(name of your school)

A I P R O J E C T R E P O R T

on

“WHAT DO PASSENGERS FEEL?”

Natural Language Processing · SDG 11: Sustainable Cities and Communities

Class X

Artificial Intelligence — Subject Code 417
Part C: Practical and Project Work

Session _____

Submitted by:

Name _____

Roll No. _____

Under the guidance of:

_____

Teacher (Artificial Intelligence)

CERTIFICATE

This is to certify that _____ of Class X, Roll No. _____, has satisfactorily completed the Artificial Intelligence project entitled **“What Do Passengers Feel?”** during the academic session _____, under my guidance and supervision.

The work has been carried out as the Project Work of Part C of Artificial Intelligence (Subject Code 417), in accordance with the curriculum prescribed by the Central Board of Secondary Education.

_____ _____

Teacher-in-charge External Examiner

_____

Principal

Date: _____

ACKNOWLEDGEMENT

I would like to thank _____, my Artificial Intelligence teacher, for the guidance and encouragement given throughout this project.

I am grateful to the Principal of _____ for the computer lab and the facilities that made this work possible.

My thanks go to _____, who helped me understand the problem this project tries to solve and how it is handled today.

I thank my classmates, who tried the model and told me where it went wrong, and my family for their support and patience while the project was built.

_____

(Candidate)

INDEX

SL. NO.	TOPIC	PAGE NO.
1	Introduction	5
2	Problem Scoping	5
3	Data Acquisition	6
4	Data Exploration	7
5	Modelling	8
6	Evaluation	8
7	Deployment	11
8	Ethical Concerns	11
9	The No-Code Version	11
10	Technical Details	12
11	Source Code	12
12	Sample Output	18
13	Testing	19
14	Results and Future Scope	20
15	Conclusion	20
16	References	21

> Page numbers refer to the pages of this report as printed. The cover page, certificate, acknowledgement and this index are not numbered.

INTRODUCTION

City bus services ask passengers for feedback, and passengers write it — on a website, an app, a WhatsApp number. Hundreds of messages arrive every week.

Somebody has to read them all to find the complaints that need action: a rude driver, a bus that skips a stop, a broken door. In practice nobody has the time, so complaints sit unread, and passengers stop writing.

Sentiment analysis is the sample NLP project in the CBSE curriculum. This version reads bus feedback and decides whether each message is positive or negative, so that the negative ones can go straight to the complaints desk.

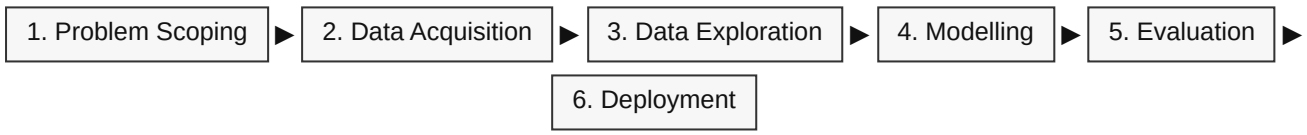
Who would use it: A city transport office receiving hundreds of feedback messages a week, and the passengers who write them.

Link to the Sustainable Development Goals

SDG 11 — Sustainable Cities and Communities. SDG target 11.2 asks for safe, affordable and accessible public transport for everyone. Buses only get better if the people running them hear what passengers say. A feedback form nobody reads changes nothing; a system that sorts the complaints from the praise, so the complaints reach someone quickly, does.

The AI Project Cycle

This is a **Natural Language Processing** project. It follows the six stages of the AI Project Cycle, and the report that follows takes them in order, one section each.



PROBLEM SCOPING

The problem was scoped with the **4Ws Problem Canvas**: who has the problem, what it is, where it happens, and why solving it matters.

Who? STAKEHOLDERS • The city transport office • Passengers who use the buses every day	**What?** THE PROBLEM • Hundreds of feedback messages, too many to read • Complaints are not noticed, so problems are not fixed
Where? CONTEXT • A city bus service with an online feedback form • Every day, as messages arrive	**Why?** BENEFIT • Complaints reach the right person quickly • Passengers see their feedback makes a difference, and keep using the buses

Problem statement

> Our **the city transport office** have a problem of **finding the complaints among hundreds of feedback messages** when **messages arrive faster than anyone can read them**. An ideal solution would **sort every message into positive or negative, so complaints go straight to the people who can act on them**.

How it is done today

Reading every message. Accurate, but takes hours every week, so in practice it does not happen.

Star ratings. Quick to count, but a star rating does not say what went wrong, and many people write without giving one.

Keyword search. Searching for words like "late" or "rude". Better than nothing, but "never late" contains "late" — this project shows how often that kind of mistake happens.

Objectives

1. To read passenger feedback from a CSV file
2. To remove repeated and empty messages
3. To normalise the text: common case, letters only, tokens, stop words removed, words stemmed
4. To build a Bag of Words and find the most common words in positive and negative messages
5. To compare a lexicon (word lists written by a person) with word scores learned from the data
6. To send new negative messages to the complaints desk

DATA ACQUISITION

Each row is one message from the feedback form: an ID, the message as the passenger typed it, and whether it is positive or negative. The label was decided by a person reading each message.

The file has 422 rows.

The messages were put together by a program from phrases a passenger might write — mixed case, route numbers, punctuation and emoticons included — so that no real passenger's words are used. For your own project, collect feedback about your school canteen or transport with a short anonymous form, label each response yourself, and credit any public dataset you use instead.

> **About this dataset.** The data used here is a LambdaLab sample, made by a program to behave like the real thing so that the project could be run and tested. It is not real data. If you submit this project, replace it with data you collected or a public dataset, and say here where it came from and that you had permission to use it.

Features and label

The features are what the model is given; the label is the answer it must learn to give.

COLUMN	TYPE	WHAT IT HOLDS	ROLE
`feedback_id`	text	An ID for the form — no names	not used
`message`	text	What the passenger wrote — turned into tokens, then a Bag of Words	feature
`sentiment`	text	positive or negative — what the model must predict	label

The file bus_feedback.csv

Passenger feedback with a positive or negative label. One form was submitted twice and one was sent empty. The whole file is **422 rows, 34.6 KB**.

The first six rows of bus_feedback.csv

FEEDBACK_ID	MESSAGE	SENTIMENT
F7180	TODAY THE AC WAS NOT WORKING ALSO NOBODY ANSWERED THE HELPLINE.	negative
F7273	The new buses are great also the bus was late again and the bus was clean	positive
F7244	TODAY THE BUS BROKE DOWN	negative
F7343	No complaints at all but the staff were friendly	positive
F7074	ROUTE 45: THE CONDUCTOR SHOUTED AT AN OLD MAN. THE DRIVER USED HIS PHONE AND THE SEATS WERE DIRTY	negative
F7343	No complaints at all but the staff were friendly	positive

DATA EXPLORATION

Before any model was built, the data was checked for problems and then drawn, to find the patterns a model could use.

Problems found in the data

WHAT WAS WRONG	HOW MUCH	WHAT WAS DONE
A form submitted twice	1 (removed)	drop_duplicates() removes a row whose ID and message are both repeated. Two different passengers can write the same words, so a repeated message with a different ID is kept.
An empty form	1 (removed)	Removed — there is nothing to analyse.
Capitals, punctuation, numbers	"BUS 118 - THE DRIVER WAS RUDE"	Text normalisation: everything is changed to lower case, and anything that is not a letter becomes a space. "RUDE", "Rude" and "rude!" all become the token rude.
Words that carry no feeling	the, was, and, it...	Stop words are removed. But not and never are deliberately kept: "not clean" means the opposite of "clean".
Different forms of one word	waited, waiting, waits	A simple stemmer cuts the endings -ing, -ed and -s from longer words, so all three become wait. Like real stemming, it sometimes makes non-words — skipped becomes skipp — which is fine, as long as it is done the same way every time.

Figure 1: The most common words in each kind of message

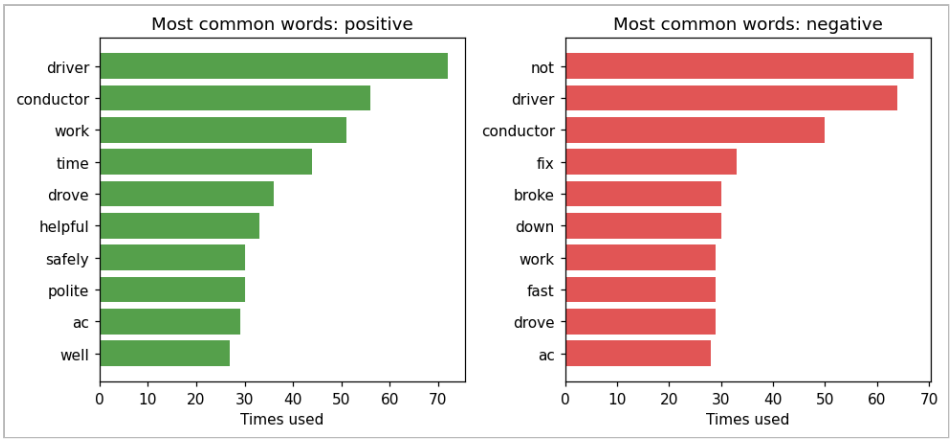


Figure 1 — The most common words in each kind of message, drawn by the program as a horizontal bar chart and saved as chart1_common_words.png.

How to read it. After normalisation, the ten tokens used most often in positive messages (left) and in negative messages (right).

Some words are near the top of both lists — driver, conductor, work, drove and ac — because people praise and complain about the same things. Those words alone cannot say which way a message leans. The useful words are the ones on one side only: helpful, safely and polite on the left; not, fix, broke and fast on the right. Notice that not is the most common word in complaints, which is why it must never be thrown away as a stop word.

MODELLING

AI model ▶ Learning based ▶ Supervised learning ▶ Classification

The answer is positive or negative, so this is classification. Each training message has a label written by a person, so it is supervised learning. The input is text, so it is Natural Language Processing.

How the model works

The messages are split into 336 for training and 84 for testing.

Model A, the lexicon, is rule based. A person wrote a list of good words (polite, clean, smooth...) and bad words (late, rude, crowd...). A message scores +1 for each good word and −1 for each bad word, and is positive if the total is 0 or more. This is how the Sentiment Analysis widget in Orange works, with much longer lists.

Model B is learning based. For every word, it counts how often the word appears in positive and in negative training messages, and gives it a score from −1 to +1. A new message's score is the total of its words' scores. Nobody writes a list: the data decides which words matter, and how much.

EVALUATION

Both models were tested on the same 84 messages. The lexicon's accuracy was 85.7 %; the learned scores reached 96.4 %.

For a transport office the key number is how many complaints are caught, because a missed complaint is a problem nobody fixes. That is the recall for negative messages: 82.2 % for the lexicon against 95.6 % for the learned scores.

The program prints some messages the lexicon got wrong, and they show why. "It was not crowded" contains crowd, so the list calls it negative. "The driver used his phone, nobody answered the helpline" contains no word on either list, so it scores 0 and is called positive. Word lists cannot handle words they were not given, or words that change meaning beside not and never. Learning from the data handles many of these cases — though not all, because a Bag of Words still looks at words one at a time.

Figure 2: What the model learned about each word

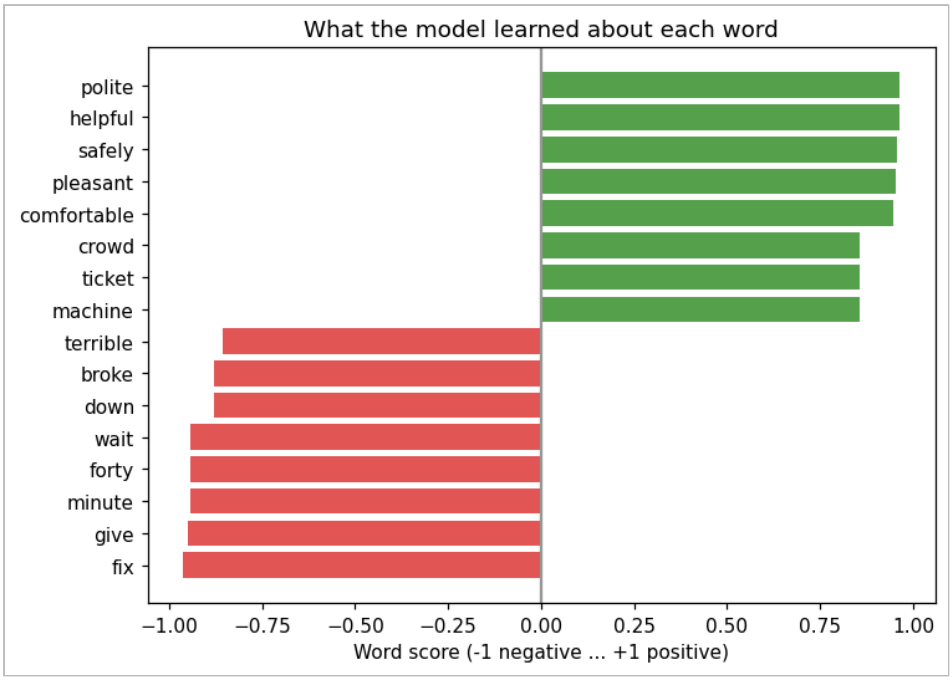


Figure 2 — What the model learned about each word, drawn by the program as a horizontal bar chart and saved as chart2_word_scores.png.

How to read it. The learned score of the most negative and most positive words, from −1 (only ever in complaints) to +1 (only ever in praise).

The model found its own word lists from the training messages. Some are obvious — polite, helpful, pleasant are positive. Others are things a person writing a list would not think of: "fix" (from "please fix this") is one of the strongest signs of a complaint.

One is surprising: crowd is learned as positive. In this data it only appears in "not crowded" — complaints say "overcrowded", which stems to a different word. A person's word list calls crowd bad; the data says otherwise.

Figure 3: Confusion matrix of the learned model

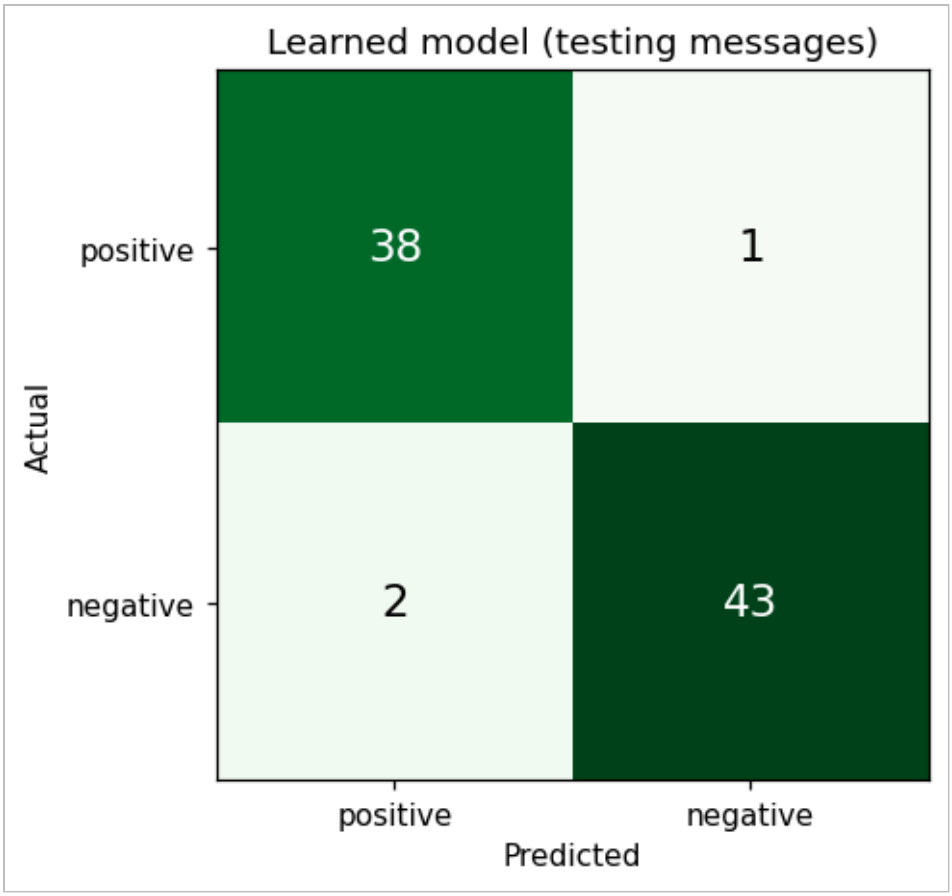


Figure 3 — Confusion matrix of the learned model, drawn by the program as a confusion matrix and saved as chart3_matrix.png.

How to read it. Rows are the real sentiment, columns what the learned model said. The diagonal holds correct answers.

Almost everything is on the diagonal. The few mistakes are messages that mix praise and complaint, where even a person might hesitate.

Figure 4: Word lists against learned word scores

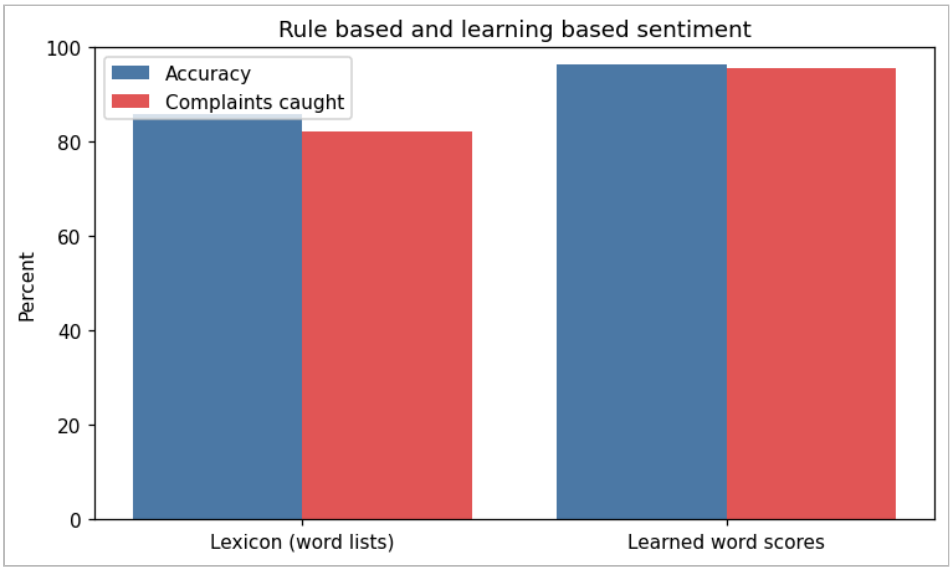


Figure 4 — Word lists against learned word scores, drawn by the program as a bar chart and saved as chart4_models.png.

How to read it. For each model: its accuracy, and the share of negative messages it caught.

The lexicon reached 85.7 % and caught 82.2 % of complaints. The learned scores reached 96.4 % and caught 95.6 %.

DEPLOYMENT

The program scores four new messages. Negative ones are marked "send to the complaints desk". Notice the first: "never late... driver very polite". The lexicon would count late as bad and polite as good, reach 0, and call it positive only because a tie counts as positive. The learned model gives it a clearly positive score.

In real use the program would run each time a message arrives. Complaints would go to a person straight away, and once a week the office would see a count of complaints by route.

ETHICAL CONCERNS

A model that works is not finished until it has been checked for fairness (bias), for whether it can explain itself (transparency), and for who answers for its mistakes (accountability).

Privacy

Feedback can contain names, phone numbers or route details that identify a person. Messages should be stored without personal details, and used only to improve the service.

Bias

The model learned from English messages written in one style. Feedback in Hindi or another language, or in Hinglish, would be scored on the few words it recognises — so those passengers' complaints might be missed. A fair system must work for everyone who uses the buses.

Transparency and accountability

Every word's score can be shown, so anyone can see why a message was sorted the way it was. The model only sorts; a person reads each complaint and decides what to do.

THE NO-CODE VERSION

The curriculum's NLP use case walkthrough is done in Orange. Its Sentiment Analysis widget is a lexicon, like model A; its Bag of Words with a model is like model B. The tool used is **Orange Data Mining (with the Text add-on)**.

STEP	WIDGET	WHAT IT DOES
1	**Corpus**	Load bus_feedback.csv. Use message as the text feature and sentiment as the target.
2	**Preprocess Text**	Lowercase, remove punctuation and numbers, tokenise, filter stop words, and normalise words (Porter stemmer). Keep not and never.
3	**Word Cloud**	See the most common words — the same idea as chart 1.
4	**Sentiment Analysis**	Try the Liu Hu or VADER lexicon. This is model A.
5	**Bag of Words**	Turn each message into word counts.
6	**Test and Score + Naive Bayes or Logistic Regression**	Learn from the word counts — model B — and compare its accuracy with the lexicon.
7	**Confusion Matrix → Corpus Viewer**	Read the messages it got wrong.

TECHNICAL DETAILS

Domain of AI	Natural Language Processing
Programming language	Python 3
Libraries	pandas, NumPy and matplotlib (Unit 7 of the curriculum)
Where to run it	Jupyter Notebook (bus_feedback.ipynb), or any Python editor (bus_feedback.py)
No-code tool	Orange Data Mining (with the Text add-on)
Model type	Learning based → Supervised learning → Classification
Charts drawn	4

Setting it up

1. Install Python 3 (Anaconda includes Jupyter Notebook and all three libraries).
2. If they are missing, install the libraries: **pip install pandas numpy matplotlib**
3. Keep the program and its data together in one folder.
4. Run **python bus_feedback.py**, or open **bus_feedback.ipynb** and run each cell in order.

The library calls it uses

CALL	STAGE	WHAT IT DOES
`pd.read_csv()`	Stage 2	Reads the messages
`drop_duplicates() / dropna()`	Stage 3	Removes repeated and empty forms
`text.lower()`	Stage 3	Converts to a common case
`ch.isalpha()`	Stage 3	Keeps letters only, removing punctuation and numbers
`clean.split()`	Stage 3	Tokenises: splits the text into words
`if word in stop_words`	Stage 3	Removes stop words
`word.endswith(...) / word[:-n]`	Stage 3	The simple stemmer
`counts[word] = counts.get(word, 0) + 1`	Stages 3, 4	Counts words — the Bag of Words
`(p - n) / (p + n + 1)`	Stage 4	A word's learned score
`plt.barh / plt.imshow / plt.bar`	Stages 3, 5	The charts

SOURCE CODE

The complete program. Every figure and number in this report came from running it.

`bus_feedback.py`

```
# ------------------------------------------------------------------------
# bus_feedback.py
```

```
#
# AI project on Natural Language Processing: sentiment analysis of feedback
# that passengers send about a city's buses - is each message positive or
# negative?
#
# CBSE Class X, Artificial Intelligence (417), Part C project work.
# SDG 11 (Sustainable Cities and Communities - target 11.2, transport).
#
# The steps are the ones Unit 6 teaches:
#   1. Text normalisation - lower case, remove punctuation and numbers,
#      split into tokens, remove stop words, and stem each word
#   2. Bag of Words - count the words in each message
#   3. A model that decides the sentiment from those words
#
# Two models are compared:
#   - LEXICON (rule based): two lists of words written by a person - good
#     words and bad words. This is how Orange's Sentiment Analysis widget works.
#   - LEARNED WORD SCORES (learning based): from the training messages, the
#     program works out how positive or negative each word tends to be.
#
# Libraries: pandas (the messages), matplotlib (charts), numpy.
# ------------------------------------------------------------------------

# %% Stage 1 - Problem scoping
print("=" * 60)
print("WHAT DO PASSENGERS FEEL?")
print("=" * 60)
print("Who   : the city transport office, and daily bus passengers")
print("What  : hundreds of feedback messages a week, read by nobody")
print("Where : a city bus service with an online feedback form")
print("Why   : complaints read quickly get fixed quickly")
print()

# %% Stage 2 - Data acquisition
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

pd.set_option("display.width", 140)
pd.set_option("display.max_colwidth", 90)

data = pd.read_csv("bus_feedback.csv")

print("--- Stage 2: Data acquisition ---")
print("Messages read :", len(data))
print()
print("The first 5 messages:")
print(data.head(5).to_string(index=False))
print()

# %% Stage 3 - Data exploration
print("--- Stage 3: Data exploration ---")

before = len(data)
# The same form submitted twice has the same ID and the same message. Two
# different passengers may write the same words - those are both kept.
data = data.drop_duplicates()
print("Messages sent twice           :", before - len(data), "(removed)")
empty = data["message"].isnull().sum()
data = data.dropna(subset=["message"])
print("Empty forms                   :", empty, "(removed)")
```

```
print("Messages left                 :", len(data))
print(data["sentiment"].value_counts().to_string())
print()

# Stop words: very common words that carry no feeling on their own.
# "not" and "never" are NOT on the list - they change the meaning.
stop_words = ["the", "a", "an", "and", "was", "were", "is", "are", "it", "to", "at",
              "of", "for", "our", "we", "i", "his", "this", "today", "also", "but",
              "very", "too", "again", "all", "every", "route", "bus", "trip",
              "morning", "evening", "please", "thanks", "an", "did", "on"]

# Text normalisation, one message at a time.
tokens_list = []
for text in data["message"]:
    text = text.lower()                                  # 1. common case
    clean = ""
    for ch in text:                                      # 2. keep letters only
        if ch.isalpha():
            clean = clean + ch
        else:
            clean = clean + " "
    tokens = []
    for word in clean.split():                           # 3. tokenise
        if word in stop_words:                           # 4. remove stop words
            continue
        for ending in ["ing", "ed", "s"]:                # 5. stem: cut endings
            if word.endswith(ending) and len(word) > 4:
                word = word[: -len(ending)]
                break
        tokens.append(word)
    tokens_list.append(tokens)
data["tokens"] = tokens_list

example = data.iloc[0]
print("Text normalisation of one message:")
print("  Original :", example["message"])
print("  Tokens   :", example["tokens"])
print()

# Bag of Words for the first three messages, on a few words.
vocabulary = []
for tokens in data["tokens"]:
    for word in tokens:
        if word not in vocabulary:
            vocabulary.append(word)
print("Words in the vocabulary :", len(vocabulary))
print()
show = ["driver", "polite", "rude", "late", "clean", "not", "never"]
bow = pd.DataFrame(0, index=["msg 1", "msg 2", "msg 3"], columns=show)
for m in range(3):
    for word in data.iloc[m]["tokens"]:
        if word in show:
            bow.loc["msg " + str(m + 1), word] = bow.loc["msg " + str(m + 1), word] + 1
print("Bag of Words (part of it) for the first three messages:")
print(bow)
print()

# Chart 1: the most common words in positive and in negative messages.
plt.figure(figsize=(9, 4.2))
for k, feeling in enumerate(["positive", "negative"]):
    counts = {}
```

```
    for tokens in data[data["sentiment"] == feeling]["tokens"]:
        for word in tokens:
            counts[word] = counts.get(word, 0) + 1
    top = pd.Series(counts).sort_values(ascending=False).head(10)
    plt.subplot(1, 2, k + 1)
    plt.barh(top.index[::-1], top.values[::-1], color=["#59a14f", "#e15759"][k])
    plt.title("Most common words: " + feeling)
    plt.xlabel("Times used")
plt.tight_layout()
plt.savefig("chart1_common_words.png")
plt.close()

# %% Stage 4 - Modelling
print("--- Stage 4: Modelling ---")
data = data.sample(frac=1, random_state=12)
cut = int(len(data) * 0.8)
train = data[:cut]
test = data[cut:]
print("Training messages :", len(train))
print("Testing messages  :", len(test))
print()

# Model A - LEXICON. Two lists written by a person (in stemmed form).
good_words = ["polite", "helpful", "clean", "comfortable", "smooth", "friendly", "great",
              "safely", "pleasant", "properly", "right"]
bad_words = ["late", "rude", "overcrowd", "crowd", "dirty", "broken", "uncomfortable",
             "terrible", "fast", "skipp", "shout", "wait", "complaint", "broke"]

# Model B - LEARNED WORD SCORES. For every word: how much more often it
# appears in positive training messages than in negative ones, on a scale
# from -1 (only ever negative) to +1 (only ever positive).
pos_count = {}
neg_count = {}
for i in range(len(train)):
    for word in train.iloc[i]["tokens"]:
        if train.iloc[i]["sentiment"] == "positive":
            pos_count[word] = pos_count.get(word, 0) + 1
        else:
            neg_count[word] = neg_count.get(word, 0) + 1
score = {}
for word in vocabulary:
    p = pos_count.get(word, 0)
    n = neg_count.get(word, 0)
    score[word] = (p - n) / (p + n + 1)

scores = pd.Series(score).sort_values()
print("Most negative words learned :", list(scores.index[:6]))
print("Most positive words learned :", list(scores.index[-6:][::-1]))
for word in ["not", "never", "complaint", "late"]:
    print("Learned score for '" + word + "'".ljust(12 - len(word)), ":", round(score.get(word, 0),
2))
print()

# Chart 2: the learned word scores.
pick = pd.concat([scores.head(8), scores.tail(8)])
plt.figure(figsize=(7, 5))
colours = []
for v in pick.values:
    if v > 0:
        colours.append("#59a14f")
    else:
```

```
        colours.append("#e15759")
plt.barh(pick.index, pick.values, color=colours)
plt.axvline(0, color="#999999")
plt.title("What the model learned about each word")
plt.xlabel("Word score (-1 negative ... +1 positive)")
plt.tight_layout()
plt.savefig("chart2_word_scores.png")
plt.close()

lexicon = []
learned = []
for tokens in test["tokens"]:
    total_a = 0
    total_b = 0
    for word in tokens:
        if word in good_words:
            total_a = total_a + 1
        if word in bad_words:
            total_a = total_a - 1
        total_b = total_b + score.get(word, 0)     # a word never seen scores 0
    if total_a >= 0:
        lexicon.append("positive")
    else:
        lexicon.append("negative")
    if total_b >= 0:
        learned.append("positive")
    else:
        learned.append("negative")

# %% Stage 5 - Evaluation
print("--- Stage 5: Evaluation ---")
actual = list(test["sentiment"])
results = []
for name, predicted in [["Lexicon", lexicon], ["Learned", learned]]:
    right = 0
    caught = 0
    negatives = 0
    for i in range(len(actual)):
        if predicted[i] == actual[i]:
            right = right + 1
        if actual[i] == "negative":
            negatives = negatives + 1
            if predicted[i] == "negative":
                caught = caught + 1
    results.append([name, right / len(actual) * 100, caught / negatives * 100])
    print(name.ljust(8), "accuracy                 :", round(right / len(actual) * 100, 1), "%")
    print(name.ljust(8), "complaints caught (recall):", round(caught / negatives * 100, 1), "%")
print()

# Messages the lexicon got wrong but the learned model got right.
print("Where the word lists went wrong:")
shown = 0
for i in range(len(actual)):
    if lexicon[i] != actual[i] and learned[i] == actual[i] and shown < 4:
        print("  [" + actual[i] + "]", test.iloc[i]["message"])
        shown = shown + 1
print()

matrix = pd.crosstab(pd.Series(actual, name="Actual"), pd.Series(learned, name="Predicted"))
matrix = matrix.reindex(index=["positive", "negative"], columns=["positive", "negative"],
fill_value=0)
```

```
print("Confusion matrix of the learned model:")
print(matrix)
print()

# Chart 3: confusion matrix.
plt.figure(figsize=(5, 4.3))
plt.imshow(matrix.values, cmap="Greens")
for r in range(2):
    for c in range(2):
        value = matrix.values[r][c]
        if value > matrix.values.max() / 2:
            ink = "white"
        else:
            ink = "black"
        plt.text(c, r, value, ha="center", va="center", fontsize=15, color=ink)
plt.xticks([0, 1], ["positive", "negative"])
plt.yticks([0, 1], ["positive", "negative"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Learned model (testing messages)")
plt.tight_layout()
plt.savefig("chart3_matrix.png")
plt.close()

# Chart 4: the two models compared.
x = np.arange(2)
plt.figure(figsize=(7, 4.2))
plt.bar(x - 0.2, [results[0][1], results[1][1]], width=0.4, label="Accuracy", color="#4e79a7")
plt.bar(x + 0.2, [results[0][2], results[1][2]], width=0.4, label="Complaints caught",
color="#e15759")
plt.xticks(x, ["Lexicon (word lists)", "Learned word scores"])
plt.ylim(0, 100)
plt.ylabel("Percent")
plt.title("Rule based and learning based sentiment")
plt.legend()
plt.tight_layout()
plt.savefig("chart4_models.png")
plt.close()

# %% Stage 6 - Deployment
print("--- Stage 6: Deployment ---")
new_messages = [
    "Bus 45 was never late this week, driver very polite!",
    "Waited 30 minutes and the bus was not clean.",
    "No complaints at all, smooth ride :)",
    "The conductor shouted at students. Terrible.",
]
for text in new_messages:
    clean = ""
    for ch in text.lower():
        if ch.isalpha():
            clean = clean + ch
        else:
            clean = clean + " "
    total = 0
    for word in clean.split():
        if word in stop_words:
            continue
        for ending in ["ing", "ed", "s"]:
            if word.endswith(ending) and len(word) > 4:
                word = word[: -len(ending)]
```

```
                break
            total = total + score.get(word, 0)
        if total >= 0:
            feeling = "POSITIVE"
        else:
            feeling = "NEGATIVE - send to the complaints desk"
        print(text.ljust(52), "score", str(round(total, 2)).rjust(6), "->", feeling)
```

SAMPLE OUTPUT

An actual run of the program, exactly as it appeared on screen.

Output — python bus_feedback.py

```
============================================================
WHAT DO PASSENGERS FEEL?
============================================================
Who   : the city transport office, and daily bus passengers
What  : hundreds of feedback messages a week, read by nobody
Where : a city bus service with an online feedback form
Why   : complaints read quickly get fixed quickly

--- Stage 2: Data acquisition ---
Messages read : 422

The first 5 messages:
feedback_id
message sentiment
      F7180                                    TODAY THE AC WAS NOT WORKING ALSO NOBODY ANSWERED THE
HELPLINE.  negative
      F7273                         The new buses are great also the bus was late again and the bus was
clean  positive
      F7244                                                                              TODAY THE BUS BROKE
DOWN  negative
      F7343                                                 No complaints at all but the staff were
friendly  positive
      F7074 ROUTE 45: THE CONDUCTOR SHOUTED AT AN OLD MAN. THE DRIVER USED HIS PHONE AND THE SEATS WERE
DIRTY  negative

--- Stage 3: Data exploration ---
Messages sent twice           : 1 (removed)
Empty forms                   : 1 (removed)
Messages left                 : 420
sentiment
negative    210
positive    210

Text normalisation of one message:
  Original : TODAY THE AC WAS NOT WORKING ALSO NOBODY ANSWERED THE HELPLINE.
  Tokens   : ['ac', 'not', 'work', 'nobody', 'answer', 'helpline']

Words in the vocabulary : 63

Bag of Words (part of it) for the first three messages:
       driver  polite  rude  late  clean  not  never
msg 1       0       0     0     0      0    1      0
msg 2       0       0     0     1      1    0      0
msg 3       0       0     0     0      0    0      0

--- Stage 4: Modelling ---
Training messages : 336
Testing messages  : 84

Most negative words learned : ['fix', 'give', 'minute', 'forty', 'wait', 'down']
```

```
Most positive words learned : ['polite', 'helpful', 'safely', 'pleasant', 'comfortable', 'crowd']
Learned score for 'not'         : -0.41
Learned score for 'never'       : -0.09
Learned score for 'complaint'   : 0.72
Learned score for 'late'        : -0.07

--- Stage 5: Evaluation ---
Lexicon  accuracy                : 85.7 %
Lexicon  complaints caught (recall): 82.2 %
Learned  accuracy                : 96.4 %
Learned  complaints caught (recall): 95.6 %

Where the word lists went wrong:
  [positive] Bus 12 - it was not crowded, the ticket machine worked
  [positive] BUS 12 - I REACHED SCHOOL ON TIME ALSO THE BUS STOPPED PROPERLY AT EVERY STOP. IT WAS A
TERRIBLE JOURNEY, IT WAS NOT CROWDED!
  [negative] The conductor did not give change. Please fix this
  [negative] The driver used his phone. nobody answered the helpline.

Confusion matrix of the learned model:
Predicted  positive  negative
Actual
positive         38         1
negative          2        43

--- Stage 6: Deployment ---
Bus 45 was never late this week, driver very polite! score   0.95 -> POSITIVE
Waited 30 minutes and the bus was not clean.         score  -2.29 -> NEGATIVE - send to the complaints
desk
No complaints at all, smooth ride :)                 score   2.97 -> POSITIVE
The conductor shouted at students. Terrible.         score  -1.46 -> NEGATIVE - send to the complaints
desk
```

TESTING

The program was tested by running it. Every case gives the whole program a small input of its own and records what actually came back.

TEST CASE	EXPECTED	ACTUAL RESULT	RESULT
All messages are read	`Messages read : 422`	`Messages read : 422`	Pass
A form submitted twice (same ID, same words) is removed once	`Messages sent twice : 1 (removed)`	`Messages sent twice : 1 (removed)`	Pass
An empty form is removed	`Empty forms : 1 (removed)`	`Empty forms : 1 (removed)`	Pass
Learned word scores beat the word lists	`Learned accuracy : 96.4 %`	`Learned accuracy : 96.4 %`	Pass
"No complaints" is learned as a good sign, though "complaint" sounds bad	`Learned score for 'complaint' : 0.72`	`Learned score for 'complaint' : 0.72`	Pass
Clear praise against clear complaints: no mistakes	`Learned accuracy : 100.0 %`	`Learned accuracy : 100.0 %`	Pass
Same words from a different passenger (new ID) are kept	`Messages sent twice : 0 (removed)`	`Messages sent twice : 0 (removed)`	Pass

A message of only numbers and punctuation becomes zero tokens and does not crash	`Messages left : 41`	`Messages left : 41`	Pass
The data file is missing	`FileNotFoundError: [Errno 2] No such file or directory: 'bus_feedback.csv'`	`FileNotFoundError: [Errno 2] No such file or directory: 'bus_feedback.csv'`	Pass

RESULTS AND FUTURE SCOPE

What the project found

1. After normalisation, the whole set of messages uses only 63 different words.
2. The learned word scores reached 96.4 %, against 85.7 % for the word lists.
3. They caught 95.6 % of complaints, against 82.2 %.
4. "Complaint" sounds negative but is learned as positive, because people write "no complaints".
5. The word lists fail on words they do not contain, and on words turned around by not and never.

Limitations

- Made-up messages, simpler and more regular than real ones
- English only
- A Bag of Words ignores word order: "not good" and "good, not bad" are hard for it
- It says whether a message is negative, not what the complaint is about

Future scope

- Count word pairs (bigrams) such as "not clean" and "never late", so word order begins to count
- Add a third class, neutral, for questions and suggestions
- Sort complaints by topic as well — the Complaint Routing project shows how
- Collect anonymous feedback on your school canteen and build the same project on it

What had to be learnt along the way

- The steps of text normalisation: case, punctuation, tokens, stop words, stemming
- Bag of Words, and why it ignores word order
- Lexicon-based and learning-based sentiment analysis
- Python string methods: lower(), isalpha(), split(), endswith()

CONCLUSION

The project sorted bus feedback into positive and negative after normalising the text and counting its words. Word scores learned from the data reached 96.4 % and caught 95.6 % of the complaints.

Comparing it with a lexicon showed what learning adds. A person's word list called "no complaints" a complaint, called "not crowded" negative, and missed complaints made of words it did not contain. The learned scores, taken from real messages, got most of these right.

The model still reads words one at a time. Teaching it that "not" changes the next word — by counting word pairs — is the clear next step.

REFERENCES

1. CBSE, Artificial Intelligence curriculum, Class X — Unit 6 practical: Introduction to Sentiment Analysis, case walkthrough.
2. Hu, M. and Liu, B. (2004). Mining and summarizing customer reviews. Proceedings of KDD 2004 — the Liu Hu lexicon used in Orange.
3. Orange Data Mining, Text add-on: Sentiment Analysis widget — https://orangedatamining.com/widget-catalog/text-mining/sentimentanalysis/
4. CBSE, Artificial Intelligence (Subject Code 417), Class X curriculum, 2026-27 — Part C, Practical and Project Work.
5. CBSE, Artificial Intelligence Facilitator Handbook, Class X.
6. pandas documentation — https://pandas.pydata.org/docs/
7. NumPy documentation — https://numpy.org/doc/
8. Matplotlib documentation — https://matplotlib.org/stable/
9. LambdaLab — https://www.lambdalab.in