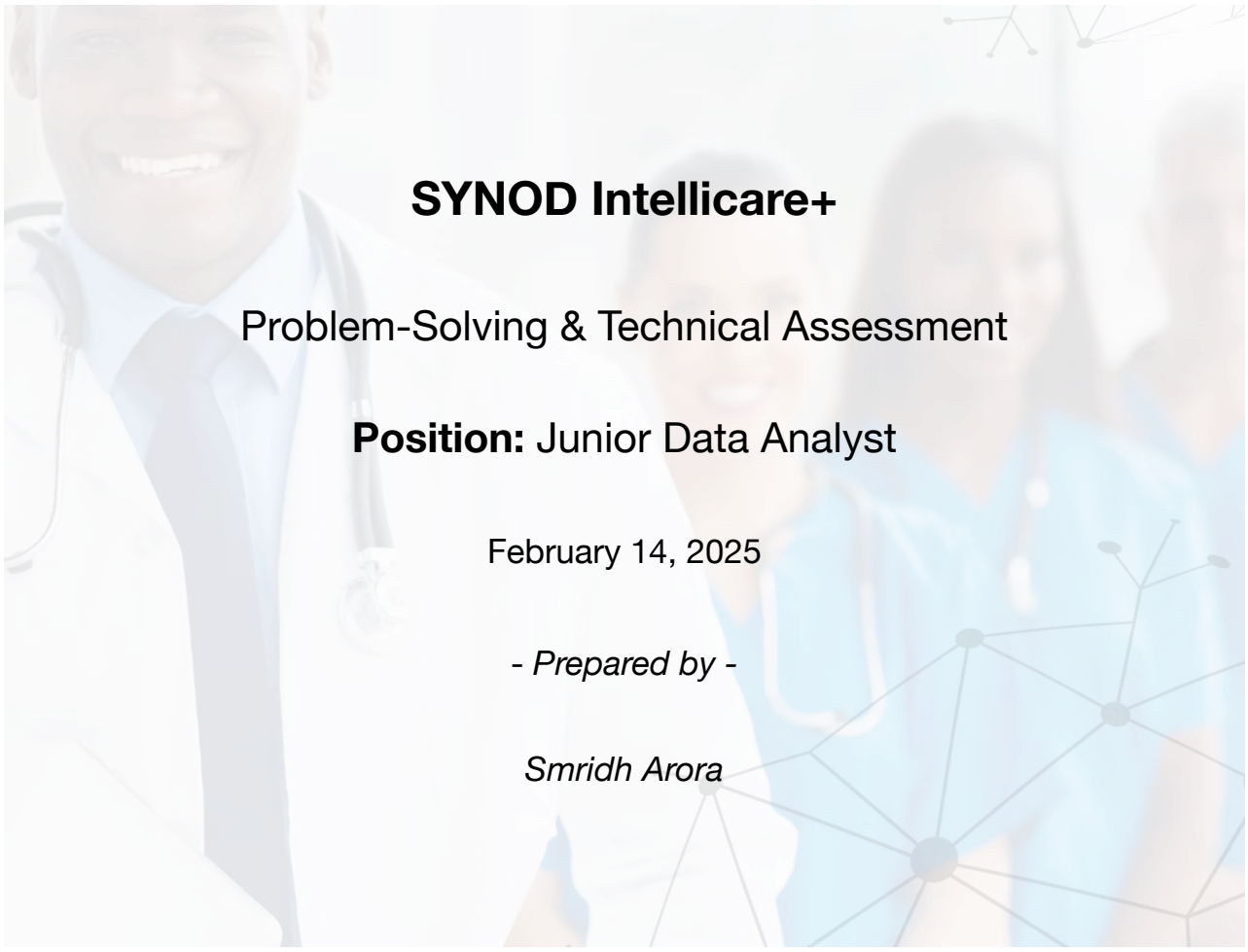

SYNOD Intellicare+

Problem-Solving & Technical Assessment

Position: Junior Data Analyst

February 14, 2025

- Prepared by -

Smridh Arora

General Questions

Question: What is greatest success you have achieved so far?

I believe my greatest success is still ahead of me. I'm constantly learning, improving, and pushing myself, so I see every project as a stepping stone toward something bigger. But if I had to pick one so far, it would be the Vaccine Confidence Surveillance System I built.

Situation:

During the peak of the COVID-19 pandemic, vaccine hesitancy was a huge challenge. There was a surge of misinformation, rumours, and negative sentiment regarding vaccines spreading across different demographics. Governments and healthcare organizations needed real-time insights into public sentiment, but a lot of that information was scattered across social media. I saw potential in using this data to make sense of the noise - help guide public health strategies

Task:

The goal was to build a **prototype vaccine hesitancy surveillance system** that could **track trends and sentiments based on reading tweets from Twitter**. By filtering keywords from the comments, the system aimed to classify user's post (sentiment) as positive or negative. These insights would help public health officials strategically place their vaccination campaigns in regions with higher hesitancy and increase awareness efforts where needed. The ultimate **objective** was to provide **actionable, data-driven intelligence** to **improve public health outreach and vaccine acceptance**.

Action:

To develop this system, I first obtained access to Twitter's API and used Tweepy to collect live tweets containing vaccine-related keywords. I **manually labeled a subset of tweets** to create a reliable **training dataset**, ensuring consistency in sentiment classification. Using this labeled data, I t**rained a machine learning model** with **natural language processing techniques** to categorize tweets into positive, neutral, or negative sentiment categories, refining the model's accuracy with iterative testing and validation.

Result:

I **successfully created both the tweet labeller and the classifier**, ensuring that the system could effectively process and categorize vaccine-related sentiments. The **system had the potential to be connected to data visualization tools** such as Tableau to **generate insights for stakeholders**, allowing them to make informed decisions and **strategize vaccination campaigns** based on public sentiment.

Seeing how data could provide meaningful insights for public health decision-making felt like a success in itself. The experience enhanced my ability to work with real-world data and reinforce my interest in data analytics.

* Further details on implementation (GitHub): *Vaccine Surveillance System*

Question: Where do you see yourself in five year?

Right now, I'm focused on **learning everything I can about AI in healthcare** with the resources available to me. My **goal is to build a strong foundation** - understanding AI models, analyzing trends, and figuring out how data can actually make a difference. There's a lot to take in, but I enjoy the challenge and the process of becoming an expert in this field.

Once I gain hands-on experience and a deeper understanding, I see myself stepping into a **managerial role** where I can **shape decision-making and have a broader impact**. I want to be in a position where I can **contribute fresh ideas** on how AI can **uncover new aspects of health**. With a strong foundation in health sciences, I see **AI driving the transition toward a preventative healthcare model** - going beyond diagnosing diseases to actually **predicting and preventing** them before they happen. For example, AI-powered tools could seamlessly integrate into daily life through wearables and smart health technologies, **empowering people to take control of their health** proactively.

I want to **cultivate a team of like-minded individuals** who see data the way I do - not just as numbers but as a tool to solve real-world problems. I truly believe in the saying, "**What gets measured gets managed**," and I see data as the key to unlocking better population health outcomes.

At its core, my journey is about keeping data at the centre of healthcare transformation. AI is already shaping industries like finance, business, and transportation, and I believe healthcare is next. **I want to be part of that shift, using AI not just to improve patient care but to change how we think about health itself**. Five years from now, I see myself leading, innovating, and playing a role in making healthcare more intelligent, efficient, and proactive for everyone.

Technical Questions

Question: How would you write an SQL query to find the top 5 most common diagnoses in a hospital dataset?

We begin by constructing a sample table called hospital_dataset (for reference)- which includes a list of patients along with their diagnoses.

Each row represents: {**patient id, diagnose, date_of admission**}

To get the top 5 most common diagnoses, we need to:

1) Use the COUNT(*)
-> to count how often each diagnose occur

2) Use GROUP BY
-> to group the same type of diagnoses

3) Use ORDER BY DESC
-> sort the result from most to least common

4) Use LIMIT
-> adjust the output to certain limit i.e. 5

Putting the query together:

```
SELECT diagnose, COUNT(*) AS Frequency
FROM hospital_dataset
GROUP BY diagnose
ORDER BY frequency DESC
LIMIT 5;
```

Hospital_Dataset

patient_id	diagnose	date_of_admission
1	Flu	2024-09-12
2	COVID-19	2025-02-06
3	Pneumonia	2024-03-05
4	Diabetes	2024-11-14
5	Hypertension	2024-10-08
6	Asthma	2024-12-13
7	COVID-19	2024-11-06
8	Flu	2024-03-01
9	Flu	2024-05-20
10	Pneumonia	2024-09-23
11	Diabetes	2025-02-01
12	Flu	2024-12-19
13	Hypertension	2024-07-03
14	Flu	2025-02-10
15	Asthma	2025-02-13
16	COVID-19	2024-06-14
17	Flu	2024-04-22
18	Pneumonia	2024-09-13
19	Diabetes	2024-05-23
20	Flu	2024-08-30

**This dataset could have thousands of rows, but the idea remains the same*

Diagnose	Frequency
Flu	7
COVID-19	3
Pneumonia	3
Diabetes	3
Hypertension	2

Output produced through the SQL query

`SELECT diagnose, COUNT(*) AS Frequency`: Selects the diagnoses column and counts how many times each diagnose appears.

`FROM hospital_dataset`: Specifies the table we are querying.

Question: Write a python function to clean and preprocess the data.

Introduction to dataset & environment

We begin by importing all the necessary libraries in Jupyter Notebook

```
import pandas as pd
import re
from datetime import datetime
import nltk
from nltk.corpus import stopwords
from IPython.display import display
```

I made a custom dataset containing unstructured medical data to showcase inconsistencies - missing values, formatting issues, and personally identifiable information etc. We will refine this raw data using mentioned instructions and make the data usable for driving insights.

```
with open("raw_medical_notes.txt" , 'r') as file:
    raw_medical_notes = json.load(file)

df = pd.DataFrame(raw_medical_notes)

display(df)
```

patient_id	name	dob	age	gender	symptoms	diagnosis	medications	timestamp	doctor	email	phone	
001	Alice Johnson	1992-03-14	Thirty-Two	FEMALE	[headache, Nausea, None, Blurred Vision]	Migraine	[sumatriptan, ibuprofen , None, Naproxen]	2024/02/09 15:45	Dr. Smith	alicej@example.com	+1 (555) 987-6543	
002	Bob Williams	1987-07-29	37	male	fever, cough, body ache	None	[Tylenol, amoxicillin , ibuprofen]	09-02-2024 9:30 AM	Dr. Adams	NaN	NaN	f
003	Charlie O'Connor	05-15-1995	29	M	[fatigue, joint pain, low appetite]	Rheumatoid arthritis	[methotrexate, prednisone, None]	February 9, 2024, 17:20	Dr. Miller	charlieoc@example.com	555-777-1234	
004	None	Unknown	None	Female	[sore throat, Cough, sore throat]	Strep throat	[penicillin, , azithromycin]	02/09/2024 12:00 PM	Dr. Brown	NaN	None	e
005	David Patel	1991-06-21	Thirty-three	Male	shortness of breath, dizziness, fatigue	Anemia	[iron supplements, None, folic acid]	2024-09-02T20:10:00	Dr. White	davidp@hospital.com	NaN	

The screenshot shows first 5 entries from our data frame containing a total of 20 entries.

Code Breakdown ->

Instead of writing a single long function that does everything, we break it down into smaller, well-defined helper functions. This approach makes the code easier to debug, test, and understand.

1) **load_clean_medical_data** -> helper function ensures sensitive information (PII) is removed at the beginning

```python
# Step 1: Remove Personally Identifiable Information (PII)


def load_clean_medical_data(file_path):
    """
    Loads a medical dataset from a JSON file and removes Personally Identifiable Information (PII).

    Steps:
    - Reads the file and converts it into a DataFrame.
    - Removes columns containing PII (e.g., 'name', 'email', 'phone', 'address').
    - Ensures the original dataset remains unchanged.

    Parameters:
    file_path (str): Path to the JSON file.

    Returns:
    pd.DataFrame: A cleaned DataFrame with PII removed.
    """

    # Open and Load the File:
    with open(file_path, 'r') as file:
        raw_medical_notes = json.load(file)

    # Convert JSON data into a Pandas DataFrame
    df = pd.DataFrame(raw_medical_notes)

    # Remove PII columns
    df_cleaned = df.drop(columns=["name", "email", "phone", "address"], errors="ignore")

    return df_cleaned
```

2) **standardize_timestamp** -> helper function ensures all timestamps are in the same format (YYYY-MM-DD HH:MM:SS), avoiding inconsistencies in time-based data:

```python
# Step 2: convert all timestamps to a single standardized format: YYYY-MM-DD HH:MM:SS

def standardize_timestamp(timestamp):
    """
    Converts various timestamp formats into the standard format: YYYY-MM-DD HH:MM:SS.
    Ensures the original DataFrame remains unchanged.

    Parameters:
    timestamp (str): The input timestamp as a string.

    Returns:
    str or None: Standardized timestamp in 'YYYY-MM-DD HH:MM:SS' format or None if invalid.
    """

    if not isinstance(timestamp, str) or timestamp.lower() == "unknown":
        return None  # Handle non-string values or "unknown" timestamps

    # List of possible timestamp formats to check
    possible_formats = [
        "%Y/%m/%d %H:%M",       # Example: 2024/02/14 14:30
        "%m-%d-%Y %I:%M %p",    # Example: 02-14-2024 02:30 PM
        "%B %d, %Y, %H:%M",     # Example: February 14, 2024, 14:30
        "%Y-%m-%dT%H:%M:%S"     # Example: 2024-02-14T14:30:00 (ISO format)
    ]
    for fmt in possible_formats:
        try:
            parsed_time = datetime.strptime(timestamp, fmt)  # Try to parse the timestamp
            return parsed_time.strftime("%Y-%m-%d %H:%M:%S")  # Convert to standard format
        except ValueError:
            continue # Move to the next format if fails

    return None  # If all formats fail, return None
```

3) **convert_text_to_lowercase** -> helper function converts all the text to lowercase for consistency, which can be later used to remove stop words as well

```python
#Step 3: Convert all text into lowercase for consistency


def convert_text_to_lowercase(df):
    """
    Converts all text (string) columns in a DataFrame to lowercase, including nested lists.
    Ensures the original DataFrame remains unchanged.

    Parameters:
    df (pd.DataFrame): The input DataFrame.

    Returns:
    pd.DataFrame: DataFrame with all text converted to lowercase.
    """
    df_copy = df.copy()  # Create a copy to avoid modifying the original DataFrame

    def to_lower(value): #helper function
        """
        Helper function to convert text to lowercase.
        Handles strings directly and applies conversion to lists of strings recursively.
        """
        if isinstance(value, str):
            return value.strip().lower()  # Trim spaces & convert to lowercase
        elif isinstance(value, list):
            return [to_lower(item) for item in value if isinstance(item, str)]  # Process lists of strings
        return value  # Return unchanged if not a string or list

    # Apply lowercase conversion to all columns values
    df_copy = df_copy.apply(lambda col: col.map(to_lower) if col.dtype == "object" else col)

    return df_copy
```

4) **remove_stopwords** -> helper function makes use of NLTK to remove stop words from dataframe

```python
# Step 4 remove stopwords to improve text processing

stop_words = set(stopwords.words('english'))


def remove_stopwords(df):
    """
    Removes stopwords from all string columns in a DataFrame, including lists of strings.
    Ensures the original DataFrame remains unchanged.

    Parameters:
    df (pd.DataFrame): The input DataFrame.

    Returns:
    pd.DataFrame: A new DataFrame with stopwords removed.
    """

    df_copy = df.copy() # Create a copy to avoid modifying the original DataFrame

    def clean_text(value): #helper function
        """Removes stopwords from a string or a list of strings."""
        if isinstance(value, str):
            # Split the string into words and filter out stopwords
            return ' '.join([word for word in value.split() if word.lower() not in stop_words])
        elif isinstance(value, list):
            # If the value is a list, clean each string item recursively
            return [clean_text(item) for item in value if isinstance(item, str)]
        return value  # # Return unchanged if it's not a string or list

    # Apply stopword removal only to text columns
    df_copy = df_copy.apply(lambda col: col.apply(clean_text) if col.dtype == "object" else col)

    return df_copy
```

5) **clean_dataframe** -> helper function fills missing gender values with 'No Gender', replaces empty text fields with 'Unknown', and fills missing numeric values with the median of each column - inspiration from GeeksforGeeks

```python
# Step 5 Identify and Handle missing or duplicated entries

def clean_dataframe(df):
    """
    Cleans a DataFrame by:
    - Handling missing values properly
    - Filling empty 'gender' values with 'No Gender'
    - Replacing empty strings with NaN and handling them appropriately
    - Removing duplicate rows

    Parameters:
    df (pd.DataFrame): The input DataFrame.

    Returns:
    pd.DataFrame: A cleaned version of the DataFrame.
    """

    df_cleaned = df.copy() # Create a copy to avoid modifying the original DataFrame

    # Convert lists -> tuples (Fix error when checking duplicates)
    for col in df_cleaned.columns:
        df_cleaned[col] = df_cleaned[col].apply(lambda x: tuple(x) if isinstance(x, list) else x)

    # Handle missing values in the 'gender' column
    if 'gender' in df_cleaned.columns:
        df_cleaned['gender'] = df_cleaned['gender'].replace("", pd.NA)  # Treat empty strings as missing
        df_cleaned['gender'] = df_cleaned['gender'].fillna('No Gender')  # Replace missing gender values with 'No Gender'

    # Replace empty strings in all object (text) columns with NaN
    text_columns = df_cleaned.select_dtypes(include='object').columns
    df_cleaned[text_columns] = df_cleaned[text_columns].replace("", pd.NA)

    # Fill missing values in text columns with "Unknown" for better clarity
    df_cleaned[text_columns] = df_cleaned[text_columns].fillna("Unknown")

    # Fill missing values in numeric columns with the median of the column
    numeric_columns = df_cleaned.select_dtypes(include='number').columns
    for col in numeric_columns:
        df_cleaned[col] = df_cleaned[col].fillna(df_cleaned[col].median())

    # Remove duplicate rows
    df_cleaned = df_cleaned.drop_duplicates()

    return df_cleaned
```

Now, instead of having a single massive function, we have successfully defined our small, reusable helper functions.

Our **main function: process_medical_notes(file_path)** calls them in the right order, keeping the logic clean and modular.
- loads a dataset containing medical notes from a given file
- cleans the data by removing Personally Identifiable Information (PII)
- standardizing timestamps
- converting text to lowercase
- removing stopwords
- and handling missing/duplicate entries.

The cleaned data is saved to a CSV file (cleaned_medical_notes.txt) and returned as a DataFrame.

```python
# Main Function: Calls All Helper Functions

def process_medical_notes(file_path):
    """
    Loads and cleans medical notes by:
    - Removing PII
    - Standardizing timestamps
    - Converting text to lowercase
    - Removing stopwords
    - Handling missing & duplicate entries

    Returns:
    Cleaned DataFrame
    saves the cleaned data in csv file called cleaned_medical_notes.txt
    """

    # Load dataset and remove PII
    df = load_clean_medical_data(file_path)
    if df is None:
        return None  # Return early if file is missing

    # Standardize timestamp if column exists
    if 'timestamp' in df.columns:
        df['timestamp'] = df['timestamp'].apply(standardize_timestamp)

    # Apply text preprocessing functions
    df = convert_text_to_lowercase(df)
    df = remove_stopwords(df)

    # Handle missing values and duplicates
    df_cleaned = clean_dataframe(df)
    df_cleaned.to_csv('cleaned_medical_notes.txt')

    return df_cleaned
```

Output: Clean data frame

	patient_id	dob	age	gender	symptoms	diagnosis	medications	timestamp	doctor	notes
0	001	1992-03-14	thirty-two	female	(headache, nausea, blurred vision)	migraine	(sumatriptan, ibuprofen, naproxen)	2024-02-09 15:45:00	dr. smith	patient reported severe pain head sensitivity ...
1	002	1987-07-29	37	male	fever, cough, body ache	Unknown	(tylenol, amoxicillin, ibuprofen)	2024-09-02 09:30:00	dr. adams	patient persistent fever past 3 days, advised ...
2	003	05-15-1995	29	No Gender	(fatigue, joint pain, low appetite)	rheumatoid arthritis	(methotrexate, prednisone)	2024-02-09 17:20:00	dr. miller	complained stiffness joints lack energy
3	004	unknown	Unknown	female	(sore throat, cough, sore throat)	strep throat	(penicillin, , azithromycin)	Unknown	dr. brown	patient experiencing throat pain several days.
4	005	1991-06-21	thirty-three	male	shortness breath, dizziness, fatigue	anemia	(iron supplements, folic acid)	2024-09-02 20:10:00	dr. white	reported frequent dizziness, low energy, recom...

[** For full code and testing, check out the .ipynb file in the GitHub **]

Approach to Data Cleaning & Preparation

Discovering Issues in Data

The first step is crucial — **getting familiar with the dataset** before making any changes. I always like to start by loading the dataset into a viewable format, like a Pandas DataFrame, so I can easily explore it. Simple but effective steps—like displaying the first few rows, checking column data types, and generating summary statistics—help uncover potential issues early.

For example, I'd **scan for empty cells in key fields** like gender, diagnosis, or medications, since missing values in these areas can impact analysis. Another important check is ensuring that timestamps follow a consistent format, which is essential for tracking trends over time. Additionally, I'd check for d**uplicate entries, as redundant records can skew results** and misrepresent the data. By identifying these issues upfront, I can create a structured plan for cleaning the dataset while ensuring that no valuable information is lost.

Ensuring Data Integrity while cleaning

Cleaning data should enhance its quality without distorting its meaning. To ensure accuracy, I followed a structured approach—removing PII, standardizing timestamps, and formatting text consistently — as outlined in the 1st part of exercise. These steps help create a dataset that is both clean and reliable without losing important details.

When **handling missing values**, I used a **context-aware approach**, meaning I filled gaps in a way that makes sense for the dataset rather than applying a one-size-fits-all method. For example, in our case, n**umerical values were replaced with the median**, ensuring the data distribution remained intact, while **missing text fields were labeled as "Unknown"** to maintain transparency rather than making assumptions.

Another important consideration is that text cleaning can sometimes alter meaning. While l**owercasing text and removing stop words** help in most cases, they can also strip away crucial medical terms or **change how information is interpreted**. That's why it's always essential to evaluate whether text preprocessing aligns with the dataset's purpose before applying it universally.

Lastly, as seen in our process, I **made copies of the DataFrame at every stage**. Since **data cleaning is an irreversible process**, **maintaining snapshots** of the raw, intermediate, and final versions of the dataset is a **good practice**. This way, if an error is introduced or an important data point is lost, we can always trace back and recover the original information. This ensures that the final dataset is not only clean but also auditable, transparent, and trustworthy.

Annotating Important Data Points for better Insights

Data is only valuable if we can extract meaningful insights from it. In our hospital dataset, one approach to enriching the data would be to use Natural Language Processing (NLP) technique to automatically **identify medical conditions based on prescribed medications** within patient records.

Another useful annotation would be **tagging timestamps related to hospital visits and treatments.** This would allow us to track disease progression, medication effectiveness, and the persistence of symptoms over time. For example, if a patient has recurring symptoms despite medication, this could signal the need for further evaluation.

Additionally, we could **prioritize records based on urgency**. Certain symptoms, like "chest pain" or "shortness of breath", could be **flagged as high priority, ensuring that they are reviewed first**. This type of categorization could help doctors and medical professionals focus on the most critical cases faster, leading to more efficient patient care and timely interventions.

Explaining Cleaned Data to Non-Technical Manager

Explaining the cleaned data to a non-technical manager is all about **focusing on value rather than the technical process.** Instead of talking about "NaN handling" or "data type conversions," I'd frame it like this:

"We've made sure every patient entry is complete, consistent, and easy to analyze. Missing values are handled so that reports won't have gaps, timestamps are standardized so trends over time are clear, and unnecessary duplicates are removed to avoid skewed statistics. Now, the dataset is reliable, structured, and ready for meaningful analysis—whether it's tracking medication effectiveness or identifying common diagnoses."

I would also include a visual comparison to highlight the transformation, showcasing how the cleaned data is more structured and organized compared to the raw, unstructured dataset.