

NosQL Database

HEALTHCARE DATASET



GROUP 5

OBIAKU VALENTINE CHIKE
JANVI
KARMJEET KAUR
RAMANDEEP SINGH

ROLE SPECIFICATION

- JANVI – Data Acquisition, Cleaning Data
- RAMANDEEP SINGH – Conversion of Data , Loading Data into NoSQL DATABASE , Querying Data
- KARMJEET KAUR – Data visualization and predictive analysis
- OBIAKU VALENTINE CHIKE- Queries in pyspark and drawing conclusions.

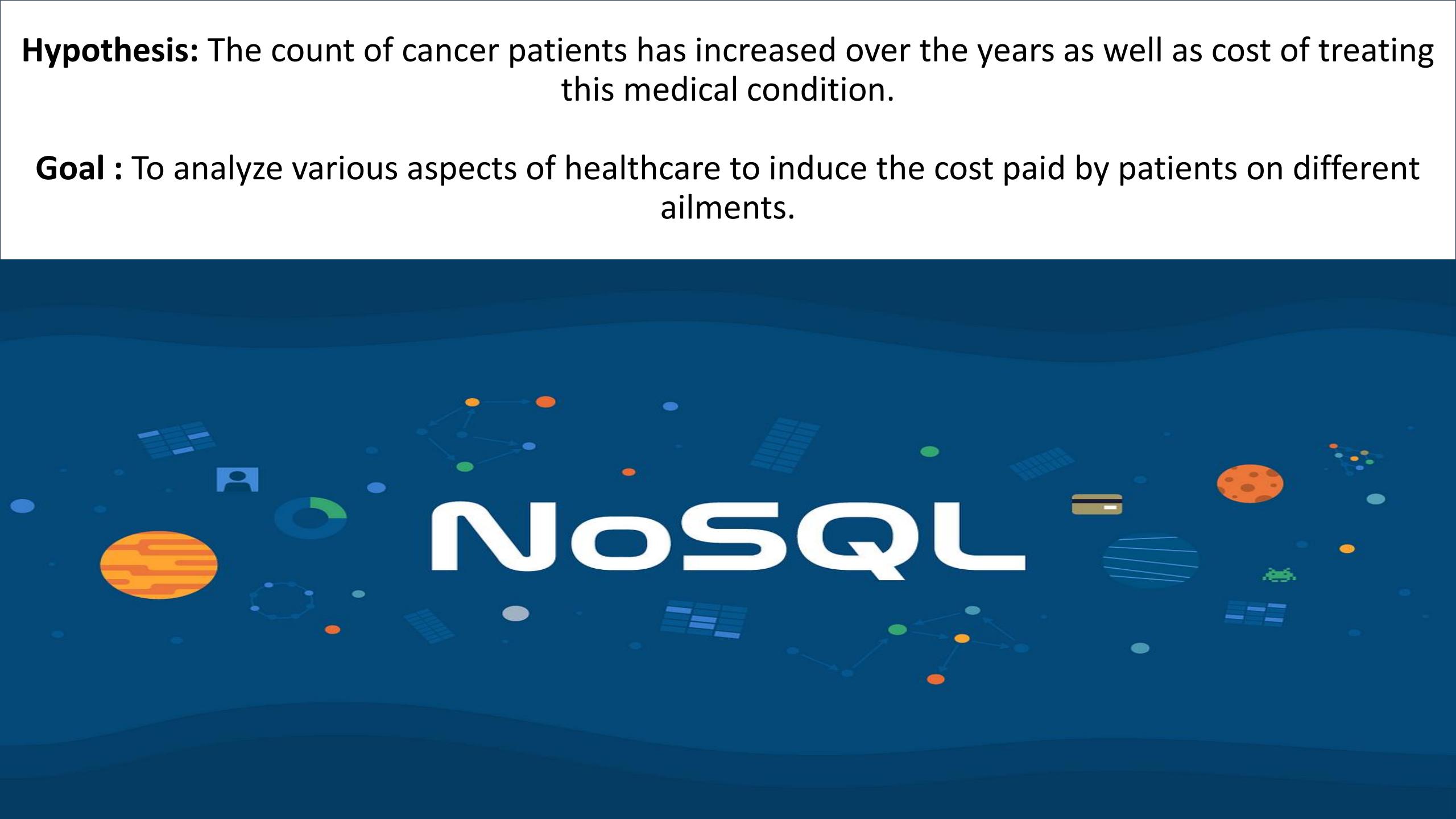
ANALYSING HEALTHCARE DATASET FROM KAGGLE

NoSQL

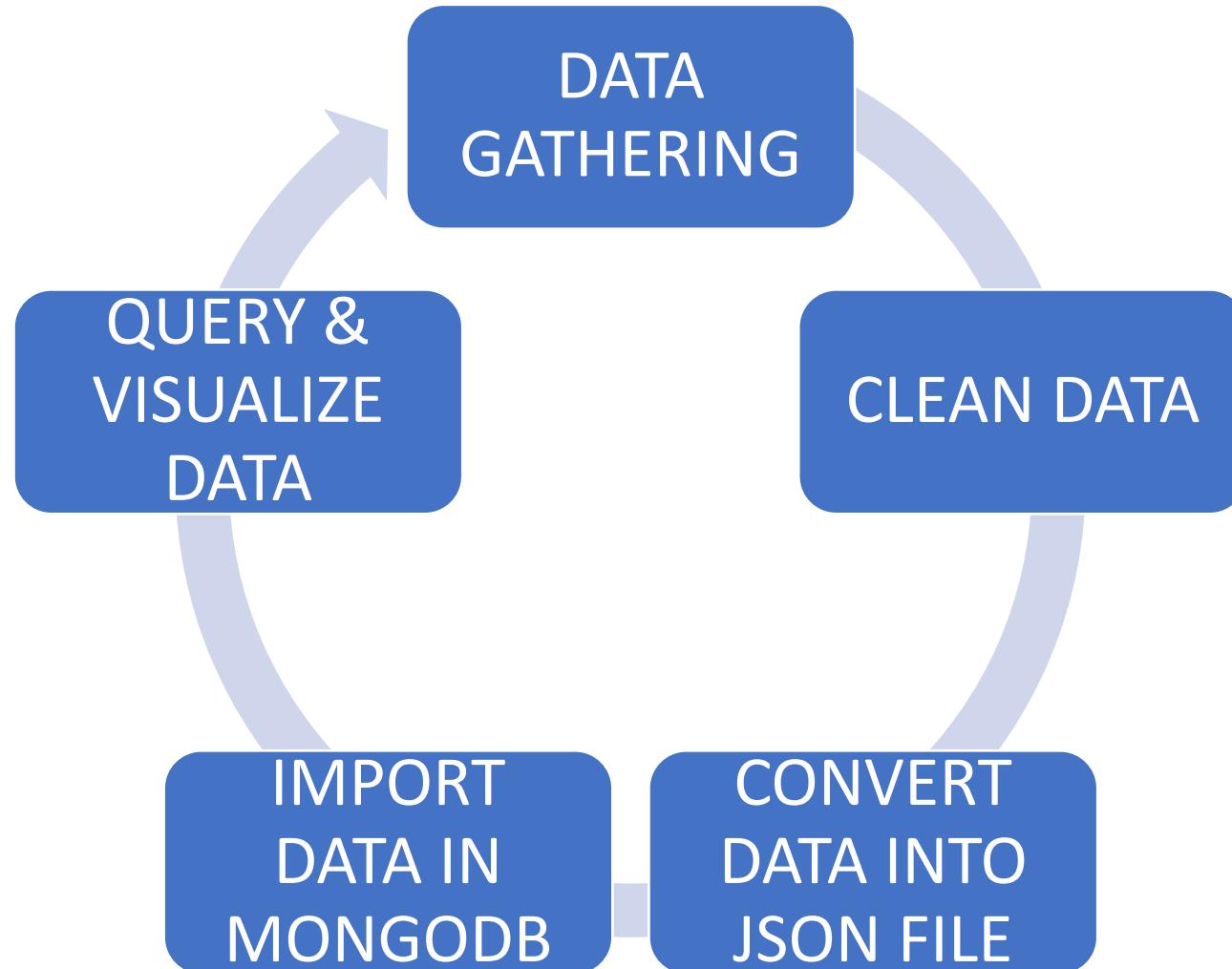


Hypothesis: The count of cancer patients has increased over the years as well as cost of treating this medical condition.

Goal : To analyze various aspects of healthcare to induce the cost paid by patients on different ailments.



NoSQL



DATASET OVERVIEW

- DATASET IS FROM KAGGLE
- HEALTHCARE DATASET 2023
 - DATASET IS IN CSV FILE

DATASET CONTAINS

- GENDER
- BLOOD TYPE
- BILLING AMOUNT
- ADMISSION
- MEDICATION
- TEST RESULTS etc.

PROJECT TOOLS

- PYTHON
- CLOUD MONGODB
 - PYMONGO
- JUPYTER NOTEBOOK
- VISUAL STUDIO CODE
 - TABLEAU
- CSV TO JSON CONVERTER
 - APACHE SPARK
 - MATPLOTLIB
 - PANDAS

healthcare_dataset (1) - Excel (Product Activation Failed)

File Home Insert Page Layout Formulas Data Review View Tell me what you want to do... Share

Cut Copy Format Painter

Font Alignment Number Styles Cells Editing

A1 Name Age Gender Blood Type Medical Condition Date of Admission Doctor Hospital Insurance Billing Amount Room Number Admission Discharge Date Medication Test Results

1 Name Age Gender Blood Type Medical Condition Date of Admission Doctor Hospital Insurance Billing Amount Room Number Admission Discharge Date Medication Test Results

2 Tiffany Ra 81 Female O- Diabetes 11/17/2022 Patrick Pai Wallace-H Medicare 37490.98336 146 Elective 12/1/2022 Aspirin Inconclusive

3 Ruben Bur 35 Male O+ Asthma 6/1/2023 Diane Jack Burke, Gri UnitedHealthcare 47304.06485 404 Emergent 6/15/2023 Lipitor Normal

4 Chad Byrd 61 Male B- Obesity 1/9/2019 Paul Baker Walton LL Medicare 36874.897 292 Emergent 2/8/2019 Lipitor Normal

5 Antonio Fr 49 Male B- Asthma 5/2/2020 Brian Chai Garcia Ltd Medicare 2303.32209 480 Urgent 5/3/2020 Penicillin Abnormal

6 Mrs. Brandy Flowers Male O- Arthritis 7/9/2021 Dustin Gri Jones, Bro UnitedHealthcare 18086.34418 477 Urgent 8/2/2021 Paracetamol Normal

7 Patrick Pai 41 Male AB+ Arthritis 8/20/2020 Robin Gre Boyd PLC Aetna 22522.36338 180 Urgent 8/23/2020 Aspirin Abnormal

8 Charles Hc 82 Male AB+ Hypertens 3/22/2021 Patricia Bi Wheeler, Cigna 39593.43576 161 Urgent 4/15/2021 Lipitor Abnormal

9 Patty Norr 55 Female O- Arthritis 5/16/2019 Brian Ken Brown Inc Blue Cross 13546.81725 384 Elective 6/2/2019 Aspirin Normal

10 Ryan Haye 33 Male A+ Diabetes 12/17/2020 Kristin Dur Smith, Edu Aetna 24903.03727 215 Elective 12/22/2020 Aspirin Abnormal

11 Sharon Pe 39 O- Asthma 12/15/2022 Jessica Bai Bonn-Go Blue Cross 22788.23603 310 Urgent 12/16/2022 Aspirin Normal

12 Amy Robe 45 Male B- Cancer 4/13/2021 Anthony F Little-Sper Aetna 40325.07139 306 Emergent 5/11/2021 Penicillin Abnormal

13 Mrs. Carol 23 Female O- Hypertens 6/9/2019 William M Rose Inc Medicare 6185.90353 126 Emergent 6/26/2019 Paracetamol Inconclusive

14 Christina L 85 Female A+ Diabetes 11/29/2021 Laura Rob Malone, T Aetna 4835.94565 444 Elective 12/14/2021 Aspirin Inconclusive

15 William Pa 72 Female A+ Diabetes 7/29/2021 James Car Richardson Cigna 13669.37774 492 Elective 8/14/2021 Aspirin Normal

16 Michael Bi 65 Female AB+ Cancer 6/5/2021 Katherine Castaneda Cigna 10342.83612 120 Emergent 6/25/2021 Ibuprofen Inconclusive

17 Brian Dor 32 Female O+ Arthritis 8/7/2021 Curtis Smi Burch-Wh Aetna 27174.94291 492 Emergent 8/14/2021 Aspirin Inconclusive

18 Olivia Gon 64 Male AB- Diabetes 11/15/2019 Clayton M Cunningham Aetna 17394.99426 315 Elective 12/4/2019 Aspirin Inconclusive

19 Teresa Cal 23 Male A+ Arthritis 3/8/2022 Debra Me Bell, McKn Medicare 45213.53763 475 Elective 3/16/2022 Ibuprofen Inconclusive

20 Desiree W 66 Male O+ Obesity 6/19/2022 Megan Sai Pugh-Rogg UnitedHealthcare 4262.911578 125 Elective 6/29/2022 Aspirin Inconclusive

21 Sally Shaw 80 Male O- Arthritis 7/10/2019 Zachary Horton DDS Blue Cross 16609.31182 366 Emergent 8/7/2019 Ibuprofen Inconclusive

22 William Jo 55 Female AB+ Arthritis 2/25/2023 Kelly Thon Pearson Li Aetna 32263.62216 238 Emergent 3/27/2023 Penicillin Normal

23 Steven Bei 79 Male O- Asthma 12/12/2022 Michael C Schultz-Po Blue Cross 42610.70456 364 Urgent 12/26/2022 Penicillin Abnormal

24 Haley Li 51 Male B- Obesity 10/9/2022 Nicole Wo Jordan Inc Medicare 16701.34713 130 Emergent 11/1/2022 Lipitor Abnormal

healthcare_dataset (1)

Kevin John 78 Male O- Asthma 2/6/2021 Michael Sr Hernandez Medicare 9007.967866 247 Urgent 3/3/2021 Lipitor Abnormal

Rebecca P 51 Male B- Asthma 1/29/2021 Pamela Br Jameson Blue Cross 39031.36264 228 Elective 2/23/2021 Paracetamol Abnormal

Linda Cha' 33 Male A- Cancer 12/1/2019 Tiffany Cr Winters ai Cigna 24609.37463 137 Emergent 12/18/2019 Penicillin Abnormal

Jennifer R 83 Female AB- Asthma 11/12/2019 Rachel Sul Dule Gro Cigna 5098.663128 192 Emergent 11/24/2019 Lipitor Abnormal

Anna Adair 70 Female A+ Obesity 9/6/2022 Samuel Ta Snow Gro Medicare 1000.180837 258 Emergent 9/18/2022 Paracetamol Abnormal

Mariah W 72 Male A- Diabetes 3/19/2021 John Harv Anderson Blue Cross 24913.55265 219 Elective 3/20/2021 Penicillin Normal

Brendan N 30 Male AB+ Diabetes 2/24/2021 Sylvia Joh Gonzalez UnitedHealthcare 24274.1416 364 Elective 3/21/2021 Paracetamol Normal

Michael Bi 19 Male B- Cancer 6/30/2021 Janice Var Wiggins Ltd 16171.94856 414 Emergent 7/14/2021 Lipitor Inconclusive

Miguel Ba 23 Male O- Arthritis 3/25/2019 Mark Hill Ramirez Jr UnitedHealthcare 39329.11431 110 Urgent 4/8/2019 Penicillin Inconclusive

Mr. Christ 81 Male AB+ Hypertens 8/15/2021 Amber Go Ortega an Cigna 22519.33618 465 Emergent 9/6/2021 Lipitor Inconclusive

Theresa M 46 Female A+ Asthma 3/3/2023 Wendy Ga Wood-Aetna 1996.310721 469 Urgent 3/5/2023 Penicillin Inconclusive

Laura Ada 48 B- Diabetes 8/7/2021 Michael H Johnson, Cigna 5470.209687 182 Emergent 8/9/2021 Ibuprofen Inconclusive

Lauren Ba 34 Male O+ Hypertens 6/6/2019 Emily Wes Gallagher-Medicare 16036.10999 119 Emergent 6/30/2019 Paracetamol Normal

Christina 84 Female O+ Hypertens 10/13/2019 Cody Wrig Harris and Aetna 10803.73034 388 Emergent 10/29/2019 Aspirin Inconclusive

Jasmine Si 37 Male AB- Obesity 10/20/2022 Melissa Ke Kerr LLC Aetna 38872.64522 412 Elective 11/4/2022 Penicillin Abnormal

John Griffi 78 Female B- Cancer 4/3/2020 Kelsey Cl Aguirre LL Aetna 25948.50719 359 Emergent 4/6/2020 Lipitor Normal

Justin Kau 82 Female AB- Diabetes 2/18/2021 Corey Sutt Rodriguez Cigna 46461.82252 186 Urgent 3/6/2021 Lipitor Abnormal

Dylan Mck 47 Male B- Cancer 6/14/2019 Ashley Ed Johnson, FMedicare 10705.0984 437 Emergent 7/13/2019 Penicillin Inconclusive

Olivia Aya 60 Male A- Arthritis 11/26/2022 Leon Price Cook-Wrig Blue Cross 49576.87403 132 Elective 12/8/2022 Penicillin Abnormal

Frank Mcc 70 Male A+ Obesity 8/25/2023 Kyle Estrada Herrera-V Blue Cross 8005.828337 271 Urgent 9/12/2023 Ibuprofen Inconclusive

Rachael D 57 Female O- Cancer 3/29/2022 Jared Harmon Cigna 41295.39975 361 Emergent 4/2/2022 Aspirin Normal

Lori Owen 19 Male A+ Arthritis 7/4/2021 Adrian Pie Diaz-Fergi Blue Cross 20080.29764 303 Emergent 7/29/2021 Ibuprofen Normal

Frank Reic 61 Female AB+ Obesity 11/4/2019 Jesus Snyc Nichols, St Blue Cross 1888.002111 317 Emergent 11/19/2019 Penicillin Inconclusive

healthcare_dataset (1)

RAW DATASET FROM KAGGLE

<https://www.kaggle.com/datasets/prasad2/healthcare-dataset>

- UNCLEANED
- MISSING DATA IN ROWS AND COLUMNS

WHY THERE IS NEED FOR DATA CLEANING

- **Missing Data:**

- Problem:** Missing values can lead to biased or inaccurate analyses.
- Solution:** Options include removing rows with missing values, filling missing values with statistical measures (mean, median, mode), or using sophisticated imputation techniques.

- **Duplicate Data:**

- Problem:** Duplicate entries can skew statistical analyses and lead to incorrect conclusions.
- Solution:** Identify and remove duplicate rows based on certain columns or criteria. Ensure data uniqueness before analysis.

- **Inaccurate Data:**

- Problem:** Errors in data entry or measurement inaccuracies can compromise the quality of analyses.
- Solution:** Review and validate data entries. Utilize external sources or cross-check information to verify accuracy. Correct errors manually or through automation.

- **Outliers:**

- Problem:** Outliers can significantly impact statistical measures and machine learning models.
- Solution:** Identify and handle outliers using statistical methods or visualization tools. Options include removing outliers, transforming data, or using robust statistical measures.

C: > Users > DELL > Downloads > No SQL project > project nosql - Copy (2).ipynb > import pandas as pd # Import the pandas library and alias it as 'pd'

+ Code + Markdown | ▶ Run All ⌂ Clear All Outputs | ⌂ Outline ...

```

import pandas as pd      # Import the pandas library and alias it as 'pd'
dataset=pd.read_csv(r"C:\Users\shuba\Desktop\healthcare_dataset (1).csv") # Read the CSV file into a pandas DataFrame
[5]
dataset.head()          # Display the first 5 rows of the DataFrame
[6]
...
   Name  Age  Gender  Blood Type  Medical Condition  Date of Admission  Doctor  Hospital  Insurance Provider  Billing Amount  Room Number  Admission Type  Discharge Date  Medication
0  Tiffany Ramirez  81.0  Female    O-        Diabetes  2022-11-17  Patrick Parker  Wallace-Hamilton  Medicare  37490.98336  146  Elective  2022-12-01  Aspirin
1  Ruben Burns    35.0  Male     O+       Asthma  2023-06-01  Diane Jackson  Burke, Griffin and Cooper  UnitedHealthcare  47304.06485  404  Emergency  2023-06-15  Lipitor
2  Chad Byrd     61.0  Male     B-       Obesity  2019-01-09  Paul Baker  Walton LLC  Medicare  36874.89700  292  Emergency  2019-02-08  Lipitor
3  Antonio Frederick  49.0  Male     B-       Asthma  2020-05-02  Brian Chandler  Garcia Ltd  Medicare  23303.32209  480  Urgent  2020-05-03  Penicillin
4  Mrs. Brandy Flowers  NaN  Male     O-      Arthritis  2021-07-09  Dustin Griffin  Jones, Brown and Murray  UnitedHealthcare  18086.34418  477  Urgent  2021-08-02  Paracetamol
[7]
dataset.info()          # Display information about the DataFrame
[8]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Name            9975 non-null    object  
 1   Age             9972 non-null    float64 
 2   Gender          9977 non-null    object  
 3   Blood Type     9981 non-null    object  
 4   Medical Condition 10000 non-null  object  
 5   Date of Admission 10000 non-null  object  
 6   Doctor          10000 non-null    object  
 7   Hospital         9983 non-null    object  
 8   Insurance Provider 9991 non-null  object  
 9   Billing Amount   10000 non-null    float64 
 10  Room Number     10000 non-null    int64  
 11  Admission Type  9998 non-null    object  
 12  Discharge Date   10000 non-null    object  
 13  Medication        10000 non-null    object  
 14  Test Results     9990 non-null    object  
dtypes: float64(2), int64(1), object(12)

```

DATA CLEANING

- USING VISUAL STUDIO CODE
- IMPORT THE PANDAS LIBRARY AS “pd”
- LOAD THE HEALTHCARE DATASET INTO THE PANDAS DATAFRAME
 - DISPLAY THE FIRST 5 ROWS
 - DISPLAY INFORMATION ABOUT THE DATA

```
dataset.describe() # Display summary statistics of the numerical columns in the DataFrame
```

11] ..

	Age	Billing Amount	Room Number
count	9972.000000	10000.000000	10000.000000
mean	51.444444	25516.806778	300.082000
std	19.588112	14067.292709	115.806027
min	18.000000	1000.180837	101.000000
25%	35.000000	13506.523967	199.000000
50%	52.000000	25258.112565	299.000000
75%	68.000000	37733.913725	400.000000
max	85.000000	49995.902280	500.000000

```
missing_values=dataset.isnull().sum() # Check for missing values in each column
print("Columns with missing values:") # Print columns with missing values and the count of missing values
print(missing_values)
```

14] ..

```
Columns with missing values:
Name          25
Age           28
Gender         23
Blood Type    19
Medical Condition   0
Data of Admition?  0
```

Ln 1, Col 197 CRLF □

```
C:\Users\DELL\Downloads>No SQL project>project nosql - Copy (2).ipynb># Fill missing values in the 'Admission Type' column with the mode
```

+ Code + Markdown | Run All | Clear All Outputs | Outline ... Select Kernel

△ 0 △ 0

```
# filling missing values of age column with mean
dataset['Age'].fillna(dataset['Age'].mean(), inplace=True)
```

15] Python

```
# Fill missing values in the 'Blood Type' column with 'O'
dataset['Blood Type'].fillna('O+', inplace=True)
```

18] Python

```
missing_values=dataset.isnull().sum() # Check for missing values in each column after filling missing values in 'Blood Type'
print("Columns with missing values:") # Print columns with missing values and the count of missing values
print(missing_values)
```

23] Python

```
# Fill missing values in the 'Admission Type' column with the mode
dataset['Admission Type'].fillna(dataset['Admission Type'].mode().iloc[0], inplace=True)
```

33] Python

```
# Fill missing values in the 'Insurance Provider' column using forward fill (pad)
dataset['Insurance Provider'].fillna(dataset['Insurance Provider'].interpolate(method='pad'), inplace=True)
```

- DISPLAY THE SUMMARY OF THE DATA
- CHECK FOR MISSING VALUES IN EACH COLUMNS
 - FILL MISSING VALUES OF “AGE” COLUMN WITH MEAN VALUES
 - FILL MISSING VALUES OF “BLOOD TYPE” COLUMN WITH “O”
- FILL MISSING VALUES OF “ADMISSION TYPE” COLUMN WITH MODE VALUES
- FILL MISSING VALUES OF “INSURANCE PROVIDER” COLUMN USING FORWARD FILL (PAD)

+ Code + Markdown | Run All Clear All Outputs | Outline ...

Select Kernel

```
new_missing_values=dataset.isnull().sum()  
print("Columns with missing values:")  
print(new_missing_values[new_missing_values>0])
```

[34]

Python

... Columns with missing values:

Name	25
Gender	23
Hospital	17
Test Results	10
dtype:	int64

```
# Drop rows with any missing values in the entire dataset  
dataset.dropna(inplace=True)
```

[35]

Python

+ Code + Markdown | Run All Clear All Outputs | Outline ...

Select Kernel

```
# Save the cleaned dataset to a CSV file  
dataset.to_csv('cleaned_dataset.csv', index=False)
```

[37]

- CHECK MISSING VALUES IN EACH COLUMN AFTER THE PREVIOUS OPERATIONS
- DROP ROWS WITH ANY MISSING VALUES IN THE ENTIRE DATASET
- FINALLY, SAVE THE CLEANED DATASET TO A CSV FILE

cleaned_dataset - Excel (Product Activation Failed)

File Home Insert Page Layout Formulas Data Review View Tell me what you want to do...

Cut Copy Format Painter Clipboard

Font Alignment Number Styles Cells Editing

Name

Name	Age	Gender	Blood Typ	Medical C	Date of Admissic	Doctor	Hospital	Insurance	Billing Amount	Room Numbe	Admission	Discharge Date	Medication	Test Re
Tiffany Ramirez	81	Female	O-	Diabetes	11/17/2022	Patrick Parker	Wallace-Hamilton	Medicare	37490.98336	146	Elective	12/1/2022	Aspirin	Inconcl
Ruben Burns	35	Male	O+	Asthma	6/1/2023	Diane Jackson	Burke, Griffin and UnitedHea	UnitedHealthcare	47304.06485	404	Emergency	6/15/2023	Lipitor	Normal
Chad Byrd	61	Male	B-	Obesity	1/9/2019	Paul Baker	Walton LLC	Medicare	36874.897	292	Emergency	2/8/2019	Lipitor	Normal
Antonio Frederick	49	Male	B-	Asthma	5/2/2020	Brian Chandler	Garcia Ltd	Medicare	23303.32209	480	Urgent	5/3/2020	Penicillin	Abnormal
Mrs. Brandy Flowe	51.44444	Male	O-	Arthritis	7/9/2021	Dustin Griffin	Jones, Brown and UnitedHea	UnitedHealthcare	18086.34418	477	Urgent	8/2/2021	Paracetamol	Normal
Patrick Parker	41	Male	AB+	Arthritis	8/20/2020	Robin Green	Boyd PLC	Aetna	22522.36338	180	Urgent	8/23/2020	Aspirin	Abnormal
Charles Horton	82	Male	AB+	Hypertens	3/22/2021	Patricia Bishop	Wheeler, Bryant a Cigna	Blue Cross Blue Shield	39593.43576	161	Urgent	4/15/2021	Lipitor	Abnormal
Patty Norman	55	Female	O-	Arthritis	5/16/2019	Brian Kennedy	Brown Inc	Blue Cross Blue Shield	13546.81725	384	Elective	6/2/2019	Aspirin	Normal
Ryan Hayes	33	Male	A+	Diabetes	12/17/2020	Kristin Dunn	Smith, Edwards ar	Aetna	24903.03727	215	Elective	12/22/2020	Aspirin	Abnormal
Amy Roberts	45	Male	B-	Cancer	4/13/2021	Anthony Rober	Little-Spencer	Aetna	40325.07139	306	Emergency	5/11/2021	Penicillin	Abnormal
Mrs. Caroline Farr	23	Female	O-	Hypertens	6/9/2019	William Miller	Rose Inc	Medicare	6185.90353	126	Emergency	6/26/2019	Paracetamol	Inconcl
Christina Williams	85	Female	A+	Diabetes	11/29/2021	Laura Roberts	Malone, Thompson	Aetna	4835.94565	444	Elective	12/14/2021	Aspirin	Inconcl
William Page	72	Female	A+	Diabetes	7/29/2021	James Carney	Richardson-Powell	Cigna	13669.37774	492	Elective	8/14/2021	Aspirin	Normal
Michael Bradshaw	65	Female	AB+	Cancer	6/5/2021	Katherine Low	Castaneda-Hardy	Cigna	10342.83612	120	Emergency	6/25/2021	Ibuprofen	Inconcl
Brian Dorsey	32	Female	O+	Arthritis	8/7/2021	Curtis Smith	Burch-White	Aetna	27174.94291	492	Emergency	8/14/2021	Aspirin	Inconcl
Olivia Gonzalez	64	Male	AB-	Diabetes	11/15/2019	Clayton McKnight	Cunningham and	Aetna	17394.99426	315	Elective	12/4/2019	Aspirin	Inconcl
Teresa Caldwell	23	Male	A+	Arthritis	3/8/2022	Debra Meyers	Bell, McKnight and	Medicare	45213.53763	475	Elective	3/16/2022	Ibuprofen	Inconcl
Desiree Williams	66	Male	O+	Obesity	6/19/2022	Megan Sanders	Pugh-Rogers	UnitedHealthcare	4262.911578	125	Elective	6/29/2022	Aspirin	Inconcl
William Johnson	55	Female	AB+	Arthritis	2/25/2023	Kelly Thompson	Pearson LLC	Aetna	32263.62216	238	Emergency	3/27/2023	Penicillin	Normal
Steven Bennett	79	Male	O-	Asthma	12/12/2022	Michael Chang	Schultz-Powers	Blue Cross Blue Shield	42610.70456	364	Urgent	12/26/2022	Penicillin	Abnormal
Haley Li	51	Male	B-	Obesity	10/9/2022	Nicole Wood	Jordan Inc	Medicare	16701.34713	130	Emergency	11/1/2022	Lipitor	Abnormal
Angela Brown	33	Female	B-	Diabetes	1/10/2019	Angela Kim	Lewis-Nelson	UnitedHealthcare	22331.28016	120	Urgent	1/31/2019	Aspirin	Abnormal
Beverly Miller	54	Male	A-	Asthma	8/5/2022	Jodi Holland	Vaughn PLC	Cigna	41319.50032	293	Urgent	9/3/2022	Paracetamol	Normal

CLEANED DATASET FROM KAGGLE

- CLEANED
- NO MISSING ROWS
- NO MISSING COLUMNS etc

Step 4: Create Custom Output via Template (optional) ▾

Step 5: Generate output

Choose Conversion Type:

CSV To JSON

CSV To Keyed JSON

CSV To JSON Array

CSV To JSON Column Array

Result Data: 

```
[  
 {  
   "Name": "Tiffany Ramirez",  
   "Age": "81.0",  
   "Gender": "Female",  
   "Blood Type": "O-",  
   "Medical Condition": "Diabetes",  
   "Date of Admission": "2022-11-17",  
   "Doctor": "Patrick Parker",  
   "Hospital": "Wallace-Hamilton",  
   ...  
 }
```

Save your result: .json  Download Result EOL: CRLF ▾

You can Save the complete data and settings, and then later Load them from your saved file.

Load Saved Form No file chosen

SOFTWARE TO TRANSFORM CSV FILE TO JSON FILE

IMPORT DATASET FROM JSON FILE INTO MONGODB

- USING VISUAL STUDIO CODE
- IMPORT THE PYMONGO LIBRARY
 - IMPORT THE JSON MODULE
- CONNECT TO THE MONGODB CLOUD SERVER
 - CREATE DATABASE IN MONGODB “PROJECT 2023”
- CREATE COLLECTION “HEALTHCARE”
- INSERT THE DATASET IN JSON FILE IN THE MONGODB COLLECTION “HEALTCARE”

The screenshot shows a Visual Studio Code interface with a dark theme. On the left is a sidebar with file navigation. The main area displays a Python script named 'Project.py'. The code imports 'pymongo' and 'json', connects to a MongoDB cloud database, reads a JSON file, and inserts its contents into a collection named 'Healthcare'. Red arrows point from the numbered steps on the left to specific lines of code: step 1 points to 'import pymongo', step 2 to 'import json', step 3 to 'client=pymongo.MongoClient(conn_str)', step 4 to 'myDB=client["Project_2023"]', step 5 to 'Healthcare=myDB["Healthcare"]', step 6 to 'data = json.load(file)', and step 7 to 'Healthcare.insert_many(data)'. The code uses color-coded syntax highlighting.

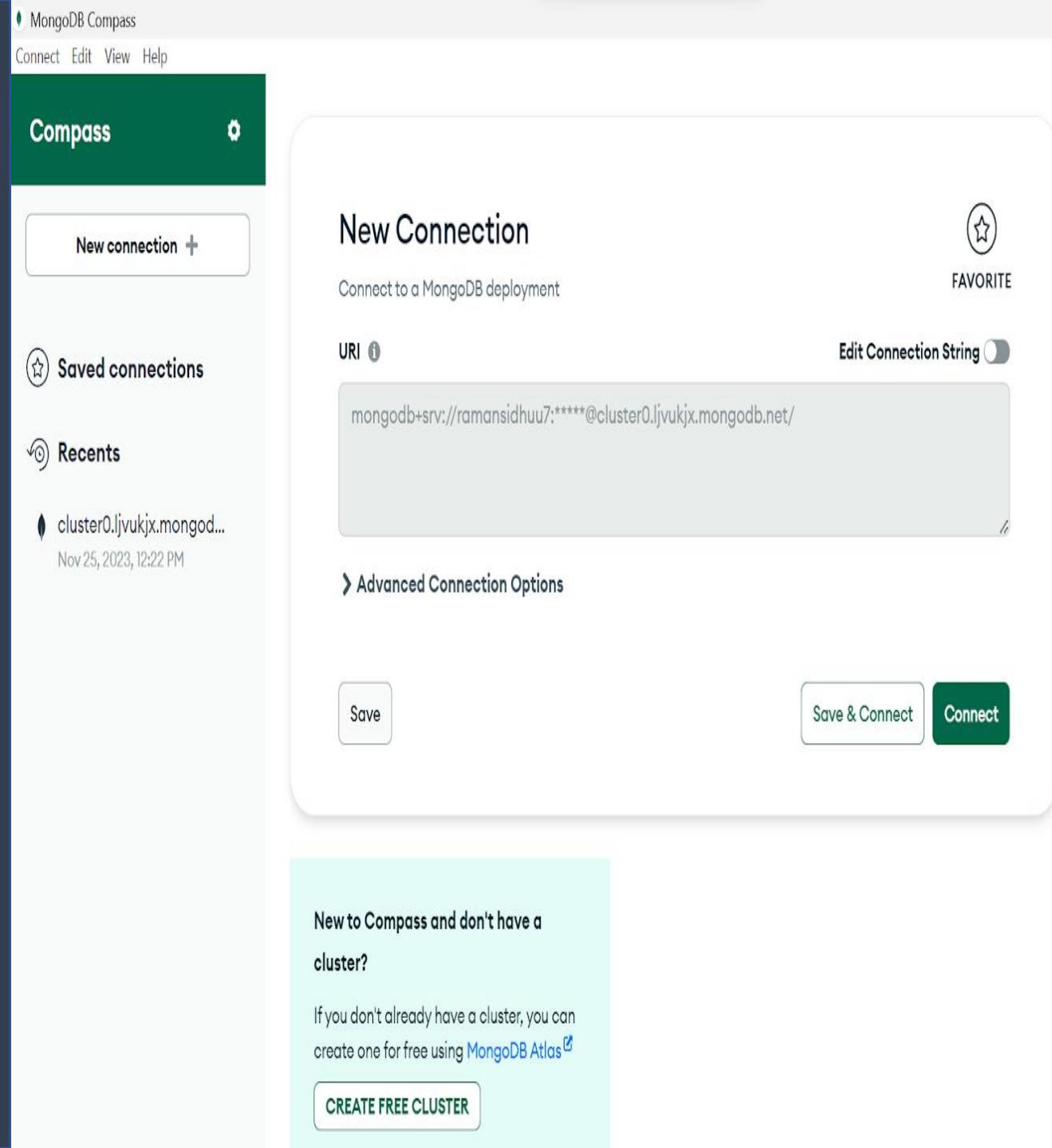
```
Project.py X cleaned_dataset.json
Project.py > ...
1 import pymongo ←
2 import json ←
3 conn_str=f"mongodb+srv://ramansidhuu7:Canada123@cluster0.ljvukjx.mongodb.net/?retryWrites=true&w=majority"
4 client=pymongo.MongoClient(conn_str) ←
5
6 myDB=client["Project_2023"] ←
7 Healthcare=myDB["Healthcare"] ←
8
9 # Read the JSON file
10 with open(r"C:/Users/91628/Desktop/NoSQL PROJECT/cleaned_dataset.json") as file:
11     data = json.load(file) ←
12
13 # Insert the data into the MongoDB collection
14 Healthcare.insert_many(data) ←
15
16
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\91628\Desktop\NoSQL PROJECT> python -u "c:\Users\91628\Desktop\NoSQL PROJECT\Project.py"
PS C:\Users\91628\Desktop\NoSQL PROJECT>
```

MONGODB CONNECTION USING MONGODB COMPASS

MongoDB Compass is the official graphical user interface (GUI) for MongoDB, a popular NoSQL database. It is designed to provide a visual representation and interaction with MongoDB databases, making it easier for developers, database administrators, and other users to work with MongoDB.



HEALTHCARE DATASET IMPORTED INTO MONGODB COMPASS

MongoDB Compass - cluster0.ljvukjx.mongodb.net/Project_2023.Healthcare

Connect Edit View Collection Help

cluster0.ljvukjx.... Project_2023.Healthcare

Documents Project_2023.Healthcare 9.9k 1

My Queries Databases Search DOCUMENTS INDEXES

Assignment Library Project_2023

Healthcare ADD DATA EXPORT DATA Explain Reset Find Options

1-20 of 9926

`_id: ObjectId('6562975265a0643e864d51de')`
Name: "Tiffany Ramirez"
Age: 81
Gender: "Female"
Blood Type: "O-"
Medical Condition: "Diabetes"
Date of Admission: "2022-11-17"
Doctor: "Patrick Parker"
Hospital: "Wallace-Hamilton"
Insurance Provider: "Medicare"
Billing Amount: 37490.98336
Room Number: 146
Admission Type: "Elective"
Discharge Date: "2022-12-01"
Medication: "Aspirin"

The screenshot shows the MongoDB Compass application interface. The title bar indicates the connection is to 'cluster0.ljvukjx.mongodb.net/Project_2023.Healthcare'. The left sidebar lists databases: Assignment, Library, Project_2023 (which is expanded to show Healthcare), Python, admin, library, local, mid_sem, no_sql, production, and regex. The main area is titled 'Project_2023.Healthcare' and shows '9.9k' documents and '1' index. It has tabs for Documents, Aggregations, Schema, Indexes, and Validation. A search bar is present. Below the tabs is a filter bar with 'Type a query: { field: 'value' } or Generate query'. There are buttons for Explain, Reset, Find, and Options. The document list shows the first document with its fields: _id, Name, Age, Gender, Blood Type, Medical Condition, Date of Admission, Doctor, Hospital, Insurance Provider, Billing Amount, Room Number, Admission Type, Discharge Date, and Medication. The document details are: _id: ObjectId('6562975265a0643e864d51de'), Name: "Tiffany Ramirez", Age: 81, Gender: "Female", Blood Type: "O-", Medical Condition: "Diabetes", Date of Admission: "2022-11-17", Doctor: "Patrick Parker", Hospital: "Wallace-Hamilton", Insurance Provider: "Medicare", Billing Amount: 37490.98336, Room Number: 146, Admission Type: "Elective", Discharge Date: "2022-12-01", Medication: "Aspirin". The bottom right corner of the document list has icons for edit, delete, copy, and refresh.

WHY MONGO DB ?

As Mongo db is an open source software which is used to query and store NoSQL data in flexible environment. Thus Jason files can be easily operated on closed services with help of cloud Mongo db . We have queried the NoSQL data in pymongo to derive feasible insights from the data.

MONGODB QUERIES

HOW MANY PATIENTS ARE AFFECTED BY ASTHMA

```
22     count=Healthcare.count_documents({"Medical Condition":"Asthma"})  
23     print("Number of patients effected by Asthma=",count)  
24
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\91628> python -u "c:\Users\91628\Desktop\NoSQL PROJECT'\Project.py"  
Number of patients effected by Asthma= 1695  
PS C:\Users\91628>
```

WHAT IS THE AVERAGE BILLING AMOUNT SPENT BY CANCER PATIENTS

```
Atlas atlas-beh2jl-shard-0 [primary] Project_2023> db.Healthcare.aggregate([ { $match: { "Medical Condition": "Cancer" } }, { $group:  
    { _id: null, averageBillingAmount: { $avg: "$Billing Amount" } } } ])  
[ { _id: null, averageBillingAmount: 25512.55250590592 } ]  
Atlas atlas-beh2jl-shard-0 [primary] Project_2023> |
```

RETRIEVE PATIENTS WHO'S AGE LIES BETWEEN 40 -80 AND FIND THE COUNT OF THOSE PATIENTS.

```
mongosh mongodb+srv://<...> + ▾
Atlas atlas-beh2jl-shard-0 [primary] Project_2023> db.Healthcare.find({ "Age": { "$gte": 40, "$lte": 80 } })
[
  {
    "_id": ObjectId("656298d4adf425360a3fc346"),
    "Name": "Chad Byrd",
    "Age": 61,
    "Gender": "Male",
    "Blood Type": "B-",
    "Medical Condition": "Obesity",
    "Date of Admission": "2019-01-09",
    "Doctor": "Paul Baker",
    "Hospital": "Walton LLC",
    "Insurance Provider": "Medicare",
    "Billing Amount": 36874.897,
    "Room Number": 292,
    "Admission Type": "Emergency",
    "Discharge Date": "2019-02-08",
    "Medication": "Lipitor",
    "Test Results": "Normal"
  },
  {
    "_id": ObjectId("656298d4adf425360a3fc347"),
    "Name": "Antonio Frederick",
    "Age": 49,
    "Gender": "Male",
    "Blood Type": "B-",
    "Medical Condition": "Asthma",
    "Date of Admission": "2020-05-02",
    "Doctor": "Brian Chandler",
    "Hospital": "Garcia Ltd",
    "Insurance Provider": "Medicare",
  }
]
Atlas atlas-beh2jl-shard-0 [primary] Project_2023> db.Healthcare.find({ "Age": { "$gte": 40, "$lte": 80 } }).count()
6000
Atlas atlas-beh2jl-shard-0 [primary] Project_2023>
```

WHAT IS THE NUMBER OF FEMALES ADMITTED FOR EMERGENCY

```
Atlas atlas-beh2jl-shard-0 [primary] Project_2023> db.Healthcare.countDocuments({ "Admission Type": "Emergency", "Gender": "Female" })
)
1687
Atlas atlas-beh2jl-shard-0 [primary] Project_2023> |
```

HOW MANY PATIENTS HAVE TEST RESULTS AS NORMAL

```
Atlas atlas-beh2jl-shard-0 [primary] Project_2023> db.Healthcare.find({"Test Results":"Normal"}).count()
3238
Atlas atlas-beh2jl-shard-0 [primary] Project_2023> |
```

Retrieve the medication prescribed to the highest number of patients also retrieve the number of patients.

```
Atlas atlas-beh2jl-shard-0 [primary] Project_2023> db.Healthcare.aggregate([ { $group: { _id: "$Medication", count: { $sum: 1 } } }, { $sort: { count: -1 } }, { $limit: 1 } ])  
[ { _id: 'Penicillin', count: 2060 } ]  
Atlas atlas-beh2jl-shard-0 [primary] Project_2023> |
```

Give the name of recent 5 patients which are discharged and mention their billing amount?

```
Atlas atlas-beh2jl-shard-0 [primary] Project_2023> db.Healthcare.find( { "Discharge Date": { $ne: null } }, { "Name": 1, "Billing Amount": 1, "_id": 0 } ).sort({ "Discharge Date": -1 }).limit(5)  
[  
  { Name: 'Angela Thomas', 'Billing Amount': 43047.41106 },  
  { Name: 'Amanda Stein DVM', 'Billing Amount': 18904.7382 },  
  { Name: 'Margaret Hayes', 'Billing Amount': 45192.47844 },  
  { Name: 'Stephanie Simpson', 'Billing Amount': 35678.11313 },  
  { Name: 'Ruth Mason', 'Billing Amount': 45390.91588 }  
]  
Atlas atlas-beh2jl-shard-0 [primary] Project_2023> |
```

TOOLS FOR DATA VISUALIZATION :

MATPLOTLIB:

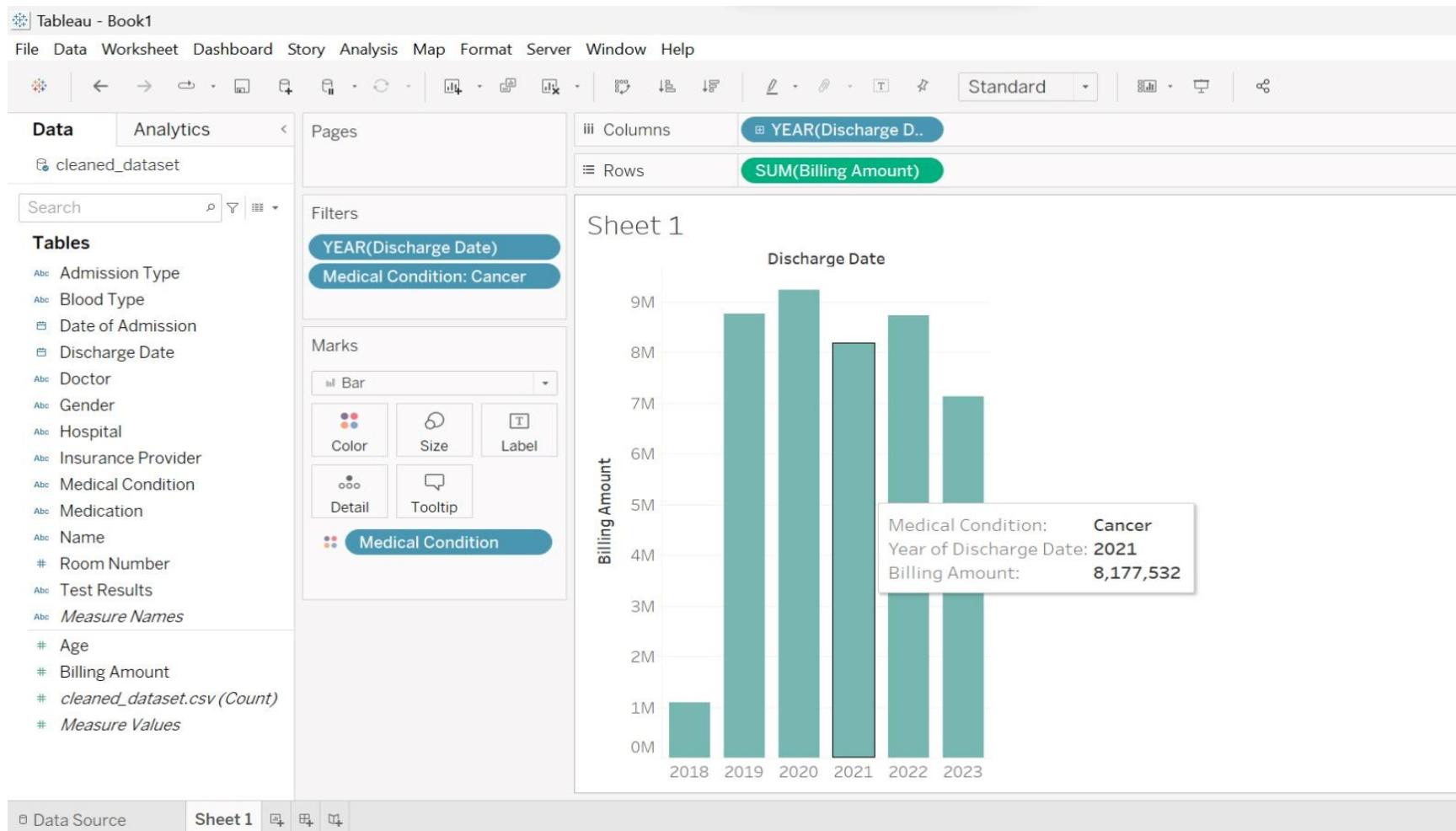
- Purpose: Matplotlib is a widely-used plotting library in Python that provides a variety of plotting functionalities for creating static, interactive, and publication-quality visualizations.
- Usage: It offers a wide range of plot types such as line plots, bar charts, histograms, scatter plots, 3D plots, etc.
- Matplotlib allows users to customize nearly every aspect of a plot, including colors, labels, markers, titles, axes, grids, and more. It can be used within Jupyter Notebooks, scripts, or any Python environment.
- Features: Flexible and granular control over plot elements. Integration with other Python libraries like Pandas for seamless data visualization.
- Matplotlib can be combined with other libraries like Seaborn for enhanced aesthetics.

TABLEAU:

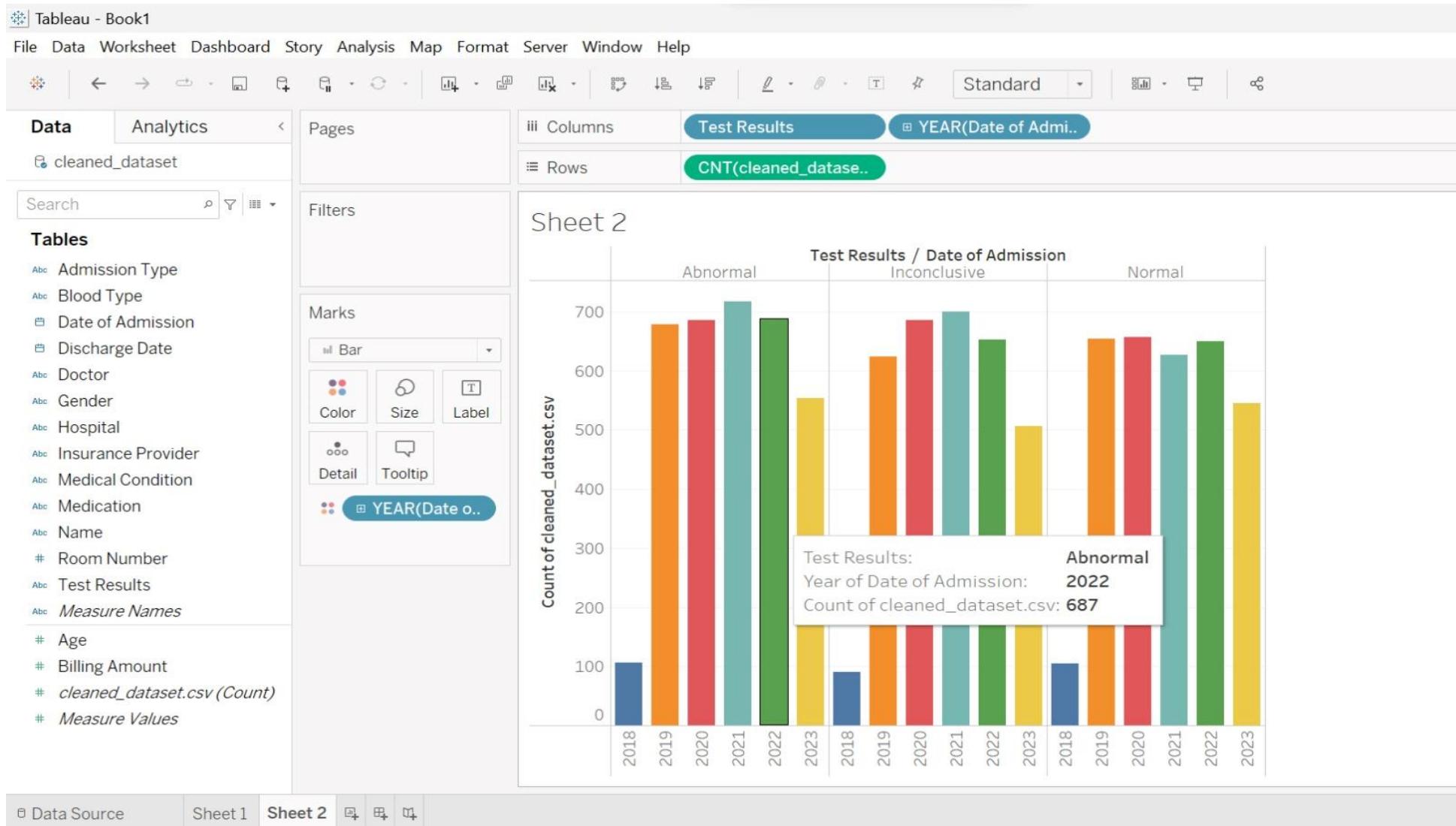
- Purpose: Tableau is a powerful and user-friendly data visualization tool that allows users to create interactive and shareable dashboards without requiring extensive programming knowledge.
- Usage: Offers a drag-and-drop interface for creating visualizations and dashboards. Connects to various data sources directly to create visualizations from real-time or static data.
- Tableau's strength lies in its ability to quickly generate interactive and visually appealing dashboards without coding.

DATA VISUALIZATION USING TABLEAU

COST OF MEDICATION ON TREATMENT OF CANCER FOR ALL YEAR



COUNT OF THE PATIENTS FOR THE TEST RESULT FOR DIFFERENT YEARS



COUNT OF PATIENTS OF CANCER FOR ALL YEARS

Tableau - Book1

File Data Worksheet Dashboard Story Analysis Map Format Server Window Help

Data Analytics <

cleaned_dataset

Search

Tables

- Admission Type
- Blood Type
- Date of Admission
- Discharge Date
- Doctor
- Gender
- Hospital
- Insurance Provider
- Medical Condition
- Medication
- Name
- Room Number
- Test Results
- Measure Names
- Age
- Billing Amount
- cleaned_dataset.csv (Count)
- Measure Values

Pages

iii Columns YEAR(Date of Adm..)

Rows Medical Condition CNT(cleaned_dataset.csv)

Filters YEAR(Date of Admis..)
Medical Condition: C..

Marks Bar

Color Size Label
Detail Tooltip
Medical Condit..

Sheet 3

Medical Condition: Cancer
Year of Date of Admission: 2023
Count of cleaned_dataset.csv: 275

Year of Date of Admission

Count of cleaned_dataset.csv

2018 2019 2020 2021 2022 2023

Sheet 1 Sheet 2 Sheet 3

COUNT OF MEDICINES CONSUMED BY MALES AND FEMALES IN DIFFERENT YEAR

Tableau - Book1

File Data Worksheet Dashboard Story Analysis Map Format Server Window Help

Data Analytics < Pages iii Columns **YEAR(Date of Adm..)** Gender

cleaned_dataset

Search

Tables

- Admission Type
- Blood Type
- Date of Admission
- Discharge Date
- Doctor
- Gender
- Hospital
- Insurance Provider
- Medical Condition
- Medication
- Name
- Room Number
- Test Results
- Measure Names
- Age
- Billing Amount
- cleaned_dataset.csv (Count)
- Measure Values

Filters **YEAR(Date of Admis..)**

Marks

- Square
- Color
- Size
- Label
- Detail
- Tooltip

CNT(cleaned_...) **CNT(cleaned_...)**

Sheet 4

Date of Admission / Gender
2023

Medication	Female	Male
Aspirin	145	156
Ibuprofen	149	145
Lipitor	162	176
Paracetamol	158	159
Penicillin	168	187

Data Source Sheet 1 Sheet 2 Sheet 3 Sheet 4

Tools for predictive analysis

Pandas:

- Purpose: Used for data manipulation and analysis.
- Usage in the Code: import pandas as pd - Importing the Pandas library.
- Role in the Code: Loading the dataset from a CSV file, manipulating dataframes, dropping unnecessary columns, and converting categorical variables to dummy/indicator variables.

Scikit-learn (sklearn):

- Purpose: A machine learning library providing various algorithms and tools for data mining and analysis.
- Usage in the Code: from sklearn.model_selection import train_test_split, from sklearn.tree import DecisionTreeClassifier, from sklearn.metrics import accuracy_score, classification_report:
- Role in the Code: Used for splitting the dataset into training and testing sets (train_test_split), implementing the Decision Tree Classifier (DecisionTreeClassifier), evaluating model accuracy and generating a classification report (accuracy_score, classification_report).

NumPy:

- Purpose: Provides support for arrays, matrices, and mathematical functions for numerical computing.
- Usage in the Code: NumPy is not explicitly used in this specific code snippet but is often used internally by Pandas and Scikit-learn for numerical computations.

Python:

- Purpose: General-purpose programming language used for scripting, data analysis, machine learning, and more.
- Usage in the Code: The entire code is written in Python, utilizing its syntax and libraries for data handling and machine learning.

PREDICTIVE ANALYSIS OF DATASET

1. IMPORT NECESSARY LIBRARIES, PANDAS, TRAIN_TEST_SPLIT, DECISIONTREE CLASSIFIER, ACCURACY SCORE, CLASSIFICATION REPORT
2. DROP UNNECESSARY COLUMNS
3. CONVERT CATERGORICAL VARIABLES TO DUMY INDICATORS
4. SPLIT DATASET INTO TRAINING AND TESTING SETS
5. CHOSE A PREDICTIVE MODEL (DECISION TREE)
6. TRAIN THE MODEL
7. MAKE PREDICTION ON THE TEST SET
8. EVALUATE THE MODEL
9. DISPLAY CLASSIFICATION REPORT

The screenshot shows two Jupyter Notebook sessions. The top session contains the full code for a Decision Tree classifier. The bottom session shows the execution results, including accuracy and a classification report.

```
# Import necessary libraries
import pandas as pd
dataset=pd.read_csv(r"C:\Users\shuba\Desktop\New folder\cleaned_dataset.csv")
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Assuming 'dataset' is your DataFrame
# Let's drop unnecessary columns for this example
features = dataset.drop(['Admission Type', 'Date of Admission', 'Discharge Date'], axis=1)
target = dataset['Admission Type']

# Convert categorical variables to dummy/indicator variables
features = pd.get_dummies(features)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

# Choose a predictive model (Decision Tree Classifier in this example)
model = DecisionTreeClassifier(random_state=42)

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Display classification report
print('Classification Report:\n', classification_report(y_test, y_pred))
```

[31] ... Accuracy: 0.34
Classification Report:
precision recall f1-score support
Elective 0.37 0.35 0.36 658
Emergency 0.33 0.33 0.33 671
Urgent 0.32 0.33 0.33 657

accuracy 0.34 0.34 0.34 1986
macro avg 0.34 0.34 0.34 1986
weighted avg 0.34 0.34 0.34 1986



D v

```
# Create a DataFrame to compare actual and predicted values  
predictions_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}) ←  
print(predictions_df.head(9927))
```

[32]

Python

```
...    Actual Predicted  
4407 Emergency Elective  
2769 Emergency Elective  
388  Emergency Urgent  
6586 Elective  Elective  
6769 Emergency Emergency  
...    ...    ...  
8565  Urgent   Urgent  
6391  Urgent Emergency  
6720  Urgent   Urgent  
7871  Urgent  Elective  
3903 Emergency Emergency
```

[1986 rows x 2 columns]

DATA FRAME CREATED TO COMPARE ACTUAL AND PREDICTED VALUES

The screenshot shows a Jupyter Notebook interface with two open files: 'NoSQL Project.ipynb' and 'project nosql - Copy (2).ipynb'. The current file is 'project nosql - Copy (2).ipynb'. The code in the notebook is as follows:

```
# Assuming 'dataset' is your DataFrame
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import pandas as pd

# Convert 'Date of Admission' to datetime format
dataset['Date of Admission'] = pd.to_datetime(dataset['Date of Admission'])

# Extract year from 'Date of Admission'
dataset['Year'] = dataset['Date of Admission'].dt.year

# Count the number of male and female patients for each year
gender_counts = dataset.groupby(['Year', 'Gender']).size().reset_index(name='Patient Count')

# Prepare the features and target for the regression model (predicting male count)
X_male = gender_counts[gender_counts['Gender'] == 'Male'][['Year']]
y_male = gender_counts[gender_counts['Gender'] == 'Male']['Patient Count']

# Split the data into training and testing sets for males
X_train_male, X_test_male, y_train_male, y_test_male = train_test_split(X_male, y_male, test_size=0.2, random_state=42)

# Train a linear regression model for males
model_male = LinearRegression()
model_male.fit(X_train_male, y_train_male)

# Prepare the features and target for the regression model (predicting female count)
X_female = gender_counts[gender_counts['Gender'] == 'Female'][['Year']]
y_female = gender_counts[gender_counts['Gender'] == 'Female']['Patient Count']

# Split the data into training and testing sets for females
```

Red arrows point to several lines of code, likely indicating steps in a checklist or highlighting specific operations:

- # Assuming 'dataset' is your DataFrame
- from sklearn.model_selection import train_test_split
- from sklearn.linear_model import LinearRegression
- import pandas as pd
- # Convert 'Date of Admission' to datetime format
- dataset['Date of Admission'] = pd.to_datetime(dataset['Date of Admission'])
- # Extract year from 'Date of Admission'
- dataset['Year'] = dataset['Date of Admission'].dt.year
- # Count the number of male and female patients for each year
- gender_counts = dataset.groupby(['Year', 'Gender']).size().reset_index(name='Patient Count')
- # Prepare the features and target for the regression model (predicting male count)
- X_male = gender_counts[gender_counts['Gender'] == 'Male'][['Year']]
- y_male = gender_counts[gender_counts['Gender'] == 'Male']['Patient Count']
- # Split the data into training and testing sets for males
- X_train_male, X_test_male, y_train_male, y_test_male = train_test_split(X_male, y_male, test_size=0.2, random_state=42)
- # Train a linear regression model for males
- model_male = LinearRegression()
- model_male.fit(X_train_male, y_train_male)
- # Prepare the features and target for the regression model (predicting female count)
- X_female = gender_counts[gender_counts['Gender'] == 'Female'][['Year']]
- y_female = gender_counts[gender_counts['Gender'] == 'Female']['Patient Count']
- # Split the data into training and testing sets for females

- CONVERT “DATE OF ADMISSION” TO DATETIME FORMAT
- EXTRACT YEAR FROM “DATE OF ADMISSION”
- COUNT THE NUMBER OF MALE AND FEMALE PATIENTS FOR EACH YEAR
- PREPARE THE FEATURES AND TARGET FOR THE REGRESSION MODEL (PREDICTING MALE COUNT)
- SPLIT THE DATA INTO TRAINING AND TESTING SETS FOR MALES
 - TRAIN THE LINEAR REGRESSION FOR MALES
 - PREPARE THE FEATURES AND TARGET THE REGRESSION MODEL (PREDICTING FEMALE COUNT)
- SPLIT THE DATA INTO TRAINING AND TESTING SETS FOR FEMALES

File Edit Selection View Go Run ... Search

C:\Users\DELL\Downloads>NoSQL Project.ipynb predictions_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

+ Code + Markdown | Run All Clear All Outputs Outline ... Select Kernel

```

# Split the data into training and testing sets for females
X_train_female, X_test_female, y_train_female, y_test_female = train_test_split(X_female, y_female, test_size=0.2, random_state=42) ←

# Train a linear regression model for females
model_female = LinearRegression()
model_female.fit(X_train_female, y_train_female) ←

# Predict the counts for the next 5 years
next_years = pd.DataFrame({'Year': [year for year in range(2024, 2029)]}) ←

# Predict the counts of male patients for the next 5 years
predicted_male_counts = model_male.predict(next_years) ←

# Predict the counts of female patients for the next 5 years
predicted_female_counts = model_female.predict(next_years) ←

# Combine the predictions into a DataFrame
predicted_counts = pd.DataFrame({
    'Year': next_years['Year'],
    'Predicted Male Count': predicted_male_counts,
    'Predicted Female Count': predicted_female_counts
}) ←

# Calculate the percentage of male and female patients
total_predicted_counts = predicted_counts['Predicted Male Count'] + predicted_counts['Predicted Female Count'] ←

predicted_counts['Percentage Male'] = (predicted_counts['Predicted Male Count'] / total_predicted_counts) * 100
predicted_counts['Percentage Female'] = (predicted_counts['Predicted Female Count'] / total_predicted_counts) * 100 ←

# Print the predicted counts and percentages
print(predicted_counts)

```

[33]

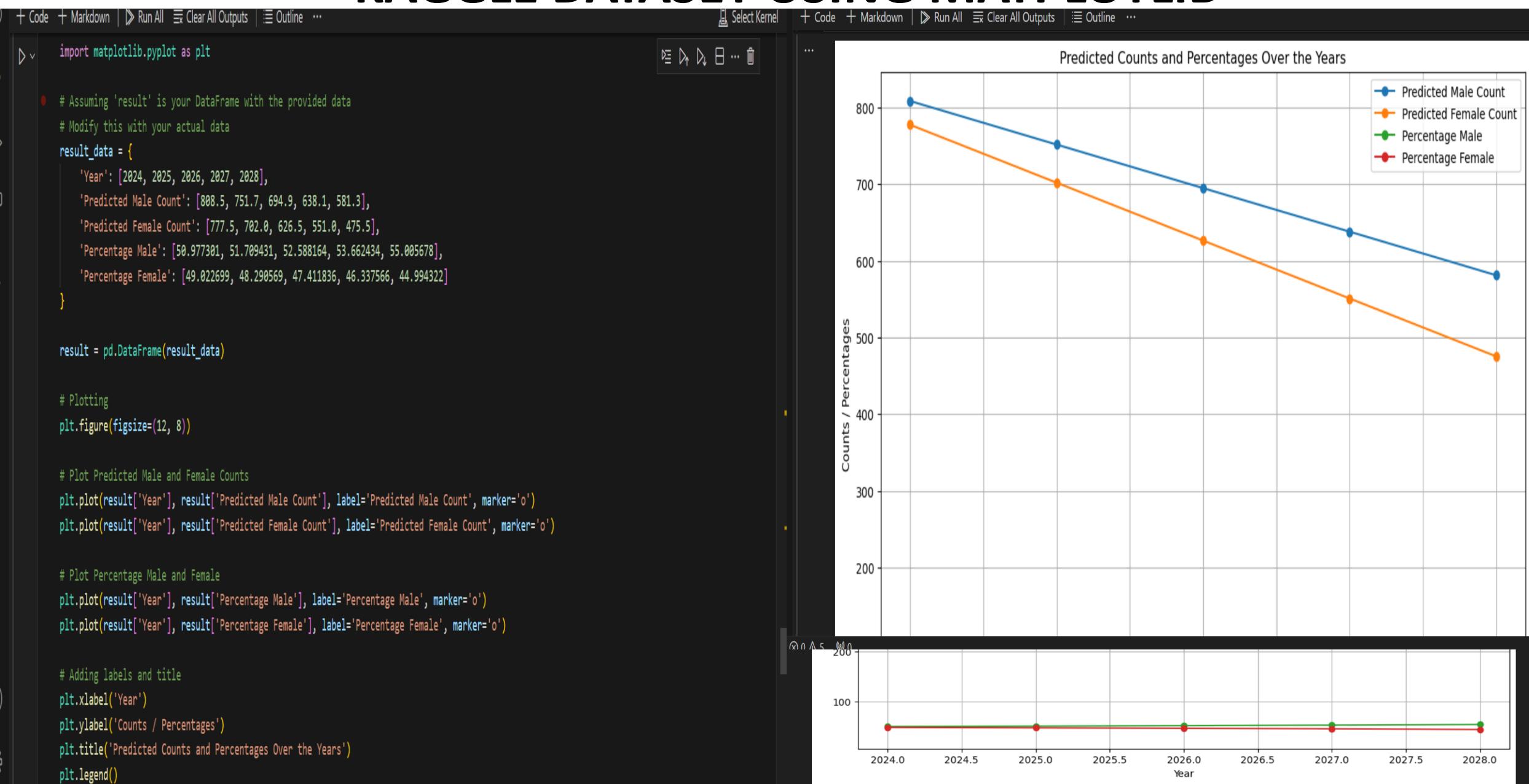
	Year	Predicted Male Count	Predicted Female Count	Percentage Male
0	2024	808.5	777.5	50.977301
1	2025	751.7	702.0	51.709431
2	2026	694.9	626.5	52.588164
3	2027	638.1	551.0	53.662434
4	2028	581.3	475.5	55.005678

	Percentage Female
0	49.022699
1	48.290569
2	47.411836
3	46.337566
4	44.994322

PREDICTING DATA FOR NEXT 5 YEARS

- TRAIN A LINEAR REGRESSION FOR FEMALES
- PREDICT THE COUNTS FOR THE NEXT 5 YEARS
- PREDICT THE COUNTS OF MALE PATIENTS FOR THE NEXT 5 YEARS
 - PREDICT THE COUNTS OF FEMALE PATIENTS FOR THE NEXT 5 YEARS
- COMBINE THE PREDICTIONS INTO A DATAFRAME
- CALCULATE THE PERCENTAGE OF MALE AND FEMALES PATIENTS

DATA VISUALIZATION FOR PREDICTIVE ANALYSIS FOR HEALTHCARE KAGGLE DATASET USING MATPLOTLIB



WHY PYSPARK ?

PySpark is used for queries because it enables distributed processing across clusters, ensuring efficiency with large datasets. Its scalability, integration with NoSQL databases, Python compatibility, and unified data processing platform make it a versatile choice. Additionally, the active Apache Spark community provides valuable support for optimizing and enhancing queries.

USING PYSPARK TO QUERY AND ANALYSE DATASET

- IMPORT THE LIBRARY “SPARKSESSION”
- CREATE A SPARKSESSION
- LOAD THE DATASET INTO PYSPARK
- SHOW THE DETAILS OF THE DATASET

```
# Inside the PySpark shell
from pyspark.sql import SparkSession ←

# Create a Spark session
spark = SparkSession.builder.appName("CSVFileLoader").getOrCreate() ←

# Specify the path to the CSV file
csv_file_path = "C:/Users/91628/Desktop/NoSQL PROJECT/cleaned_dataset.csv" ←

# Read the CSV file into a DataFrame
csv_data_frame = spark.read.csv(csv_file_path, header=True, inferSchema=True) ←

# Perform operations on the DataFrame as needed
csv_data_frame.show() ←
```

The screenshot shows a Jupyter Notebook interface with a terminal tab active. The terminal output displays the command `python -u "c:/Users/91628/Desktop/NoSQL PROJECT/Project.py"` being run. The script sets the log level to `"WARN"`. It then reads a CSV file named `cleaned_dataset.csv` located at `C:/Users/91628/Desktop/NoSQL PROJECT`. The resulting DataFrame is shown with the following schema:

Name	Age	Gender	Blood Type	Medical Condition	Date of Admission	Doctor	Hospital	Insurance Provider	Billing Amount	Room Number	Admission Type	Discharge Date	Medication	Test Results
Tiffany Ramirez	81	Female	O-	Diabetes	2022-11-17	Patrick Parker	Wallace-Hamilton	Medicare	37490.98336	146	Elective	2022-12-01	Aspirin	Inconclusive
Ruben Burns	35	Male	O+	Asthma	2023-06-01	Diane Jackson	Burke, Griffin and...	UnitedHealthcare	47304.06485	404	Emergency	2023-06-15	Lipitor	Normal
Chad Byrd	61	Male	B-	Obesity	2019-01-09	Paul Baker	Walton LLC	Medicare	36874.897	292	Emergency	2019-02-08	Lipitor	Normal
Antonio Frederick	49	Male	B-	Asthma	2020-05-02	Brian Chandler	Garcia Ltd	Medicare	23303.32209	480	Urgent	2020-05-03	Penicillin	Abnormal
Mrs. Brandy Flowers	51	Male	O-	Arthritis	2021-07-09	Dustin Griffin	Jones, Brown and...	UnitedHealthcare	18086.34418	477	Urgent	2021-08-02	Paracetamol	Normal
Patrick Parker	41	Male	AB+	Arthritis	2020-08-20	Robin Green	Boyd PLC	Aetna	22522.36338	180	Urgent	2020-08-23	Aspirin	Abnormal
Charles Horton	82	Male	AB+	Hypertension	2021-03-22	Patricia Bishop	Wheeler, Bryant and...	Cigna	39593.43576	161	Urgent	2021-04-15	Lipitor	Abnormal
Patty Norman	55	Female	O-	Arthritis	2019-05-16	Brian Kennedy	Brown Inc	Blue Cross	13546.81725	384	Elective	2019-06-02	Aspirin	Normal
Ryan Hayes	33	Male	A+	Diabetes	2020-12-17	Kristin Dunn	Smith, Edwards and...	Aetna	24903.03727	215	Elective	2020-12-22	Aspirin	Abnormal
Amy Roberts	45	Male	B-	Cancer	2021-04-13	Anthony Roberts	Little-Spencer	Aetna	40325.07139	306	Emergency	2021-05-11	Penicillin	Abnormal
Mrs. Caroline Far...	23	Female	O-	Hypertension	2019-06-09	William Miller	Rose Inc	Medicare	6185.90353	126	Emergency	2019-06-26	Paracetamol	Inconclusive
Christina Williams	85	Female	A+	Diabetes	2021-11-29	Laura Roberts	Malone, Thompson and...	Aetna	4835.94565	444	Elective	2021-12-14	Aspirin	Inconclusive
William Page	72	Female	A+	Diabetes	2021-07-29	James Carney	Richardson-Powell	Cigna	13669.37774	492	Elective	2021-08-14	Aspirin	Normal
Michael Bradshaw	65	Female	AB+	Cancer	2021-06-05	Katherine Lowe	Castaneda-Hardy	Cigna	10342.83612	120	Emergency	2021-06-25	Ibuprofen	Inconclusive
Brian Dorsey	32	Female	O+	Arthritis	2021-08-07	Curtis Smith	Burch-White	Aetna	27174.94291	492	Emergency	2021-08-14	Aspirin	Inconclusive

QUERY THE AVERAGE BILLING AMOUNT FOR EACH ADMISSION TYPE

- INITIALIZE A “SPARKSESSION”
- LOAD THE DATASET INTO PYSPARK
- REGISTER THE DATAFRAME AS A TEMPORARY SQL TABLE
- SHOW OUPUT OF QUERY

```
from pyspark.sql import SparkSession

# Initialize a Spark session
spark = SparkSession.builder.getOrCreate() ←

# Assuming 'healthcare_data.csv' is your CSV file
csv_file_path = "C:/Users/91628/Desktop/NoSQL PROJECT/cleaned_dataset.csv" ←

# Read CSV data into a PySpark DataFrame
csv_data_frame = spark.read.csv(csv_file_path, header=True, inferSchema=True) ←

# Register the DataFrame as a temporary SQL table
csv_data_frame.createOrReplaceTempView("healthcare_table") ←

# SQL Query to find the average billing amount for each admission type
query = """
    SELECT `Admission Type`, AVG(`Billing Amount`) AS avg_billing_amount
    FROM healthcare_table
    GROUP BY `Admission Type` ←
.... ←

result = spark.sql(query)
result.show() ←
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Setting default log level to "WARN".

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

Admission Type	avg_billing_amount
Elective	25878.63010234268
Emergency	24703.655081673987
Urgent	25987.10712893025

PS C:\Users\91628\Desktop\NoSQL PROJECT> SUCCESS: The process with PID 25920 (child process of PID 27000) has been terminated.
SUCCESS: The process with PID 27000 (child process of PID 23936) has been terminated.
SUCCESS: The process with PID 23936 (child process of PID 12716) has been terminated.

QUERY TO ANALYZE THE COST PAID BY PATIENTS FOR DIFFERENT AILMENTS

- INITIALIZE A “SPARKSESSION”
- LOAD THE DATASET INTO PYSPARK
- REGISTER THE DATAFRAME AS A TEMPORARY SQL TABLE
- SHOW OUPUT OF QUERY

```
from pyspark.sql import SparkSession

# Initialize a Spark session
spark = SparkSession.builder.getOrCreate() ←

# Assuming 'healthcare_data.csv' is your CSV file
csv_file_path = "C:/Users/91628/Desktop/NosQL PROJECT/cleaned_dataset.csv" ←

# Read CSV data into a PySpark DataFrame
csv_data_frame = spark.read.csv(csv_file_path, header=True, inferSchema=True) ←

# Register the DataFrame as a temporary SQL table
csv_data_frame.createOrReplaceTempView("healthcare_table") ←

query="""
SELECT
    `Medical Condition`,
    COUNT(*) AS total_patients,
    AVG(`Billing Amount`) AS avg_billing_amount,
    MAX(`Billing Amount`) AS max_billing_amount,
    MIN(`Billing Amount`) AS min_billing_amount,
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY `Billing Amount`) AS median_billing_amount
FROM
    healthcare_table
WHERE
    `Medical Condition` IS NOT NULL -- Filter out rows without medical condition information
GROUP BY
    `Medical Condition`
ORDER BY
    avg_billing_amount DESC; -- order by average billing amount in descending order
"""

Result = spark.sql(query)
Result.show() ←
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\91628\Desktop\NosQL PROJECT> python -u "c:\Users\91628\Desktop\NosQL PROJECT\Project.py"
Setting default log level to "WARN".

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

Medical Condition	total_patients	avg_billing_amount	max_billing_amount	min_billing_amount	median_billing_amount
Diabetes	1608	26068.46606168724	49954.96833	1071.456127	26141.25889
Obesity	1621	25739.1970177179	49974.16046	1000.180837	25369.47637
Cancer	1690	25512.552505905907	49994.98474	1020.33779	25474.1011
Asthma	1695	25416.424132660733	49974.29914	1032.263087	25084.61065
Arthritis	1636	25234.245521301942	49985.97307	1009.417327	24760.97501000002
Hypertension	1676	25175.053821493417	49995.90228	1084.422303	24920.455175

PS C:\Users\91628\Desktop\NosQL PROJECT> SUCCESS: The process with PID 10708 (child process of PID 27160) has been terminated.
SUCCESS: The process with PID 27160 (child process of PID 9756) has been terminated.
SUCCESS: The process with PID 9756 (child process of PID 29028) has been terminated.

DATA VISUALIZATION USING PYSPARK

CALCULATE THE AGE DISTRIBUTION

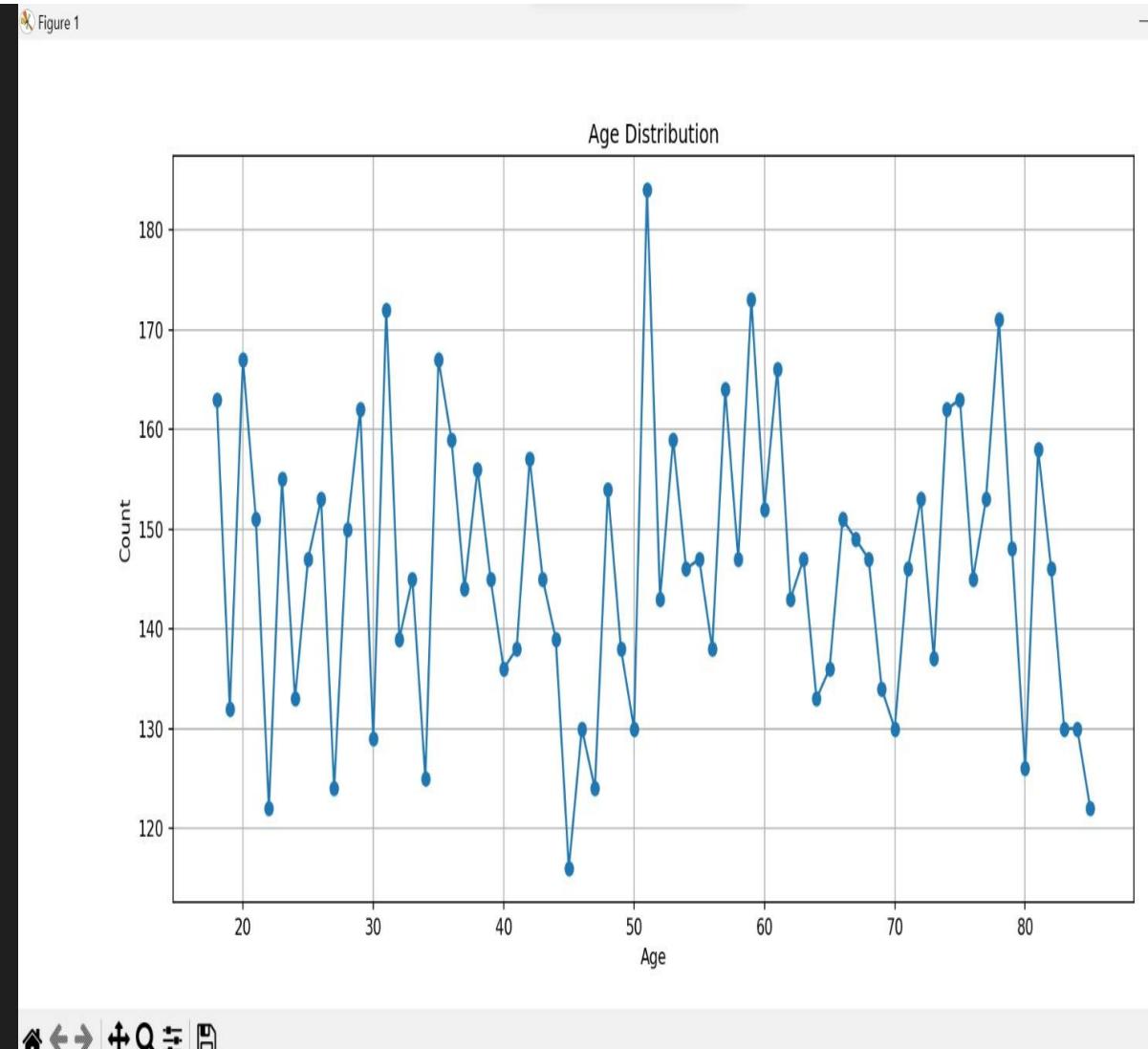
```
import matplotlib.pyplot as plt
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

# Calculate the age distribution
age_distribution = csv_data_frame.groupBy("age").count().orderBy("age")

# Show the results
age_distribution.show(truncate=False)

# Convert PySpark DataFrame to Pandas DataFrame for plotting
age_distribution_pandas = age_distribution.toPandas()

# Plot the line graph
plt.figure(figsize=(10, 6))
plt.plot(age_distribution_pandas['age'], age_distribution_pandas['count'], marker='o')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Count')
plt.grid(True)
plt.show()
```



DATA INSIGHTS FOR FUTURE

From predictive analysis, it can be seen that count of males and females is declining towards 2028. Also it is predicted that in next 5 year percent of male patients will be more as compared to female patients.

INSIGHTS ON HYPOTHESIS

After analysis we concluded that we have to reject our hypothesis discussed earlier as data shows that count of cancer patients has decreased over years as well as expenses had also declined.

CONCLUSION

- People dealing with hypertension (high blood pressure), end up spending the most on hospital bills, while those with obesity end up spending the least.
- Urgent medical cases turn out to be the most expensive on average, while surprisingly, emergency situations results in the lowest average cost of patients.
- Furthermore, our research extends beyond expenses, revealing that penicillin medication is widely prescribed by doctors.
- Notably, there is a correlation between individuals with AB- blood group exhibit the highest number of medical condition, as inferred from our test results.



THANK YOU

FOR LISTENING AND YOUR PATIENCE