MCO1 PRESENTATION

Presentation by Group 2

Alvarez, Jumilla, (Orendain)*, Park

STADVDB S14

*WITHDRAWL

INTRODUCTION

The Appointment dataset, comprising records from the SeriousMD startup company, serves as a source of information containing appointments, doctors, clinics, and patient data. With the intention of constructing a comprehensive data warehouse, this project focuses on designing a dimensional model in a star schema format.

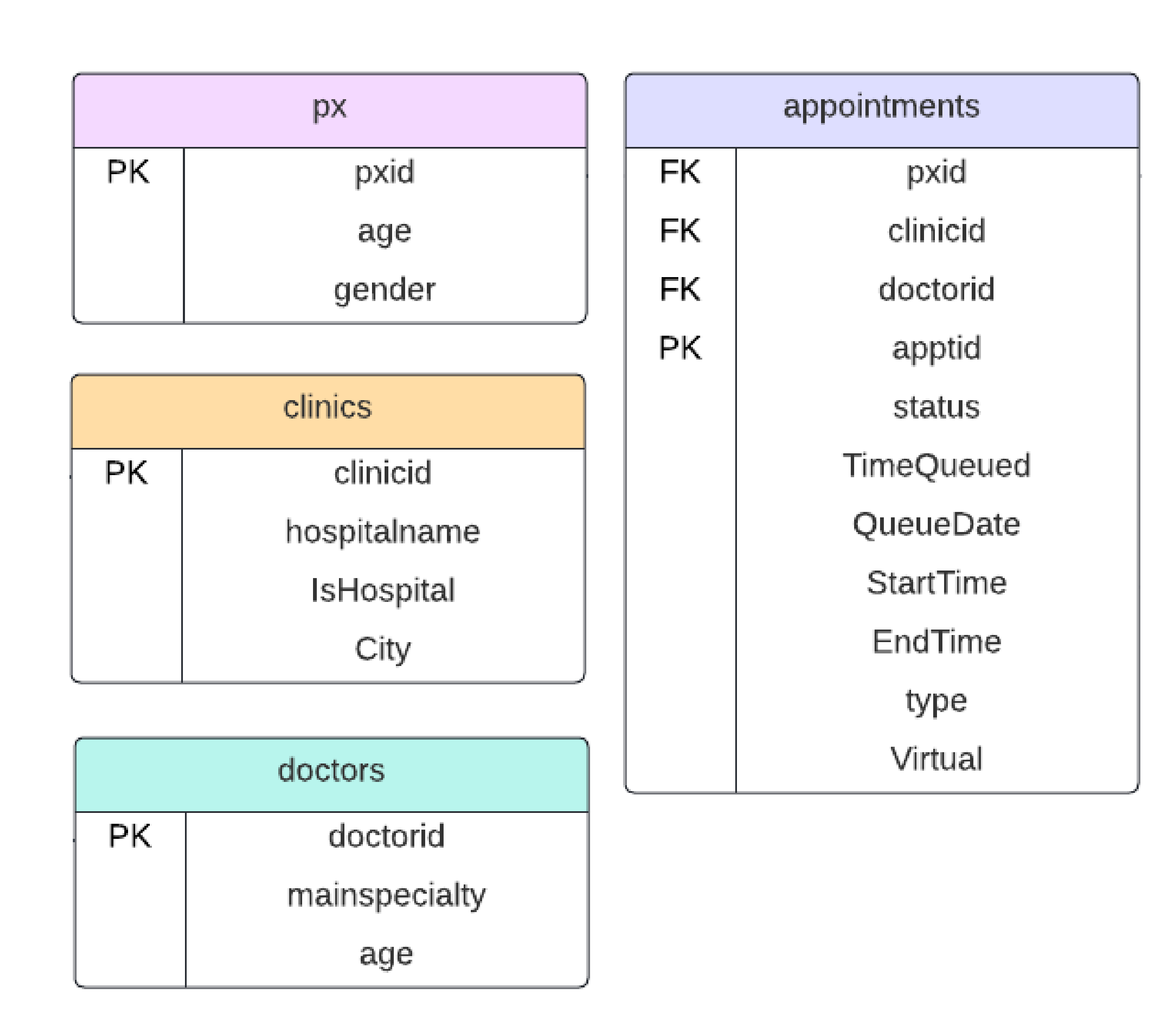
DATA WAREHOUSE

DESIGNING THE SCHEMA

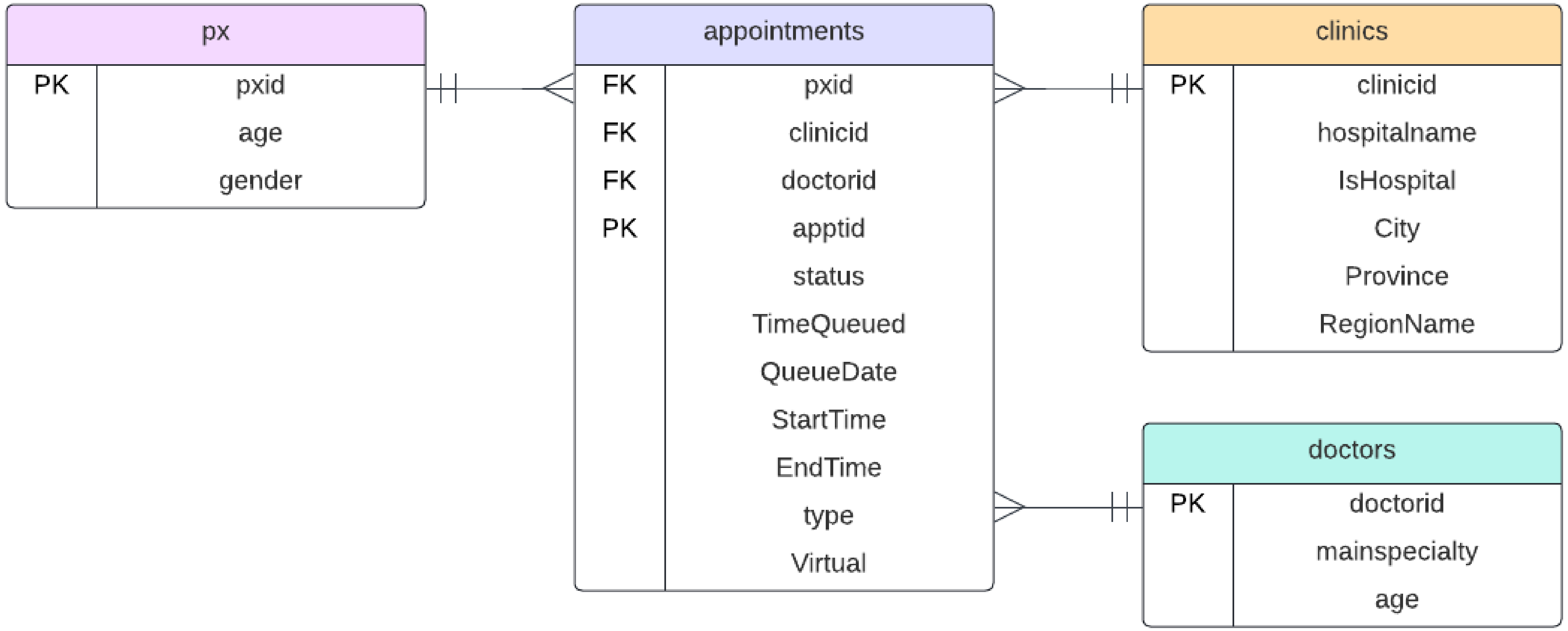
Data sets includes for different csv files, namely appointments, doctors, clinics, and px

Each data contains a unique id to be used as a primary id, and appointments data includes the id from different data to be used as a foreign key.

The dimensional model the group has designed to follow a star schema format



STAR SCHEMA DESIGN



WHY STAR SCHEMA DESIGN?

The design of the star schema modeling includes a fact table in the middle connected towards a different dimension table surrounding it. The data sets were formatted to fit the star schema the best

The large file size of each of the data sets may cause a delay in data loading time when designed incorrectly. Compared to snowflake schema, star schema has relatively faster data loading, and is best suited for OLAP operations such as slice and dice

DATA CLEANING

To clean the, data. The group decided to use Jupyter Notebook.

DATA CLEANING

```python
import pandas as pd

df = pd.read_csv("/content/px.csv", engine='python', encoding='utf-8', error_bad_lines=False)
df = df.dropna(subset=['age','pxid','gender']) # Removes NaN
df = df.drop_duplicates(subset=['pxid']) # Removes duplicate pxid
df = df[df['gender'].isin(['MALE', 'FEMALE'])] # Removes any value not male or female
df['age'] = pd.to_numeric(df['age'], errors='coerce') # Changes dtype of age to float64
df = df[(df['age'] >= 0) & (df['age'] <= 100)] # Removes if age is negative or greater than 100
df.to_csv('cleaned_px.csv', index=False) # Save changes
```

In cleaning px.csv;

- We first removed rows with null values.
- Then rows that have the same pxid (duplicates) as previous rows were removed.
- In the column "gender", if it is not MALE or FEMALE, the row will be removed.
- The data type of age was changed from int to float since it was easier to use.
- Then if the age is a negative value or is greater than 100, the row was also removed.

DATA CLEANING

```python
import pandas as pd
import re
!pip install fuzzywuzzy
from fuzzywuzzy import process

# Read the CSV file into a DataFrame
df_doctors = pd.read_csv('/content/doctors.csv' , engine='python', encoding = "ISO-8859-1", error_bad_lines=False)

# Remove rows with no age and mainspecialty
df_doctors = df_doctors.dropna(subset=['age', 'mainspecialty'])

# Remove newline characters from 'mainspecialty' column
df_doctors['mainspecialty'] = df_doctors['mainspecialty'].replace('\n', '', regex=True)

# Only load data if doctorid's string length is > 25
df_doctors = df_doctors[df_doctors['doctorid'].str.len() > 25]

# Only load data if mainspecialty's string length is >= 2 or null
df_doctors = df_doctors[(df_doctors['mainspecialty'].str.len() >= 2) | df_doctors['mainspecialty'].isnull()]

# Only load data if age is between 18 - 100
df_doctors = df_doctors[(df_doctors['age'] >= 18) & (df_doctors['age'] <= 100)]

# Remove rows with both null or invalid char mainspecialty and age
df_doctors = df_doctors.dropna(subset=['mainspecialty'], how='all')
```

In cleaning doctors.csv;

- Removed rows with no values in the columns ‘age’ and ‘mainspecialty’.
- Newline characters from ‘mainspecialty’ were also removed to be uniform with others.
- If the string length for the doctorid is less than or equal to 25, it is removed.
- If the mainspecialty’s string length is greater than or equal to 2 or null, it is loaded into the dataset.
- If the age is between 18-100, the data will be loaded because doctors can only be 18 years old or above.
- Rows with both null or invalid characters in mainspecialty and age are removed.

DATA CLEANING

```
df_doctors = df_doctors.dropna(subset=['age'], how='all')

# Drop duplicate data
df_doctors = df_doctors.drop_duplicates()

# Replace commas with "&" and convert to all caps
df_doctors['mainspecialty'] = df_doctors['mainspecialty'].str.replace(',', '&')

# All Caps for Consistency in mainspecialty
df_doctors['mainspecialty'] = df_doctors['mainspecialty'].str.upper()

# Remove numeric characters from mainspecialty
df_doctors['mainspecialty'] = df_doctors['mainspecialty'].apply(lambda x: re.sub(r'\d', '', str(x)))

# Replace special signs with "AND"
df_doctors['mainspecialty'] = df_doctors['mainspecialty'].replace({'&': 'AND', '/': 'AND', '+': 'AND', '-': ' ', ':': ' ', ')': ' '})

# Replace "&" that is attached to a character and convert to all caps
df_doctors['mainspecialty'] = df_doctors['mainspecialty'].str.replace(r'&([A-Za-z])', r'\1 AND ').str.upper()

# Replace values with 2 or fewer characters to "unspecified"
df_doctors.loc[df_doctors['mainspecialty'].str.len() <= 2, 'mainspecialty'] = 'UNSPECIFIED'

# Remove quotation marks
df_doctors['mainspecialty'] = df_doctors['mainspecialty'].str.replace('"', '')

# Remove rows with emails in the 'mainspecialty' column
```

- Duplicate rows were removed.
- Commas were replaced with “&” and are converted to uppercase letters.
- Numeric characters from mainspecialty were removed. Special signs were replaced with “AND”.
- Strings with “&” attached to a string were replaced with “ AND “ with space to detach it from the string.
- Values with 2 or fewer characters were renamed to “UNSPECIFIED”.
- Quotation marks were removed so that the data will be uniform.

DATA CLEANING

- Rows with emails in their mainspecialty columns were removed.
- Rows with values ``N/A, na, NA, non, none, NOT" in their mainspecialty are removed if there is no age.
- Rows with repeated characters in their mainspecialty were also removed.
- The use of fuzzy wuzzy was used in def correct_spelling(value, choices). This function was used to alter similar or misspelled words into one for consistency. If the words matched 90% of the choices, it will be replaced.

DATA CLEANING

Then the group created a specific list to remove rows where mainspecialty is not a specialty for doctors:

- SOFTWARE DEVELOPER
- SOLSE
- ADASD
- SDAF
- SUPER SAIYAN
- FPOGSF AND PSMFMF AND PSUOG
- KDSJFLKFDSJG
- SDFSDA
- CFVHGBJNK
- DOCTOR PAYTON
- UNSPECIFIED
- ASDASD
- ADASD
- SASA
- WQEQWE

DATA CLEANING

The group also created a list for specific spelling instructions in case the fuzzy wuzzy function was not applied to these:

- 'SURGUY', 'SURGERY
- 'SIRGERY', 'SURGERY'
- 'SUPORT', 'SUPPORT'
- 'ASESMENT', 'ASSESSMENT'
- 'GALBLADER', 'GALLBLADDER'
- 'WELNES', 'WELLNESS'
- 'SESIONS', 'SESSIONS'
- 'STRES', 'STRESS'
- 'SICKNES', 'SICKNESS'
- 'KNE', 'KNEE'
- 'GENRAL', 'GENERAL'
- 'INTENAL', 'INTERNAL'

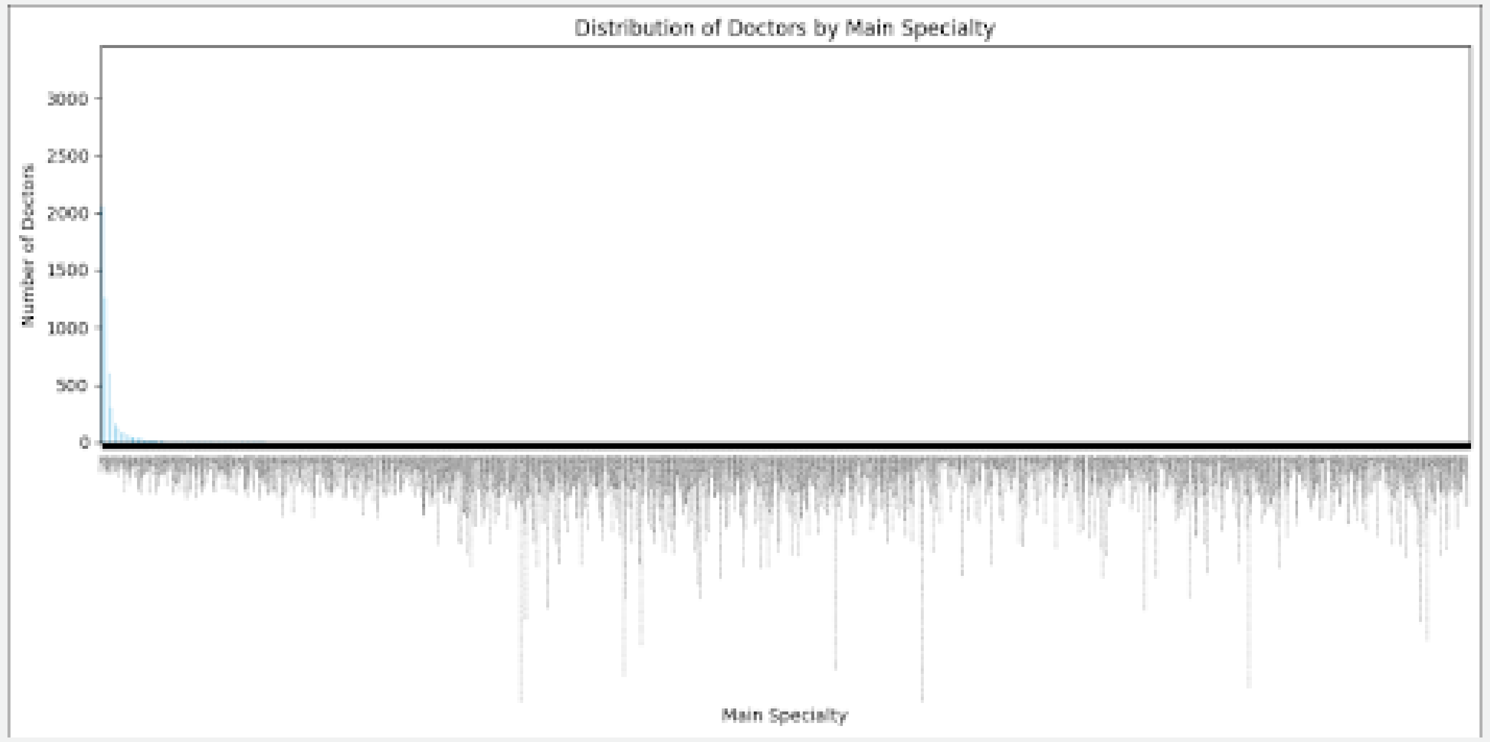
DATA CLEANING

```
df_doctors['mainspecialty'] = df_doctors['mainspecialty'].str.replace('SESIONS', 'SESSIONS', case=False)
df_doctors['mainspecialty'] = df_doctors['mainspecialty'].str.replace('STRES', 'STRESS', case=False)
df_doctors['mainspecialty'] = df_doctors['mainspecialty'].str.replace('SICKNES', 'SICKNESS', case=False)
df_doctors['mainspecialty'] = df_doctors['mainspecialty'].str.replace('KNE', 'KNEE', case=False)
df_doctors['mainspecialty'] = df_doctors['mainspecialty'].str.replace('GENRAL', 'GENERAL', case=False)
df_doctors['mainspecialty'] = df_doctors['mainspecialty'].str.replace('INTRNAL', 'INTERNAL', case=False)

df_doctors['mainspecialty'] = df_doctors['mainspecialty'].apply(correct_spelling, choices=['ADDICTION', 'AESTHETIC', 'ANESTHESIOLOGY','APPLIED', 'ASSESSMENT', 'CARDIOLOGY',
                                                                 'CHIROPRACTOR', 'DERMETOLOGY','ENDOCRINOLOGY', 'GYNE','PRACTITIONER', 'SURGERY', 'SUPPORT', 'UROLOGY'])
# Save the cleaned DataFrame to a new CSV file
df_doctors.to_csv('/content/cleaned_doctors.csv', index=False)
```

- The group also created a correct_spelling function to correct some other misspelled words in the mainspecialty.

DATA CLEANING



Overall, doctors.csv is not yet clean even with all the preprocessing and cleaning the group did on the dataset. The dataset is full of outliers and the only viable way of cleaning it is to manually clean it.

DATA CLEANING

```python
import pandas as pd
# Link your gdrive and upload the appointments.csv there
# Change the directory according to your google drive
df = pd.read_csv("/content/drive/MyDrive/STADVDB/appointments.csv", engine='python', encoding = "ISO-8859-1", error_bad_lines=False)
df_doctors = pd.read_csv("/content/cleaned_doctors.csv", engine='python', encoding = "ISO-8859-1", error_bad_lines=False)
# Only get the rows where doctorid exists in cleaned_doctors csv
df = df[df['doctorid'].isin(df_doctors['doctorid'])]

# Automatically converts the field into datetime format
df['QueueDate'] = pd.to_datetime(df['QueueDate'], errors='coerce', infer_datetime_format=True)
df['TimeQueued'] = pd.to_datetime(df['TimeQueued'], errors='coerce', infer_datetime_format=True)
df['StartTime'] = pd.to_datetime(df['StartTime'], errors='coerce', infer_datetime_format=True)
df['EndTime'] = pd.to_datetime(df['EndTime'], errors='coerce', infer_datetime_format=True)

# Changes the formatting of the datetime
df['QueueDate'] = df['QueueDate'].dt.strftime('%m/%d/%Y') # QueueDate doesn't need time, so remove time
df['TimeQueued'] = df['TimeQueued'].dt.strftime('%m/%d/%Y %I:%M:%S %p') # Others, keep the time
df['StartTime'] = df['StartTime'].dt.strftime('%m/%d/%Y %I:%M:%S %p')
df['EndTime'] = df['EndTime'].dt.strftime('%m/%d/%Y %I:%M:%S %p')

df.to_csv('cleaned_appointments.csv', index=False) # Save changes
```

In cleaning appointments.csv;

- The group had to upload the csv file into google drive in order to upload it into google colab since the raw uncleaned file size was about 2GB.
- The cleaned_doctors.csv file was also read in order to just get the rows in appointments.csv wherein the doctorid exists in cleaned_doctors.csv.
- Then ‘QueueData’, ‘TimeQueued’, ‘StartTime’, and ‘EndTime’ were converted into datetime format into month, date, year, time. QueueDate does not need time so time was removed in the conversion.

DATA CLEANING

```python
import pandas as pd

# Assuming the file path for clinic.csv
clinics_file_path = '/content/clinics.csv'

# Read the CSV file into a DataFrame with an alternative encoding
df_clinics = pd.read_csv(clinics_file_path, encoding='latin-1')

# Drop rows with missing values in important columns
df_clinics = df_clinics.dropna(subset=['clinicid', 'IsHospital', 'City', 'Province', 'RegionName'])

# Drop duplicate rows based on the 'clinicid' column
df_clinics = df_clinics.drop_duplicates(subset=['clinicid'])

# Save the cleaned DataFrame to a new CSV file
df_clinics.to_csv('/content/cleaned_clinics.csv', index=False)
```

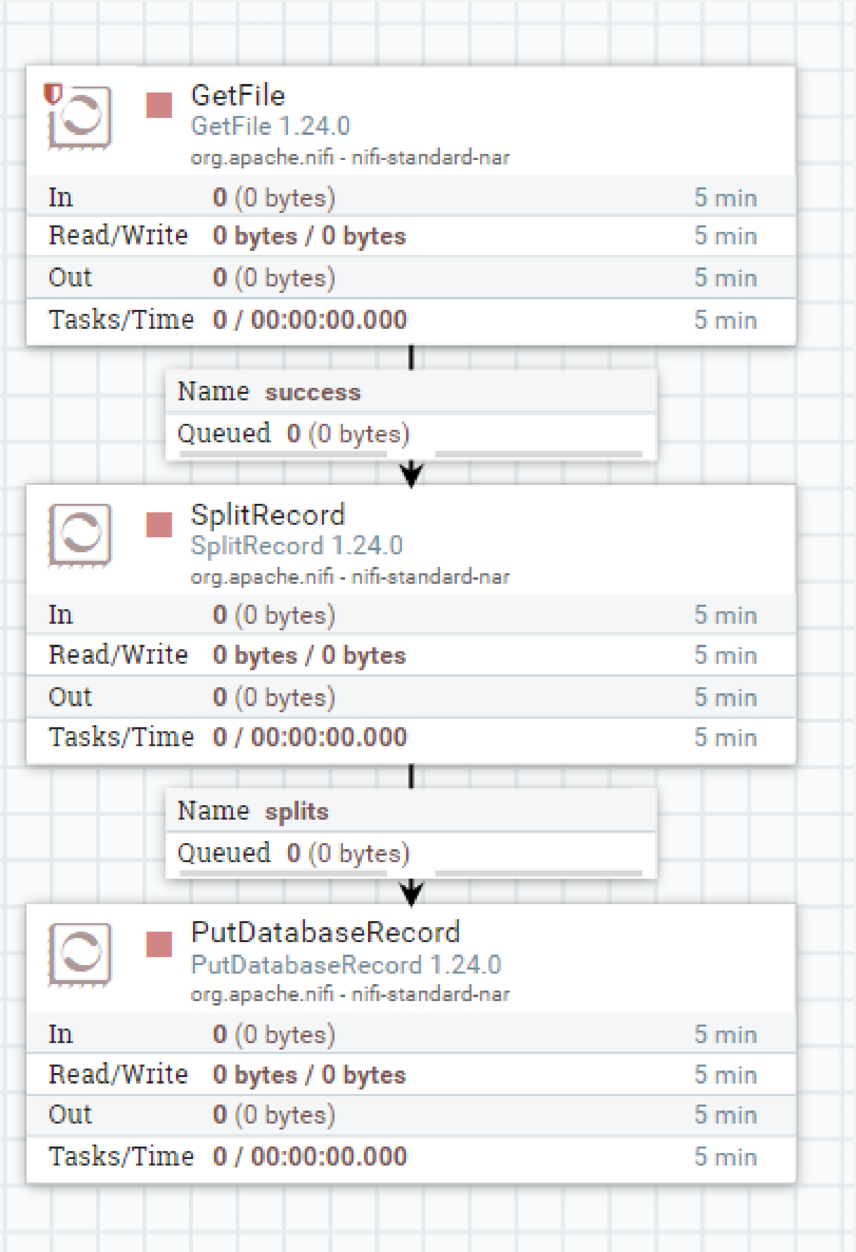
In cleaning clinics.csv;

- Rows with missing values in important columns: 'clinicid', 'IsHospital', 'City', 'Province', and 'RegionName' were dropped.
- Duplicate rows based on the ‘clinicid’ column were removed.

ETL PIPELINE WITH APACHE NIFI

Automating the Transfer of Data to the Data Warehouse

INITIAL PIPELINE DESIGN



GETFILE

Retrieve the files within the path specified by the user and allow the downstream processor to retrieve them as a flowfile

SPLITRECORD

Convert the flowfile file type from csv to JSON using the default controllers, and partitions the data into smaller files

PUTDATABASERECORD

Reads the JSON flowfile with the Nifi-provided JsonTreeReader controller and perform INSERT keyword into the data warehouse table users have specified in the controller found at Database Connection Pooling Service

PROBLEM: 1

Apache Nifi has started sending out an error statement: *Unable to write flowfile content to content repository container default due to archive file size constraints.*

This was caused by having a low number of Records Per Split in the SplitRecord processor, This would cause Apache Nifi to create a large number of small flowfiles, causing increased process overhead which ultimately results in memory restriction.

Simplest solution to address this problem was to increase the number of Records Per Split to reduce the number of flowfiles

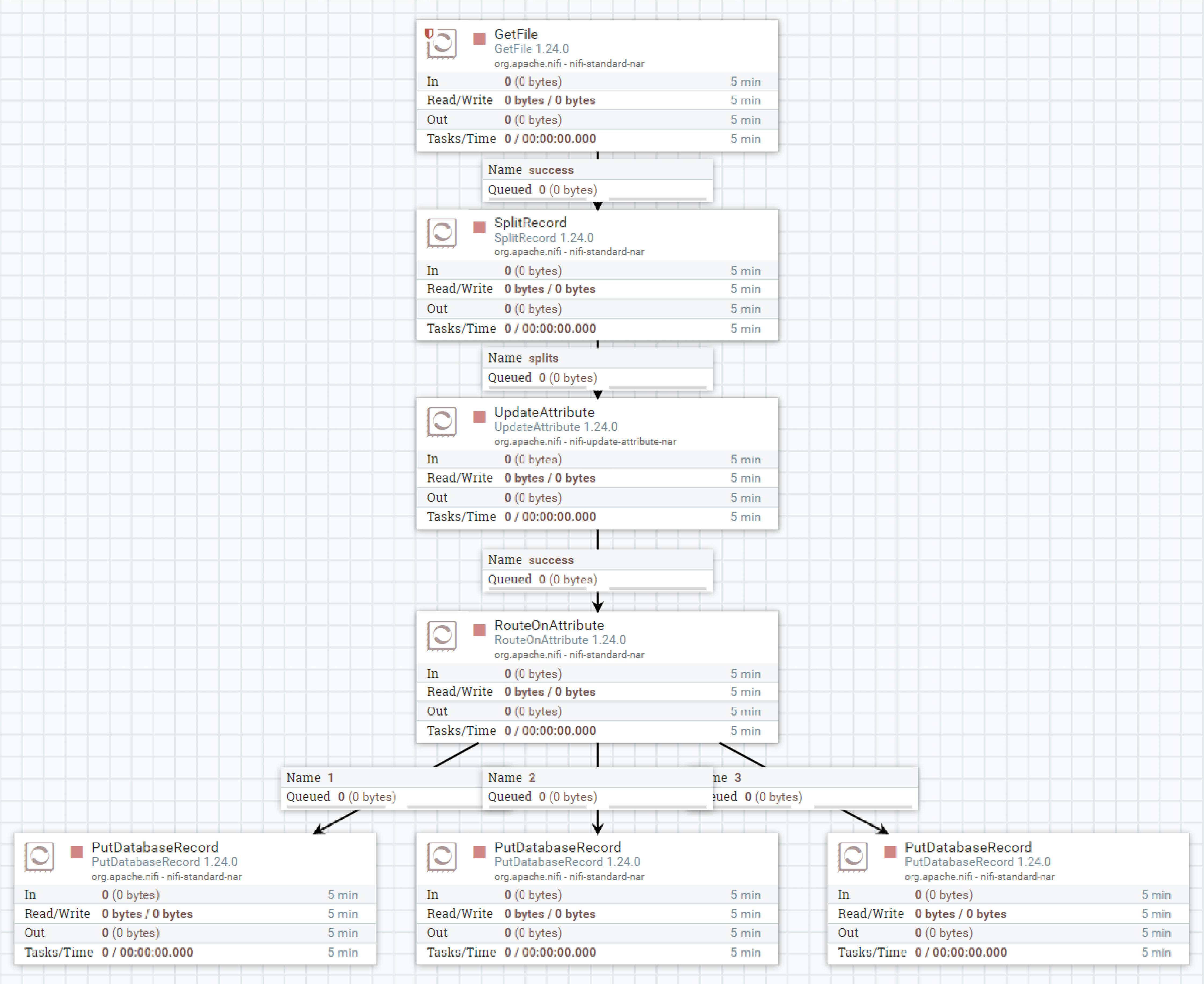
PROBLEM: 2

The second problem our team has encountered was the speed of the ETL pipeline transactions.

There was a bottleneck in the flowfile coming from SplitRecord to PutDatabaseRecord, as there was only a single route to transact a large amount of data.

With this, a new ETL pipeline had to be designed to address the extreme bottleneck found at the end of the pipeline.

REVISED DESIGN



UPDATE ATTRIBUTE

UpdateAttribute processor can be utilized to assign a specific property to each flowfiles

UpdateAttribute process has a configuration to store state locally and we can utilize the local variable inside the processor to assign an unique id to the flowfiles

Required field

Property	Value
Delete Attributes Expression	No value set
Store State	**Store state locally**
Stateful Variables Initial Value	0
Cache Value Lookup Cache Size	**100**
seq	${getStateValue("seq"):plus(1)}

ROUTE ON ATTRIBUTE

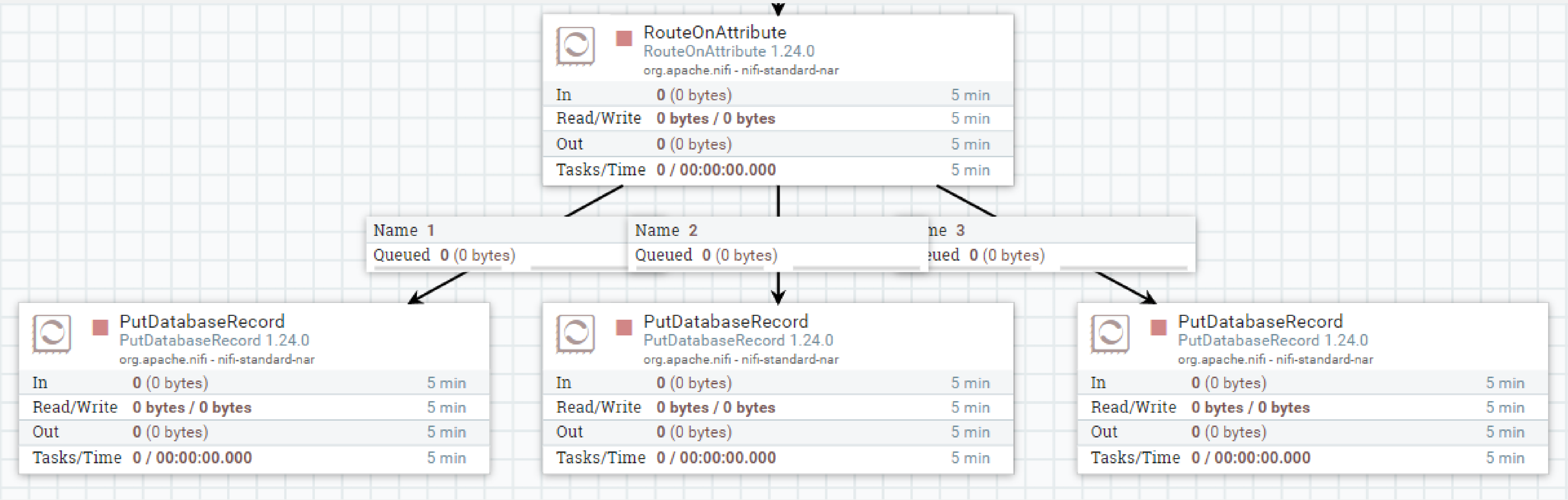
This processor allows the user to create a routing condition for the upcoming flowfile and redirect them accordingly

The group has configured the processor to have three different routes, and utilized the following query to segregate them.

${seq:mod(3):equals(x)}

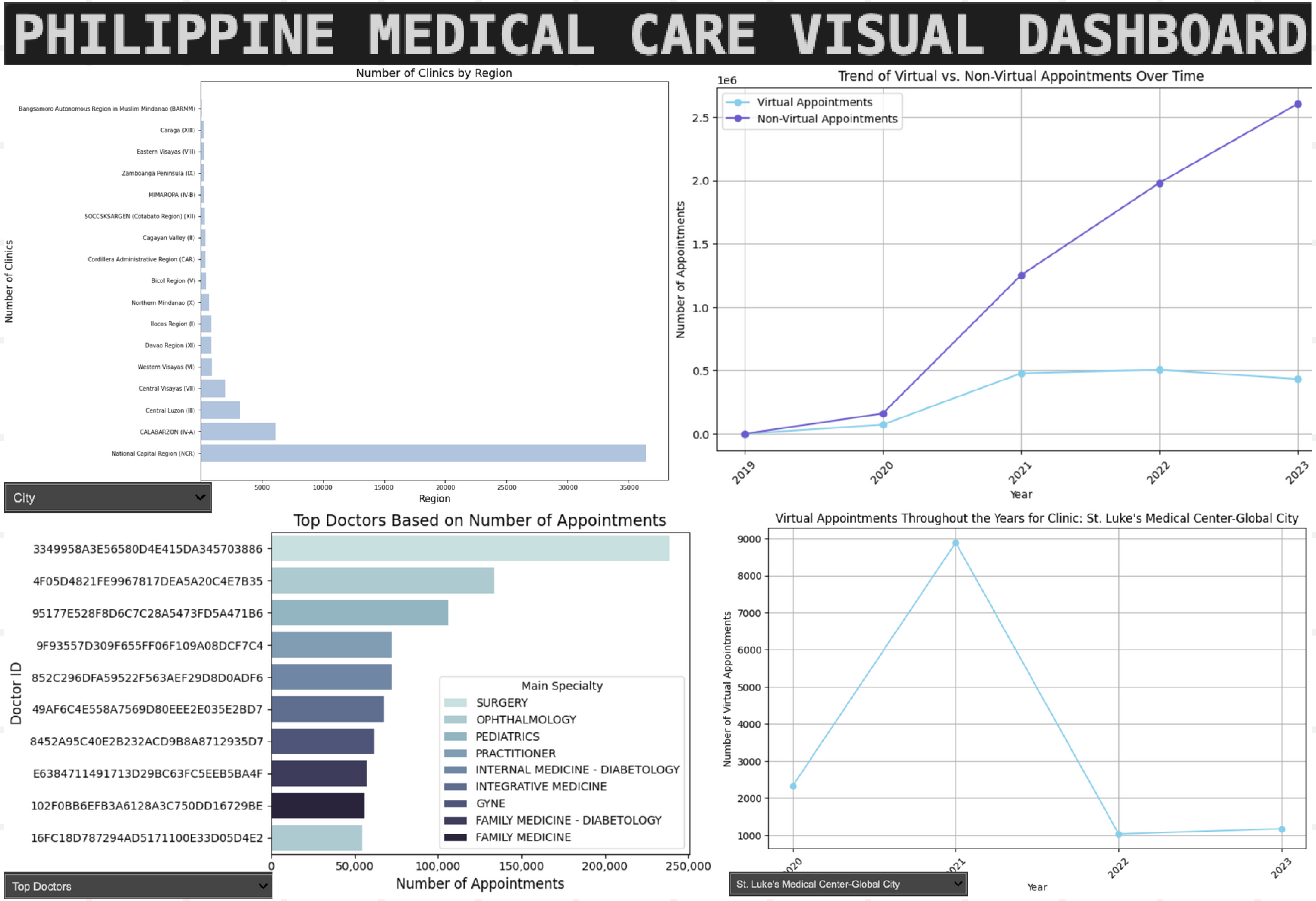
Required field

Property	Value
1	${seq:mod(3):equals(0)}
2	${seq:mod(3):equals(1)}
3	${seq:mod(3):equals(2)}
Routing Strategy	Route to Property name

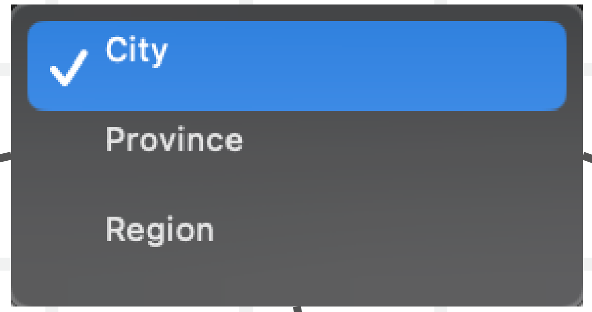


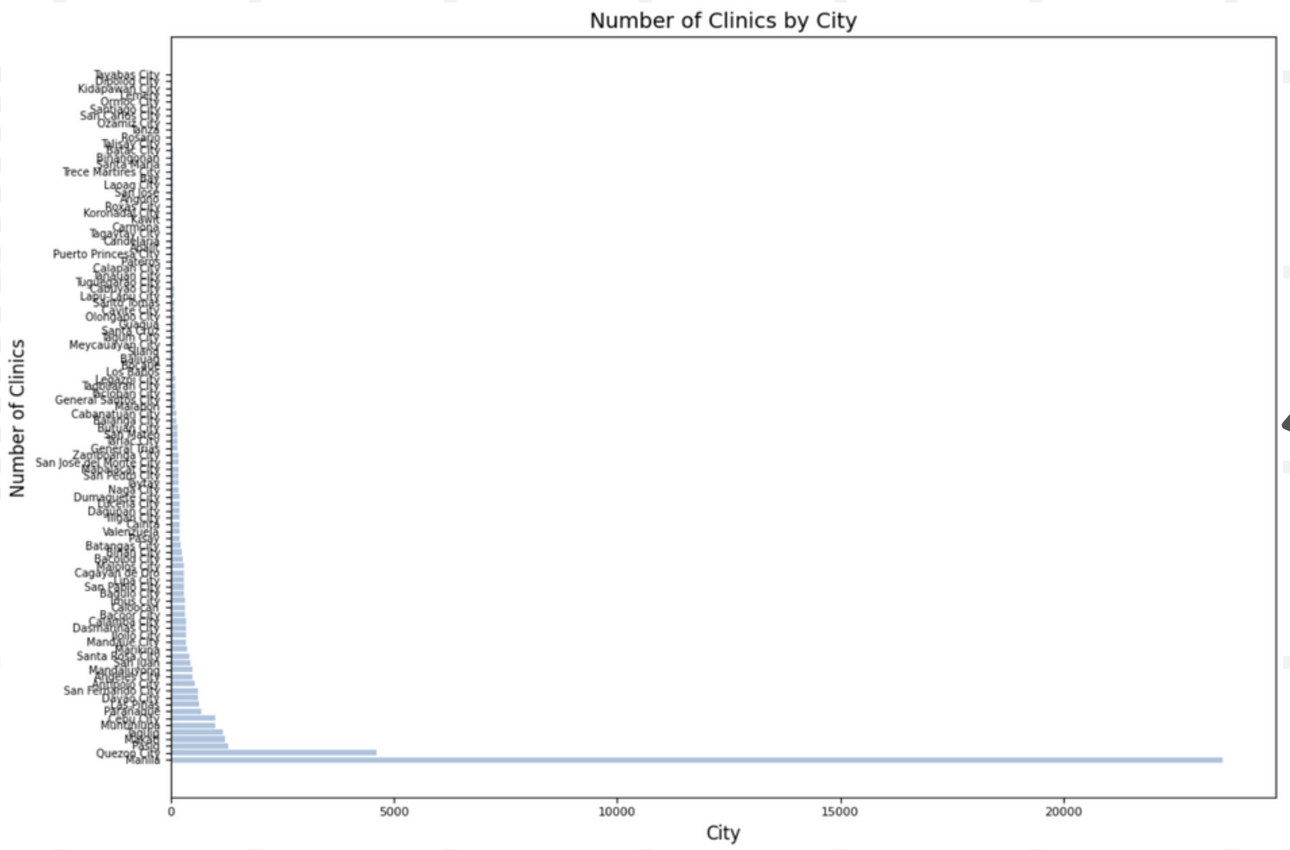
This design would allow creation of concurrent processing of flowfile in the ETL pipeline, and the amount of concurrency can be increased by simply increasing the number of routes in the processor.

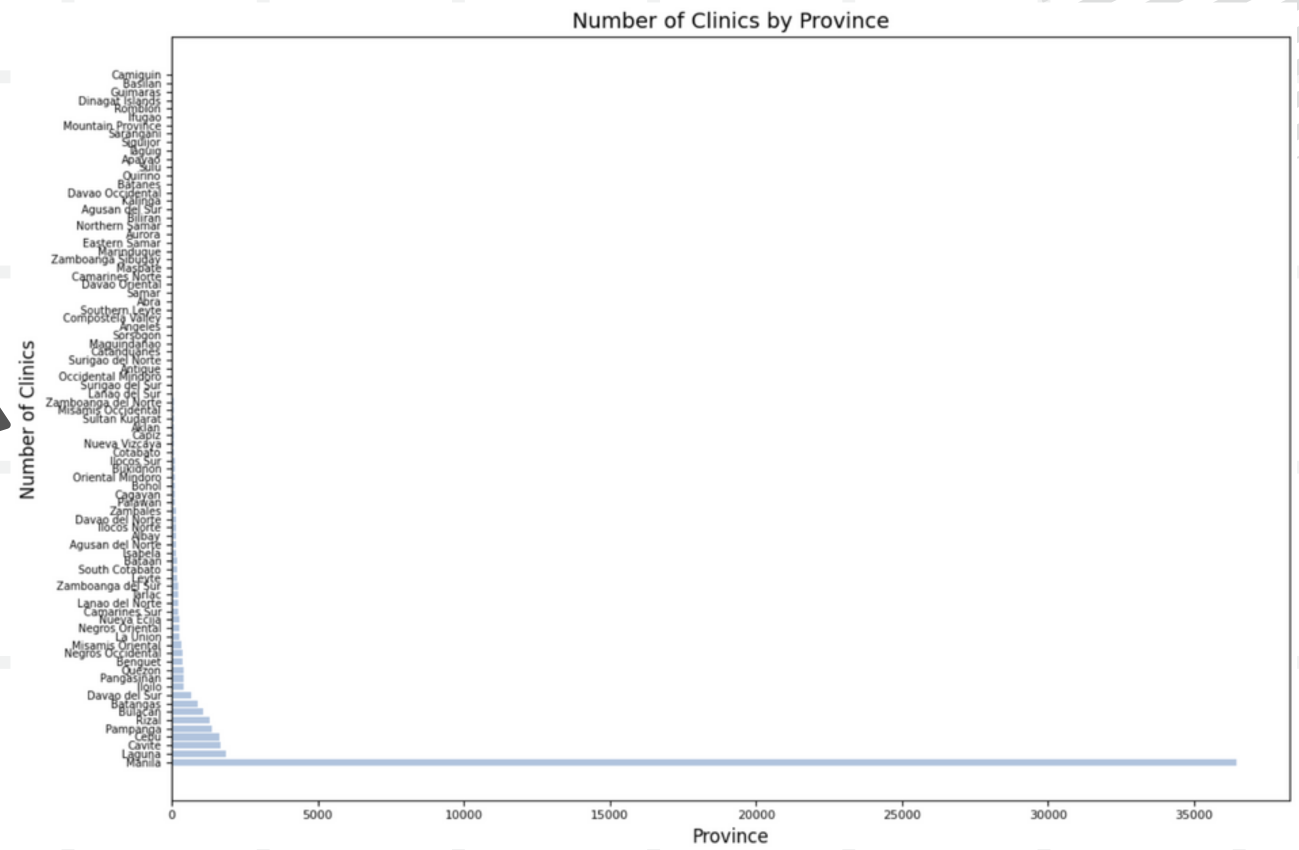
OLAP APPLICATION

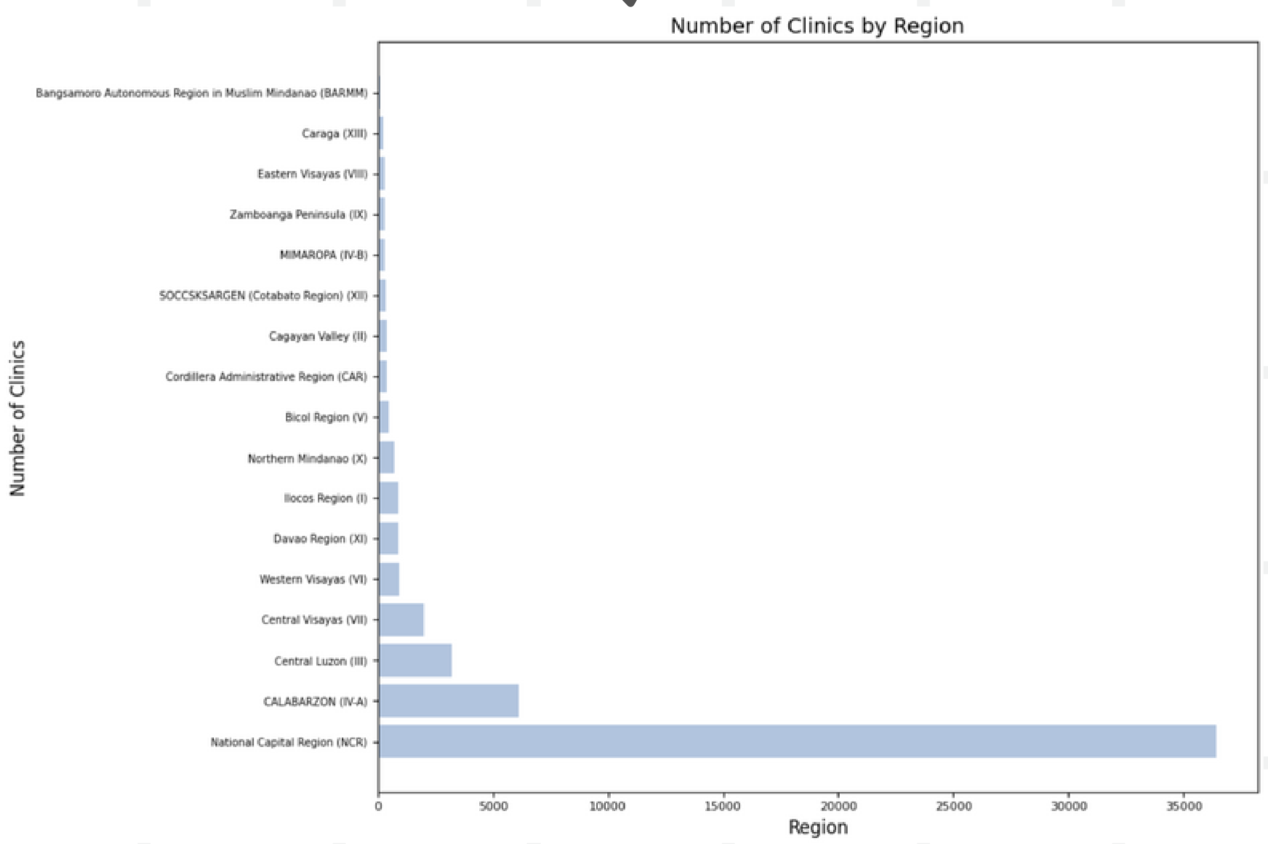


ROLL-UP: NUMBER OF CLINICS BY LOCATION

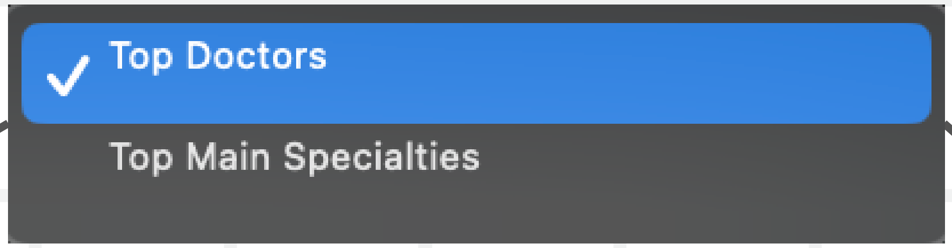


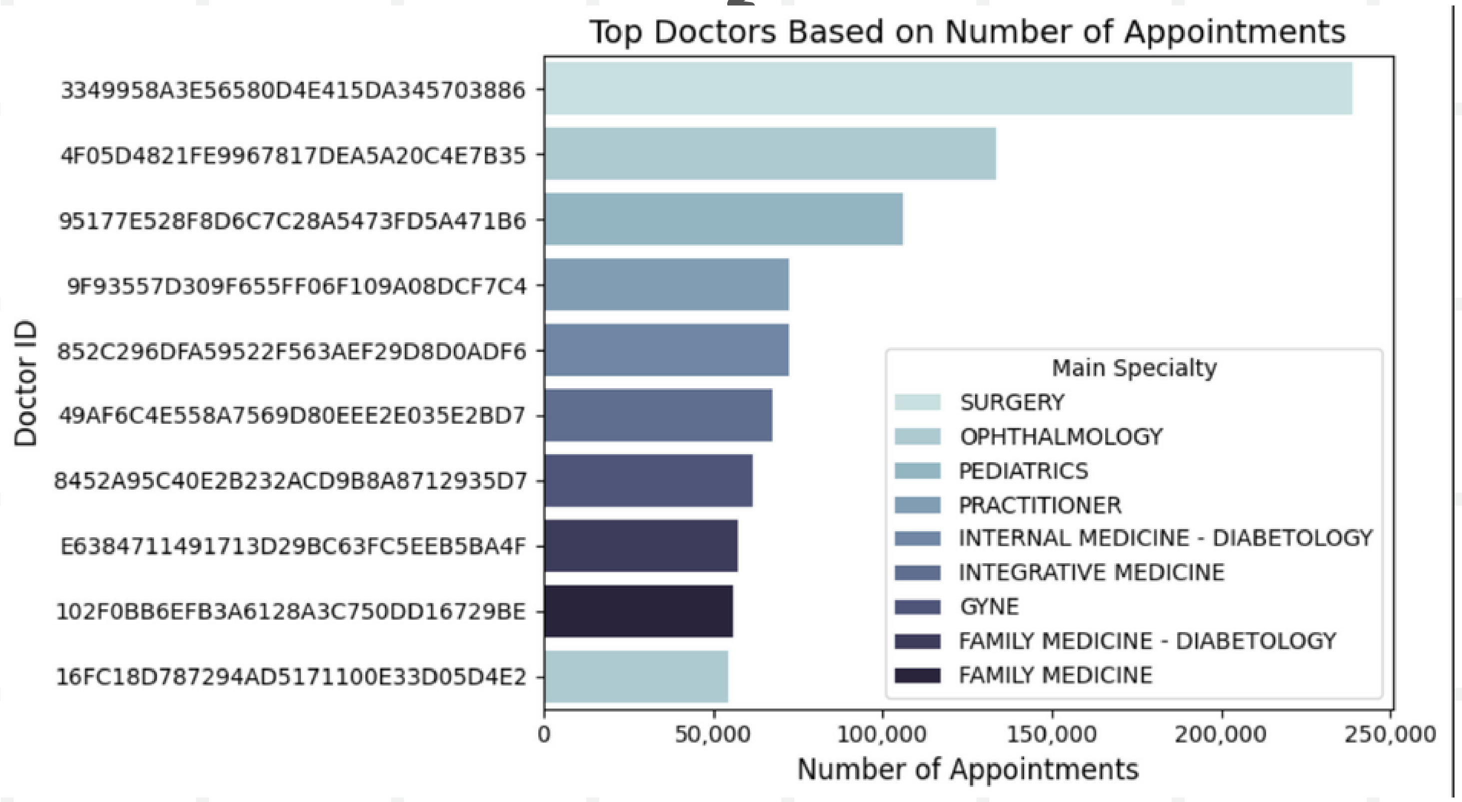


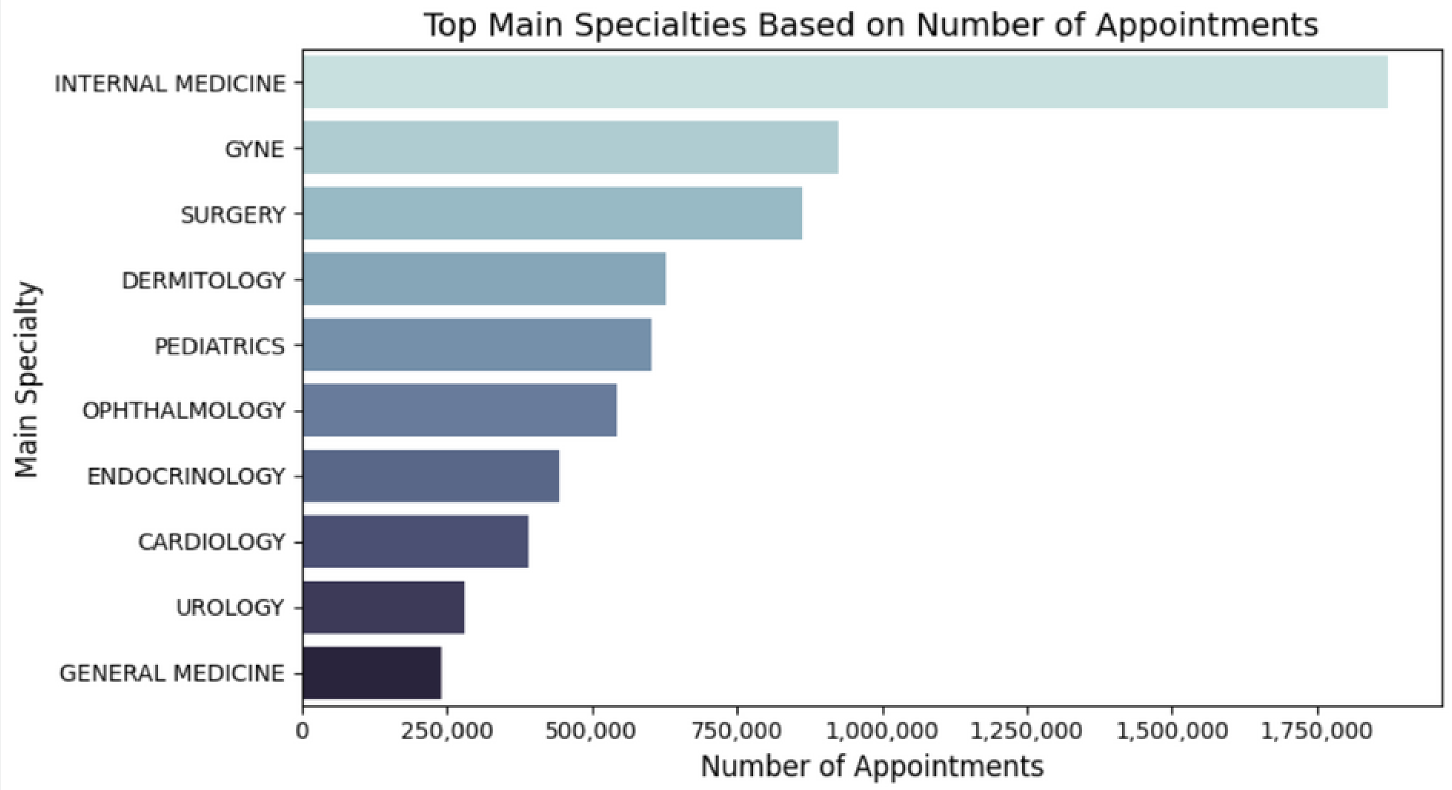




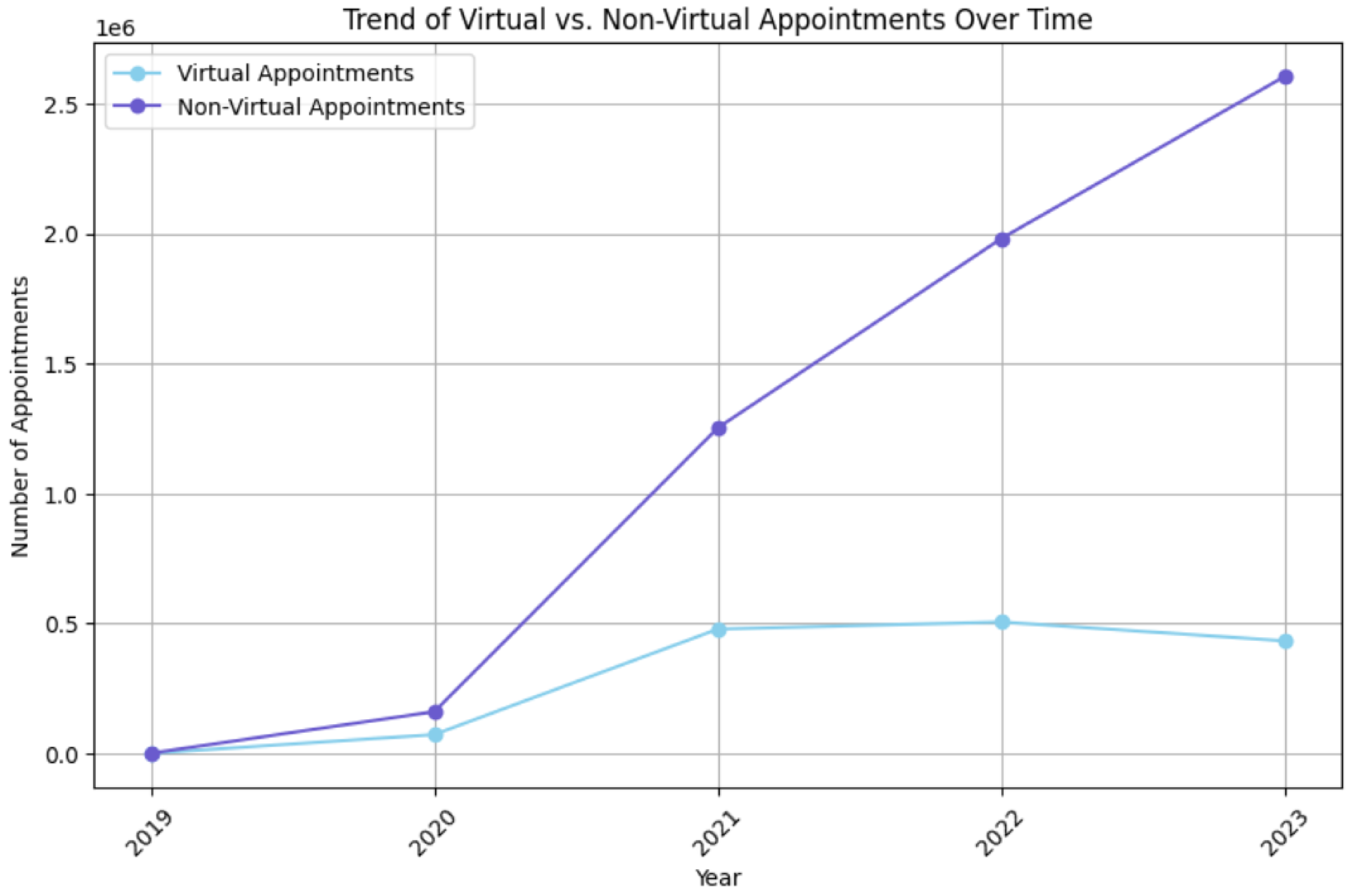
DRILL-DOWN: BASED ON NUMBER OF APPOINTMENTS





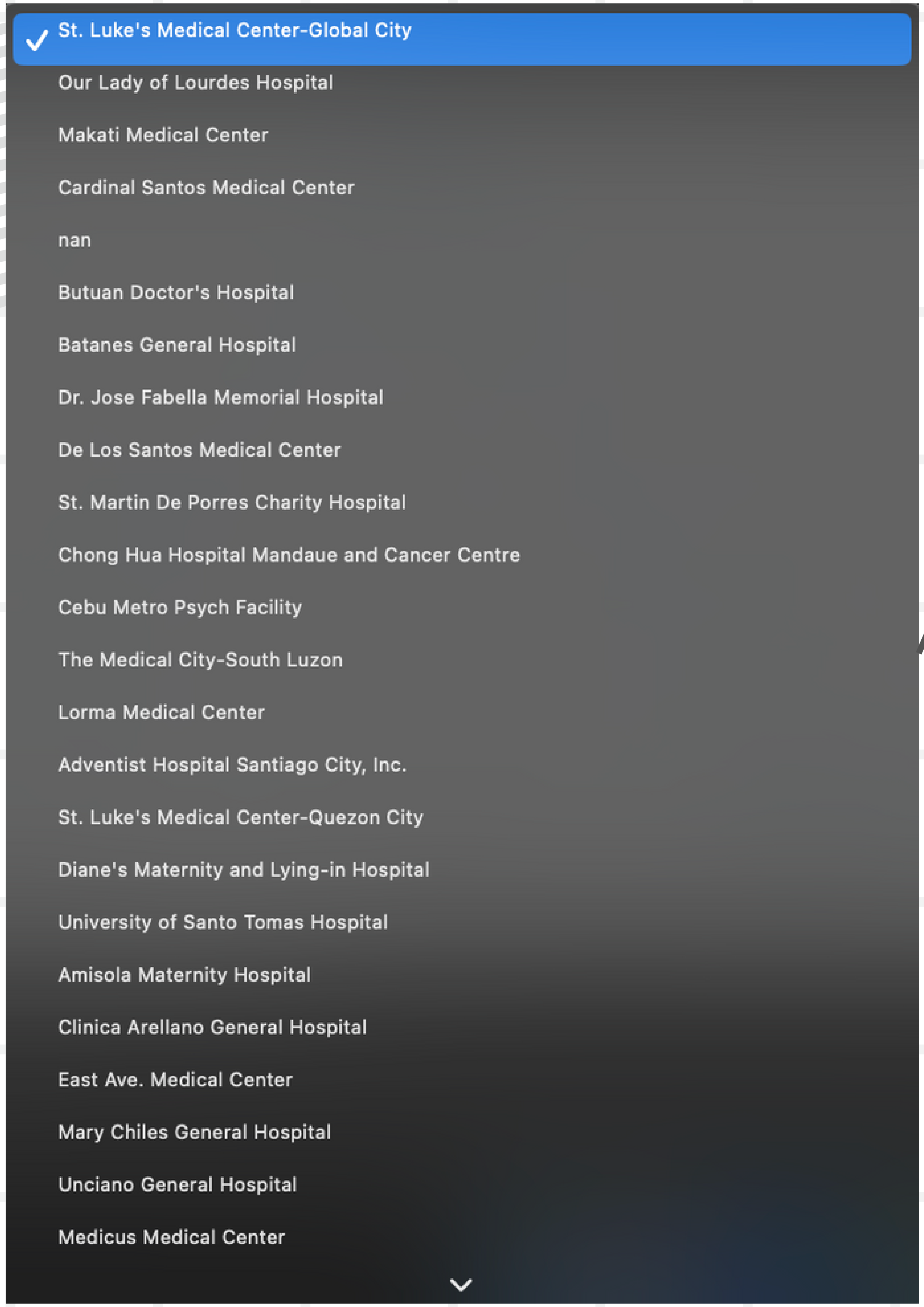


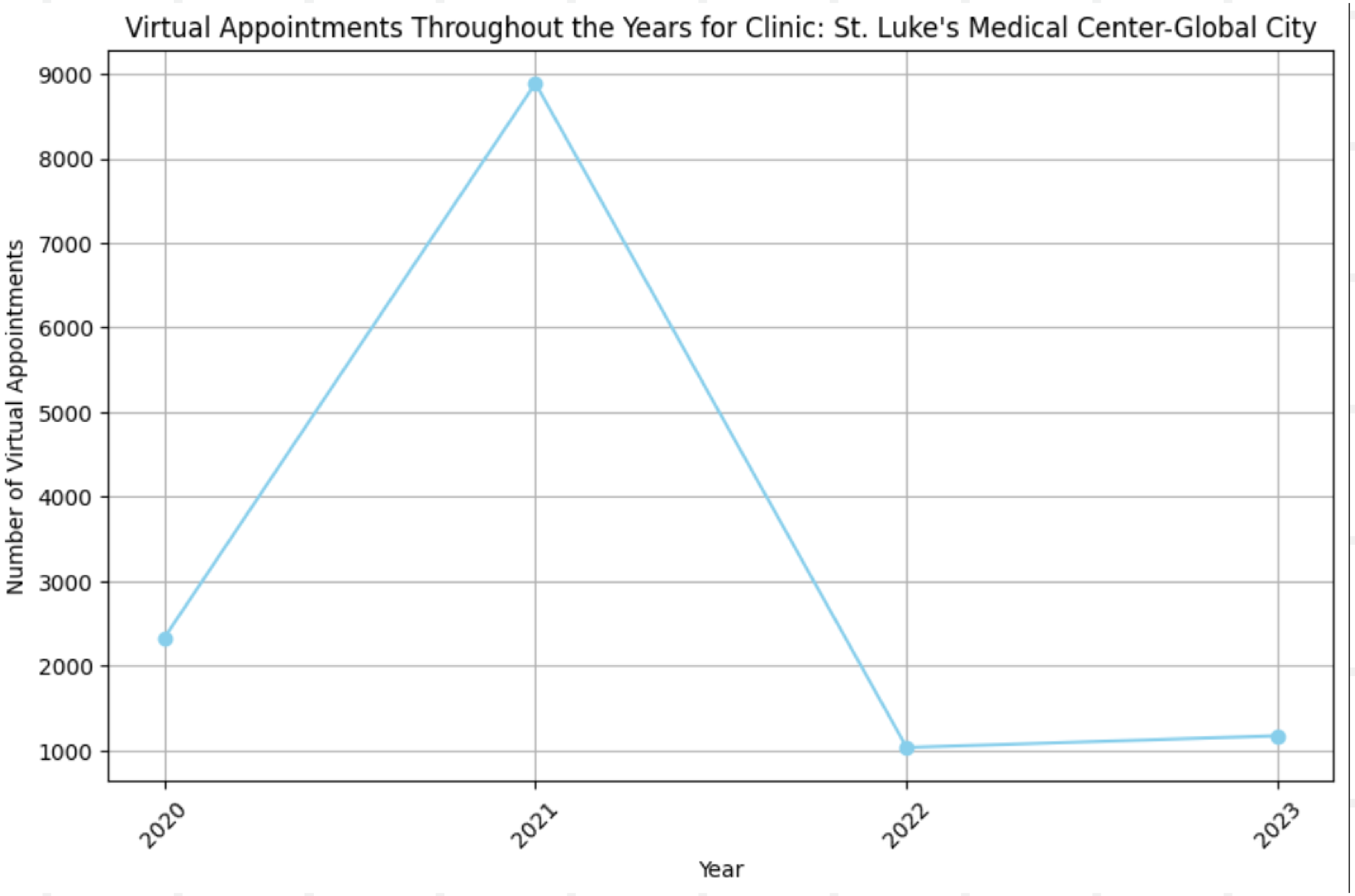
SLICE: VIRTUAL AND NON-VIRTUAL APPOINTMENTS OVER TIME




```sql
SELECT
    SUBSTR(a.QueueDate,-4) AS Year,
    COUNT(*) AS VirtualAppointmentCount
FROM
    appointments a
WHERE
    a.Virtual = 'True'
    AND SUBSTR(a.QueueDate, -4) <= '2023'
GROUP BY
    Year
ORDER BY
    Year;
```

DICE: VIRTUAL APPOINTMENTS OVER TIME PER CLINIC






```
WHERE
    a.Virtual = 'True'
    AND c.hospitalname = ?
```

QUERY OPTIMIZATION

INDEXING

Creates a separate data structure which allows the queries to retrieve and locate data in a more efficient way

Best utilized when performing GROUP BY or ORDER BY clauses, as data structure created through indexing reduces the time it takes for sorting and grouping operations by allowing the query to search for conditional data faster

PARTITIONING

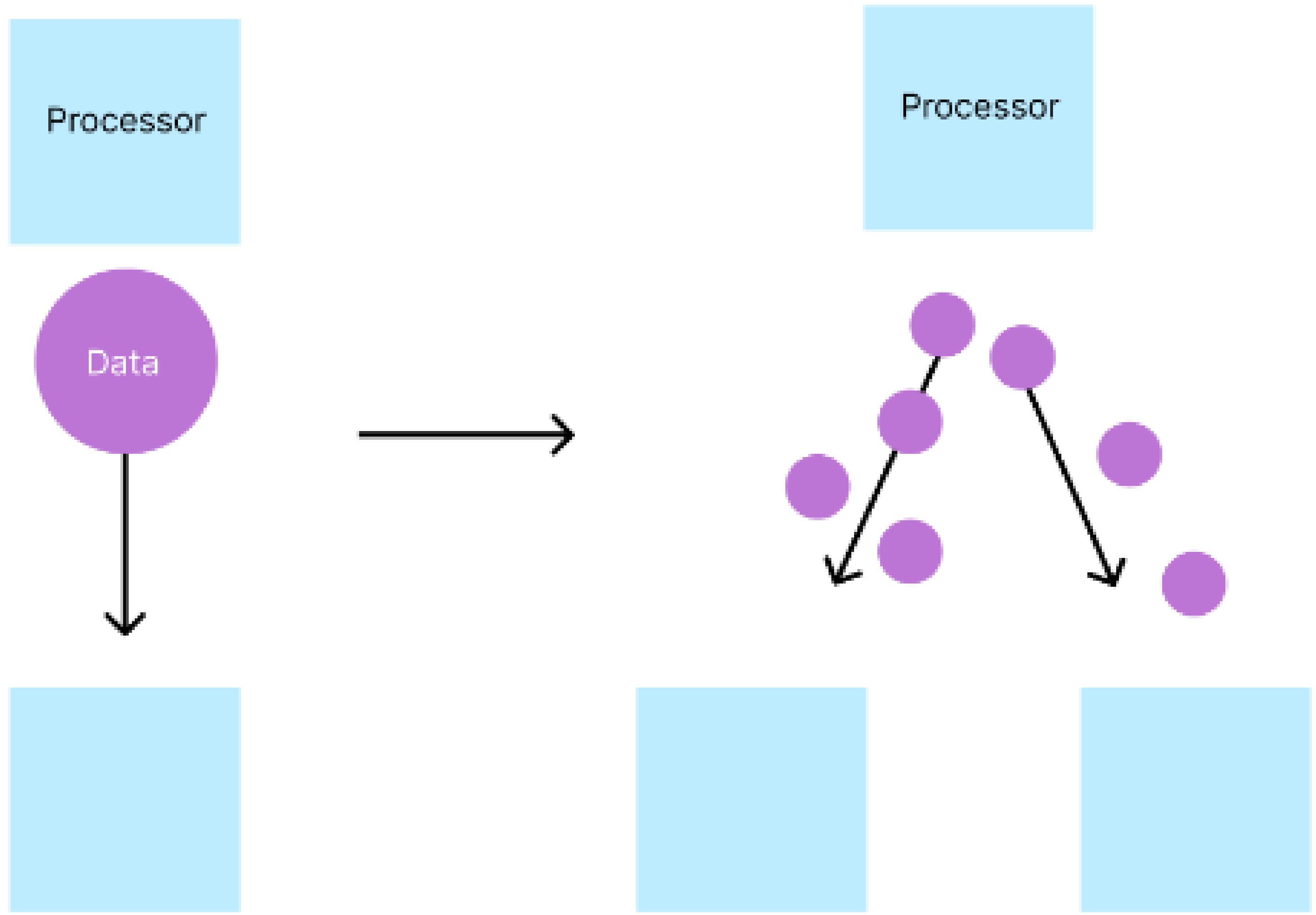
is a technique where a large amount of data is divided into a smaller group of data.

improves management and maintenance of data, while also allowing segregation of data.

PARTITIONING THE DATA

During the performance of ETL while utilizing an apache NIFI, the initial design the group created encountered a slow ETL transaction speed as there was a just a single route for the flowfile which was causing a bottleneck at the end of the pipeline.

To resolve the problem, one of the solutions applied to the ETL pipeline was to partition the data set into a bigger “split” and allow different processors to handle each segregated data separately



INDEXING THE DATA

In the later segment of the OLAP application, indexing optimization technique was utilized to decrease the query execution time of data visualization in python jupyter.

The following set of query involves costly operations such as JOIN and GROUP BY.

To reduce the query execution time of the above query, indexing can be utilized to create a separate data structure which the query can use to perform the statements faster.

Listing 3. query for top doctors based on the number of appointments

```
SELECT
   d.doctorid,
   d.mainspecialty,
   COUNT(a.apptid) AS AppointmentCount
FROM
   appointments a
JOIN
   doctors d ON a.doctorid = d.doctorid
GROUP BY
   d.doctorid,
   d.mainspecialty
ORDER BY
   AppointmentCount DESC;
```

INDEXING THE DATA

The query above is used to create a two separate index for the JOIN conditions.

The first query simply creates an index with doctorid sourced from the appointments table, while the second query creates an index with both doctorid and mainspecialty sourced from doctors table.

With the existence of an index in an appropriate column, when the SQL query is called in the OLAP application, it will automatically utilize the index created to retrieve and perform the statement faster

Listing 4. Indexing Statements

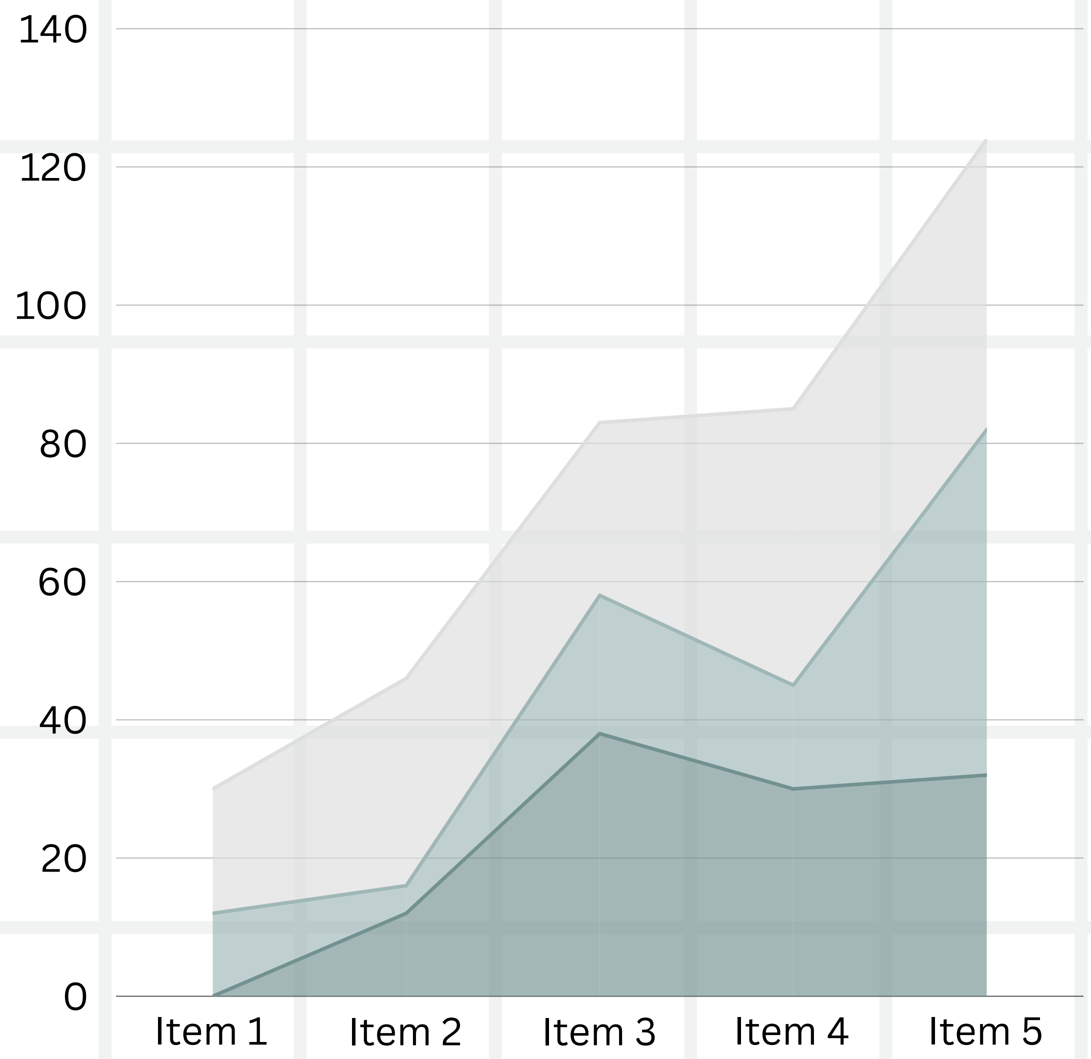
```
CREATE INDEX IF NOT EXISTS
  idx_appointments_doctorid
ON
  appointments(doctorid);

CREATE INDEX IF NOT EXISTS
  idx_doctors_doctorid_mainspecialty
ON
  doctors(doctorid, mainspecialty);
```

RESULT AND ANALYSIS

FUNCTIONAL TESTING

PERFORMANCE TESTING



FUNCTIONAL

- Functional testing aids in detecting potential defects which can be addressed early in the development process, leading to a more reliable and high-quality software product.

- When creating and designing an ETL pipeline through Apache Nifi, the group had to ensure that the pipeline is properly directing the flow files into the data warehouse schema.

FUNCTIONAL

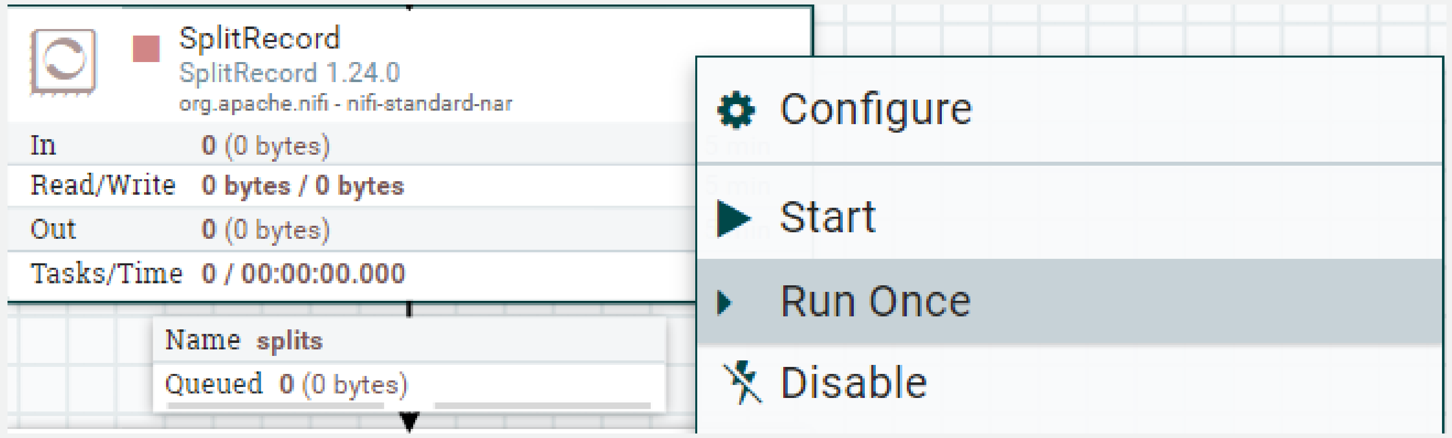
- To apply functional testing for ETL pipeline, the group has utilized a Run Once feature found in Apache Nifi. Run once feature allows its user to instruct each processor to perform just a single instance of processing allowing processor such as UpdateAttribute to retrieve just a single flow file from its upstream processor.

- We have utilized this feature to examine the flow of the ETL pipeline with a small amount of data, allowing the group to easily debug any flaws when faced with a problem.

FUNCTIONAL

- Upon verifying the correctness of the ETL pipeline design and its functionality, the group proceeded to test the structure of the pipeline with bigger data.

- When conducting a test with a bigger data, few errors which weren't present during the test cases with a small data appeared, such as the memory problem - which the group was able to resolve



PERFORMANCE

- Our group has performed a performance testing to ensure that each OLAP operation runs under 5 seconds. Performance testing of a query was conducted using the indexing optimization method we have applied to the queries.

- The group will focus on indexing optimization method for the performance testing

PERFORMANCE

- **The query which we will utilize to explain Performance Testing in this presentation is as follows:**
- **This query is a part of the OLAP application, used to generate reports on top doctors based on number of appointments in relation to their main specialty.**

```
SELECT
    d.doctorid,
    d.mainspecialty,
    COUNT(a.apptid) AS AppointmentCount
FROM
    appointments a
JOIN
    doctors d ON a.doctorid = d.doctorid
GROUP BY
    d.doctorid,
    d.mainspecialty
ORDER BY
    AppointmentCount DESC;
```

PERFORMANCE

- Initially, the group had created three separate indices, segregating the three columns used for JOIN and GROUP BY statements, namely; appointments.doctorid, doctors.doctorid, and doctors.mainspecialty.

- This segregation of indices did not affect the query execution time significantly, due to inefficient usage of indices as the query had to go through multiple indices to produce a result.

```
CREATE INDEX idx_appointments_doctorid
        ON appointments(doctorid);

CREATE INDEX idx_doctors_mainspecialty
        ON doctors(mainspecialty);

CREATE INDEX idx_doctors_doctorid
        ON doctors(doctorid);
```

PERFORMANCE

- To optimize the performance of the query, two indices were created with the following queries:

```
CREATE INDEX idx_appointments_doctorid
        ON appointments(doctorid);

CREATE INDEX idx_doctors_doctorid_mainspecialty
        ON doctors(doctorid, mainspecialty);
```

PERFORMANCE

- **Table below shows the performance results of the queries after optimizing the code with indices, with over 10 seconds reduced the optimization shows significant effect in reducing the query execution time in generating reports.**

	Trial #1	Trial #2	Trial #3	Avg
Before Optimization	11.7709 seconds	11.9590 seconds	11.9237 seconds	11.8845 seconds
Initial Optimization	8.5437 seconds	8.9351 seconds	8.9680 seconds	8.8156 seconds
Final Optimization	1.5339 seconds	1.5202 seconds	1.5242 seconds	1.5261 seconds

CONCLUSION

- The functional testing ensured the correctness and reliability of the ETL pipeline, addressing issues identified during testing with different data scales.
- The performance testing, particularly the indexing optimization of queries, led to a remarkable reduction in query execution time, meeting the requirement of running under 5 seconds. This contributes to an enhanced user experience for the OLAP application.

REFERENCES

[1] Apache Nifi. Documentation for 2.0.0-M1 - Apache Nifi. Official Documentation. November 22, 2023. [Online]. Available: https://nifi.apache.org/documentation/v2/

[2] B. Bende. “Apache Nifi - OutofMemory Error”, StackOverflow, July 29, 2019. [Online]. Available: https://stackoverflow.com/questions/38653745/apache-nifi-outofmemory-error-gc-overhead-limit-exceeded-on-splittext-process

[3] M. Hajibaba, “NiFi unable to write flowfile content to content repository error”, StackOverflow. June 27, 2022. [Online]. Available: https://stackoverflow.com/questions/72774274/nifi-unable-to-write-flowfile-content-to-content-repository

[4] M. Guru. “How to re-route flows into 4 different processors”, Cloudera. March 06, 2018. [Online]. Available: https://community.cloudera.com/t5/Support-Questions/How-to-re-route-flow-files-to-4-Different-processors/td-p/233627

[5] IBM. “What is a data warehouse?”, International Business Machines. n.d. [Online]. Available: https://www.ibm.com/topics/data-warehouse

[6] A. Roy. “What is SQL Optimization? How to do it?”, Linkedin. May 23, 2023. [Online]. Available: https://www.linkedin.com/pulse/what-sql-optimization-how-do-amit-roy/

THANK YOU