BSD2343 DATA WAREHOUSING 2022/2023
SEMESTER I

DIABETES ON HOSPITAL READMISSION RATES
Topic: Good Health and wellbeing(SDG 3)

PREPARED FOR:
DR AZUANA BINTI RAMLI

PREPARED BY: Group 11(GlucoCrew)

MATRIC ID	NAME	SECTION
SD22003	NUR ATIEKA RAFIEKAH BINTI RAZAK	01G
SD22039	NURUL SYAFIQAH NATASHA BINTI MOHD RAZI	02G
SD22053	NUR NABILAH BINTI SUZELAN AMIR	01G
SD22050	MUHAMMAD BIN ABDUL HADI	02G
SD22014	FARIS HAIKAL BIN FAIRUL RIZAL	02G

TABLE OF CONTENT

1.0 BACKGROUND

1.1 Project Background

High blood sugar is a major issue for diabetics, particularly in the hospital. Normally, our bodies use insulin to regulate blood sugar levels. Diabetes occurs when the body does not produce enough insulin (type 1) or cannot effectively use it (type 2). This will lead in high blood sugar, which, if not managed, can lead to serious complications. Doctors in intensive care care unit (ICUs) use particular instructions to keep blood sugar when they become too high. Research has shown that sugar levels consistently. This can lead to severe fluctuations in blood sugar levels, which is harmful to everyone. There is also a lack of information on how diabetes typically managed in hospital globally. This makes it difficult to assess our performance and identify areas for improvement.

This project seeks to fill this knowledge gap by examining a huge database of patient information from a US hospital and investigating how diabetes care is currently provided to patients admitted with diabetes. One specific area of interest is the HbA1c test, HbA1c stands for Haemoglobin A1c or Glycated Haemoglobin and is a type haemoglobin molecule found in red blood cells. Haemoglobin is the protein in red blood cells that carries oxygen throughout your body. This test gives clinicians are indication of a patient’s typical blood sugar control during the last few months. The researchers suggest that testing HbA1c levels of diabetic patients admitted to the hospital may be associated with a lower probability of them needing to be readmitted.

1.2 Description of Data

The diabetes data from the US is contained in the dataset we utilized for this study. This information is broken down into various tables, including those for the patient, amission, diabetic diagnoses, visit and test diabetic.

1.2.1 Patient

Variable	Description	Data Type
encounter_Id	Unique identifier of an encounter	Integer
Patient_nbr	Unique identifier of a patient	Integer
Gender	Gender of the patient; female, male, unknown/invalid	String
Age	Age group of the patients in 10 year intervals	String
Race	Race of the patient	String

1.2.2 Admission

Attributes	Description	Data Type
Encounter_Id	Unique identifier of an encounter	Integer
Patient_nbr	Unique identifier of a patient	Integer
Admission_Source_Id	The source of admission	Integer
Admission_Type_Id	Type of admission	Integer
Discharge_Disposition_Id	Identifier for the discharge status	Integer
Payer_code	Identifier for the payer	String
Time_in_hospital	Number of days between admission and discharge	Integer

1.2.3 Diagnosis

Attributes	**Descriptions**	**Data Type**
Encounter_Id	Unique identifier of an encounter	Integer
Patient_nbr	Unique identifier of a patient	Integer
diag_1	The primary diagnosis (coded as first three digits of ICD9): 715 different type	String
diag_2	Secondary diagnosis (coded as first three digits of ICD9: 743 different type	String
diag_3	Additional secondary diagnosis: 789 different type	String
number_diagnoses	Number of diagnoses entered into the system	Integer
medical_specialty	Medical specialty for diabetic patients	String

1.2.5 Visit

Attributes	**Descriptions**	**Data Type**
Encounter_Id	Unique identifier of an encounter	Integer
Patient_nbr	Unique identifier of a patient	Integer
number -emergency	Emergency visits of the patient	Integer
number-inpatient	Inpatient visits of the patient	Integer
number-outpatient	Outpatient visits of the patient	Integer

1.2.4 Test

Attributes	**Descriptions**	**Data Type**
Encounter_Id	Unique identifier of an encounter	Integer
Patient_nbr	Unique identifier of a patient	Integer
A1Cresult	Average level of blood sugar over the past 2-3 months.	String
acarbose	Medication acarbose usage indication.	String
acetohexamide	Medication acetohexamide usage indication	String
change	Change in diabetes medication	String
chlorpropamide	Medication chlorpropamide usage indication	String
citoglipton	Medication citoglipton usage indication	String
diabetesMed	Indicates if any diabetes medication was prescribed	String
examide	Medication examide usage indication	String
glimepiride	Medication glimepiride usage indication	String
glimepiride-pioglitazone	Combination medication usage indication	String
glipizide	Medication glipizide usage indication	String
glipizide-metformin	Combination medication usage indication	String
glyburide	Medication glyburide usage indication	String
glyburide-metformin	Combination medication usage indication	String
insulin	Medication insulin usage indication	String
max_glu_serum	Maximum glucose serum result:	String
metformin	Medication metformin usage indication	String
metformin-pioglitazone	Combination medication usage indication	String

metformin-rosiglitazone	Combination medication usage indication	String
miglitol	Medication miglitol usage indication	String
nateglinide	Medication nateglinide usage indication	String
pioglitazone	Medication pioglitazone usage indication	String
repaglinide	Medication repaglinide usage indication	String
rosiglitazone	Medication rosiglitazone usage indication	String
tolazamide	Medication tolazamide usage indication	String
tolbutamide	Medication tolbutamide usage indication	String
troglitazone	Medication troglitazone usage indication	String

1.3 Problem to be Solved

A comprehensive approach is needed in managing diabetes in non-ICU inpatient settings. The approach must include insights from data, evidence based practices and patient-centered care as main priorities. In these settings, however, a lack of clear guidelines often results in fragmented care and inconsistent treatment. This can have negative effects on the patients' results and how healthcare resources are used.

We are therefore here to completely alter inpatient diabetes. Our aim is to bring together state-of-the-art data analysis methods, strong clinical research, and creative ways of providing care. It is crucial that we delineate what an outpatient diabetic v clinical database entails and how it should be analyzed. We seek to examine patients' records chronologically so that we can get an overview of how diabetes is managed within non-ICU hospital settings.

The use of this evidence-based methodology will enable us to identify trends and disparities related to care delivery. Accurate strategies and directives aimed at addressing the varied needs specific to this population may thus be derived from using these figures. Our study has a focus area that examines the significance of HbA1c testing. This test depicts the extent to which diabetes management affects rates of readmissions into hospitals. The rate at which patients with diabetes are readmitted is an essential indicator of their wellbeing and the effectiveness of the healthcare system.

As such, in order to improve care for patients diagnosed with diabetes in non-ICU units, we will be using modern technology that has advanced statistical analysis capabilities, and machine learning approaches in visualizing how HbA1c levels are related to glycemic control, treatment adherence, and clinical outcomes.

1.4 Objectives

1) To explore patterns and inconsistencies in managing diabetes using data-driven insights
2) To create and apply evidence-based guidelines customized for ICU inpatient environments
3) To examine how HbA1c measurement can improve the quality of diabetes care and lower readmission rates.

1.5 Data Schema

A data schema is a collection of database objects, including tables, views, indexes, and synonyms. There are a variety of ways of arranging schema objects in the schema models designed for data warehousing. As know our dataset consist of 5 table and we will show the type and info for each table.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import psycopg2 as ps
import pandas.io.sql as sqlio
import missingno as msno
```

Figure 1.3.1 Libraries were used to find the data schema

Figure 1.3.1 shows all the libraries used to play around with the dataset in the Google Colab. By using libraries pandas and numpy we can display data schema.

	encounter_id	patient_nbr	race	gender	age
0	2278392	8222157	Caucasian	Female	[0-10)
1	149190	55629189	Caucasian	Female	[10-20)
2	64410	86047875	AfricanAmerican	Female	[20-30)
3	500364	82442376	Caucasian	Male	[30-40)
4	16680	42519267	Caucasian	Male	[40-50)

Figure 1.3.2 Patient table

```
[11] patient.dtypes

encounter_id      int64
patient_nbr       int64
race             object
gender           object
age              object
dtype: object
```

```
[12] patient.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 5 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   encounter_id  101766 non-null  int64 
 1   patient_nbr   101766 non-null  int64 
 2   race          101766 non-null  object
 3   gender        101766 non-null  object
 4   age           101766 non-null  object
dtypes: int64(2), object(3)
memory usage: 3.9+ MB
```

Figure 1.3.3 Data Schema of Patient Table

Figure 1.3.2 shows the top 5 datasets in the patient table. The patient table in this study provides demographic and identifying information for each patient involved in the dataset. then we observe the data schema for the patient table via Figure 1.3.3. It shows the data schema for the order consisting of 4 columns and several data types such as 2 attributes are integer and the left are string data type.

```python
amission= pd.read_csv("/content/Amission_Clean.csv")
amission.head()
```

	encounter_id	patient_nbr	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code
0	2278392	8222157	6	25	1	1	Other
1	149190	55629189	1	1	7	3	Other
2	64410	86047875	1	1	7	2	Other
3	500364	82442376	1	1	7	2	Other
4	16680	42519267	1	1	7	1	Other

Figure 1.3.4 Admission Table

```
[3] amission.dtypes

encounter_id                int64
patient_nbr                 int64
admission_type_id           int64
discharge_disposition_id    int64
admission_source_id         int64
time_in_hospital            int64
payer_code                 object
dtype: object
```

```
amission.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 7 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   encounter_id              101766 non-null  int64
 1   patient_nbr               101766 non-null  int64
 2   admission_type_id         101766 non-null  int64
 3   discharge_disposition_id  101766 non-null  int64
 4   admission_source_id       101766 non-null  int64
 5   time_in_hospital          101766 non-null  int64
 6   payer_code                101766 non-null  object
dtypes: int64(6), object(1)
memory usage: 5.4+ MB
```

Figure 1.3.5 Data Schema of Admission Table

Figure 1.3.4 shows the Admission dataset and Figure 1.3.5 data schema of the admission table. This table is about patient hospital admissions, including details about the source, type, and outcome of each admission, as well as financial information and the length of stay. It consists of 8 attributes with 1 string attribute which are payer_code and the others are integer data types.

```
[5] diagnoses= pd.read_csv("/content/Diagnoses_Clean.csv")
    diagnoses.head()
```

	encounter_id	patient_nbr	diag_1	diag_2	diag_3	number_diagnoses	medical_specialty
0	149190	55629189	276	250.01	255	9	NoMed
1	64410	86047875	648	250	V27	6	NoMed
2	500364	82442376	8	250.43	403	7	NoMed
3	16680	42519267	197	157	250	5	NoMed
4	35754	82637451	414	411	250	9	NoMed

Figure 1.3.6 Diagnose Tables

```
diagnoses.dtypes
```

```
encounter_id          int64
patient_nbr           int64
diag_1               object
diag_2               object
diag_3               object
number_diagnoses      int64
medical_specialty    object
dtype: object
```

```
[8] diagnoses.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100244 entries, 0 to 100243
Data columns (total 7 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   encounter_id       100244 non-null  int64
 1   patient_nbr        100244 non-null  int64
 2   diag_1             100244 non-null  object
 3   diag_2             100244 non-null  object
 4   diag_3             100244 non-null  object
 5   number_diagnoses   100244 non-null  int64
 6   medical_specialty  100244 non-null  object
dtypes: int64(3), object(4)
memory usage: 5.4+ MB
```

Figure 1.3.7 Data Schema of Diagnose Table

Based on figure 1.3.6 shows the top 5 data about the diagnosis table that provides detailed information about the diagnoses recorded for each patient encounter. From Figure 1.3.4 we can see that there are 7 total columns where three of which are integer data types, and the other four columns which are diag_1, diag_2, diag_3, medical_specialty are string data types.

```
[16] visit= pd.read_csv("/content/Visit_Clean.csv")
     visit.head()
```

	encounter_id	patient_nbr	number_outpatient	number_emergency	number_inpatient
0	2278392	8222157	0	0	0
1	149190	55629189	0	0	0
2	64410	86047875	2	0	1
3	500364	82442376	0	0	0
4	16680	42519267	0	0	0

Figure 1.3.8 Visit Table

```
[17] visit.dtypes

encounter_id         int64
patient_nbr          int64
number_outpatient    int64
number_emergency     int64
number_inpatient     int64
dtype: object
```

```
visit.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 5 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   encounter_id       101766 non-null  int64
 1   patient_nbr        101766 non-null  int64
 2   number_outpatient  101766 non-null  int64
 3   number_emergency   101766 non-null  int64
 4   number_inpatient   101766 non-null  int64
dtypes: int64(5)
memory usage: 3.9 MB
```

Figure 1.3.9 Data Schema of Visit Table

The visit table shown in Figure 1.3.8, contains information about the different types of healthcare visits a patient had in the year preceding a specific encounter. It includes data on emergency, inpatient, and outpatient visits. The table consists of five attributes with the same data type which are integers.

```python
[13] test= pd.read_csv("/content/Test_Clean.csv")
     test.head()
```

	encounter_id	patient_nbr	metformin	repaglinide	nateglinide	chlorpropamide	glimepiride	acetohexamide	glipizide	glyburide	...	examide	citoglipton	insulin	glyburidemetformin	glipizidemetformin
0	2278392	8222157	No	No	No	No	No	No	No	No	...	No	No	No	No	No
1	149190	55629189	No	No	No	No	No	No	No	No	...	No	No	Up	No	No
2	64410	86047875	No	No	No	No	No	No	Steady	No	...	No	No	No	No	No
3	500364	82442376	No	No	No	No	No	No	No	No	...	No	No	Up	No	No
4	16680	42519267	No	No	No	No	No	No	Steady	No	...	No	No	Steady	No	No

5 rows × 27 columns

Figure 1.3.10 Test Table

```python
test.dtypes
```

```
encounter_id                    int64
patient_nbr                     int64
metformin                      object
repaglinide                    object
nateglinide                    object
chlorpropamide                 object
glimepiride                    object
acetohexamide                  object
glipizide                      object
glyburide                      object
tolbutamide                    object
pioglitazone                   object
rosiglitazone                  object
acarbose                       object
miglitol                       object
troglitazone                   object
tolazamide                     object
examide                        object
citoglipton                    object
insulin                        object
glyburidemetformin             object
glipizidemetformin             object
glimepiridepioglitazone        object
metforminrosiglitazone         object
metforminpioglitazone          object
change                         object
diabetesmed                    object
dtype: object
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 27 columns):
 #   Column                   Non-Null Count   Dtype 
---  ------                   --------------   ----- 
 0   encounter_id             101766 non-null  int64 
 1   patient_nbr              101766 non-null  int64 
 2   metformin                101766 non-null  object
 3   repaglinide              101766 non-null  object
 4   nateglinide              101766 non-null  object
 5   chlorpropamide           101766 non-null  object
 6   glimepiride              101766 non-null  object
 7   acetohexamide            101766 non-null  object
 8   glipizide                101766 non-null  object
 9   glyburide                101766 non-null  object
 10  tolbutamide              101766 non-null  object
 11  pioglitazone             101766 non-null  object
 12  rosiglitazone            101766 non-null  object
 13  acarbose                 101766 non-null  object
 14  miglitol                 101766 non-null  object
 15  troglitazone             101766 non-null  object
 16  tolazamide               101766 non-null  object
 17  examide                  101766 non-null  object
 18  citoglipton              101766 non-null  object
 19  insulin                  101766 non-null  object
 20  glyburidemetformin       101766 non-null  object
 21  glipizidemetformin       101766 non-null  object
 22  glimepiridepioglitazone  101766 non-null  object
 23  metforminrosiglitazone   101766 non-null  object
 24  metforminpioglitazone    101766 non-null  object
 25  change                   101766 non-null  object
 26  diabetesmed              101766 non-null  object
```

Figure 1.3.11 Data Schema of Test Table

Figure 1.3.10 shows the test table contains information related to medical tests conducted during patient encounters, particularly focusing on tests and medications relevant to diabetes management. Then we look up figure 1.3.11 to know more about the data schema of the test table. It consists of only two attributes with numeric data type and 25 attributes are string. To sum up, the visit table consists of 27 columns or attributes.

2.0 ARCHITECTURE
2.1 Pipeline Structure

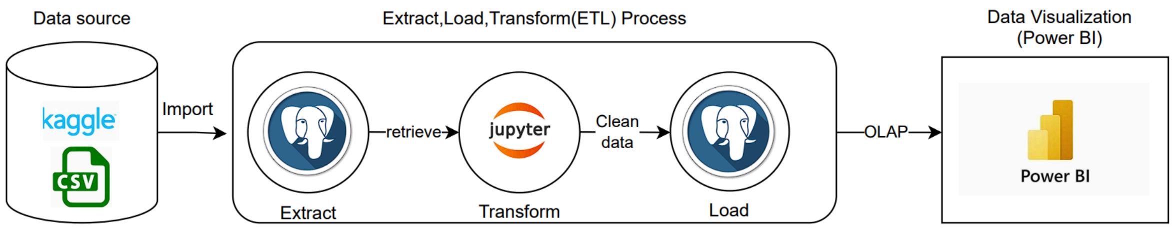


Figure 2.1 Pipeline Structure

In light of this, Kimball's technique is most suitable for our project because it involves data processing, which is central to our objectives of establishing a relationship between the diabetic cases and US. With this technique; more customized data marts can be created besides improving the speed that the results are delivered. We can easily analyze and certainly forecast certain factors for risks patients, diagnosis and tests using various data marts. Figure 2. 1, tells about survey dataset collected from kaggle. The dataset includes five tables: patient, amission, diagnosis, test, and visit as the most frequently used ones. The first steps before analysis can be to load the dataset to PostgreSQL where we create a database and tables. He directly got the tables through importing and after that, he operated on the data on the help of Jupyter.

First of all, we set up the Pandas library that is responsible for data manipulation and necessities such as cleaning, loading as well as storing the data. We have uploaded the datasets,scanning the data types of each column and missing values;we have also removed the incomplete data. There then is a link between these tables to have an integrated view of the results and the data generated is then written into new CSV files. When we import these files again into PostgreSQL, we again employ Python for data preprocessing. Some data may contain NULL values, to eliminate them you need to install the required libraries and set up a connection. This process ensure that data is well transformed as well as cleaned hence being prepared for the next step which is reporting. The data is then stored in new CSV file and it's again being imported back into PostgreSQL after data cleaning and transformation processes. Once the data has been loaded, some operations such as OLAP operations can be performed to enable visualization and multidimensional analysis. These involves operations, such as, cube , roll-up, slicing, dicing, and drill-down operations that may offer further details of the data. Lastly, after using OLAP processes, PowerBI is used to open the outcomes as well as to create visualization. Microsoft PowerBI can easily work on an integration of raw data and make it meaningful information through engaging and interactive visualizations.

in six successive phases.

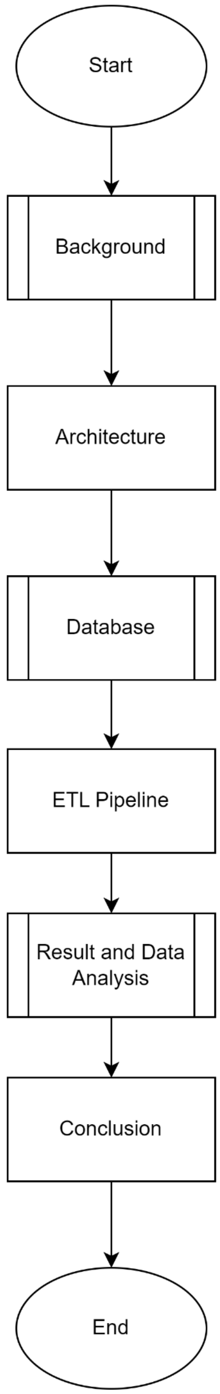


Figure 2.2 Flow of this project

Figure 2.2 illustrates the comprehensive sequence of our project, which we will execute in six successive phases.

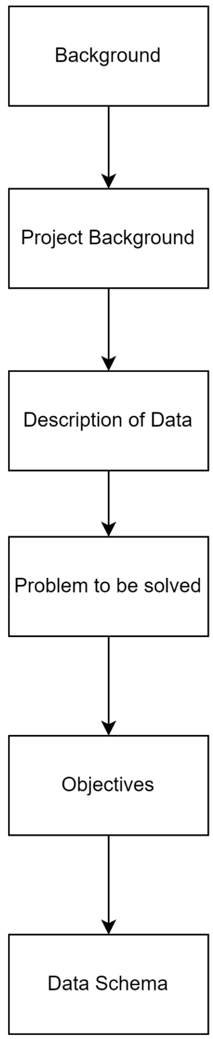


Figure 2.3 Process of The Background

Figure 2.3 illustrates the background process, detailing the project's context, the data description, the problem to be addressed, the project's objectives, and the data schema. The data used in this study was sourced from Kaggle. Subsequently, the architecture was designed to ensure the project proceeded smoothly and as planned.

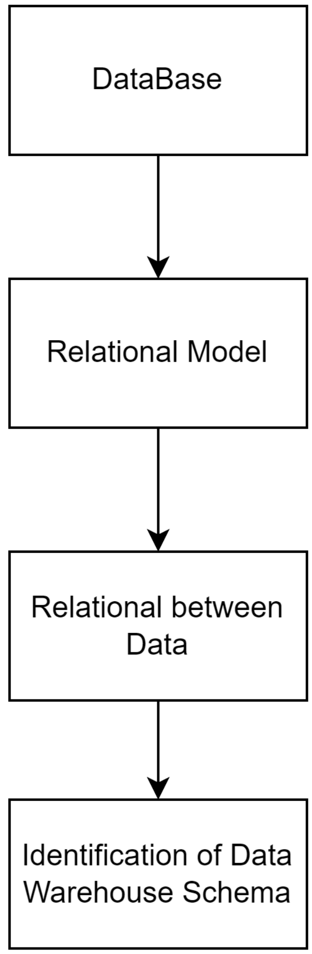


Figure 2.3 Process of Database

Figure 2.4 illustrates the database process, covering the relational model, the relationships within the model, and the identification of the data warehouse schema. For this project, our group utilized Microsoft Power BI and pgAdmin to create the relational model. We then implemented the Extract, Transform, and Load (ETL) pipeline using Jupyter Notebook and pgAdmin. The raw data was extracted into Jupyter Notebook for cleaning and merging. Subsequently, the clean data was loaded into pgAdmin for the OLAP process, and into Tableau and Power BI for data visualization.

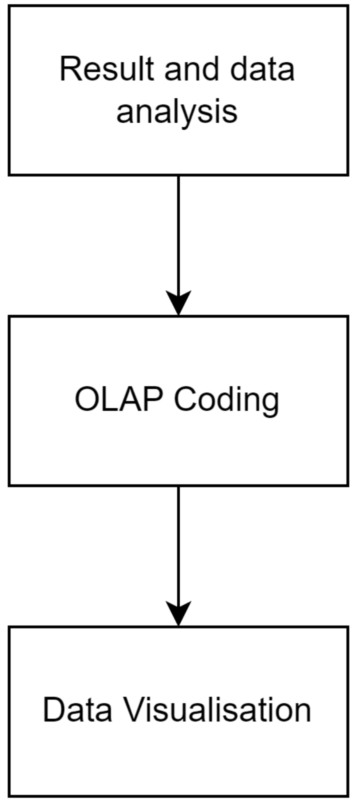


Figure 2.4 Process of Result and Data Analysis

Figure 2.5 illustrates the findings and data analysis, including data visualization. We will use pgAdmin for data analysis tasks such as roll-up and slicing. Following this, Microsoft Power BI and Tableau will be employed to create the data visualizations. Based on the results of the data analysis and visualization, we will draw conclusions.

3.0 ETL PIPELINE

3.1 ETL Pipeline

Figure 3.1 ETL Pipeline

The dataset for this project was sourced from Kaggle and is on heart diseases. It contains five tables: test, visit, diagnosis, amission, and patient relate to each other. First, we integrate data into the PostgreSQL data warehouse that initiates the overall ETL process. Through Jupter Notebook, they retrieve these tables from the PostgreSQL database for data cleaning as well as integration.

This extraction method enables us realize certain procedures on the data. These includes exercises like formatting the data properly and ensuring that the data is clean and ready for use. To clean the data and organize it appropriately, we conduct data cleaning and transformations and write the cleaned data to a new CSV file. Last but not the least, the clean transformed data are then worked on and pushed back into the PostgreSQL database for further analytical and visualization purposes in the next process step.

3.2 ETL Process

3.2.1 Extract

Create new database:

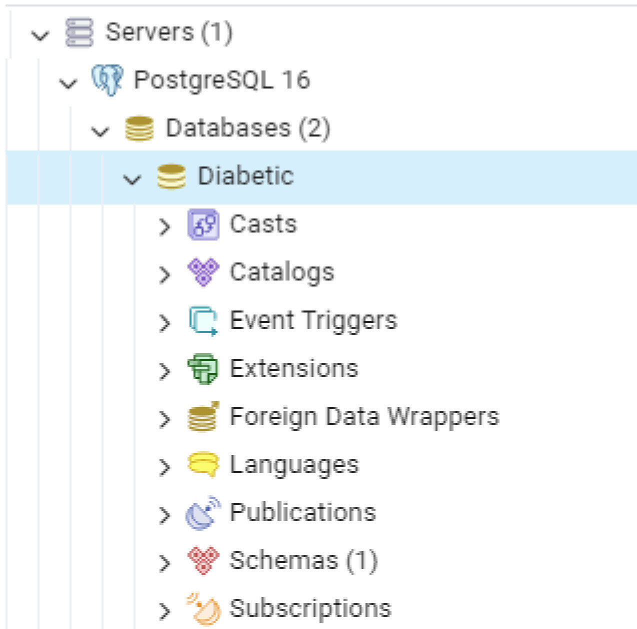


Figure 3.2.1.1 Database in PostgreSQL

Figure 3.2.1 shows that we have created a database named 'Diabetic' in PostgreSQL.

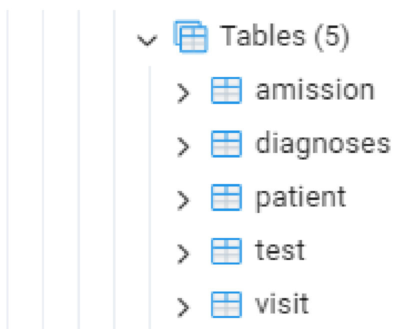


Figure 3.2.1.2 Tables created

Query to create tables:

```
Create table Amission
(
	encounter_id int,
	patient_nbr int,
	admission_type_id int,
	discharge_disposition_id int,
	admission_source_id int,
	time_in_hospital int,
	payer_code text
);

Create table Diagnoses
(
	encounter_id int,
	patient_nbr int,
	diag_1 text,
	diag_2 text,
	diag_3 text,
	number_diagnoses int,
	medical_specialty text
);

Create table Patient
(
	encounter_id int,
	patient_nbr int,
	race text,
	gender text,
	age text,
	weight text
);

Create table Test
(
	encounter_id int,
	patient_nbr int,
	max_glu_serum text,
	A1Cresult text,
	metformin text,
	repaglinide text,
	nateglinide text,
	chlorpropamide text,
	glimepiride text,
	acetohexamide text,
```

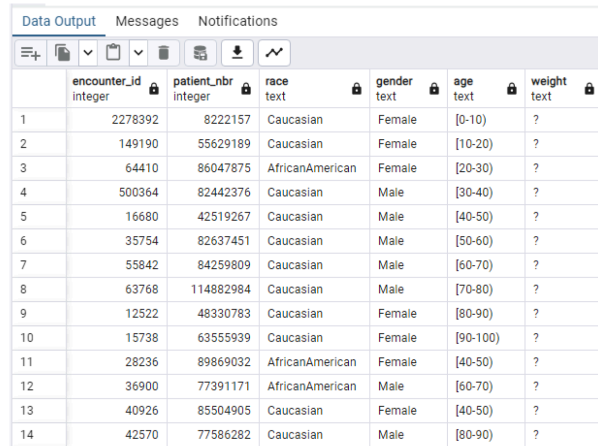
```
        glipizide text,
        glyburide text,
        tolbutamide text,
        pioglitazone text,
        rosiglitazone text,
        acarbose text,
        miglitol text,
        troglitazone text,
        tolazamide text,
        examide text,
        citoglipton text,
        insulin text,
        glyburideMetformin text,
        glipizideMetformin text,
        glimepiridePioglitazone text,
        metforminRosiglitazone text,
        metforminPioglitazone text,
        change text,
        diabetesMed text
);

Create table Visit
(
        encounter_id int,
        patient_nbr int,
        number_outpatient int,
        number_emergency int,
        number_inpatient int
);

COPY amission
FROM
'C:\Users\faris\OneDrive\Desktop\Draft\Unclean\Amission_Table_UnClean
.csv'
DELIMITER ','
CSV HEADER;

COPY diagnoses
FROM
'C:\Users\faris\OneDrive\Desktop\Draft\Unclean\Diabetic_Diagnoses_UnC
lean.csv'
DELIMITER ','
CSV HEADER;

COPY patient
```

```
FROM
'C:\Users\faris\OneDrive\Desktop\Draft\Unclean\Patient_Table_UnClean.
csv'
DELIMITER ','
CSV HEADER;

COPY test
FROM
'C:\Users\faris\OneDrive\Desktop\Draft\Unclean\Test_Diabetic_UnClean.
csv'
DELIMITER ','
CSV HEADER;

COPY visit
FROM
'C:\Users\faris\OneDrive\Desktop\Draft\Unclean\Visits_Table_UnClean.c
sv'
DELIMITER ','
CSV HEADER;
```

<u>Run a query (Select * from ‘table name’) to view the data in the table</u>

<table>
<tr><th>QUERY</th><th>OUTPUT</th></tr>
<tr><td><code>SELECT * FROM amission</code></td><td>
Data Output Messages Notifications
<table>
<tr><th></th><th>encounter_id
integer</th><th>patient_nbr
integer</th><th>admission_type_id
integer</th><th>discharge_disposition_id
integer</th><th>admission_source_id
integer</th><th>time_in_hospital
integer</th><th>payer_code
text</th></tr>
<tr><td>1</td><td>2278392</td><td>8222157</td><td>6</td><td>25</td><td>1</td><td>1</td><td>?</td></tr>
<tr><td>2</td><td>149190</td><td>55629189</td><td>1</td><td>1</td><td>7</td><td>3</td><td>?</td></tr>
<tr><td>3</td><td>64410</td><td>86047875</td><td>1</td><td>1</td><td>7</td><td>2</td><td>?</td></tr>
<tr><td>4</td><td>500364</td><td>82442376</td><td>1</td><td>1</td><td>7</td><td>2</td><td>?</td></tr>
<tr><td>5</td><td>16680</td><td>42519267</td><td>1</td><td>1</td><td>7</td><td>1</td><td>?</td></tr>
<tr><td>6</td><td>35754</td><td>82637451</td><td>2</td><td>1</td><td>2</td><td>3</td><td>?</td></tr>
<tr><td>7</td><td>55842</td><td>84259809</td><td>3</td><td>1</td><td>2</td><td>4</td><td>?</td></tr>
<tr><td>8</td><td>63768</td><td>114882984</td><td>1</td><td>1</td><td>7</td><td>5</td><td>?</td></tr>
<tr><td>9</td><td>12522</td><td>48330783</td><td>2</td><td>1</td><td>4</td><td>13</td><td>?</td></tr>
<tr><td>10</td><td>15738</td><td>63555939</td><td>3</td><td>3</td><td>4</td><td>12</td><td>?</td></tr>
<tr><td>11</td><td>28236</td><td>89869032</td><td>1</td><td>1</td><td>7</td><td>9</td><td>?</td></tr>
<tr><td>12</td><td>36900</td><td>77391171</td><td>2</td><td>1</td><td>4</td><td>7</td><td>?</td></tr>
<tr><td>13</td><td>40926</td><td>85504905</td><td>1</td><td>3</td><td>7</td><td>7</td><td>?</td></tr>
<tr><td>14</td><td>42570</td><td>77586282</td><td>1</td><td>6</td><td>7</td><td>10</td><td>?</td></tr>
</table>
</td></tr>
<tr><td><code>SELECT * FROM diagnoses</code></td><td>
Data Output Messages Notifications
<table>
<tr><th></th><th>encounter_id
integer</th><th>patient_nbr
integer</th><th>diag_1
text</th><th>diag_2
text</th><th>diag_3
text</th><th>number_diagnoses
integer</th><th>medical_specialty
text</th></tr>
<tr><td>1</td><td>2278392</td><td>8222157</td><td>250.83</td><td>?</td><td>?</td><td>1</td><td>Pediatrics-Endocrinology</td></tr>
<tr><td>2</td><td>149190</td><td>55629189</td><td>276</td><td>250.01</td><td>255</td><td>9</td><td>?</td></tr>
<tr><td>3</td><td>64410</td><td>86047875</td><td>648</td><td>250</td><td>V27</td><td>6</td><td>?</td></tr>
<tr><td>4</td><td>500364</td><td>82442376</td><td>8</td><td>250.43</td><td>403</td><td>7</td><td>?</td></tr>
<tr><td>5</td><td>16680</td><td>42519267</td><td>197</td><td>157</td><td>250</td><td>5</td><td>?</td></tr>
<tr><td>6</td><td>35754</td><td>82637451</td><td>414</td><td>411</td><td>250</td><td>9</td><td>?</td></tr>
<tr><td>7</td><td>55842</td><td>84259809</td><td>414</td><td>411</td><td>V45</td><td>7</td><td>?</td></tr>
<tr><td>8</td><td>63768</td><td>114882984</td><td>428</td><td>492</td><td>250</td><td>8</td><td>?</td></tr>
<tr><td>9</td><td>12522</td><td>48330783</td><td>398</td><td>427</td><td>38</td><td>8</td><td>?</td></tr>
<tr><td>10</td><td>15738</td><td>63555939</td><td>434</td><td>198</td><td>486</td><td>8</td><td>InternalMedicine</td></tr>
<tr><td>11</td><td>28236</td><td>89869032</td><td>250.7</td><td>403</td><td>996</td><td>9</td><td>?</td></tr>
<tr><td>12</td><td>36900</td><td>77391171</td><td>157</td><td>288</td><td>197</td><td>7</td><td>?</td></tr>
<tr><td>13</td><td>40926</td><td>85504905</td><td>428</td><td>250.43</td><td>250.6</td><td>8</td><td>Family/GeneralPractice</td></tr>
<tr><td>14</td><td>42570</td><td>77586282</td><td>428</td><td>411</td><td>427</td><td>8</td><td>Family/GeneralPractice</td></tr>
</table>
</td></tr>
</table>

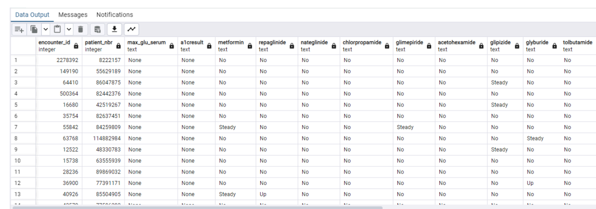
```
SELECT * FROM patient
```



Data Output Messages Notifications

	encounter_id integer	patient_nbr integer	race text	gender text	age text	weight text
1	2278392	8222157	Caucasian	Female	[0-10)	?
2	149190	55629189	Caucasian	Female	[10-20)	?
3	64410	86047875	AfricanAmerican	Female	[20-30)	?
4	500364	82442376	Caucasian	Male	[30-40)	?
5	16680	42519267	Caucasian	Male	[40-50)	?
6	35754	82637451	Caucasian	Male	[50-60)	?
7	55842	84259809	Caucasian	Male	[60-70)	?
8	63768	114882984	Caucasian	Male	[70-80)	?
9	12522	48330783	Caucasian	Female	[80-90)	?
10	15738	63555939	Caucasian	Female	[90-100)	?
11	28236	89869032	AfricanAmerican	Female	[40-50)	?
12	36900	77391171	AfricanAmerican	Male	[60-70)	?
13	40926	85504905	Caucasian	Female	[40-50)	?
14	42570	77586282	Caucasian	Male	[80-90)	?

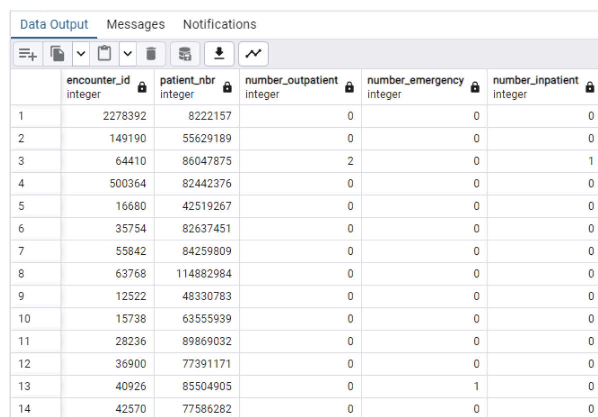
```
SELECT * FROM test
```



Data Output Messages Notifications

	encounter_id integer	patient_nbr integer	max_glu_serum text	a1cresult text	metformin text	repaglinide text	nateglinide text	chlorpropamide text	glimepiride text	acetohexamide text	glipizide text	glyburide text	tolbutamide text
1	2278392	8222157	None	None	No	No	No	No	No	No	No	No	No
2	149190	55629189	None	None	No	No	No	No	No	No	No	No	No
3	64410	86047875	None	None	No	No	No	No	No	No	Steady	No	No
4	500364	82442376	None	None	No	No	No	No	No	No	No	No	No
5	16680	42519267	None	None	No	No	No	No	No	No	Steady	No	No
6	35754	82637451	None	None	No	No	No	No	No	No	No	No	No
7	55842	84259809	None	None	Steady	No	No	No	Steady	No	No	No	No
8	63768	114882984	None	None	No	No	No	No	No	No	No	Steady	No
9	12522	48330783	None	None	No	No	No	No	No	No	Steady	No	No
10	15738	63555939	None	None	No	No	No	No	No	No	No	No	No
11	28236	89869032	None	None	No	No	No	No	No	No	No	No	No
12	36900	77391171	None	None	No	No	No	No	No	No	No	Up	No
13	40926	85504905	None	None	Steady	Up	No	No	No	No	No	No	No

```
SELECT * FROM visit
```



Data Output Messages Notifications

	encounter_id integer	patient_nbr integer	number_outpatient integer	number_emergency integer	number_inpatient integer
1	2278392	8222157	0	0	0
2	149190	55629189	0	0	0
3	64410	86047875	2	0	1
4	500364	82442376	0	0	0
5	16680	42519267	0	0	0
6	35754	82637451	0	0	0
7	55842	84259809	0	0	0
8	63768	114882984	0	0	0
9	12522	48330783	0	0	0
10	15738	63555939	0	0	0
11	28236	89869032	0	0	0
12	36900	77391171	0	0	0
13	40926	85504905	0	1	0
14	42570	77586282	0	0	0

After the raw data has been extracted into pgAdmin, we need to connect our pgAdmin with the Jupyter Notebook to proceed the next step which transforms the data.

Before starting the process, we are required to install a few packages.

- ! pip install ipython-sql
- ! pip install sqlalchemy
- ! pip install psycopg2
- ! pip install python-sql
- ! pip install pandas-sql
- ! pip install sql-queries
- ! pip install missingno

After install all these packages, we need to load ipython-sql using the following command:

```
%reload_ext sql
[3] ✓ 0.0s
```

Figure 3.2.1.3 This figure shows the load of ipython-sql.

Call the create engine function:

```
from sqlalchemy import create_engine
[4] ✓ 0.0s
```

Figure 3.2.1.4 This figure shows a call to create engine.

```
import pandas as pd
import psycopg2 as ps
import pandas.io.sql as sqlio
import missingno as msno
[7] ✓ 5.6s
```

Figure 3.2.1.5 Import necessary libraries for the ETL process.

```
conn=ps.connect(dbname="Diabetic",
                user="postgres",password= "1234",host="localhost",
                port="5432")
```

[10] 0.0s

Figure 3.2.1.6 Connect the PgAdmin with Jupyter Notebook.

```
sql="""SELECT * FROM pg_catalog.pg_tables"""
```

[11] 0.0s

```
sql="""SELECT * FROM amission """
```

[12] 0.0s

```
df1 = sqlio.read_sql_query(sql,conn)
df1
```

[14] 0.7s

Figure 3.2.1.7 Extract data from PgAdmin to Jupyter Notebook.

3.2.2 Transforms

After the connecting process, then the data needs to be cleaned as the data cleaning process is a crucial part in data processing. Some connectors were installed to ensure that data can be transferred from PostgreSQL to Python. To make it easy for the cleaning process, the data can be stored into data frame using pandas library.

Table amission:

```
df1 = sqlio.read_sql_query(sql,conn)
df1
```

[7] ✓ 0.2s

```
C:\Users\faris\AppData\Local\Temp\ipykernel_16172\1573172249.py:1: UserWarning: pandas only supports SQLAlchemy connectable
  df1 = sqlio.read_sql_query(sql,conn)
```

	encounter_id	patient_nbr	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code
0	2278392	8222157	6	25	1	1	?
1	149190	55629189	1	1	7	3	?
2	64410	86047875	1	1	7	2	?
3	500364	82442376	1	1	7	2	?
4	16680	42519267	1	1	7	1	?
...	...	...	...	...	...	...	...
101761	443847548	100162476	1	3	7	3	MC
101762	443847782	74694222	1	4	5	5	MC
101763	443854148	41088789	1	1	7	1	MC
101764	443857166	31693671	2	3	7	10	MC
101765	443867222	175429310	1	1	7	6	?

101766 rows × 7 columns

Figure 3.2.2.1 data store into the data frame.

```
df1.replace('?', pd.NA, inplace=True)
df1.isnull().sum()
```

[8] ✓ 0.0s

```
encounter_id                 0
patient_nbr                  0
admission_type_id            0
discharge_disposition_id     0
admission_source_id          0
time_in_hospital             0
payer_code               40256
dtype: int64
```

Figure 3.2.2.2 Checking missing values.

```
df1['payer_code'] = df1['payer_code'].fillna(value='Other')
df1.head()
```

[9] ✓ 0.0s

	encounter_id	patient_nbr	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	payer_code
0	2278392	8222157	6	25	1	1	Other
1	149190	55629189	1	1	7	3	Other
2	64410	86047875	1	1	7	2	Other
3	500364	82442376	1	1	7	2	Other
4	16680	42519267	1	1	7	1	Other

Figure 3.2.2.3 Modify to replace the missing value with 'Other'.

```
newamission = df1
df1.isnull().sum()
```

[10] ✓ 0.0s

```
encounter_id                0
patient_nbr                 0
admission_type_id           0
discharge_disposition_id    0
admission_source_id         0
time_in_hospital            0
payer_code                  0
dtype: int64
```

Figure 3.2.2.4 Check again if missing value still available.

```
#Extract to csv file
df1.to_csv('Amission_Clean.csv')
```

[44] ✓ 0.4s

Figure 3.2.2.5 Extract the clean data to csv file.

Table diagnoses:

```
sql2="""SELECT * FROM diagnoses """
df2=sqlio.read_sql_query(sql2,conn)
df2
```

[12] ✓ 0.2s

```
C:\Users\faris\AppData\Local\Temp\ipykernel_16172\901037663.py:2: UserWarning: pandas only support
  df2=sqlio.read_sql_query(sql2,conn)
```

	encounter_id	patient_nbr	diag_1	diag_2	diag_3	number_diagnoses	medical_specialty
0	2278392	8222157	250.83	?	?	1	Pediatrics-Endocrinology
1	149190	55629189	276	250.01	255	9	?
2	64410	86047875	648	250	V27	6	?
3	500364	82442376	8	250.43	403	7	?
4	16680	42519267	197	157	250	5	?
...	...	...	...	...	...	...	...
101761	443847548	100162476	250.13	291	458	9	?
101762	443847782	74694222	560	276	787	9	?
101763	443854148	41088789	38	590	296	13	?
101764	443857166	31693671	996	285	998	9	Surgery-General
101765	443867222	175429310	530	530	787	9	?

101766 rows × 7 columns

Figure 3.2.2.6 data store into the data frame.

```
# Check total of missing value
df2.replace('?', pd.NA, inplace=True)
df2.isnull().sum()
```

[13] ✓ 0.0s

```
encounter_id             0
patient_nbr              0
diag_1                  21
diag_2                 358
diag_3                1423
number_diagnoses         0
medical_specialty    49949
dtype: int64
```

Figure 3.2.2.7 Checking missing values.

```python
# Modify the missing value be 'NoMed'
df2['medical_specialty'] = df2['medical_specialty'].fillna(value='NoMed')
df2.head()
```

[14] ✓ 0.0s

	encounter_id	patient_nbr	diag_1	diag_2	diag_3	number_diagnoses	medical_specialty
0	2278392	8222157	250.83	<NA>	<NA>	1	Pediatrics-Endocrinology
1	149190	55629189	276	250.01	255	9	NoMed
2	64410	86047875	648	250	V27	6	NoMed
3	500364	82442376	8	250.43	403	7	NoMed
4	16680	42519267	197	157	250	5	NoMed

Figure 3.2.2.8 Modify the missing value for medical_specialty to be 'NoMed'.

```python
# Delete rows that contains missing value
df2.dropna(inplace=True)
df2.head()
```

[15] ✓ 0.0s

	encounter_id	patient_nbr	diag_1	diag_2	diag_3	number_diagnoses	medical_specialty
1	149190	55629189	276	250.01	255	9	NoMed
2	64410	86047875	648	250	V27	6	NoMed
3	500364	82442376	8	250.43	403	7	NoMed
4	16680	42519267	197	157	250	5	NoMed
5	35754	82637451	414	411	250	9	NoMed

Figure 3.2.2.9 Delete rows for the missing value.

```
# Check total of missing value
newdiagnoses = df2
df2.replace('?', pd.NA, inplace=True)
df2.isnull().sum()
```

[16] ✓ 0.0s

```
encounter_id         0
patient_nbr          0
diag_1               0
diag_2               0
diag_3               0
number_diagnoses     0
medical_specialty    0
dtype: int64
```

Figure 3.2.2.10 Checking missing values if still available.

```
#Extract to csv file
df2.to_csv('Diagnoses_Clean.csv')
```

[17] ✓ 0.1s

Figure 3.2.2.11 Extract the clean data to csv file.

Table Patient:

```
sql3="""SELECT * FROM patient """
df3=sqlio.read_sql_query(sql3,conn)
df3
```

[18] ✓ 0.1s

```
C:\Users\faris\AppData\Local\Temp\ipykernel_16172\1111652906.py:2: UserWarning:
  df3=sqlio.read_sql_query(sql3,conn)
```

	encounter_id	patient_nbr	race	gender	age	weight
0	2278392	8222157	Caucasian	Female	[0-10)	?
1	149190	55629189	Caucasian	Female	[10-20)	?
2	64410	86047875	AfricanAmerican	Female	[20-30)	?
3	500364	82442376	Caucasian	Male	[30-40)	?
4	16680	42519267	Caucasian	Male	[40-50)	?
...	...	...	...	...	...	...
101761	443847548	100162476	AfricanAmerican	Male	[70-80)	?
101762	443847782	74694222	AfricanAmerican	Female	[80-90)	?
101763	443854148	41088789	Caucasian	Male	[70-80)	?
101764	443857166	31693671	Caucasian	Female	[80-90)	?
101765	443867222	175429310	Caucasian	Male	[70-80)	?

101766 rows × 6 columns

Figure 3.2.2.12 data store into the data frame.

```
# Check total of missing value
df3.replace('?', pd.NA, inplace=True)
df3.isnull().sum()
```

[19] ✓ 0.0s

```
encounter_id        0
patient_nbr         0
race             2273
gender              0
age                 0
weight          98569
dtype: int64
```

Figure 3.2.2.13 Checking missing values.

```
# Delete column
del df3['weight']
df3.head()
```

[20] ✓ 0.0s

	encounter_id	patient_nbr	race	gender	age
0	2278392	8222157	Caucasian	Female	[0-10)
1	149190	55629189	Caucasian	Female	[10-20)
2	64410	86047875	AfricanAmerican	Female	[20-30)
3	500364	82442376	Caucasian	Male	[30-40)
4	16680	42519267	Caucasian	Male	[40-50)

Figure 3.2.2.14 Delete column because missing values >80%.

```
# Modify missing data
df3['race'] = df3['race'].fillna(value='Other')
df3.head()
```

[21] ✓ 0.0s

	encounter_id	patient_nbr	race	gender	age
0	2278392	8222157	Caucasian	Female	[0-10)
1	149190	55629189	Caucasian	Female	[10-20)
2	64410	86047875	AfricanAmerican	Female	[20-30)
3	500364	82442376	Caucasian	Male	[30-40)
4	16680	42519267	Caucasian	Male	[40-50)

Figure 3.2.2.15 Modify column race for missing value to be 'Other'.

```
# Check total of missing value
newpatient = df3
df3.replace('?', pd.NA, inplace=True)
df3.isnull().sum()
```
[22] ✓ 0.0s

```
encounter_id    0
patient_nbr     0
race            0
gender          0
age             0
dtype: int64
```

Figure 3.2.2.16 Checking again missing values if available.

```
#Extract to csv file
df3.to_csv('Patient_Clean.csv')
```
[23] ✓ 0.1s

Figure 3.2.2.17 Extract the clean data to csv file.

Table Test:

```
sql4="""SELECT * FROM test """
df4=sqlio.read_sql_query(sql4,conn)
df4
```

[24] ✓ 1.0s

```
C:\Users\faris\AppData\Local\Temp\ipykernel_16172\752704432.py:2: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAP
  df4=sqlio.read_sql_query(sql4,conn)
```

	encounter_id	patient_nbr	max_glu_serum	a1cresult	metformin	repaglinide	nateglinide	chlorpropamide	glimepiride	acetohexamide	...	examide	citoglipton	insulin	glyburidemetf
0	2278392	8222157	None	None	No	No	No	No	No	No	...	No	No	No	
1	149190	55629189	None	None	No	No	No	No	No	No	...	No	No	Up	
2	64410	86047875	None	None	No	No	No	No	No	No	...	No	No	No	
3	500364	82442376	None	None	No	No	No	No	No	No	...	No	No	Up	
4	16680	42519267	None	None	No	No	No	No	No	No	...	No	No	Steady	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
101761	443847548	100162476	None	>8	Steady	No	No	No	No	No	...	No	No	Down	
101762	443847782	74694222	None	None	No	No	No	No	No	No	...	No	No	Steady	
101763	443854148	41088789	None	None	Steady	No	No	No	No	No	...	No	No	Down	
101764	443857166	31693671	None	None	No	No	No	No	No	No	...	No	No	Up	
101765	443867222	175429310	None	None	No	No	No	No	No	No	...	No	No	No	

101766 rows × 29 columns

Figure 3.2.2.18 Data store into the data frame.

```
# Check total of missing value
df4.replace('?', pd.NA, inplace=True)
df4.isnull().sum()
```

[25] ✓ 0.5s

```
encounter_id                0
patient_nbr                 0
max_glu_serum               0
a1cresult                   0
metformin                   0
repaglinide                 0
nateglinide                 0
chlorpropamide              0
glimepiride                 0
acetohexamide               0
glipizide                   0
glyburide                   0
tolbutamide                 0
pioglitazone                0
rosiglitazone               0
acarbose                    0
miglitol                    0
troglitazone                0
tolazamide                  0
examide                     0
citoglipton                 0
insulin                     0
glyburidemetformin          0
glipizidemetformin          0
glimepiridepioglitazone     0
metforminrosiglitazone      0
metforminpioglitazone       0
change                      0
diabetesmed                 0
dtype: int64
```

Figure 3.2.2.19 Checking missing values.

```
# Delete column
columns_to_delete = ['max_glu_serum']
newtest = df4
df4.drop(columns=columns_to_delete, inplace=True)
df4.head()
```

[38] ✓ 0.1s

	encounter_id	patient_nbr	metformin	repaglinide	nateglinide	chlorpropamide	glimepiride	acetohexamide	glipizide	glyburide
0	2278392	8222157	No	No	No	No	No	No	No	No
1	149190	55629189	No	No	No	No	No	No	No	No
2	64410	86047875	No	No	No	No	No	No	Steady	No
3	500364	82442376	No	No	No	No	No	No	No	No
4	16680	42519267	No	No	No	No	No	No	Steady	No

5 rows × 27 columns

Figure 3.2.2.20 Delete column that didnt use.

```
#Extract to csv file
df4.to_csv('Test_Clean.csv')
```

[27] ✓ 0.4s

Figure 3.2.2.21 Extract the clean data to csv file.

Table Visit:

```
sql5="""SELECT * FROM visit """
df5=sqlio.read_sql_query(sql5,conn)
df5
```

[28] ✓ 0.2s

```
C:\Users\faris\AppData\Local\Temp\ipykernel_16172\2857390204.py:2: UserWarning: pandas only
  df5=sqlio.read_sql_query(sql5,conn)
```

	encounter_id	patient_nbr	number_outpatient	number_emergency	number_inpatient
0	2278392	8222157	0	0	0
1	149190	55629189	0	0	0
2	64410	86047875	2	0	1
3	500364	82442376	0	0	0
4	16680	42519267	0	0	0
...	...	...	...	...	...
101761	443847548	100162476	0	0	0
101762	443847782	74694222	0	0	1
101763	443854148	41088789	1	0	0
101764	443857166	31693671	0	0	1
101765	443867222	175429310	0	0	0

101766 rows × 5 columns

Figure 3.2.2.22 data store into the data frame.

```
# Check total of missing value
newvisit = df5
df5.replace('?', pd.NA, inplace=True)
df5.isnull().sum()
```

[29] ✓ 0.0s

```
encounter_id         0
patient_nbr          0
number_outpatient    0
number_emergency     0
number_inpatient     0
dtype: int64
```

Figure 3.2.2.23 Checking missing values.

```
#Extract to csv file
df5.to_csv('Visit_Clean.csv')
```

[30] ✓ 0.1s

Figure 3.2.2.24 Extract the clean data to csv file.

3.2.3 Load

After the data has been cleaned, we load our data to PostgreSQL. We have created a database and table in pgAdmin, by using the code below we directly import the data from Jupyter Notebook into our PostgreSQL :

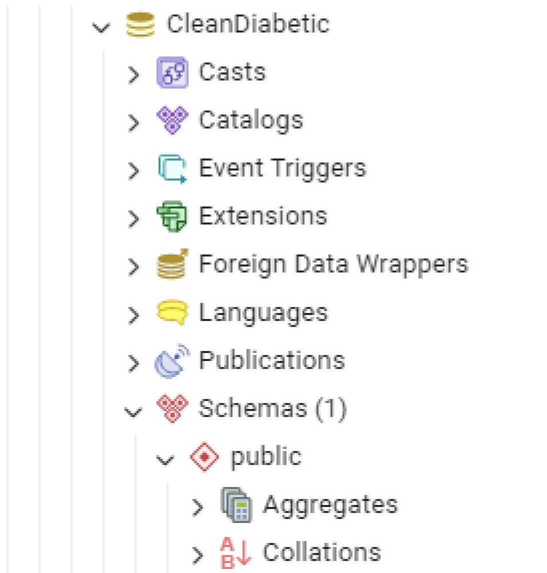
```
import psycopg2
import numpy as np
import pandas as pd
import psycopg2.extras as extras

def execute_values(conn, df, table):
    df = df.astype(object).where(pd.notnull(df), None)
    tuples = [tuple(x) for x in df.to_numpy()]
    cols = ','.join(list(df.columns))
    query = "INSERT INTO %s(%s) VALUES %%s" % (table, cols)
    cursor = conn.cursor()
    try:
        extras.execute_values(cursor, query, tuples)
        conn.commit()
        print("The dataframe for table '%s' was successfully inserted" % table)
    except (Exception, psycopg2.DatabaseError) as error:
        print("Error: %s" % error)
        conn.rollback()
    finally:
        cursor.close()

try:
    conn = psycopg2.connect(database="CleanDiabetic",
                            user="postgres",password= "1234",host="localhost",
                            port="5432")

    execute_values(conn, newamission, 'amission')
    execute_values(conn, newdiagnoses, 'diagnoses')
    execute_values(conn, newpatient, 'patient')
    execute_values(conn, newtest, 'test')
    execute_values(conn, newvisit, 'visit')

except (Exception, psycopg2.DatabaseError) as error:
    print("Error while connecting to the database: %s" % error)

finally:
    if conn:
        conn.close()
        print("Database connection closed.")
```

[38] ✓ 9.2s

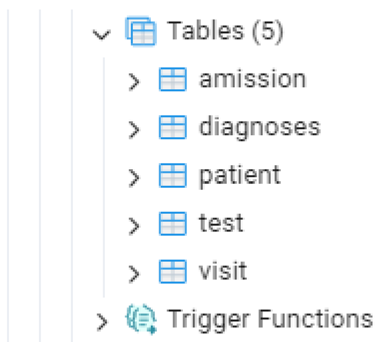
```
The dataframe for table 'amission' was successfully inserted
The dataframe for table 'diagnoses' was successfully inserted
The dataframe for table 'patient' was successfully inserted
The dataframe for table 'test' was successfully inserted
The dataframe for table 'visit' was successfully inserted
Database connection closed.
```

This figure shows the data is being loaded into PostgreSQL.

Create a new database for loading the data from Jupyter to PgAdmin



Create tables for loading the data from Jupyter to PgAdmin with coding part



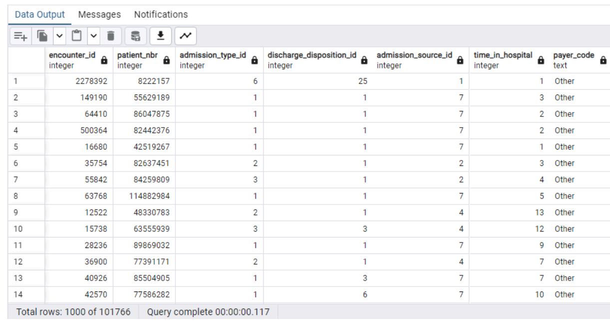
```
Create table Amission
(
	encounter_id int,
	patient_nbr int,
	admission_type_id int,
	discharge_disposition_id int,
	admission_source_id int,
	time_in_hospital int,
	payer_code text
);

Create table Diagnoses
(
	encounter_id int,
	patient_nbr int,
	diag_1 text,
	diag_2 text,
	diag_3 text,
	number_diagnoses int,
	medical_specialty text
);

Create table Patient
(
	encounter_id int,
	patient_nbr int,
	race text,
	gender text,
	age text
);

Create table Visit
(
	encounter_id int,
	patient_nbr int,
	number_outpatient int,
	number_emergency int,
	number_inpatient int
);
```

```
Create table Test
(
	encounter_id int,
	patient_nbr int,
	metformin text,
	repaglinide text,
	nateglinide text,
	chlorpropamide text,
	glimepiride text,
	acetohexamide text,
	glipizide text,
	glyburide text,
	tolbutamide text,
	pioglitazone text,
	rosiglitazone text,
	acarbose text,
	miglitol text,
	troglitazone text,
	tolazamide text,
	examide text,
	citoglipton text,
	insulin text,
	glyburideMetformin text,
	glipizideMetformin text,
	glimepiridePioglitazone text,
	metforminRosiglitazone text,
	metforminPioglitazone text,
	change text,
	diabetesMed text
);
```

Prove data is already loading to PgAdmin

QUERY	OUTPUT

```
SELECT * FROM amission
```

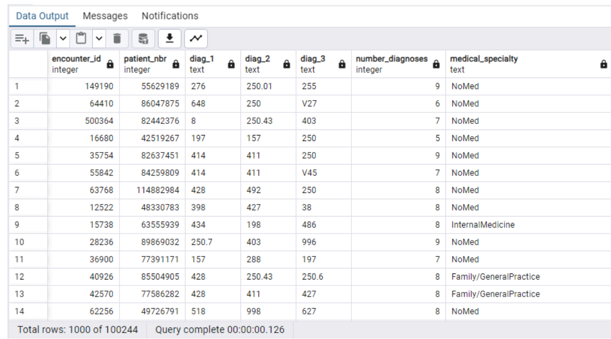


Data Output Messages Notifications

	encounter_id integer	patient_nbr integer	admission_type_id integer	discharge_disposition_id integer	admission_source_id integer	time_in_hospital integer	payer_code text
1	2278392	8222157	6	25	1	1	Other
2	149190	55629189	1	1	7	3	Other
3	64410	86047875	1	1	7	2	Other
4	500364	82442376	1	1	7	2	Other
5	16680	42519267	1	1	7	1	Other
6	35754	82637451	2	1	2	3	Other
7	55842	84259809	3	1	2	4	Other
8	63768	114882984	1	1	7	5	Other
9	12522	48330783	2	1	4	13	Other
10	15738	63555939	3	3	4	12	Other
11	28236	89869032	1	1	7	9	Other
12	36900	77391171	2	1	4	7	Other
13	40926	85504905	1	3	7	7	Other
14	42570	77586282	1	6	7	10	Other

Total rows: 1000 of 101766 Query complete 00:00:00.117

```
SELECT * FROM diagnoses
```

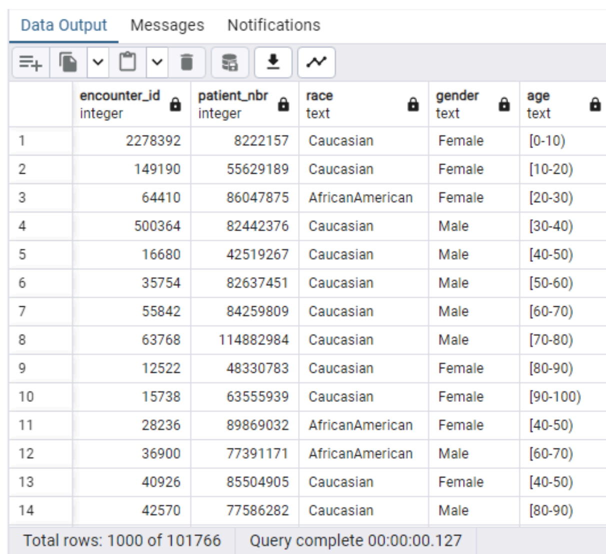


Data Output Messages Notifications

	encounter_id integer	patient_nbr integer	diag_1 text	diag_2 text	diag_3 text	number_diagnoses integer	medical_specialty text
1	149190	55629189	276	250.01	255	9	NoMed
2	64410	86047875	648	250	V27	6	NoMed
3	500364	82442376	8	250.43	403	7	NoMed
4	16680	42519267	197	157	250	5	NoMed
5	35754	82637451	414	411	250	9	NoMed
6	55842	84259809	414	411	V45	7	NoMed
7	63768	114882984	428	492	250	8	NoMed
8	12522	48330783	398	427	38	8	NoMed
9	15738	63555939	434	198	486	8	InternalMedicine
10	28236	89869032	250.7	403	996	9	NoMed
11	36900	77391171	157	288	197	7	NoMed
12	40926	85504905	428	250.43	250.6	8	Family/GeneralPractice
13	42570	77586282	428	411	427	8	Family/GeneralPractice
14	62256	49726791	518	998	627	8	NoMed

Total rows: 1000 of 100244 Query complete 00:00:00.126

```
SELECT * FROM patient
```

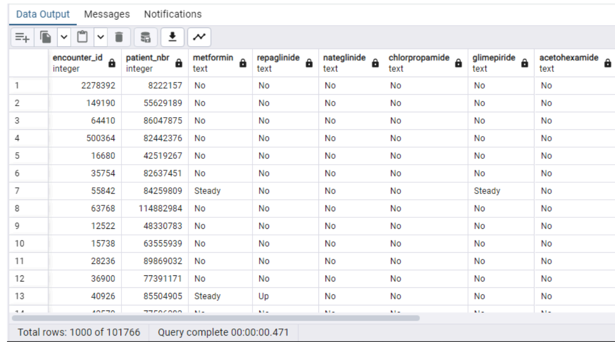


Data Output Messages Notifications

	encounter_id integer	patient_nbr integer	race text	gender text	age text
1	2278392	8222157	Caucasian	Female	[0-10)
2	149190	55629189	Caucasian	Female	[10-20)
3	64410	86047875	AfricanAmerican	Female	[20-30)
4	500364	82442376	Caucasian	Male	[30-40)
5	16680	42519267	Caucasian	Male	[40-50)
6	35754	82637451	Caucasian	Male	[50-60)
7	55842	84259809	Caucasian	Male	[60-70)
8	63768	114882984	Caucasian	Male	[70-80)
9	12522	48330783	Caucasian	Female	[80-90)
10	15738	63555939	Caucasian	Female	[90-100)
11	28236	89869032	AfricanAmerican	Female	[40-50)
12	36900	77391171	AfricanAmerican	Male	[60-70)
13	40926	85504905	Caucasian	Female	[40-50)
14	42570	77586282	Caucasian	Male	[80-90)

Total rows: 1000 of 101766 Query complete 00:00:00.127

```
SELECT * FROM test
```

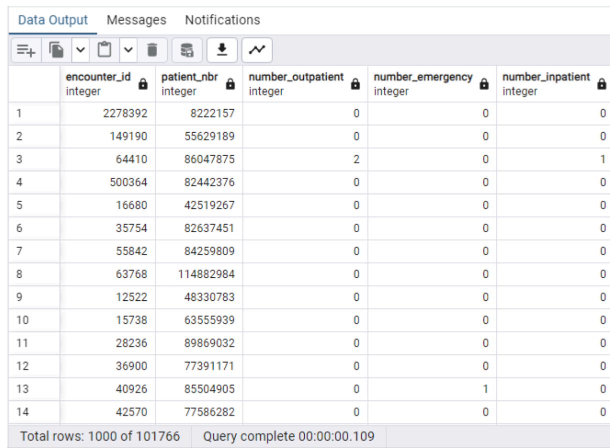


Data Output | Messages | Notifications

	encounter_id integer	patient_nbr integer	metformin text	repaglinide text	nateglinide text	chlorpropamide text	glimepiride text	acetohexamide text
1	2278392	8222157	No	No	No	No	No	No
2	149190	55629189	No	No	No	No	No	No
3	64410	86047875	No	No	No	No	No	No
4	500364	82442376	No	No	No	No	No	No
5	16680	42519267	No	No	No	No	No	No
6	35754	82637451	No	No	No	No	No	No
7	55842	84259809	Steady	No	No	No	Steady	No
8	63768	114882984	No	No	No	No	No	No
9	12522	48330783	No	No	No	No	No	No
10	15738	63555939	No	No	No	No	No	No
11	28236	89869032	No	No	No	No	No	No
12	36900	77391171	No	No	No	No	No	No
13	40926	85504905	Steady	Up	No	No	No	No

Total rows: 1000 of 101766 | Query complete 00:00:00.471

```
SELECT * FROM visit
```



Data Output | Messages | Notifications

	encounter_id integer	patient_nbr integer	number_outpatient integer	number_emergency integer	number_inpatient integer
1	2278392	8222157	0	0	0
2	149190	55629189	0	0	0
3	64410	86047875	2	0	1
4	500364	82442376	0	0	0
5	16680	42519267	0	0	0
6	35754	82637451	0	0	0
7	55842	84259809	0	0	0
8	63768	114882984	0	0	0
9	12522	48330783	0	0	0
10	15738	63555939	0	0	0
11	28236	89869032	0	0	0
12	36900	77391171	0	0	0
13	40926	85504905	0	1	0
14	42570	77586282	0	0	0

Total rows: 1000 of 101766 | Query complete 00:00:00.109

4.0 DATABASE

4.1 Relational Model

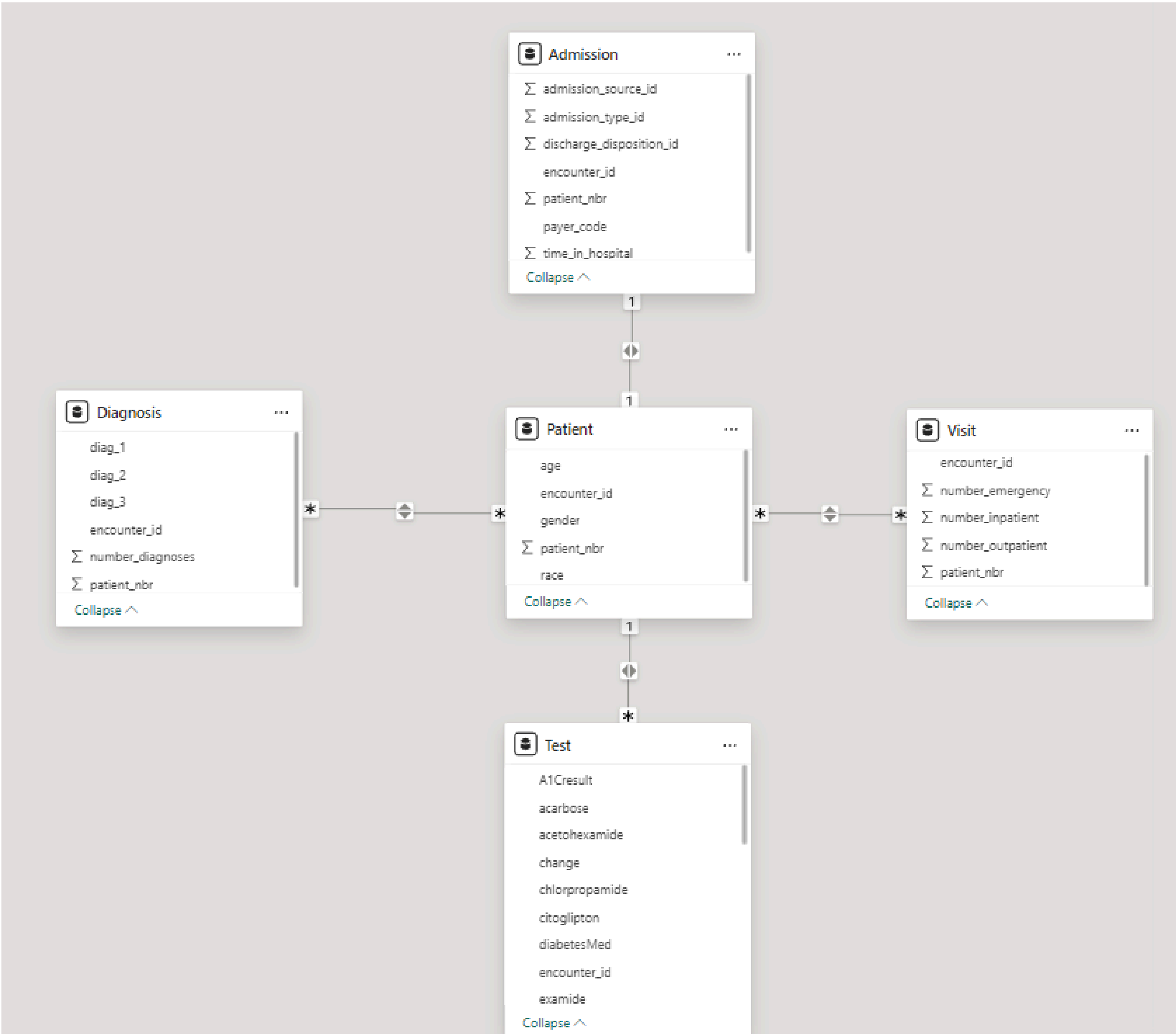


Figure 4.1 Relational Model using Power BI

Figure 4.1 shows relational models using Power BI. This view not only allows for visualization of the relationships between data but also displays the primary keys and foreign keys.

Relationship between data

Data	Relationship
Amission_Clean->Patient_Clean	One-to-One
Test_Clean->Patient_Clean	Many-to-One
Diagnose_Clean->Patient_Clean	Many-to-many
Visit_Clean->Patient_Clean	Many-to-many
Patient_Clean->Amission_Clean	One-to-many
Patient_Clean->Test_Clean	One-to-many
Patient_Clean->Diagnose_Clean	One-to-many
Patient_Clean->Visit_Clean	One-to-many

4.2 Identification of Data Warehouse Schema

According to Figure 4.1, the data warehouse schema for these datasets is Star Schema because they contain only one fact table, which is Patient table. This table links to four dimension tables: Admission, Test, Diagnose, and Visit. Based on this schema, we will achieve fast query response time because of the reduced join complexity and ease of understanding the data.

5.0 RESULT AND DATA ANALYSIS

5.1 OLAP Coding

Olap is the short form of the Online Analytical Processing serve that allows us to analyze information from multiple database system at the same time. Basically olap consist of five operation which are cube, rollup, drilldown, slicing, dicing and we also did OLAP operation using the PostgreSQL software

Cube

Query Query History

```
-- OLAP Cube Operation
SELECT admission_source_id, discharge_disposition_id, payer_code, SUM(time_in_hospital) AS total_time_in_hospital
FROM admission
GROUP BY CUBE (admission_source_id, discharge_disposition_id, payer_code);
```

Output

	admission_source_id real	discharge_disposition_id real	payer_code text	total_time_in_hospital real
1	[null]	[null]	[null]	894724
2	1	6	Other	15376
3	1	22	CP	152
4	1	7	MD	42
5	7	27	UN	8
6	17	16	SP	4
7	7	14	CH	4
8	4	14	OT	14
9	4	11	Other	376
10	1	6	BC	2018
11	7	7	OG	158
12	5	5	CM	26
13	4	1	OG	86
14	1	8	Other	368
15	17	2	MD	2
16	4	5	MC	72
17	7	14	UN	60

Interpretation

The {total_time_in_hospital}, {admission_source_id}, {discharge_disposition_id}, and {payer_code} fields are all filled in. All of the records show that 894,724 units of time were spent in the hospital. This is likely measured in hours or days. For Admission Source ID 1, there is more than one record with a different `discharge_disposition_id` and `payer_code`. This is clear when we sort the data by `admission_source_id}. There are 15,376 units of hospital time for people who came in from source 1 and were sent home under {discharge_disposition_id} 6. Also, Admission Source ID 7 appears a lot, for example in a record with 27 units for {discharge_disposition_id} 27 and payment code {UN}.

When we sort the data by `payer_code} and different mixes of `admission_source_id` and `discharge_disposition_id}, we see that `Other` often comes up a lot of times. In this case, the payment code "Other" with "admission_source_id" 1 and "discharge_disposition_id" 6 finds 15,376 units. The fact that the payment number {MD} shows up more than once says that a lot of people use this kind of provider.

That being said, release ID 22 has 152 units for entry source ID 1 and payment code {CP}. It shows that entry source ID 17 and payment code {SP} added up to 4 units in release decision ID 16. In the first row, there are no numbers for `admission_source_id`, `discharge_disposition_id`, or `payer_code`. This row has a very high total hospital time of 894,724 units too. This could mean a grouping or a simple summary row.

It's clear from this study that most of the time spent in the hospital is tied to certain combinations of how someone got there, when they were released, and how they were paid. Many times, pay codes like "MD" and "Other" are used. There are also entry sources that are shown more often, like 1 and 7. This could mean that these are the main ways that people get into the hospital.

.

ROLL UP

Query | Query History

```
select*from diagnoses

SELECT
    COALESCE(medical_specialty, 'All Specialties') AS medical_specialty,
    COUNT(DISTINCT encounter_id) AS num_encounters,
    COUNT(DISTINCT patient_nbr) AS num_patients,
    COUNT(DISTINCT diag_1) AS unique_diag_1,
    COUNT(DISTINCT diag_2) AS unique_diag_2,
    COUNT(DISTINCT diag_3) AS unique_diag_3,
    SUM(number_diagnoses) AS total_diagnoses
FROM diagnoses
GROUP BY ROLLUP(medical_specialty);
```

Output

Data Output | Messages | Notifications

	medical_specialty text	num_encounters bigint	num_patients bigint	unique_diag_1 bigint	unique_diag_2 bigint	unique_diag_3 bigint	total_diagnoses bigint
1	AllergyandImmunology	7	7	5	7	6	41
2	Anesthesiology	11	11	10	10	10	68
3	Anesthesiology-Pediatric	12	11	1	6	9	45
4	Cardiology	5321	4783	206	230	269	37853
5	Cardiology-Pediatric	5	5	2	2	3	25
6	DCPTEAM	6	6	6	6	6	59
7	Dentistry	4	4	3	4	3	31
8	Dermatology	1	1	1	1	1	9
9	Emergency/Trauma	7496	5078	361	390	403	59496
10	Endocrinology	116	108	58	59	58	731
11	Endocrinology-Metabolism	8	7	7	7	7	60
12	Family/GeneralPractice	7299	5939	387	379	390	52774
13	Gastroenterology	560	507	141	143	149	4263
14	Gynecology	53	52	21	29	22	374

Interpret:

The roll-up operation that has been accomplished on the diagnoses table has achieved a comprehensive tabular presentation of medical data rolling, functional by specialty and summary. In the Cardiology-Pediatric specialty, 5, one patient encounter was noted and 5 patients in total, with 2 diag_1, 2 diag_2, and 3 diag_3, the results in 25 diagnosis notations. Emergency/Trauma had the highest number of unique encounters among patients with 7496 and a total of 59496 diagnoses; 361 diagnoses in diag_1, 390 in diag_2 and 403 in diag_3. Dermatology has had 1 total encounters that was for 1 patients, Diag_1 had 1 unique diagnosis, diag_2 also had 1 unique diagnosis and diag_3 also had 1 unique diagnosed, all in total that resulted to 9 diagnosis. The sum total of roll-up for all specialties showed 100244 encounters across all patient specialties where 715 unique diagnoses for diag_1, 743 for diag_2, and 789 for diag_3 were made reaching an approximate total of 751530 diagnoses.

From this process, it illustrates how many patients have been distributed and how many patients are diagnosed in a particular area or field of medicine. It provides a broad view that helps in deciding where to best invest in the healthcare system, to identify trends in medical science, and for effective management decisions. The summary is quite comprehensive but when one looks at the specialty of everyday cases and patient encounters, they see much more and even some intriguing things.

SLICING

```
SELECT
    p.encounter_id,
    p.patient_nbr,
    p.race,
    p.gender,
    p.age,
    a.time_in_hospital
FROM
    patient p
JOIN
    amission a
ON
    p.encounter_id = a.encounter_id
    AND p.patient_nbr = a.patient_nbr
WHERE
    p.age = '[0-10)' OR p.age = '[10-20)';
```

Output:

	encounter_id integer	patient_nbr integer	race text	gender text	age text	time_in_hospital integer
1	2278392	8222157	Caucasian	Female	[0-10)	1
2	149190	55629189	Caucasian	Female	[10-20)	3
3	715086	3376278	Caucasian	Male	[10-20)	1
4	2671290	3492477	AfricanAmerican	Male	[10-20)	10
5	2735964	2359485	Caucasian	Female	[0-10)	3
6	2817642	107102214	Caucasian	Male	[10-20)	2
7	2913624	5073354	AfricanAmerican	Female	[10-20)	1
8	2968386	8568180	Caucasian	Female	[0-10)	2
9	3039162	539910	Caucasian	Female	[10-20)	2
10	3048198	3454722	Other	Male	[10-20)	4
11	3108096	5832918	Caucasian	Female	[0-10)	1
12	3557748	1582326	AfricanAmerican	Female	[10-20)	3
13	3692532	3201255	AfricanAmerican	Female	[10-20)	6
14	3863238	109797327	AfricanAmerican	Female	[10-20)	1
15	3974694	40158	Caucasian	Female	[10-20)	1
16	3983004	45144	Caucasian	Male	[10-20)	13
17	4065138	9029196	Caucasian	Male	[10-20)	3
18	4086876	2892654	Caucasian	Male	[10-20)	1
19	4139334	1791090	Caucasian	Female	[10-20)	2
20	4140282	7443135	Caucasian	Female	[0-10)	2

Interpretation:

This query to analyse data for pediatric patients. The query joins patient and amission tables on ‘encounter_id’ and ‘patient_nbr’ that include only patients whose age is between [0-10) or [10-20) includes both male and female patients. The majority of patients belong to Caucassion race followed AfricanAmerican and others too. The length of hospital stay varies from 1 day to 13 days. Female patients seem to be more frequent in this dataset compared to male patients. The age group [10-20) has both the highest individual stay which is 13 days and compared to the age group [0-10). Hence, the age group [0-10) is at higher risk of longer hospital stays.

Dicing

```sql
SELECT P.patient_nbr, P.gender, P.age, A.time_in_hospital
FROM Patient P
JOIN Amission A
    ON P.encounter_id = A.encounter_id
JOIN Test T
    ON P.encounter_id = T.encounter_id
WHERE T.insulin = 'Up'
  AND (P.gender = 'Male' OR P.gender = 'Female')
  AND (P.age IN ('[0-10)'))
order by A.time_in_hospital desc;
```

Output:

	patient_nbr integer	gender text	age text	time_in_hospital integer
1	82666917	Male	[0-10)	6
2	30383676	Female	[0-10)	5
3	28863864	Female	[0-10)	5
4	25574562	Male	[0-10)	4
5	76430070	Male	[0-10)	4
6	84369024	Female	[0-10)	4
7	64230939	Female	[0-10)	3
8	108410454	Male	[0-10)	3
9	5274396	Female	[0-10)	3
10	22911381	Female	[0-10)	3
11	17327511	Male	[0-10)	3
12	3685653	Female	[0-10)	3
13	42247008	Male	[0-10)	2
14	1068030	Male	[0-10)	2
15	1647108	Female	[0-10)	2
16	42417819	Female	[0-10)	2
17	10001988	Male	[0-10)	2
18	1735551	Male	[0-10)	1

Total rows: 19 of 19 Query complete 00:00:00.116

Intepretation

The dicing operation aims to subset the data into a small cube by selecting two dimensions which are patient age group 0 to 10 years old then the insulin test result is an increase or up. We want to observe the gender and the time they stay at the hospital. This result is important to the hospital take note of the pediatric patient who has received insulin and was admitted to the hospital.

The results show in total we have 19 pediatric patients who need to be admitted due to insulin and the longest is 6 days. We can consider this data to improve hospital facilities, research and development specifically catered to the pediatric patient with insulin related condition. This to resucin the need for prolonged hospital stays exceeding one week. We also observe that the gender distribution is slightly balanced. The hospital needs to take note of they patient distribution to improve their management in control the kids patients.

Drilldown:

```sql
SELECT encounter_id, patient_nbr, admission_type_id, discharge_disposition_id, admission_source_id, time_in_hospital
FROM amission
WHERE payer_code = 'Other';
```

Code:

SELECT encounter_id, patient_nbr, admission_type_id, discharge_disposition_id, admission_source_id, time_ in_hospital
FROM amission
WHERE payer_code = 'Other';

Output:

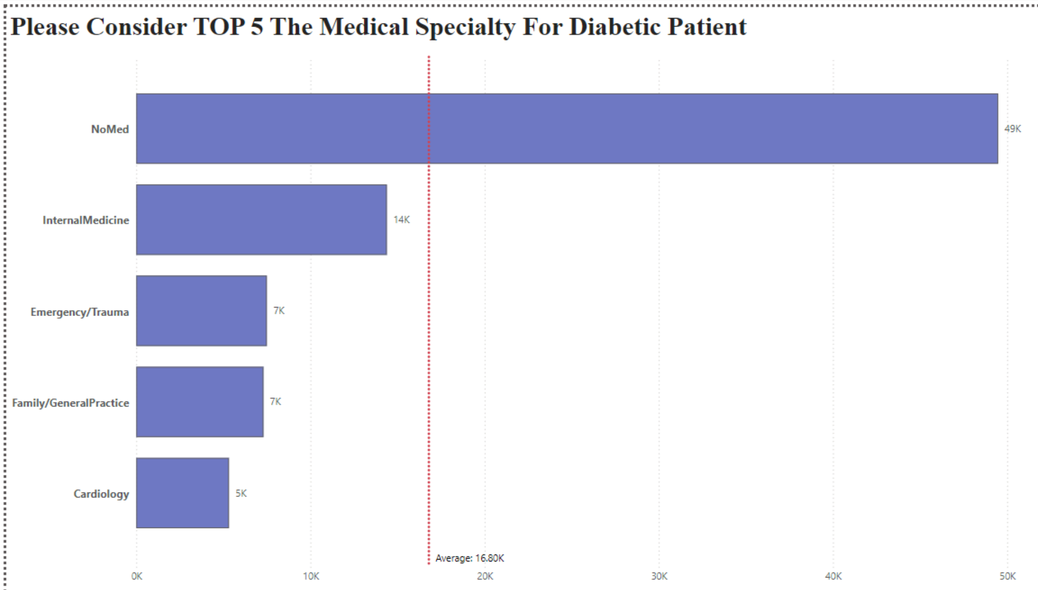
	encounter_id integer	patient_nbr integer	admission_type_id integer	discharge_disposition_id integer	admission_source_id integer	time_in_hospital integer
1	2278392	8222157	6	25	1	1
2	149190	55629189	1	1	7	3
3	64410	86047875	1	1	7	2
4	500364	82442376	1	1	7	2
5	16680	42519267	1	1	7	1
6	35754	82637451	2	1	2	3
7	55842	84259809	3	1	2	4
8	63768	114882984	1	1	7	5
9	12522	48330783	2	1	4	13
10	15738	63555939	3	3	4	12
11	28236	89869032	1	1	7	9
12	36900	77391171	2	1	4	7
13	40926	85504905	1	3	7	7
14	42570	77586282	1	6	7	10
15	62256	49726791	3	1	2	1
16	73578	86328819	1	3	7	12
17	77076	92519352	1	1	7	4
18	84222	108662661	1	1	7	3
19	89682	107389323	1	1	7	5
20	148530	69422211	3	6	2	6

Total rows: 3000 of 40256 Query complete 00:00:00.084

Interpretation:

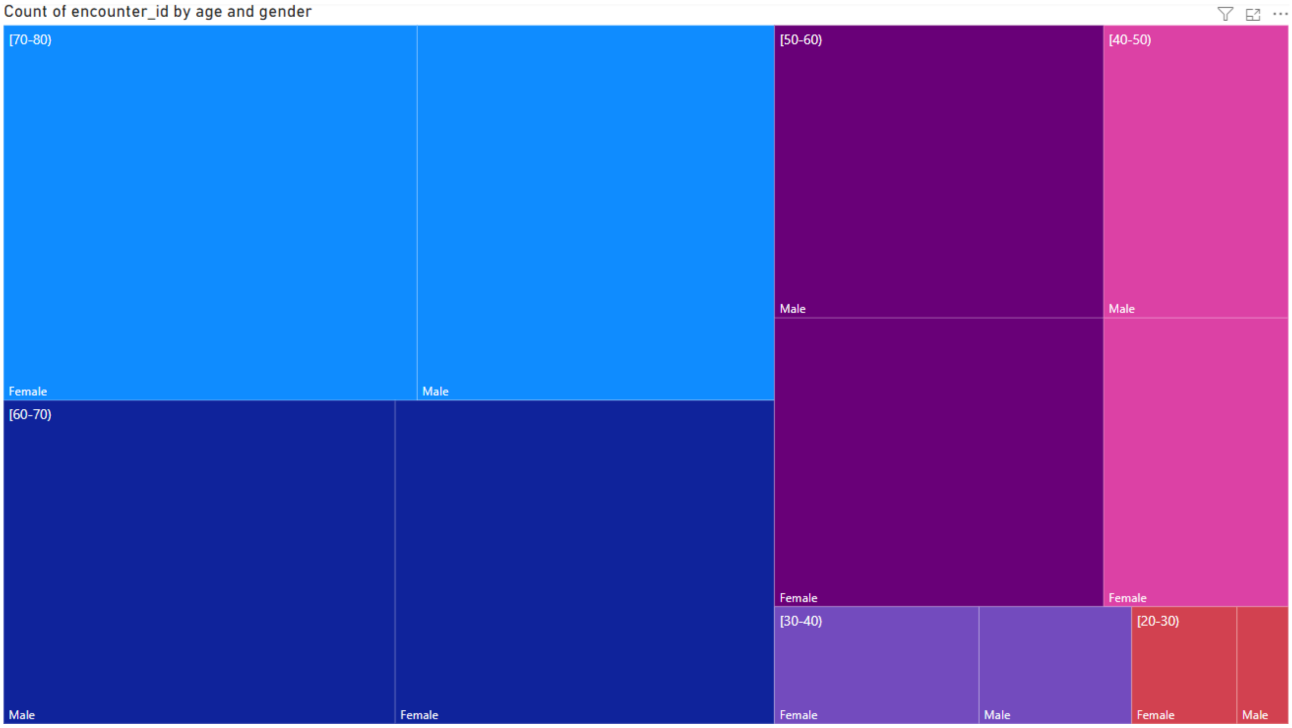
Amission table data is depicted in the picture, giving hospital admissions a deeper look through drill-down analysis. First, we categorize admissions by the admission_type_id with commonly occurring ones being emergency, urgent and elective. Taking this further, we are going to consider discharge outcomes (discharge_disposition_id) and average hospital stays for each admission type as for example discharge to home or transfer to another facility. In addition, it reveals the sources of admissions (admission_source_id) like referrals and emergency room entries. Finally via patient demographics patient characteristics such as race, gender, age and length of stay can be accessed. This exhaustive drill down helps find patterns and peculiarities in hospitalizations that support operational activities and clinical care management.

5.2 Data Visualisation



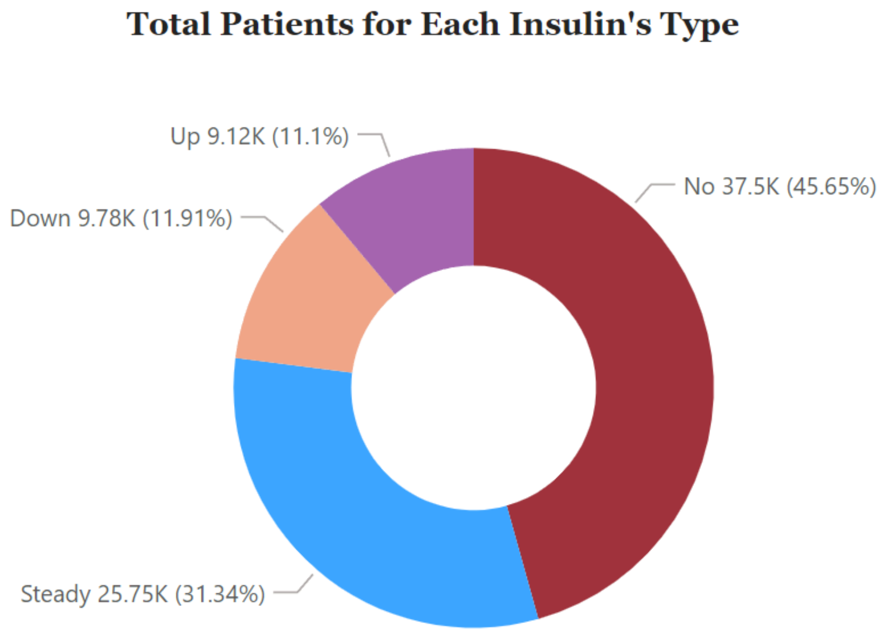
The bar chart shows the Top 5 Medical Specialty For Diabetic Patients in the No medical category are the highest with 49k patients and the second highest from the Internal Medicine category with 14K. It indicates that a large proportion of the diabetic patient may not have any medical specialty. Since the data was collected for 10 years many patients might still lack awareness or access to specialized care for diabetes in the early year of data collection.

Third with 7k patients we have Emergency/trauma and family/general practice. This indicates that diabetic patients require emergency or trauma care maybe due to anxiety with the acute complications of diabetes. Cardiology is also the top 5 Medical Specialties with 5k patients. Diabetic patients also experience cardiovascular conditions therefore some of them need the Cardiology specialty. The red dotted line represents the average number of diabetic patients across these specialties, which is 16.5K. The average line indicates that most specialties, except for "NoMed," handle fewer patients than the overall average.

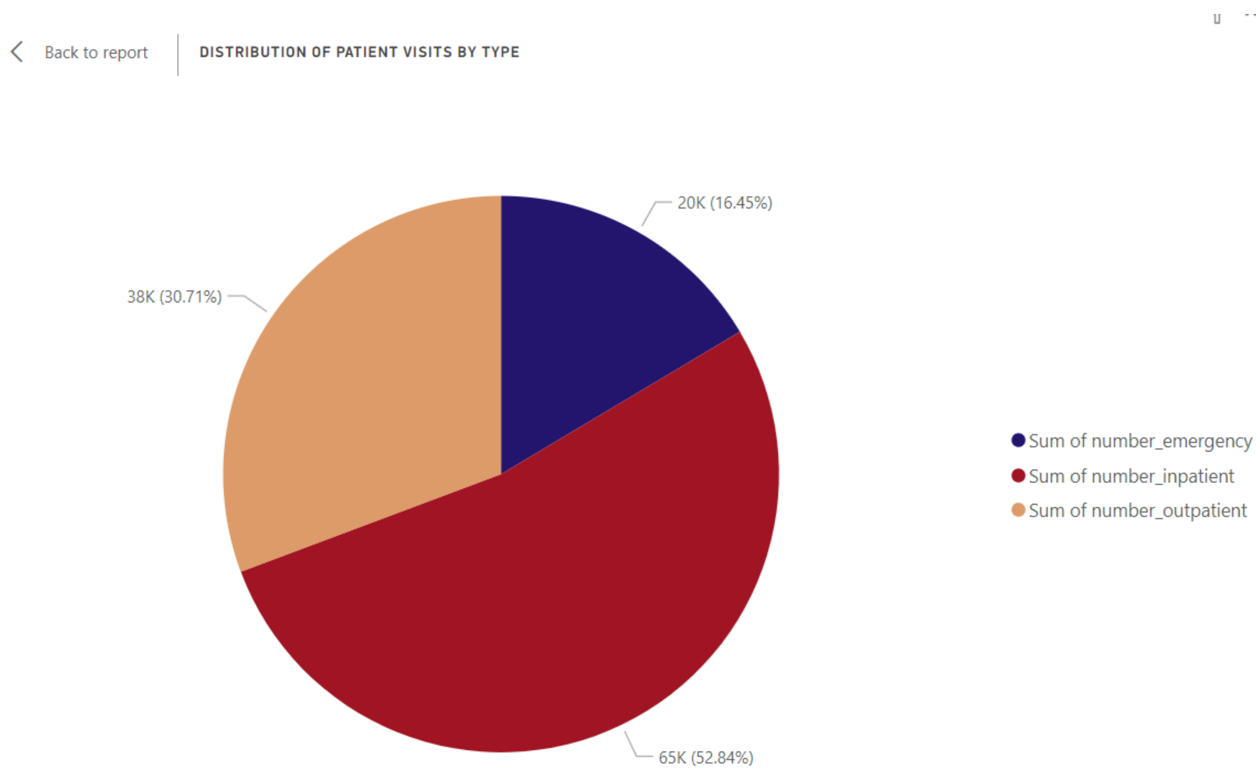


This treemap display the number of diabetic patient encounters according to age and gender. Subsequently, the age ranges are 20-30, 30-40, 40-50, 50-60, 60-70, and 70-80, and each age bracket contains both male and female participants. The size of each rectangle shows the count of encounters, such that rectangles larger than others demonstrate greater encounter counts. The above categories based on the age and gender are color coded to easily distinguish each of them.

From the description of the graph, it is clear that the highest interaction is experienced among the females aged from 70-80 years while male counterparts are the second highest. Other big categories of patients are males who are 60-70 years, female patients of the same age. It can be observed that as age groups become younger, the count of encounters reduces with the least number of encounters falling within the 20-30 & 30-40 age brackets. Both males and females are observed in 50-60 and 40-50 age range groups; however, females appear to obtain slightly higher encounter counts. This treemap would prove helpful to analyze the demographic factor of the diabetic patient encounters highlighting that the demographic mainly affects the elderly females.



The visualization is a donut chart showing the total patients for each insulin's type. For No Insulin (45.65%) is the largest segment of the chart that represents 37.5k patients. These patients are not on any insulin treatment.This might indicate that they manage diabetes through diet,medications and exercise. Next, we go to the Steady(31.34%) which represent 25.75k patients. This type of insulin are taken by patients are on a steady insulin dosage. For Down (11.91%) which represent 9.78k patients. These patients had their insulin dosage decreased. Lastly, for Up (11.1%) is the smallest segment that represents 9.12k patients. All these patients had their insulin dosage increased. A smaller portion of the patients has had changes in their insulin dosages. Specifically, 11.91% have had their insulin dosage decreased and 11.1% have had it increased. This might reflect dynamic management of diabetes where patients' conditions necessitate changes in treatment.



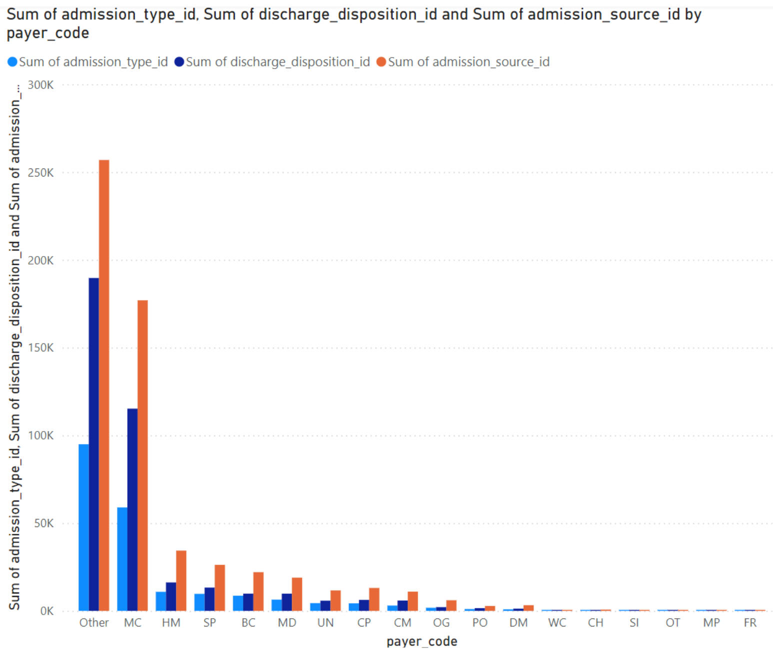
With 52.84% of the visits, inpatients accounted for the highest patient visits followed by outpatients with 30.71% and lastly emergency with 16.45%.

That is perhaps why more patients visit hospitals for inpatient care than any other reason, This is because health care providers often concentrate on areas they are good at; It becomes justifiable then that 52.84% of the visits were, for inpatients when a hospital specializes on such field like: surgery or therapy or chronic disease management. The specialization into different areas of medical care translates into figures for admissions.

Secondly, such requirements are referred to as Population Health; this happens when a community has individuals with long term sick persons. As a result, inpatient care (52.84%) does take the larger proportion of people who visit hospitals for admission as patients (pie chart).

Similarly, outpatient and emergency departments signify thirty point seven one percent (30.71%) and sixteen point four five percent (16.45%).

Thirdly, out-patient care: with restricted access to out-patient services being a possible cause of increased inpatient numbers. Insufficient availability or staffing of the out-patient facilities may result in admission as an inpatient for serious diseases, contributing to 52.84% of all patients who use them and are admitted.



Data on distribution of total numbers of admission type IDs, discharge disposition IDs and admission source IDs by different payer codes are presented in the bar chart that is grouped together. The axis y represents the total number of each ID while the x axis shows different payer codes. Clearly, the highest totals for all three categories occur with 'Other' and 'MC' (Medicare) payer codes as evident from this graph. This points to their significant contribution in terms of hospital admissions, discharges and sources as indicated by a chart. Notably, 'Other' category has very high values especially considering that the discharge disposition ID total is highest followed by admission source ID and then admission type ID. Medicare also has high totals but they are slightly less compared to 'Other'. On the other hand, there are some payer code which has very low frequency in overall data set and in this criterion, 'HM' (Health Maintenance), 'SP' (Self-pay), BC (Blue Cross) and MD (Medicaid) are prominent where ID of discharge disposition is higher among the three groups . Payer codes like "PO" for Private Insurers, "DM" for Department of Military, “WC” for Workers compensation, and others; their ‘even less total’ manifested generally fewer hospital activities associated with these payers.

6.0 CONCLUSION

In conclusion, this project has provided an opportunity to analyze the data regarding diabetic atients in the US hospitals more effectively to address the need of such patients within the non-ICU settings. In exploring aspects of the patterns &/or lack of consistency employed within managing diabetes using data analysis or proposing evidence based guidelines with applying to ICU inpatient and reviewing how better measurement of HbA1c can enhance the quality of care given to diabetic patients and reduce follow-up hospitalisation, the project has revealed further possible other ways of improving diabetic care in other non-ICU hospitalised patients setting .

These objectives have been achieved with features like the project architecture, ETL pipeline, database, and data analysis. The concept of Kimball's technique to build data marts ad hoc, made it possible for faster delivery of outcomes and ETL for cleaning, transforming, and loading the data part properly. The relational model along with the data warehouse schema gave necessary and sufficient understanding of the linkage between the tables in the dataset; OLAP coding and data visualization was insightful.

This project's recommendations emphasize the significance of data analytics in enhancing diabetes care in other non-ICU hospital contexts. In order to come up with best practices on diabetes, and its management Disparities that were revealed from the data was used. The study on the assessment of HbA1c revealed the fact that it has the potential to direct enhancement of the quality of diabetes care as well as reduced readmission. These results speak volumes when it comes to the patient management of diabetes in non-ICU hospital environments with the overall management of diabetes being enhanced. In general, this project has offered the right guideline and implication for enhanced diabetes care in non-IC.

6.1 Limitation

While completing the diabetic person on hospital readmission rate project we are involved in several barriers that must be considered and improved in the next research. First, the main barriers are time limitations that we cannot explore more on data collecting, data cleaning, and analysis. With additional time we will explore the data more deeply and give valuable insights into each factor that influences diabetic person the diabetic hospital readmission rate

The limited time made us have a barrier with data quality and completeness, which our dataset has dirty data. There is some confusion since the data mostly uses numeric that indicate the categorical data such as admission type, discharge disposition ID, admission source, and more attributes. The dataset also has missing values and messy data that we need to do for a longer time in the data cleaning process. It's due to ensure the data quality and not affect the reliability of the finding.

7.0 REFERENCES

Cleveland Clinic. (2023, February 17). *Diabetes*. Cleveland Clinic; Cleveland Clinic. https://my.clevelandclinic.org/health/diseases/7104-diabetes

Data Warehousing - OLAP - Tutorialspoint. (n.d.). Www.tutorialspoint.com. https://www.tutorialspoint.com/dwh/dwh_olap.htm

Ramachandran, A. (2024, February 13). *OLAP Operations*. Medium. https://medium.com/@beeindian04/olap-operations-773294c1d3e4

Scardina, J. (2022). *What is Microsoft Power BI? - Definition from WhatIs.com*. SearchContentManagement. https://www.techtarget.com/searchcontentmanagement/definition/Microsoft-Power-BI

Tyagi, A. (2023, December 8). *The Ultimate Guide to Power BI Visualizations*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2023/12/the-ultimate-guide-to-power-bi-visualizations/

United Nations. (2024). *The 17 Sustainable Development Goals*. United Nations. https://sdgs.un.org/goals

Kaggle: https://www.kaggle.com/datasets/brandao/diabetes/data?select=diabetic_data.csv

8.0 APPENDIX

1. **Data description**

   Age : [0-10),[10-20),[20-30), [30-40),[40-50),[50-60),[60-70),[70-80),[80-90), [90-100)

   Race: Caucasian, African American, Other, Asian"

   Admission_type_id:

1	Emergency
2	Urgent
3	Elective
4	Newborn
5	Not Available
6	NULL
7	Trauma Center
8	Not Mapped

Admission_source_id:

1	Physician Referral
2	Clinic Referral
3	HMO Referral
4	Transfer from a hospital
5	Transfer from a Skilled Nursing Facility (SNF)
6	Transfer from another health care facility
7	Emergency Room
8	Court/Law Enforcement
9	Not Available
10	Transfer from critial access hospital
11	Normal Delivery
12	Premature Delivery
13	Sick Baby
14	Extramural Birth
15	Not Available
17	NULL
18	Transfer From Another Home Health Agency
19	Readmission to Same Home Health Agency
20	Not Mapped
21	Unknown/Invalid
22	Transfer from hospital inpt/same fac reslt in a sep claim
23	Born inside this hospital
24	Born outside this hospital
25	Transfer from Ambulatory Surgery Center
26	Transfer from Hospice

discharge_disposition_id

1 Discharged to home
2 Discharged/transferred to another short term hospital
3 Discharged/transferred to SNF
4 Discharged/transferred to ICF
5 Discharged/transferred to another type of inpatient care institution
6 Discharged/transferred to home with home health service
7 Left AMA
8 Discharged/transferred to home under care of Home IV provider
9 Admitted as an inpatient to this hospital
10 Neonate discharged to another hospital for neonatal aftercare
11 Expired
12 Still patient or expected to return for outpatient services
13 Hospice / home
14 Hospice / medical facility
15 Discharged/transferred within this institution to Medicare approved swing bed
16 Discharged/transferred/referred another institution for outpatient services
17 Discharged/transferred/referred to this institution for outpatient services
18 NULL
19 Expired at home. Medicaid only, hospice.
20 Expired in a medical facility. Medicaid only, hospice.
21 Expired, place unknown. Medicaid only, hospice.
22 Discharged/transferred to another rehab fac including rehab units of a hospital .
23 Discharged/transferred to a long term care hospital.
24 Discharged/transferred to a nursing facility certified under Medicaid but not certified under Medicare.
25 Not Mapped
26 Unknown/Invalid
30 Discharged/transferred to another Type of Health Care Institution not Defined Elsewhere
27 Discharged/transferred to a federal health care facility.
28 Discharged/transferred/referred to a psychiatric hospital of psychiatric distinct part unit of a hospital
29 Discharged/transferred to a Critical Access Hospital (CAH).

Payer code:

:Other,MC,MD, HM, UN, BC, SP, CP,SI, DM, CM,CH, PO, WC, OT, OG, MP, FR

2. OLAP Operation

Cube:

```
SELECT admission_source_id, payer_code, SUM (time_in_hospital) AS
total_time_in_hospital
FROM admission
GROUP BY CUBE (admission_source_id, discharge_disposition_id,
payer_code);
```

Rollup:

```
select*from diagnoses
SELECT
COALESCE(medical_specialty, 'All Specialties') AS medical_specialty,
    COUNT(DISTINCT encounter_id) AS num_encounters,
    COUNT(DISTINCT patient_nbr) AS num_patients,
    COUNT(DISTINCT diag_1) AS unique_diag_1,
    COUNT(DISTINCT diag_2) AS unique_diag_2,
    COUNT(DISTINCT diag_3) AS unique_diag_3,
    SUM(number_diagnoses) AS total_diagnoses
FROM diagnoses
GROUP BY ROLLUP(medical_specialty);
```

Slicing:

```
SELECT
    p.encounter_id, p.patient_nbr,  p.race, p.gender, p.age,
  a.time_in_hospital
FROM
    patient p
JOIN
    amission a
ON
    p.encounter_id = a.encounter_id
    AND p.patient_nbr = a.patient_nbr
WHERE
    p.age = '[0-10)' OR p.age = '[10-20)';
```

Dicing:

```
SELECT P.patient_nbr, P.gender, P.age, A.time_in_hospital
FROM Patient P
JOIN Amission A
     ON P.encounter_id = A.encounter_id
JOIN Test T
     ON P.encounter_id = T.encounter_id
WHERE T.insulin = 'Up'
  AND (P.gender = 'Male' OR P.gender = 'Female')
  AND (P.age IN ('[0-10)'))
order by A.time_in_hospital desc, P.gender;
```

DrillDown:

```
SELECT encounter_id, patient_nbr, admission_type_id,
discharge_disposition_id, admission_source_id, time_in_hospital
FROM amission
WHERE payer_code = 'Other';
```