Martin      120
Alra Romeo         77
Rolls-Royce        39
Bugatti             9
NaN                 2
Name: vehicle_make, dtype: int64
```

```
df["vehicle_make"].describe().T
```

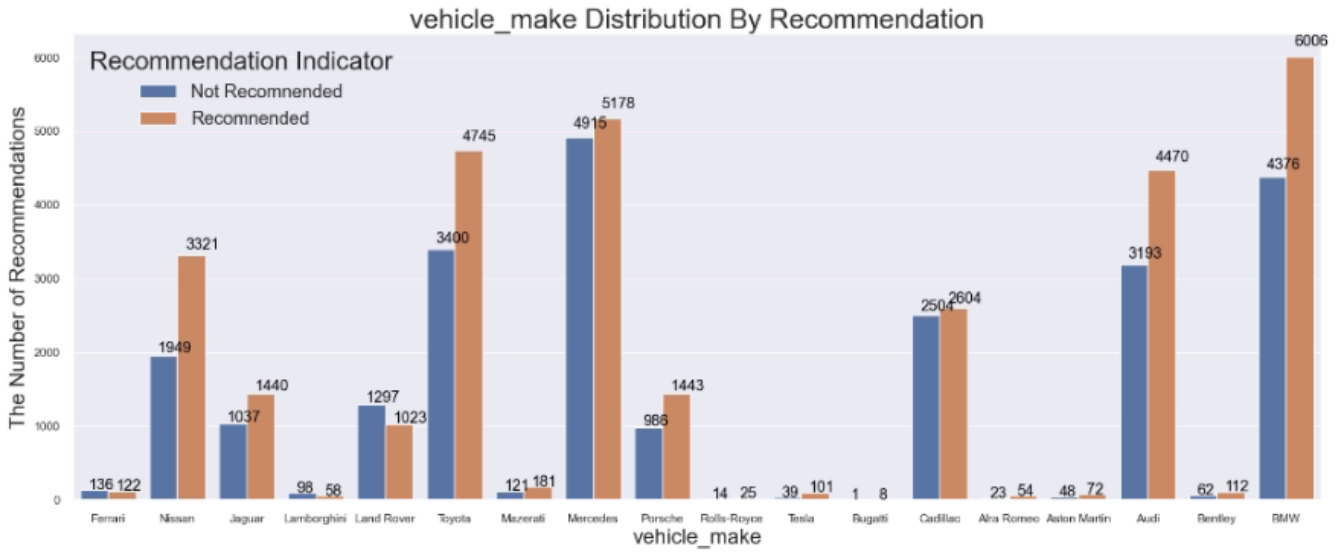
```
count     55162
unique       18
top         BMW
freq      10382
Name: vehicle_make, dtype: object
```

```
plt.figure(figsize=(16, 8))
vehicle_make_counts = df['vehicle_make'].value_counts()

# Define a color map with the number of colors equal to the number of unique vehicle makes
colors = plt.cm.viridis(np.linspace(0, 1, len(vehicle_make_counts)))

# Create a bar plot with different colors for each bar
bars = plt.bar(vehicle_make_counts.index, vehicle_make_counts.values, color=colors)

plt.title('Vehicle Make Distribution', fontsize=26)
plt.xlabel("Vehicle Make", fontsize=20)
plt.ylabel("The Number of Vehicles", fontsize=20)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha="right", rotation_mode="anchor")

# Add annotations inside each bar
for bar in bars:
    plt.text(bar.get_x() + bar.get_width() / 2., bar.get_height(),
             f"{int(bar.get_height())}", ha='center', va='bottom', fontsize=13, color='black')

plt.show()
```



The two largest counts for vehicle make in the dataset are BMW with 10,382 reviews and Mercedes with 10,093 reviews. These high counts suggest that BMW and Mercedes are popular choices among consumers, resulting in a significant number of reviews. On the other hand, the two lowest counts belong to Bugatti with only 9 reviews and Rolls-Royce with 39 reviews. The low review counts for Bugatti and Rolls-Royce may be attributed to the exclusivity and limited market presence of these luxury brands.

```
g = sns.catplot(data = df, x ="vehicle_make", hue = "recommended_ind", kind='count', height=7, aspect=2.5, legend_out=False)

plt.title('vehicle_make Distribution By Recommendation', fontsize=26)
plt.xlabel("vehicle_make", fontsize=20)
plt.ylabel("The Number of Recommendations", fontsize=20)
plt.legend(title='Recommendation Indicator', loc='upper left', labels=['Not Recomnended', 'Recomnended'], fontsize='x-large', tit

ax = g.facet_axis(0, 0)
for p in ax.patches:
    ax.text(p.get_x() + 0.12,
            p.get_height() * 1.025,
            '{0:.0f}'.format(p.get_height()),
            color='black', rotation='horizontal', size='large')

plt.show()
```



**Here are some observations:

**BMW (6006 rec, 4376 not):

BMW has a higher count of recommendations (6006) compared to non-recommendations (4376), indicating a positive trend. Mercedes (5178 rec, 4915 not):
Mercedes has a higher count of recommendations (5178), but the margin is narrower compared to BMW. The balance between recommendations and non-recommendations is relatively close.
Toyota (4745 rec, 3400 not):
Toyota also shows a positive trend with a higher count of recommendations (4745) compared to non-recommendations (3400). Ferrari (122 rec, 136 not):
Ferrari has a lower count of recommendations (122) compared to non-recommendations (136), which may indicate a more critical audience or specific issues mentioned in the reviews.
Lamborghini

creating categorical and numerical sets provides a structured approach to data analysis, enabling the use of specific tools and techniques tailored to each data type. This segregation enhances the efficiency and depth of the analysis, leading to more meaningful insights and informed decision-making.

- create categorical and numerical sets for the examination of crosstab information.

```
df_cat = df[['review_date', 'author_name', 'vehicle_make', "recommended_ind"]]
df_cat["recommended_ind"] = df_cat["recommended_ind"].apply(lambda x: "Recommended" if x>=1 else "Not Recommended")
df_cat.rename({'review_date': 'review_date', 'author_name': 'author_name', 'vehicle_make': 'vehicle_make', 'recommended_ind': 'Re
df_cat
```

	review_date	author_name	vehicle_make	Recommendation Indicator
0	on 04/28/17 08:08 AM (PDT)	Garrett Stites	Ferrari	Recommended
1	on 11/19/11 16:47 PM (PST)	debu99	Ferrari	Recommended
2	on 06/28/07 22:12 PM (PDT)	Arnell Baylet	Ferrari	Recommended
3	on 05/30/07 10:56 AM (PDT)	gregMTU	Ferrari	Recommended
4	on 01/21/07 12:58 PM (PST)	Darrel McOnnery	Ferrari	Recommended
...	...	...	...	...
55159	on 09/15/10 00:00 AM (PDT)	JORGE	BMW	Recommended
55160	on 05/27/10 14:20 PM (PDT)	Walter	BMW	Recommended
55161	on 11/28/16 21:17 PM (PST)	M5 owner	BMW	Recommended
55162	on 11/13/16 14:01 PM (PST)	Mike McManigal	BMW	Recommended
55163	on 03/13/16 18:22 PM (PDT)	K Adams	BMW	Recommended

55164 rows × 4 columns

```
df_num = df[['age', 'rating', 'recommended_ind']]
df_num["recommended_ind"] = df_num["recommended_ind"].apply(lambda x: "Recommended" if x>=1 else "Not Recommended")
df_num.rename({'age': 'Age', 'rating': 'Rating', 'positive_feedback_count': 'Positive Feedback', 'recommended_ind': 'Recommendati
df_num
```

	Age	Rating	Recommendation Indicator
0	33	5.00	Recommended
1	34	4.75	Recommended
2	60	5.00	Recommended
3	50	5.00	Recommended
4	47	5.00	Recommended
...	...	...	...
55159	51	4.88	Recommended
55160	50	4.12	Recommended
55161	25	5.00	Recommended
55162	59	5.00	Recommended
55163	36	5.00	Recommended

55164 rows × 3 columns

- generate and print cross-tabulations of categorical variables in a DataFrame

```
for i, col in enumerate(df_cat.columns):
    xtab = pd.crosstab(df_cat[col], df_cat["Recommendation Indicator"], normalize=True)

    # Print the separator line with color
    print(colored('-' * 55, 'red', attrs=['bold']))

    # Print the cross-tabulation multiplied by 100 for percentage display
    print(xtab * 100)
```

```
-------------------------------------------------------
Recommendation Indicator     Not Recommended  Recommended
review_date
 on 01/01/03 00:00 AM (PST)             0.01         0.01
 on 01/01/04 00:00 AM (PST)             0.01         0.01
 on 01/01/05 08:19 AM (PST)             0.00         0.00
 on 01/01/05 09:35 AM (PST)             0.00         0.00
 on 01/01/05 10:39 AM (PST)             0.00         0.00
...                                      ...          ...
4.5                                     0.29         0.00
4.625                                   0.43         0.00
4.75                                    0.78         0.00
4.875                                   1.43         0.00
5                                       2.08         0.00

[32048 rows x 2 columns]
-------------------------------------------------------
Recommendation Indicator  Not Recommended  Recommended
author_name
                                     0.01         0.02
  Z4  in Florida                     0.00         0.00
  moosie                             0.00         0.00
 2009 HSE LR3                        0.00         0.00
 Audi S5 Quattro                     0.00         0.00
...                                   ...          ...
zzdaf38                              0.00         0.00
zzzzzzz                              0.00         0.00
zzzzzzzzippy                         0.00         0.00
}{ {} }{                             0.00         0.00
§teve                                0.00         0.00

[34517 rows x 2 columns]
-------------------------------------------------------
Recommendation Indicator  Not Recommended  Recommended
vehicle_make
Alra Romeo                           0.04         0.10
Aston Martin                         0.09         0.13
Audi                                 5.79         8.10
BMW                                  7.93        10.89
Bentley                              0.11         0.20
Bugatti                              0.00         0.01
Cadillac                             4.54         4.72
Ferrari                              0.25         0.22
Jaguar                               1.88         2.61
Lamborghini                          0.18         0.11
Land Rover                           2.35         1.85
Mazerati                             0.22         0.33
Mercedes                             8.91         9.39
Nissan                               3.53         6.02
Porsche                              1.79         2.62
Rolls-Royce                          0.03         0.05
Tesla                                0.07         0.18
Toyota                               6.16         8.60
-------------------------------------------------------
Recommendation Indicator  Not Recommended  Recommended
Recommendation Indicator
Not Recommended                     43.87         0.00
Recommended                          0.00        56.13
```

```
for i, col in enumerate(df_num.columns):
    xtab = pd.crosstab(df_num[col], df_num["Recommendation Indicator"], normalize=True)
    print(colored('-'*55, 'red', attrs=['bold']), sep='')
    print(xtab*100)
```

```
-------------------------------------------------------
Recommendation Indicator  Not Recommended  Recommended
Age
18                                  0.01         0.00
19                                  0.06         0.09
20                                  0.20         0.29
21                                  0.18         0.23
22                                  0.31         0.34
23                                  0.49         0.62
24                                  0.46         0.55
25                                  0.62         0.78
26                                  0.76         0.99
27                                  0.58         0.82
28                                  0.80         1.04
29                                  1.01         1.16
30                                  0.75         0.94
31                                  1.05         1.37
32                                  1.18         1.51
33                                  1.32         1.78
```

5) TEXT PREPROCESSING

- From now on, the DataFrame I will work with should contains two columns: "review " and "recommended_ind". I can do the missing value detection operations from now on.

5.1 Feature Selection

```
df.columns
```

```
Index(['review_date', 'author_name', 'age', 'vehicle_make', 'vehicle_title',
       'review_title', 'review', 'rating', 'recommended_ind', 'rating_range'],
      dtype='object')
```

- For later parts of the analysis, I will drop unnecassary columns for NLP.keep only review and recommended_ind columns

```
df.drop(['review_date', 'author_name', 'age', 'vehicle_make', 'vehicle_title',
       'review_title', 'rating','rating_range'], axis=1, inplace=True)
```

```
df.head(1)
```

	review_date	author_name	age	vehicle_make	vehicle_title	review_title	review	rating	recommended_ind	rating_range
0	on 04/28/17 08:08 AM (PDT)	Garrett Stites	33	Ferrari	2015 Ferrari 458 Italia Convertible Spider 2dr Convertible (4.5L 8cyl 7AM)	The best car around!	This car gets great gas mileage and is the best car you could buy.	5.00	1	NaN

5.2 Dealing with Missing Values

```
df['review'].isnull().value_counts()
```

```
False    42381
True     12783
Name: review, dtype: int64
```

- It looks like I have 12,783 missing values (NaNs) in the 'review' column of my
- DataFrame. To handle these missing values, I can either drop the rows containing NaNs or fill them with a specific value, no missing values inrecommended_ind colum shown below.

```
df['recommended_ind'].isnull().value_counts()
```

```
False    55164
Name: recommended_ind, dtype: int64
```

- I will remove the rows where the 'review' column has NaN values.

```
df = df.dropna()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55164 entries, 0 to 55163
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   review_date     45127 non-null  object
 1   author_name     42380 non-null  object
 2   age             55164 non-null  int64
 3   vehicle_make    55162 non-null  object
 4   vehicle_title   42381 non-null  object
 5   review_title    42379 non-null  object
 6   review          42381 non-null  object
 7   rating          39635 non-null  float64
 8   recommended_ind 55164 non-null  int64
 9   rating_range    27319 non-null  category
dtypes: category(1), float64(1), int64(2), object(6)
memory usage: 3.8+ MB
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 27317 entries, 1 to 55160
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   review_date     27317 non-null  object
 1   author_name     27317 non-null  object
 2   age             27317 non-null  int64
 3   vehicle_make    27317 non-null  object
 4   vehicle_title   27317 non-null  object
 5   review_title    27317 non-null  object
 6   review          27317 non-null  object
 7   rating          27317 non-null  float64
 8   recommended_ind 27317 non-null  int64
 9   rating_range    27317 non-null  category
dtypes: category(1), float64(1), int64(2), object(6)
memory usage: 2.1+ MB
```

```
df['review'].isnull().value_counts()
```

```
False    27317
Name: review, dtype: int64
```

```
df['recommended_ind'].isnull().value_counts()
```

```
False    27317
Name: recommended_ind, dtype: int64
```

```
analyze_missing_values(df)
```

	Missing_Count	Missing_Percentage

- Sometimes I cannot detect missing values if they consist of empty (blank) string such as " ". In this situation I can use the following syntax.

```
blanks = []  # start with an empty list

for rv in df.itertuples(): # iterate over the DataFrame
    if type(rv)==str and rv.isspace(): # avoid NaN values and test 'review' for whitespace
        blanks.append(i)
blanks
```

```
[]
```

```
df["review"].str.isspace().sum()
```

```
49
```

```
df[df["review"].str.isspace() == True].index
```

```
Int64Index([  818,  3281,  3285,  4491,  5424,  5522,  5930,  6321,  7889,
            10434, 12608, 12613, 12626, 14045, 14757, 14929, 17859, 20033,
            20497, 20907, 21403, 21409, 22047, 22216, 22220, 22228, 23199,
            29098, 32689, 33321, 33418, 34154, 34963, 35177, 35272, 35294,
            36629, 39513, 39673, 40615, 42797, 43227, 44836, 44895, 46427,
            46428, 49780, 49798, 54757],
           dtype='int64')
```

6) TEXT PREPROSSING

Text, being highly unstructured, contains various forms of noise that hinder immediate analysis. Text preprocessing is the crucial procedure of cleaning and standardizing textual data to eliminate noise, making it suitable for analysis.

The text preprocessing process involves three key steps:

Tokenization: the process of breaking down a piece of text into individual words or tokens.

Noise Removal: Identifying and eliminating irrelevant text elements, including language stopwords, URLs, case variations, punctuation, and industry-specific words.

Lexicon Normalization: Lexicon normalization involves transforming words to a common base or root, making variations of the same word comparable. The two common techniques for lexicon normalization are stemming and lemmatization.

The initial step involves converting text to tokens and lowercasing all words. Subsequently, punctuation, special characters, numbers, and stop words are removed. The second step focuses on normalizing the text through the Lemmatization method.

6.1 Tokenization

```
from nltk.tokenize import word_tokenize


# Tokenize using NLTK
df["tokenized_review"] = df["review"].apply(word_tokenize)

# Print the "review" and "tokenized_review" columns for the first 5 records
print(df[["review", "tokenized_review"]].head())
```

```
review  \
1          Owning the 612 now over 3 years and using it in evey day traffic it just keeps on being great,The paint job is the o
nly problem, technical it's fantastic and by far the best car i ever owned..
5    The reliability of this car is only fair. I initially had severe computer problems that started the car on 6-cylinders and
after two minutes switched to 12 cylinders. The car shook and sputtered ...
6    I have had this top of the line Ferrari and it's the best of the three that I've had. I wanted to get an Enzo, but it is t
oo much money. The 612 Scaglietti is nicer and cheaper. I also have a Ben...
7    I just purchased this Black on Black with yellow stiching 612 and it is EXCELLENT! Peoples' jaws drop and heads are flying
from all directions when I drive by. The exterior is great with side sca...
10   When I first purchased the car I was extremly thrilled. However, after few days I discovered so many faults in the car. Th
e handling and performance of the car is amazing!! However: 1. The car ha...


tokenized_review
1   [Owning, the, 612, now, over, 3, years, and, using, it, in, evey, day, traffic, it, just, keeps, on, being, great, ,, The,
paint, job, is, the, only, problem, ,, technical, it, 's, fantastic, and,...
5   [The, reliability, of, this, car, is, only, fair, ., I, initially, had, severe, computer, problems, that, started, the, ca
r, on, 6-cylinders, and, after, two, minutes, switched, to, 12, cylinders,...
6   [I, have, had, this, top, of, the, line, Ferrari, and, it, 's, the, best, of, the, three, that, I, 've, had, ., I, wanted,
to, get, an, Enzo, ,, but, it, is, too, much, money, ., The, 612, Scaglie...
7   [I, just, purchased, this, Black, on, Black, with, yellow, stiching, 612, and, it, is, EXCELLENT, !, Peoples, ', jaws, dro
p, and, heads, are, flying, from, all, directions, when, I, drive, by, ., ...
10  [When, I, first, purchased, the, car, I, was, extremly, thrilled, ., However, ,, after, few, days, I, discovered, so, many,
faults, in, the, car, ., The, handling, and, performance, of, the, car, ...
```

6.2 Noise Removal

```
# Function to clean tokenized text
def clean_text(tokens):
    stop_words = set(stopwords.words('english'))
    cleaned_tokens = [word.lower() for word in tokens if word.isalpha() and word.lower() not in stop_words and not re.match(r'^ht
    return cleaned_tokens

# Apply the cleaning function to the "tokenized_review" column
df["cleaned_review"] = df["tokenized_review"].apply(clean_text)

# Print the original tokenized text and the cleaned text for the first 5 records
for index, row in df.head().iterrows():
    print(f"Original Tokenized: {row['tokenized_review']}")
    print(f"Cleaned Text: {row['cleaned_review']}\n")
```

```
Original Tokenized: ['Owning', 'the', '612', 'now', 'over', '3', 'years', 'and', 'using', 'it', 'in', 'evey', 'day', 'traffic',
'it', 'just', 'keeps', 'on', 'being', 'great', ',', 'The', 'paint', 'job', 'is', 'the', 'only', 'problem', ',', 'technical', 'i
t', "'s", 'fantastic', 'and', 'by', 'far', 'the', 'best', 'car', 'i', 'ever', 'owned', '..']
Cleaned Text: ['owning', 'years', 'using', 'evey', 'day', 'traffic', 'keeps', 'great', 'paint', 'job', 'problem', 'technical',
'fantastic', 'far', 'best', 'car', 'ever', 'owned']

Original Tokenized: ['The', 'reliability', 'of', 'this', 'car', 'is', 'only', 'fair', '.', 'I', 'initially', 'had', 'severe',
'computer', 'problems', 'that', 'started', 'the', 'car', 'on', '6-cylinders', 'and', 'after', 'two', 'minutes', 'switched', 't
o', '12', 'cylinders', '.', 'The', 'car', 'shook', 'and', 'sputtered', 'for', 'those', 'two', 'minutes', '.', 'This', 'was', 'c
orrected', 'after', 'two', 'visits', 'in', 'which', 'a', 'computer', 're-programming', 'took', 'place', '.', 'Now', 'I', 'hav
e', 'my', 'check', 'engine', 'light', 'constantly', 'flashing', 'even', 'after', 'being', 'deactivated', ',', 'twice', '.', 'I
t', 'again', 'recently', 'activated', ',', 'and', 'I', 'have', 'to', 'bring', 'it', 'in', 'again', 'for', 'the', 'third', 'tim
e', '.']
Cleaned Text: ['reliability', 'car', 'fair', 'initially', 'severe', 'computer', 'problems', 'started', 'car', 'two', 'minutes',
'switched', 'cylinders', 'car', 'shook', 'sputtered', 'two', 'minutes', 'corrected', 'two', 'visits', 'computer', 'took', 'plac
e', 'check', 'engine', 'light', 'constantly', 'flashing', 'even', 'deactivated', 'twice', 'recently', 'activated', 'bring', 'th
ird', 'time']
```

6.3 Stemming

```
# Function to apply stemming to a list of words
def apply_stemming(tokens):
    stemmer = PorterStemmer()
    stemmed_tokens = [stemmer.stem(word) for word in tokens]
    return stemmed_tokens

# Apply stemming to the "cleaned_review" column
df["stemmed_review"] = df["cleaned_review"].apply(apply_stemming)

# Print the original cleaned text and the stemmed text for the first 5 records
for index, row in df.head().iterrows():
    print(f"Original Cleaned Text: {row['cleaned_review']}")
    print(f"Stemmed Text: {row['stemmed_review']}\n")
```

```
Original Cleaned Text: ['car', 'gets', 'great', 'gas', 'mileage', 'best', 'car', 'could', 'buy']
Stemmed Text: ['car', 'get', 'great', 'ga', 'mileag', 'best', 'car', 'could', 'buy']

Original Cleaned Text: ['owning', 'years', 'using', 'evey', 'day', 'traffic', 'keeps', 'great', 'paint', 'job', 'problem', 'tec
hnical', 'fantastic', 'far', 'best', 'car', 'ever', 'owned']
Stemmed Text: ['own', 'year', 'use', 'evey', 'day', 'traffic', 'keep', 'great', 'paint', 'job', 'problem', 'technic', 'fantas
t', 'far', 'best', 'car', 'ever', 'own']

Original Cleaned Text: ['best', 'controllable', 'acceleration', 'ever', 'witnessed', 'vehicle', 'breaking', 'mind', 'blowing',
'handling', 'exceptional', 'except', 'tears', 'tires', 'seat', 'comfort', 'visibility', 'best', 'witnessed', 'ferrari', 'lineu
p', 'lucky', 'enough', 'owned', 'prior']
Stemmed Text: ['best', 'control', 'acceler', 'ever', 'wit', 'vehicl', 'break', 'mind', 'blow', 'handl', 'except', 'except', 'te
ar', 'tire', 'seat', 'comfort', 'visibl', 'best', 'wit', 'ferrari', 'lineup', 'lucki', 'enough', 'own', 'prior']

Original Cleaned Text: ['engine', 'strong', 'pulling', 'right', 'rpm', 'handling', 'excellent', 'yet', 'remains', 'fairly', 'qu
iet', 'interior', 'wise', 'unlike', 'former', 'aston', 'really', 'enjoyable', 'car']
Stemmed Text: ['engin', 'strong', 'pull', 'right', 'rpm', 'handl', 'excel', 'yet', 'remain', 'fairli', 'quiet', 'interior', 'wi
se', 'unlik', 'former', 'aston', 'realli', 'enjoy', 'car']

Original Cleaned Text: ['car', 'scaglietti', 'overlooked', 'compared', 'spider', 'enzo', 'however', 'performs', 'better', 'ferr
ari', 'ever', 'owned', 'driven', 'people', 'go', 'nuts', 'car', 'run', 'comforts', 'cadillac', 'yet', 'performance', 'viper',
'truly', 'extremely', 'satisfied', 'ferrari']
Stemmed Text: ['car', 'scaglietti', 'overlook', 'compar', 'spider', 'enzo', 'howev', 'perform', 'better', 'ferrari', 'ever', 'o
wn', 'driven', 'peopl', 'go', 'nut', 'car', 'run', 'comfort', 'cadillac', 'yet', 'perform', 'viper', 'truli', 'extrem', 'satisf
i', 'ferrari']
```

6.4 Lemmatization

- perform lemmatization on the "cleaned_review" column in DataFrame, I can use spaCy,

```
import spacy
# Download spaCy English model
nlp = spacy.load("en_core_web_sm")
# Function to apply lemmatization to a text
def apply_lemmatization(text):
    doc = nlp(" ".join(text))
    lemmatized_tokens = [token.lemma_ for token in doc]
    return lemmatized_tokens

# Apply lemmatization to the "cleaned_review" column
df["lemmatized_review"] = df["cleaned_review"].apply(apply_lemmatization)

# Print the original cleaned text and the lemmatized text for the first 5 records
for index, row in df.head().iterrows():
    print(f"Original Cleaned Text: {row['cleaned_review']}")
    print(f"Lemmatized Text: {row['lemmatized_review']}\n")
```

```
Original Cleaned Text: ['car', 'gets', 'great', 'gas', 'mileage', 'best', 'car', 'could', 'buy']
Lemmatized Text: ['car', 'get', 'great', 'gas', 'mileage', 'good', 'car', 'could', 'buy']

Original Cleaned Text: ['owning', 'years', 'using', 'evey', 'day', 'traffic', 'keeps', 'great', 'paint', 'job', 'problem', 'tec
hnical', 'fantastic', 'far', 'best', 'car', 'ever', 'owned']
Lemmatized Text: ['own', 'year', 'use', 'evey', 'day', 'traffic', 'keep', 'great', 'paint', 'job', 'problem', 'technical', 'fan
tastic', 'far', 'good', 'car', 'ever', 'own']

Original Cleaned Text: ['best', 'controllable', 'acceleration', 'ever', 'witnessed', 'vehicle', 'breaking', 'mind', 'blowing',
'handling', 'exceptional', 'except', 'tears', 'tires', 'seat', 'comfort', 'visibility', 'best', 'witnessed', 'ferrari', 'lineu
p', 'lucky', 'enough', 'owned', 'prior']
Lemmatized Text: ['well', 'controllable', 'acceleration', 'ever', 'witness', 'vehicle', 'break', 'mind', 'blow', 'handle', 'exc
eptional', 'except', 'tears', 'tire', 'seat', 'comfort', 'visibility', 'well', 'witness', 'ferrari', 'lineup', 'lucky', 'enoug
h', 'own', 'prior']

Original Cleaned Text: ['engine', 'strong', 'pulling', 'right', 'rpm', 'handling', 'excellent', 'yet', 'remains', 'fairly', 'qu
iet', 'interior', 'wise', 'unlike', 'former', 'aston', 'really', 'enjoyable', 'car']
Lemmatized Text: ['engine', 'strong', 'pull', 'right', 'rpm', 'handle', 'excellent', 'yet', 'remain', 'fairly', 'quiet', 'inter
ior', 'wise', 'unlike', 'former', 'aston', 'really', 'enjoyable', 'car']

Original Cleaned Text: ['car', 'scaglietti', 'overlooked', 'compared', 'spider', 'enzo', 'however', 'performs', 'better', 'ferr
ari', 'ever', 'owned', 'driven', 'people', 'go', 'nuts', 'car', 'run', 'comforts', 'cadillac', 'yet', 'performance', 'viper',
'truly', 'extremely', 'satisfied', 'ferrari']
Lemmatized Text: ['car', 'scaglietti', 'overlook', 'compare', 'spider', 'enzo', 'however', 'perform', 'well', 'ferrari', 'eve
r', 'own', 'drive', 'people', 'go', 'nuts', 'car', 'run', 'comfort', 'cadillac', 'yet', 'performance', 'viper', 'truly', 'extre
mely', 'satisfied', 'ferrari']
```

- Lemmatization, is a more sophisticated process that involves reducing words to their base or dictionary form (lemma) While The goal of stemming is to reduce words to their base or root form by removing suffixes or prefixes.
- creates a new column named "review" and copies the content of the "lemmatized_review" column to it. I can then use the "review" column for my text representation task

```
df["review"] = df["lemmatized_review"]
df["review"].head()
```

```
0
[car, get, great, gas, mileage, good, car, could, buy]
1                                                                                [own, year, use, evey, day, traffic, keep,
great, paint, job, problem, technical, fantastic, far, good, car, ever, own]
2    [well, controllable, acceleration, ever, witness, vehicle, break, mind, blow, handle, exceptional, except, tears, tire, se
at, comfort, visibility, well, witness, ferrari, lineup, lucky, enough, ow...
3                                                [engine, strong, pull, right, rpm, handle, excellent, yet, remain,
fairly, quiet, interior, wise, unlike, former, aston, really, enjoyable, car]
4    [car, scaglietti, overlook, compare, spider, enzo, however, perform, well, ferrari, ever, own, drive, people, go, nuts, ca
r, run, comfort, cadillac, yet, performance, viper, truly, extremely, sati...
Name: review, dtype: object
```

6.5 Handling Rare Words

```
# Create a new column named "review" and join the lemmatized tokens into a single string
df["review"] = df["lemmatized_review"].apply(lambda x: ' '.join(x))

# Let's print the first 5 records of the "review" column
print(df["review"].head())
```

```
0
car get great gas mileage good car could buy
1                                                                    own year use evey day traffic keep grea
t paint job problem technical fantastic far good car ever own
2        well controllable acceleration ever witness vehicle break mind blow handle exceptional except tears tire seat comfort
visibility well witness ferrari lineup lucky enough own prior
3                                                engine strong pull right rpm handle excellent yet remain fairly
quiet interior wise unlike former aston really enjoyable car
4    car scaglietti overlook compare spider enzo however perform well ferrari ever own drive people go nuts car run comfort cad
illac yet performance viper truly extremely satisfied ferrari
Name: review, dtype: object
```

- Calculate and display the word frequencies.

```
word_values = pd.Series(" ".join(df["review"]).split()).value_counts()
word_values
```

```
car               69619
drive             30730
get               18819
great             16511
well              14825
                  ...
variate               1
windsheld             1
individualistic       1
nelli                 1
moneypit              1
Length: 27101, dtype: int64
```

- filters out words with frequencies less than or equal to 2 the calculate the total number using rare_words.value_counts**()**

```
rare_words = word_values[word_values <= 2]
rare_words
```

```
stx               2
tsunami           2
flaccid           2
snout             2
powerless         2
                 ..
variate           1
windsheld         1
individualistic   1
nelli             1
moneypit          1
Length: 15866, dtype: int64
```

```
rare_words.value_counts()
```

```
1    12764
2     3102
dtype: int64
```

```
len(rare_words)
```

```
15866
```

```
rare_words.index
```

```
Index(['stx', 'tsunami', 'flaccid', 'snout', 'powerless', 'engeneere', 'hre',
       'zillion', 'volumes', 'lugbolt',
       ...
       'montrose', 'butte', 'jagx', 'reimbursable', 'autonomy', 'variate',
       'windsheld', 'individualistic', 'nelli', 'moneypit'],
      dtype='object', length=15866)
```

In

- remove words with frequencies less than or equal to 2 from the "review" column in DataFrame

```
df["review"] = df["review"].apply(lambda x: " ".join([i for i in x.split() if i not in rare_words.index]))
df["review"].head()
```

```
0
car get great gas mileage good car could buy
1                                                                                  own year use evey day traffic keep grea
t paint job problem technical fantastic far good car ever own
2        well controllable acceleration ever witness vehicle break mind blow handle exceptional except tears tire seat comfort
visibility well witness ferrari lineup lucky enough own prior
3                                                          engine strong pull right rpm handle excellent yet remain fairly
quiet interior wise unlike former aston really enjoyable car
4    car scaglietti overlook compare spider enzo however perform well ferrari ever own drive people go nuts car run comfort cad
illac yet performance viper truly extremely satisfied ferrari
Name: review, dtype: object
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 42381 entries, 0 to 55163
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   review             42381 non-null  object
 1   recommended_ind    42381 non-null  int64 
 2   tokenized_review   42381 non-null  object
 3   cleaned_review     42381 non-null  object
 4   stemmed_review     42381 non-null  object
 5   lemmatized_review  42381 non-null  object
 6   cleaned            42381 non-null  object
dtypes: int64(1), object(6)
memory usage: 2.6+ MB
```

```
df.head(3)
```

	review	recommended_ind	tokenized_review	cleaned_review	stemmed_review	lemmatized_review	cleaned
0	car get great gas mileage good car could buy	1	[This, car, gets, great, gas, mileage, and, is, the, best, car, you, could, buy, .]	[car, gets, great, gas, mileage, best, car, could, buy]	[car, get, great, ga, mileag, best, car, could, buy]	[car, get, great, gas, mileage, good, car, could, buy]	[car, get, great, gas, mileage, good, car, could, buy]

- After Text Preprocessing I can notice the   following
- review: Original text or the input text column
- recommended_ind: Possibly a column indicating whether a recommendation is present or not (binary recommendation indicator). tokenized_review: Column containing tokenized versions of the text.
- cleaned_review: Column containing text after cleaning (removing stopwords, converting to lowercase, etc.). stemmed_review: Column containing text after stemming. lemmatized_review: Column containing text after lemmatization.
- cleaned: The column I mentioned earlier, which I indicated want to use for text

7) TEXT REPRESENTATION

Creating word clouds for positive and negative reviews can provide insightful visualizations. this task can be done as follows:

Detect Reviews: Filter Data: Separate the dataset into positive and negative reviews based on the recommended status.

Collect Words: 1-Text Preprocessing: Apply text preprocessing techniques (tokenization, lowercasing, removal of stopwords, etc.) to clean the review text. 2-Word Frequency: Count the frequency of each word in positive and negative reviews separately.

Create Word Cloud: 1-Generate Word Clouds: Utilize a word cloud library (e.g., wordcloud in Python) to generate word clouds for positive and negative reviews. 2-Visualization: Visualize the word clouds, where words are displayed with sizes proportional to their frequencies.

7.1 The Detection of Positive and Negative Reviews

```
df.columns
```

```
Index(['review', 'recommended_ind', 'tokenized_review', 'cleaned_review',
       'stemmed_review', 'lemmatized_review', 'cleaned'],
      dtype='object')
```

```
df.drop(['tokenized_review', 'cleaned_review',
       'stemmed_review', 'lemmatized_review', 'cleaned'], axis=1, inplace=True)
```

- creates a new DataFrame (filtered_df) containing only the rows where the value in the "recommended_ind" column is equal to 1

```
df[df["recommended_ind"] == 1]
```

	review	recommended_ind
0	car get great gas mileage good car could buy	1
1	own year use evey day traffic keep great paint job problem technical fantastic far good car ever own	1
2	well controllable acceleration ever witness vehicle break mind blow handle exceptional except tears tire seat comfort visibility well witness ferrari lineup lucky enough own prior	1
3	engine strong pull right rpm handle excellent yet remain fairly quiet interior wise unlike former aston really enjoyable car	1
4	car scaglietti overlook compare spider enzo however perform well ferrari ever own drive people go nuts car run comfort cadillac yet performance viper truly extremely satisfied ferrari	1
...	...	...
55159	pleasure tell guy get one never ever day without fun road drive really nice sedan sport car time	1
55160	fifth bmw third car first I ve series also expect comfort car harsh harsh right everyday car see nyc street imagine drive nyc live los angeles street little well nyc performance excellent however ...	1
55161	far good car week family fit comfortable entertainment system rear ride smooth power command fun drive car go daily driver fun enthusiastic car push button love car design	1
55162		1
55163	great large car	1

30963 rows × 2 columns

- creates a new DataFrame (filtered_df) containing only the rows where the value in the"recommended_ind" column is equal to 0

```
df[df["recommended_ind"] == 0]
```

	review	recommended_ind
10	first purchase car extremly thrill however day discover many fault car handle performance car amazing however car strong smell fuel inside every hole road sound like car go break handbrake stop ca...	0
18	first buy ferrari scaglietti anticipate great experience life great car first couple day lose excitement fast lack sporty design tend turn many head power take line fast car find	0
28	ferrari fun drive	0
37	think car well make	0
46	one good car	0
...	...	...
55092	look believe	0
55113	magazine right new	0
55119	excellent engineering masterpiece	0
55125	pricey repair give back bang buck gas mileage well way grab stout car	0
55127	buy bmw may mile drive home oil cooler break start leak oil repair addition money spend purchase car approximately taxis week later passenger restraint malfunction show battery cable replace spend...	0

11418 rows × 2 columns

7.2 The Collection of Positive and Negative Words

- Collect Words (positive and negative separately)

```
" ".join(df["review"]).split()
```

```
['car',
 'get',
 'great',
 'gas',
 'mileage',
 'good',
 'car',
 'could',
 'buy',
 'own',
 'year',
 'use',
 'evey',
 'day',
 'traffic',
 'keep',
 'great',
 'paint',
 'job',
```

```
neg_words = " ".join(df[df["recommended_ind"] == 0].review).split()
neg_words
```

```
['first',
 'purchase',
 'car',
 'extremly',
 'thrill',
 'however',
 'day',
 'discover',
 'many',
 'fault',
 'car',
 'handle',
 'performance',
 'car',
 'amazing',
 'however',
 'car',
 'strong',
 'smell',
```

```
pos_words =" ".join(df[df["recommended_ind"] == 1].review).split()
pos_words
```

```
['car',
 'get',
 'great',
 'gas',
 'mileage',
```

7.3 Creating of Word Cloud

- Let's create Word Cloud for most common words in recommended not recommended reviews separately.

```
review = df["review"]
```

```
all_words = " ".join(review)
```

```
all_words[:100]
```

```
'car get great gas mileage good car could buy own year use evey day traffic keep great paint job prob'
```

```
from wordcloud import WordCloud

wordcloud = WordCloud(background_color="white", max_words =250).generate(all_words)

plt.figure(figsize = (13, 13))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```


```
wordcloud = WordCloud(background_color="white", max_words =250, colormap='gist_heat').generate(str(neg_words))

plt.figure(figsize = (13, 13))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```


```
wordcloud = WordCloud(background_color="white", max_words =250, colormap='cool').generate(str(pos_words))

plt.figure(figsize = (13, 13))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

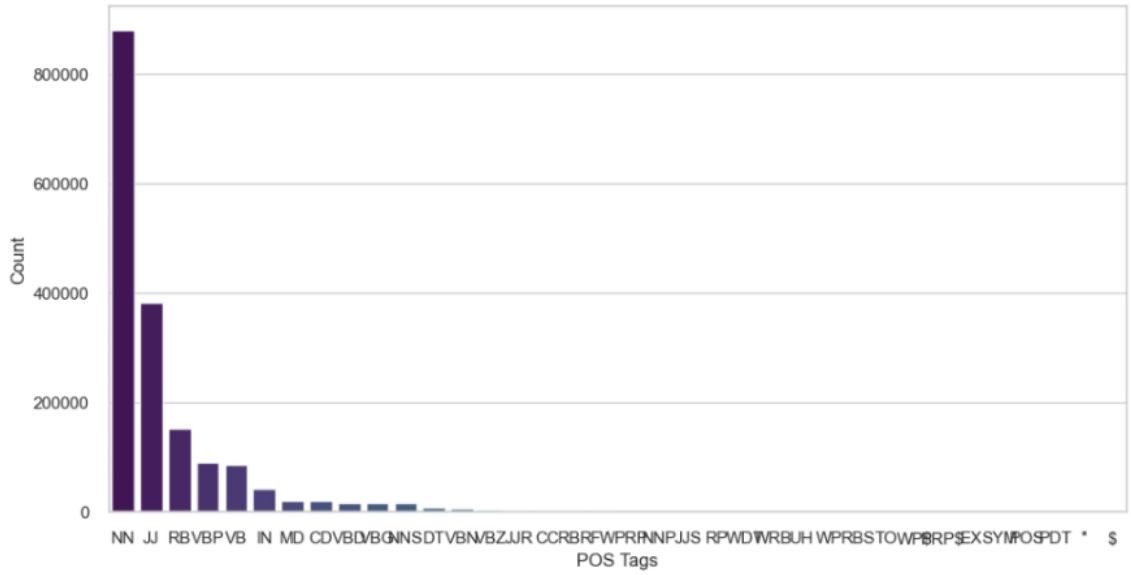
7.4 Part Of Speech Tagging

- implement part-of-speech tagging in the review column, useing the Natural Language Toolkit (nltk) library in Python.

```
import nltk
from nltk import word_tokenize, pos_tag

# Download the required resource for nltk
nltk.download('averaged_perceptron_tagger')

# Function to perform part-of-speech tagging on a text
def pos_tagging(text):
    tokens = word_tokenize(text)
    pos_tags = pos_tag(tokens)
    return pos_tags

# Apply the function to the 'review' column
df['pos_tags'] = df['review'].apply(pos_tagging)

# Display the DataFrame with the new 'pos_tags' column
print(df[['review', 'pos_tags']].head())
```

```
review  \
0
car get great gas mileage good car could buy
1                                                                                own year use evey day traffic keep great
paint job problem technical fantastic far good car ever own
2      well controllable acceleration ever witness vehicle break mind blow handle exceptional except tears tire seat comfort vi
sibility well witness ferrari lineup lucky enough own prior
3                                                  engine strong pull right rpm handle excellent yet remain fairly q
uiet interior wise unlike former aston really enjoyable car
4  car scaglietti overlook compare spider enzo however perform well ferrari ever own drive people go nuts car run comfort cadil
lac yet performance viper truly extremely satisfied ferrari


pos_tags
0                                                                                [(car, NN), (get, NN), (great, J
J), (gas, NN), (mileage, NN), (good, JJ), (car, NN), (could, MD), (buy, VB)]
1  [(own, JJ), (year, NN), (use, NN), (evey, JJ), (day, NN), (traffic, NN), (keep, VB), (great, JJ), (paint, JJ), (job, NN), (p
roblem, NN), (technical, JJ), (fantastic, JJ), (far, RB), (good, JJ), (c...
2  [(well, RB), (controllable, JJ), (acceleration, NN), (ever, RB), (witness, JJ), (vehicle, NN), (break, NN), (mind, NN), (blo
w, NN), (handle, VBD), (exceptional, JJ), (except, IN), (tears, NNS), (t...
3  [(engine, NN), (strong, JJ), (pull, NN), (right, NN), (rpm, NN), (handle, VB), (excellent, JJ), (yet, RB), (remain, VBP), (f
airly, RB), (quiet, JJ), (interior, JJ), (wise, NN), (unlike, IN), (form...
4  [(car, NN), (scaglietti, NNS), (overlook, VBP), (compare, JJ), (spider, NN), (enzo, NN), (however, RB), (perform, RB), (wel
l, RB), (ferrari, VB), (ever, RB), (own, JJ), (drive, NN), (people, NNS),...
```

```
# Flatten the list of POS tags
all_pos_tags = [tag for tags in df['pos_tags'] for _, tag in tags]

# Create a bar plot of POS tag frequencies
sns.set(style="whitegrid")
plt.figure(figsize=(12, 6))
sns.countplot(x=all_pos_tags, order=pd.Series(all_pos_tags).value_counts().index, palette="viridis")
plt.title('Distribution of POS Tags in Reviews')
plt.xlabel('POS Tags')
plt.ylabel('Count')
plt.show()
```



- the bar plot showing the distribution of different POS tags,This suggests that nouns are commonly used in the reviews, followed by adjectives and adverbs. The distribution of POS tags provides insights into the linguistic characteristics of the reviews, indicating the types of words that are more prevalent in the dataset.

7.5 Named Entity Recognition

- implement NER in the review column, useing the spaCy library in Python.

```
import spacy

# Load the spaCy English model
nlp = spacy.load("en_core_web_sm")

# Process each review in the DataFrame
for review in df['review']:
    # Process the text with spaCy
    doc = nlp(review)

    # Print entities and their labels
    for ent in doc.ents:
        print(f"Entity: {ent.text}, Label: {ent.label_}")
```

```
Entity: ferrari lineup, Label: PERSON
Entity: ferrari, Label: NORP
Entity: cadillac, Label: ORG
Entity: ferrari, Label: PERSON
Entity: two minute, Label: TIME
Entity: sputter two minute, Label: TIME
Entity: two, Label: CARDINAL
Entity: third, Label: ORDINAL
Entity: three, Label: CARDINAL
Entity: two, Label: CARDINAL
Entity: first, Label: ORDINAL
Entity: ferrari, Label: NORP
Entity: ferrari, Label: NORP
Entity: eleven year, Label: DATE
Entity: first, Label: ORDINAL
Entity: thrill, Label: GPE
Entity: day, Label: DATE
Entity: ferrari, Label: NORP
Entity: daily, Label: DATE
```

7.6 Word Embeddings

- perform word embeddings for the review column, by using pre-trained word embeddings models such as Word2Vec ,the example below shows the work embedding for word Nissan

```
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize


# Tokenize the reviews
tokenized_reviews = df['review'].apply(word_tokenize)

# Train Word2Vec model
model = Word2Vec(sentences=tokenized_reviews, vector_size=100, window=5, min_count=1, workers=4)

# Save the model
model.save("word2vec_model")

# Load the model
loaded_model = Word2Vec.load("word2vec_model")

# Example: Get the word embeddings for a specific word
word_embedding = loaded_model.wv['nissan']
print("Word Embedding for 'nissan':", word_embedding)
```

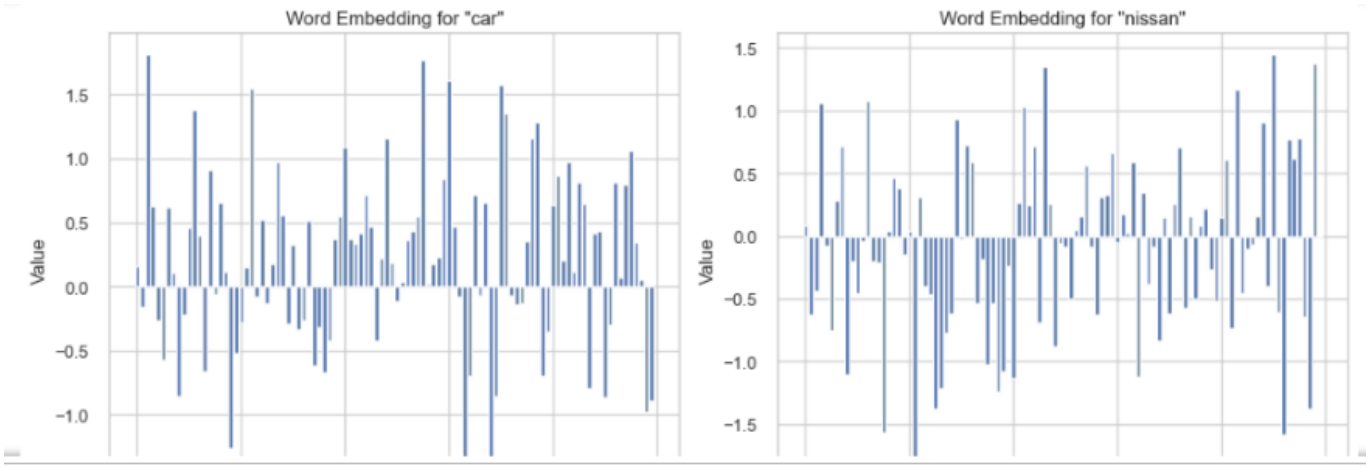
```
print("Word Embedding for 'nissan':", word_embedding)
```

```
Word Embedding for 'nissan': [ 0.08639639 -0.6270252  -0.4375927   1.0637482  -0.07530611 -0.7516972
  0.28647026  0.71707016 -1.107011   -0.20401742 -0.45911    -0.04180208
  1.0798647  -0.20406154 -0.20868543 -1.5625921   0.0451594   0.4658138
  0.38151258 -0.1453106   0.0446414  -1.8285491   0.31170243 -0.40148845
 -0.46178472 -1.3741212  -1.2148334  -0.7745817  -0.6182383   0.93100524
 -0.01803452  0.7306245   0.58939916 -0.538749   -0.187668   -1.0230409
 -0.5338696  -1.239502   -1.0816375  -0.23532937 -1.1358083   0.2636207
  1.0357869   0.24531142  0.7167884  -0.6920887   1.3508252   0.26120928
 -0.8834119  -0.06010116 -0.08294437 -0.49851257  0.05353352  0.16182746
  0.56183445 -0.08693614 -0.6295075   0.30689934  0.3296621   0.6665591
 -0.04770869  0.17483391  0.02627775  0.5955533  -1.1248307   0.3467304
 -0.3805028  -0.08813678 -0.8340094   0.14928633 -0.61650467  0.25464237
  0.70997536 -0.57636267  0.15791214 -0.50170153  0.0851689   0.22365522
 -0.2620873  -0.51844245  0.15328152  0.61188585 -0.73187304  1.1721041
 -0.45617744 -0.10049542 -0.07185008  0.15905617  0.9089551  -0.39864317
  1.4524674  -0.6064189  -1.5802296   0.76959777  0.6145474   0.7808994
 -0.6446828  -1.3789202   1.3720043   0.00455372]
```

```
# Example: Get the word embeddings for a specific word
word_embedding = loaded_model.wv['car']
print("Word Embedding for 'car':", word_embedding)
```

```
Word Embedding for 'nissan': [ 0.16415197 -0.1614567   1.8159173   0.6245974  -0.26308402 -0.57111996
  0.6224355   0.10610139 -0.8568868  -0.22137032  0.45957485  1.3828104
  0.3987513  -0.66193926  0.9095229  -0.05618567  0.651007    0.11731259
 -1.2579136  -0.5235533  -0.28049204  0.15204787  1.547604   -0.07779082
  0.5239806  -0.13278575  0.17726809  0.97323143  0.5584831  -0.29448497
  0.32764858 -0.3366444  -0.26342693  0.51782066 -0.6186903  -0.32007292
 -0.6692068  -0.4231954   0.3764604   0.5457451   1.0896342   0.37066662
  0.33479226  0.4129302   0.7136679   0.4702131  -0.42541265  0.22375585
  1.1580565   0.18793356 -0.1098281   0.03310351  0.3639466   0.43029433
  0.54758435  1.7668366   0.0181258   0.17976244  0.2324422   0.8419419
  1.6057953   0.46759397 -0.07475555 -1.395743   -0.6987211   0.71787107
 -0.06728774  0.6578383  -1.3351368  -0.85515636  1.5700903   1.3502468
 -0.07120968 -0.13719186 -0.13406041  0.35414478  1.1558019   1.2775717
 -0.6944631  -0.3526105   0.63604695  0.86824095  0.20703118  0.9704457
  0.11714419  0.8133405   0.64445573 -0.790503    0.41301706  0.43428925
 -0.86293095 -0.29630825  0.8177734   0.07409792  0.7954441   1.0576521
  0.347752    0.05609295 -0.9813725  -0.8945012 ]
```

```
import matplotlib.pyplot as plt

# Words to visualize
words_to_visualize = ['car', 'nissan', 'engine', 'fuel']

# Create a subplot with 2 rows and 2 columns
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))

# Flatten the 2D array of axes for easier indexing
axes = axes.flatten()

# Plot the word embeddings for each word
for i, word in enumerate(words_to_visualize):
    word_embedding = loaded_model.wv[word]
    axes[i].bar(range(len(word_embedding)), word_embedding)
    axes[i].set_title(f'Word Embedding for "{word}"')
    axes[i].set_xlabel('Dimension')
    axes[i].set_ylabel('Value')

# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```


Word Embedding for "car"
Value
1.5
1.0
0.5
0.0
−0.5
−1.0
Word Embedding for "nissan"
Value
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5

7.7 Bag of Words

```
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd


corpus = df['review'].tolist()

# Create an instance of CountVectorizer with a limited vocabulary size
vectorizer = CountVectorizer(max_features=5000)

# Fit and transform the corpus to get the bag-of-words representation as a sparse matrix
X = vectorizer.fit_transform(corpus)

# Convert the sparse matrix to a DataFrame for better visualization
bow_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())

# Display the bag-of-words DataFrame
print(bow_df)
```

```
       aaa  ab  abc  ability  able  abound  absolute  absolutely  absolutly  \
0        0   0    0        0     0       0         0           0          0
1        0   0    0        0     0       0         0           0          0
2        0   0    0        0     0       0         0           0          0
3        0   0    0        0     0       0         0           0          0
4        0   0    0        0     0       0         0           0          0
...    ...  ..  ...      ...   ...     ...       ...         ...        ...
42376    0   0    0        0     0       0         0           0          0
42377    0   0    0        0     0       0         0           0          0
42378    0   0    0        0     0       0         0           0          0
42379    0   0    0        0     0       0         0           0          0
42380    0   0    0        0     0       0         0           0          0

       abundant  abuse  abused  abysmal  ac  accelerate  accelerated  \
0             0      0       0        0   0           0            0
1             0      0       0        0   0           0            0
2             0      0       0        0   0           0            0
3             0      0       0        0   0           0            0
4             0      0       0        0   0           0            0
```

8) SENTIMENT CLASSIFICATION WITH MACHINE LEARNING & DEEP LEARNING

Before delving into the modeling phase, it's essential to undertake data preprocessing steps. One crucial aspect involves vectorization and the initial implementation of it will occur at this stage.

Machine learning algorithms commonly expect numeric feature vectors as input. Consequently, when dealing with text documents, a mechanism is required to convert each document into a numeric vector. This process is termed text vectorization. For my specific case, I will employ a widely used vectorization approach: representing each text as a vector of word counts.

At this juncture, my review text column is in token form, devoid of punctuation and stopwords. To convert the text collection into a matrix of token counts, I can leverage Scikit-learn's CountVectorizer. This resulting matrix can be envisioned as a 2-D matrix where each row corresponds to a unique word, and each column corresponds to a review.

For the deep learning model, I will incorporate an embedding layer for all words.

Following the completion of data preprocessing, I will proceed to construct models utilizing the following classification algorithms:

- **Naive Bayes :**

it is computationally efficient and simple to implement. Particularly effective for text classification tasks, such as spam filtering or sentiment analysis.also it performs well in situations where the number of features is high compared to the number of samples.

- **Support Vector Machine**

SVM works well in high-dimensional feature spaces, making it suitable for a variety of applications.SVM often performs well in practice, especially when the data is not too large.

- **Deep Learning Model**

Deep learning models automatically learn hierarchical representations from the data, potentially capturing complex patterns. can benefit from large amounts of labeled data

8.1 Train | Test & Split

- To run machine learning algorithms, I need to convert text files into numerical feature vectors. I will use bag of words model for my analysis.
- Let's first I split the data into train and test sets:

```
df.head()
```

	review	recommended_ind	pos_tags
0	car get great gas mileage good car could buy	1	[(car, NN), (get, NN), (great, JJ), (gas, NN), (mileage, NN), (good, JJ), (car, NN), (could, MD), (buy, VB)]
1	own year use evey day traffic keep great paint job problem technical fantastic far good car ever own	1	[(own, JJ), (year, NN), (use, NN), (evey, JJ), (day, NN), (traffic, NN), (keep, VB), (great, JJ), (paint, JJ), (job, NN), (problem, NN), (technical, JJ), (fantastic, JJ), (far, RB), (good, JJ), (c...
2	well controllable acceleration ever witness vehicle break mind blow handle exceptional except tears tire seat comfort visibility well witness ferrari lineup lucky enough own prior	1	[(well, RB), (controllable, JJ), (acceleration, NN), (ever, RB), (witness, JJ), (vehicle, NN), (break, NN), (mind, NN), (blow, NN), (handle, VBD), (exceptional, JJ), (except, IN), (tears, NNS), (t...
3	engine strong pull right rpm handle excellent yet remain fairly quiet interior wise unlike former aston really enjoyable car	1	[(engine, NN), (strong, JJ), (pull, NN), (right, NN), (rpm, NN), (handle, VB), (excellent, JJ), (yet, RB), (remain, VBP), (fairly, RB), (quiet, JJ), (interior, JJ), (wise, NN), (unlike, IN), (form...
4	car scaglietti overlook compare spider enzo however perform well ferrari ever own drive people go nuts car run comfort cadillac yet performance viper truly extremely satisfied ferrari	1	[(car, NN), (scaglietti, NNS), (overlook, VBP), (compare, JJ), (spider, NN), (enzo, NN), (however, RB), (perform, RB), (well, RB), (ferrari, VB), (ever, RB), (own, JJ), (drive, NN), (people, NNS),...

```
from sklearn.model_selection import train_test_split
```

```
X = df["review"]
y= df["recommended_ind"]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=101)
```

- In the next step, i will create a numerical feature vector for each document:

8.2 Vectorization

Vectorization is a process of converting textual data into a numerical format that can be used as input for machine learning models Count Vectorization and TF-IDF (Term Frequency-Inverse Document Frequency) Vectorization are two popular techniques for converting text data into numerical representations suitable for machine learning models.

i will use Count Vectorization for the following reasons,

1- Simple and computationally efficient.

2-Suitable for tasks where the emphasis is on the frequency of terms within documents.

3- Good for tasks like spam detection, sentiment analysis, and topic modeling.

then i will implement TF-IDF Vectorization because:

1-Effective in capturing the importance of terms across the entire corpus.

2-Suitable for tasks where distinguishing terms based on their uniqueness is important.

3 Commonly used in information retrieval text classification and document clustering tasks

8.2.a Count Vectorization

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
vectorizer = CountVectorizer()

X_train_count = vectorizer.fit_transform(X_train)
X_test_count = vectorizer.transform(X_test)
```

```
type(X_train_count)
```

```
scipy.sparse._csr.csr_matrix
```

- **X**_train_count_array will be a dense NumPy array representing the document-term matrix where each row corresponds to a document, and each column corresponds to a unique word, with the values representing the count of each word in each document.

```
X_train_count.toarray()
```

```
array([[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]], dtype=int64)
```

```
vectorizer.get_feature_names_out()
```

```
array(['aa', 'aaa', 'aah', ..., 'zooming', 'zsp', 'zx'], dtype=object)
```

- creating a Pandas DataFrame from the dense NumPy array obtained from the toarray(),The resulting df_train_count DataFrame will have rows corresponding to different text samples and columns corresponding to the unique words in the training data, with the values representing the word counts for each sample.

```
pd.DataFrame(X_train_count.toarray(), columns = vectorizer.get_feature_names_out())
```

	aa	aaa	aah	ab	abandon	abc	ability	abit	able	abnormal	abomination	abound	abroad	abrupt	abruptly	abs	absence	absent	absolute	absolut
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	2	0	1	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
33899	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
33900	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
33901	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
33902	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
33903	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

33904 rows × 11180 columns

8.2.b TF-IDF Vectorization

Represents each document as a vector where each component corresponds to the TF-IDF weight of a term. The resulting matrix has rows corresponding to documents and columns corresponding to unique terms in the entire corpus.

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
tf_idf_vectorizer = TfidfVectorizer()

X_train_tf_idf = tf_idf_vectorizer.fit_transform(X_train)
X_test_tf_idf = tf_idf_vectorizer.transform(X_test)
```

```
X_train_tf_idf.toarray()
```

```
array([[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.]])
```

```
pd.DataFrame(X_train_tf_idf.toarray(), columns = tf_idf_vectorizer.get_feature_names_out())
```

	aa	aaa	aah	ab	abandon	abc	ability	abit	able	abnormal	abomination	abound	abroad	abrupt	abruptly	abs	absence	absent	absolute	al
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
2	0.00	0.00	0.00	0.00	0.00	0.00	0.15	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
33899	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
33900	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
33901	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
33902	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
33903	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	

33904 rows × 11180 columns

[197]:

- Before diving into modelling, I will create a User-Defined-Function for comparing models at the end.

```
from sklearn.metrics import confusion_matrix,classification_report, f1_score, recall_score, accuracy_score, precision_score
```

```
!pip install scikit-plot
from scikitplot.metrics import plot_confusion_matrix

def eval(model, X_train, X_test):
    y_pred = model.predict(X_test)
    y_pred_train = model.predict(X_train)

    print("Test_Set")
    print(classification_report(y_test, y_pred))
    print("Train_Set")
    print(classification_report(y_train, y_pred_train))

    fig, ax = plt.subplots(figsize=(8, 8))

    plot_confusion_matrix(y_test, y_pred, ax=ax)
    plt.show()
```

- Now it's time to train all models using TFIDF and Count vectorizer data.

9) MACHINE LEARNING MODELLING

9.1 Naive Bayes

I want to make comparison between Naive Bayes with Count Vectorizer and TF-IDF (Term Frequency-Inverse Document Frequency) by evaluating their performance in the context of current dataset.

9.1.a Naive Bayes with Count Vectorizer

```
from sklearn.naive_bayes import MultinomialNB, BernoulliNB # BernoulliNB for binary model
```

```
nb = MultinomialNB()
nb.fit(X_train_count, y_train)
```

```
MultinomialNB()
```

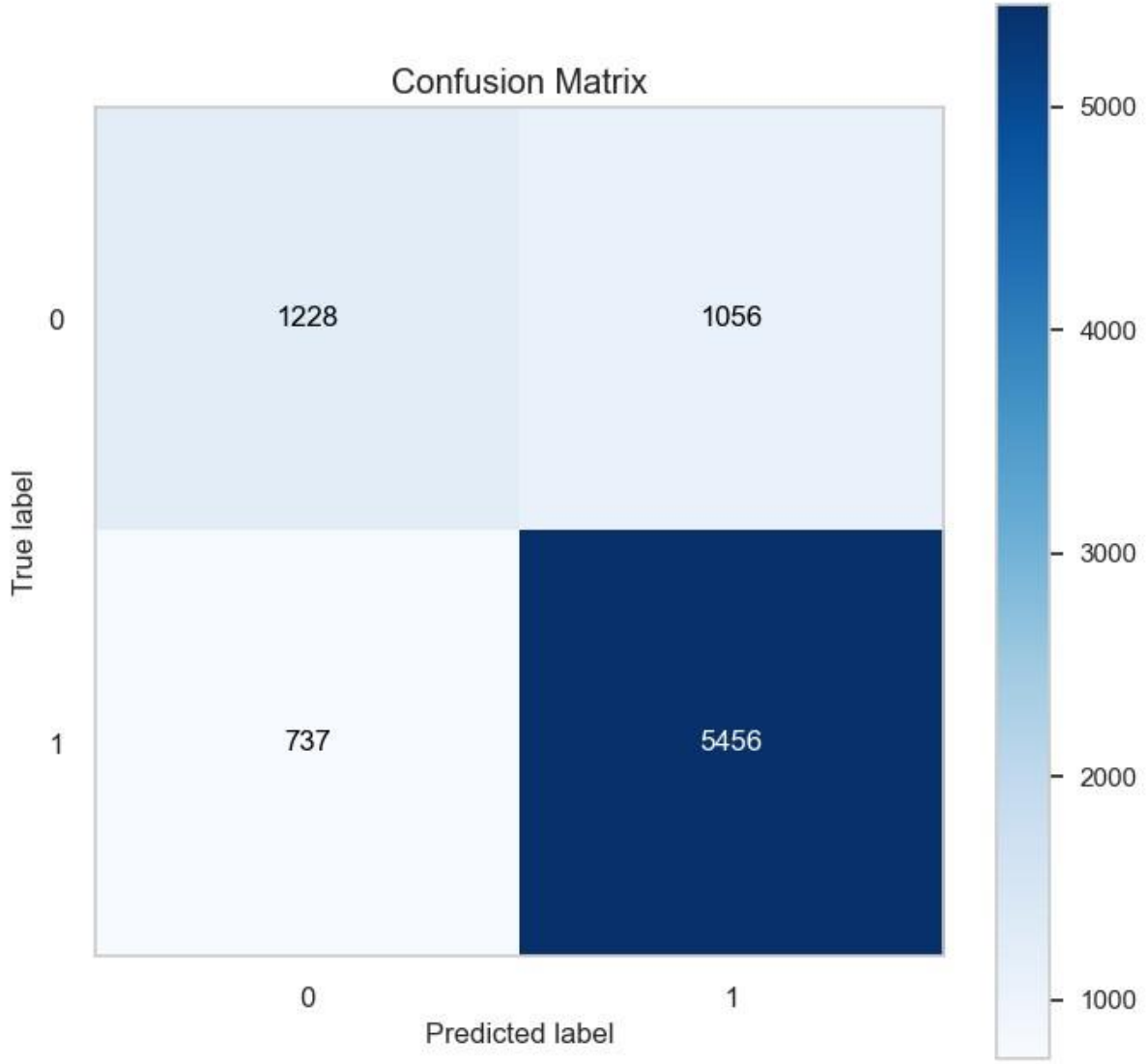
```
print("NB MODEL")

eval(nb, X_train_count, X_test_count)
```

```
NB MODEL
Test_Set
              precision    recall  f1-score   support

           0       0.62      0.54      0.58      2284
           1       0.84      0.88      0.86      6193

    accuracy                           0.79      8477
   macro avg       0.73      0.71      0.72      8477
weighted avg       0.78      0.79      0.78      8477

Train_Set
              precision    recall  f1-score   support

           0       0.67      0.59      0.63      9134
           1       0.86      0.89      0.87     24770

    accuracy                           0.81     33904
   macro avg       0.77      0.74      0.75     33904
weighted avg       0.81      0.81      0.81     33904
```

Interpretation:

- The model performs well in identifying instances labeled as "Recommended," showing high precision and recall.
- There's room for improvement in correctly identifying instances labeled as "Not Recommended" (class 0).
- The overall accuracy is decent, but considering both precision and recall is essential, especially in cases of class imbalance.

while the model demonstrates good performance, further optimization, especially for class 0, could enhance its predictive capabilities.



- **how to read a confusion matrix**

After completing the model training, I proceeded to generate predictions for a set of 8477 reviews within the validation dataset. For these predictions, I have access to the true labels, enabling us to assess the model's predictive performance.

• True Positive (TP):

This corresponds to the bottom-right (dark blue) corner of the evaluation matrix. It represents the instances where the model correctly identified positive cases. In the context of review prediction, this translates to the accurate prediction of recommendations (1). In my specific example, the count of true positives stands at 5456.

• True Negative (TN):

Found in the top-left (sky blue) corner of the matrix, this indicates the number of correctly identified negative cases. These are instances where the actual label is negative, and the model accurately predicted it as negative. In

review prediction, this aligns with correctly anticipated non-recommendations (0). For the given example, the true negatives total 1228.

• False Positive (FP):

Positioned in the top-right (light blue) corner of the matrix, this represents the count of incorrectly predicted positive cases. In simpler terms, these are instances where the actual label is negative, but the model erroneously predicted it as positive. Often referred to as false alarms or Type 1 errors, in review prediction, this denotes the number of predicted recommendations incorrectly labeled as (1). In my example, false positives amount to 1056.

• False Negative (FN):

Situated in the bottom-left (white) corner of the matrix, this signifies the instances of incorrectly predicted negative cases. Essentially, these are situations where the actual label is positive, but the model predicted it as negative. Described as missed cases or Type 2 errors, in review prediction, this corresponds to the number of recommendations that were overlooked.

```
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score

custom_scorer = {'accuracy': make_scorer(accuracy_score),
                 'precision-0': make_scorer(precision_score, pos_label=0),
                 'recall-0': make_scorer(recall_score, pos_label=0),
                 'f1-0': make_scorer(f1_score, pos_label=0),
                 'precision-1': make_scorer(precision_score, pos_label=1),
                 'recall-1': make_scorer(recall_score, pos_label=1),
                 'f1-1': make_scorer(f1_score, pos_label=1)
                }

for i, j in custom_scorer.items():
    model = MultinomialNB()
    scores = cross_val_score(model, X_train_count, y_train, cv = 10, scoring = j).mean()
    if i == "recall-1":
        nb_count_rec = scores
    elif i == "f1-1":
        nb_count_f1 = scores
    print(f" {i:20} score for count : {scores}\n")
```

```
 accuracy             score for count : 0.801469280561333

 precision-0          score for count : 0.6496067297588252

 recall-0             score for count : 0.5710550976533946

 f1-0                 score for count : 0.6077326571618079

 precision-1          score for count : 0.8486059536146892

 recall-1             score for count : 0.8864352038756561

 f1-1                 score for count : 0.8670985272554189
```

- The information obove includes various evaluation metrics for a Naive Bayes model used to predict customer recommendations. Let's break down each metric and discuss how to analyze the information:

1. Accuracy: Accuracy Score: 0.8015

This metric represents the overall correctness of the predictions, indicating the ratio of correctly predicted instances to the total instances. In my case, the model achieves an accuracy of approximately 80.15%.

2. Precision, Recall, and F1-Score for Class 0 (Non-recommendation):

Precision-0: 0.6496

Recall-0: 0.5711

F1-0: 0.6077

These metrics focus on the performance of the model concerning instances labeled as non-recommendation (Class 0).

Precision-0: The proportion of correctly predicted non-recommendations among all instances predicted as non-recommendation.

Recall-0: The proportion of correctly predicted non-recommendations among all actual non-recommendations.

F1-0: The harmonic mean of precision-0 and recall-0, providing a balanced measure.

3. Precision, Recall, and F1-Score for Class 1 (Recommendation):

Precision-1: 0.8486

Recall-1: 0.8864

F1-1: 0.8671

These metrics focus on the performance of the model concerning instances labeled as recommendation (Class 1).

Precision-1: The proportion of correctly predicted recommendations among all instances predicted as recommendation.

Recall-1: The proportion of correctly predicted recommendations among all actual recommendations.

F1-1: The harmonic mean of precision-1 and recall-1, providing a balanced measure.

4. Analysis:

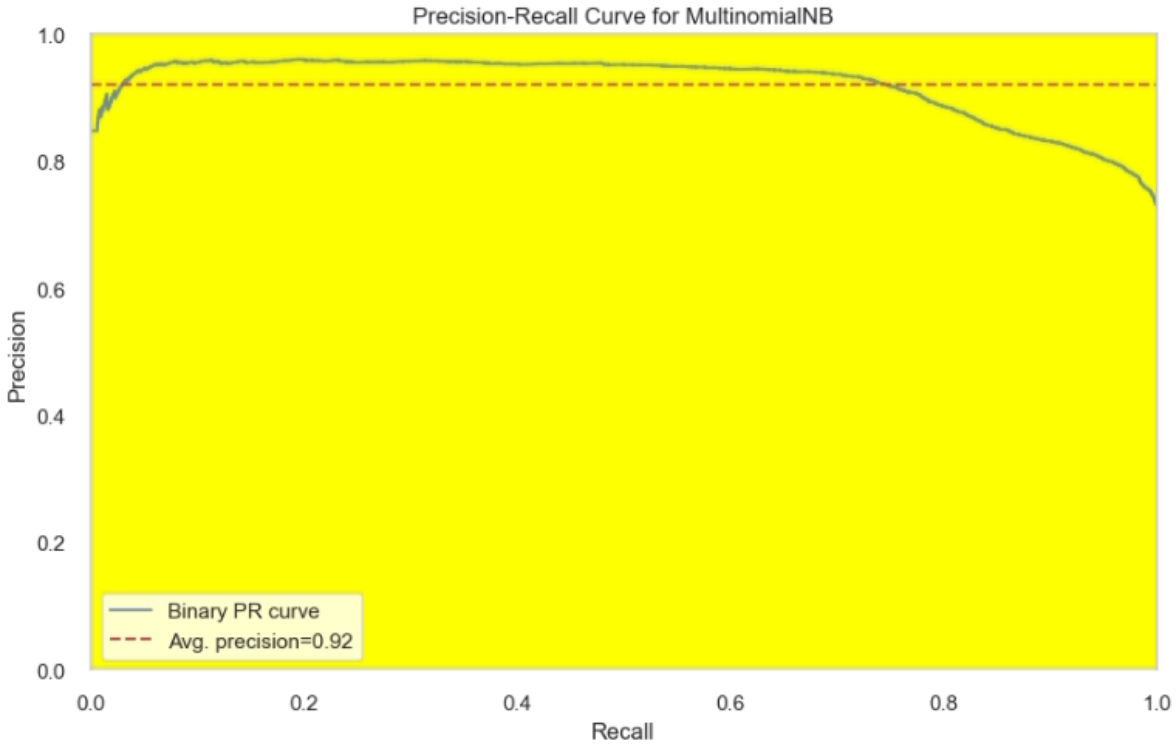
- Overall Model Performance:

The accuracy of 80.15% suggests that the model performs reasonably well in making correct predictions across both classes.

Class 0 (Non-recommendation) Analysis: The lower precision (0.6496) and recall (0.5711) for Class 0 indicate that the model may have challenges correctly identifying instances of non-recommendation. It has a relatively higher rate of false negatives. The F1-0 score (0.6077) considers the balance between precision and recall for Class 0.

Class 1 (Recommendation) Analysis: The higher precision (0.8486) and recall (0.8864) for Class 1 suggest that the model is effective in correctly identifying instances of recommendation. It has a relatively lower rate of false negatives. The F1-1 score (0.8671) considers the balance between precision and recall for Class 1.

```
viz = PrecisionRecallCurve(
                           MultinomialNB(),
                           classes=nb.classes_,
                           per_class=True,
                           cmap="Set1"
                          )

fig, ax = plt.subplots(figsize=(10, 6))
ax.set_facecolor('yellow')

viz.fit(X_train_count,y_train)
viz.score(X_test_count, y_test)
viz.show();
```



- In the evaluation of the model's Precision-Recall curve, it is observed that with an increase in the classification threshold from 8.8 to 9.4, there is a corresponding improvement in precision, indicating enhanced accuracy in positive predictions.
- Concurrently, recall also rises, affirming the model's capability to capture a greater number of positive instances with higher thresholds.

- The inherent trade-off between precision and recall underscores the strategic importance of selecting a threshold aligned with specific application objectives.

- Noteworthy is the identification of optimal operating points, particularly around thresholds of 9.3 to 9.4, where the model exhibits heightened precision, emphasizing its proficiency in accurately predicting positive instances. This nuanced analysis provides valuable insights for strategic decision-making in the deployment of the model for realworld applications

```
nb_AP_count = viz.score_
```

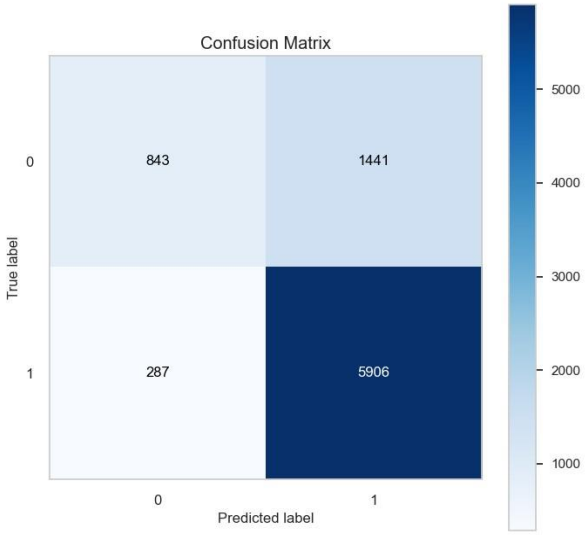
9.1.b Naive Bayes with TF-IDF Vectorizer

```
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
nb = MultinomialNB()
nb.fit(X_train_tf_idf, y_train)
```

```
MultinomialNB()
```

```
print("NB MODEL")
eval(nb, X_train_tf_idf, X_test_tf_idf)
```

```
NB MODEL
Test_Set
              precision    recall  f1-score   support

           0       0.75      0.37      0.49      2284
           1       0.80      0.95      0.87      6193

    accuracy                           0.80      8477
   macro avg       0.77      0.66      0.68      8477
weighted avg       0.79      0.80      0.77      8477

Train_Set
              precision    recall  f1-score   support

           0       0.81      0.42      0.56      9134
           1       0.82      0.96      0.89     24770

    accuracy                           0.82     33904
   macro avg       0.81      0.69      0.72     33904
weighted avg       0.82      0.82      0.80     33904
```

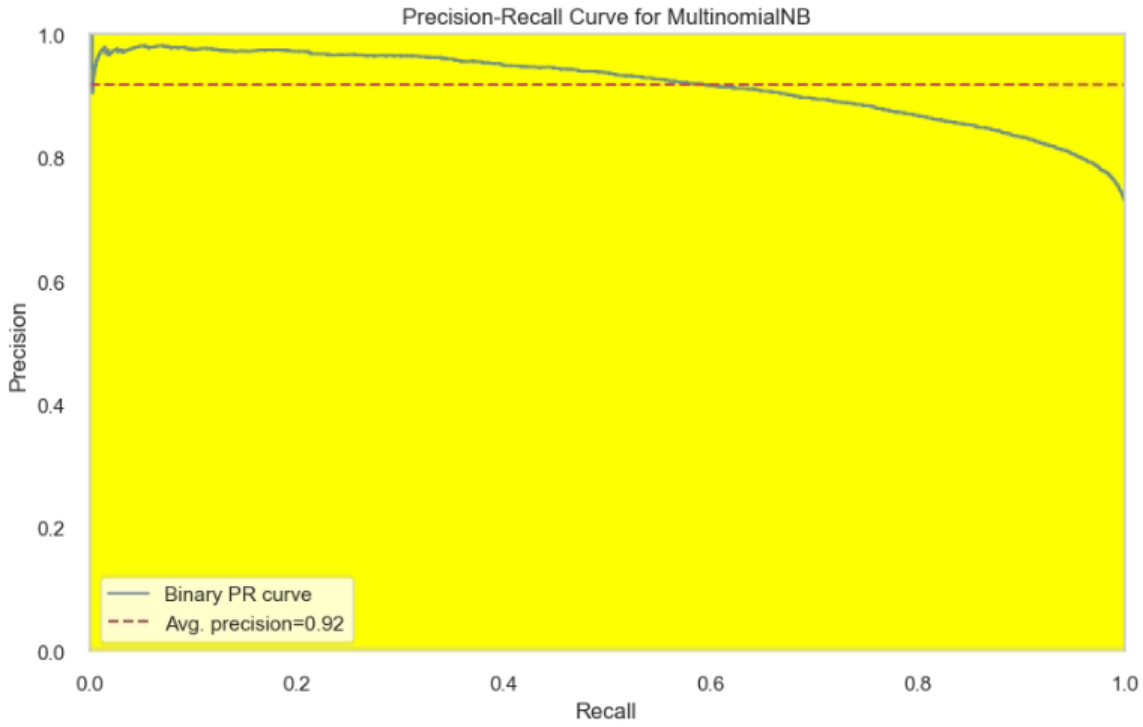


```
custom_scorer = {'accuracy': make_scorer(accuracy_score),
                 'precision-0': make_scorer(precision_score, pos_label=0),
                 'recall-0': make_scorer(recall_score, pos_label=0),
                 'f1-0': make_scorer(f1_score, pos_label=0),
                 'precision-1': make_scorer(precision_score, pos_label=1),
                 'recall-1': make_scorer(recall_score, pos_label=1),
                 'f1-1': make_scorer(f1_score, pos_label=1)
                 }

for i, j in custom_scorer.items():
    model = BernoulliNB()
    scores = cross_val_score(model, X_train_tf_idf, y_train, cv = 10, scoring = j).mean()
    if i == "recall-1":
        nb_tfidf_rec = scores
    elif i == "f1-1":
        nb_tfidf_f1 = scores
    print(f" {i:20} score for tfidf : {scores}\n")
```

```
 accuracy             score for tfidf : 0.7956587409497116

 precision-0          score for tfidf : 0.6386912637070589

 recall-0             score for tfidf : 0.5563845595231532

 f1-0                 score for tfidf : 0.5946546001579927

 precision-1          score for tfidf : 0.843836393630539

 recall-1             score for tfidf : 0.8838918046023416

 f1-1                 score for tfidf : 0.8633931251904017
```

- Here is The comparison between Naive Bayes with Count Vectorizer and TF-IDF
  - The Naive Bayes model and Naive Bayes TF-IDF model show almost identical performance metrics.
  - Both models achieve high accuracy, precision, recall, and F1-scores, indicating their effectiveness in classifying instances.
  - The TF-IDF feature representation doesn't significantly impact the model's performance in this case.

```
viz = PrecisionRecallCurve(
                           MultinomialNB(),
                           classes=nb.classes_,
                           per_class=True,
                           cmap="Set1"
                          )

fig, ax = plt.subplots(figsize=(10, 6))
ax.set_facecolor('yellow')

viz.fit(X_train_tf_idf, y_train)
viz.score(X_test_tf_idf, y_test)
viz.show();
```




```
nb_AP_tfidf = viz.score_
```

9.2 Support Vector Machine (SVM)

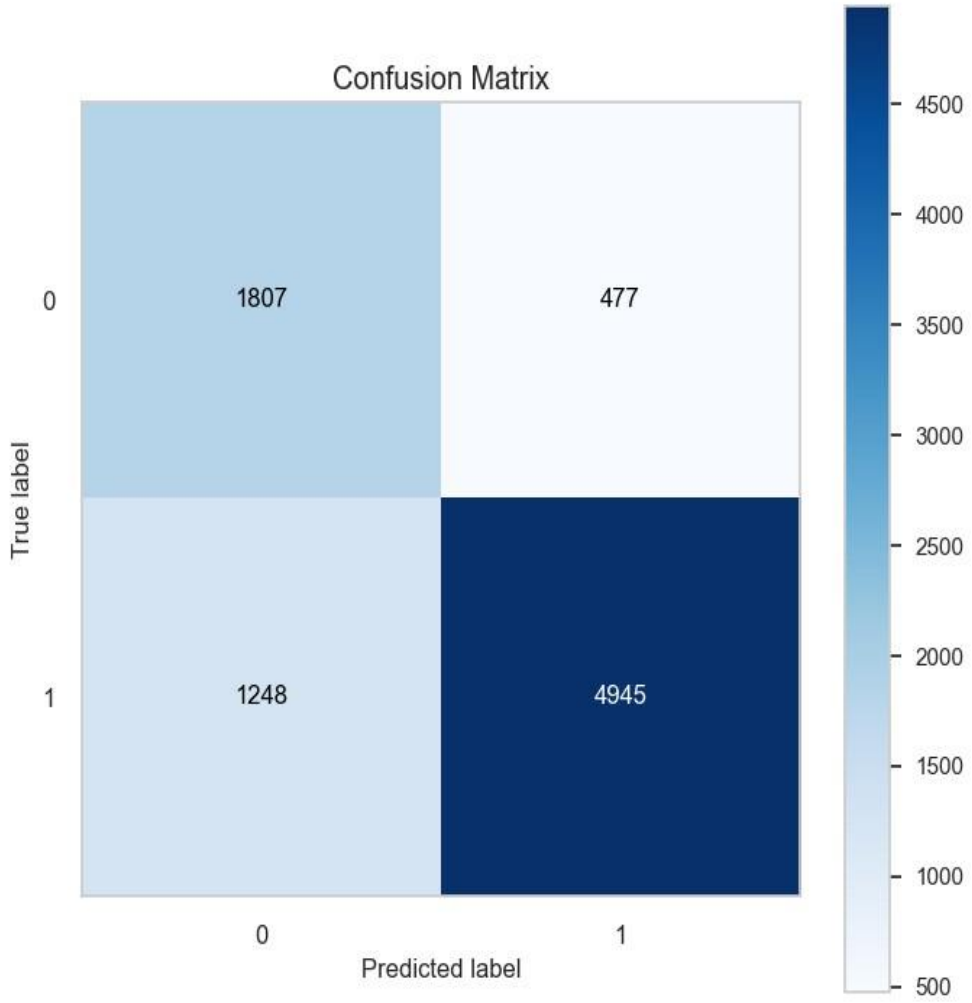
I want to make comparison between SVM with Count Vectorizer and TF-IDF (Term Frequency-Inverse Document Frequency) by evaluating their performance in the context of current dataset.

9.2.a Support Vector Machine (SVM) With Count Vectorizer

```
from sklearn.svm import LinearSVC

svc = LinearSVC(C=0.01, class_weight="balanced", random_state=101)
svc.fit(X_train_count,y_train)
```

```
print("SVC MODEL")

eval(svc, X_train_count, X_test_count)
```

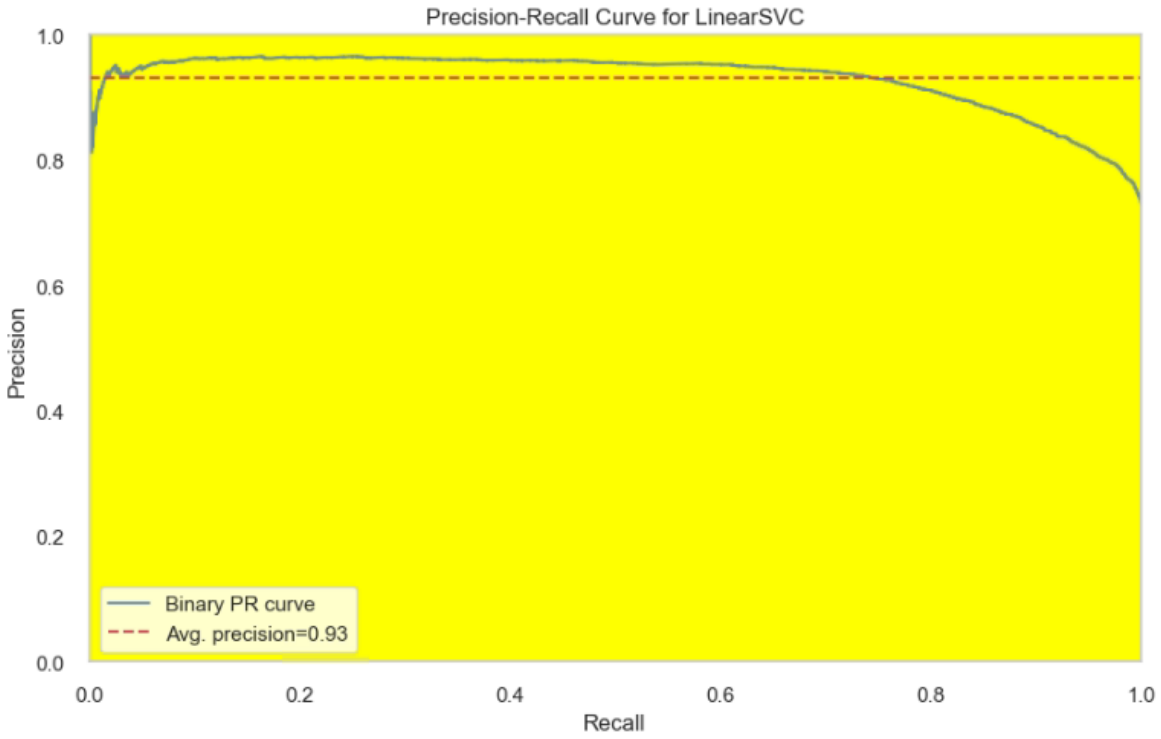
```
SVC MODEL
Test_Set
              precision    recall  f1-score   support

           0       0.59      0.79      0.68      2284
           1       0.91      0.80      0.85      6193

    accuracy                           0.80      8477
   macro avg       0.75      0.79      0.76      8477
weighted avg       0.83      0.80      0.80      8477

Train_Set
              precision    recall  f1-score   support

           0       0.66      0.88      0.75      9134
           1       0.95      0.83      0.89     24770

    accuracy                           0.84     33904
   macro avg       0.80      0.86      0.82     33904
weighted avg       0.87      0.84      0.85     33904
```


Confusion Matrix
1807
477
1248
4945
0
1
True label
0
1
Predicted label
4500
4000
3500
3000
2500
2000
1500
1000
500

```
custom_scorer = {'accuracy': make_scorer(accuracy_score),
                 'precision-0': make_scorer(precision_score, pos_label=0),
                 'recall-0': make_scorer(recall_score, pos_label=0),
                 'f1-0': make_scorer(f1_score, pos_label=0),
                 'precision-1': make_scorer(precision_score, pos_label=1),
                 'recall-1': make_scorer(recall_score, pos_label=1),
                 'f1-1': make_scorer(f1_score, pos_label=1)
                }

for i, j in custom_scorer.items():
    model = LinearSVC(C=0.01, class_weight="balanced", random_state=101)
    scores = cross_val_score(model, X_train_count, y_train, cv = 10, scoring = j).mean()
    if i == "recall-1":
        svc_count_rec = scores
    elif i == "f1-1":
        svc_count_f1 = scores
    print(f" {i:20} score for count : {scores}\n")
```

```
 accuracy             score for count : 0.8055976039298891

 precision-0          score for count : 0.6032434266204743

 recall-0             score for count : 0.8139898763544332

 f1-0                 score for count : 0.692892508272392

 precision-1          score for count : 0.9212820816975273

 recall-1             score for count : 0.8025030278562777

 f1-1                 score for count : 0.8577776449210497
```

```
viz = PrecisionRecallCurve(
                           LinearSVC(C=0.01, class_weight="balanced", random_state=101),
                           classes=svc.classes_,
                           per_class=True,
                           cmap="Set1"
                          )

fig, ax = plt.subplots(figsize=(10, 6))
ax.set_facecolor('yellow')

viz.fit(X_train_count,y_train)
viz.score(X_test_count, y_test)
viz.show();
```



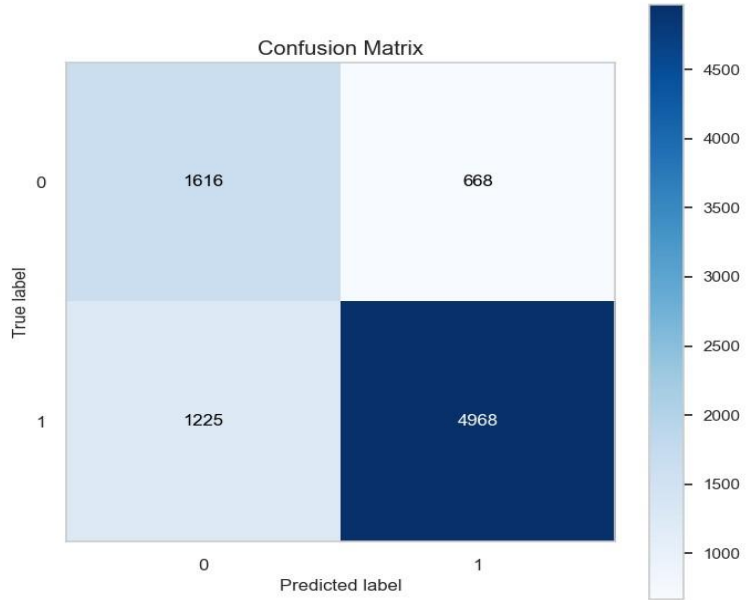

```
svc_AP_count = viz.score_
```

9.2.b Support Vector Machine (SVM) With TF-IDF Vectorizer

```
svc = LinearSVC(C=0.01, class_weight="balanced", random_state=101)

svc.fit(X_train_tf_idf, y_train)
```

```
print("SVC MODEL")

eval(svc, X_train_tf_idf, X_test_tf_idf)
```

```
SVC MODEL
Test_Set
              precision    recall  f1-score   support

           0       0.57      0.71      0.63      2284
           1       0.88      0.80      0.84      6193

    accuracy                           0.78      8477
   macro avg       0.73      0.75      0.74      8477
weighted avg       0.80      0.78      0.78      8477

Train_Set
              precision    recall  f1-score   support

           0       0.60      0.77      0.67      9134
           1       0.90      0.81      0.86     24770

    accuracy                           0.80     33904
   macro avg       0.75      0.79      0.77     33904
weighted avg       0.82      0.80      0.81     33904
```

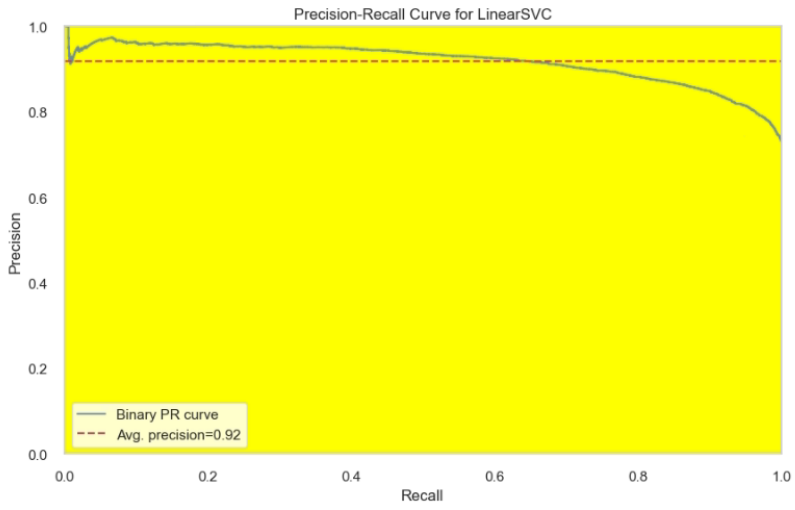


```
custom_scorer = {'accuracy': make_scorer(accuracy_score),
                 'precision-0': make_scorer(precision_score, pos_label=0),
                 'recall-0': make_scorer(recall_score, pos_label=0),
                 'f1-0': make_scorer(f1_score, pos_label=0),
                 'precision-1': make_scorer(precision_score, pos_label=1),
                 'recall-1': make_scorer(recall_score, pos_label=1),
                 'f1-1': make_scorer(f1_score, pos_label=1)
                 }

for i, j in custom_scorer.items():
    model = LinearSVC(C=0.01, class_weight="balanced", random_state=101)
    scores = cross_val_score(model, X_train_tf_idf, y_train, cv = 10, scoring = j).mean()
    if i == "recall-1":
        svc_tfidf_rec = scores
    elif i == "f1-1":
        svc_tfidf_f1 = scores
    print(f" {i:20} score for tfidf : {scores}\n")
```

```
accuracy             score for tfidf : 0.7891695699791831

precision-0          score for tfidf : 0.58528446401157

recall-0             score for tfidf : 0.746989030320606

f1-0                 score for tfidf : 0.656253960702193

precision-1          score for tfidf : 0.8961298988365762

recall-1             score for tfidf : 0.8047234557932985

f1-1                 score for tfidf : 0.8479455028110575
```

- **Here is The comparison between SVM with Count Vectorizer and TF-IDF**
  - The SVM model with the count vectorization scheme performs better overall compared to the SVM TF-IDF model.
  - The count-based SVM has higher accuracy, precision, recall, and F1-scores for both classes.
  - The TF-IDF representation has a slightly lower performance, especially in precision and F1-score for Class 0.

```
viz = PrecisionRecallCurve(
                           LinearSVC(C=0.01, class_weight="balanced", random_state=101),
                           classes=svc.classes_,
                           per_class=True,
                           cmap="Set1"
                          )

fig, ax = plt.subplots(figsize=(10, 6))
ax.set_facecolor('yellow')

viz.fit(X_train_tf_idf,y_train)
viz.score(X_test_tf_idf, y_test)
viz.show();
```



10) DEEP LEARNING MODELLING

```
import numpy as np
import pandas as pd
import tensorflow as tf
import keras
from keras import layers
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Dense, GRU, Embedding
from keras.optimizers import Adam
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

```
df0 = pd.read_csv('Luxury-Car-Review.csv')
df_dl = df0.copy()
df_dl.head()
```

	Unnamed: 0	review_date	author_name	age	vehicle_make	vehicle_title	review_title	review	rating	recommended_ind
0	0	on 04/28/17 08:08 AM (PDT)	Garrett Stites	33	Ferrari	2015 Ferrari 458 Italia Convertible Spider 2dr Convertible (4.5L 8cyl 7AM)	The best car around!	This car gets great gas mileage and is the best car you could buy.	5.00	1
1	1	on 11/19/11 16:47 PM (PST)	debu99	34	Ferrari	2006 Ferrari 612 Scaglietti Coupe F1 2dr Coupe (5.7L 12cyl 6AM)	keeps on beeing just great	Owning the 612 now over 3 years and using it in evey day traffic it just keeps on being great,The paint job is the only problem, technical it's fantastic and by far the best car i ever owned..	4.75	1
2	2	on 06/28/07 22:12 PM (PDT)	Arnell Baylet	60	Ferrari	2006 Ferrari 612 Scaglietti Coupe F1 2dr Coupe (5.7L 12cyl 6AM)	Incredible Ride, Sticker Shock, Low MPG	Best controllable acceleration ever witnessed in a vehicle. Breaking is mind blowing. Handling exceptional (except tears through $400 tires). Seat comfort and visibility are the best I have wit...	5.00	1
3	3	on 05/30/07 10:56 AM (PDT)	gregMTU	50	Ferrari	2006 Ferrari 612 Scaglietti Coupe F1 2dr Coupe (5.7L 12cyl 6AM)	612 Scag The Best!	The engine is strong pulling right up to 7000 rpm, handling is excellent, yet remains fairly quiet interior wise unlike our former Aston. 612 really is an enjoyable car!	5.00	1
4	4	on 01/21/07 12:58 PM (PST)	Darrel McOnnery	47	Ferrari	2006 Ferrari 612 Scaglietti Coupe 2dr Coupe (5.7L 12cyl 6M)	612 Scaglietti! Quietly the Best Ferrari Ever!	This car, the 612 Scaglietti, is overlooked compared to the Spider and Enzo, however it performs better than any Ferrari I have ever owned or driven in. People go nuts over this car when you "r...	5.00	1

```
df_dl = df_dl[["review","recommended_ind"]]
df_dl.head()
```

	review	recommended_ind
0	This car gets great gas mileage and is the best car you could buy.	1
1	Owning the 612 now over 3 years and using it in evey day traffic it just keeps on being great,The paint job is the only problem, technical it's fantastic and by far the best car i ever owned..	1
2	Best controllable acceleration ever witnessed in a vehicle. Breaking is mind blowing. Handling exceptional (except tears through $400 tires). Seat comfort and visibility are the best I have wit...	1
3	The engine is strong pulling right up to 7000 rpm, handling is excellent, yet remains fairly quiet interior wise unlike our former Aston. 612 really is an enjoyable car!	1
4	This car, the 612 Scaglietti, is overlooked compared to the Spider and Enzo, however it performs better than any Ferrari I have ever owned or driven in. People go nuts over this car when you "r...	1

```
df_dl.shape
```

```
(55164, 2)
```

```
df_dl.dropna(inplace = True)
```

```
df_dl.shape
```

```
(42381, 2)
```

10.1 Tokenization

```
X = df_dl['review'].values
y = df_dl['recommended_ind'].values
```

```
num_words = 10000
# We have defined the most frequent 10000 repeated words in corpus for tokenizing. We ignore the rest.
import nltk
from nltk import word_tokenize#add this one from internet
#from nltk.tokenize import Tokenizer

# Create a Tokenizer instance
tokenizer = Tokenizer()
tokenizer = Tokenizer(num_words=num_words)
# The default values of "filters" are '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n'.
# If you also want to filters the numbers, then just "1234567890" at the end.
```

```
tokenizer.fit_on_texts(X)
```

10.2 Creating Word Index

```
tokenizer.word_index
```

```
{'the': 1,
 'and': 2,
 'i': 3,
 'a': 4,
 'is': 5,
 'to': 6,
 'it': 7,
 'car': 8,
 'in': 9,
 'of': 10,
 'this': 11,
 'for': 12,
 'with': 13,
 'my': 14,
 'have': 15,
 'on': 16,
 'but': 17,
 'that': 18,
 'you': 19,
```

```
len(tokenizer.word_index)
```

```
50888
```

10.3 Converting Tokens To Numeric

```
X_num_tokens = tokenizer.texts_to_sequences(X)
```

```
num_tokens = [len(tokens) for tokens in X_num_tokens]
num_tokens = np.array(num_tokens)
```

```
np.array(X_num_tokens)
```

```
array([list([11, 8, 300, 27, 106, 127, 2, 5, 1, 75, 8, 19, 146, 110]),
       list([510, 1, 7751, 74, 85, 89, 70, 2, 716, 7, 9, 215, 565, 7, 41, 1054, 16, 302, 27, 1, 635, 773, 5, 1, 57, 135, 2924,
48, 448, 2, 100, 122, 1, 75, 8, 3, 112, 67]),
       list([75, 9520, 221, 112, 9, 4, 45, 1399, 5, 743, 3231, 103, 903, 441, 6684, 345, 766, 145, 144, 164, 2, 1025, 25, 1, 7
5, 3, 15, 9, 1135, 6873, 2, 3, 81, 1718, 282, 6, 15, 67, 89, 969]),
       ...,
       list([33, 122, 33, 62, 23, 1, 8, 12, 53, 4, 414, 399, 10, 129, 1211, 101, 1501, 99, 9, 1, 143, 458, 149, 2, 87, 16, 130
9, 68, 6, 29, 8, 503, 40, 142, 568, 265, 6, 68, 2, 7502, 8, 13, 1, 929, 10, 4, 722, 44, 1, 8, 2, 236]),
       list([]), list([27, 481, 8])], dtype=object)
```

```
X[105]
```

```
' This is the best Ferrari you can buy. I have had three other Ferraris. This is simply the best!'
```

```
print(X_num_tokens[105])
```

```
[11, 5, 1, 75, 1135, 19, 61, 110, 3, 15, 23, 364, 79, 5268, 11, 5, 557, 1, 75]
```

```
# tokenizer.word_index["The"]
# This code will give you an error since "The" which is not among the most repeated 10000 words was excluded while tokenizing
```

```
tokenizer.word_index["shirt"]
```

```
8043
```

```
tokenizer.word_index["exactly"]
```

```
1223
```

10.4 The Determination of Maximum Number of Tokens

- This part is the preparation for padding.

```
num_tokens.mean()
```

```
85.88980911257403
```

```
num_tokens.max()
```

```
963
```

```
num_tokens.argmax()
```

```
36777
```

```
X[16263]
```

```
" After leaving the silver star in 2007, I went to a BMW 528I and then a Cadillac SRX. I have owned and leased 5 Mercedes since
1997 and I missed not just the brand but the service from the dealership.My 2011 C300 Sport Sedan is fun to drive, comfortable
and quite good on gas averaging about 27 on the highway. I've had it 3 weeks and after comparing it to a few vehicles like the
BMW,Lexus, Infiniti and Cadillac, I decided to go back to Mercedes. The japanese manufacturers offer a little more bang for you
r buck, but I have all the features that I need in the P1 package. Its sharp,nimble and so far, an all around great vehicle."
```

```
len(X[16263])
```

```
632
```

```
num_tokens.argmin()
```

```
442
```

```
X[820]
```

```
' What a fun car to drive I have a big smile on my face when driving my M35. We have tested the X3, Volvo S80, MDX, Avalon, GS
350 and the M35 was the best bang for the buck, roomier then most others in the same class. It seems like that most carmakers m
ake the cars smaller and smaller hope that Infiniti stays with their roomier layout I love the Lexus GS but it is too small for
me. I am getting AVG of 20.6 mpg will try manual shifting see if I get a better mileage. I tinted the windows and it looks so c
ool, this car rocks.'
```

```
len(X[820])
```

```
528
```

10.5 Fixing Token Counts of All documents (Pad Sequences)

```
len(X_num_tokens[105])
```

```
19
```

```
np.array(X_num_tokens[105])
```

```
array([  11,    5,    1,   75, 1135,   19,   61,  110,    3,   15,   23,
        364,   79, 5268,   11,    5,  557,    1,   75])
```

```
len(X_num_tokens[106])
```

```
101
```

```
np.array(X_num_tokens[106])
```

```
array([1526,   89,   70,   12,   11,    8,   24,    3,  200,    4, 7506,
         12,    4, 2454,   94,   21,  476, 4035,    6,  204,  255,   12,
          1, 5371,    7,   21,   65,  306,    1,  595,   11,    8,  108,
       2536,   56,    6,    1, 1135, 1770,    1,  236,   96,    2,  186,
         47,  246,    5,  237,   28,  260,    6,   29,    9,  563,  565,
         24,   65,   17,   37,    1,   93, 3099,   56,    5,   37,   19,
         25, 2313,    6,   97,   11,    8,    5,   30,   53,   75,    8,
         80,  112,  205,  437,   86,  291, 1458, 1479,  291, 1401, 1479,
        286,  824,  272,  286,  947,  349, 1720, 1171, 1720, 5931,  140,
       1294, 1171])
```

```
num_tokens = [len(tokens) for tokens in X_num_tokens]

num_tokens = np.array(num_tokens)
```

```
num_tokens
```

```
array([14, 38, 40, ..., 52,  0,  3])
```

```
max_tokens = 103
```

```
sum(num_tokens < max_tokens) / len(num_tokens)
```

```
0.656261060380831
```

```
sum(num_tokens < max_tokens) # the number of documents which have 103 or less tokens
```

```
27813
```

```
len(num_tokens)  # total number of all documents in corpus which is constrained by num_words as 20000
```

```
42381
```

```
X_pad = pad_sequences(X_num_tokens, maxlen=max_tokens)
```

```
X_pad.shape
```

```
(42381, 103)
```

```
X_pad[105]
```

```
array([   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,    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,    0,   11,    5,    1,   75,
       1135,   19,   61,  110,    3,   15,   23,  364,   79, 5268,   11,
          5,  557,    1,   75])
```

```
X_pad[106]
```

```
array([   0,    0, 1526,   89,   70,   12,   11,    8,   24,    3,  200,
          4, 7506,   12,    4, 2454,   94,   21,  476, 4035,    6,  204,
        255,   12,    1, 5371,    7,   21,   65,  306,    1,  595,   11,
          8,  108, 2536,   56,    6,    1, 1135, 1770,    1,  236,   96,
          2,  186,   47,  246,    5,  237,   28,  260,    6,   29,    9,
        563,  565,   24,   65,   17,   37,    1,   93, 3099,   56,    5,
         37,   19,   25, 2313,    6,   97,   11,    8,    5,   30,   53,
         75,    8,   80,  112,  205,  437,   86,  291, 1458, 1479,  291,
       1401, 1479,  286,  824,  272,  286,  947,  349, 1720, 1171, 1720,
       5931,  140, 1294, 1171])
```

10.6 Train | Set & Split

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_pad, y, test_size=0.2, stratify=y, random_state=101)

# we have been using stratify to prevent imbalance.
```

10.7 Modeling

```
model = Sequential()
```

```
WARNING:tensorflow:From C:\Users\Administrator\AppData\Roaming\Python\Python310\site-packages\keras\src\backend.py:873: The nam
e tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.
```

```
embedding_size = 100
```

```
model.add(Embedding(input_dim=num_words,
                    output_dim=embedding_size,
                    input_length=max_tokens,
                    name='embedding_layer'))
```

```
model.add(GRU(units=48, return_sequences=True))
model.add(GRU(units=24, return_sequences=True))
model.add(GRU(units=12))
model.add(Dense(1, activation='sigmoid'))
```

```
optimizer = Adam(learning_rate=0.006)
```

```
model.compile(loss='binary_crossentropy',
              optimizer=optimizer,
              metrics=['Recall'])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_layer (Embedding  (None, 103, 100)          1000000
 )

 gru (GRU)                   (None, 103, 48)           21600

 gru_1 (GRU)                 (None, 103, 24)           5328

 gru_2 (GRU)                 (None, 12)                1368

 dense (Dense)               (None, 1)                 13

_________________________________________________________________
```

```
=================================================================
Total params: 1028309 (3.92 MB)
Trainable params: 1028309 (3.92 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
from keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor="val_loss", mode="auto",
                           verbose=1, patience = 10, restore_best_weights=True)
```

```
pd.Series(y_train).value_counts(normalize=True)
```

```
1   0.73
0   0.27
dtype: float64
```

```
weights = {0:82, 1:18}
```

```
model.fit(X_train, y_train, epochs=30, batch_size=256, class_weight=weights,
          validation_data=(X_test, y_test), callbacks=[early_stop])
```

```
Epoch 1/30
WARNING:tensorflow:From C:\Users\Administrator\AppData\Roaming\Python\Python310\site-packages\keras\src\utils\tf_utils.py:492:
The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Administrator\AppData\Roaming\Python\Python310\site-packages\keras\src\engine\base_layer_util
s.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_func
tions instead.

133/133 [==============================] - 54s 296ms/step - loss: 16.6155 - recall: 0.6736 - val_loss: 0.4255 - val_recall: 0.8
397
Epoch 2/30
133/133 [==============================] - 37s 279ms/step - loss: 11.7321 - recall: 0.8086 - val_loss: 0.5090 - val_recall: 0.7
491
Epoch 3/30
133/133 [==============================] - 38s 283ms/step - loss: 8.8575 - recall: 0.8529 - val_loss: 0.4900 - val_recall: 0.77
98
Epoch 4/30
133/133 [==============================] - 38s 286ms/step - loss: 6.5288 - recall: 0.8904 - val_loss: 0.4476 - val_recall: 0.85
81
Epoch 5/30
133/133 [==============================] - 38s 284ms/step - loss: 4.8585 - recall: 0.9201 - val_loss: 0.5476 - val_recall: 0.83
06
...
Epoch 11/30
133/133 [==============================] - ETA: 0s - loss: 1.6919 - recall: 0.9752Restoring model weights from the end of the b
est epoch: 1.
133/133 [==============================] - 37s 281ms/step - loss: 1.6919 - recall: 0.9752 - val_loss: 0.7615 - val_recall: 0.86
45
Epoch 11: early stopping

<keras.src.callbacks.History at 0x1534f4c9cc0>
```

```
model.save('NLP_Sentiment_Analysis_Project')
```

```
INFO:tensorflow:Assets written to: NLP_Sentiment_Analysis_Project\assets
```

10.8 Model Evaluation

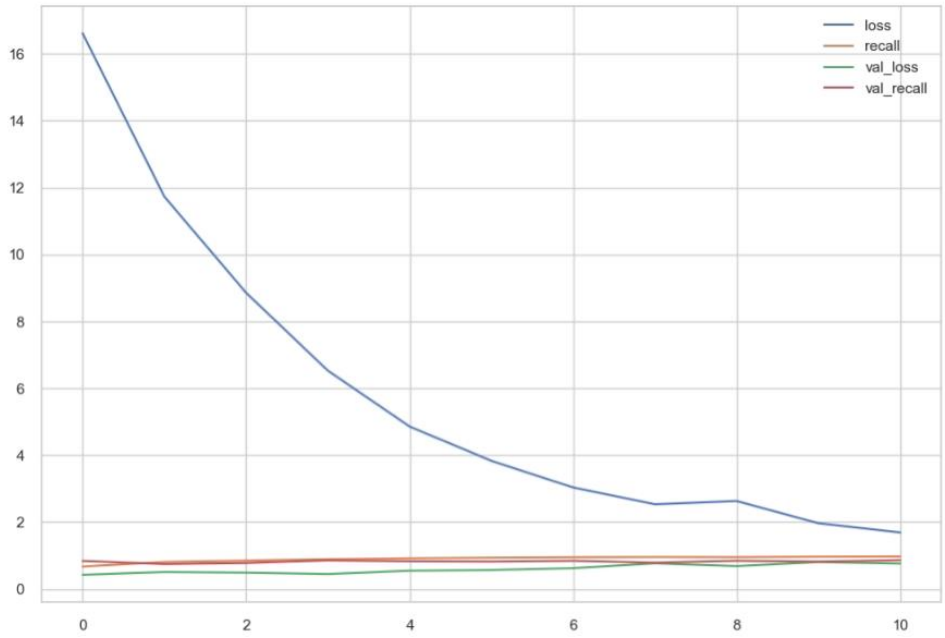
```
model_loss = pd.DataFrame(model.history.history)
model_loss.head()
```

	loss	recall	val_loss	val_recall
0	16.62	0.67	0.43	0.84
1	11.73	0.81	0.51	0.75
2	8.86	0.85	0.49	0.78
3	6.53	0.89	0.45	0.86
4	4.86	0.92	0.55	0.83

```
model_loss.plot();
```

Interpretation:

1. **Training Progress:**
   - As training progresses (from epoch 0 to epoch 4), the loss decreases, indicating that the model is learning and improving its predictions.
2. **Recall Improvement:**
   - The recall on the training set increases from 0.67 to 0.92, suggesting that the model becomes better at identifying positive instances in the training data.
3. **Validation Performance:**
   - The validation loss initially decreases, but in later epochs, it starts to increase. This could indicate potential overfitting, where the model is too closely fitting the training data and not generalizing well to new data.
4. **Validation Recall:**
   - The validation recall fluctuates but generally remains high, indicating that the model maintains good performance on capturing relevant instances in the validation dataset.



```
model.evaluate(X_train, y_train)
```

```
1060/1060 [==============================] - 37s 35ms/step - loss: 0.3794 - recall: 0.8604

[0.37942779064178467, 0.860395610332489]
```

```
model.evaluate(X_test, y_test)
```

```
265/265 [==============================] - 9s 34ms/step - loss: 0.4255 - recall: 0.8397

[0.4254623353481293, 0.8396576642990112]
```

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1_score, roc_auc_score

y_train_pred = (model.predict(X_train) >= 0.5).astype("int32")

print(confusion_matrix(y_train, y_train_pred))
print("-----------------------------------------------------------")
print(classification_report(y_train, y_train_pred))
```

```
1060/1060 [==============================] - 39s 33ms/step
[[ 7921  1213]
 [ 3458 21312]]
-----------------------------------------------------------
              precision    recall  f1-score   support

           0       0.70      0.87      0.77      9134
           1       0.95      0.86      0.90     24770

    accuracy                           0.86     33904
   macro avg       0.82      0.86      0.84     33904
weighted avg       0.88      0.86      0.87     33904
```

```
y_pred = (model.predict(X_test) >= 0.5).astype("int32")

print(confusion_matrix(y_test, y_pred))
print("-----------------------------------------------------------")
print(classification_report(y_test, y_pred))
```

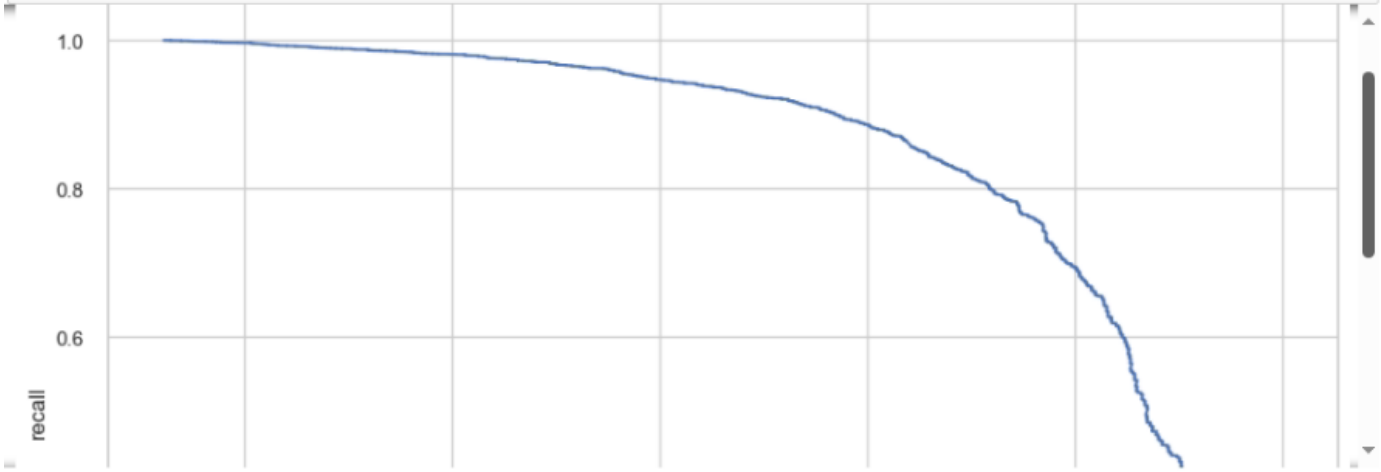
```
1060/1060 [==============================] - 39s 33ms/step
[[ 7921  1213]
 [ 3458 21312]]
-----------------------------------------------------------
              precision    recall  f1-score   support

           0       0.70      0.87      0.77      9134
           1       0.95      0.86      0.90     24770

    accuracy                           0.86     33904
   macro avg       0.82      0.86      0.84     33904
weighted avg       0.88      0.86      0.87     33904
```

```
y_pred = (model.predict(X_test) >= 0.5).astype("int32")

print(confusion_matrix(y_test, y_pred))
print("-----------------------------------------------------------")
print(classification_report(y_test, y_pred))
```

```
265/265 [==============================] - 9s 33ms/step
[[1812  472]
 [ 993 5200]]
-----------------------------------------------------------
              precision    recall  f1-score   support

           0       0.65      0.79      0.71      2284
           1       0.92      0.84      0.88      6193

    accuracy                           0.83      8477
   macro avg       0.78      0.82      0.79      8477
weighted avg       0.84      0.83      0.83      8477
```

- Accuracy: 83% (proportion of correctly predicted instances out of the totalو Macro Avg and Weighted Avg metrics provide additional insights, considering class distribution.

```
from sklearn.metrics import precision_recall_curve, average_precision_score

y_pred_proba = model.predict(X_test)
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_proba)

# plt.plot([1, 0], [0, 1],'k--')
plt.plot(precision, recall)
plt.xlabel('precision')
plt.ylabel('recall')
plt.title('Precision Recall Curve')
plt.show()
```




```
from sklearn.metrics import precision_recall_curve, average_precision_score, recall_score

DL_AP = average_precision_score(y_test, y_pred_proba)
DL_f1 = f1_score(y_test, y_pred)
DL_rec = recall_score(y_test, y_pred)
```

11) PREDICTION

```
review1= "Be sure you're ready to be coddled in the ultimate in comfort and true American luxury"
review2= 'I bought my Navigator a few months ago and about a month after buying it we were in a 5 car collision ie Bauer! This ve
review3= 'This is a wonderful SUV very comfortable very comfortable and eliability is outstanding.  I would highly recommend this
review4= 'Ive had my Navigator for a few months now. I was leasing a 2008 GMC Yukon Denali before I decided to buy. I originally
review5= 'This car is an overall good vehicle. It has room for 7 passengers which makes it convenient for families with children.
review6= 'I bought this 6 months ago.  it took a few weeks to get the hang of driving it.  now I love it.  Plenty of power.  Real
review7= 'The exterior chrome and square gauges have a retro 1960s look.  The interior is all about comfort, luxury and modern te
review8= 'This is my third Navigator. All have been 2 year leases. I intend to buy this one at lease end. My first two had some e
review9= 'Love it! This is the most comfortable car I have ever been in.  When my friends and I go out their Hondas and BMWs stay
review10= 'This is our second Navigator and this is much better then the 2005 model we replaced. Recently towed a 7500 lb boat ar
review11= 'I was going to buy an Infiniti QX56 but it was too tall for our garage. I love this Lincoln in all respects. It has al
review12= 'a very happy with my new Navigator. It is very comfortable and the build quality is just as good as any other luxury v
review13= 'I had a 2000 Expedition before that I bought new, so, I have been driving these beasts for some time. I love the Expec
review14= 'This is our 2nd version of the Navigator. Our previous was a 2004 and this model is definitely improved. We have a 200
reviews=[review1,review2,review3,review4,review5,review6,review7,review8,review9,review10,review11,review12,review13,review14]
```

Let's convert reviews above to numeric by tokenizing.

```
tokens = tokenizer.texts_to_sequences(reviews)
```

Let's pad the tokenized reviews.

```
tokens_pad = pad_sequences(tokens, maxlen=max_tokens)
tokens_pad.shape
```

```
(14, 103)
```

Let's predict the sentiment of our reviews.

```
mod_pred = model.predict(tokens_pad)
```

```
1/1 [==============================] - 0s 131ms/step
```

```
mod_pred
```

```
mod_pred
```

```
array([[0.6714144 ],
       [0.6277289 ],
       [0.8225942 ],
       [0.82905275],
       [0.80469924],
       [0.83737713],
       [0.8458951 ],
       [0.69438416],
       [0.7622067 ],
       [0.67345405],
       [0.81147975],
       [0.81266606],
       [0.5680306 ],
       [0.87765026]], dtype=float32)
```

Let's create DataFrame for visually a better understanding.

```
df_pred = pd.DataFrame(mod_pred, index=reviews)
df_pred.rename(columns={0: 'Pred_Proba'}, inplace=True)
```

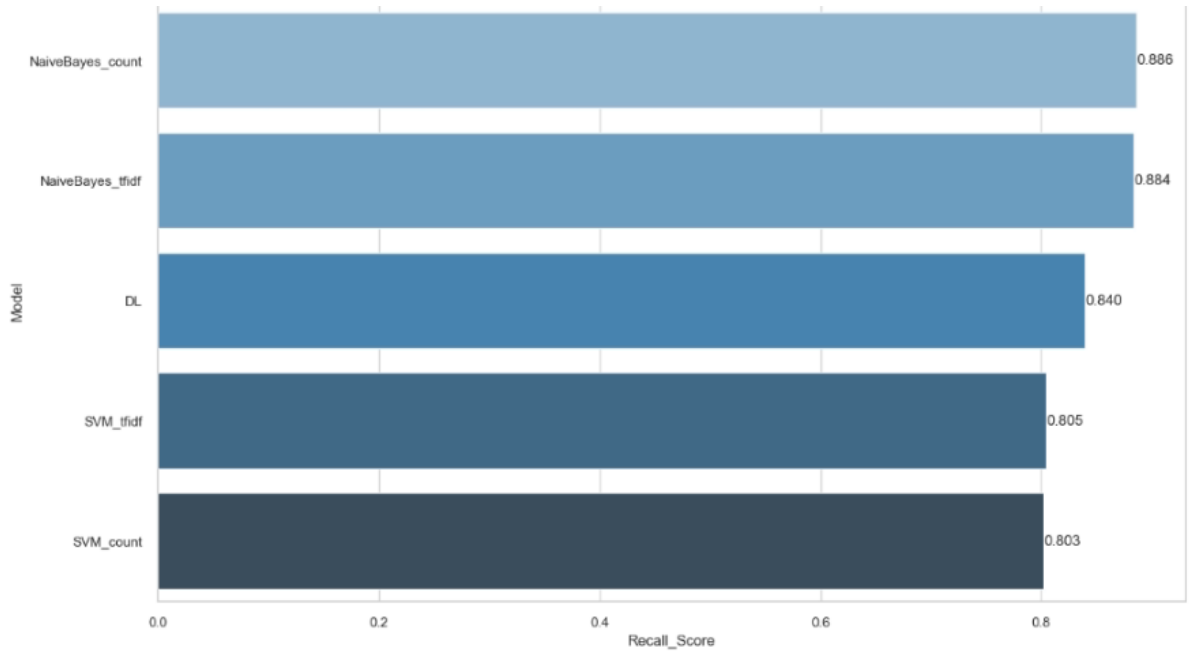
```
df_pred["Predicted_Feedbaack"] = df_pred["Pred_Proba"].apply(lambda x: "Recommended" if x>=0.5 else "Not Recommended")
```

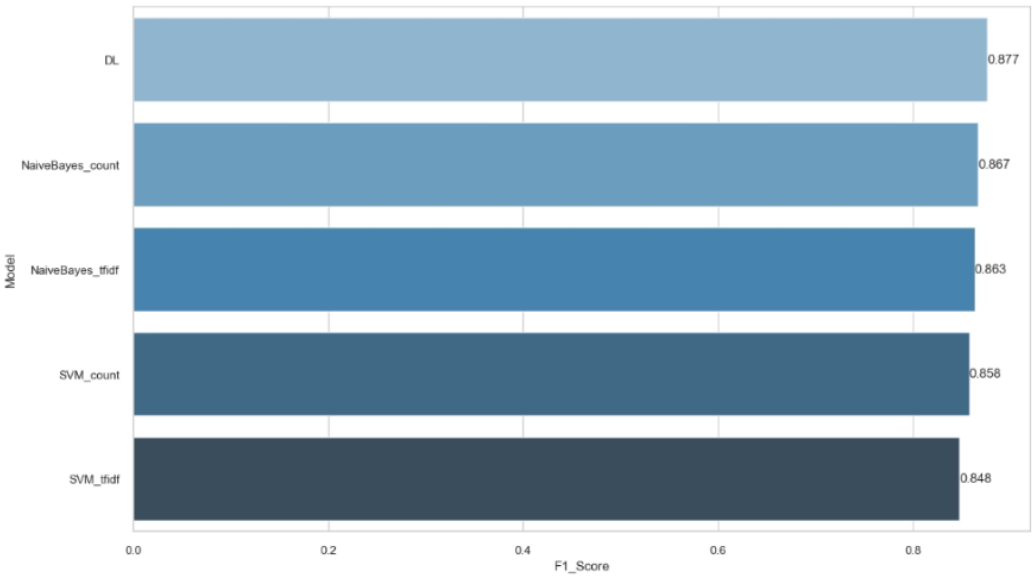
```
df_pred
```

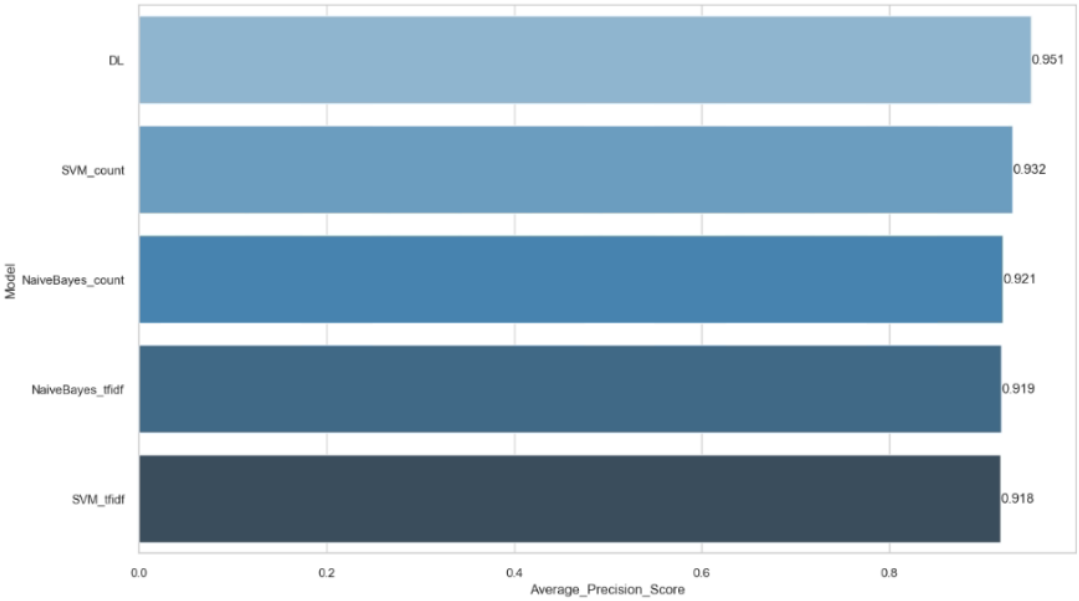
	Pred_Proba	Predicted_Feedbaack
Be sure you're ready to be coddled in the ultimate in comfort and true American luxury	0.67	Recommended
I bought my Navigator a few months ago and about a month after buying it we were in a 5 car collision ie Bauer! This vehicle saved my family	0.63	Recommended
This is a wonderful SUV very comfortable very comfortable and eliability is outstanding. I would highly recommend this vehicle.	0.82	Recommended
Ive had my Navigator for a few months now. I was leasing a 2008 GMC Yukon Denali before I decided to buy. I originally bought a 2007 Cadillac Escalade but took it back because it was too much like the Denali. The Escalade does handle better and has more power than the Nav but I do like the Navs interior a bit better. Love the 3rd row seat and the power fold feature. Denali and Escalade 3rd row was terrible. Also like that the Nav can switch from 2W to 4W drive. Denali and Escalade was AWD. Nav takes Regular Fuel while Denali and Escalade took Premium Fuel. Overall I am satisfied with the Nav though there is room to improve on it to better compete with other luxury brands.	0.83	Recommended
This car is an overall good vehicle. It has room for 7 passengers which makes it convenient for families with children. The car is very large and takes a little bit t	0.80	Recommended
I bought this 6 months ago. it took a few weeks to get the hang of driving it. now I love it. Plenty of power. Really comfortable. Tons of room for a 220 lb 6 footer like me. just drove 3400 miles in two weeks on vacation with tons of gear and bicycles on the back. made the trip very easy. 17.9 MPG on trip. about 15 commuting to work and around town.	0.84	Recommended

12) COMPARING THE MODELS

```
compare = pd.DataFrame({"Model": ["NaiveBayes_count",  "SVM_count", "NaiveBayes_tfidf", "SVM_tfidf",  "DL"],

                        "F1_Score": [nb_count_f1,  svc_count_f1, nb_tfidf_f1, svc_tfidf_f1,  DL_f1],

                        "Recall_Score": [nb_count_rec,  svc_count_rec, nb_tfidf_rec,svc_tfidf_rec, DL_rec],


                        "Average_Precision_Score": [nb_AP_count,  svc_AP_count , nb_AP_tfidf, svc_AP_tfidf, DL_AP]})

def labels(ax):

    for p in ax.patches:
        width = p.get_width()                       # get bar length
        ax.text(width,                              # set the text at 1 unit right of the bar
                p.get_y() + p.get_height() / 2,     # get Y coordinate + X coordinate / 2
                '{:1.3f}'.format(width),            # set variable to display, 2 decimals
                ha = 'left',                        # horizontal alignment
                va = 'center')                      # vertical alignment

plt.figure(figsize=(15,30))
plt.subplot(311)
compare = compare.sort_values(by="Recall_Score", ascending=False)
ax=sns.barplot(x="Recall_Score", y="Model", data=compare, palette="Blues_d")
labels(ax)

plt.subplot(312)
compare = compare.sort_values(by="F1_Score", ascending=False)
ax=sns.barplot(x="F1_Score", y="Model", data=compare, palette="Blues_d")
labels(ax)


plt.subplot(313)
compare = compare.sort_values(by="Average_Precision_Score", ascending=False)
ax=sns.barplot(x="Average_Precision_Score", y="Model", data=compare, palette="Blues_d")
labels(ax)
plt.show();
```




DL
NaiveBayes_count
NaiveBayes_tfidf
SVM_count
SVM_tfidf
Model
0.877
0.867
0.863
0.858
0.848
0.0
0.2
0.4
0.6
0.8
F1_Score


DL
SVM_count
NaiveBayes_count
NaiveBayes_tfidf
SVM_tfidf
Model
0.951
0.932
0.921
0.919
0.918
0.0
0.2
0.4
0.6
0.8
Average_Precision_Score

13) CONCLUSION

In this project, sentiment analysis has been employed to assess product recommendations. Various machine learning algorithms, Naive Bayes, Support Vector Machine (SVM), have been utilized to enhance prediction accuracy. Additionally, a deep learning algorithm has been incorporated for comparison purposes with the machine learning models. The dataset originates from Edmunds-Consumer Car Ratings and Reviews and is accessible on the Kaggle website at:https://www.kaggle.com/datasets/ankkur13/edmundsconsumer-car-ratings-andreviews/data (https://www.kaggle.com/datasets/ankkur13/edmundsconsumer-car-ratingsand-reviews/data).

In the overall comparison of models, distinguishing a clear preference among the top performing five models proves challenging due to their closely matched scores. Naïve Bayes (Count) and Naïve Bayes (TF-IDF) have the highest recall scores, indicating that they perform well in capturing positive instances. Deep Learning follows, while SVM (Count) and SVM (TF-IDF) have lower recall scores

Deep Learning has the highest F1 score, indicating a good balance between precision and recall. Naïve Bayes (Count) and Naïve Bayes (TF-IDF) follow closely, while SVM (Count) and SVM (TF-IDF) have lower F1 scores.

Also Deep Learning has the highest average precision score, indicating its ability to rank positive instances higher. SVM (Count) and Naïve Bayes (Count) follow, while Naïve Bayes (TF-IDF) and SVM (TF-IDF) have slightly lower average precision scores.

Overall Conclusion:

Best Overall Performer: Deep Learning has the highest scores in F1 and average precision, making it the best overall performer.

Naïve Bayes Strengths: Naïve Bayes performs well, particularly in recall, making it suitable when identifying positive instances is critical.

SVM Limitations: SVM, while competitive, lags slightly behind in most metrics.

Considerations:

Computational Complexity: Deep Learning models may have higher computational requirements during training. Dataset Size: The results might be influenced by the size and nature of the dataset. In summary, based on the provided metrics, Deep Learning appears to be the top-performing algorithm for sentiment analysis in this context.
However the choice of the best model also depends on specific project requirements

14) RECOMMINDATION

The sentiment analysis project compared various machine learning algorithms, including Naive Bayes, Support Vector Machine (SVM), and Deep Learning, using the Edmunds-Consumer Car Ratings and Reviews dataset. Here are the key findings and recommendations:

1. **Overall Performance:**

    - **Best Performer:** Deep Learning achieved the highest scores in F1 and average precision, making it the best overall performer.

    - **Naive Bayes Strengths:** Naive Bayes, especially in its Count and TF-IDF variants, excelled in recall, making it suitable for capturing positive instances.

2. **SVM Considerations:**

    - SVM, both in Count and TF-IDF, performed well but had slightly lower scores in recall and F1 compared to other models.

3. **Average Precision Scores:**

    - Deep Learning, SVM (Count), and Naive Bayes (Count) showed high average precision scores, with Deep Learning excelling in ranking positive instances.

4. **Project-Specific Decision:**

    - The choice of the best model depends on specific project requirements, considering factors such as computational complexity, dataset size, and project goals.

15) REFERANCES

https://www.kaggle.com/andrewmvd/heart-failure-clinical-data

https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

https://machinelearningmastery.com/handle-missing-data-python/

https://en.wikipedia.org/wiki/Dummy_variable_(statistics)

https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html