Medical Dataset Analysis: Python, SQL, and Insights

Step into the world of data-driven healthcare analysis alongside Sarah, the healthcare data explorer, in our project, "Medical Dataset Analysis: Python, SQL, and Insights." With Python as her trusted tool and SQL as her analytical compass, Sarah embarks on a mission to unlock the potential hidden in interconnected medical datasets.

This project focuses on three critical datasets: "hospitalization_details," "medical_examinations," and "names." Divided into two essential modules, the project commences with meticulous data cleaning, ensuring the data's accuracy and structure. Once the data shines brilliantly, the second module unleashes the power of SQL queries to extract valuable insights.

Our journey is a deep dive into healthcare data, unearthing insights that can revolutionize healthcare decision-making and resource optimization. It's not just data analysis; it's a transformative voyage into the world of medical dataset analysis.

By the end of this project, you won't just crunch numbers; you'll possess the tools to drive data-powered improvements in healthcare, enhancing the lives of patients and streamlining resource allocation.

Join Sarah on this captivating journey, where every line of code and every SQL query unravels the mysteries of medical data. Together, we'll illuminate the path to actionable insights, shaping the future of healthcare with data-driven solutions.

Module 1

Task 1: Loading Hospitalization Details

In this task, we load the hospitalization details from the 'hospitalisation_details.csv' file into a Pandas DataFrame named 'hosp_details.' This step is essential for our new project, "Medical Dataset Analysis: Python, SQL, and Insights," as it forms the foundation for the data analysis and insights that we aim to derive from the medical dataset.

In [1]:

```
#--- Import Pandas ---
import pandas as pd

#--- Read in dataset(hospitalisation_details.csv) ----
hosp_details = pd.read_csv('hospitalisation_details.csv')

#--- Inspect data ---
print(hosp_details.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2432 entries, 0 to 2431
Data columns (total 11 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   c_id                   2432 non-null   object
 1   yr                     2432 non-null   int64 
```

```
 2   mth                    2431 non-null   object
 3   date?                  2432 non-null   int64
 4   children?              2432 non-null   int64
 5   charges?               2432 non-null   float64
 6   host_tier              2432 non-null   object
 7   Ct_tier                2432 non-null   object
 8   st_id                  2432 non-null   object
 9   Has_Children           2432 non-null   object
 10  Is_Frequent_Treatment  2432 non-null   object
dtypes: float64(1), int64(3), object(7)
memory usage: 209.1+ KB
None
```

Task 2: Identifying Null Values in Hospitalization Details

In this task, we identify and count the null values in the 'hosp_details' dataset. This step is crucial for our new project, "Medical Dataset Analysis: Python, SQL, and Insights," as it helps us understand the extent of missing data within the dataset. Recognizing and handling null values is essential for ensuring the accuracy and quality of our data analysis and insights.

In [2]:

```
# --- WRITE YOUR CODE FOR MODULE 1 TASK 2 ---
null_values = hosp_details.isnull().sum()
print(null_values)

#--- Inspect data ---
```

```
c_id                    0
yr                      0
mth                     1
date?                   0
children?               0
charges?                0
host_tier               0
Ct_tier                 0
st_id                   0
Has_Children            0
Is_Frequent_Treatment   0
dtype: int64
```

Task 3: Identifying Data Types in Hospitalization Details

In this task, we determine the data types of the columns in the 'hosp_details' dataset. This step is vital for our data analysis, as it provides insights into how the data is stored and helps us select appropriate methods for further analysis. Understanding the data types is crucial for working with the dataset effectively.

In [3]:

```
# --- WRITE YOUR CODE FOR MODULE 1 TASK 3 ---
```

```
datatype = hosp_details.info()

#--- Inspect data ---
print(datatype)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2432 entries, 0 to 2431
Data columns (total 11 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   c_id                   2432 non-null   object 
 1   yr                     2432 non-null   int64  
 2   mth                    2431 non-null   object 
 3   date?                  2432 non-null   int64  
 4   children?              2432 non-null   int64  
 5   charges?               2432 non-null   float64
 6   host_tier              2432 non-null   object 
 7   Ct_tier                2432 non-null   object 
 8   st_id                  2432 non-null   object 
 9   Has_Children           2432 non-null   object 
 10  Is_Frequent_Treatment  2432 non-null   object 
dtypes: float64(1), int64(3), object(7)
memory usage: 209.1+ KB
None
```

Task 4: Identifying Duplicate Data in Hospitalization Details

In this task, we aim to identify and quantify the presence of duplicate data within the 'hosp_details' dataset. The count of duplicates (referred to as 'duplicates' in the code) is an important metric. It helps us understand the extent of redundancy in the dataset, which is crucial for data quality and accuracy in our analysis. By recognizing and handling duplicate records, we ensure that our insights and conclusions are based on unique, meaningful data, preventing any potential distortions caused by repeated entries.

In [4]:

```
# --- WRITE YOUR CODE FOR MODULE 1 TASK 4 ---
duplicates = hosp_details.duplicated()

#--- Inspect data ---
print(duplicates)
```

```
0       False
1       False
2       False
3       False
4       False
        ...
2427     True
2428     True
```

```
2429     True
2430     True
2431     True
Length: 2432, dtype: bool
```

Task 5: Data Preprocessing and Cleaning for Hospitalization Details

In this task, we perform data preprocessing and cleaning on the 'hosp_details' dataset. We start by removing duplicate records to ensure data quality and accuracy. Next, we remove specific columns, 'Has_Children' and 'Is_Frequent_Treatment,' as they are not relevant to our analysis. We also rename columns to improve clarity and understanding of the data. Finally, we save the cleaned dataset as 'hospitalisation_details_cleaned.csv.' This data preprocessing and cleaning is crucial for our analysis, as it ensures that we work with accurate and meaningful data in our new project.

In [5]:

```
# --- WRITE YOUR CODE FOR TASK 5 ---
duplicates_removed = hosp_details.drop_duplicates()
check_duplicates = duplicates_removed.duplicated()
#print(check_duplicates)
cleaned_data = duplicates_removed
#cleaned_data = cleaned_data.drop(columns=['Has_Children','Is_Frequent_Treatment'])

cleaned_data = cleaned_data.rename(columns={'date?':'Date','children?':'children','charges?':'charges'})
print(cleaned_data.head())

#--- Export the df as "hospitalisation_details_cleaned.csv" ---
hospitalisation_details_cleaned = cleaned_data.to_csv('hospitalisation_details_cleaned.csv')

hosp_details = pd.read_csv('hospitalisation_details_cleaned.csv')

#--- Inspect data ---
print(hosp_details.head())
```

```
     c_id    yr  mth  Date  children  charges host_tier   Ct_tier  st_id  \
0  Id2335  1992  Jul     9         0   563.84  tier - 2  tier - 3  R1013
1  Id2334  1992  Nov    30         0   570.62  tier - 2  tier - 1  R1013
2  Id2333  1993  NaN    30         0   600.00  tier - 2  tier - 1  R1013
3  Id2332  1992  Sep    13         0   604.54  tier - 3  tier - 3  R1013
4  Id2331  1998  Jul    27         0   637.26  tier - 3  tier - 3  R1013

  Has_Children Is_Frequent_Treatment
0           no                   no
1           no                   no
2           no                   no
3           no                   no
4           no                   no
   Unnamed: 0    c_id    yr  mth  Date  children  charges host_tier   Ct_tier
\
```

```
0           0  Id2335  1992  Jul     9         0   563.84  tier - 2  tier - 3
1           1  Id2334  1992  Nov    30         0   570.62  tier - 2  tier - 1
2           2  Id2333  1993  NaN    30         0   600.00  tier - 2  tier - 1
3           3  Id2332  1992  Sep    13         0   604.54  tier - 3  tier - 3
4           4  Id2331  1998  Jul    27         0   637.26  tier - 3  tier - 3
```

```
   st_id Has_Children Is_Frequent_Treatment
0  R1013           no                    no
1  R1013           no                    no
2  R1013           no                    no
3  R1013           no                    no
4  R1013           no                    no
```

Task 6: Loading Medical Examination Data

In this task, we load the medical examination data from the 'medical_examinations.csv' file into a Pandas DataFrame named 'med_exam.' This step is essential for our new project, "Medical Dataset Analysis: Python, SQL, and Insights," as it forms the foundation for the data analysis and insights that we aim to derive from the medical dataset.

In [6]:

```
#--- Read in dataset (medical_examinations.csv) ----
# ---WRITE YOUR CODE FOR TASK 6 ---
med_exam = pd.read_csv('medical_examinations.csv')
#--- Inspect data ---
print(med_exam.describe())
```

```
              b_m_i         HBA1C
count  2374.000000  2374.000000
mean     31.152721     6.586049
std       8.839381     2.232334
min      15.010000     4.000000
25%      24.712500     4.900000
50%      30.510000     5.810000
75%      36.575000     7.997500
max      55.050000    12.000000
```

Task 7: Identifying Null Values in Medical Examination Data

In this task, we identify and count the null values in the 'med_exam' dataset. This step is crucial for our new project, as it helps us understand the extent of missing data within the dataset. Recognizing and handling null values is essential for ensuring the accuracy and quality of our data analysis and insights.

In [7]:

```
# --- WRITE YOUR CODE FOR MODULE 1 TASK 7 ---
null_values = med_exam.isna().sum()


#--- Inspect data ---
```

```
print(null_values)
cid                       0
b_m_i                     0
HBA1C                     0
h_Issues                  0
any_transplant            0
cancer_hist               0
noofmajorsurgeries        0
smoker??                  0
recovery_period        1096
dtype: int64
```

Task 8: Identifying Data Types in Medical Examination Data

In this task, we determine the data types of the columns in the 'med_exam' dataset. This step is vital for our data analysis, as it provides insights into how the data is stored and helps us select appropriate methods for further analysis. Understanding the data types is crucial for working with the dataset effectively.

In [8]:

```
# --- WRITE YOUR CODE FOR MODULE 1 TASK 8 ---
datatype = med_exam.info()

#--- Inspect data ---
print(datatype)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2374 entries, 0 to 2373
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   cid                 2374 non-null   object
 1   b_m_i               2374 non-null   float64
 2   HBA1C               2374 non-null   float64
 3   h_Issues            2374 non-null   object
 4   any_transplant      2374 non-null   object
 5   cancer_hist         2374 non-null   object
 6   noofmajorsurgeries  2374 non-null   object
 7   smoker??            2374 non-null   object
 8   recovery_period     1278 non-null   object
dtypes: float64(2), object(7)
memory usage: 167.0+ KB
None
```

Task 9: Identifying Duplicate Data in Medical Examination Data

In this task, we aim to identify and quantify the presence of duplicate data within the 'med_exam' dataset. The count of duplicates (referred to as 'duplicates' in the code) is an important metric. It

helps us understand the extent of redundancy in the dataset, which is crucial for data quality and accuracy in our analysis. By recognizing and handling duplicate records, we ensure that our insights and conclusions are based on unique, meaningful data, preventing any potential distortions caused by repeated entries.

In [9]:

```
# --- WRITE YOUR CODE FOR MODULE 1 TASK 9 ---
duplicates = med_exam.duplicated()
duplicates_count = med_exam.apply(lambda x: x.duplicated().sum())

#--- Inspect data ---
print(duplicates_count)
print(med_exam.head(10))
print(med_exam.shape)
```

```
cid                     39
b_m_i                 1039
HBA1C                 1707
h_Issues              2372
any_transplant        2372
cancer_hist           2372
noofmajorsurgeries    2370
smoker??              2372
recovery_period       2370
dtype: int64
    cid   b_m_i  HBA1C h_Issues any_transplant cancer_hist noofmajorsurgeries
\
0   Id1  47.410   7.47       No             No          No   No major surgery
1   Id2  30.360   5.77       No             No          No   No major surgery
2   Id3  34.485  11.87      yes             No          No                   2
3   Id4  38.095   6.05       No             No          No   No major surgery
4   Id5  35.530   5.45       No             No          No   No major surgery
5   Id6  32.800   6.59       No             No          No   No major surgery
6   Id7  36.400   6.07       No             No          No   No major surgery
7   Id8  36.960   7.93       No             No          No                   3
8   Id9  41.140   9.58      yes             No         Yes                   1
9  Id10  38.060  10.79       No             No          No   No major surgery

  smoker?? recovery_period
0      yes             NaN
1      yes             NaN
2      yes        Moderate
3      yes             NaN
4      yes             NaN
5      yes             NaN
6      yes             NaN
7      yes        Extended
```

```
8       yes            Short
9       yes              NaN
(2374, 9)
```

Task 10: Data Preprocessing and Cleaning for Medical Examination Data

In this task, we perform data preprocessing and cleaning on the 'med_exam' dataset. We start by removing duplicate records to ensure data quality and accuracy. Next, we remove a specific column, 'recovery_period,' as it may not be relevant to our analysis. We also rename columns to improve clarity and understanding of the data. Finally, we save the cleaned dataset as 'medical_examinations_cleaned.csv.' This data preprocessing and cleaning is crucial for our analysis, as it ensures that we work with accurate and meaningful data in our new project.

In [17]:

```
# --- WRITE YOUR CODE FOR  MODULE 1 TASK 10 ---
med_exam.info()
duplicates_removed = med_exam.drop_duplicates()
print(duplicates_removed.duplicated().sum()) #duplicates removed and validated
print(med_exam.columns)
#columns_dropped = med_exam.drop(columns=['recovery_period'])
#print(columns_dropped.columns)
#print(columns_dropped.head())
columns_renamed = columns_dropped.rename(columns={'smoker??':'smoker','cid':'C_Id','b_m_i':'BMI','noofmajorsurgeries':'Major_Surgeries_Count','any_transplant':'Transplant'})
print(columns_renamed)
med_exam = columns_renamed
print(med_exam.isna().sum())
print(med_exam.columns)

med_exam.to_csv('medical_examinations_cleaned.csv')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2374 entries, 0 to 2373
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   C_Id                  2374 non-null   object 
 1   BMI                   2374 non-null   float64
 2   HBA1C                 2374 non-null   float64
 3   h_Issues              2374 non-null   object 
 4   Transplant            2374 non-null   object 
 5   cancer_hist           2374 non-null   object 
 6   Major_Surgeries_Count  2374 non-null   object 
 7   smoker                2374 non-null   object 
dtypes: float64(2), object(6)
memory usage: 148.5+ KB
```

```
0
Index(['C_Id', 'BMI', 'HBA1C', 'h_Issues', 'Transplant', 'cancer_hist',
       'Major_Surgeries_Count', 'smoker'],
      dtype='object')
      C_Id     BMI  HBA1C h_Issues Transplant cancer_hist  \
0      Id1  47.410   7.47       No         No          No
1      Id2  30.360   5.77       No         No          No
2      Id3  34.485  11.87      yes         No          No
3      Id4  38.095   6.05       No         No          No
4      Id5  35.530   5.45       No         No          No
...    ...     ...    ...      ...        ...         ...
2369  Id128  32.775   4.72       No         No          No
2370  Id129  34.200   5.91      yes         No          No
2371  Id130  30.200   9.58       No         No          No
2372  Id131  48.320   5.77       No         No          No
2373  Id132  44.860   4.38      yes         No          No

     Major_Surgeries_Count smoker
0         No major surgery    yes
1         No major surgery    yes
2                        2    yes
3         No major surgery    yes
4         No major surgery    yes
...                     ...    ...
2369      No major surgery    yes
2370      No major surgery    yes
2371      No major surgery    yes
2372      No major surgery    yes
2373      No major surgery    yes

[2374 rows x 8 columns]
C_Id                    0
BMI                     0
HBA1C                   0
h_Issues                0
Transplant              0
cancer_hist             0
Major_Surgeries_Count   0
smoker                  0
dtype: int64
Index(['C_Id', 'BMI', 'HBA1C', 'h_Issues', 'Transplant', 'cancer_hist',
       'Major_Surgeries_Count', 'smoker'],
      dtype='object')
```