Report on Basic statistical analysis in HealthCare (COVID-19 Data Analysis)

CHETAN GOWDA

March 2025

Abstract

This report explores the COVID-19 dataset provided in the `COVID_19.xlsx` file. The dataset comprises various demographic, medical, and behavioral variables, such as age, gender, vaccination status, smoking habits, and COVID-19-related health conditions. The purpose of this report is to examine these variables to uncover potential trends, associations, and patterns using Python and associated libraries such as `Pandas`, `Matplotlib`, and `Seaborn`. The goal is to gain insights that can inform public health policy and contribute to understanding the COVID-19 pandemic's impact.

1 Introduction

This report explores the COVID-19 dataset provided in the `COVID_19.xlsx` file. The data comprises various demographic, medical, and behavioral variables such as age, gender, vaccination status, smoking habits, and COVID-19-related health conditions. The purpose of this report is to examine these variables to uncover potential trends, associations, and patterns using Python and the associated libraries `Pandas`, `Matplotlib`, and `Seaborn`. By performing detailed data analysis and visualization, we aim to draw actionable insights that could be valuable for public health monitoring and policy planning.

The steps in this report involve:

- **Data Preprocessing**: Handling missing values, parsing and transforming data into usable formats.
- **Statistical Analysis**: Performing descriptive and inferential statistics.
- **Data Visualization**: Generating insightful plots and graphs to highlight key findings.
- **Conclusions**: Summarizing the analysis and suggesting potential implications for public health research.

2 Data Import and Initial Exploration

The dataset is first loaded into a `Pandas` DataFrame, which allows us to work with the data efficiently. The initial exploration involves inspecting the structure of the dataset and checking the contents of the columns.

2.1 Loading the Data

We begin by importing the necessary libraries for handling and visualizing the data. `Pandas` is used for data manipulation, `Matplotlib` and `Seaborn` are used for visualization, and `Datetime` is used for handling date-time values.

```
import pandas as pd
from datetime import datetime

# Load dataset
df = pd.read_excel(r"COVID_19.xlsx", 'Sheet1')

# Display the first few rows of the dataset to inspect the structure
df.head()
```

2.2 Dataset Overview

The dataset contains the following columns:

- **Date time**: The timestamp when the data was collected.
- **Age**: The age of the respondent.

- **Gender**: The gender of the respondent.
- **Region**: The region where the respondent resides.
- **Do you smoke?**: Whether the respondent smokes or not.
- **Have you had Covid'19 this year?**: Indicates whether the respondent had COVID-19 this year.
- **IgM level** and **IgG level**: Immunoglobulin M and G levels to detect the presence of COVID-19 antibodies.
- **Blood group**: The respondent's blood group.
- **Vaccination status** for influenza and tuberculosis.

The first step is to clean and preprocess this data, as real-world datasets often contain inconsistencies, missing values, and data type issues.

3 Data Preprocessing

Data preprocessing is an essential part of any data analysis pipeline. It ensures that the dataset is clean, consistent, and ready for statistical analysis.

3.1 Handling Missing Values

It's common to encounter missing values in real-world datasets, and we must decide how to handle these missing values. In our dataset, we are particularly concerned with columns such as **Gender**, where missing values would affect the analysis. Therefore, rows with missing **Gender** values are dropped using `dropna()`.

```
df = df.dropna(subset=['Gender'])
```

This operation removes all rows where the **Gender** column has missing values, ensuring that we only analyze respondents with complete gender data.

3.2 Data Transformation

Some columns may contain data that needs to be converted into more appropriate formats for analysis. One example is the **Age** column. Although it is numeric, it is more useful for statistical analysis if it is categorized. Therefore, we convert **Age** into a categorical column.

```
df['Age'] = df['Age'].astype('category')
```

Additionally, **Date time** is stored in a string format, which can complicate temporal analysis. We use a function to parse the date and time correctly.

```
def parse(x):
    y = x.split()
    t = y[1][:8]
    z = y[0] + " " + t
    d = datetime.strptime(z, '%Y-%m-%d %H:%M:%S')
    return d

df['Date time'] = df['Date time'].apply(parse)
```

3.3 Replacing Categorical Values

Some columns contain categorical values that could be more useful as binary or boolean values. For example, in the **Do you vaccinated influenza?** column, we want to replace the textual values "Yes" and "No" with boolean values `True` and `False` respectively.

```
d = {'No': False, 'Yes': True}
df['Do you vaccinated influenza?'] = df['Do you vaccinated influenza?'].map(d)
```

This simplifies analysis by allowing us to work with boolean data, which is easier to analyze than text-based categories.

4 Data Exploration

4.1 Dataset Information

The `info()` method provides us with a summary of the dataset's columns, non-null values, and data types.

```
df.info()
```

The output tells us if any columns still contain missing values, the data types for each column, and how many non-null values exist in each column.

4.2 Summary Statistics

For numerical columns, we can use the `describe()` function to obtain summary statistics such as the mean, standard deviation, minimum, maximum, and percentiles. This helps in understanding the overall distribution of values.

```
df.describe()
```

For categorical columns, we can use `describe(include=['category'])` to get a breakdown of the frequencies of the categories.

```
df.describe(include=['category'])
```

4.3 Value Counts

The `value_counts()` function helps us count the number of occurrences of each unique value in a column. For instance, we can examine the distribution of **Gender**:

```
df['Gender'].value_counts()
```

4.4 Normalized Counts

We can normalize the value counts to get the proportion of each category rather than the raw counts.

```
df['Gender'].value_counts(normalize=True)
```

This gives us a better understanding of the relative distribution of gender in the dataset.

4.5 Sorting Data

To explore specific patterns in the data, we can sort it. For instance, sorting by **Age** in ascending order or by **Gender** in descending order can help us identify any trends in the dataset.

```
df.sort_values(by=['Age', 'Gender'], ascending=[True, False]).head()
```

This operation sorts the data first by age (ascending) and then by gender (descending).

5 Statistical Analysis

5.1 Grouping Data

By grouping the data by specific columns, we can calculate aggregated statistics for those groups. For example, to examine the average **Maximum body temperature** by **Gender**, we can group the data as follows:

```
df.groupby(['Gender'])['Maximum body temperature'].describe()
```

This command will output descriptive statistics for each gender separately, including the mean, standard deviation, and other statistical measures.

5.2 Pivot Tables

A pivot table allows us to summarize data by grouping it across multiple dimensions. For instance, we can create a pivot table to display the mean **Maximum body temperature** for each age group by **Gender**.

```
pd.pivot_table(df, values='Maximum body temperature', index=['Age'], columns=['Gender
```

The resulting table will show the average body temperature for males and females across different age groups.

5.3 Crosstab

A crosstab is a specific type of pivot table that is useful for summarizing categorical data. For example, we can use `crosstab` to examine the relationship between **Age** and **Gender**:

```
pd.crosstab(df['Age'], df['Gender'])
```

This will display a table that shows how many men and women are in each age group.

6 Data Visualization

Visualization is a powerful tool for understanding and communicating the patterns within a dataset. Here, we will explore several types of plots to gain insights into the relationships between various variables in the dataset.

6.1 Count Plots

Count plots are useful for visualizing the distribution of categorical variables. For example, we can use a count plot to display the distribution of **Gender** across different **Age** groups:

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(data=df, x='Age', hue='Gender')
plt.show()
```

This plot helps us understand how the number of respondents varies across different age groups and whether there are more males or females in each age group.

6.2 Distribution Plots

We can use distribution plots (with `displot()`) to visualize the distribution of a continuous variable, such as **Maximum body temperature**, for people who had COVID-19.

```
df_t = df[df['Have you had Covid‘19 this year?'] == 'Yes'].dropna(subset=['Maximum bo
sns.displot(df_t['Maximum body temperature'], kde=True, height=6, aspect=1.5)
plt.show()
```

6.3 Box Plots

Box plots are ideal for visualizing the distribution and spread of data across categories. For example, we can use a box plot to visualize how **Maximum body temperature** is distributed across different **Age** groups:

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Age', y='Maximum body temperature', hue='Region', data=df, palette="Se
plt.title("Temperature Distribution by Region and Age Group")
plt.xlabel("Age Group")
plt.ylabel("Maximum Body Temperature")
plt.show()
```

6.4 Violin Plots

Violin plots combine aspects of box plots and kernel density plots, providing more detailed insights into the distribution of data. We can use a violin plot to visualize **Maximum body temperature** by **Age** group.

```
plt.figure(figsize=(10, 6))
sns.violinplot(x='Age', y='Maximum body temperature', data=df, inner="quart", palette=
plt.title("Distribution of Maximum Body Temperature by Age Group")
plt.xlabel("Age Group")
plt.ylabel("Maximum Body Temperature")
plt.show()
```

6.5 Heatmaps

Heatmaps are ideal for visualizing correlation matrices, which allow us to examine the relationships between multiple variables. For example, we can generate a heatmap of the correlation matrix for the numerical variables in the dataset:

```
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix of Numerical Variables")
plt.show()
```

6.6 KDE Plots

Kernel Density Estimation (KDE) plots help us visualize the probability distribution of a continuous variable. We can create KDE plots for the **Maximum body temperature** based on **Age**:

```
plt.figure(figsize=(10, 6))
sns.kdeplot(data=df, x='Maximum body temperature', hue='Age', common_norm=False, fill=
plt.title("Density Plot of Maximum Body Temperature by Age Group")
plt.xlabel("Maximum Body Temperature")
plt.ylabel("Density")
plt.show()
```

7 Data Categorization

To further understand the body temperature data, we categorize **Maximum body temperature** into three groups: **Hypothermia**, **Normal**, and **Fever**.

7.1 Categorization Function

We define a function that takes a **Maximum body temperature** value and assigns a category based on predefined thresholds.

```
def categorize_temp(temp):
    if temp < 36.0:
        return 'Hypothermia'
    elif 36.0 <= temp <= 37.5:
        return 'Normal'
    else:
        return 'Fever'
```

```
# Apply the function to the dataset
df['Temp Category'] = df['Maximum body temperature'].apply(categorize_temp)
```

This categorization allows us to investigate how temperature categories vary across different **Age** groups:

```
sns.countplot(data=df, x='Age', hue='Temp Category', palette="Set2")
plt.show()
```

8 Conclusions

8.1 Summary of Findings

- **Data Preprocessing**: We successfully handled missing values, parsed dates correctly, and transformed categorical and numerical data into suitable formats for analysis.
- **Demographic Insights**: The dataset reveals a reasonably balanced distribution of gender and age groups, with notable differences in the proportions of males and females across different age groups.
- **Health-related Insights**: The analysis showed that the majority of respondents had a normal body temperature, with smaller groups experiencing fever or hypothermia.
- **COVID-19 Trends**: A higher body temperature was noted in people who reported having had COVID-19 this year, although the variability was high across different age groups and regions.

8.2 Implications for Public Health

This analysis demonstrates that a significant number of people experienced abnormal body temperatures during the pandemic, potentially indicating the effects of COVID-19. Further research could explore the relationship between vaccination status and body temperature or investigate regional variations in COVID-19 severity.

The insights gathered from this dataset can be useful for understanding how demographic and health behaviors correlate with the severity of COVID-19, providing valuable information for public health interventions and policy development.

8.3 Future Work

- **Expand Dataset**: Incorporate more health indicators, such as vaccination history, comorbidities, and genetic predispositions.
- **Modeling**: Build predictive models to forecast the likelihood of COVID-19 infection based on demographic and health factors.
- **Longitudinal Analysis**: Analyze the changes in temperature and other health variables over time to identify emerging patterns during and after the pandemic.