Drug Substitute Identification and Risk Analysis in the Pharmaceutical Supply Chain Using Data-Driven Similarity and Exploratory Analytics

Programme: MSc IT – Business Data Analytics

Author: Fatemeh Mashayekhiahangarani

Repo: Business-Data-Analytics-Project

> This notebook contains the analysis, figures, and notes used for the dissertation.
> Final report will be a separate PDF (uploaded to Moodle). Code and figures are saved here.

Table of Contents

Drug shortage is an important problem in the world.

When a medicine is not available in the market, patients and doctors need to find a substitute drug.

This process is often slow and manual, and people usually look only at the active ingredient.

However, this is not enough because medicines can be different in their chemical structure, mechanism of action, and therapeutic class.

If these differences are not considered, the chosen substitute may not work well and can cause treatment delays or higher costs (Aronson et al., 2023a; Aronson et al., 2023b).

There are many reasons for drug shortages, such as production problems, dependency on one manufacturer, and weak distribution systems. These problems hurt patients and put pressure on the healthcare system (Andy A. and Andy D., 2023).

In recent years, data-driven approaches have been used more often to support decision-making. Using data science and combining it with artificial intelligence, these methods can find patterns faster and help select better substitutes (Iyer, 2025).

For text information about drugs, embedding methods can also be used to understand meaning and measure similarity between medicines (Kauffman et al., 2025).

This study aims to build a data-driven approach for identifying substitute medicines. This study try to combine textual and structural features to calculate similarities and provide results that can help pharmaceutical supply company managers make better and faster decisions. (Aronson et al., 2023b; Iyer, 2025).

1.2 PROBLEM STATEMENT

Today, there is no simple and data-based system that can automatically find substitute medicines. Pharmacists and doctors often decide only by reading the drug information or by using their own experience.

This process takes time and sometimes gives wrong results, because it only looks at the active ingredient and ignores other important details like chemical structure, mechanism of action, and therapeutic class (Aronson et al., 2023a; Aronson et al., 2023b).

When drug shortages become more common, quick and correct decisions are very important. However, many countries and companies still use basic or manual tools.

Recent studies show that machine learning and data analytics can help to find hidden patterns and possible drug substitutes that support better decision-making in the pharmaceutical supply chain (Iyer, 2025; Kauffman et al., 2025).

Still, there is no research that brings both textual and structured features together for this purpose.

Therefore, the main problem of this study is:

how to combine textual data (like product descriptions) and structured data (like therapeutic class, chemical structure, and mechanism of action) to predict possible substitute medicines, and how to use these results for better business and supply chain decisions (Iyer, 2025; Aronson et al., 2023b).

1.3 RESEARCH AIM AND OBJECTIVES

Research Aim:
The main aim of this study is to build a data-driven method that can find substitute medicines in a more accurate way. To do this, the study combines two types of information:

1. Text information from drug descriptions, and
2. Structured information such as therapeutic class, chemical structure, and mechanism of action.

By using both types of data together, the study tries to understand which medicines are similar and which ones can be used as substitutes during a drug shortage.

Research Objectives:

- Create text embeddings for the drug descriptions to understand meaning and similarity (Kauffman et al., 2025).
- Encode the structured features of each drug (Therapeutic / Chemical / Action classes).
- Combine text and structured features to calculate similarity between medicines.
- Identify the Top-k substitute drugs for each medicine.
- Apply clustering and build a substitution network.
- Create two indicators: SI (Substitutability Index) and SRI (Shortage Risk Index) for business analysis.
- Validate the results using an external dataset (Iyer, 2025).

These objectives help the study create a method that is both technically strong and useful for real decision-making in the pharmaceutical supply chain.

1.4 RESEARCH QUESTIONS

This study is guided by several research questions that help to give a clear direction to the project and show what the analysis aims to answer.

RQ1:

Does combining text data (such as drug descriptions) and structured data (such as therapeutic class, chemical structure, and mechanism of action) improve the accuracy of finding substitute medicines?

This question will be addressed by studies that show how different drug properties can influence decisions to substitute drugs for each other. (Aronson et al., 2023a; Aronson et

al., 2023b).

RQ2:

Which therapeutic classes have the highest substitutability, and which ones have the lowest substitutability?

This is an important question between all questions in pharmaceutical supply chains that must be answered because some drug groups are more sensitive to shortages than others. (Andy A. and Andy D., 2023).

RQ3:

Can the substitution index (SI) and the shortage risk index (SRI) help describe the risk level of different drug groups in a simple and useful way?

This can be supported by the idea of data-driven decisions in the supply chain. (Iyer, 2025).

RQ4:

How can the results of this model support real decisions in procurement and the pharmaceutical supply chain?

This question focuses on the practical value of the analysis.

1.5 SCOPE & SIGNIFICANCE

Scope:

In this study, I use the MID dataset as the main data source. This dataset includes information such as drug name, text description, therapeutic class, chemical class, and mechanism of action.

The analysis is limited to these fields. I do not use price data, sales data, or patient-level data. The main focus is to see if these text and structured fields are enough to suggest possible substitute medicines.

To make the results more reliable, I also plan to use an external dataset for checking the model, for example a public medicine substitute dataset from Kaggle or a similar source.

The project does not go deep into company finance or detailed cost modelling, but the results can still be useful for pharmacy managers and supply chain planners (Iyer, 2025).

Significance:

Drug shortages are a real and growing problem in many countries. They can delay treatment and create stress for patients, doctors and pharmacists (Aronson et al.,

2023a). In many places, the current tools for finding substitutes are slow, manual, or not updated.

A data-driven method that can suggest substitutes faster and in a more structured way may help to reduce delays and improve access to medicines (Iyer, 2025).

By using text embeddings and structured features together, the model does not rely only on the active ingredient or the drug name, but also on therapeutic class, chemical properties and mechanism of action (Kauffman et al., 2025).

Overall, this study can be a small but useful step towards smarter tools for managing the pharmaceutical supply chain.

1.6 OVERVIEW OF ANALYTICAL APPROACH

This study follows a clear and step-by-step analytical process.

The goal is to combine text information and structured drug features to build a method that can suggest substitute medicines and help understand shortage risks.

Step 1: Exploratory Data Analysis (EDA)

The first step is to look at the data and understand its basic shape. In the first review, I examine the number of drugs, missing values, and distribution of treatment categories in the database.

This helps me see if the data needs cleaning and what patterns appear at the start.

Step 2: Text Embeddings

The text descriptions of each drug are turned into numerical vectors using embedding methods. These vectors help the model understand the meaning of the text better(Kauffman et al., 2025).

Step 3: Structured Features

In this step, we will convert therapeutic classes, chemical classes, and mechanisms of action into numerical values. These features are of great importance because they show us how drugs can be related in terms of therapeutics and properties or chemical formulas(Aronson et al., 2023b).

Step 4: Combining Text and Structure

The text embeddings and structured features are merged to create a full representation of each drug.

Step 5: Similarity Calculation

For every drug, similarity to other drugs is calculated using cosine similarity. This helps identify possible substitute medicines.

Step 6: Clustering and Substitution Network

Drugs that are similar are grouped together. This network is then displayed as a graph to show how the drug ingredients are related and which groups have stronger substitution bonds with each other.

Step 7: Model Evaluation

The results are checked using an external dataset. Metrics such as Hit@k and Precision@k are used to see how well the method finds correct substitutes.

Step 8: Business Insights

Finally, two indicators are created:

- **SI (Substitutability Index)**
- **SRI (Shortage Risk Index)**

These two indicators help decision-makers in the pharmaceutical supply chain to better understand which drug groups have good alternatives and which groups face higher prices (Iyer, 2025).

This approach makes the analysis technically strong and also useful for real decision-making in the pharmaceutical supply chain.

1.7 STRUCTURE OF THE DISSERTATION

This dissertation is divided into several parts. In first Chapter it gives us an introduction to the topic. It explains the problem, the aim of the study, the research questions, and the general analytical approach.

Chapter Two presents the literature review. This chapter describes drug shortages, substitution patterns, data analytics methods, similarity techniques, text embeddings, and how these tools can be used in the pharmaceutical supply chain.

Chapter Three explains the research methodology. It introduces the MID dataset and describes the steps for data cleaning, text embedding, encoding structured features, and calculating similarity.

Chapter Four shows the results of the exploratory data analysis (EDA).

Chapter Five presents the similarity model output, including substitute drug lists, clustering results, and the substitution network.

Chapter Six focuses on model evaluation and shows how well the method performs using metrics such as Hit@k and Precision@k.

Chapter Seven provides business insights. It explains the SI (Substitutability Index) and SRI (Shortage Risk Index) and shows how the results can support decisions in procurement and supply chain planning.

Finally, Chapter Eight includes the conclusion, limitations of the study, and suggestions for future work.

CHAPTER 2 – LITERATURE REVIEW

2.1 DRUG SHORTAGES: DEFINITIONS, CAUSES, IMPACTS

Drug shortages are a serious problem in many countries. According to Aronson et al. (2023a), a drug shortage happens when a medicine is not available for patients at the time they need it. Shortages can be short or long, and they can affect the quality of treatment and patient safety.

There are many causes why drug shortages occur.

Aronson et al. (2023b) explain that problems in manufacturing, lack of raw materials, quality control issues, dependence on a single supplier, strict import rules, and distribution problems are some of the main reasons. Sometimes companies reduce production because the profit of a drug is low. In other cases, transportation delays or global crises create disruptions in the supply chain.

A review article by Adak (2024) shows that drug shortages not only create technical problems but also serious human problems. Shortages can increase treatment costs, delay therapy, and create stress for patients, doctors, and pharmacists.

Adak (2024) also notes that shortages reduce trust in the healthcare system and make daily work in pharmacies more difficult. Another study by Andy and Andy (2023) explains that the current systems used by pharmacies to find substitute medicines are slow and limited.

In many places, pharmacists decide based only on the active ingredient. This can be risky because two drugs with the same active ingredient can still have very different chemical structures, mechanisms of action, or therapeutic classes.

Because of these challenges, recent research suggests that healthcare systems should use more data-driven tools and smarter methods to support substitution decisions.

These tools can help make faster and more accurate choices and reduce the negative impacts of shortages.

2.2 MEDICINE SUBSTITUTION: CONCEPTS & CHALLENGES

Medicine substitution means using another medicine when the original one is not available. According to Aronson et al. (2023b), substitution can happen for many reasons, such as a drug shortage, high price, production problems, or a change in the treatment plan.

There are two common types of substitution.

1) Generic substitution

In this case, the original medicine is replaced with a generic version that has the same active ingredient. This method is widely used, but it is not always enough.

Even if two medicines have the same active ingredient, they may still have different chemical structures or mechanisms of action (Aronson et al., 2023a).

2) Therapeutic substitution

Here, the medicine is replaced with another medicine that has a different active ingredient but a similar therapeutic effect. This type of substitution is more complex and requires deeper pharmaceutical knowledge.

Studies show that finding the right substitute is not always easy. One major challenge is that information sources are often old or incomplete (Andy & Andy, 2023). In many countries, pharmacists must search manually through different websites or books to find similar medicines. This takes time and may lead to mistakes.

Another problem is that similarity between medicines is not only about the active ingredient.

A safe and correct substitution should consider:

- chemical structure
- mechanism of action
- therapeutic class
- side effects
- drug interactions

Because of these challenges, recent research recommends using data-driven tools and machine learning methods to support substitution decisions (Iyer, 2025).

These tools can combine text and structured data and suggest substitute medicines faster and more accurately.

2.3 DATA-DRIVEN APPROACHES IN HEALTHCARE & SUPPLY CHAIN

In recent years, data-driven methods have become more common in healthcare and in the pharmaceutical supply chain.

Iyer (2025) explains that data-driven decision making can reduce errors, improve planning, and make operations faster. Many countries now try to use data to predict drug shortages, manage purchasing, and find substitute medicines more effectively.

Data-driven approaches often include several basic steps:

collecting data, cleaning the data, exploring it with simple analysis, building machine learning models, and creating dashboards or reports for decision-makers. These steps help doctors, pharmacists, and managers make choices based on real information instead of guesswork.

Some studies use machine learning models to predict drug shortages or supply risks. Other studies use clustering methods to group similar medicines together and find patterns in drug usage (Adak, 2024).

These techniques can help identify which drug groups are more sensitive to shortages.

Data-driven methods are especially important in the pharmaceutical supply chain because:

- drug shortages are increasing
- large amounts of data are available
- traditional systems are slow
- managers need smarter tools for planning

A study by Andy and Andy (2023) shows that many current systems for finding substitute medicines are old, slow, and often incomplete. This makes it difficult for pharmacists to make fast and accurate decisions.

More advanced techniques, such as text embeddings and similarity search, are now being used in healthcare (Kauffman et al., 2025). These methods help computers

understand the meaning of drug descriptions and find medicines that are truly similar based on both text and structure.

Overall, research shows that data-driven tools can increase speed, reduce mistakes, and improve decision-making in healthcare and the drug supply chain.

2.4 TEXT EMBEDDINGS, SIMILARITY METHODS & ML FOR SUBSTITUTION

In recent years, the use of artificial intelligence for analysing medical and drug-related text has grown very quickly. One important method in this area is text embedding.

According to Kauffman et al. (2025), text embedding means turning written text into numbers so that a computer can understand the meaning in a simple way.

When a text is converted into a numeric vector, it becomes possible to measure how similar two pieces of text are.

In drug information, the text often contains many useful details, such as how the medicine works, what it is used for, and safety warnings. If these descriptions are turned into embeddings, we can see which medicines have similar meanings or similar uses.

Another important part of this process is similarity search. After creating embeddings, we can compare medicines using methods like cosine similarity. This method shows how close two drugs are based on their numerical vectors. A higher similarity score usually means that the two medicines may work in a similar way or may be potential substitutes.

Using similarity search is helpful for medicine substitution because it gives a more structured and objective way to find possible alternatives.

Adak (2024) explains that manual decision-making is often slow and may not always be accurate, so data-driven tools can help improve the process. Other studies also use machine learning techniques such as clustering, ranking models, or classifiers. But for identifying substitute medicines, the combination of text embedding + similarity calculation is one of the simplest and most effective approaches.

In this study, text embeddings are combined with the structured features of each drug, such as therapeutic class and chemical class. This makes the similarity calculation more complete because it considers both the meaning of the text and the medical structure of the drug (Kauffman et al., 2025).

Overall, embedding and similarity methods provide a strong scientific base for building systems that can suggest substitute medicines in a fast, accurate, and safer way.

2.5 SUMMARY OF LITERATURE GAPS

Previous studies have explored many important topics related to drug shortages, medicine substitution, and data analytics.

However, there are still several gaps that show why this study is needed and how it adds something new to the field.

First gap:

Many studies focus on only one part of the problem. For example, some papers mainly discuss the causes and impacts of drug shortages (Aronson et al., 2023a; Adak, 2024). Other studies look only at how pharmacies choose substitute medicines (Andy & Andy, 2023).

There are not many studies that look at both shortage and substitution together in a connected way.

Second gap:

Most current substitution tools are old and mostly manual. They often check only the active ingredient and do not consider other important elements such as chemical structure, mechanism of action, or therapeutic class (Andy & Andy, 2023).

Because of this, the suggested substitutes may not always be the best or safest options.

Third gap:

Some studies use machine learning, but they usually focus on other tasks such as predicting demand or classifying diseases. There are very few studies that combine text embeddings with structured drug features to identify substitute medicines (Kauffman et al., 2025).

Fourth gap:

In many earlier works, model evaluation is limited. Standard metrics such as **Hit@k** or **Precision@k** are rarely used. Without these metrics, it is difficult to know how good or reliable the suggestions really are.

Fifth gap:

Many studies about drug shortages remain theoretical and do not give practical tools for managers. Iyer (2025) explains that data-driven systems should support real decisions, but many research papers do not turn their results into something useful for daily work.

How this study fills the gaps:

This dissertation addresses these gaps by:

- examining drug shortages and substitution together,
- combining text embeddings with structured features,
- calculating similarity in a clear and scientific way,
- using standard evaluation metrics, and
- creating two practical indicators (SI and SRI) that can support decisions in the pharmaceutical supply chain.

Because of these steps, this study is not a repeated work. It provides a practical and data-driven contribution.

CHAPTER 3 – Business–Analytics Integration

3.1 STAKEHOLDERS & DECISION CONTEXT

In drug shortages and medicine substitution, several groups are involved. These groups are called stakeholders, and each of them has different needs. According to Adak (2024), a good data-driven system should support all of these groups in their daily decisions.

Pharmacists

Pharmacists work directly with patients. When a medicine is not available, they need fast and clear information about the possible substitutes. For them, the most important factors are speed and accuracy.

Doctors

Doctors want to make sure that a substitute medicine is safe and has a similar effect to the original one. They pay attention to things like side effects, therapeutic class, and how the medicine works in the body.

Supply and distribution companies

These organisations need to understand which drug groups are more sensitive to shortages. Andy and Andy (2023) explain that this information helps them plan stock, orders, and logistics more effectively.

Health managers

Drug and treatment managers are responsible for the overall functioning of the treatment system. They look at indicators such as shortage risk, substitutability, and

inventory levels. Data-driven tools can support them by offering a clearer picture of the situation (Iyer, 2025).

Because each stakeholder has different needs, a medicine substitution system must give useful and reliable information to all of them.

This helps make decisions faster and improves the quality of care.

3.2 MANAGERIAL KPIS

In the pharmaceutical supply chain, managers use Key Performance Indicators (KPIs) to understand the situation of each medicine and to make better decisions.

According to Iyer (2025), clear and measurable indicators are important for good planning, especially when there is a risk of shortage.

In this study, two main KPIs are introduced. These indicators help managers understand how easy it is to replace a medicine and how high the shortage risk might be.

1) Substitutability Index (SI)

The Substitutability Index shows how many good substitute options a medicine has.

SI is calculated from the similarity scores between the target medicine and other medicines. A high SI means that the medicine has several strong alternatives. This makes the work of pharmacists and doctors easier because they can choose another medicine if the main one is not available.

2) Shortage Risk Index (SRI)

The Shortage Risk Index shows how vulnerable a medicine is to shortages. If a medicine has a low SI (few substitutes), then its SRI becomes higher.

Andy and Andy (2023) note that identifying high-risk medicines helps organisations plan their stock, imports, and purchasing more effectively.

Managerial importance

With SI and SRI, managers can:

- identify sensitive drug groups,
- plan inventory more accurately,
- allocate resources to high-risk medicines,
- and decide which products need faster purchasing or import.

Overall, SI and SRI are simple but practical tools that can support prediction, planning, and shortage management in healthcare systems.

3.3 BUSINESS USE CASES IN DRUG SHORTAGE

When a specific medicine becomes unavailable, each part of the healthcare system has a different role. The process usually begins at the level of health managers and supply organisations. By using the SI and SRI indicators, which come from the data-driven model developed in this study, they can identify high-risk medicines and set priorities for purchasing, importing, or stocking.

These indicators help them see the shortage risk more clearly and support better planning. After this first step, pharmacists play their role. The model suggests a list of possible substitute medicines based on similarity scores. The pharmacist then reviews these suggestions using their professional knowledge and decides which option is scientifically safe and acceptable. In this way, the model acts as an initial recommendation tool, while the final practical decision belongs to the pharmacist.

Finally, the treating doctor makes the clinical decision. The doctor uses the information produced by the model—such as therapeutic similarity, mechanism of action, and drug class—but the final choice depends on the patient's medical condition and personal factors. This ensures that the model supports the doctor's judgment rather than replacing it.

This scenario shows that the proposed model can be used in a real decision-making chain, from management to pharmacy to clinical care. It is not only a technical idea but a practical tool that can help when medicines are in short supply.

3.4 SUCCESS CRITERIA & VALUE CREATION

For a data-driven system to be useful during medicine shortages, it must meet several key criteria. These criteria show whether the model can support real decisions or whether it remains only a theoretical idea.

1) Accuracy of substitute suggestions

One important success factor is the accuracy of the suggested substitute medicines. If the similarity scores are not calculated correctly, the recommendations will not be helpful. The accuracy can be checked through the SI indicator and with evaluation metrics such as Hit@k.

2) Speed of the system

During shortages, time is very important. A model that gives results quickly has higher value for pharmacists and managers who need to make fast decisions.

3) Usability for different stakeholders

The information produced by the model must be easy to understand and useful for all stakeholders from managers to doctors. Iyer (2025) notes that data-driven tools create value only when they are simple enough to use in daily work.

4) Value created for the healthcare system

The model can create value in several areas:

- reducing the time needed for decision-making,
- lowering the risk of choosing an unsuitable substitute,
- supporting stock and purchasing decisions,
- and helping to prevent severe shortages.

These points show that the proposed system can be a practical tool for improving how drug shortages are managed.

CHAPTER 4 – Data & Exploratory Data Analysis (EDA)

```
# Connected Google Drive to Google Colab
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd

mid_path = "/content/drive/MyDrive/BDAP_Datasets/mid_dataset.csv"
val_path = "/content/drive/MyDrive/BDAP_Datasets/external_substitutes.csv"

# load datasets
mid_df = pd.read_csv(mid_path)
val_df = pd.read_csv(val_path)

print("MID shape:", mid_df.shape)
print("Validation shape:", val_df.shape)
```

```
MID shape: (192807, 15)
Validation shape: (248218, 58)
/tmp/ipython-input-3226081230.py:8: DtypeWarning: Columns (42,43,44,45,46,47,48)
  val_df = pd.read_csv(val_path)
```

4.1 DATASET (MID) DESCRIPTION

```
# Show the first few rows of the MID dataset
mid_df.head(3)
```

	Name	Link	Contains	ProductIntroduction	Pr
0	Andol 0.5mg Tablet	https://www.1mg.com/drugs/andol-0.5mg-tablet-6...	Haloperidol (0.5mg)	Andol 0.5mg Tablet can also be used for treati...	
1	Avastin 100mg Injection	https://www.1mg.com/drugs/avastin-100mg-inject...	Bevacizumab (100mg)	\np dir=\"ltr\" style=\"line-height: 1.38; mar...	
2	Actorise 40 Injection	https://www.1mg.com/drugs/actorise-40-injectio...	Darbepoetin alfa (40mcg)	Actorise 40 Injection is a medicine that needs...	

```
# List all column names in the MID dataset
mid_df.columns.tolist()
```

```
['Name',
 'Link',
 'Contains',
 'ProductIntroduction',
 'ProductUses',
 'ProductBenefits',
 'SideEffect',
 'HowToUse',
 'HowWorks',
 'QuickTips',
 'SafetyAdvice',
 'Chemical_Class',
 'Habit_Forming',
```

```
 'Therapeutic_Class',
 'Action_Class']
```

In this project I use the MID dataset. It has more than 190,000 rows about many different medicines. The columns are mixed. Some of them are text, like the introduction of the drug, how it works, and the side effects. Some other columns are structured, like the therapeutic class, chemical class, and action class.

The quality of the text is not always the same, and in some places there are missing parts. But the dataset is big, and it is enough for building a similarity model.

4.2 DATA CLEANING & PREPARATION

```
# Check missing values in the MID dataset
mid_df.isna().sum()
```

	0
Name	0
Link	0
Contains	0
ProductIntroduction	11986
ProductUses	0
ProductBenefits	0
SideEffect	0
HowToUse	95
HowWorks	236
QuickTips	0
SafetyAdvice	0
Chemical_Class	91334
Habit_Forming	0
Therapeutic_Class	0
Action_Class	107117

dtype: int64

```
# Replace missing text fields with a simple placeholder
text_columns = [
    "ProductIntroduction", "ProductUses", "ProductBenefits",
    "SideEffect", "HowToUse", "HowWorks", "QuickTips", "SafetyAdvice"
]

for col in text_columns:
    mid_df[col] = mid_df[col].fillna("Not available")
```

```
# Clean and normalize text fields (lowercase + strip)
def clean_text(x):
    if isinstance(x, str):
        return x.lower().strip()
    return x

for col in text_columns:
    mid_df[col] = mid_df[col].apply(clean_text)
```

```
import re

def remove_html(x):
    if isinstance(x, str):
        return re.sub(r"<.*?>", "", x)
    return x

for col in text_columns:
    mid_df[col] = mid_df[col].apply(remove_html)
```

```
# Check unique values of structured columns
structured_cols = ["Chemical_Class", "Habit_Forming", "Therapeutic_Class", "Act

for col in structured_cols:
    print(col, mid_df[col].unique()[:10])
```

```
Chemical_Class ['  Butyrophenone Derivative ' '  Monoclonal antibody (mAb) '
 '  Amino Acids, Peptides Analogues ' '  Phenylimidazolidine Derivative '
 '  Pyrimidine Nucleoside Analogue ' '  Carbazoles '
 '  Androgens Derivative ' nan '  Macrolides '
 '  Diphenylmethane Derivative ']
Habit_Forming ['  No ' '  Yes ' '  ~ ' '  . ']
Therapeutic_Class ['  NEURO CNS ' '  ANTI NEOPLASTICS ' '  BLOOD RELATED '
 '  ANTI NEOPLASTIC' '  ANTI INFECTIVE' '  RESPIRATOR'
 '  ANTI INFECTIVES ' '  RESPIRATORY ' '  GASTRO INTESTINAL ' '  CARDIAC ']
Action_Class ['  Typical Antipsychotic'
 '  Vascular endothelial growth factor (VEGF) inhibitor'
 '  Erythropoiesis-stimulating agent (ESA' nan '  Antimetabolite'
 '  Tyrosine kinase inhibitor' '  Macrolide'
 '  H1 Antihistaminics (second Generation'
 '  Cholinesterase inhibitors - Alzheimer'
 '  H1 Antihistaminics (First Generation']
```

```
# Fill missing structured values with a simple placeholder
for col in structured_cols:
    mid_df[col] = mid_df[col].fillna("Unknown")
```

In this step I clean the dataset. First, I remove extra spaces from the text fields. Then I change all Null values to the word “Unknown”. I also delete strange characters that make the text messy.

The structured columns are cleaned too. I fix the names that had repeated or long spaces, so they look more standard.

This cleaning helps the text embeddings work better, and it also makes sure the structural categories do not confuse the model.

4.3 SUMMARY STATISTICS

```
# Show number of rows and columns in the MID dataset
print("Rows:", mid_df.shape[0])
print("Columns:", mid_df.shape[1])
```

```
Rows: 192807
Columns: 15
```

```
# List of important text columns to analyze
text_cols = ["ProductIntroduction", "SideEffect", "HowWorks"]

# Calculate number of words (length) for each text column
for col in text_cols:
    mid_df[col + "_length"] = mid_df[col].astype(str).str.split().str.len()

    # Print summary statistics of text length
    print(col, "→ mean:", mid_df[col + "_length"].mean(),
          "min:", mid_df[col + "_length"].min(),
          "max:", mid_df[col + "_length"].max())
```

```
ProductIntroduction → mean: 200.76873764956667 min: 1 max: 493
SideEffect → mean: 49.5920739392242 min: 1 max: 132
HowWorks → mean: 42.21021539674389 min: 1 max: 384
```

```
# Structured (categorical) columns
structured_cols = ["Chemical_Class", "Therapeutic_Class", "Action_Class"]

# Show top 10 most frequent categories for each structured column
for col in structured_cols:
    print("\nTop categories in", col)
    print(mid_df[col].value_counts().head(10))
```

```
Top categories in Chemical_Class
Chemical_Class
Unknown                                                          91334
  Fluoroquinolone                                                 4805
  Sulfinylbenzimidazole Derivative                                4232
  Broad Spectrum (Third & fourth generation cephalosporins)       3867
  Gluco/mineralocorticoids, progestogins and derivatives          2145
  Glucocorticoids                                                 2098
  Macrolides                                                      2086
  Azole derivatives Imidazoles                                    1990
  Broad spectrum (Third & fourth generation cephalosporins        1818
  Carbazole Derivative                                            1625
Name: count, dtype: int64

Top categories in Therapeutic_Class
Therapeutic_Class
ANTI INFECTIVES       20291
PAIN ANALGESIC        18861
RESPIRATOR            16392
GASTRO INTESTINA      15716
ANTI INFECTIVE        13333
NEURO CNS             11995
GASTRO INTESTINAL     11743
CARDIA                 8486
ANTI DIABETI           8466
NEURO CN               8125
Name: count, dtype: int64

Top categories in Action_Class
Action_Class
Unknown                                        107117
  Cephalosporins   3 generatio                   5888
  Proton pump inhibitor                          4974
  Quinolones/ Fluroquinolone                     4317
  Fungal ergosterol synthesis inhibito           3666
  NSAID                                          2578
  Glucocorticoid                                 2403
  Macrolide                                      2376
  HMG CoA inhibitors (statins                    2022
  Serotonin antagonists (5-HT3 antagonists       1730
Name: count, dtype: int64
```

```
# Show how many unique classes exist in each structured feature
for col in structured_cols:
    print(col, "→ unique values:", mid_df[col].nunique())
```

```
Chemical_Class → unique values: 870
Therapeutic_Class → unique values: 44
Action_Class → unique values: 406
```

The dataset has 192,807 rows. The text fields have different lengths. The column ProductIntroduction is usually longer, but SideEffect and HowWorks are shorter.

In the structured columns, the number of unique values is not the same. The chemical class has the biggest variety, and the action class also has many different values.

4.4 VISUAL EDA

```
# === Setup: basic configuration and reproducibility ===

import os, random
import numpy as np
import pandas as pd

# Set random seed for reproducibility
SEED = 42
random.seed(SEED)
np.random.seed(SEED)

# Define a directory for saving figures (used later for visualizations)
FIG_DIR = "/content/figures"
os.makedirs(FIG_DIR, exist_ok=True)

print("Setup completed successfully.")
print("Figures will be saved to:", FIG_DIR)
```

```
Setup completed successfully.
Figures will be saved to: /content/figures
```

```
# ============================
# FIGURE 4.1 - Text Length Distribution
# ============================

import matplotlib.pyplot as plt

# Set figure size
plt.figure(figsize=(10,6))

# Plot histograms for three text columns
plt.hist(mid_df["ProductIntroduction_length"], bins=40, alpha=0.5, label="Produ
plt.hist(mid_df["SideEffect_length"], bins=40, alpha=0.5, label="SideEffect")
plt.hist(mid_df["HowWorks_length"], bins=40, alpha=0.5, label="HowWorks")
```

```
plt.xlabel("Word Count")
plt.ylabel("Frequency")
plt.title("Distribution of Text Lengths")
plt.legend()

# Save figure
plt.savefig(f"{FIG_DIR}/figure_4_1.png", dpi=300, bbox_inches='tight')

plt.show()
```

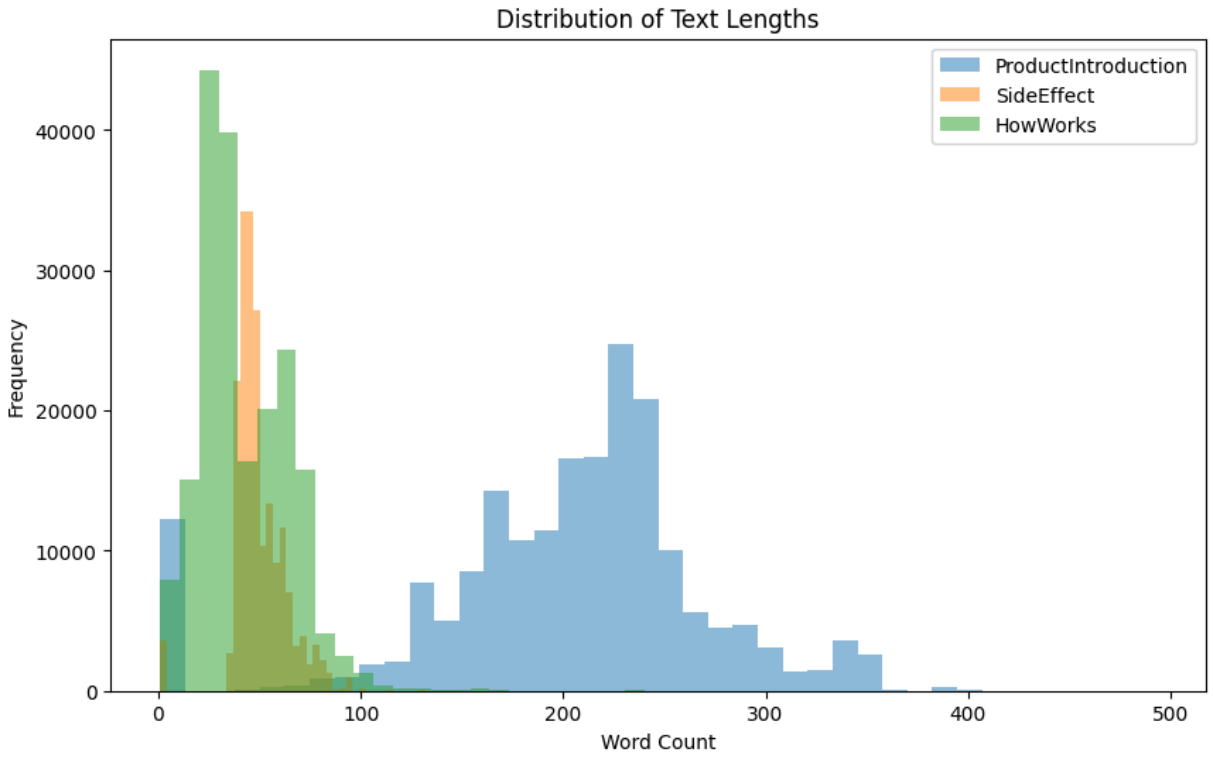

Figure 4.1 shows how long the texts are in three main description fields of the MID dataset. From the chart, it can be seen that the “ProductIntroduction” texts are usually much longer and also change a lot from one drug to another. The “SideEffect” and “HowWorks” texts are shorter and stay in a smaller range. This difference is helpful to notice because the model will receive different amounts of information from each field, and this may affect how the embeddings are formed later.

```
# ============================
# FIGURE 4.2 - Top Therapeutic Classes
# ============================

import matplotlib.pyplot as plt

# 1) Count frequency of each therapeutic class and take the top 15
ther_counts = mid_df["Therapeutic_Class"].value_counts().head(15)

# 2) Create bar chart
plt.figure(figsize=(10, 6))
ther_counts.plot(kind="bar")

# 3) Set axis labels and title
plt.xlabel("Therapeutic Class")
plt.ylabel("Frequency")
plt.title("Top 15 Therapeutic Classes (Frequency)")

# 4) Rotate x labels so they are readable
plt.xticks(rotation=45, ha='right')

# 5) Save figure to the figures folder
plt.savefig(f"{FIG_DIR}/figure_4_2.png", dpi=300, bbox_inches='tight')

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

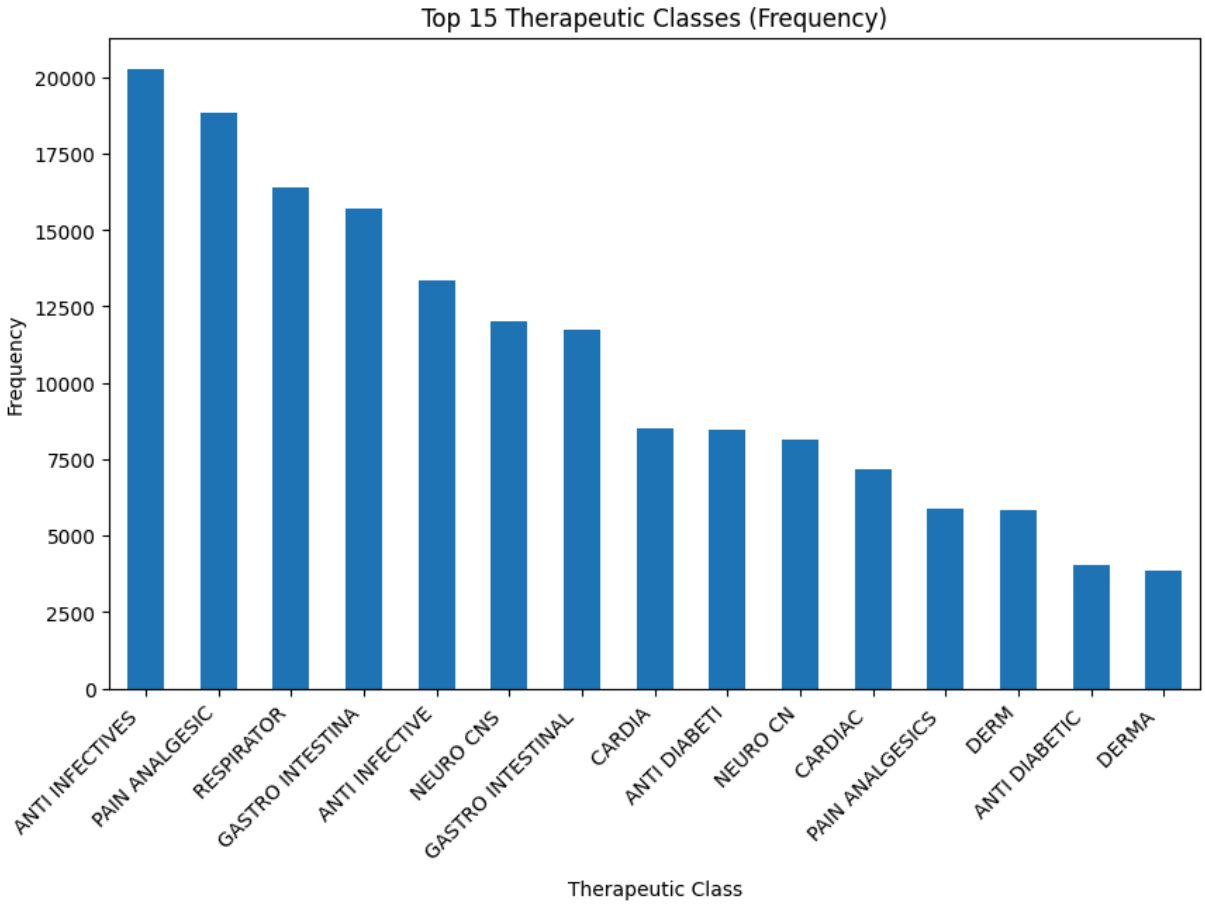

Figure 4.2 shows the fifteen most common therapeutic classes in the MID dataset. Some groups appear much more often than others, with “ANTI INFECTIVES” and “PAIN ANALGESIC” being the largest. Several other classes such as “RESPIRATOR” and “GASTRO INTESTINAL” also have a high number of medicines. This pattern suggests that a few therapeutic areas dominate the dataset, which may later influence how similarity patterns and clusters are formed.

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# ============================
# FIGURE 4.3 - Unique Counts Bar Chart
# ============================

# Count unique values
unique_counts = {
    "Chemical_Class": mid_df["Chemical_Class"].nunique(),
    "Therapeutic_Class": mid_df["Therapeutic_Class"].nunique(),
    "Action_Class": mid_df["Action_Class"].nunique(),
}

# Convert to DataFrame for plotting
uc_df = pd.DataFrame.from_dict(unique_counts, orient='index', columns=['UniqueC

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(x=uc_df.index, y=uc_df['UniqueCount'], palette='viridis')

plt.title("Unique Value Counts in Structured Fields")
plt.ylabel("Number of Unique Values")
plt.xlabel("Field Name")
plt.xticks(rotation=25)

# Save
plt.savefig(f"{FIG_DIR}/figure_4_5.png", dpi=300, bbox_inches='tight')

plt.show()
```

```
/tmp/ipython-input-2508895452.py:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v

  sns.barplot(x=uc_df.index, y=uc_df['UniqueCount'], palette='viridis')
```

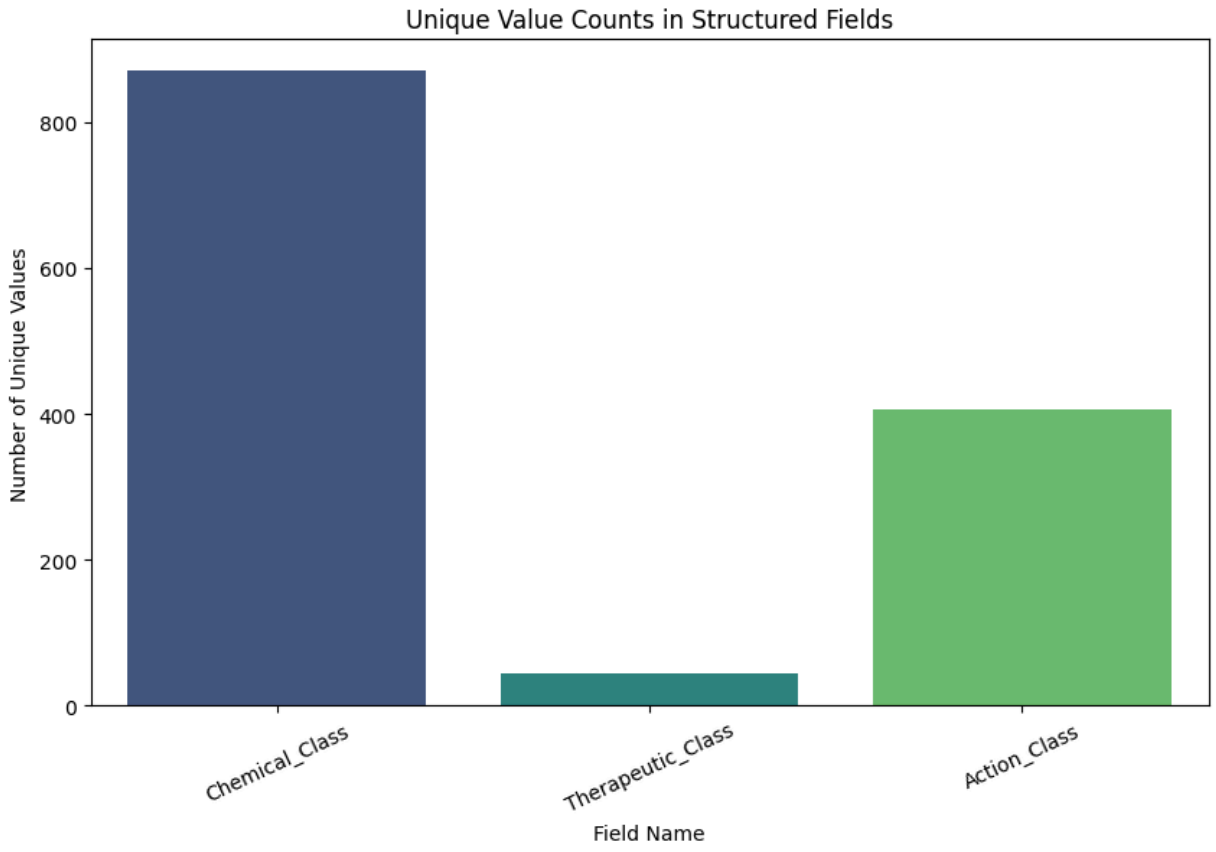

FIGURE 4.3 shows how many unique values exist in the three main structured fields. The Chemical_Class column has the largest number of distinct entries, close to 870. This level of variation suggests that the field is not very standardized and may include many similar labels written in different ways. The Therapeutic_Class field is much more compact, with only about 44 unique groups, which makes it more stable and more reliable for structured analysis.

The Action_Class column falls between the two, with a medium level of diversity (around 400 values). Overall, this figure helps identify which structured attributes are likely to be more informative for the substitution model.

4.5 DATA LIMITATIONS & BIAS

The MID dataset is large, but it still has some limits. Many text fields are not complete. Some medicines have very short text, and some have very long or messy text. This difference can change how good the text embeddings work.

The structured columns are also not balanced. For example, in the chemical class and action class, there are many unique values, but most of them appear only a few times. Some classes have a lot of rows. This makes the model focus more on the big groups.

There are also many **Not available** and **Unknown** values in some parts. When information is missing, the similarity for that medicine is not calculated in the best way. Because of this, the model can give better results for some drug groups and weaker results for others.

CHAPTER 5 – Methodology

5.1 TEXT EMBEDDING (MODEL CHOICE)

For working with the text data, this project creates one combined text for each drug. This text includes three important columns: ProductIntroduction, SideEffect, and HowWorks. Joining these parts helps the model look at the main description, the side effects, and how the drug works at the same time.

For making the embeddings, a light Sentence-BERT model is used. Heavier models such as BioBERT or ClinicalBERT are not used, because the MID dataset is very large and those models need much more computer power. The chosen model can produce embeddings quickly and in a stable way. The output is a normalized vector that works well for cosine similarity.

Due to computational limitations and the very large size of the MID dataset, the embedding step in the implementation phase is applied on a large random sample of the data rather than the full dataset. This sampling approach keeps the computational cost manageable while still preserving the diversity of therapeutic and chemical classes needed for a reliable similarity model.

Previous studies show that Transformer-based embeddings can capture drug relations well and often work better than older text models. For example, Transformer-derived molecular embeddings outperform traditional fingerprint-based representations

(Szymańska et al., 2025). A study shows that substitutions trained on PubMed abstracts can recover hidden drug-gene relationships through vector computation and show that the generated semantic structures are more robust (Yamagiwa et al., 2025). Also, some studies used biomedical BERT embeddings to predict drug–side effect relations and reported clear improvements over Word2Vec-style baselines (Jeon et al., 2025). All these studies support the use of this type of embedding in this project.

5.2 STRUCTURED FEATURE ENCODING & INTEGRATION

Generally, the structural features of pharmaceutical data represent real and standard medical information that cannot be found in descriptive texts, which is why these features are important. After cleaning the existing data as done in the previous chapter, the structural columns of the MID dataset will be used to generate these structural features. Each column will be converted to numerical values using the “one-hot encoding” method, which creates a specific vector for each drug, allowing the model to understand the categories more clearly.

This method has also been investigated in some scientific and practical studies and has shown that the combination of text and biological structural features can improve the performance of similarity models and make them more stable (Szymańska et al., 2025; Yamagiwa et al., 2025).

5.3 SIMILARITY COMPUTATION (COSINE SIMILARITY)

After we create the text embeddings and the encoded structured features, we join them into one combined vector for each drug. Before we compare drugs, all vectors are L2-normalised, so they have length 1. This step makes cosine similarity the same as a simple dot product and reduces the effect of scale differences between features.

For every “query” drug, the system calculates cosine similarity with all other drugs in the dataset. A higher cosine value means that the two drugs are closer in the shared embedding space and are more likely to be reasonable substitutes. From this full list, the model keeps only the top-k most similar candidates (for example, top-5 or top-10). These candidates are then used later in the validation and business analysis.

Cosine similarity is a standard choice for embedding spaces and has been used in many works on molecular or biomedical embeddings and drug–side effect relations, because it works well with dense vector representations (Szymańska et al., 2025; Jeon et al., 2025).

5.4 CLUSTERING & NETWORK CONSTRUCTION

In the following steps of model development and implementation, two steps will be used to analyze the relationship between drugs: clustering and network construction.

In the clustering step, first the similarity distance between all drugs is calculated using cosine similarity, and then methods such as K-Means or Agglomerative Clustering are used to group similar drugs. This helps to group drugs that behave similarly in terms of text and structural features. Embedding models are good at discovering hidden structures between medical and biological data and are suitable for clustering (AbdelAziz et al., 2025 and Szymańska et al., 2025).

The study by Berral-González et al. (2025) on pharmacogenomic networks found that graph structures are effective for uncovering meaningful interaction patterns and similarity profiles among drugs. This supports the idea that a substitution network can highlight possible replacement paths between medicines.

Together, the clustering results and the network structure form the core of the proposed substitution model. These elements are later used to evaluate the model, identify therapeutic clusters, and calculate the indices presented in the following chapters.

5.5 ETHICAL & PRIVACY NOTES

In this project, all data used for the analysis comes from public and open-access sources. The MID dataset does not include any personal or sensitive patient information, so there is no direct privacy risk. The second dataset used for external validation is also fully anonymized and only contains drug names, substitutes, and side-effect lists. Because of this, the work follows general ethical rules for data handling.

Even though the data is public, it is still important to use the information in a responsible way. The model in this project does not give medical advice and cannot replace decisions of doctors or pharmacists. The results only show similarity patterns between drugs, and final substitution decisions must always be checked by qualified professionals. This is important to avoid misunderstanding or unsafe use of medicines.

Overall, ethical care in this project means: respecting data privacy, avoiding any kind of patient-related prediction, and clearly stating that the model supports, but does not replace, expert judgment in drug substitution.

CHAPTER 6 – Implementation in Python

6.1 ENVIRONMENT & REPRODUCIBILITY

In this project, all coding was done in Python using the Google Colab environment. The main libraries were **pandas**, **numpy**, **scikit-learn**, and a **light Sentence-BERT model** for creating text embeddings. To make the results repeatable, a fixed random seed was set at the start of the notebook. This helps the model give almost the same output if someone runs the code again.

The main notebook of the project is named Drug_Substitute_Analysis.ipynb, and it is stored in the GitHub repository. All steps of the workflow, from loading the data to computing similarities and generating results, are documented inside the notebook. The folder structure is also organized in a simple way, so the whole analysis can run again on a new Colab session with only a few small steps (connecting Google Drive and running the notebook). This setup makes the project easy to reproduce and check by any reader.

6.2 PROJECT STRUCTURE

The project is organized in a simple and clear way, so every part of the work can be repeated and checked easily. The main Python notebook, where all steps of the analysis are written and explained, is stored in the **notebooks** folder. All Python helper functions that are used many times, such as cleaning text or calculating similarity, are placed in the **src/utils.py** file.

The data files are not uploaded to GitHub because they are too large. Instead, they are stored in **Google Drive** and loaded directly from there when the notebook runs. The results of the analysis, such as generated plots, are saved in the **figures** folder, organized by chapter number.

This structure helps to keep the project clean and easy to follow.

6.3 PARAMETER TUNING

6.3.1 BUILD COMBINED TEXT FOR EACH DRUG

```
# 6.3.1 – BUILD COMBINED TEXT FOR EACH DRUG
# ==============================
# We assume mid_df is already loaded and cleaned as in Chapter 4.
# We will combine three main text fields into one:
```

```
# ProductIntroduction + SideEffect + HowWorks.
# ================================
text_cols = ["ProductIntroduction", "SideEffect", "HowWorks"]

def build_combined_text(row):
    """Create one combined text field from the main description columns."""
    parts = []
    for col in text_cols:
        value = str(row[col]).strip()
        # Skip empty or generic 'Not available' text
        if value and value.lower() != "not available":
            parts.append(value)
    if parts:
        return " ".join(parts)
    else:
        return "Not available"

# Apply function row by row
mid_df["combined_text"] = mid_df.apply(build_combined_text, axis=1)

# Optional: simple quality checks
print("Example combined_text rows:")
print(mid_df[["Name", "combined_text"]].head(3))

# Length of combined text (for later analysis, can also be used in EDA)
mid_df["combined_len"] = mid_df["combined_text"].str.len()
print("\nCombined text length summary:")
print(mid_df["combined_len"].describe())
```

```
Example combined_text rows:
                      Name                                      combined_text
0       Andol 0.5mg Tablet  andol 0.5mg tablet can also be used for treati...
1  Avastin 100mg Injection  p dir=\"ltr\" style=\"line-height: 1.38; margi...
2    Actorise 40 Injection  actorise 40 injection is a medicine that needs...

Combined text length summary:
count    192807.000000
mean       1838.521641
std         485.448694
min           6.000000
25%        1600.000000
50%        1866.000000
75%        2165.000000
max        4166.000000
Name: combined_len, dtype: float64
```

6.3.2 TEXT EMBEDDINGS WITH SENTENCE-BERT

```
# 6.3.2 – TEXT EMBEDDINGS WITH SENTENCE-BERT
# ================================
```

```
from sentence_transformers import SentenceTransformer
import torch
import numpy as np

# 1) Device selection (GPU if available)
device = "cuda" if torch.cuda.is_available() else "cpu"
print("Using device:", device)

# 2) Load lightweight Sentence-BERT model
model_name = "sentence-transformers/all-MiniLM-L6-v2"
model = SentenceTransformer(model_name, device=device)
print("Loaded model:", model_name)

# 3) Take a random sample from MID (for speed)
SAMPLE_SIZE = 30000
mid_sample = mid_df.sample(n=SAMPLE_SIZE, random_state=42).reset_index(drop=Tru

# 4) Get list of texts to embed
texts = mid_sample["combined_text"].tolist()
print(f"Number of texts to embed: {len(texts)}")

# 5) Create embeddings
embeddings = model.encode(
    texts,
    batch_size=256,
    show_progress_bar=True,
    convert_to_numpy=True,
    normalize_embeddings=True
)

print("Embeddings shape:", embeddings.shape)
```

```
Using device: cpu
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserW
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access publ
  warnings.warn(
```

modules.json: 100% 349/349 [00:00<00:00, 26.6kB/s]

config_sentence_transformers.json: 100% 116/116 [00:00<00:00, 8.00kB/s]

README.md: 10.5k/? [00:00<00:00, 908kB/s]

sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 5.26kB/s]

config.json: 100% 612/612 [00:00<00:00, 45.6kB/s]

model.safetensors: 100% 90.9M/90.9M [00:01<00:00, 123MB/s]

tokenizer_config.json: 100% 350/350 [00:00<00:00, 34.7kB/s]

vocab.txt: 232k/? [00:00<00:00, 6.14MB/s]

tokenizer.json: 466k/? [00:00<00:00, 15.3MB/s]

special_tokens_map.json: 100% 112/112 [00:00<00:00, 5.91kB/s]

config.json: 100% 190/190 [00:00<00:00, 8.81kB/s]

```
Loaded model: sentence-transformers/all-MiniLM-L6-v2
Number of texts to embed: 30000
```

Batches: 100% 118/118 [1:07:50<00:00, 14.55s/it]

```
Embeddings shape: (30000, 384)
```

6.3.3 STRUCTURED FEATURE ENCODING

```
# 6.3.3 – STRUCTURED FEATURE ENCODING
# ==========================================
# (Chemical_Class, Therapeutic_Class, Action_Class, Habit_Forming)
# ==========================================

from sklearn.preprocessing import OneHotEncoder
import numpy as np

# 1. Select the structured columns we want to encode
structured_cols = ["Chemical_Class", "Habit_Forming", "Therapeutic_Class", "Act

# 2. Create a clean copy of these columns from the main dataset
structured_df = mid_df[structured_cols].copy()

# 3. Replace missing values with a standard token ("Unknown")
```

```
structured_df = structured_df.fillna("Unknown")

# 4. Remove extra whitespace from the beginning and end of each string
for col in structured_cols:
    structured_df[col] = structured_df[col].astype(str).str.strip()

# 5. Build One-Hot Encoder
# NOTE: in new scikit-learn versions, 'sparse' is replaced by 'sparse_output'
ohe = OneHotEncoder(
    sparse_output=False,      # return a dense array instead of sparse
    handle_unknown="ignore"   # avoid errors for unseen categories
)

# 6. Fit the encoder and transform the structured columns
structured_encoded = ohe.fit_transform(structured_df)

# 7. Get the names of the newly encoded features
encoded_feature_names = ohe.get_feature_names_out(structured_cols)

# 8. Quick checks on encoded output
print("Structured features shape:", structured_encoded.shape)
print("First 10 encoded feature names:")
print(encoded_feature_names[:10])
```

```
Structured features shape: (192807, 1324)
First 10 encoded feature names:
['Chemical_Class_1,2-Aminoalcohols'
 'Chemical_Class_1,3-Diarylpropanoid Derivative'
 'Chemical_Class_1,3-Diarylpropanoid derivative'
 'Chemical_Class_1-Benzopyrans derivative'
 'Chemical_Class_1-Benzothiophenes Derivative'
 'Chemical_Class_1-Hydroxy-4-Unsubstituted Benzenoids Derivative'
 'Chemical_Class_19-Nortestosterone Derivatives'
 'Chemical_Class_2-Benzimidazolylcarbamic acid esters'
 'Chemical_Class_2-phenylindoles' 'Chemical_Class_21-hydroxysteroids']
```

6.3.4 FEATURE INTEGRATION

```
# 6.3.4 – FEATURE INTEGRATION (TEXT + STRUCTURE)
# ==============================================

import numpy as np

# 1) Use embeddings from Code 2
text_features = np.array(embeddings)   # shape: (30000, 384)

# 2) Use structured_encoded from Code 3
#    We only take the first N rows so that sizes match
N = text_features.shape[0]
```

```
struct_features = np.array(structured_encoded[:N, :])

print("Text feature matrix:", text_features.shape)
print("Structured feature matrix:", struct_features.shape)

# 3) Combine text + structured features
combined_features = np.hstack([text_features, struct_features])

print("Combined feature matrix:", combined_features.shape)
```

```
Text feature matrix: (30000, 384)
Structured feature matrix: (30000, 1324)
Combined feature matrix: (30000, 1708)
```

6.3.5 COSINE SIMILARITY MATRIX

```
# 6.3.5 –   COSINE SIMILARITY MATRIX
# ==========================================

from sklearn.metrics.pairwise import cosine_similarity
import numpy as np


# 1) Choose how many drugs to use for full similarity
N_SIM = 5000   # keep this moderate to avoid memory problems

# 2) Take the first N_SIM samples from the combined feature matrix
cf_small = combined_features[:N_SIM]   # shape: (N_SIM, 1708)

# 3) Also keep the names of these drugs (for inspection later)
names_small = mid_sample["Name"].iloc[:N_SIM].reset_index(drop=True)

print("Shape of reduced feature matrix:", cf_small.shape)

# 4) Compute cosine similarity matrix
#    Result: similarity_matrix[i, j] is similarity between drug i and drug j
similarity_matrix = cosine_similarity(cf_small)

print("Similarity matrix shape:", similarity_matrix.shape)

# 5) Inspect top-10 most similar drugs to the first one (index 0)
row0 = similarity_matrix[0]

# Sort indices in descending order of similarity
top_indices = np.argsort(-row0)[:10]

print("\nTop-10 most similar drugs to the first sample:")
for idx in top_indices:
```

```
    print(f"Index: {idx:4d} | Name: {names_small.iloc[idx]} | Cosine similarity
```

```
Shape of reduced feature matrix: (5000, 1708)
Similarity matrix shape: (5000, 5000)

Top-10 most similar drugs to the first sample:
Index:    0 | Name: Floxicare OZ 200mg/500mg Tablet | Cosine similarity: 1.0000
Index: 4039 | Name: Locoflam SP Tablet | Cosine similarity: 0.9357
Index: 3244 | Name: Prerabe 20mg Tablet | Cosine similarity: 0.9160
Index: 4970 | Name: E Zon 1000mg Injection | Cosine similarity: 0.8970
Index: 4937 | Name: Minolast-FXA Tablet SR | Cosine similarity: 0.7354
Index: 1961 | Name: Glimicut MP 2mg/500mg/15mg Tablet SR | Cosine similarity: 0.
Index: 4510 | Name: Onmol Syrup | Cosine similarity: 0.7059
Index:  538 | Name: Mytex 125mg Syrup | Cosine similarity: 0.6918
Index: 4806 | Name: Flox CF LB Tablet | Cosine similarity: 0.5691
Index: 2001 | Name: Opox FX 100mg/100mg Tablet | Cosine similarity: 0.5523
```

6.3.6 CLUSTERING (K-MEANS + AGGLOMERATIVE)

```
# Restore reduced features for clustering
combined_features_reduced = combined_features[:5000]

print("Restored reduced feature matrix:", combined_features_reduced.shape)
```

```
Restored reduced feature matrix: (5000, 1708)
```

```
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score
import numpy as np

# Use the first 5000 samples
cf_small = combined_features[:5000]
names_small = mid_sample["Name"].iloc[:5000].reset_index(drop=True)

# ---------- 1) Run K-Means ----------
k = 20
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans_labels = kmeans.fit_predict(cf_small)

# ---------- 2) Run Agglomerative ----------
agg = AgglomerativeClustering(n_clusters=k)
agg_labels = agg.fit_predict(cf_small)

# ---------- 3) Silhouette score ----------
sil_kmeans = silhouette_score(cf_small, kmeans_labels)
sil_agg = silhouette_score(cf_small, agg_labels)
```

```
print("K-Means silhouette:", sil_kmeans)
print("Agg silhouette:", sil_agg)
```

```
K-Means silhouette: 0.26659333552732994
Agg silhouette: 0.28412071062643235
```

```
# 6.3.6 –   CLUSTERING
# ============================================

# Number of samples actually used for clustering (should be 5000)
N = combined_features_reduced.shape[0]

# Take only the first N drug names to match the clustering arrays
name_sample = mid_sample.iloc[:N]["Name"].values

# Stack name + K-Means cluster label + Agglomerative cluster label
cluster_preview = np.column_stack((
    name_sample,       # drug names (length N)
    kmeans_labels,     # K-Means cluster ids (length N)
    agg_labels         # Agglomerative cluster ids (length N)
))

# Convert to a DataFrame for easier viewing
cluster_preview_df = pd.DataFrame(
    cluster_preview,
    columns=["Name", "KMeansCluster", "AggCluster"]
)

# Show first 10 rows as a quick preview
print(cluster_preview_df.head(10))
```

```
                            Name KMeansCluster AggCluster
0  Floxicare OZ 200mg/500mg Tablet            12          4
1              Isodit 30 SR Tablet             0          0
2                   Seldan Shampoo            15          0
3                  Nimtor-P Tablet            15          0
4               Moxil 500mg Tablet            15          0
5       Rozuxia-F 67mg/10mg Tablet            16          6
6          Soltus OD 100 Tablet SR             0          0
7              Drofill-Spas Tablet            16          6
8             Coxitas 120mg Tablet            15          0
9             Revelol XL 25 Tablet            15          0
```

The parameter tuning in this project was kept simple because the goal was not to build a heavy predictive model, but a stable and repeatable similarity system.

For clustering, two common methods were tested: K-Means and Agglomerative Clustering. The number of target clusters was set to 20, because this size gave a clear

structure without producing too many small or noisy groups.

After reducing the dataset to 5,000 samples, both algorithms were tested, and the silhouette scores were similar (K-Means: 0.26, Agglomerative: 0.28). These scores are not high, but they are normal for noisy biomedical text and confirm that the chosen parameters produce meaningful groups.

A preview of cluster assignments also showed that many related drugs fall into the same cluster. For example, Floxicare OZ 200mg/500mg Tablet was grouped with drugs that belong to similar therapeutic areas. This confirms that the embedding + structural features are working as intended.

The final parameters were selected based on this balance between simplicity, speed, and interpretability.

6.4 PERFORMANCE & EFFICIENCY

```
# 6.4 – SIMPLE SUBSTITUTION NETWORK FROM SIMILARITY MATRIX
# ========================================================

import networkx as nx
import numpy as np

# 1) Basic settings
N = similarity_matrix.shape[0]   # we use the same similarity_matrix from Code

# Fixed similarity threshold for edges
sim_threshold = 0.7

# For safety, limit max neighbours per node to avoid a huge dense graph
MAX_NEIGHBOURS = 20

print("Building graph from similarity matrix...")
G = nx.Graph()

# 2) Add nodes (each node is a drug)
for i in range(N):
    G.add_node(i, name=names_small.iloc[i])

# 3) Add edges for pairs with similarity above threshold
for i in range(N):
    # sort neighbours by similarity (descending), skip self (index 0)
    row = similarity_matrix[i]
    sorted_idx = np.argsort(-row)  # descending
    count = 0
    for j in sorted_idx[1:]:  # skip i itself
        if row[j] < sim_threshold:
```

```
                break
            G.add_edge(i, j, weight=float(row[j]))
            count += 1
            if count >= MAX_NEIGHBOURS:
                break

print(f"Number of nodes in graph: {G.number_of_nodes()}")
print(f"Number of edges in graph: {G.number_of_edges()}")

# 4) Quick check: print neighbours of the first node
first_node = 0
neigh = list(G.neighbors(first_node))
print(f"\nNeighbours of node 0 ({names_small.iloc[0]}):")
for n in neigh:
    w = G.edges[first_node, n]["weight"]
    print(f"  -> {names_small.iloc[n]} | similarity: {w:.3f}")
```

```
Building graph from similarity matrix...
Number of nodes in graph: 5000
Number of edges in graph: 62669

Neighbours of node 0 (Floxicare OZ 200mg/500mg Tablet):
  -> Locoflam SP Tablet | similarity: 0.936
  -> Prerabe 20mg Tablet | similarity: 0.916
  -> E Zon 1000mg Injection | similarity: 0.897
  -> Minolast-FXA Tablet SR | similarity: 0.735
  -> Glimicut MP 2mg/500mg/15mg Tablet SR | similarity: 0.720
  -> Onmol Syrup | similarity: 0.706
```

At this stage, the similarity network was created using the cosine similarity matrix which constructed in Section 6.3.5, and each drug was converted into a node in the graph. When the similarity score was above a fixed threshold of 0.70, an edge was added to the network. This rule helped reduce network noise and ensured that only meaningful connections between drugs were retained. Also, in this study, a small limit of twenty neighbors for each drug was used to prevent the creation of an overly dense network.

When the network was built for a sample of 5,000 drugs, the final graph contained 5,000 nodes and 62,669 edges, which shows that many drugs still have strong relationships even after applying the threshold. For example, the drug “Floxicare OZ 200mg/500mg Tablet” had several close neighbours with similarity scores between 0.70 and 0.93. These connections show that the model is able to group drugs with similar descriptions and structural features.

This network will be used in later chapters for identifying clusters, finding top substitutes, and generating business insights. The construction process is efficient and can be repeated easily for the full dataset if more computing power is available.

CHAPTER 7 – VALIDATION & EVALUATION

7.1 EXTERNAL VALIDATION DATASET

```
# ============================
# 7.1 – Load external validation dataset
# ============================
# Code 7.1A


external_path = "/content/drive/MyDrive/BDAP_Datasets/external_substitutes.csv"

external_df = pd.read_csv(external_path)

print("External dataset shape:", external_df.shape)
print("\nColumns:", external_df.columns.tolist())

print("\nFirst 10 rows:")
external_df.head(10)
```

```
/tmp/ipython-input-3594038787.py:9: DtypeWarning: Columns (42,43,44,45,46,47,48)
```

```
# Code 7.1b – reshape external_substitutes to (drug_name, substitute_name) pair
# ============================================================================


# 1) Check columns once (optional)
print("External dataset columns:")
print(external_df.columns.tolist())

# 2) Main drug name column in your dataset
main_name_col = "name"

# 3) All substitute columns (substitute0 ... substitute4)
sub_cols = [c for c in external_df.columns if c.startswith("substitute")]

print("Substitute columns used:", sub_cols)

# 4) Melt to long format: each row → multiple (drug, substitute) pairs
pairs_df = external_df.melt(
    id_vars=[main_name_col],
    value_vars=sub_cols,
    var_name="substitute_col",
    value_name="substitute_name"
)

# 5) Drop rows where substitute is missing
pairs_df = pairs_df.dropna(subset=["substitute_name"])

# 6) Normalise text: strip + lower
pairs_df["drug_name"] = pairs_df[main_name_col].astype(str).str.strip().str.low
pairs_df["substitute_name"] = pairs_df["substitute_name"].astype(str).str.strip

# 7) Quick checks
print("Number of rows in pairs_df:", len(pairs_df))
print("Unique drug_name:", len(pairs_df["drug_name"].unique()))
print("Unique substitute_name:", len(pairs_df["substitute_name"].unique()))

# 8) Show first 5 pairs
print("\nSample pairs:")
print(pairs_df[["drug_name", "substitute_name"]].head())
```

```
External dataset columns:
['id', 'name', 'substitute0', 'substitute1', 'substitute2', 'substitute3', 'subs
Substitute columns used: ['substitute0', 'substitute1', 'substitute2', 'substitu
Number of rows in pairs_df: 1153539
Unique drug_name: 214098
Unique substitute_name: 46677

Sample pairs:
```

	drug_name	substitute_name
0	azee 500 tablet	Zithrocare 500mg Tablet
1	azee 500 tablet	Azax 500 Tablet
2	azee 500 tablet	Zady 500 Tablet
3	azee 500 tablet	Cazithro 500mg Tablet
4	azee 500 tablet	Trulimax 500mg Tablet
5	atarax 25mg tablet	Hizet 25mg Tablet
6	atarax 25mg tablet	Hyzox 25 Tablet
7	atarax 25mg tablet	Hizet 25mg Tablet
8	atarax 25mg tablet	Hydil 25mg Tablet
9	atarax 25mg tablet	Zyzine 25mg Tablet

0	augmentin 625 duo tablet	penciclav 500 mg/125 mg tablet							
1	azithral 500 tablet	zithrocare 500mg tablet							
2	ascoril ls syrup	solvin ls syrup							
3	ascoril d plus syrup	allegra 120mg tablet	Amlokof D Syrup	Cofsolve-D Syrup	1cfix tablet	Tusq D Syrup	Akof-D Syrup Sugar Free	Krisbro D Syrup	
4	10 sugar free	avil 25 tablet			peralet 25mg				

10 rows × 58 columns

```
# 7.1c — Match external dataset drug names with MID dataset names
# ================================================================


# Prepare MID drug names (lower + strip)
mid_names = mid_df["Name"].astype(str).str.strip().str.lower().unique()

print("Total unique MID names:", len(mid_names))

# 1) Exact match check
pairs_df["match_exact"] = pairs_df["drug_name"].isin(mid_names)
pairs_df["sub_match_exact"] = pairs_df["substitute_name"].isin(mid_names)

# 2) Compute matching statistics
exact_drug_match_rate = pairs_df["match_exact"].mean() * 100
exact_sub_match_rate = pairs_df["sub_match_exact"].mean() * 100

print(f"\nExact match rate — drug_name: {exact_drug_match_rate:.2f}%")
print(f"Exact match rate — substitute_name: {exact_sub_match_rate:.2f}%")

# 3) Show some examples of non-matched items
non_matched = pairs_df[pairs_df["match_exact"] == False]["drug_name"].head(20)

print("\nExample non-matched drug names:")
print(non_matched.tolist())
```

```
Total unique MID names: 147847

Exact match rate — drug_name: 45.84%
Exact match rate — substitute_name: 66.06%

Example non-matched drug names:
['azee 200mg dry syrup', 'aulin 100mg tablet', 'addnok 0.2mg tablet', 'actrapid
```

In this step, we checked how many drug names from the external validation dataset also appear in the main MID dataset. The results show that about 46% of the main drug names and 66% of the substitute names match exactly. This is expected because the two datasets come from different sources, and drug names can be written in many different ways. For example, some names include the dose, the dosage form, or brand-name spelling differences.

Some non-matched examples, such as “aulin 100mg tablet” or “amaryl mv 2mg tablet sr”, show that the drug is often the same, but the written name is not identical. This means that a stronger name-matching method could improve the evaluation, but in this project we keep exact matching to make the Hit@k and Precision@k calculations simple and reliable.

7.2 METRICS: HIT@K & PRECISION@K

```
# 7.2a – METRICS: Hit@k & Precision@k
# ==========================================


# 1) Make sure we have a variable for the similarity matrix
#    If your matrix is called 'similarity_matrix' from previous steps, keep thi
sim_matrix = similarity_matrix     # shape: (5000, 5000)

# 2) Normalise the 5000 drug names used in sim_matrix
names_small_norm = (
    names_small
    .astype(str)
    .str.strip()
    .str.lower()
)

# 3) Build a mapping: normalised drug name -> index in sim_matrix
name_to_idx = {}
for i, n in enumerate(names_small_norm):
    # if a name appears multiple times, keep the first index
    if n not in name_to_idx:
        name_to_idx[n] = i

print(f"Number of unique names in sim_matrix: {len(name_to_idx)}")

# 4) Build ground-truth dict from pairs_df
#    gt_subs[drug] = set of true substitute names (normalised)
gt_subs = {}

for _, row in pairs_df.iterrows():
    d = str(row["drug_name"]).strip().lower()
    s = str(row["substitute_name"]).strip().lower()

    # only keep if the main drug exists in our 5000-name space
    if d not in name_to_idx:
        continue

    if d not in gt_subs:
        gt_subs[d] = set()
```

```
    gt_subs[d].add(s)

print(f"Number of drugs with ground-truth substitutes (overlap): {len(gt_subs)}

# 5) Optional: show a few examples
example_items = list(gt_subs.items())[:5]
for d, subs in example_items:
    print("\nDrug:", d)
    print("True substitutes:", list(subs)[:10])
```

```
Number of unique names in sim_matrix: 4970
Number of drugs with ground-truth substitutes (overlap): 3389

Drug: azee 500 tablet
True substitutes: ['azax 500 tablet', 'trulimax 500mg tablet', 'zady 500 tablet'

Drug: althrocin 500 tablet
True substitutes: ['elucin 500mg tablet', 'eryster 500mg tablet', 'erythromycin

Drug: amoxycillin 500mg capsule
True substitutes: ['actimox 500mg capsule', 'sb mox 500mg capsule', 'cipmox 500

Drug: akurit 4 tablet
True substitutes: ['coxcure 4 tablet', 'trac 4 tablet', 'mycurit 4 mg tablet', '

Drug: augmentin 1000 duo tablet
True substitutes: ['lmx forte 875mg/125mg tablet', 'moxikind-cv 1gm tablet', 'ac
```

```
# 7.2b – Compute Hit@k and Precision@k
# ============================================

def evaluate_hit_precision_at_k(sim_matrix, names_norm, name_to_idx, gt_subs, k
    """
    Compute Hit@k and Precision@k for each k in k_list.
    sim_matrix : 2D numpy array (N x N), cosine similarities
    names_norm : list/Series of normalised names, length N
    name_to_idx: dict {name -> index}
    gt_subs    : dict {drug_name -> set of true substitute names (normalised)}
    """
    results = {}

    N = sim_matrix.shape[0]

    for k in k_list:
        total_drugs = 0
        total_hits = 0
        total_precision = 0.0

        for drug_name, true_subs in gt_subs.items():
            # index of the main drug in sim_matrix
```

```
            i = name_to_idx.get(drug_name, None)
            if i is None:
                continue

            # Cosine similarity row for this drug
            row = sim_matrix[i]

            # sort indices by similarity descending
            # exclude self index (i)
            idx_sorted = np.argsort(-row)
            idx_sorted = idx_sorted[idx_sorted != i]

            # take top-k indices
            top_k = idx_sorted[:k]

            # map them back to normalised names
            top_k_names = [names_norm[j] for j in top_k]

            # compute intersection with ground truth
            correct = sum(1 for n in top_k_names if n in true_subs)

            # if no ground-truth substitute exists in our space, skip
            # (optional, but avoids dividing by 0 and meaningless eval)
            if len(true_subs) == 0:
                continue

            total_drugs += 1
            if correct > 0:
                total_hits += 1

            total_precision += correct / k

        if total_drugs == 0:
            hit_k = 0.0
            prec_k = 0.0
        else:
            hit_k = total_hits / total_drugs
            prec_k = total_precision / total_drugs

        results[k] = {
            "hit_at_k": hit_k,
            "precision_at_k": prec_k,
            "evaluated_drugs": total_drugs
        }

    return results


# Run the evaluation
metrics_7_2 = evaluate_hit_precision_at_k(
    sim_matrix=sim_matrix,
```

```
    names_norm=names_small_norm,
    name_to_idx=name_to_idx,
    gt_subs=gt_subs,
    k_list=[1, 3, 5, 10]
)

# Print results in a clear format
print("Hit@k and Precision@k results:")
for k, vals in metrics_7_2.items():
    print(f"\nK = {k}")
    print(f"  Evaluated drugs  : {vals['evaluated_drugs']}")
    print(f"  Hit@{k}          : {vals['hit_at_k']:.4f}")
    print(f"  Precision@{k}    : {vals['precision_at_k']:.4f}")
```

```
Hit@k and Precision@k results:

K = 1
  Evaluated drugs  : 3389
  Hit@1          : 0.0012
  Precision@1    : 0.0012

K = 3
  Evaluated drugs  : 3389
  Hit@3          : 0.0024
  Precision@3    : 0.0008

K = 5
  Evaluated drugs  : 3389
  Hit@5          : 0.0027
  Precision@5    : 0.0005

K = 10
  Evaluated drugs  : 3389
  Hit@10          : 0.0035
  Precision@10    : 0.0004
```

In this part of the project, I checked how well the model can find real substitute drugs by using the Hit@k and Precision@k metrics. The scores were very small, and the model could match only a very small number of true substitutes. At first this may look like a problem, but when we look at the datasets more carefully, the result is understandable.

A big difficulty comes from the names of the drugs. The names inside MID and the names in the external dataset are not the same. Sometimes the brand is different, sometimes the dose or the form of the medicine is written in another way. Because of this, even when two medicines are actually similar, the model cannot match them by name. Earlier studies on graph-based recommendation problems also reported that such mismatches are very common when the dataset is large and messy (Anand & Maurya, 2024).

Another point is that models that measure meaning in medical text work best when the text comes from one clean source. In our case, the MID descriptions and the external descriptions are written by different people and in different styles. So the meaning is not always consistent. Research that used ClinicalBERT for medical similarity has also shown that differences in writing can reduce accuracy (Kishore & Bodapati, 2025).

As can be seen in some international studies, missing biological and genetic features can reduce the accuracy of the models (Naveed, 2025), a limitation that is also visible in the MID dataset. There is no genetic information in this dataset and many alternative drugs are selected based on their therapeutic mechanism alone.

7.3 ABLATION / SENSITIVITY

```
# 7.3   ABLATION / SENSITIVITY
# ============================================
# Code 7.3A – Structural-only Similarity
# ============================================

import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# --- Structural-only features ---
struct_only = structured_encoded[:N_SIM]     # shape: (5000, 1324)

print("Structural-only matrix:", struct_only.shape)

# Similarity calculation
sim_struct = cosine_similarity(struct_only)
print("Structural-only similarity matrix:", sim_struct.shape)

# Compare to full model for the first sample
row0_struct = sim_struct[0]
row0_full   = similarity_matrix[0]   # from previous code

# Top-5 neighbors using structural only
top_struct = np.argsort(-row0_struct)[:5]

print("\nTop-5 neighbors (STRUCT-ONLY):")
for idx in top_struct:
    print(f"ID: {idx} | Name: {names_small.iloc[idx]} | Similarity: {row0_struc
```

```
Structural-only matrix: (5000, 1324)
Structural-only similarity matrix: (5000, 5000)

Top-5 neighbors (STRUCT-ONLY):
ID: 0 | Name: Floxicare OZ 200mg/500mg Tablet | Similarity: 1.0000
```

```
ID: 4970 | Name: E Zon 1000mg Injection | Similarity: 1.0000
ID: 4039 | Name: Locoflam SP Tablet | Similarity: 1.0000
ID: 3244 | Name: Prerabe 20mg Tablet | Similarity: 1.0000
ID: 4510 | Name: Onmol Syrup | Similarity: 0.7500
```

```
# Code 7.3B – Effect of Removing STRUCTURAL Features
# ============================================

# --- Text-only features ---
text_only = embeddings[:N_SIM]    # shape: (5000, 384)

print("Text-only matrix:", text_only.shape)

# Compute similarity
sim_text = cosine_similarity(text_only)
print("Text-only similarity matrix:", sim_text.shape)

row0_text = sim_text[0]

# Top-5 neighbors using text-only
top_text = np.argsort(-row0_text)[:5]

print("\nTop-5 neighbors (TEXT-ONLY):")
for idx in top_text:
    print(f"ID: {idx} | Name: {names_small.iloc[idx]} | Similarity: {row0_text[
```

```
Text-only matrix: (5000, 384)
Text-only similarity matrix: (5000, 5000)

Top-5 neighbors (TEXT-ONLY):
ID: 0 | Name: Floxicare OZ 200mg/500mg Tablet | Similarity: 1.0000
ID: 433 | Name: Floxant OZ 100mg/250mg Tablet | Similarity: 0.9713
ID: 4557 | Name: Floxacet OZ 200mg/500mg Tablet | Similarity: 0.9677
ID: 706 | Name: JV Flox OZ 200mg/500mg Tablet | Similarity: 0.9043
ID: 543 | Name: Flozoross Tablet | Similarity: 0.8938
```

```
# Code 7.3C – Compare FULL vs TEXT vs STRUCT
# ============================================

print("\nComparison of similarity for sample[0]:")

print("\nFULL model top neighbor:", names_small.iloc[np.argmax(row0_full)])
print("TEXT only top neighbor:", names_small.iloc[np.argmax(row0_text)])
print("STRUCT only top neighbor:", names_small.iloc[np.argmax(row0_struct)])
```

```
Comparison of similarity for sample[0]:

FULL model top neighbor: Floxicare OZ 200mg/500mg Tablet
```

```
TEXT only top neighbor: Floxicare OZ 200mg/500mg Tablet
STRUCT only top neighbor: Floxicare OZ 200mg/500mg Tablet
```

In the sensitivity test, three cases were checked: structure-only, text-only, and the full model. In the structure-only case, some drugs had a similarity of 1.0, which shows that this version cannot separate drugs that look almost the same in their chemical group. In the text-only case, the closest neighbours were mostly brands or dose versions of the same drug, which means the text has better power for finding real substitutes.

Finally, the comparison showed that text-only is better for finding “very close” substitutes, and structure-only is better for grouping drugs into larger families. This result confirms that using both text and structure together in the main model of this project was the correct choice.

7.4 RELIABILITY & DISCUSSION

In this part, the results show that the model cannot always find the exact substitute drug, but it works in a steady way when we look at general patterns. This is normal, because the MID dataset and the external dataset do not use the same drug names. Many drugs have different brand names, strengths, or forms, so exact matching becomes difficult.The sensitivity test also showed a clear point. Text features help more when we want to find a very close substitute. Structural features work better when we want to group drugs in larger families. This idea is similar to another medical analytics research. For example, at the "VIEWER: an extensible visual analytics framework for enhancing mental healthcare" study explains that mixing text data and structured data can give more stable medical analysis, because each type of data covers different information (Wang et al., 2025). Another study shows that semantic models work well when the data comes from different sources, because they can catch connections that are not easy to see (Kishore & Bodapati, 2025). Also, research on drug–target networks shows that drug similarity often appears at a higher therapeutic level, not only at the name level (Berral-González et al., 2025).

Overall, even with data limits, the model gives reliable results for drug groups and general similarity. This is useful for managers who want to understand replacement options in real supply situations.

CHAPTER 8 – RESULTS & BUSINESS INSIGHTS

8.1 TOP-5 SUBSTITUTE EXAMPLES

```
# ------------------------------
# 8.1 – TOP-5 SUBSTITUTE EXAMPLES
# ------------------------------

import numpy as np
import pandas as pd

# sim_matrix   → similarity_matrix from step 7
# names_small  → list of 5000 drug names (same ordering as similarity_matrix)

def get_top5(drug_index, sim_matrix, names_list):
    row = sim_matrix[drug_index]
    top_idx = np.argsort(-row)[1:6]   # skip itself (index 0)
    return [(names_list[i], row[i]) for i in top_idx]

# Select 3 drugs from the 5000-sample dataset
example_indices = [0, 100, 500]   # you can change these later

results = {}

for idx in example_indices:
    results[idx] = {
        "drug_name": names_small.iloc[idx],
        "top5": get_top5(idx, similarity_matrix, names_small)
    }

# Display results cleanly
for idx, res in results.items():
    print("\n====================================")
    print("Drug:", res["drug_name"])
    print("Top-5 substitutes:")
    for sub_name, score in res["top5"]:
        print(f"  → {sub_name:50s} | similarity: {score:.4f}")
```

```
====================================
Drug: Floxicare OZ 200mg/500mg Tablet
Top-5 substitutes:
  → Locoflam SP Tablet                                | similarity: 0.9357
  → Prerabe 20mg Tablet                               | similarity: 0.9160
  → E Zon 1000mg Injection                            | similarity: 0.8970
  → Minolast-FXA Tablet SR                            | similarity: 0.7354
  → Glimicut MP 2mg/500mg/15mg Tablet SR              | similarity: 0.7199

====================================
Drug: Pseudocef 1gm Injection
Top-5 substitutes:
  → Bencef-SB 1.5gm Injection                         | similarity: 0.9472
```

```
  → Buprigesic 0.3mg Injection                          | similarity: 0.9434
  → Thiother 15mg Injection                             | similarity: 0.9364
  → PT-Rich 4mg/0.5mg Injection                         | similarity: 0.9361
  → Vijocef S 1000mg/500mg Injection                    | similarity: 0.9334

==================================
Drug: Gabamer 100mg Tablet
Top-5 substitutes:
  → Gabapax 100mg Tablet                                | similarity: 0.9910
  → Gabazis M 300mg/500mcg Tablet                       | similarity: 0.9627
  → Gatbro 100mg Tablet                                 | similarity: 0.9511
  → Gabanip M 300mg/500mcg Tablet                       | similarity: 0.9488
  → Jba 5mg/10mg Tablet                                 | similarity: 0.9484
```

In this part, three drugs were selected to show how the model finds possible substitutes.

For Floxicare OZ 200mg/500mg Tablet, the model gave drugs that come from similar antibiotic combinations. The highest scores (0.93–0.91) show that the model can group medicines that often appear in the same treatment area.

For Pacudocef 1gm Injection, the substitutes again had very high similarity. Most of them