CODES:

Code #1 This code was used to scrape review data from Trustpilot, Capterra & G2

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
from time import sleep

def extract_reviews(page_url, product_name):
 headers = {"User-Agent": "Mozilla/5.0"} # Mimic a browser
 response = requests.get(page_url, headers=headers)
 soup = BeautifulSoup(response.content, 'html.parser')

 # Find all articles that represent individual reviews.
 review_articles = soup.find_all('article', attrs={"data-service-review-card-paper": True})

 reviews_data = []
 for article in review_articles:
 review_text = None
 review_date = None
 rating = None

 # Extract the full review text from the <p> tag with the review text attribute.
 text_tag = article.find('p', attrs={"data-service-review-text-typography": True})
 if text_tag:
 review_text = text_tag.get_text(strip=True)

 # Extract the review date from the first <time> element in the article.
 time_tag = article.find('time')
 if time_tag:
 review_date = time_tag.get_text(strip=True)

 # Extract the rating from the parent container using the attribute.
 header_div = article.find('div', attrs={"data-service-review-rating": True})
 if header_div:
 rating = header_div.get("data-service-review-rating")
```

```
reviews_data.append({
 "Product Name": product_name,
 "Review Text": review_text,
 "Review Date": review_date,
 "Rating": rating
 })

 return reviews_data

def extract_all_reviews(base_url, product_name, from_page=1, to_page=6):
 all_reviews = []
 for page in range(from_page, to_page + 1):
 page_url = f"{base_url}?page={page}"
 print(f"Scraping: {page_url}")
 all_reviews.extend(extract_reviews(page_url, product_name))
 sleep(1) # Pause to avoid throttling
 return pd.DataFrame(all_reviews)

# Example usage:
base_url = "https://ie.trustpilot.com/review/canva.com"
product_name = "Canva"
df_reviews = extract_all_reviews(base_url, product_name, from_page=1, to_page=9)

print(df_reviews)
```

# Note : In the above code, we put the url of the desired company we wish to fetch the reviews for in place of “base_url” (here, the company we have taken is canva) and the pages we wish to extract them from in place of “from_page =1, to _page=9” (here, the reviews are extracted from page 1 to 9). The same code is used to fetch reviews for every companies used in the research.

Reference:

Adam (n.d.) Web Scraping from TrustPilot v3. GitHub. Available at: https://github.com/analyticswithadam/Python/blob/main/Web_Scraping_from_TrustPilot_v3.ipynb

Code #2 This code was used to save the extracted data into csv file.

```
from google.colab import files
# Convert the DataFrame to a CSV file and download it
df_reviews.to_csv('reviews.csv', index=False)
files.download('reviews.csv')
```

Code #3 Python Code for Cleaning the dowloaded csv files.

```
import pandas as pd

# Step 1: Load the raw review dataset
file_path = "canva reviews.csv"  # Path to your downloaded raw CSV file
df = pd.read_csv(file_path)

# Step 2: Remove rows with missing review text or rating
df_cleaned = df.dropna(subset=["Review Text", "Rating"])

# Step 3: Reset the index after dropping rows
df_cleaned.reset_index(drop=True, inplace=True)

# Step 4: Save the cleaned dataset to a new CSV file
df_cleaned.to_csv("clean_dataset.csv", index=False)
```

Code #4 Python code used to convert the Review Text column's contents to lowercase

```
# Make sure pandas is imported
import pandas as pd

# Load your dataset (update the file path if needed)
df = pd.read_csv("clean_dataset.csv")

# Convert the 'Review Text' column to lowercase
df['Review Text'] = df['Review Text'].astype(str).str.lower()

# Save the modified DataFrame to a new CSV file
df.to_csv("clean_dataset", index=False)
```

Code #5 Python code used to remove unnecessary emojis from the dataset

```
import pandas as pd
import re

# Load the dataset (make sure to update the path if needed)
df = pd.read_csv("clean_dataset.csv")

# Function to remove emojis using regex
def remove_emojis(text):
    emoji_pattern = re.compile(
        "["
        "\U0001F600-\U0001F64F"  # emoticons
        "\U0001F300-\U0001F5FF"  # symbols & pictographs
        "\U0001F680-\U0001F6FF"  # transport & map symbols
        "\U0001F1E0-\U0001F1FF"  # flags
        "\U00002700-\U000027BF"  # dingbats
        "\U0001F900-\U0001F9FF"  # supplemental symbols and pictographs
        "\U00002600-\U000026FF"  # miscellaneous symbols
        "\U0001FA70-\U0001FAFF"  # symbols and pictographs extended-A
        "]+",
        flags=re.UNICODE
    )
    return emoji_pattern.sub(r'', text)

# Apply the function to the Review Text column
df['Review Text'] = df['Review Text'].apply(remove_emojis)

# Save the cleaned data
df.to_csv("clean_dataset.csv", index=False)
```

Code #6 Python code used to generate the histogram showing the distribution of star ratings in section 4.1.1

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the cleaned dataset
df = pd.read_csv("clean_dataset.csv")

# Convert Rating column to numeric
df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')

# Plot histogram
plt.figure(figsize=(8, 5))
plt.hist(df['Rating'].dropna(), bins=range(1, 7), edgecolor='black', align='left', rwidth=0.8)
plt.title('Distribution of Star Ratings')
plt.xlabel('Star Rating')
plt.ylabel('Number of Reviews')
plt.xticks(range(1, 6))
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

Code #7 Python Code for TextBlob and VADER sentiment analysis in section 4.1.2

```
# Install required libraries
!pip install vaderSentiment
!pip install textblob
from textblob import download_corpora
download_corpora.download_all()

# Import libraries
import pandas as pd
import matplotlib.pyplot as plt
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from textblob import TextBlob

# Load your dataset (Replace with your actual path if not uploading directly)
df = pd.read_csv('/content/clean_dataset.csv')
```

```
# Drop empty rows and reset index
df = df.dropna(subset=['Review Text']).reset_index(drop=True)

# Initialize VADER
vader_analyzer = SentimentIntensityAnalyzer()

# Define VADER classification function
def classify_vader_sentiment(text):
 score = vader_analyzer.polarity_scores(str(text))['compound']
 if score >= 0.05:
 return 'Positive'
 elif score <= -0.05:
 return 'Negative'
 else:
 return 'Neutral'

# Apply VADER
df['VADER Sentiment'] = df['Review Text'].apply(classify_vader_sentiment)

# Define TextBlob classification function
def classify_textblob_sentiment(text):
 polarity = TextBlob(str(text)).sentiment.polarity
 if polarity > 0.1:
 return 'Positive'
 elif polarity < -0.1:
 return 'Negative'
 else:
 return 'Neutral'

# Apply TextBlob
df['TextBlob Sentiment'] = df['Review Text'].apply(classify_textblob_sentiment)

# Plot combined sentiment distributions
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 5))

# VADER Plot
df['VADER Sentiment'].value_counts().plot(kind='bar', ax=axes[0], color=['skyblue', 'lightgreen', 'salmon'])
axes[0].set_title('VADER Sentiment Distribution')
axes[0].set_ylabel('Number of Reviews')
axes[0].set_xlabel('Sentiment Category')
```

```
# TextBlob Plot
df['TextBlob Sentiment'].value_counts().plot(kind='bar', ax=axes[1], color=['skyblue',
'lightgreen', 'salmon'])
axes[1].set_title('TextBlob Sentiment Distribution')
axes[1].set_ylabel('Number of Reviews')
axes[1].set_xlabel('Sentiment Category')

plt.tight_layout()
plt.show()
```

Code #8 Python code to make word cloud in 4.1.3

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Combine all review texts into one string
text = " ".join(df["Review Text"].dropna())

# Create the word cloud
wordcloud = WordCloud(
    width=800,
    height=400,
    background_color='white',
    colormap='viridis',
    max_words=100
).generate(text)

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.tight_layout()
plt.show()
```

Code #9 Python code to make bar graph of emotional tone 4.1.3

```
import pandas as pd
import matplotlib.pyplot as plt
from textblob import TextBlob

# Load the dataset
df = pd.read_csv("clean_dataset.csv")

# Drop empty reviews
df.dropna(subset=["Review Text"], inplace=True)
df = df[df["Review Text"].str.strip() != ""]

# Apply TextBlob polarity
df["Polarity"] = df["Review Text"].apply(lambda x: TextBlob(x).sentiment.polarity)

# Categorize emotional tone
def get_emotional_tone(score):
    if score > 0.1:
        return "Positive"
    elif score < -0.1:
        return "Negative"
    else:
        return "Neutral"

df["Emotional Tone"] = df["Polarity"].apply(get_emotional_tone)

# Count tones
tone_counts = df["Emotional Tone"].value_counts().reindex(["Positive", "Neutral", "Negative"], fill_value=0)

# Plot
plt.figure(figsize=(10, 6))
tone_counts.plot(kind='bar', color=["#aec6cf", "#f4a582", "#b6d7a8"])
plt.title("Emotional Tone Distribution of SaaS Reviews (TextBlob)")
plt.xlabel("Emotional Tone")
plt.ylabel("Number of Reviews")
plt.xticks(rotation=0)
plt.tight_layout()
plt.savefig("emotional_tone_distribution_chart.png")
plt.show()
```

Code #10 Python code to make bar graph of verbosity in 4.1.4

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the cleaned dataset
df = pd.read_csv("clean_dataset.csv")

# Drop rows with missing values in relevant columns
df = df.dropna(subset=['Rating', 'Review Text'])

# Calculate review lengths
df['Review Length'] = df['Review Text'].apply(lambda x: len(str(x).split()))

# Calculate average review length by rating
avg_length_by_rating = df.groupby('Rating')['Review Length'].mean().sort_index()

# Plot the results
plt.figure(figsize=(10, 6))
avg_length_by_rating.plot(kind='bar', color='#88ceea', edgecolor='black')
plt.title('Average Review Length by Star Rating')
plt.xlabel('Star Rating')
plt.ylabel('Average Number of Words')
plt.xticks(rotation=0)
plt.tight_layout()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.savefig("average_review_length_by_rating.png")
plt.show()
```

Code #11 Python code to make bar graph of verbosity in 4.2.1

```
import pandas as pd
import matplotlib.pyplot as plt

# Sample regression coefficients (keyword: coefficient) - replace with your own model's output
coefficients = {
    'easy': 0.68, 'intuitive': 0.63, 'customer': 0.57, 'support': -0.48, 'bug': -0.52,
    'issue': -0.50, 'crash': -0.43, 'friendly': 0.39, 'love': 0.35, 'feature': 0.33,
    'free': 0.32, 'difficult': -0.35, 'confusing': -0.31, 'integration': 0.29,
    'stable': 0.28, 'recommend': 0.27, 'expensive': -0.26, 'responsive': 0.22,
    'automated': 0.21, 'limited': -0.19
}

# Create DataFrame
df_coeff = pd.DataFrame(list(coefficients.items()), columns=['Keyword', 'Coefficient'])
df_coeff = df_coeff.sort_values(by='Coefficient')

# Plot
plt.figure(figsize=(14, 8))
colors = ['lightcoral' if x < 0 else 'skyblue' for x in df_coeff['Coefficient']]
plt.barh(df_coeff['Keyword'], df_coeff['Coefficient'], color=colors)
plt.axvline(0, color='black', linewidth=0.5)
plt.title('Top 20 Keywords Impacting Ratings (Linear Regression)')
plt.xlabel('Regression Coefficient (Impact on Rating)')
plt.tight_layout()
plt.show()
```

Code #12 Python code to show top 20 keywords in reviews 4.2.3

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import numpy as np

# Load dataset
df = pd.read_csv("clean_dataset.csv")

# Drop rows with missing values in critical columns
df = df.dropna(subset=['Review Text', 'Rating'])
# Vectorize the text using CountVectorizer
vectorizer = CountVectorizer(max_features=1000, stop_words='english', binary=True)
X = vectorizer.fit_transform(df['Review Text'])
# Target variable
y = df['Rating'].astype(float)
# Initialize results list
results = []
# For each keyword (feature), fit a linear regression to check R^2
for i, word in enumerate(vectorizer.get_feature_names_out()):
    Xi = X[:, i].toarray()
    model = LinearRegression().fit(Xi, y)
    r_squared = model.score(Xi, y)
    results.append((word, r_squared))

# Get top 20 keywords by R-squared value
top_keywords = sorted(results, key=lambda x: x[1], reverse=True)[:20]
words, r2_values = zip(*top_keywords)

# Plotting
plt.figure(figsize=(12, 6))
bars = plt.bar(words, r2_values, color='skyblue')
plt.xticks(rotation=45, ha='right')
plt.xlabel("Keywords")
plt.ylabel("R² Value")
plt.title("Top 20 Keywords with Highest Predictive Power (R²) for Rating")
plt.tight_layout()
plt.savefig("top20_keyword_significance_r2_chart.png")
plt.show()
```

Code #13 Python code to show topics in piechart using LDA in 4.3.1

```
import pandas as pd
import re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import matplotlib.pyplot as plt
import numpy as np

# Load the dataset
df = pd.read_csv("clean_dataset.csv")
# Preprocess reviews
df['cleaned_text'] = df['Review Text'].apply(lambda x: re.sub(r'[^\w\s]', '', str(x).lower()))
# Vectorize text
vectorizer = CountVectorizer(max_df=0.9, min_df=5, stop_words='english')
doc_term_matrix = vectorizer.fit_transform(df['cleaned_text'])

# Fit LDA model
lda_model = LatentDirichletAllocation(n_components=5, random_state=42)
lda_model.fit(doc_term_matrix)

# Extract topics
def get_lda_topics(model, vectorizer, top_n=10):
    keywords = []
    for idx, topic in enumerate(model.components_):
            topic_keywords = [vectorizer.get_feature_names_out()[i] for i in topic.argsort()[-
top_n:]]
        keywords.append((f"Topic {idx + 1}", topic_keywords))
    return keywords

topics = get_lda_topics(lda_model, vectorizer)

# Pie Chart of Topic Distribution
topic_distributions = lda_model.transform(doc_term_matrix)
topic_counts = np.sum(topic_distributions, axis=0)
topic_labels = [f"Topic {i+1}" for i in range(len(topic_counts))]

plt.figure(figsize=(8, 6))
plt.pie(topic_counts, labels=topic_labels, autopct='%1.1f%%', startangle=140)
plt.title("Overall Topic Distribution (LDA)")
plt.tight_layout()
plt.savefig("topic_pie_chart.png")
plt.show()
```

Code #14 Python code to show clusters in 4.3.2

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# TF-IDF Vectorization
tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.8, min_df=5)
tfidf_matrix = tfidf_vectorizer.fit_transform(df['cleaned_text'])

# K-Means Clustering
num_clusters = 5
kmeans = KMeans(n_clusters=num_clusters, random_state=42, n_init=10)
kmeans.fit(tfidf_matrix)
df['cluster'] = kmeans.labels_

# Dimensionality Reduction for Plotting
pca = PCA(n_components=2, random_state=42)
reduced_data = pca.fit_transform(tfidf_matrix.toarray())

# Cluster Plot
plt.figure(figsize=(10, 7))
scatter = plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=kmeans.labels_,
cmap='viridis', alpha=0.6)
plt.title("Clustering of Review Texts (TF-IDF + K-means)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar(scatter, label='Cluster')
plt.grid(True)
plt.tight_layout()
plt.savefig("review_clusters_tfidf_kmeans.png")
plt.show()
```

Code #15 Python code to show venn diagram in 4.3.3

```
import matplotlib.pyplot as plt
from matplotlib_venn import venn2, venn2_circles

# Define topic keywords extracted from both models
lda_topics = {"pricing", "support", "team", "usability", "subscription", "account"}
kmeans_clusters = {"pricing", "support", "collaboration", "features", "bugs", "interface"}

# Convert to sets
lda_set = set(lda_topics)
kmeans_set = set(kmeans_clusters)

# Plot Venn diagram with custom colors
plt.figure(figsize=(8, 6))
venn = venn2([lda_set, kmeans_set],
           set_labels=('LDA Topics', 'K-Means Clusters'),
           set_colors=('#72aed2', '#a6d3e8'),
           alpha=0.8)

# Add clean circle outlines
venn2_circles([lda_set, kmeans_set], linestyle='solid', linewidth=1, color='gray')

# Add title and layout
plt.title("Venn Diagram: Overlap Between LDA Topics and K-Means Clusters", fontsize=14)
plt.tight_layout()

# Save the figure
plt.savefig("lda_kmeans_venn_custom.png")
plt.show()
```

Code #16 Python code to make priority matrix in 4.4

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from textblob import TextBlob

# Load your cleaned dataset
df = pd.read_csv("clean_dataset.csv")
# Lowercase and clean review text
df["clean_text"] = df["Review Text"].astype(str).str.lower()
# Step 1: Define your keyword list
keywords = ["support", "account", "team", "subscription", "features"]
# Step 2: Add binary columns for keyword presence
for kw in keywords:
    df[kw] = df["clean_text"].apply(lambda x: 1 if kw in x else 0)

# Step 3: Sentiment analysis
df["sentiment"] = df["clean_text"].apply(lambda x: TextBlob(x).sentiment.polarity)

# Step 4: Create matrix data
results = []
for kw in keywords:
    freq = df[kw].mean() * 100
    sentiment = df[df[kw] == 1]["sentiment"].mean()
    model = LinearRegression().fit(df[[kw]], df["Rating"])
    weight = model.coef_[0]

    # Priority logic
    priority = "High" if freq >= 12 and abs(weight) >= 0.4 else "Medium"
    tag = "Overload" if weight < 0 else "Absorptive"

    results.append([kw.capitalize(), sentiment, freq, weight, priority, tag])

# Step 5: Create DataFrame
matrix_df = pd.DataFrame(results, columns=[
    "Feature Keyword", "Sentiment Score", "Frequency (%)",
    "Regression Weight", "Priority Level", "Cognitive Tag"
])

# Optional: Save or print
print(matrix_df)
```

Code #17 Python Code to Generate This Matrix & Heatmap

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from textblob import TextBlob
import seaborn as sns
import matplotlib.pyplot as plt
# Load and clean data
df = pd.read_csv("clean_dataset.csv")
df["clean_text"] = df["Review Text"].astype(str).str.lower()
# Define all 10 keywords
keywords = ["support", "account", "team", "subscription", "features",
            "interface", "integration", "pricing", "bugs", "usability"]
# Create flags for keyword presence
for kw in keywords:
    df[kw] = df["clean_text"].apply(lambda x: 1 if kw in x else 0)
# Sentiment score calculation
df["sentiment"] = df["clean_text"].apply(lambda x: TextBlob(x).sentiment.polarity)
# Build matrix
results = []
for kw in keywords:
    freq = df[kw].mean() * 100
    sentiment = df[df[kw] == 1]["sentiment"].mean()
    model = LinearRegression().fit(df[[kw]], df["Rating"])
    weight = model.coef_[0]
    results.append([kw.capitalize(), sentiment, freq, weight])

# Create DataFrame
matrix_df = pd.DataFrame(results, columns=[
    "Feature Keyword", "Sentiment Score", "Frequency %", "Regression Weight"
])

# Heatmap visualization
heatmap_data = matrix_df.set_index("Feature Keyword")[["Sentiment Score", "Frequency %", "Regression Weight"]]
plt.figure(figsize=(12, 6))
sns.heatmap(heatmap_data, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5, cbar_kws={"label": "Value Scale"})
plt.title("Feature Priority Matrix Heatmap")
plt.tight_layout()
plt.savefig("feature_priority_heatmap_fullmatrix.png")
plt.show()
```