Big Data Engineer II

By Christan Versteeg, 500859503

Contents

[1. Summary/Abstract 2](#_Toc163105671)

[2. Introduction 2](#_Toc163105672)

[3. Background 2](#_Toc163105673)

[4. Methods 3](#_Toc163105674)

[4.1. Data Collection 3](#_Toc163105675)

[4.2. Data Preparation 4](#_Toc163105676)

[4.3. Sentiment Analysis Implementation 4](#_Toc163105677)

[5. Results 4](#_Toc163105678)

[6. Conclusion and Recommendations 4](#_Toc163105679)

[7. Bibliography 5](#_Toc163105680)

# Summary/Abstract

This document reports on a sentiment analysis project that evaluates hotel reviews using three different methods: the rule-based VADER and TextBlob models, and a supervised Naive Bayes classifier. We discuss the theoretical underpinnings of these models, describe our data collection and preparation methodology, detail the training process of the Naive Bayes classifier, and analyze its performance. We conclude with a discussion of the results and provide recommendations for future work.

# Introduction

The document outlines the execution and results of a sentiment analysis task performed on hotel reviews. Sentiment analysis is an invaluable tool in understanding customer sentiment and can inform business decisions and strategies. This analysis is particularly pertinent to the hospitality industry, where customer satisfaction is paramount.

# Background

First I started of by tackling what NoSQL program I would use. I decided to use MongoDB since it is suggested in the assignment, plus it has a clean UI called MongoDB Compass. I downloaded it on the website. (MongoDB, 2024)

A screenshot of a computer

Description automatically generated

I then created a Database and a collection for it, and imported the hotel reviews csv:

A screenshot of a computer

Description automatically generated

It imported roughly half a million entries:

A screenshot of a computer

Description automatically generated

After importing the csv into mongo I wanted to access it in the Python script, which I did like so:  
A screenshot of a computer

Description automatically generated

Which I did at the hand of PyMongo and Pandas documentation (MongoDB, 2024) (Pandas, 2023).

Interestingly, I figured out a limitation of the DB, for reasons I do not know yet, essentially, importing the csv allows for easy and quick display of ALL 500k reviews. However, if the DB tries to display even 100k reviews, it will given an overflow message saying it has exceeded usage of 200MB, (exceeded as 253MB specifically). I figured this out because I wanted a quick and easy local variant of my program, like so:

from pymongo import MongoClient

import pandas as pd

db\_driven = False

if db\_driven:

    client = MongoClient("localhost", 27017)

    db = client['Big']

    collection = db['Data']

    df = pd.DataFrame(list(collection))

else:

    csv\_file = r"C:\Users\pooti\Desktop\Big-Data-Engineer\App\Hotel\_Reviews.csv"

    df = pd.read\_csv(csv\_file)

Then I decided to work on a basic implementation of the dashboard. I wanted to figure out what dashboard library would be the best to utilize, so I compared it online. (Pavlovych, 2024). I ended up choosing Streamlit because of having experience with it before, and it being very quick to set up prototypes and easy to use, I do not need much flexibility.

The basic implementation of the dashboard ended up looking like this:

A screenshot of a hotel registration

Description automatically generated

A screen shot of a computer

Description automatically generated

Which due to the simplistic nature of streamlit, only required a few lines of code:  
A screen shot of a computer code

Description automatically generated

Which I wrote with the help of the streamlit documentation. (Streamlit, 2024).

After this, I wrote a very simple new script that stores the columns:

A screenshot of a computer program

Description automatically generated

This helps to later work with the columns, as well as currently filtering the only relevant columns in the dashboard:

A screenshot of a computer program

Description automatically generated

Which only changes the default view, they can still be added by the user:

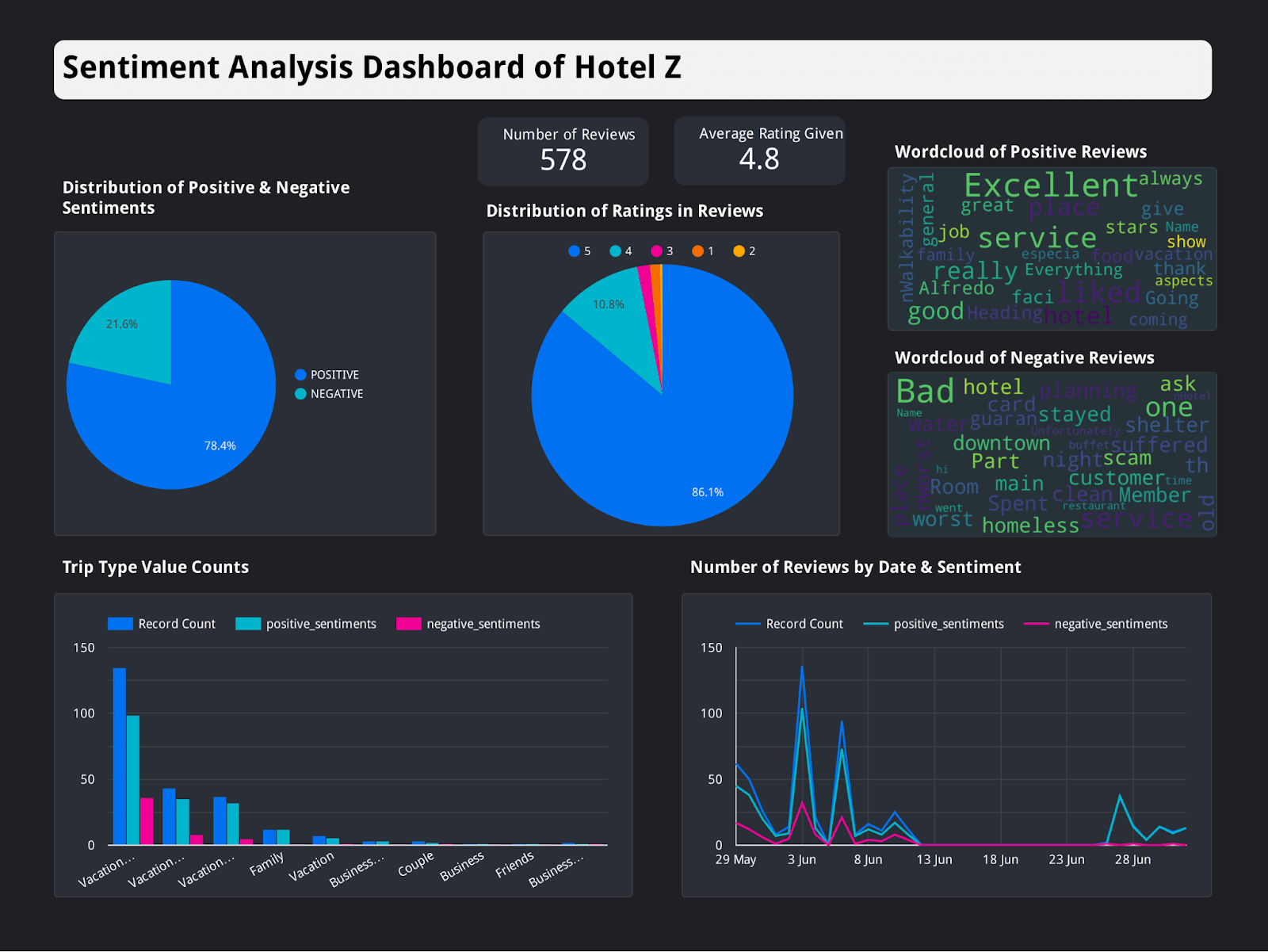
A screenshot of a phone

Description automatically generated

But it now gives a much cleaner table view, which you is visible without having to scroll horizontally:  
A screenshot of a computer

Description automatically generated

Next up I wanted to flash out my dashboard more, and took this source as a good reference for what I’d want my dashboard to roughly look like (Bagaskara, Güler, Rasyid, & Torcato, 2024):



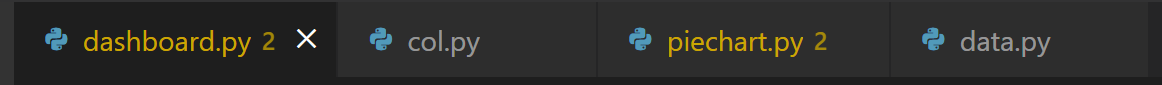
I needed a way to draw these graphs in python, which I chose matplotlib for. It’s the most popular visualization library in python. (Talaj, 2024).

With a basic implementation of the pie chart added into the program:

A green and red circle with white text

Description automatically generated

I decided to split up my classes:



Initially I tried to import the data file of dashboard into piechart, but this gave a circular dependency because piechart was being used in dashboard. So I created a data script to resolve this dependency, now every file that uses data will grab from the data file.

After this I created a new class for showing totals, like in the reference image. With some simple code I managed to get some a very simple layout:

A black background with white text

Description automatically generated

import col

import data

import streamlit as st

num\_positive = data.df[col.POSITIVE\_REVIEW].shape[0]

num\_negative = data.df[col.NEGATIVE\_REVIEW].shape[0]

total\_reviews = data.df.shape[0]

average\_rating = data.df[col.REVIEWER\_SCORE].mean()

def draw():

    st.markdown(

    f'<p>Number of Reviews</p><h1>{total\_reviews}</h1>',

    unsafe\_allow\_html=True

    )

    st.markdown(

    f'<p>Average Rating Given</p><h1>{average\_rating:.2f}</h1>',

    unsafe\_allow\_html=True

    )

This however didn’t look very good so I decided to use the build in columns feature of streamlit (Streamlit, 2024):

A screen shot of a number

Description automatically generated

def draw():

    col1, col2 = st.columns(2)

    with col1:

        st.markdown(

        f'<p>Number of Reviews</p><h1>{total\_reviews}</h1>',

        unsafe\_allow\_html=True

        )

    with col2:

        st.markdown(

        f'<p>Average Rating Given</p><h1>{average\_rating:.2f}</h1>',

        unsafe\_allow\_html=True

        )

Which already gave it a much cleaner design, and more clear too. However it could still use a bit of polish to stand apart, just like in the refence image. To do this, I use some simple CSS to define a box (Mmdn web docs, 2024) around my text:

A black square object with a black background

Description automatically generated

import col

import data

import streamlit as st

num\_positive = data.df[col.POSITIVE\_REVIEW].shape[0]

num\_negative = data.df[col.NEGATIVE\_REVIEW].shape[0]

total\_reviews = data.df.shape[0]

average\_rating = data.df[col.REVIEWER\_SCORE].mean()

def draw():

    st.html("""

    <style>

    .metric-box {

        background-color: #1e1e1e;

        color: white;

        padding: 10px;

        border-radius: 8px;

        text-align: center;

        box-shadow: 0px 2px 4px rgba(0, 0, 0, 0.2);

    }

    .metric-box h1 {

        margin: 0;

        font-size: 2em;

    }

    .metric-box p {

        margin: 0;

        font-size: 1.2em;

        font-weight: bold;

    }

    </style>

    """)

    col1, col2 = st.columns(2)

    with col1:

        st.html(f'<div class="metric-box"><p>Number of Reviews</p><h1>{total\_reviews}</h1></div>')

    with col2:

        st.html(f'<div class="metric-box"><p>Average Rating Given</p><h1>{average\_rating:.2f}</h1></div>')

After this I continued to work on the piechart class, because I wanted another piechart, just like the reference image, because having an overall score distribution on the dashboard presented as a piechart is a good metric for the user to see. I did this by first renaming the draw function in the class to the specific pie it draws, so draw\_positive\_negative, and then for the new piechart, gave it the name draw\_score\_distribution, so that both could be called in the dashboard.py class like so:

piechart.draw\_positive\_negative()

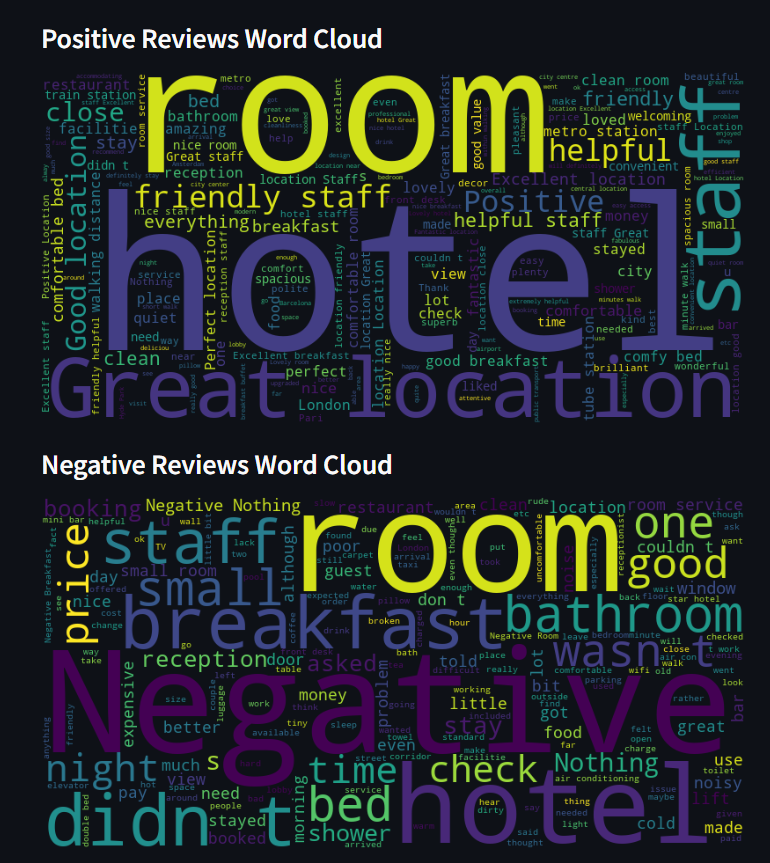
piechart.draw\_score\_distribution()

Other than this, the piechart makes use of histogram math (the graphs that are usually displayed as bars), to section it in each pie. Notably, every pie has been normalized to fit into 1 through 5, since the actual review score is from 1 to 10. (So 1 is 1-2, 2 is 3-4 and so on).

A pie chart with numbers and a number on it

Description automatically generated

For some playful visualization, I also decided to add a wordcloud, like the reference image. Just like the reference image I separated it into two separate wordclouds for negative and positive words. I used an article about wordcloud as reference, despite really liking the masks applied within the article, I don’t think it would be beneficial to add it to the dashboard, so I did not use that step. (Wang, 2022). There’s also some funkiness when it comes to the drawing of streamlit, which makes use of matplotlib, by doing this it will by default give my wordcloud a white background, as denoted in a source found (Mueller, 2020). After fixing the transparency issue, the wordcloud came out clean, looking like this:



Next up, I started working on the machine learning part of the assignment, for this I choice Pytorch and Keras to compare the results. And article written by Kaggle itself was used as basis to support the claims made here (Agrawal, 2021). Keras is said to be easier to use but more constrained, while Pytorch is more flexible but in general hard to use. However; it turns out Keras has several deprecated libraries related to it, such as Bleach (Draves, 2018). This makes the installation process quite tricky. Eventually I managed to fix it, which from what I could tell there were a couple distinct issues. First of all my attempt at trying to fix it with a package manager – Anaconda got me no further, so after that I tried to fix it within the local pip install again. One of the first issues I solved was that I accidentally named my script the same as the library, which then when you try to import your library, it will actually try to import your script (Tilwankar, 2015). Another one was trying to import it through Tensorflow, instead of Keras directly – there was conflicting information online, but eventually found a source that stated recent use that stated since Tensorflow 2.16 the import should detonate separately from tensorflow (Rakita, 2021). Because of dependencies the numpy version that Tensor/Keras needed was also lower than the default, the specific version 1.21.6, below the 2.0 release (Zhang, 2022). Later this was bumped up twice, ending up with 1.23.5, because specific packages required a higher version of it, this did not conflict with the other package requirements. Finally, the last issue was an obscure package dependency, for tensorflow.keras.preprocessing.text, for that you had to install keras-preprocessing, but that didn’t fix it. Eventually I found a obscure Stack Overflow post that said you to install tensorflow-text, which that ended up fixing it (Afnan, 2024). After fixing all of it up I used a simple example script (Awan, 2023) to test out if it was working correctly, and it worked like expected.

I rewrote the keras script to accommodate an actual ML sentiment analysis model on basis of an in depth article (Brownlee, 2022). I first wrote the foundational layers and then grouped it into functions. After that I did the same for PyTorch, the same author wrote an article about PyTorch as well (Brownlee, 2023), which I used as guidance to write the pytorch script, which I also did in the way of first overall structure and then sorting it into functions. After both scripts had their initial function I made a class that overarches both of them to hold common methods.

After finishing the commonalities, I wanted to have the the primitive form of the sentiment analysis of both keras and pytorch uploaded. Which I did by altering the data class, adding a new upload function to it:

def upload(data, collection\_name):

    records = data.to\_dict('records')

    for record in records:

        record.pop('\_id', None)

    client = MongoClient("localhost", 27017)

    db = client['Big']

    target\_coll = db[collection\_name]

    target\_coll.insert\_many(records)

    client.close()

Which was primitively implemented in the keras and pytorch class like so:

    p1 = predict\_sentiment("The hotel was fantastic!")

    p2 = predict\_sentiment("The room was dirty and the service was terrible.")

    df\_preds = pd.DataFrame({"Prediction": [p1, p2]})

    data.upload(df\_preds, "Keras")

prediction()

Which ended up looking like this in the database:

A screenshot of a computer

Description automatically generated

The Naive Bayes classifier is a probabilistic model based on Bayes' theorem, which is particularly effective for text classification tasks due to its simplicity and speed. VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool optimized for social media text. TextBlob is a Python library that offers a simple API for common natural language processing (NLP) tasks, including sentiment analysis, which it performs using a trained Naive Bayes classifier. (Korab, 2023) (Pius, 2023) (Navlani, 2020)

# Methods

## Data Collection

The dataset was obtained from the Kaggle repository, consisting of hotel reviews categorized as positive or negative. Additional reviews were scraped from TripAdvisor to augment the dataset.

A computer screen shot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generated

## Data Preparation

Reviews were preprocessed by removing non-textual elements and normalizing the text. (Small sample is taken in this code for increased compilation time, loading in all of the reviews works perfectly fine, it just takes long). All of the data is send to the database.



A screenshot of a computer

Description automatically generated

## Sentiment Analysis Implementation

VADER and TextBlob were applied to the dataset without further training, leveraging their built-in sentiment analysis capabilities. Python's NLTK library facilitated the use of VADER, while TextBlob was directly applied for sentiment evaluation. The Naive Bayes classifier was trained on the Kaggle data.

# Results

The Naive Bayes classifier achieved an accuracy of 85-90%, while VADER and TextBlob provided fast and consistent sentiment assessments across the dataset. It general the Naive Bayes classifier was the most accurate, then VADER and afterwards TextBlob.

# Conclusion and Recommendations

The comparative analysis showed that while the Naive Bayes classifier provided a high accuracy rate, rule-based models like VADER and TextBlob offer rapid sentiment assessment for large datasets and ease of use. For real-time analysis, VADER and TextBlob are recommended due to their simplicity and efficiency.

# Bibliography

Korab, P. (2023, May 14). *Fine-tuning VADER Classifier with Domain-specific Lexicons*. Opgehaald van Medium: https://pub.towardsai.net/fine-tuning-vader-classifier-with-domain-specific-lexicons-1b23f6882f2

Navlani, A. (2020, September 5). *Naive Bayes Classification using Scikit-learn*. Opgehaald van Medium: https://avinashnavlani.medium.com/naive-bayes-classification-using-scikit-learn-60bc5176f868

Pius, A. (2023, November 22). *Using Python TextBlob for Text Classification*. Opgehaald van Medium: https://medium.com/chat-gpt-now-writes-all-my-articles/using-python-textblob-for-text-classification-7953014f54e6