Definition of Data Analysis:

Data analysis is the process of inspecting, cleaning, transforming, and modeling data with the goal of discovering useful information, drawing conclusions, and making decisions based on that information. It involves using various statistical and computational methods to explore, summarize, and visualize large and complex data sets to uncover patterns, relationships, and insights. Data analysis can be applied to many different fields, including business, science, medicine, social sciences, and more.

It is a crucial step in the process of making data-driven decisions, as it allows analysts to identify trends, outliers, and other important factors that can inform business strategies, scientific hypotheses, or policy decisions. The process of data analysis typically involves several stages, including data collection, data cleaning and pre-processing, exploratory data analysis, statistical modeling, and interpretation of results. Through these stages, data analysts can gain a deeper understanding of the data they are working with and use that understanding to make informed decisions.

Cause to use data analysis:

There are many different causes for using data analysis, as it has many different applications across a wide range of industries and fields. Here are some of the main causes for using data analysis:

1. Decision-making: Data analysis can help inform decision-making by providing insights and information that can help individuals and organizations make more informed choices.

2. Improved efficiency: Data analysis can help improve efficiency by identifying inefficiencies and areas for improvement within an organization or process.

3. Identification of opportunities: Data analysis can help identify new opportunities by identifying patterns and trends that may not be immediately apparent.

4. Risk management: Data analysis can help manage risk by identifying potential risks and predicting outcomes.

5. Research and development: Data analysis can help drive innovation by providing insights and information that can inform the development of new products, services, or technologies.

Overall, data analysis is a powerful tool that can provide valuable insights and information for individuals and organizations across a wide range of industries and fields. By analyzing data from various sources, data analysis can help inform decision-making, improve efficiency, identify opportunities, manage risk, and drive innovation.

In our application:

Causes of use data analysis:

As we have data about:

Doctors: Name, Phone, Address, Age.

Patients: National ID, Name, Phone, Address, Age, patients ID.

Patients: National ID, Name, Phone, Age, Address, Gender, his doctor id

Diseases: If the patient injured by diabetes, hypertension, cholesterol

Medical Result: Xray Result, Xray Date for each user.

Survey Result: Answers of survey questions in the application and it's date.

So, we decided to make use of this data that we have, in:

1. Assist in diagnosing patients:

   relying on my data and identify patterns within it. This will be after collecting many records and having many patients. Therefore, I will be able to assist in diagnosing patients based on the symptoms they exhibit.

   The ability to diagnose patients based on their symptoms is a crucial skill for medical professionals. By using data to inform these diagnoses, healthcare providers can ensure that patients receive the most appropriate treatments and care. This can be especially important for patients with complex or rare conditions, who may require specialized knowledge and expertise to diagnose and treat effectively.

2. In marketing:

- We want people to use our application, and we can achieve this through data analysis. By analyzing user data, we can identify areas where users may be experiencing difficulties or friction points and take steps to improve the user experience. We can also use data analysis to identify features or content that are particularly popular or engaging and highlight these to encourage further usage. Additionally, analyzing user data can help us identify patterns and trends in user behavior, which can inform our marketing and outreach efforts to attract more users. Overall, data analysis is a powerful tool for increasing user adoption and engagement and can help us create a more successful and widely used application.
- In order to be able to locate the places where patients and doctors use the application, and therefore I can do marketing and expand more in the number of users.

3. Generate Ratios:

   I will be able to use the ratios generated from data analysis and incorporate them into our application. These ratios can provide valuable insights into various aspects of the application, such as user behavior, engagement, and retention. For example, by analyzing the ratio of new users to returning users, we can gain insights into user loyalty and identify areas where we may need to improve the user experience. Similarly, by analyzing the ratio of clicks to conversions, we can identify areas where users may be experiencing difficulties or friction points and take steps to improve the user experience. Overall, incorporating data analysis ratios into our application can help us make data-driven decisions, improve the user experience, and ultimately drive business growth.

4. Display the charts in a dashboard: by creating a dashboard that displays the charts and insights from your data analysis. This can be a great way to provide a quick overview of the key findings in the data.
5. Use the charts to inform your application's user interface: You can use the insights from your data analysis to inform the design of your application's user interface. For example, if you discover that a particular feature of your application is not being used as much as you expected, you can modify the user interface to make it more prominent.

Using Python in Our Analysis:

Python is a popular programming language for data analysis for several reasons:

1. Ease of use: Python is a relatively easy language to learn and use. Its syntax is simple and easy to read, making it accessible to a wide range of users.

2. Large and active community: Python has a large and active community of developers who contribute to the language and develop tools and libraries specifically for data analysis. This community provides a wealth of resources and support for users of all skill levels.

3. Versatility: Python is a versatile language that can be used for a wide range of applications, including data analysis, machine learning, web development, and more. This makes it a useful language to learn for individuals and organizations looking to develop a variety of skills and applications.

4. Interoperability: Python can be easily integrated with other tools and languages commonly used in data analysis, such as SQL, R, and Excel. This allows users to leverage the strengths of multiple tools and languages to achieve their data analysis goals.

5. Powerful data analysis libraries: Python has several powerful data analysis libraries, such as Pandas, NumPy, and Matplotlib, that provide a wide range of functions and tools for data manipulation, cleaning, visualization, and statistical analysis.

Overall, Python is a popular language for data analysis due to its ease of use, large and active community, versatility, interoperability, and powerful data analysis libraries. These factors make it a useful language for individuals and organizations looking to perform data analysis tasks efficiently and effectively.

Data analysis typically involves several stages or processes, which can be summarized as follows:
1. Data collection:
2. Data cleaning and preprocessing
3. Data exploration
4. Data analysis
5. Data visualization and reporting
Overall, data analysis involves several stages or processes, including data collection, cleaning and preprocessing, data exploration, data analysis, and data visualization and reporting. By following these processes, data analysts can gain valuable insights and make informed decisions based on the data.

1. Data collection

The first step in data analysis is to collect the data that will be analyzed. This may involve gathering data from various sources, such as databases, sensors, or surveys.

Through the data collected from the application, when the patient enters the application and collects his data, it is stored in the data base, and when the image is uploaded, the user fills out a survey containing some questions related to the symptoms he has, as well as the doctor when he enters the application, his data is preserved, and with this The data used in the analysis are collected.

And we prepare the data for the application that we have stored. and that is through:

```python
import psycopg2
import pandas as pd

# Define the database connection parameters
host = "database-1.cudlkdpouzkp.eu-north-1.rds.amazonaws.com"
port = "5432"
dbname = "Pneumonia"
user = "postgres"
password = "123456789"

# Connect to the database
conn = psycopg2.connect(host = host, port = port, dbname = dbname, user = user, password = password)

doctor_query = "SELECT * FROM doctor"
patient_query = "SELECT * FROM patient"
diseases_query = "SELECT * FROM diseases"
imagelabel_query = "SELECT * FROM imagelabel"
medicalresult_query = "SELECT * FROM medicalresult"
survay_result_query = "SELECT * FROM survey_results"

# Execute the queries and store the results in pandas dataframes
doctor_df = pd.read_sql(doctor_query, conn)
patient_df = pd.read_sql(patient_query, conn)
diseases_df = pd.read_sql(diseases_query, conn)
imagelabel_df = pd.read_sql(imagelabel_query, conn)
medicalresult_df = pd.read_sql(medicalresult_query, conn)
survay_result_df = pd.read_sql(survay_result_query, conn)
```

[1] ✓ 11.9s Python

2. Data cleaning and preprocessing

Once the data is collected, it must be cleaned and preprocessed to ensure that it is accurate, complete, and in a format that can be analyzed. This may involve removing outliers or missing values, converting data into appropriate formats, and normalizing data.

1. Check completeness of data and handling any missing values

Missing values

```python
# Check if current data contains any missing values
doctor_df.isnull().any()
```

[12] ✓ 0.0s

```
National_ID    False
Name           False
Phone          False
Address        False
Age            False
dtype: bool
```

But, what if the data will contain missing in the future?

Handle of this is:

1) In Doctor table:

Fillin Missing values of Address

- Fill null in Address with the most frequent address in the data
- Fill null in Age with the Average address in the data

```python
# compute the most frequent value in column 'Address'
most_frequent_value = doctor_df['Address'].mode()[0]

# fill missing values in column 'Address' with the most frequent value
doctor_df['Address'] = doctor_df['Address'].fillna(most_frequent_value)

# print the updated DataFrame
doctor_df['Address'].head(7)
```

[13] ✓ 0.0s

```
0      Mansoura
1         Cairo
2        Assuit
3         Luxor
4         Dahab
5    Alexandria
6       Mallawi
Name: Address, dtype: object
```

```python
# compute the mean value in column 'Age'
mean_doctor_age = doctor_df['Age'].mean()

# fill missing values in column 'Age' with the Average
doctor_df['Age'] = doctor_df['Age'].fillna(mean_doctor_age)

# print the updated DataFrame
doctor_df['Age'].head(7)
```

[14] ✓ 0.0s

```
0    30
1    31
2    45
3    34
4    50
5    35
6    28
Name: Age, dtype: int64
```

Filling missing values with most frequent values in column

1. Diabetic

```python
# compute the most frequent value in column 'Diabetic'
diabetic_most_freq = diseases_df['Diabetic'].mode()[0]

# fill missing values in column 'Diabetic' with the most frequent value
diseases_df['Diabetic'] = diseases_df['Diabetic'].fillna(diabetic_most_freq)

# print the updated DataFrame
print(diseases_df['Diabetic'].head(7))
```

0.0s

```
0     True
1    False
2     True
3    False
4     True
5     True
6    False
Name: Diabetic, dtype: bool
```

2. Hypertension

```python
# compute the most frequent value in column 'Hypertension'
Hypertension_most_freq = diseases_df['Hypertension'].mode()[0]

# fill missing values in column 'Hypertension' with the most frequent value
diseases_df['Hypertension'] = diseases_df['Hypertension'].fillna(Hypertension_most_freq)

# print the updated DataFrame
print(diseases_df['Hypertension'].head(7))
```

0.0s

```
0    False
1    False
2     True
3    False
4    False
5     True
6    False
Name: Hypertension, dtype: bool
```

3. Cholesterol

```python
# compute the most frequent value in column 'Cholesterol'
Cholesterol_most_freq = diseases_df['Cholesterol'].mode()[0]

# fill missing values in column 'Cholesterol' with the most frequent value
diseases_df['Cholesterol'] = diseases_df['Cholesterol'].fillna(Cholesterol_most_freq)

# print the updated DataFrame
print(diseases_df['Cholesterol'].head(7))
```

0.0s

```
0     True
1    False
2     True
3     True
4    False
5    False
6     True
Name: Cholesterol, dtype: bool
```

Missing values in Medical Result:

```python
medicalresult_df = medicalresult_df.dropna(subset = ['xray_result'])
medicalresult_df.head(5)
```

✓ 0.0s

	id	xray_result	xray_date	p_nid
0	5420	0	2022-09-10	20320154852369
1	5421	1	2020-12-04	30201548596352
2	5422	0	2021-04-05	30245849635712
3	5423	1	2022-06-30	42589635715284
4	5424	1	2023-05-01	20301548967415

Filling missing values in each questions' Survey column with the most frequent value

```python
for col in survay_result_df.columns:
    # check if column contains boolean values
    if survay_result_df[col].dtype == bool:
        # compute the most frequent value
        most_frequent_value = survay_result_df[col].mode()[0]
        # fill missing values with the most frequent value
        survay_result_df[col] = survay_result_df[col].fillna(most_frequent_value)

# print the updated DataFrame
print(survay_result_df.head(7))
```

[164] ✓ 0.2s

```
   survey_id  q_one  q_two  q_three  q_four  q_five  q_six  q_seven  q_eight  \
0       4520      1      0        1       1       1      0        0        0
1       4521      0      1        0       1       1      1        1        0
2       4522      0      1        1       0       0      1        0        0
3       4523      1      1        1       0       1      0        0        0
4       4524      0      1        0       0       0      1        1        1
5       4525      0      1        1       0       1      0        1        1
6       4526      0      0        1       0       1      0        1        1

   q_nine  q_ten survey_date Patient_National_ID
0       1      1  2022-04-20      20320154852369
1       0      0  2016-07-24      30201548596352
2       1      1  2022-06-22      30245849635712
3       1      0  2022-02-22      42589635715284
4       1      0  2018-12-12      20301548967415
5       0      0  2023-03-08      25352015984520
6       0      0  2019-11-11      40475302156985
```

2) Datatypes of columns in each table, as done in:

```
# Convert Age data type into numeric
patient_df.age = pd.to_numeric(patient_df.age)
```

[30] ✓ 0.0s

Datatypes of Diseases columns

```
# After processing
diseases_df.dtypes
```

[44] ✓ 0.0s

```
d_id            int64
diabetes        int64
hypertention    int64
cholesterol     int64
p_nid          object
dtype: object
```

```
# Convert each data type into suitable format
diseases_df.diabetes = diseases_df.diabetes.astype(bool)
diseases_df.hypertention = diseases_df.hypertention.astype(bool)
diseases_df.cholesterol = diseases_df.cholesterol.astype(bool)
```

[45] ✓ 0.0s

```
# Before proceeding
diseases_df.dtypes
```

[46] ✓ 0.0s

```
d_id            int64
diabetes         bool
hypertention     bool
cholesterol      bool
p_nid          object
dtype: object
```

3) converting data into appropriate formats:

```python
doctor_df.Name = doctor_df.Name.str.title()
doctor_df.Name.head(7)
```

[114] ✓ 0.0s

```
0          Karma Ali
1         Muhra Malek
2        Hamza Khaled
3       Ahmed Mahmoud
4        Nour Mohamed
5          Maleka Ali
6            Zain Ali
Name: Name, dtype: object
```

Age for doctors

- 1: Older adults (65 and older)
- 2: Adults (18 years or older)

```python
def age_stage(age):
    if age < 65:
        return 'Adult'
    else:
        return 'Older'
```

[18] ✓ 0.0s

```python
doctor_df["Age_Stage"] = doctor_df["Age"].apply(age_stage)
doctor_df.head(7)
```

[115] ✓ 0.0s

	National_ID	Name	Phone	Address	Age	Age_Stage
0	20301548520137	Karma Ali	01236548520	Mansoura	30	Adult
1	20315059632104	Muhra Malek	01152480236	Cairo	31	Adult
2	20102365215422	Hamza Khaled	01025879631	Assuit	45	Adult
3	01025485236541	Ahmed Mahmoud	01002542023	Luxor	34	Adult
4	20103654852012	Nour Mohamed	01022554102	Dahab	50	Adult
5	14203025698410	Maleka Ali	0112548001	Alexandria	35	Adult
6	20152563201253	Zain Ali	01024586351	Mallawi	28	Adult

4) Removing Outliers, improve the accuracy and reliability of your data analysis. This can be done using various statistical techniques, such as Z-score analysis, interquartile range (IQR) analysis, or box plots. These techniques help to identify data points that are significantly different from the rest of the dataset and remove them from the analysis.

- Z-score: used to identify and remove outliers from the dataset before performing any further analysis.

Normalization

```
# Z score
doctor_df['Age_Normalization'] = np.abs(stats.zscore(doctor_df['Age'])).round(2)
doctor_df['Age_Normalization'].head(7)
```

[152] ✓ 0.2s

```
0    1.01
1    0.92
2    0.33
3    0.66
4    0.77
5    0.57
6    1.19
Name: Age_Normalization, dtype: float64
```

```
z_doctor_df = doctor_df['Age_Normalization']
print(np.where(z_doctor_df > 2))
```

[133] ✓ 0.0s

Remove Outlier Values in doctor_df table

```
# IQR
# Calculate the upper and lower limits
Q1 = doctor_df['Age'].quantile(0.25)
Q3 = doctor_df['Age'].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5*IQR
upper = Q3 + 1.5*IQR

# Create arrays of Boolean values indicating the outlier rows
upper_array = np.where(doctor_df['Age'] >= upper)[0]
lower_array = np.where(doctor_df['Age'] <= lower)[0]

# Removing the outliers
doctor_df.drop(index=upper_array, inplace=True)
doctor_df.drop(index=lower_array, inplace=True)
```

✓ 0.0s

doctor_df after removing outlier

```python
# Plot Age Distribution without outliers in Age Column
plt.figure(figsize=(16,8))
# add title to the entire figure
plt.suptitle('Age Before Removing Outliers', fontsize=16)
plt.subplot(2,2,1)
sns.distplot(doctor_df['Age'])
plt.subplot(2,2,2)
sns.boxplot(doctor_df['Age'])
plt.show()
```

[137] ✓ 0.2s

```
c:\Users\sohil\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use e
flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
c:\Users\sohil\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `
keyword will result in an error or misinterpretation.
  warnings.warn(
```

-

```python
# Map 'female' to 1 and 'male' to 0
patient_df['gender'] = patient_df['gender'].map({'F': 1,
                                                 'M': 0})
```

[31] ✓ 0.0s

Patient Age

- 1: Older adults (65 and older)
- 2: Adults (18 years or older)
- 3: Adolescents (13 years through 17 years. They may also be referred to as teenagers depending on the context).

```python
def age_stage_Patient(age):
    if age < 18:
        return 'Adolescent'
    elif age > 18 & age < 65:
        return 'Adult'
    else:
        return 'Older'
```

[150] ✓ 0.1s

```python
patient_df["Age_Stage"] = patient_df["Age"].apply(age_stage_Patient)
patient_df.head(7)
```

[151] ✓ 0.0s

	National_ID	Name	Phone	Age	Address	Doctor_Phone	Gender	Follower_Phone	doctor_nid	Age_Normalization_Patient	Age_Stage
0	30111132400351	Momen Khaled	01099473695	25	Minia	01128922921	0	01111111111	None	0.89	Adult
1	20320154852369	Ali Bakr	01025486358	30	Mallawi	01258963251	0	01158263541	20152563201253	0.55	Adult
2	30201548596352	Abdelrahman Mohamed	01045863251	20	Cairo	01258963251	0	01024589637	20301548520137	1.24	Adult
3	30245849635712	Karem Assem	01235896541	22	Minia	01258963251	0	01253658477	20315059632104	1.10	Adult
4	42589635715284	Hazem Khaled	01028963547	26	Mansora	01258963251	0	01258963457	20102365215422	0.82	Adult
5	20301548967415	Mustafa Yahia	01036587460	28	Assuit	01258963251	0	01236584754	01025485236541	0.68	Adult
6	25352015984520	Gmela Sayed	01025879634	22	Sohag	01258963251	1	01254589632	20103654852012	1.10	Adult

Cleaning data before analysis is important for several reasons:

1. Accuracy of analysis.

2. Consistency of analysis.

3. Efficiency of analysis.

4. Improved data quality.

5. Better decision-making.

Overall, cleaning data before analysis is an important step in the data analysis process. It helps ensure the accuracy, consistency, and efficiency of the analysis, and can lead to improved data quality and better decision-making.

3. **Data exploration** the next step is to explore the data to gain insights and identify patterns or trends. This may involve visualizing data using graphs or charts, calculating summary statistics, or applying machine learning algorithms to identify correlations or clusters.

```
# Doctor table before preprocessing and cleaning
doctor_df.head(7)
```

[138] ✓ 0.0s

	National_ID	Name	Phone	Address	Age	Age_Normalization	Age_Stage
0	20301548520137	Karma Ali	01236548520	Mansoura	30	1.01	Adult
1	20315059632104	Muhra Malek	01152480236	Cairo	31	0.92	Adult
2	20102365215422	Hamza Khaled	01025879631	Assuit	45	0.33	Adult
3	01025485236541	Ahmed Mahmoud	01002542023	Luxor	34	0.66	Adult
4	20103654852012	Nour Mohamed	01022554102	Dahab	50	0.77	Adult
5	14203025698410	Maleka Ali	0112548001	Alexandria	35	0.57	Adult
6	20152563201253	Zain Ali	01024586351	Mallawi	28	1.19	Adult

```
# Patient table before processing and cleaning
patient_df.head(7)
```

[153] ✓ 0.0s

	National_ID	Name	Phone	Age	Address	Doctor_Phone	Gender	Follower_Phone	doctor_nid	Age_Normalization_Patient	Age_Stage
0	30111132400351	Momen Khaled	01099473695	25	Minia	01128922921	0	01111111111	None	0.89	Adult
1	20320154852369	Ali Bakr	01025486358	30	Mallawi	01258963251	0	01158263541	20152563201253	0.55	Adult
2	30201548596352	Abdelrahman Mohamed	01045863251	20	Cairo	01258963251	0	01024589637	20301548520137	1.24	Adult
3	30245849635712	Karem Assem	01235896541	22	Minia	01258963251	0	01253658477	20315059632104	1.10	Adult
4	42589635715284	Hazem Khaled	01028963547	26	Mansora	01258963251	0	01258963457	20102365215422	0.82	Adult
5	20301548967415	Mustafa Yahia	01036587460	28	Assuit	01258963251	0	01236584754	01025485236541	0.68	Adult
6	25352015984520	Gmela Sayed	01025879634	22	Sohag	01258963251	1	01254589632	20103654852012	1.10	Adult

```python
# Diseases table before processing and cleaning
diseases_df.head(7)
```

[162] ✓ 0.1s

	Doctor_ID	diabetes	hypertention	cholesterol	Patient_ID
0	1001	1	0	1	20320154852369
1	1002	0	0	0	30201548596352
2	1003	1	1	1	30245849635712
3	1004	0	0	1	42589635715284
4	1005	1	0	0	20301548967415
5	1006	1	1	0	25352015984520
6	1007	0	0	1	40475302156985

```python
# Survay table before processing and cleaning
survay_result_df.head(7)
```

[165] ✓ 0.0s

	survey_id	q_one	q_two	q_three	q_four	q_five	q_six	q_seven	q_eight	q_nine	q_ten	survey_date	Patient_National_ID
0	4520	1	0	1	1	1	0	0	0	1	1	2022-04-20	20320154852369
1	4521	0	1	0	1	1	1	1	0	0	0	2016-07-24	30201548596352
2	4522	0	1	1	0	0	1	0	0	1	1	2022-06-22	30245849635712
3	4523	1	1	1	0	1	0	0	0	1	0	2022-02-22	42589635715284
4	4524	0	1	0	0	0	1	1	1	1	0	2018-12-12	20301548967415
5	4525	0	1	1	0	1	0	1	1	0	0	2023-03-08	25352015984520
6	4526	0	0	1	0	1	0	1	1	0	0	2019-11-11	40475302156985

Marge all tables:

```python
data = pd.concat([patient_df.set_index('Patient_National_ID'),
                  medicalresult_df.set_index('Patient_National_ID'),
                 diseases_df.set_index('Patient_ID'),
                 survay_result_df.set_index('Patient_National_ID')],
                   axis=1, join='outer')
data.reset_index()
```

[88] Python

	index	Patient_Name	Patient_Phone	Patient_Age	Patient_Address	Doctor_Phone	Patient_Gender	Follower_Phone	Doctor_National_ID	Ag
0	30111132400351	Momen Khaled	01099473695	25	Minia	01128922921	0	01111111111	None	
1	20320154852369	Ali Bakr	01025486358	30	Mallawi	01258963251	0	01158263541	20152563201253	
2	30201548596352	Abdelrahman Mohamed	01045863251	20	Cairo	01258963251	0	01024589637	20301548520137	
3	30245849635712	Karem Assem	01235896541	22	Minia	01258963251	0	01253658477	20315059632104	
4	42589635715284	Hazem Khaled	01028963547	26	Mansora	01258963251	0	01258963457	20102365215422	

```python
final_merged_data = pd.concat([data.set_index('Doctor_National_ID'), doctor_df.set_index('Doctor_National_ID')],axis=1, join='inner')
final_data.to_csv('Finale Merged Data.csv', index = False)
```

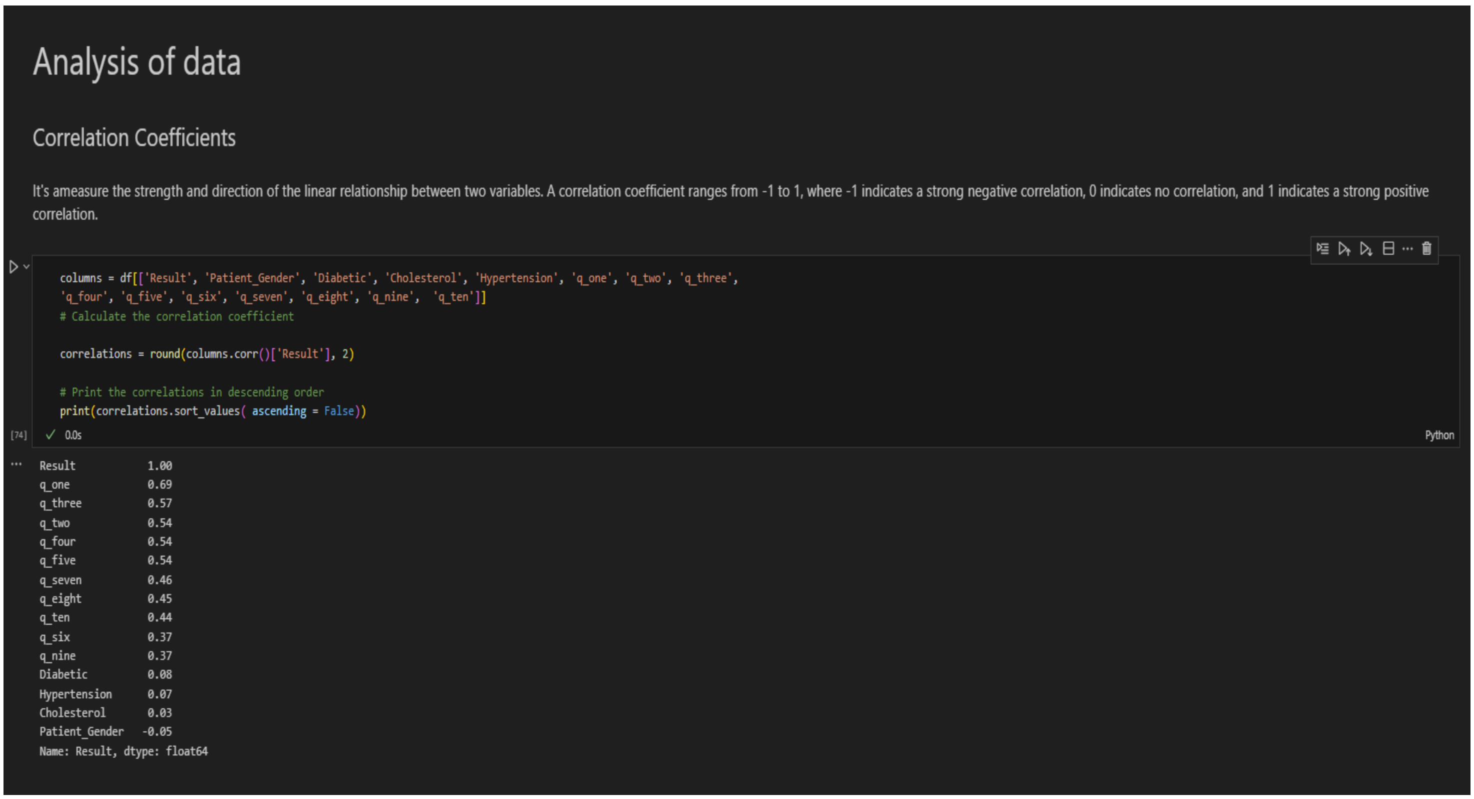
✓ 0.1s Python

6. Data analysis, Data visualization and reporting:

Once the data has been explored, the next step is to perform the actual data analysis. This may involve applying statistical models or machine learning algorithms to the data or conducting hypothesis testing to determine the significance of observed patterns or trends.

The choice of statistical models used in data analysis depends on the specific goals and questions being addressed. However, some common statistical models that are often used in data analysis include:

1. Linear regression: Linear regression is a statistical model used to establish a relationship between an independent variable and a dependent variable. It is often used to predict the value of a dependent variable based on the value of one or more independent variables.



Analysis of data

Correlation Coefficients

It's ameasure the strength and direction of the linear relationship between two variables. A correlation coefficient ranges from -1 to 1, where -1 indicates a strong negative correlation, 0 indicates no correlation, and 1 indicates a strong positive correlation.

```python
columns = df[['Result', 'Patient_Gender', 'Diabetic', 'Cholesterol', 'Hypertension', 'q_one', 'q_two', 'q_three',
'q_four', 'q_five', 'q_six', 'q_seven', 'q_eight', 'q_nine',  'q_ten']]
# Calculate the correlation coefficient

correlations = round(columns.corr()['Result'], 2)

# Print the correlations in descending order
print(correlations.sort_values( ascending = False))
```

[74] ✓ 0.0s Python

```
Result          1.00
q_one           0.69
q_three         0.57
q_two           0.54
q_four          0.54
q_five          0.54
q_seven         0.46
q_eight         0.45
q_ten           0.44
q_six           0.37
q_nine          0.37
Diabetic        0.08
Hypertension    0.07
Cholesterol     0.03
Patient_Gender  -0.05
Name: Result, dtype: float64
```

- It is important to explore and understand the relationships between different variables in the dataset. One way to do this is through data visualization, which can help identify patterns and trends in the data that may not be immediately apparent from looking at the raw numbers. Some commonly used visualization techniques for data analysis include scatter plots, histograms, bar charts, and line graphs. These can help us to visualize the distribution of data, identify outliers or anomalies, and understand the relationships between different variables.
- The results of the data analysis must be communicated to stakeholders. This may involve creating visualizations or charts to help convey the results or writing reports or presentations to communicate the findings.

Some analysis and visualizations from code

Injured Male & female

```python
# Convert Result, Patient_Gender to int datatypes
df['Result'] = df['Result'].astype('int')
df['Patient_Gender'] = df['Patient_Gender'].astype('int')
```

✓ 0.0s

```python
df['Result'] = df['Result'].astype(str)
df['Patient_Gender'] = df['Patient_Gender'] .astype(str)
# Replace the values in the 'resultlabel' column
df['Result'] = df['Result'].replace({'0': 'Not Injured', '1': 'Injured'})
df['Patient_Gender'] = df['Patient_Gender'].replace({'0': 'Male', '1': 'Female'})
```

✓ 0.0s

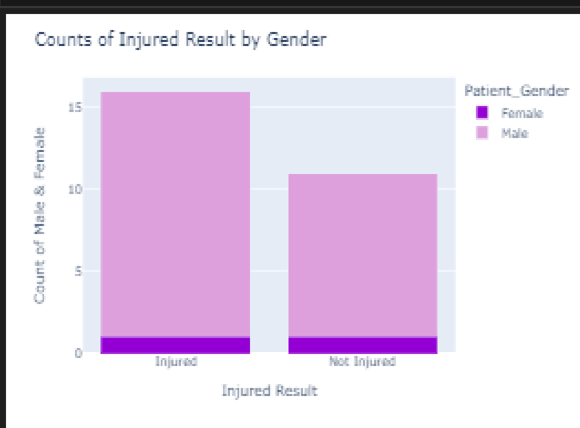
```python
# Group the DataFrame by 'resultlabel' and 'Gender' columns and count the occurences
grouped_df_resultlabel_gender = df.groupby(['Result', 'Patient_Gender']).size().reset_index(name='counts')
grouped_df_resultlabel_gender
```

✓ 0.0s

	Result	Patient_Gender	counts
0	Injured	Female	1
1	Injured	Male	15
2	Not Injured	Female	1
3	Not Injured	Male	10

```python
# Create a stacked bar chart of the counts grouped by 'resultlabel' and 'Gender' using Plotly Express
fig = ex.bar(grouped_df_resultlabel_gender, x = 'Result', y = 'counts', color = 'Patient_Gender',
             barmode = 'stack', color_discrete_sequence=['darkviolet', 'plum'])

# Add title and axis labels
fig.update_layout(title = 'Counts of Injured Result by Gender')
fig.update_xaxes(title = 'Injured Result')
fig.update_yaxes(title = 'Count of Male & Female')

fig.update_layout(width=600, height=400)
# Display the chart using Graph Objects
fig.show()
```

✓ 0.1s



Observation:

- Males have more infections than females in remarkable percentage.

Diseases within Injuries

```
# Rename Survay_Result_Year
df = df.rename(columns = {'Survay_Result_Year':'Survay_Year'})
```
[139] ✓ 0.0s

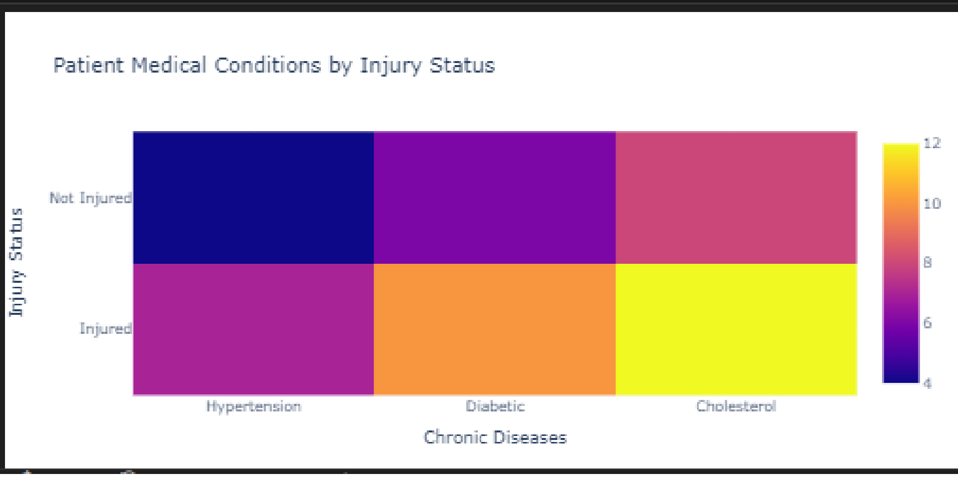
```
df.Medical_Result_Year = df.Medical_Result_Year.astype(int)
df.Survay_Year = df.Survay_Year.astype(int)
```
[140] ✓ 0.0s

```
# replace 1 with "Injured" and 0 with "Not_Injured" in the "Result" column
df['Result_Inj_not'] = df['Result'].replace({1: 'Injured', 0: 'Not_Injured'})
# temporary dataframe containing chronic diseases as: Hypertension, Cholesterol, Diabetic and the result of detection
temp = df[['Hypertension', 'Cholesterol', 'Diabetic','Result_Inj_not']]
disease_counts = temp.groupby('Result_Inj_not')['Hypertension', 'Diabetic', 'Cholesterol'].sum()
disease_counts
```
[181] ✓ 0.0s

```
C:\Users\sohil\AppData\Local\Temp\ipykernel_11124\4097901107.py:3: FutureWarning:

Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.
```

	Hypertension	Diabetic	Cholesterol
Result_Inj_not			
Injured	7	10	12
Not Injured	4	6	8

```
index = pd.MultiIndex.from_product([['Injured', 'Not Injured'], ['Result_Inj_not']])


# create heatmap chart using plotly
fig = go.Figure(data=go.Heatmap(z = disease_counts.values,
                                x = disease_counts.columns,
                                y = disease_counts.index.get_level_values(0)))

# update chart layout and axis labels
fig.update_layout(title='Patient Medical Conditions by Injury Status',
                  xaxis_title='Chronic Diseases',
                  yaxis_title='Injury Status')

fig.update_layout(width=800, height=400)
# show the chart
fig.show()
```
[144] ✓ 0.0s



Injuries Patients by Age Stage

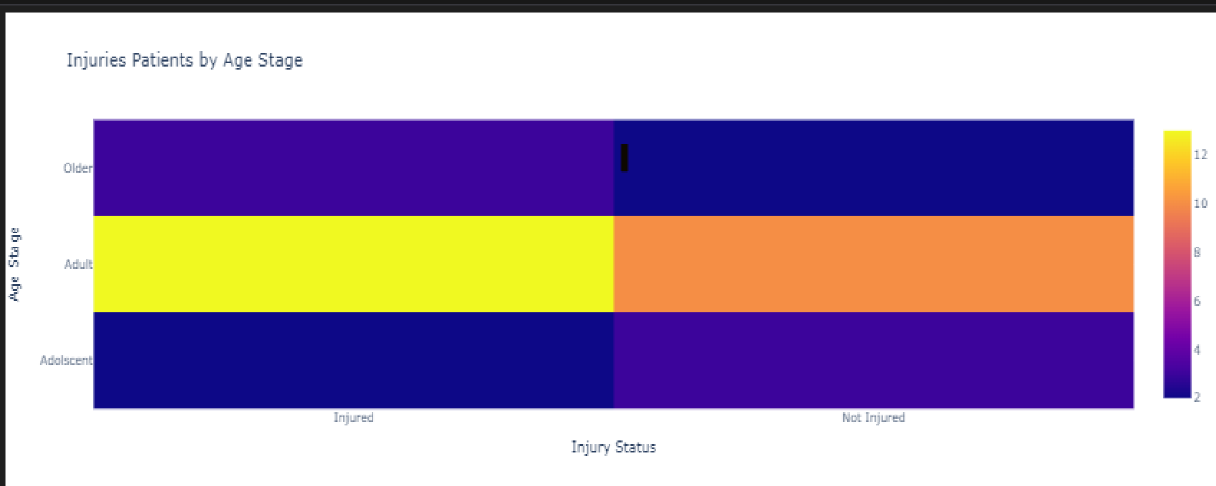
```python
# Group data of 'Patient_Age_Stage' and 'Result' with number of Injured and not injured patients.
cnt = df.groupby(['Patient_Age_Stage', 'Result_Inj_not']).size().reset_index(name='counts')
# Pivot data and make Injured and not injured number is columns
cnt = cnt.pivot(index = 'Patient_Age_Stage', columns = 'Result_Inj_not', values = 'counts').reset_index()
# Set index is 'Patient_Age_Stage'
cnt = pd.DataFrame(cnt).set_index('Patient_Age_Stage')
cnt
```

✓ 0.0s

Result_Inj_not	Injured	Not Injured
Patient_Age_Stage		
Adolscent	2	3
Adult	13	10
Older	3	2

```python
# create MultiIndex from columns 'Not Injured' and 'Injured'
cnt.columns = pd.MultiIndex.from_product([cnt.columns, ['Count']])

# create heatmap chart using plotly
fig = go.Figure(data=go.Heatmap(z = cnt.values,
                                x = cnt.columns.levels[0],
                                y = cnt.index))



# update chart layout and axis labels
fig.update_layout(title='Injuries Patients by Age Stage',
                  xaxis_title='Injury Status',
                  yaxis_title='Age Stage')

# show the chart
fig.show()
```

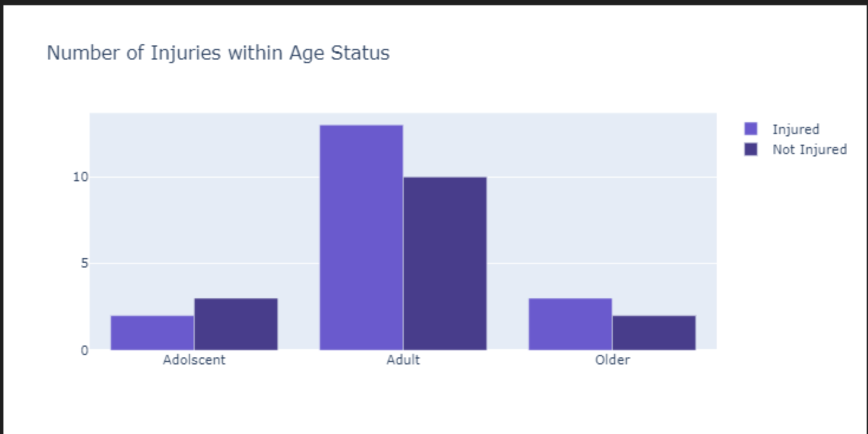
✓ 0.0s



Number of Injuries within Age Status

```python
# Group data of 'Patient_Age_Stage' and 'Result' with number of Injured and not injured patients.
cnt = df.groupby(['Patient_Age_Stage', 'Result']).size().reset_index(name='counts')
# Pivot data and make Injured and not injured number is columns
cnt = cnt.pivot(index='Patient_Age_Stage', columns='Result', values= 'counts').reset_index()
# Create a figure
fig = go.Figure()
fig.add_trace(go.Bar(y = cnt.Injured, x = cnt.Patient_Age_Stage,
                     name = 'Injured', marker_color = 'slateblue'))
fig.add_trace(go.Bar(y = cnt['Not Injured'], x = cnt.Patient_Age_Stage,
                     name = 'Not Injured', marker_color = 'darkslateblue'))
fig.update_layout(title='Number of Injuries within Age Status')

fig.update_layout(width = 800, height = 400)
fig.show()
```

✓ 0.0s



Make temp contian subset of data of just injured patients

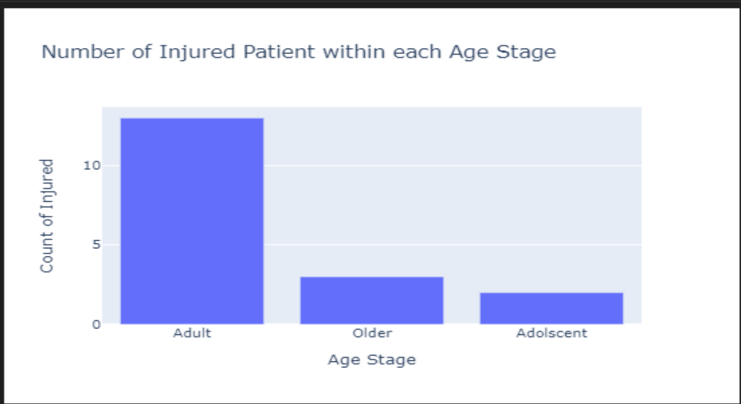
```python
injured = df[df['Result'] == 'Injured']
Age_stage_value_counts = injured.Patient_Age_Stage.value_counts()
Age_stage_value_counts
```

✓ 0.0s

```
Adult        13
Older         3
Adolscent     2
Name: Patient_Age_Stage, dtype: int64
```

```python
fig = ex.bar(Age_stage_value_counts, x=Age_stage_value_counts.index, y=Age_stage_value_counts.values,
             title='Number of Injured Patient within each Age Stage')

fig.update_layout(width = 600, height = 400,  yaxis_title='Count of Injured', xaxis_title = 'Age Stage')
fig.show()
```

✓ 0.1s



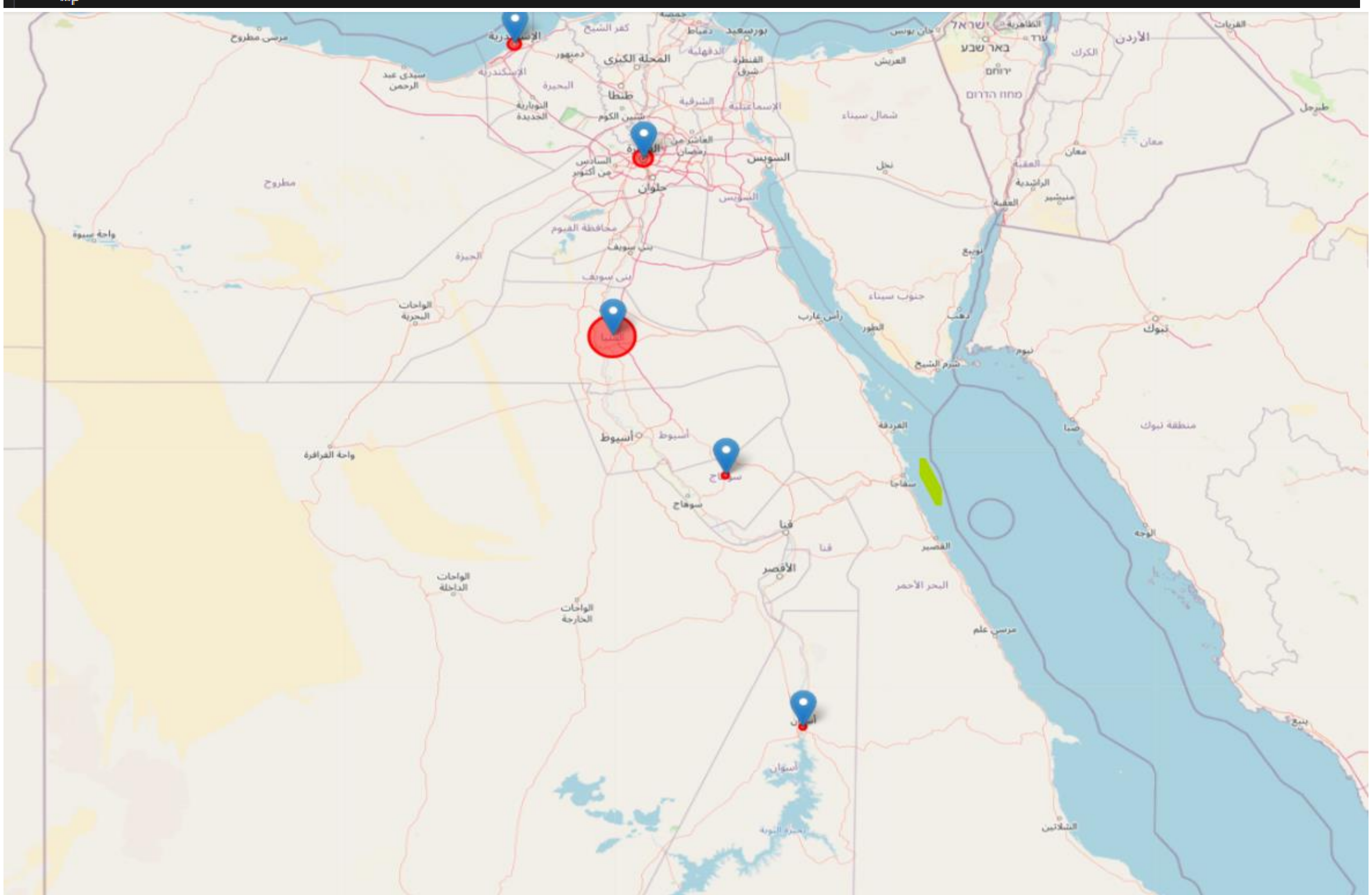
Governments with injuries

```python
govr_with_injured = injured.groupby('Patient_Address')['Patient_National_ID'].count()
govr_with_injured
govr_with_injured_dict = govr_with_injured.to_dict()
print('govrnments with injured and number of injuries', govr_with_injured_dict)
print('Keys', govr_with_injured_dict.keys())
```

✓ 0.0s

```
Govrnments with injured and number of injuries {'Alexandria': 2, 'Assuit': 3, 'Aswan': 1, 'Cairo': 3, 'Minya': 8, 'Sohag': 1}

Keys dict_keys(['Alexandria', 'Assuit', 'Aswan', 'Cairo', 'Minya', 'Sohag'])
```

```python
dict_gov = {}
rad = 500
mp = folium.Map(location=[26.8206, 30.8025], zoom_start=6)#, width=300, height=300)
for gov in govr_with_injured_dict.keys():
    geolocator = Nominatim(user_agent='my-app')
    place_name = '{}, Egypt'.format(gov)
    location = geolocator.geocode(place_name)
    if location is not None:
        lat = location.latitude
        lon = location.longitude
        folium.Marker(location=[lat, lon], tooltip = gov).add_to(mp)
        folium.Circle(location=[lat, lon],
                      radius = govr_with_injured_dict[gov] * 3000,
                      color='red',
                      fill=True,
                      fill_color='red',
                      fill_opacity=0.5,
                      tooltip=f"{gov} ({govr_with_injured_dict[gov]})").add_to(mp)
mp
```



In this chapter, we analyzed a dataset of Phenomena for our application. Our objective was to identify key factors that marketing, Generate Ratios, Display the charts in a dashboard.
As our data is still small, the results will change in increase of data and uses use the applications.