In [1]:
```
import tkinter as tk
from tkinter import ttk
import pandas as pd
from sklearn.cluster import KMeans
```

In [2]:
```
feature_entries={}
features_listbox=None
selected_features=[]
root=None
```

In [3]:
```
def assess_knowledge():
    score=0
    if true_false_answer_1_var.get():
        score+=5
    if true_false_answer_2_var.get():
        score+=1
    if true_false_answer_3_var.get():
        score+=1
    if true_false_answer_4_var.get():
        score+=2
    if true_false_answer_5_var.get():
        score+=5

    threshold_score=8

    if score >= threshold_score:
        open_interface1()
    else:
        show_interface_error_message("sorry, you need to know more about the pr
```

In [4]:
```
df=pd.read_excel("C:\\Users\\User\\Desktop\\exptdataset1.xlsx")
```

In [5]:
```
def open_interface1():
    global root,features_listbox
    #root.destroy()

    df=pd.read_excel("C:\\Users\\User\\Desktop\\exptdataset1.xlsx")

    root = tk.Tk()
    root.title("Feature Selection")

    features_listbox = tk.Listbox(root, selectmode=tk.MULTIPLE)
    for feature in df['Feature']:
        features_listbox.insert(tk.END, feature)
    features_listbox.pack(padx=20, pady=20)

    ttk.Button(root, text="Proceed", command=display_selected_features).pack()

    root.mainloop()
```

In [6]:
```
def display_selected_features():
    global selected_features_listbox,root
    #selected_features = []
    selected_features.clear()
    selected_indices=features_listbox.curselection()

    if selected_indices:
        for index in selected_indices:
            selected_features.append(features_listbox.get(index))
        root.destroy()
        open_interface()
    else:
        tk.messagebox.showwarning("No Selection", "Please select at least one f
```

In [7]:
```
def ask_true_false_questions():
    global true_false_answer_1_var,true_false_answer_2_var,true_false_answer_3_
    assessment_window = tk.Tk()
    assessment_window.title("Product Knowledge Assessment")

    # Create true/false questions
    true_false_answer_1_var = tk.BooleanVar()
    true_false_answer_2_var = tk.BooleanVar()
    true_false_answer_3_var = tk.BooleanVar()
    true_false_answer_4_var = tk.BooleanVar()
    true_false_answer_5_var = tk.BooleanVar()

    true_false_answer_1 = ttk.Checkbutton(assessment_window, text="Question 1:
    true_false_answer_1.grid(row=0, padx=200, pady=5)

    true_false_answer_2 = ttk.Checkbutton(assessment_window, text="Question 2:
    true_false_answer_2.grid(row=1, padx=200, pady=5)

    true_false_answer_3 = ttk.Checkbutton(assessment_window, text="Question 3:
    true_false_answer_3.grid(row=2, padx=200, pady=5)

    true_false_answer_4 = ttk.Checkbutton(assessment_window, text="Question 4:
    true_false_answer_4.grid(row=3, padx=200, pady=5)

    true_false_answer_5 = ttk.Checkbutton(assessment_window, text="Question 5:
    true_false_answer_5.grid(row=4, padx=200, pady=5)

    assess_button=ttk.Button(assessment_window,text="Assess Knowledge",command=
    assess_button.grid(row=5,padx=200,pady=5)

    assessment_window.mainloop()
```

In [8]:
```
def show_interface_error_message(message):
    error_window=tk.Toplevel()
    error_window.title("Error")

    error_label=ttk.Label(error_window,text=message)
    error_label.pack(padx=10,pady=5)

    ok_button=ttk.Button(error_window,text="OK",command=error_window.destroy)
    ok_button.pack(pady=5)

    error_window.mainloop()
```

```
In [9]: def open_interface():
            global root,selected_features,feature_entries
            #root.destroy()

            root=tk.Tk()
            root.title("Review and rating collection")

            frame=ttk.Frame(root)
            frame.pack(fill='both',expand=True)

            canvas=tk.Canvas(frame)
            scrollbar=ttk.Scrollbar(frame,orient='vertical',command=canvas.yview)
            scrollable_frame=ttk.Frame(canvas)

            scrollable_frame.bind(
                "<Configure>",
                lambda e: canvas.configure(
                    scrollregion=canvas.bbox("all")
                )
            )
            canvas.create_window((0,0),window=scrollable_frame,anchor="nw")
            canvas.configure(yscrollcommand=scrollbar.set)

            canvas.pack(side="left",fill="both",expand=True)
            scrollbar.pack(side="right",fill="y")

            feature_label=ttk.Label(scrollable_frame,text='Feature')
            feature_label.grid(row=0,column=0,padx=5,pady=5)

            feature_label=ttk.Label(scrollable_frame,text='Review')
            feature_label.grid(row=0,column=1,padx=5,pady=5)

            feature_label=ttk.Label(scrollable_frame,text='Rating')
            feature_label.grid(row=0,column=2,padx=5,pady=5)

            #global feature_entries
            for i,feature in enumerate(selected_features,start=1):
                ttk.Label(scrollable_frame,text=feature).grid(row=i,column=0,padx=5,pac
                feature_entries[feature]={
                    'review':ttk.Entry(scrollable_frame),
                    'rating':ttk.Entry(scrollable_frame)
                }
                feature_entries[feature]['review'].grid(row=i,column=1,padx=5,pady=5)
                feature_entries[feature]['rating'].grid(row=i,column=2,padx=5,pady=5)

            submit_button=ttk.Button(root,text="submit Reviews and Ratings",command=sub
            submit_button.pack(pady=10)

            root.mainloop()
```

In [10]:
```
def submit_reviews_ratings():
    global data,selected_features,feature_entries
    data=[]
    for feature in selected_features:
        review=feature_entries[feature]['review'].get()
        rating=feature_entries[feature]['rating'].get()
        data.append({"Feature":feature,"Review":review,"Rating":rating})
    root.destroy()
```

In [11]:
```
ask_true_false_questions()
```

In [12]:
```
for entry in data:
    print(entry)
```

```
{'Feature': 'camera', 'Review': '', 'Rating': '4'}
{'Feature': 'battery', 'Review': 'Takes lot of time to charge.. Not recommend
ed', 'Rating': ''}
{'Feature': 'screen', 'Review': '', 'Rating': '5'}
{'Feature': 'services', 'Review': '', 'Rating': '3'}
{'Feature': 'sound', 'Review': 'Good audio quality.', 'Rating': ''}
{'Feature': 'picture', 'Review': '', 'Rating': '5'}
{'Feature': 'ram', 'Review': 'High gb ram at this cost.', 'Rating': ''}
{'Feature': 'design', 'Review': '', 'Rating': '5'}
{'Feature': 'storage', 'Review': '', 'Rating': '3'}
{'Feature': 'speed', 'Review': 'High speed. Best fir general use', 'Rating':
''}
{'Feature': 'hardware', 'Review': 'Good hardware is used may not get easily d
amaged.', 'Rating': ''}
{'Feature': 'updates', 'Review': '', 'Rating': '4'}
{'Feature': 'software', 'Review': 'Better software compatible for many applic
ations.', 'Rating': ''}
```

In [13]:
```
reviews_ratings_df=pd.DataFrame(data)
```

In [14]:
```
reviews_ratings_df
```

Out[14]:

```
    Feature                                            Review  Rating
 0   camera                                                         4
 1  battery         Takes lot of time to charge.. Not recommended
 2   screen                                                         5
 3 services                                                         3
 4    sound                               Good audio quality.
 5  picture                                                         5
 6      ram                           High gb ram at this cost.
 7   design                                                         5
 8  storage                                                         3
 9    speed                     High speed. Best fir general use
10 hardware   Good hardware is used may not get easily damaged.
11  updates                                                         4
12 software        Better software compatible for many applications.
```

In [15]:
```
reviews_ratings_df.to_excel("C:\\Users\\User\\Desktop\\reviews_ratings_df.xlsx"
```

In [16]:
```
df1=pd.read_excel("C:\\Users\\User\\Desktop\\reviews_ratings_df.xlsx")
```

In [17]:
```
df1["Review"].fillna("Unknown",inplace=True)
```

In [18]:
```
df1["Rating"].fillna(0,inplace=True)
```

In [19]:
```
df1.to_excel("C:\\Users\\User\\Desktop\\reviews_ratings_df.xlsx",index=False)
```

In [20]:
```
df1
```

Out[20]:

	Feature	Review	Rating
0	camera	Unknown	4.0
1	battery	Takes lot of time to charge.. Not recommended	0.0
2	screen	Unknown	5.0
3	services	Unknown	3.0
4	sound	Good audio quality.	0.0
5	picture	Unknown	5.0
6	ram	High gb ram at this cost.	0.0
7	design	Unknown	5.0
8	storage	Unknown	3.0
9	speed	High speed. Best fir general use	0.0
10	hardware	Good hardware is used may not get easily damaged.	0.0
11	updates	Unknown	4.0
12	software	Better software compatible for many applications.	0.0

In [21]:
```
#preprocessing
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
import string
```

In [22]:
```
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[22]:
```
True
```

In [23]:
```
custom_stopwords=["the","and","is","are","in","it","on","of","this","was","a",'
```

In [24]:
```
def preprocess_text(text,custom_stopwords):
    text=text.lower()
    text=text.translate(str.maketrans('','',string.punctuation))
    tokens=word_tokenize(text)
    #stop_words=set(stopwords.words('english'))
    tokens=[word for word in tokens if word not in custom_stopwords]
    preprocessed_text=' '.join(tokens)
    return preprocessed_text
```

In [25]:
```
df1["preprocessed_review"]=df1["Review"].apply(lambda x:preprocess_text(x,custo
```

In [26]:
```
df1.to_excel("C:\\Users\\User\\Desktop\\reviews_ratings_df.xlsx",index=False)
```

In [27]:
```
df1
```

Out[27]:

	Feature	Review	Rating	preprocessed_review
0	camera	Unknown	4.0	unknown
1	battery	Takes lot of time to charge.. Not recommended	0.0	takes lot time charge not recommended
2	screen	Unknown	5.0	unknown
3	services	Unknown	3.0	unknown
4	sound	Good audio quality.	0.0	good audio quality
5	picture	Unknown	5.0	unknown
6	ram	High gb ram at this cost.	0.0	high gb ram cost
7	design	Unknown	5.0	unknown
8	storage	Unknown	3.0	unknown
9	speed	High speed. Best fir general use	0.0	high speed best fir general use
10	hardware	Good hardware is used may not get easily damaged.	0.0	good hardware used may not get easily damaged
11	updates	Unknown	4.0	unknown
12	software	Better software compatible for many applications.	0.0	better software compatible many applications

In [28]:
```
from nltk.sentiment import SentimentIntensityAnalyzer
```

In [29]:
```
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Out[29]:
```
True
```

In [30]:
```
sia=SentimentIntensityAnalyzer()
```

In [31]:
```
def get_rating(text):
    if 'unknown' in text.lower():
        return 0
    else:
        sentiment_score=sia.polarity_scores(text)['compound']
        if sentiment_score>=0.5:
            return 5
        elif sentiment_score>=0.1:
            return 4
        elif sentiment_score>=-0.1:
            return 3
        elif sentiment_score>=0.5:
            return 2
        else:
            return 1
```

In [32]:
```
df1["sentiment_rating"]=df1["preprocessed_review"].apply(get_rating)
```

In [33]:
```
df1.to_excel("C:\\Users\\User\\Desktop\\reviews_ratings_df.xlsx",index=False)
```

In [34]:
```
df1
```

Out[34]:

	Feature	Review	Rating	preprocessed_review	sentiment_rating
0	camera	Unknown	4.0	unknown	0
1	battery	Takes lot of time to charge.. Not recommended	0.0	takes lot time charge not recommended	1
2	screen	Unknown	5.0	unknown	0
3	services	Unknown	3.0	unknown	0
4	sound	Good audio quality.	0.0	good audio quality	4
5	picture	Unknown	5.0	unknown	0
6	ram	High gb ram at this cost.	0.0	high gb ram cost	3
7	design	Unknown	5.0	unknown	0
8	storage	Unknown	3.0	unknown	0
9	speed	High speed. Best fir general use	0.0	high speed best fir general use	5
10	hardware	Good hardware is used may not get easily damaged.	0.0	good hardware used may not get easily damaged	5
11	updates	Unknown	4.0	unknown	0
12	software	Better software compatible for many applications.	0.0	better software compatible many applications	4

In [35]:
```
df1["average_rating"]=df1.apply(lambda row:row["sentiment_rating"] if row["Rati
```

In [36]:
```
df1.to_excel("C:\\Users\\User\\Desktop\\reviews_ratings_df.xlsx",index=False)
```

In [37]:
```
df1
```

Out[37]:

	Feature	Review	Rating	preprocessed_review	sentiment_rating	average_rating
0	camera	Unknown	4.0	unknown	0	4.0
1	battery	Takes lot of time to charge.. Not recommended	0.0	takes lot time charge not recommended	1	1.0
2	screen	Unknown	5.0	unknown	0	5.0
3	services	Unknown	3.0	unknown	0	3.0
4	sound	Good audio quality.	0.0	good audio quality	4	4.0
5	picture	Unknown	5.0	unknown	0	5.0
6	ram	High gb ram at this cost.	0.0	high gb ram cost	3	3.0
7	design	Unknown	5.0	unknown	0	5.0
8	storage	Unknown	3.0	unknown	0	3.0
9	speed	High speed. Best fir general use	0.0	high speed best fir general use	5	5.0
10	hardware	Good hardware is used may not get easily damaged.	0.0	good hardware used may not get easily damaged	5	5.0
11	updates	Unknown	4.0	unknown	0	4.0
12	software	Better software compatible for many applications.	0.0	better software compatible many applications	4	4.0

In [36]:
```
#categorizing the features using k-means
#weights=df1['average_rating'].apply(lambda x:2 if x in [4,5] else (1 if x==3 e
```

In [38]:
```
X=df1[['average_rating']].values
```

In [39]:
```
Kmeans=KMeans(n_clusters=3,random_state=42)
```

In [40]:
```
Kmeans.fit(X)
```

```
E:\Python\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: T
he default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
E:\Python\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarning: KMe
ans is known to have a memory leak on Windows with MKL, when there are less c
hunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[40]:
```
KMeans(n_clusters=3, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [41]:
```
cluster_labels=Kmeans.labels_
```

In [42]:
```
recommendation_map={
    0:"moderate",
    1:"weak",
    2:"strong"
}
```

In [43]:
```
df1["Recommendation"]=[recommendation_map[label] for label in cluster_labels]
```

In [44]:
```
df1
```

Out[44]:

	Feature	Review	Rating	preprocessed_review	sentiment_rating	average_rating	Recor
0	camera	Unknown	4.0	unknown	0	4.0	
1	battery	Takes lot of time to charge.. Not recommended	0.0	takes lot time charge not recommended	1	1.0	
2	screen	Unknown	5.0	unknown	0	5.0	
3	services	Unknown	3.0	unknown	0	3.0	
4	sound	Good audio quality.	0.0	good audio quality	4	4.0	
5	picture	Unknown	5.0	unknown	0	5.0	
6	ram	High gb ram at this cost.	0.0	high gb ram cost	3	3.0	
7	design	Unknown	5.0	unknown	0	5.0	
8	storage	Unknown	3.0	unknown	0	3.0	
9	speed	High speed. Best fir general use	0.0	high speed best fir general use	5	5.0	
10	hardware	Good hardware is used may not get easily damaged.	0.0	good hardware used may not get easily damaged	5	5.0	
11	updates	Unknown	4.0	unknown	0	4.0	
12	software	Better software compatible for many applications.	0.0	better software compatible many applications	4	4.0	

In [45]:
```
df1.to_excel("C:\\Users\\User\\Desktop\\reviews_ratings_df.xlsx",index=False)
```

In [46]:
```
from sklearn.metrics import silhouette_score
```

In [47]:
```
silhouette_avg=silhouette_score(X,cluster_labels)
print("silhouette score:",silhouette_avg)
```

```
silhouette score: 0.6923076923076923
```

In [48]:
```
strong_count=(df1['Recommendation']=='strong').sum()
moderate_count=(df1['Recommendation']=='moderate').sum()
```

In [49]:
```
if (strong_count+moderate_count) > len(df1)/2:
    print("strongly recommended")
else:
    print("Weakly recommended")
```

```
strongly recommended
```