TABLE OF CONTENT

1.Business Understanding

1.1 Company Background

Situated in rural Bangladesh, Kota Hospital represents a beacon of hope and care for local communities. Established in 2014 by a group of dedicated healthcare professionals and community leaders, including esteemed members like Dr. Chin Bao Sheng, Dr. Kok Ka Ket, Dr. Wong Rui Sean, and Dr. Wong Jun Wei, the hospital's mission is clear: to ensure everyone, regardless of where they live or their background, has access to good healthcare. Using smart technology called IoT, Kota Hospital can keep an eye on health risks and provide help when needed. Through health education and outreach programs, Kota Hospital not only treats people when they're sick but also teaches them how to stay healthy, making a real difference in people's lives.

1.2 Problem Background

Maternal health, referring to the health of women during pregnancy, childbirth, and the postpartum period, is a significant concern globally, particularly in regions where access to healthcare is limited. Despite efforts to improve maternal health outcomes, including advancements in medical technology and increased awareness, maternal mortality rates remain unacceptably high. Complications during pregnancy and childbirth continue to pose serious risks to the health and well-being of mothers, with preventable factors often contributing to adverse outcomes. Challenges such as insufficient healthcare infrastructure, inadequate prenatal care, and socio-economic disparities further exacerbate the issue, particularly in resource-constrained settings. Addressing maternal health risks requires a multifaceted approach, encompassing both medical and social interventions, to ensure that mothers receive the care and support they need to have safe pregnancies and deliveries. In this context, leveraging data science techniques and methodologies can play a crucial role in identifying patterns, trends, and risk factors associated with maternal health outcomes, thereby informing targeted interventions and strategies to improve maternal health and reduce mortality rates.

1.3 Problem Statement

The problem at hand is to develop data-driven approaches to identify and address maternal health risks in resource-constrained settings. Specifically, the goal is to leverage data science techniques and methodologies to analyze maternal health data and identify patterns, trends, and risk factors associated with adverse maternal outcomes. By understanding the underlying drivers of poor maternal health and developing predictive models to anticipate and mitigate these risks, we aim to improve maternal health outcomes and reduce maternal mortality rates. This involves developing robust data models, leveraging advanced analytic techniques, and collaborating with healthcare stakeholders to implement targeted interventions that address maternal health risks effectively.

1.4 Company Aim

At the core of Kota Hospital's mission lies a steadfast commitment to prioritizing the health and vitality of every individual within the community. With a two-pronged approach aimed at addressing critical healthcare challenges, the hospital is dedicated to ensure the safety of expectant mothers during pregnancy and nurturing resilient heart health for all individuals. Central to this endeavor is the collection and analysis of essential health data, encompassing vital parameters such as age, blood pressure, blood sugar levels, body temperature, and heart rate.

By utilizing the insights gained from data analysis, Kota Hospital aims to promptly identify any potential risks or concerns related to pregnancy complications or heart disease. Through proactive interventions and tailored healthcare approaches, the hospital strives to improve health outcomes and enhance the overall well-being of its patients. Aligned with global healthcare guidelines, Kota Hospital remains committed to employ advanced technologies and innovative medical practices to provide excellent healthcare services to its community.

With an unwavering focus on fostering a culture of health and happiness, Kota Hospital aims to empower individuals to lead fulfilling and vibrant lives. Through its comprehensive approach to healthcare delivery, the hospital strives to create a supportive environment where every mother can thrive in optimal health and well-being.

1.5 Company Objectives

1. Understand Patient Needs and Preferences

The first objective is to understand the needs and preferences of pregnant individuals and mothers regarding maternal health risk management. This involves conducting surveys, interviews, or focus groups to gather insights into patient experiences, concerns, and expectations. By understanding patient needs and preferences, we can design data science solutions that are patient-centered and responsive to their unique circumstances, ultimately improving the quality of maternal healthcare delivery.

2. Assess Data Availability and Quality

The second objective is to assess the availability and quality of data relevant to maternal health risk, including demographic information, medical records, and health outcomes. This involves understanding the sources of data, potential biases or limitations, and data collection processes. By assessing data availability and quality, we can determine the feasibility of data science approaches and identify any gaps or areas for improvement.

3. Analyze Market Trends

The third objective is to analyze market trends and industry developments related to maternal health risk management. This involves monitoring advancements in healthcare technology, changes in regulatory frameworks, and emerging best practices in maternal health care. By staying informed about market trends, we can identify opportunities for innovation and collaboration, as well as potential risks or challenges that may impact the success of data science initiatives.

2.0 Data Understanding

2.1 Data Collection

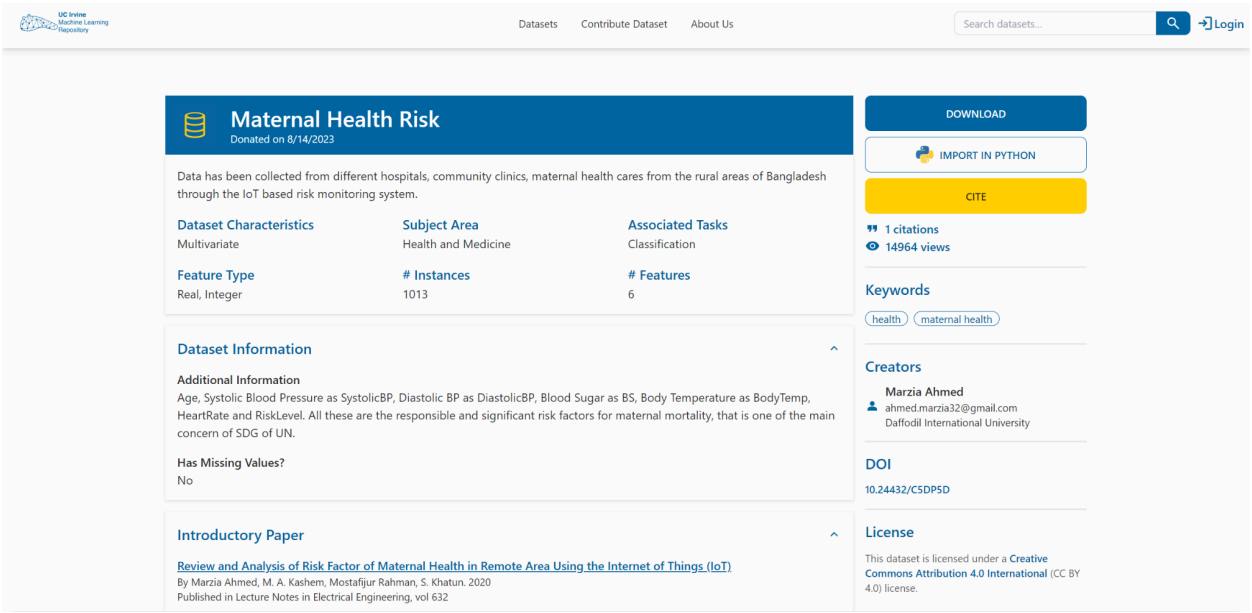


Figure 2.1: Dataset's source

In order to monitor and collect vital health metrics from various healthcare facilities in Bangladesh's rural areas, a complete method utilizing IoT technology was used for the maternal health risk dataset data collection procedure. As stated in their paper "Review and Analysis of Risk Factor of Maternal Health in Remote Area Using the Internet of Things (IoT)" published in Lecture Notes in Electrical Engineering, Marzia Ahmed and her team from Daffodil International University led the initiative, which was carried out in 2020. The dataset consists of 1013 cases and six major characteristics: age, blood pressure levels (both systolic and diastolic), blood sugar levels, body temperature, heart rate, and associated risk factors. In order to accurately represent the wide range of demographics and healthcare environments seen in rural Bangladesh, these characteristics were carefully gathered from various hospitals, community clinics, and maternal healthcare centers. Through the utilization of Internet of Things devices to enable real-time monitoring, this data gathering methodology guarantees the prompt acquisition of critical health indicators that are essential for evaluating the hazards to maternal health. In addition, compliance with the Creative

Commons Attribution 4.0 International (CC BY 4.0) license emphasizes the dedication to transparent data exchange and cooperative research initiatives targeted at tackling maternal health issues in environments with limited resources.

2.2 Data Description

The following is a list of the variables that were utilized in this dataset:

Name	Type	Dependent/ Independent Variable	Description
Age	Integer	Independent Variable	Any **age in years** when women are pregnant.
SystolicBP	Integer	Independent Variable	Upper value of Blood Pressure in mmHg, another significant attribute during pregnancy, measured in mmHg.
DiastolicBP	Integer	Independent Variable	Lower value of Blood Pressure in mmHg, another significant attribute during pregnancy, measured in mmHg.
BS	Integer	Independent Variable	Blood glucose levels is in terms of a molar concentration (mmol/L).
BodyTemp	Integer	Independent Variable	Body temperature measured in Fahrenheit (°F).
HeartRate	Integer	Independent Variable	Heart rate measured in beats per minute (bpm).
RiskLevel	Categorical	Dependent Variable	Predicted Risk Intensity Level during pregnancy considering the previous attribute.

2.2.1 Load Dataset

The 'Maternal Health Risk Data Set.csv' CSV file is read into a Pandas dataframe in order to get the data ready for processing.

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
0	25	130	80	15.0	98.0	86	high risk
1	35	140	90	13.0	98.0	70	high risk
2	29	90	70	8.0	100.0	80	high risk
3	30	140	85	7.0	98.0	70	high risk
4	35	120	60	6.1	98.0	76	low risk
...	...	...	...	...	...	...	...
1009	22	120	60	15.0	98.0	80	high risk
1010	55	120	90	18.0	98.0	60	high risk
1011	35	85	60	19.0	98.0	86	high risk
1012	43	120	90	18.0	98.0	70	high risk
1013	32	120	65	6.0	101.0	76	mid risk

1014 rows × 7 columns

Figure 2.2.1: Maternal Health Risk Data Set.csv

There are 1014 rows of data in 7 columns in this dataset. There is 1 dependent variable and 6 independent variables in the 7 columns. All of the data in the dataset is shown as numbers, with the exception of the object-type data found in the "RiskLevel" data column.

2.2.2 Rename data column

The column names "SystolicBP", "DiastolicBP", "BS" and "BodyTemp" are unclear and might lead to misunderstandings regarding the interpretation of the data, therefore we renamed them to make it more apparent what the data is about.

```
#rename Column name
df = df.rename(columns = {'SystolicBP':'Systolic Blood Pressure', 'DiastolicBP':'Diastolic Blood Pressure', 'BS':'Blood Sugar','BodyTemp':'Body
df.head(10)
```

	Age	Systolic Blood Pressure	Diastolic Blood Pressure	Blood Sugar	Body Temperature	Heart Rate	Risk Level
0	25	130	80	15.00	98.0	86	high risk
1	35	140	90	13.00	98.0	70	high risk
2	29	90	70	8.00	100.0	80	high risk
3	30	140	85	7.00	98.0	70	high risk
4	35	120	60	6.10	98.0	76	low risk
5	23	140	80	7.01	98.0	70	high risk
6	23	130	70	7.01	98.0	78	mid risk
7	35	85	60	11.00	102.0	86	high risk
8	32	120	90	6.90	98.0	70	mid risk
9	42	130	80	18.00	98.0	70	high risk

Figure 2.2.2.1: Renamed column

```
df.dtypes
```

```
Age                          int64
Systolic Blood Pressure      int64
Diastolic Blood Pressure     int64
Blood Sugar                float64
Body Temperature           float64
Heart Rate                   int64
Risk Level                  object
dtype: object
```

Figure 2.2.2.2: Data Types

The output that was given describes the various data kinds of columns in a dataset like age, blood pressure, heart rate, blood sugar, body temperature, and risk level. While "Blood Sugar" and "Body Temperature" are represented as floating-point numbers ('float64'), indicating they may contain decimal values, "Age," "Systolic Blood Pressure," "Diastolic Blood Pressure," and "Heart Rate" are all represented as integers ('int64'), indicating whole number values. Given that

the 'Risk Level' column is classified as an 'object' data type, it is likely that the data it contains is categorical in nature and contains strings or a combination of data types. In data-driven jobs, having a thorough understanding of column data types is essential for appropriate data interpretation, manipulation, and analysis.

```
df.describe()
```

	Age	Systolic Blood Pressure	Diastolic Blood Pressure	Blood Sugar	Body Temperature	Heart Rate
count	1014.000000	1014.000000	1014.000000	1014.000000	1014.000000	1014.000000
mean	29.871795	113.198225	76.460552	8.725986	98.665089	74.301775
std	13.474386	18.403913	13.885796	3.293532	1.371384	8.088702
min	10.000000	70.000000	49.000000	6.000000	98.000000	7.000000
25%	19.000000	100.000000	65.000000	6.900000	98.000000	70.000000
50%	26.000000	120.000000	80.000000	7.500000	98.000000	76.000000
75%	39.000000	120.000000	90.000000	8.000000	98.000000	80.000000
max	70.000000	160.000000	100.000000	19.000000	103.000000	90.000000

Figure 2.2.2.3: Descriptive statistics table

According to this dataset, there are 1014 observations in each of the following six columns: Age, Systolic Blood Pressure, Diastolic Blood Pressure, Blood Sugar, Body Temperature, and Heart Rate. The mean value of age is 29.87, with a standard deviation of 13.47. The average systolic blood pressure is estimated to be 113.20, with a higher standard deviation of about 18.40, indicating greater variability in blood pressure measurements. Similarly, the mean values and standard deviations for other variables such as Diastolic Blood Pressure, Blood Sugar, Body Temperature, and Heart Rate provide insights into their distributions and variations.

2.2.3 Correlation between each variable

```
                              Age  Systolic Blood Pressure  \
Age                      1.000000                 0.416045
Systolic Blood Pressure  0.416045                 1.000000
Diastolic Blood Pressure  0.398026                 0.787006
Blood Sugar              0.473284                 0.425172
Body Temperature        -0.255323                -0.286616
Heart Rate               0.079798                -0.023108

                         Diastolic Blood Pressure  Blood Sugar  \
Age                                      0.398026     0.473284
Systolic Blood Pressure                  0.787006     0.425172
Diastolic Blood Pressure                 1.000000     0.423824
Blood Sugar                              0.423824     1.000000
Body Temperature                        -0.257538    -0.103493
Heart Rate                              -0.046151     0.142867

                         Body Temperature  Heart Rate
Age                             -0.255323    0.079798
Systolic Blood Pressure         -0.286616   -0.023108
Diastolic Blood Pressure        -0.257538   -0.046151
Blood Sugar                     -0.103493    0.142867
Body Temperature                 1.000000    0.098771
Heart Rate                       0.098771    1.000000
```

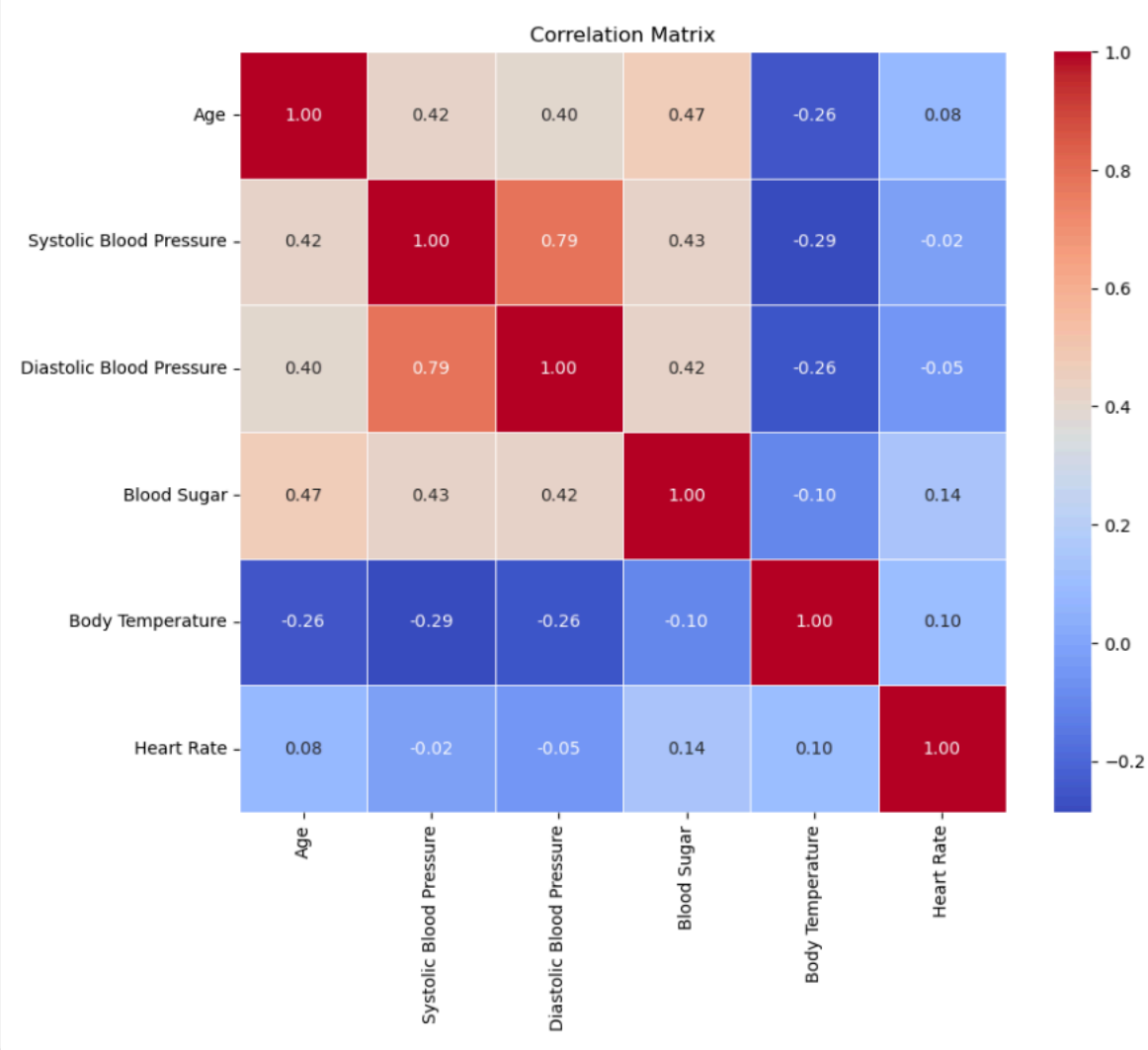


Figure 2.2.3: Correlation matrix table

The figure above is a correlation matrix table, it is used to show the pairwise correlations between all the variables in the dataset. The values range from -1 to 1, with -1 indicating a perfect negative correlation, 0 indicating no correlation, and 1 indicating a perfect positive correlation. This diagram visualizes the correlation matrix as a heatmap. The color intensity and shade represent the strength and direction of the correlation, respectively (blue for negative, red for positive).

Age is positively correlated with systolic blood pressure (0.42), diastolic blood pressure (0.40), and blood sugar (0.47), indicating that as age increases, these variables tend to increase as well. However, age is negatively correlated with body temperature (-0.26), suggesting that as age increases, body temperature tends to decrease. As people age, their metabolic rate tends to decrease. Metabolic processes generate heat in the body, contributing to body temperature. With a slower metabolic rate, less heat is produced, leading to a lower body temperature.

Systolic blood pressure is strongly positively correlated with diastolic blood pressure (0.79), which is expected as they are related measures of blood pressure. Both systolic and diastolic blood pressures are negatively correlated with body temperature and heart rate. This can be explained by the fact that blood vessels dilate (expand) to disperse heat when body temperature rises, which may cause blood pressure to drop. Similar to this, blood arteries may widen to enhance blood supply to muscles during physical activity or stress (which can boost heart rate), resulting in a drop in blood pressure.

Diastolic blood pressure is positively correlated with blood sugar (0.43), indicating that higher diastolic blood pressure is associated with higher blood sugar levels. High blood pressure, especially diastolic blood pressure, can result from endothelial dysfunction and arterial stiffness, which are diseases associated with elevated blood sugar levels such as insulin resistance or diabetes.

2.3 Independent Variable

Independent data are the variables that are utilized to forecast or explain changes in the dependent variable but are not impacted by other factors in the dataset. Heart rate, body temperature, blood sugar levels, systolic and diastolic blood pressure, and body temperature are examples of independent data in this dataset. These factors are thought to have the ability to affect maternal health because they are measured or observed directly from the study group of mothers. For example, measurements of the systolic and diastolic blood pressure offer information about cardiovascular health; blood sugar levels show metabolic function; body temperature indicates physiological status; and heart rate provides information about cardiac activity. These independent factors are studied to determine how they relate to the dependent variable, the risk level, and act as predictors or indicators of the health status of the mother.

2.3.1 Age

```
Summary of Age :
--------------------
count    1014.000000
mean       29.871795
std        13.474386
min        10.000000
25%        19.000000
50%        26.000000
75%        39.000000
max        70.000000
Name: Age, dtype: float64
--------------------
```

Figure 2.3.1.1: Summary of Age

The dataset includes 1014 age observations, indicating a wide variety of ages among the people who were examined. The dataset's average age is nearly 29.87 years, with a standard deviation of roughly 13.47 years, demonstrating a significant range in respondents' ages. The oldest individuals ever recorded is 70 years old, and the youngest is 10 years old, illustrating a wide range of age groups. When the distribution is further broken down, one-fourth of the people are under the age of 19, and the median age is 26, indicating that the population as a whole is rather young. In addition, the majority of respondents in the dataset are 39 years of age or younger, highlighting the presence of younger adults in the sample. Collectively, these figures provide insightful information about the age distribution of the sample population, emphasizing both its central tendency and variability.

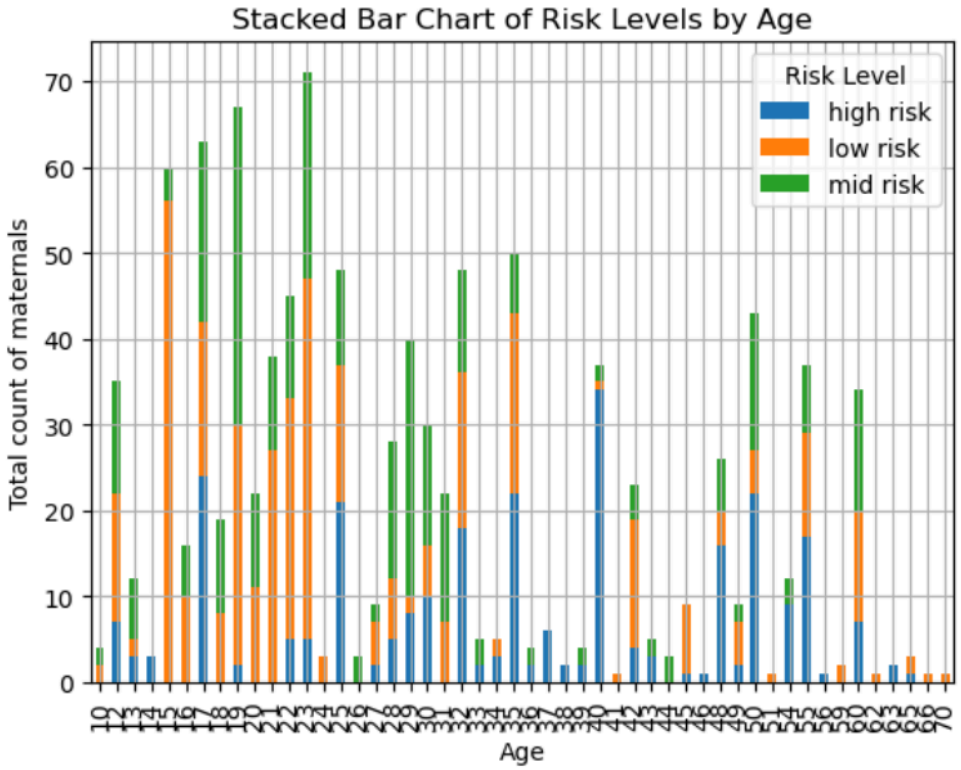


Figure 2.3.1.2: Stacked bar chart

The stacked bar chart depicts the distribution of individuals across different risk levels (high, mid, and low) within various age groups.

Examining the chart and data reveals several noteworthy trends. High-risk individuals are most prevalent in age groups such as 17, 25, 32, 35, 40, 48, 50, 54, and 55, indicating a concentration of individuals classified as high-risk within these age brackets. Conversely, mid-risk counts appear relatively high among late teens, early to mid-20s, and around ages 50 and 60.

Low-risk individuals generally dominate in younger age groups, notably ages 15-16, and some older age brackets, including ages 60-70. However, there are also significant low-risk counts in select age groups such as 19, 21-23, and 35.

Analysis of the age distribution suggests a varied distribution of risk levels across different age ranges. Younger individuals (below 20) exhibit higher occurrences of low and mid risk while those in their mid-30s to late 50s exhibit higher occurrences of high and mid-risk cases, whereas specific younger age brackets (15-16) show a higher prevalence of low-risk cases.

Distinct peaks and valleys in the chart indicate fluctuations in risk levels across age groups. Notable peaks in high-risk counts occur around ages 17, 25, 32, 40, 48-50, and 55, highlighting variability in risk levels across different stages of life.

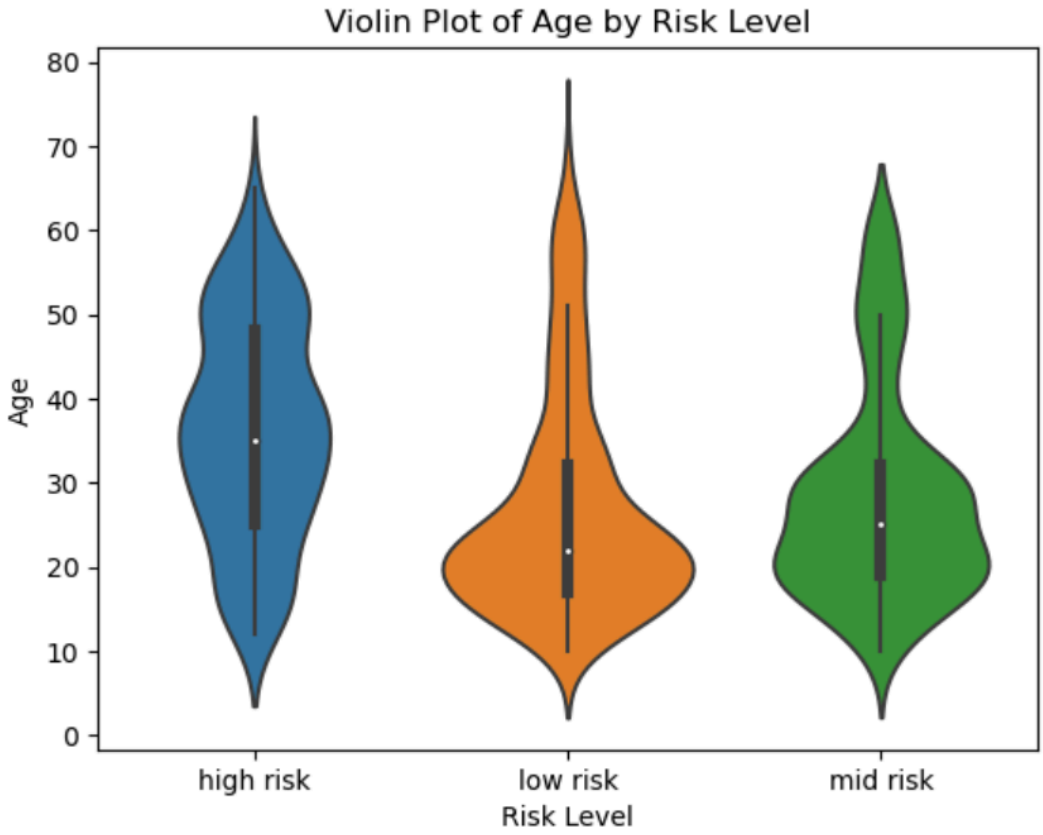


Figure 2.3.1.3: Violin Plot of Age by Risk Level

The diagram presented is a violin plot portraying age distributions across different risk levels: high, low, and mid-risk. Each "violin" shape illustrates the probability density of ages within these risk categories.

There are a few ways to read the data that the violin plot displayed. The shape of the violin plot represents the distribution of the data. It shows the probability density of the data at different values. Wider sections indicate a higher probability density, while narrower sections indicate a lower probability density. The thick gray bar in the center represents the interquartile range. The white dot represents the median. The thinner line inside the violin plot typically represents the 1.5 x interquartile range of the data. All observations lying outside the thinner line can be considered as outliers. Thus in this case, 3 risk levels with different ages indicates outliers. This is due to the fact that maybe the range of age differs greatly from the observations in the dataset.

In the high-risk level (blue violin), a bimodal distribution is evident, characterized by prominent peaks around ages 17, 25, 35 and mostly on 40. This suggests a higher likelihood of individuals

in their late teens and their 40s being classified as high risk, with additional smaller peaks or concentrations observed around ages 25, 32, and 55.

Conversely, the low-risk level (orange violin) exhibits a single pronounced peak centered around ages 15-25, skewed towards younger ages. This indicates that individuals in their late teens and early 20s are more likely to be categorized as low risk, with a secondary peak or concentration around ages 60-65, implying some older individuals may also fall into this category.

For the mid-risk level (green violin), the distribution for the mid-risk group is more spread out compared to the other two groups. It has a broad peak spanning ages around 19 to 30, indicating a wider age range for individuals in this risk category. Potential outliers can be observed in the tails, particularly around ages 45-50 and 55-60, which are higher ages for the mid-risk group.

Overall, the violin plot underscores distinct age distributions across different risk levels, with high risk being more prevalent among late teens and middle-aged individuals, low risk more common among younger individuals (late teens and early 20s) and some older individuals (around 60-65), and mid-risk levels spanning a wider age range but with a slight concentration in the late teens and early 20s.

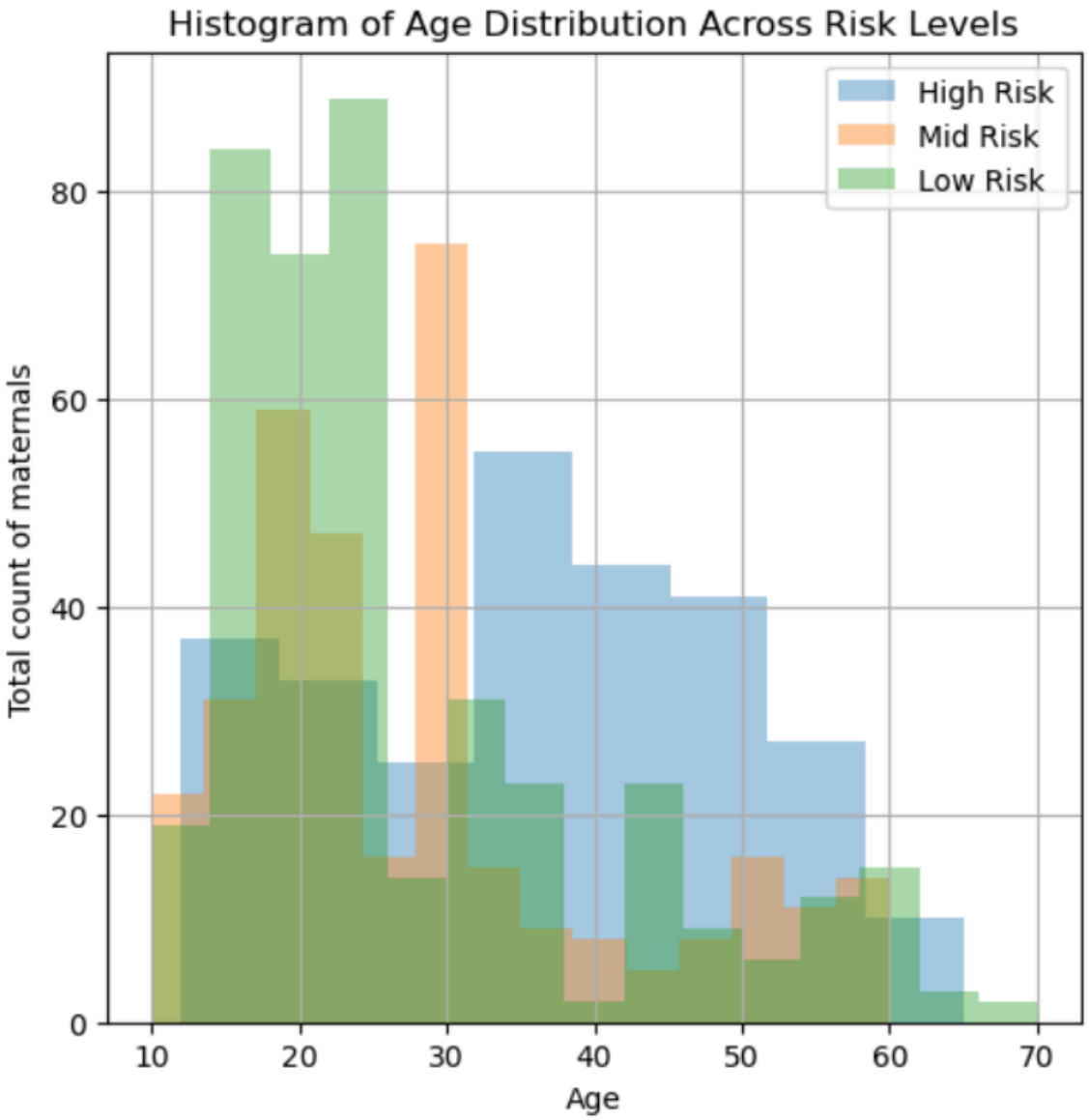


Figure 2.3.1.4: Histogram of age distribution across risk levels

The data collected strongly suggests that the distribution of risk levels across age groups in Bangladesh was significantly impacted by the prevailing COVID-19 pandemic. Throughout the year, Bangladesh witnessed multiple waves of COVID-19 cases, with notable peaks occurring around June-July and towards the year's end. There are several trends observed in the data that can be explained within the context of the pandemic. For high-risk levels, the bimodal distribution of high-risk levels, particularly with peaks among late teens (17-18) and middle-aged individuals (40-50), indicates the heightened vulnerability of these age groups to severe COVID-19 outcomes. Older adults and individuals with underlying health conditions were recognized as being at greater risk of COVID-19 complications, potentially elucidating the peak around ages 30-50. The peak among late teens may be attributed to factors such as increased

social interactions, which could contribute to higher transmission rates in this demographic. For low-risk levels, the concentration of low-risk levels among younger individuals, notably those in their late teens and early 20s, likely stems from their generally lower susceptibility to severe COVID-19 outcomes compared to older age groups. Additionally, the smaller peak observed around ages 60-65 for low-risk levels might be influenced by factors such as better adherence to precautionary measures or reduced exposure within this age group. For mid-risk levels, the broader distribution of mid-risk levels across age groups reflects the varying degrees of risk associated with different age demographics and their potential exposure or adherence to preventive measures. The slight peak observed around ages 19-23 might be linked to the heightened mobility and social interactions characteristic of this age group, thereby contributing to a higher risk of transmission. In essence, the data underscores the intricate interplay between age demographics and COVID-19 risk levels, shedding light on the complex dynamics shaping the pandemic's impact across different segments of the population in Bangladesh.

2.3.2 Systolic Blood Pressure

Systolic blood pressure (SBP) plays a crucial role in assessing maternal health risk during pregnancy. High blood pressure during pregnancy can be a sign of gestational hypertension or preeclampsia, which can cause dangers to the health of the mother and fetus, including preterm birth and damage to the mother's organs. It is also frequently accompanied by symptoms like proteinuria. Low SBP, on the other hand, could indicate insufficient blood supply to the placenta and uterus, which would impair embryonic growth. SBP is a critical parameter in prenatal care because healthcare providers regularly monitor SBP during pregnancy to detect issues early and intervene as necessary to enhance mother and fetal outcomes through lifestyle modifications or medication therapies.

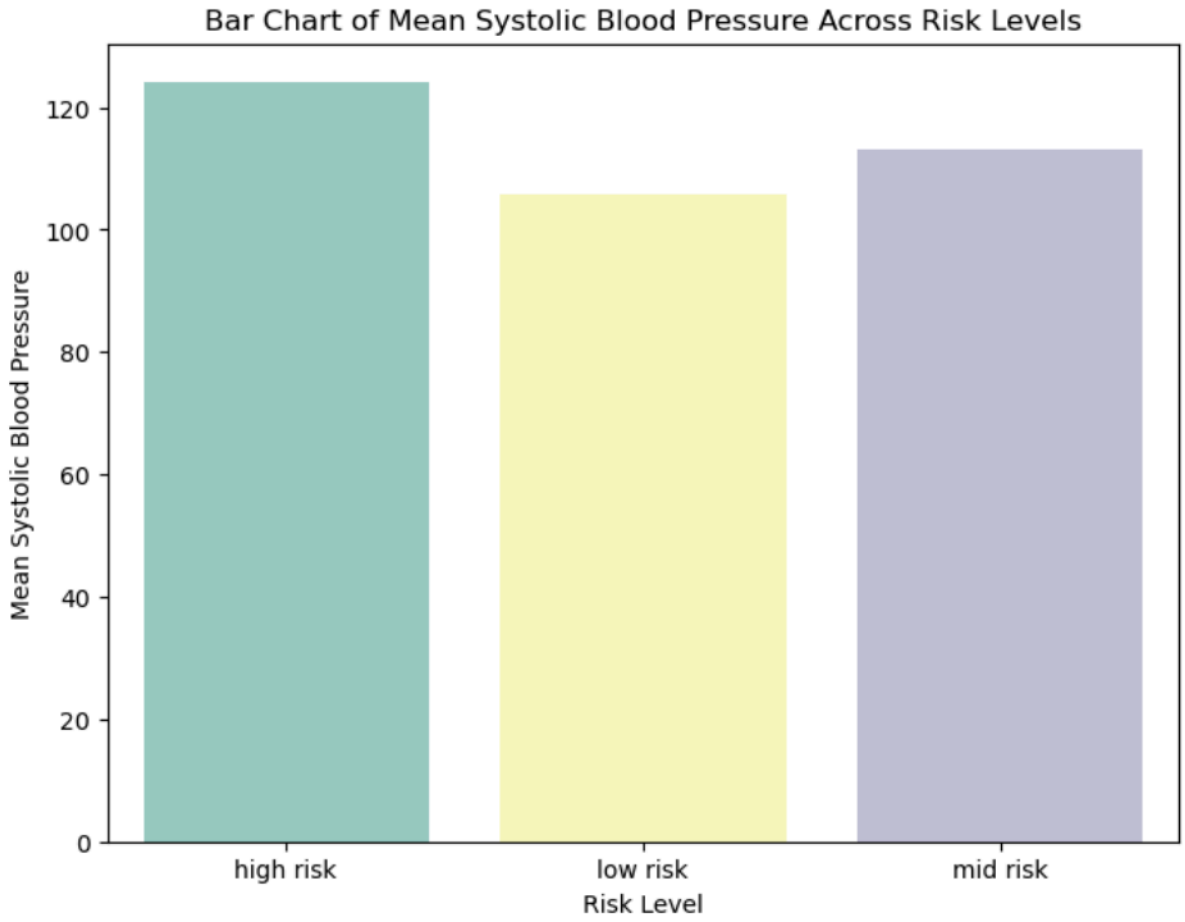


Figure 2.3.2.1: Bar chart of mean systolic blood pressure across risk levels

According to the bar chart, those who are at high risk have the highest mean systolic blood pressure, which is followed by those who are at mid- and low-risk. The values for mean systolic blood pressure are on the y-axis, and the heights of the bars show the relative differences.

The mean (average) of systolic blood pressure is frequently utilized in studies and charts for several reasons. First of all, it provides a succinct overview of total pressure, accurately representing the mean pressure over a specified time span despite daily variations in the natural world. This combined measure

makes it easier to compare values across groups or monitor changes in a maternal blood pressure over time. Furthermore, monitoring systolic pressure—the highest point reached during a heartbeat—provides important information about cardiovascular health, especially as people age. Researchers and medical experts can determine the total force applied to arteries during this crucial stage by measuring the mean systolic blood pressure, which helps with risk assessment and management. Furthermore, mean values are more useful for population-level analyses and group comparisons due to their comparability and simplicity, which offer a clear indication of the average systolic pressure within particular risk categories. All things considered, the average systolic blood pressure is an invaluable instrument for comprehending and observing cardiovascular health, providing information on patterns at the individual and community levels.

Systolic blood pressure typically recorded in millimeters of mercury (mmHg). A normal systolic blood pressure is less than 120 mmHg, according to the American Heart Association. The range of an increased systolic blood pressure is 120–129 mmHg. The range of stage 1 high blood pressure is 130–139 mmHg. 140 mmHg or greater is considered stage 2 high blood pressure.

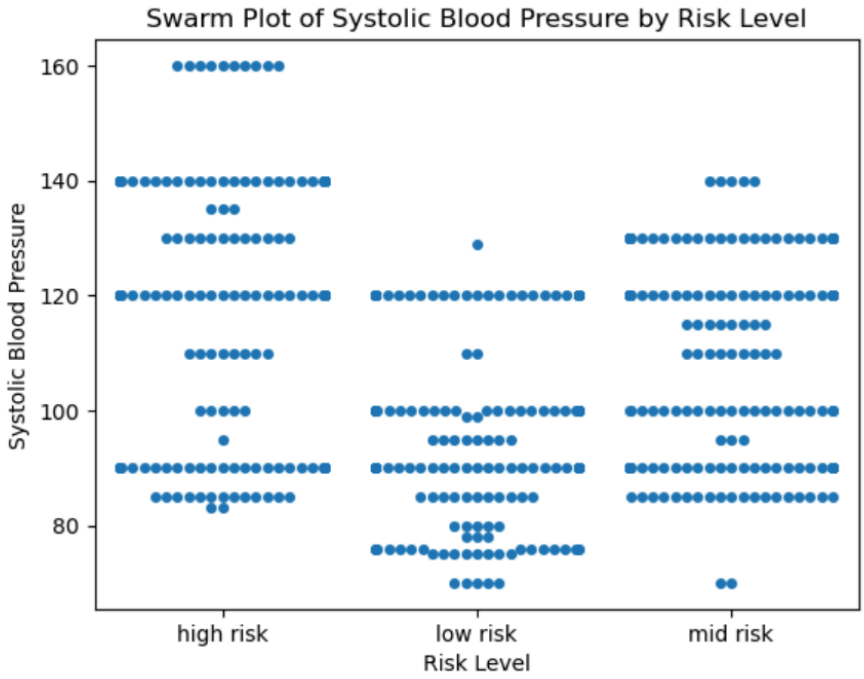


Figure 2.3.2.2: Swarm Plot if Systolic Blood Pressure by Risk Level

A swarm plot is a type of categorical scatter plot that displays individual data points along an axis based on categorical variables. It's particularly useful for visualizing the distribution of data points within different categories and identifying patterns or outliers. Each individual point is represented by a single point along the categorical axis. The density of points within each category provides insight into the

distribution of data. Denser areas indicate a higher concentration of data points, while sparser areas indicate fewer data points. Outliers may appear as individual points that are far removed from the main cluster of data points within a category.

In this plot, the two variables depicted are systolic blood pressure, which is the force that the blood exerts on the artery walls during a heartbeat, and risk level, which is a categorical variable that can be classified as high, medium, or low, indicate the likelihood of developing certain medical conditions like heart attacks or strokes. The plot observations show a general upward trend in the data, suggesting that the risk level tends to grow as systolic blood pressure rises, aligning with the established correlation between high blood pressure and elevated risks of heart disease and stroke. Still, there is plenty of variation in the data, as shown by the cases in which people with high blood pressure (Over 120 mmHg) also have low risk levels, thus it might have outliers. These blood pressure readings can fluctuate throughout the day and may be influenced by factors such as stress, physical activity, and medication. Additionally, the accuracy of blood pressure measurements can vary depending on the method used and the individual's condition at the time of measurement.

In comparison to the other two categories, the high-risk group typically has a larger systolic blood pressure distribution and a swarm of points that shift to the right, indicating greater systolic blood pressure readings, ranging around 100 to 160 mmHg.

The swarm plot indicates that the high-risk group may have a few outliers because some of the data points are located quite far from the group's main cluster.

There seems to be some overlap between the medium-risk group's systolic blood pressure distribution and the high- and low-risk groups' distributions. As a result, not all members of the high-risk group and low-risk group have high or low blood pressure, respectively. Overall, the swarm plot raises the possibility that risk level and systolic blood pressure are related. Systolic blood pressure values are often higher in the high-risk group than in the medium and low-risk categories.

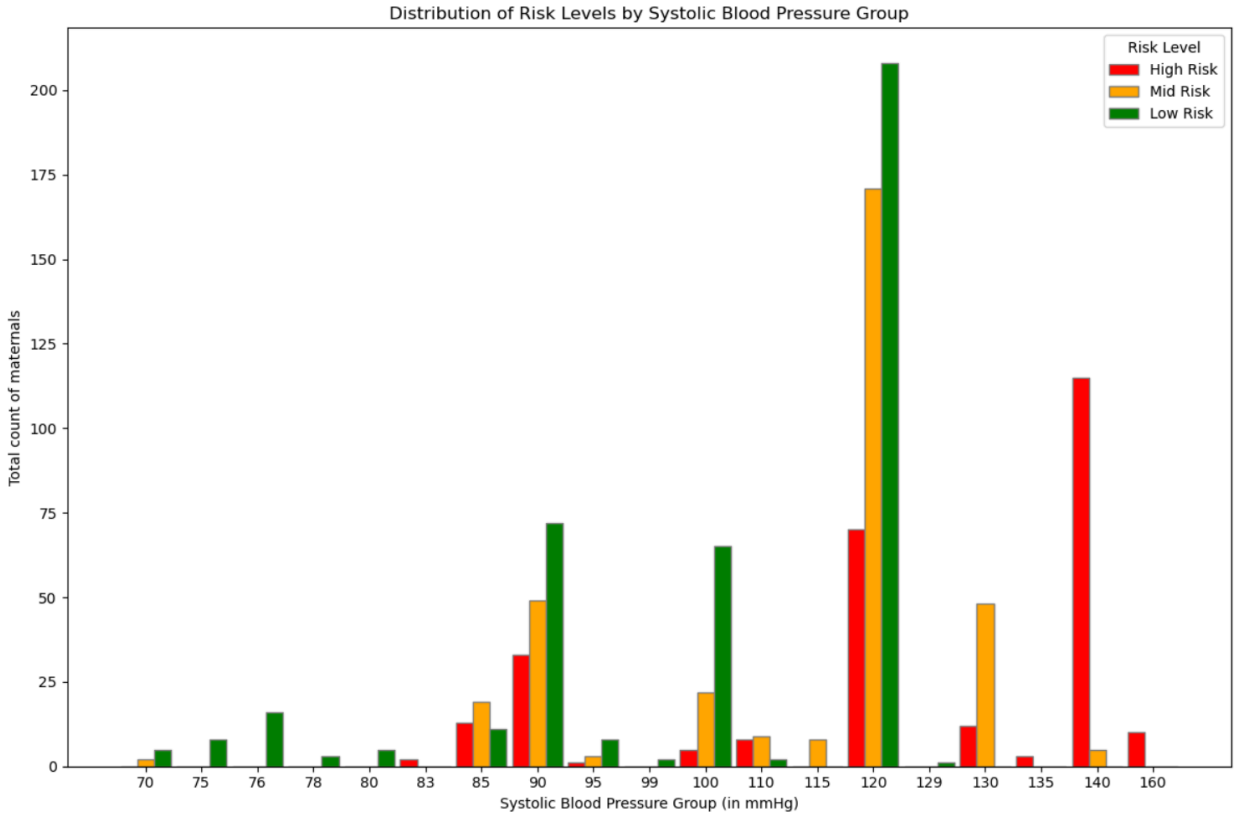


Figure 2.3.2.3: Distribution of risk levels by systolic blood pressure

The distribution of patients in different systolic blood pressure (SBP) groups and the corresponding risk levels are shown in the data. Patients classified as high, mid, or low-risk varied in frequency throughout the SBP groups, which ranged from 70 mmHg to 160 mmHg. There were often more mid and low-risk cases but fewer high-risk cases in lower SBP categories, such as 70 mmHg to 80 mmHg. On the other hand, the number of high-risk patients significantly increased when SBP spiked, especially over 90 mmHg. High-risk cases increased significantly in SBP groups 90 mmHg to 140 mmHg, with SBP 140 mmHg showing the greatest incidence. On the other hand, SBP groups over 140 mmHg demonstrated a decline in high-risk cases and mid-risk cases, particularly between SBP 140 mmHg and 160 mmHg.

From the data above, as systolic blood pressure increases, the proportion of people classified as high risk also increases.There is a higher proportion of low-risk pregnancies for lower systolic blood pressure readings which is around 120 mmHg.

Overall, the data suggests that higher systolic blood pressure is associated with a higher risk of complications during pregnancy.

Taking into consideration variables that might have affected healthcare and pregnancy outcomes in Bangladesh in 2020, the following hypothesis explains why such trends might appear:

- Healthcare Infrastructure: In Bangladesh in 2020, different areas and socioeconomic groups may have varying levels of awareness on maternal health and access to healthcare services. Pregnancy

problems, especially those involving raised SBP, may be more common in those with less access to healthcare services or less knowledge of the value of prenatal care.

- Lifestyle and nutrition: During pregnancy, socioeconomic factors might impact lifestyle and nutrition decisions. Pregnant people with lower socioeconomic status may be more susceptible to issues linked to increased SBP in Bangladesh, where dietary patterns and lifestyle factors may differ across different communities. These difficulties could include inadequate nutrition or restricted access to prenatal care.

- Environmental stressors: Pollution, crowded living conditions, and restricted access to clean water are a few examples of environmental elements that can raise stress levels and have an effect on the health of pregnant women. Bangladesh has to deal with issues relating to environmental deterioration in 2020, such as pollution of the air and water, which can raise stress levels and have an effect on maternal health.

- Public health interventions: Public health interventions can impact the identification and treatment of pregnancy-related complications, including those associated with elevated SBP. Examples of such interventions include prenatal screening programs and awareness campaigns about the value of antenatal care. These interventions are intended to improve maternal health outcomes.

2.3.3 Diastolic Blood Pressure

The pressure in arteries between heartbeats, or diastolic blood pressure (DBP), is an essential sign of a mother's health during pregnancy. Diastolic blood pressure is typically recorded in millimeters of mercury (mmHg). When DBP rises over normal limits, it indicates dangers such preeclampsia and gestational hypertension, which can endanger the health of the mother and the fetus. These risks include cardiovascular problems, premature delivery, and fetal growth limitation. The importance of DBP in evaluating maternal health during pregnancy is highlighted by the fact that routine prenatal care, including blood pressure monitoring, allows for early identification and management, which is essential for controlling these diseases and lowering related risks.

```
Overview of Diastolic blood pressure
----------------------------
count    1014.000000
mean       76.460552
std        13.885796
min        49.000000
25%        65.000000
50%        80.000000
75%        90.000000
max       100.000000
Name: Diastolic Blood Pressure, dtype: float64
----------------------------
```

Figure 2.3.3.1: Overview of diastolic blood pressure

The data provides a detailed look at the distribution and features of diastolic blood pressure (DBP) within the dataset. This dataset, which has 1,014 data points, is a large sample size that allows for reliable analysis. The mean DBP of approximately 76.46 mmHg serves as a central measure around which individual readings can be compared. In addition, the standard deviation, which is around 13.89 mmHg, shows how variable or dispersed the DBP readings are from the mean. A higher standard deviation indicates more variation in the results, which can be an indication of a range of medical issues or demographic characteristics within the sample group.

Analyzing the data further reveals that the maximum DBP of 100 mmHg is the greatest figure, while the minimum recorded DBP of 49 mmHg is the lowest. These high values provide important benchmarks for comprehending the dataset's DBP range. The quartile measurements also offer further context: the first quartile (Q1), or 25th percentile, is located at 65 mmHg, meaning that 25% of the data is below this number. Conversely, the third quartile (Q3), or 75th percentile, is situated at 90 mmHg, indicating that

75% of the data is below this value. This quartile data provides information about the lowest and higher ranges of the distribution, which aids in defining the spread of DBP values.

Additionally, the middle DBP, which is 80 mmHg, is a critical reference point, meaning that half of the recorded DBP readings are lower than this and half are higher. This median value supplements the mean measurement by providing a fair-minded viewpoint on the DBP central tendency within the dataset. In essence, this synopsis provides a thorough grasp of the distribution and features of DBP, providing essential data for evaluating cardiovascular health and possible risk factors in the population being studied.

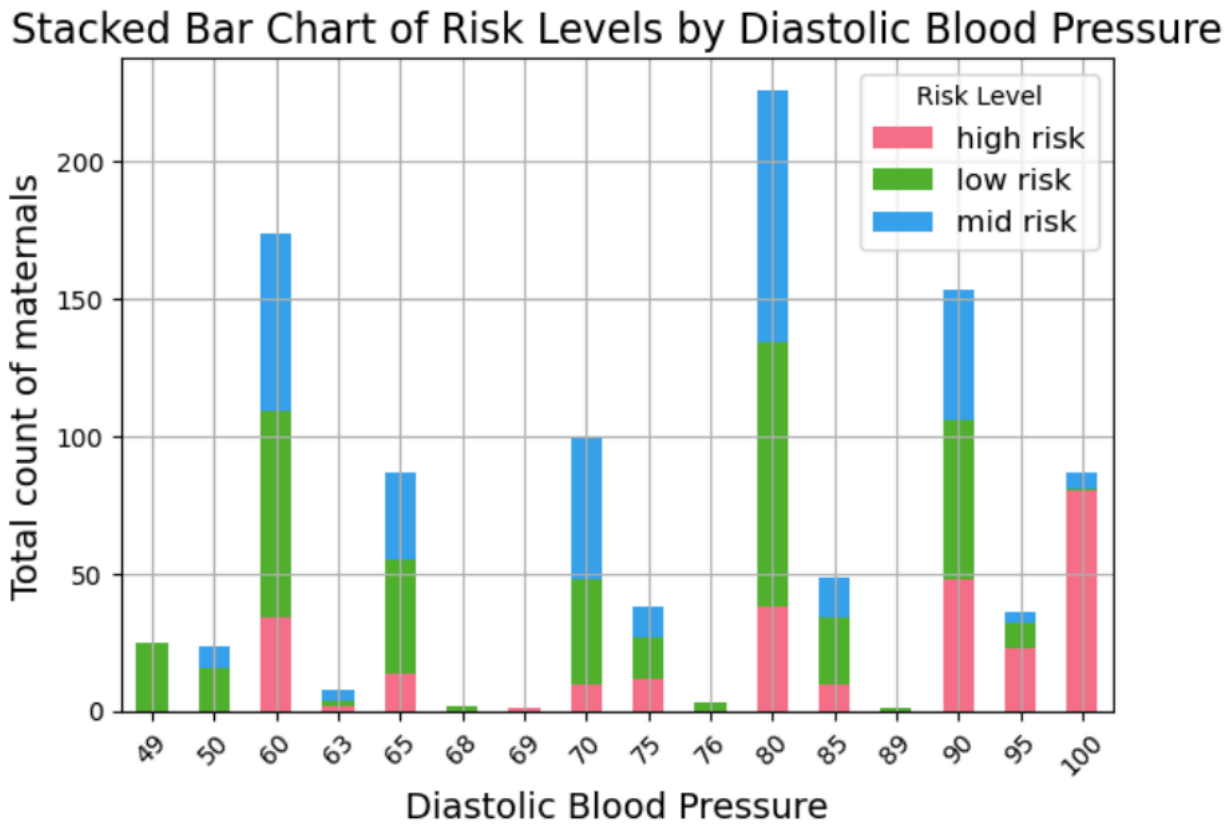


Figure 2.3.3.2: Stacked bar chart of risk levels by diastolic blood pressure

An important factor in determining cardiovascular health is the diastolic blood pressure, which is the lowest figure in a blood pressure measurement. Diastolic blood pressure values less than 80 mmHg are generally regarded as typically range in the context of risk assessment, indicating a decreased chance of cardiovascular problems. People who are considered to be at mid risk often have diastolic blood pressure readings between 80 and 89 mmHg, which indicates a moderate risk of cardiovascular problems and calls for lifestyle modifications and routine monitoring. Diastolic blood pressure readings of 90 mmHg or greater are indicative of high risk individuals, indicating an increased risk of cardiovascular issues that should be managed with prompt attention, lifestyle changes, and potentially medication. While this information is not 100% accurate as there are many factors that contribute to the variability maternals' health risk.

From the dataset provided, the diastolic blood pressure ranges between 49 and 100 encompass diverse proportions of individuals across various risk categories. Lower diastolic blood pressure groups, spanning from 49 to 76, are predominantly associated with the low-risk category, albeit with fluctuating counts between 60 to 76 mm Hg. The highest concentration within mid-range groups is observed at 60 mm Hg. Notably, at 68 mm Hg, there are 2 individuals classified as low risk, 1 as high risk at 69 mm Hg, and 3 as low risk at 76 mm Hg. Conversely, mid-range diastolic blood pressure groups, ranging from 80 to 89, depict a declining trend in individual counts. The peak within this range is at 80 mm Hg, which represents the highest bar in the overall distribution, housing 38 high-risk individuals, 92 mid-risk individuals, and 96 low-risk individuals. Additionally, there is 1 individual classified as low risk at 89 mm Hg. Finally, the high-risk diastolic blood pressure groups, spanning from 90 to 100 mm Hg. It shows a decreasing trend in individual counts overall. However, the number of high-risk individuals tends to rise from 48 at 90 mm Hg to 80 at 100 mm Hg.

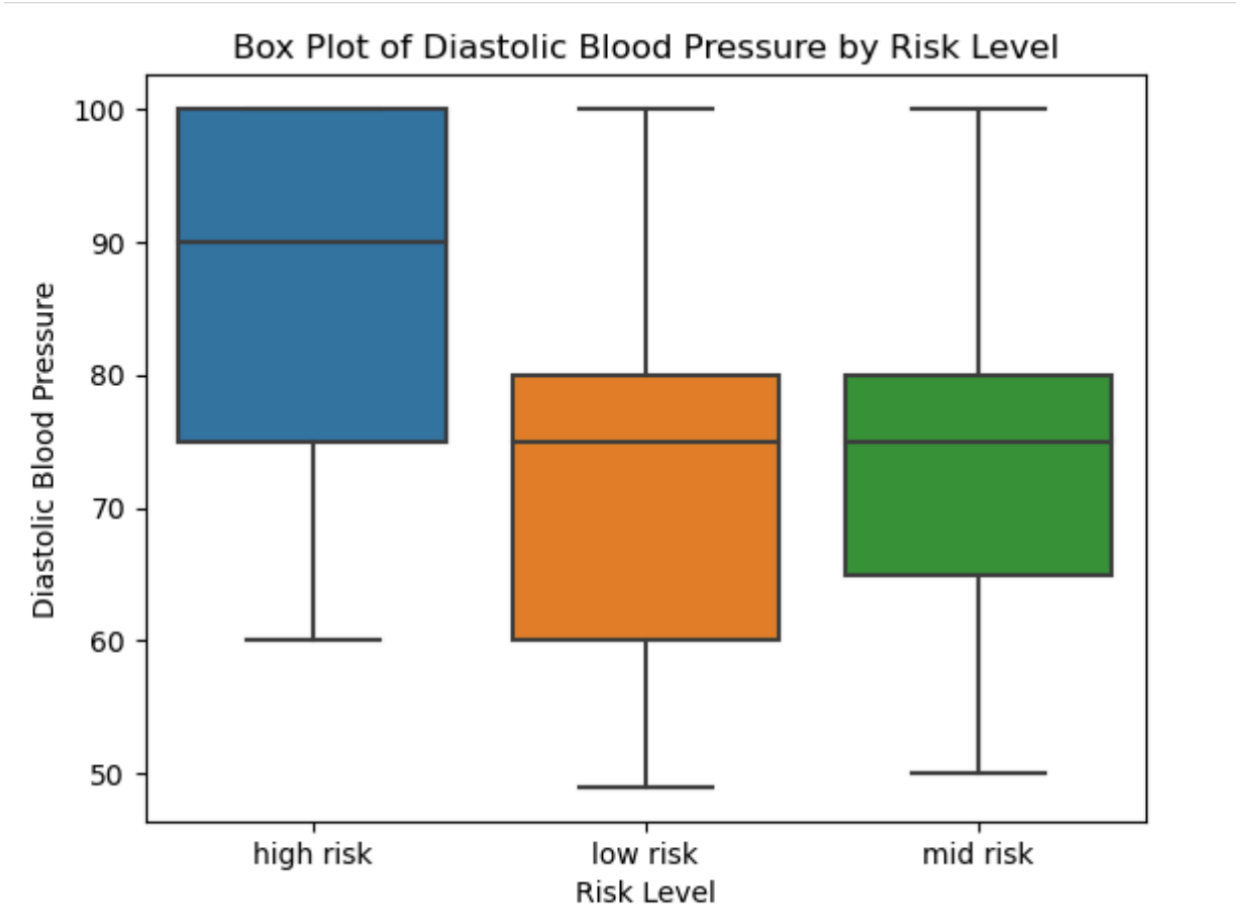


Figure 2.3.3.3: Box Plot of Diastolic Blood Pressure by Risk Level

Important findings from the boxplot show that there are significant variations in diastolic blood pressure between various risk groups. In particular, the higher median shown by the box's center line indicates that the high-risk group often has greater diastolic blood pressure than the medium and low-risk groups. This discrepancy in medians implies a statistically significant difference in diastolic blood pressure between

the high-risk group and the others. Furthermore, the higher-risk group had a larger diastolic blood pressure distribution, as seen by the box spans a larger range, from around 80mmHg to 100 mmHg, indicating higher variability within this group. The low-risk group's diastolic blood pressure values are typically lower, with the box covering a range of approximately 65-75 mmHg. In comparison to the group at high risk, the distribution is more compact. Regarding diastolic blood pressure levels, the mid-risk group lies between the high-risk and low-risk categories. The box range, which is roughly 67–77 mmHg, is comparable to the low-risk category but somewhat higher.

Notably, for any of the risk level groupings, this box plot shows no obvious outliers. Typically, individual data points outside of the whiskers—vertical lines that extend from the boxes—would be used to depict outliers. But in this plot, every piece of data appears to be contained inside the whisker range, suggesting that there aren't any extreme values or outliers in the data.

The observed differences in diastolic blood pressure between various risk groups in Bangladesh in 2020 could be caused by a number of variables. First, differences in socioeconomic status could be a major factor. Bangladesh features a wide range of socioeconomic classes, including differences in the availability of jobs, healthcare, and educational possibilities. People from lower socioeconomic origins may be more stressed out, have less access to healthcare, and have inferior eating habits, all of which can lead to high blood pressure.

Furthermore, there are significant genetic predispositions and familial histories of hypertension that can influence the differences in blood pressure readings between various demographic groups. High blood pressure, or hypertension, frequently runs in families, suggesting that there may be a genetic component to the illness. People who have a family history of hypertension are more likely to get the illness themselves. This familial tendency may be explained by common genetic variables, such as those involved in the renin-angiotensin-aldosterone system (RAAS), which is essential for blood pressure management.

Furthermore, genetic variants can influence an individual's response to environmental factors that are known to contribute to hypertension, such as stress, nutrition, and physical exercise. For instance, some genetic variations may make a mother more sensitive to the amount of sodium in food, which could result in an increase in blood pressure while eating a diet heavy in salt.

2.3.4 Blood Sugar

From Figure 2.3.4.1 below, the first column indicates the blood sugar levels while the second column represents the frequency of maternals who have blood sugar levels falling within the corresponding range.

```
Blood Sugar
7.50     176
6.90     113
6.80      88
7.00      79
7.90      60
15.00     54
6.10      53
11.00     52
7.80      45
6.70      33
9.00      31
18.00     29
7.70      24
19.00     22
8.00      22
6.00      21
7.20      20
12.00     18
16.00     17
7.01      15
6.40      10
13.00      9
7.10       8
17.00      5
10.00      4
6.30       2
6.60       2
6.50       1
7.60       1
Name: count, dtype: int64
```

Figure 2.3.4.1: Frequency of maternals with corresponding blood sugar levels

Maternal blood glucose level, often known as blood sugar level, is a vital component of fetal health throughout pregnancy. This measurement relates to the amount of glucose present in a pregnant woman's blood. Blood sugar control must be done correctly since it has an immediate effect on the health of the mother and the fetus. High blood sugar levels can result in difficulties like preeclampsia, premature birth, and delivery issues. These disorders are frequently linked to elevated blood sugar levels, such as pre-existing diabetes or gestational diabetes. Low blood sugar, on the other hand, can result in weakness, dizziness, and even injury to the mother and the unborn child. Furthermore, an uncontrolled diabetic fetus is more likely to have birth abnormalities.

From the dataset, variability in blood glucose levels may be a sign of changes in metabolic health in a mother whose blood sugar ranges from 6 to 19 mmol/L (millimoles per liter). An impaired fasting glucose level, indicated by a fasting blood sugar level between 6 and 6.9 mmol/L (108 and 125 mg/dL), may call for additional research or lifestyle changes in order to prevent diabetes. In order to prevent difficulties, postprandial levels, which can fluctuate greatly based on carbohydrate intake, should ideally stay below

7.8 mmol/L (140 mg/dL). Further gestational diabetes screening may be required if blood sugar levels between 6 and 7.9 mmol/L (108 and 142 mg/dL) one hour after a glucose solution ingestion are present throughout pregnancy. Diabetes may be indicated by consistently high levels exceeding 7 mmol/L (126 mg/dL) when fasting or 11.1 mmol/L (200 mg/dL) two hours after a meal. Diagnosis and treatment of this condition require medical attention. The maintenance of stable blood sugar levels and the avoidance of related health concerns necessitate regular monitoring, lifestyle modifications, and medical advice.

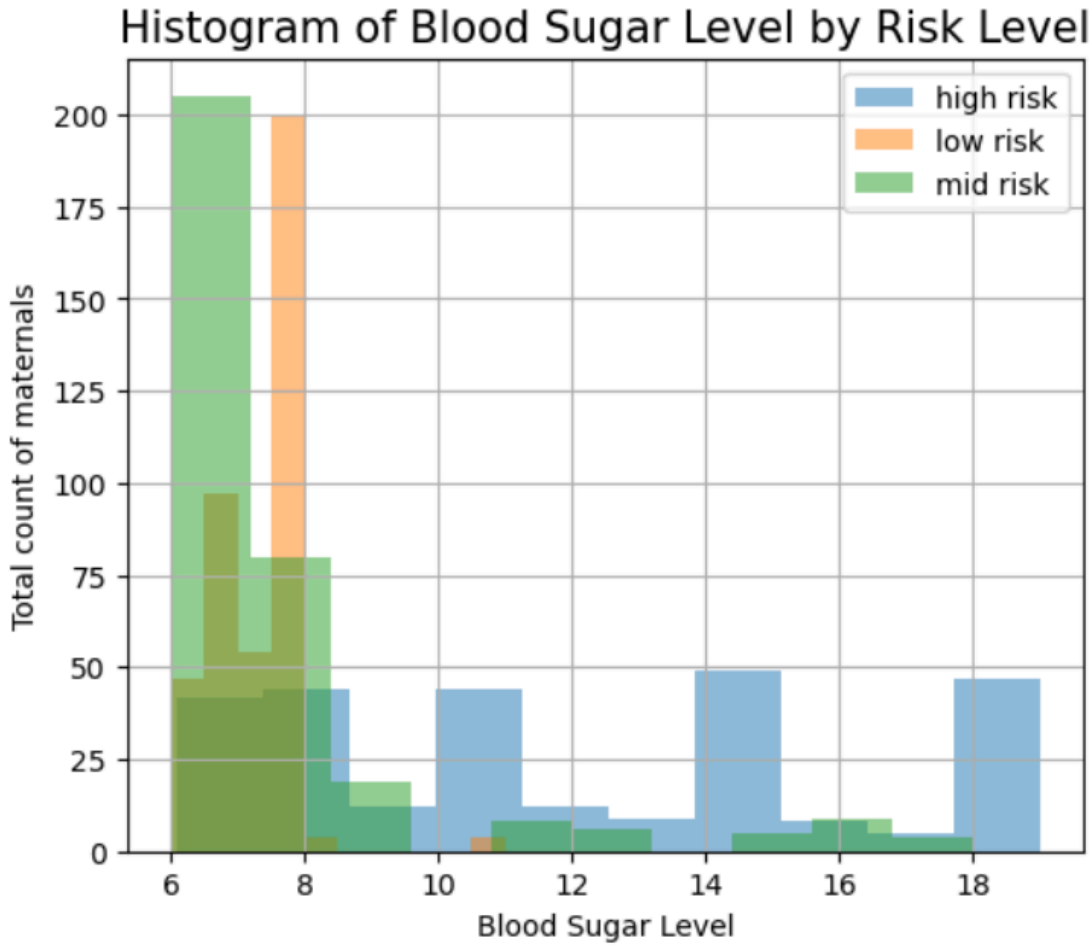


Figure 2.3.4.2: Histogram of blood sugar level by risk level

Figure 2.3.4.2 is a histogram with blood sugar levels ranging from 6.0 to 19.0 mmol/L. The stacked bar shows the possible low-risk range, where most people are categorized as low or mid risk, for blood sugar levels between 6.0 and 8.0 mmol/L. There are 129 people at low risk, 36 at mid risk, and 11 at high risk at a peak of 7.5 mmol/L. Similarly, since so many people fall within this range, the hypothetical mid-risk zone is between 6.0 and 10.0 mmol/L. The greatest count of 6.9 mmol/L blood sugar level which is 58 people from mid risk level, closely followed by 41 people of mid risk level at 6.8 mmol/L and 7.0 mmol/L. On the other hand, the possible high-risk category covers blood sugar readings between 10.0 and 19.0 mmol/L. Notably, the highest count of individuals at high risk, 40 in total, is observed at a blood sugar level of 11.0 mmol/L, while the second highest count, 25 individuals, is recorded at 18.0 mmol/L.

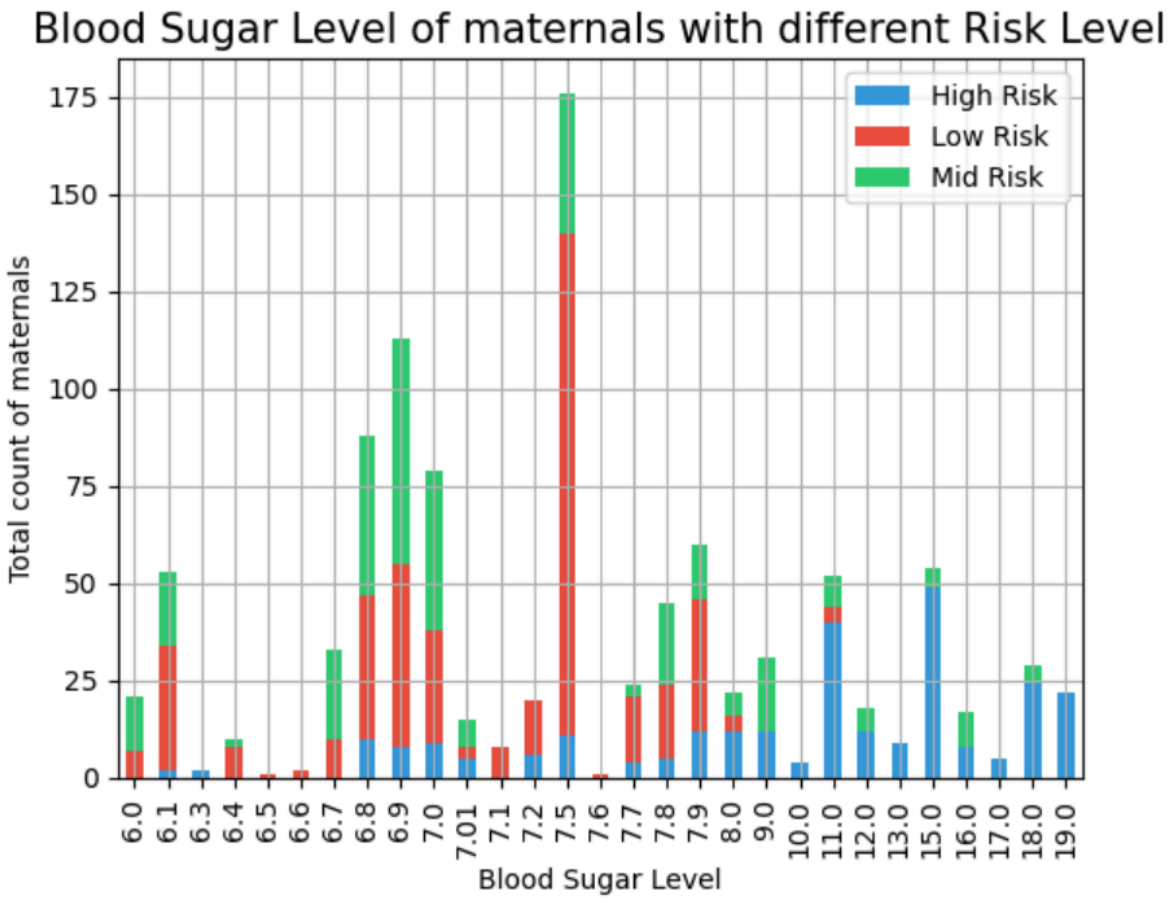


Figure 2.3.4.3: Blood sugar level of maternals with different risk level

According to Figure 2.3.4.3 presented, there will be multiple trends in blood sugar levels and related risk categories among Bangladeshi patients in 2020. A common blood sugar range may be reflected in the peak patient count of 7.5 mmol/L across all risk categories. This range may be influenced by dietary practices, lifestyle choices, and genetic predispositions that are widespread in Bangladesh.

Effective preventative approaches or early interventions in managing blood sugar levels within this range may be indicated by the relatively constant number of low-risk patients across blood sugar levels from 6.0 to 11.0 mmol/L. This rise may have been influenced by health education initiatives, greater accessibility to healthcare, and raised knowledge of the risk factors for diabetes.

The peak of 6.9 mmol/L for the moderate-risk patient count indicates a significant percentage of people whose blood sugar levels are getting close to the diagnosis standard for diabetes. This may be related to lifestyle factors that are linked to an increased risk of diabetes, such as eating habits, inactivity, and urbanization resulting in sedentary lifestyles.

There could be a number of reasons for the low number of moderate-risk individuals at the highest blood sugar levels. The underrepresentation in the data may be due to the fact that people with very high blood sugar levels may already be receiving treatment after being diagnosed, or they may encounter obstacles in their quest for healthcare services.

The much higher number of high-risk patients at 15.0 mmol/L might point to a key point at which people are more susceptible to serious consequences from uncontrolled diabetes. This may be impacted by things like a delayed diagnosis, restricted access to medical care, and difficulties treating the disease in its severe stages.

Amidst the COVID-19 pandemic in 2020, Bangladesh might encounter obstacles concerning healthcare infrastructure, medical service accessibility, and public health campaigns. These difficulties might impact on the patterns found in the data, emphasizing how crucial it is to deal with structural problems and put specific interventions in place in order to manage diabetes and reduce related risks in the population.

2.3.5 Body Temperature

```
Overview of Body Temperature
----------------------------
count    1014.000000
mean       98.665089
std         1.371384
min        98.000000
25%        98.000000
50%        98.000000
75%        98.000000
max       103.000000
Name: Body Temperature, dtype: float64
----------------------------
```

```
Body Temperature
98.0     804
101.0     98
102.0     66
100.0     20
103.0     13
99.0      10
98.4       2
98.6       1
Name: count, dtype: int64
```

Figure 2.3.5.1: Overview statistics of body temperature and frequency distribution of temperature readings

Figure 2.3.5.1 above provides a thorough summary of body temperature readings, including frequency distribution of temperature readings and summary statistics. A notable degree of variability around the mean is indicated by the 1014 observations that were initially recorded, which have an average body temperature of roughly 98.67 degrees Fahrenheit and a standard deviation of roughly 1.37 degrees Fahrenheit. The temperature range is 98.0 degrees Fahrenheit

at minimum and 103.0 degrees Fahrenheit at maximum, the spread of the distribution is indicated by the quartile values.

Furthermore, the frequency distribution section offers a detailed breakdown of how frequently each temperature reading occurs. For instance, temperatures of 98.0 degrees Fahrenheit dominate the dataset, observed 804 times, while other temperatures have varying frequencies, reflecting the diversity of temperature measurements collected.

Upon analysis, it is clear that the majority of body temperatures (around 79%) that have been recorded fall between 98.0 and 98.6 degrees Fahrenheit. The dataset reveals that 98.0 degrees Fahrenheit is the most often observed temperature, indicating its widespread occurrence. Higher temperatures, on the other hand, like 102.0 and 103.0 degrees Fahrenheit, happen less frequently but are still there in the data. This indicates potential outliers or abnormal observations, suggesting a degree of variability beyond the typical range.

This data provides important information on the distribution and features of body temperature readings, clarifying normal temperature ranges and pointing out any potential anomalies or odd trends. Such information can prove instrumental in understanding normal temperature variations and identifying deviations that may warrant further investigation or medical attention.

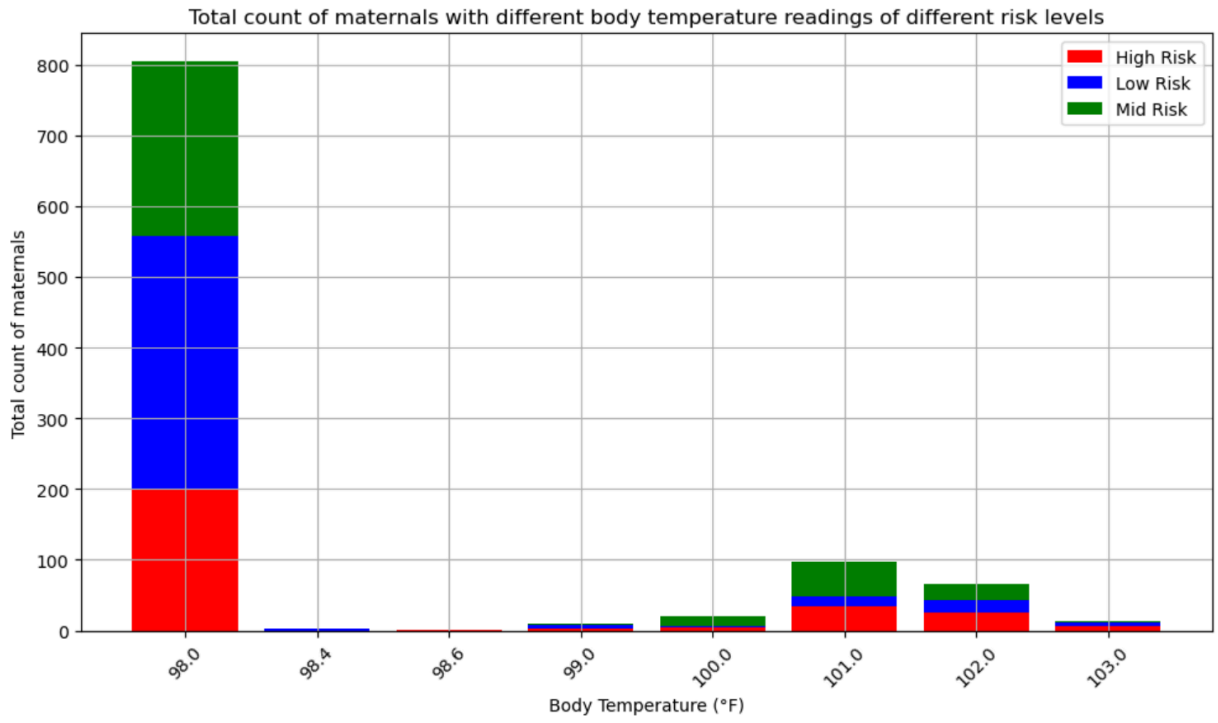


Figure 2.3.5.2: Stacked bar chart showing total count of maternal with different body temperature readings of different risk levels

The stacked bar chart above serves as an apt choice for visualizing the provided data due to its ability to illustrate the frequency counts associated with distinct risk categories (high, low, mid) across various body temperature levels. This graphical representation facilitates a comprehensive comparison, aiding in the discernment of patterns and trends. For instance, at a body temperature of 98.0°F, the chart reveals a prevalence of low-risk cases, followed by mid-risk occurrences and then high-risk instances. Moreover, as the temperature incrementally rises from 98.4°F to 99.0°F, the counts across all risk levels demonstrate a relative diminution.

Furthermore, the visualization delineates noticeable shifts in risk level frequencies with changes in body temperature. At 100.0°F, there is a discernible surge in mid-risk cases, accompanied by a lesser prevalence of high and low-risk scenarios. This trend intensifies at 101.0°F, where mid-risk instances dominate, followed by high-risk and subsequently low-risk cases. As the temperature reaches 102.0°F, the frequencies exhibit a slight decline, with high-risk cases ranking second after mid-risk occurrences. Lastly, at 103.0°F, a significant decrease in overall frequencies is observed, with high-risk cases emerging as the most prevalent, trailed by low-risk and mid-risk instances.

Therefore, the stacked bar chart effectively elucidates these nuanced trends and variations, offering a lucid depiction of how risk levels evolve with fluctuating body temperatures. By providing a visual snapshot of the data, it facilitates comprehensive analysis and interpretation, empowering researchers to gain deeper insights into the relationships between body temperature and associated risk levels.

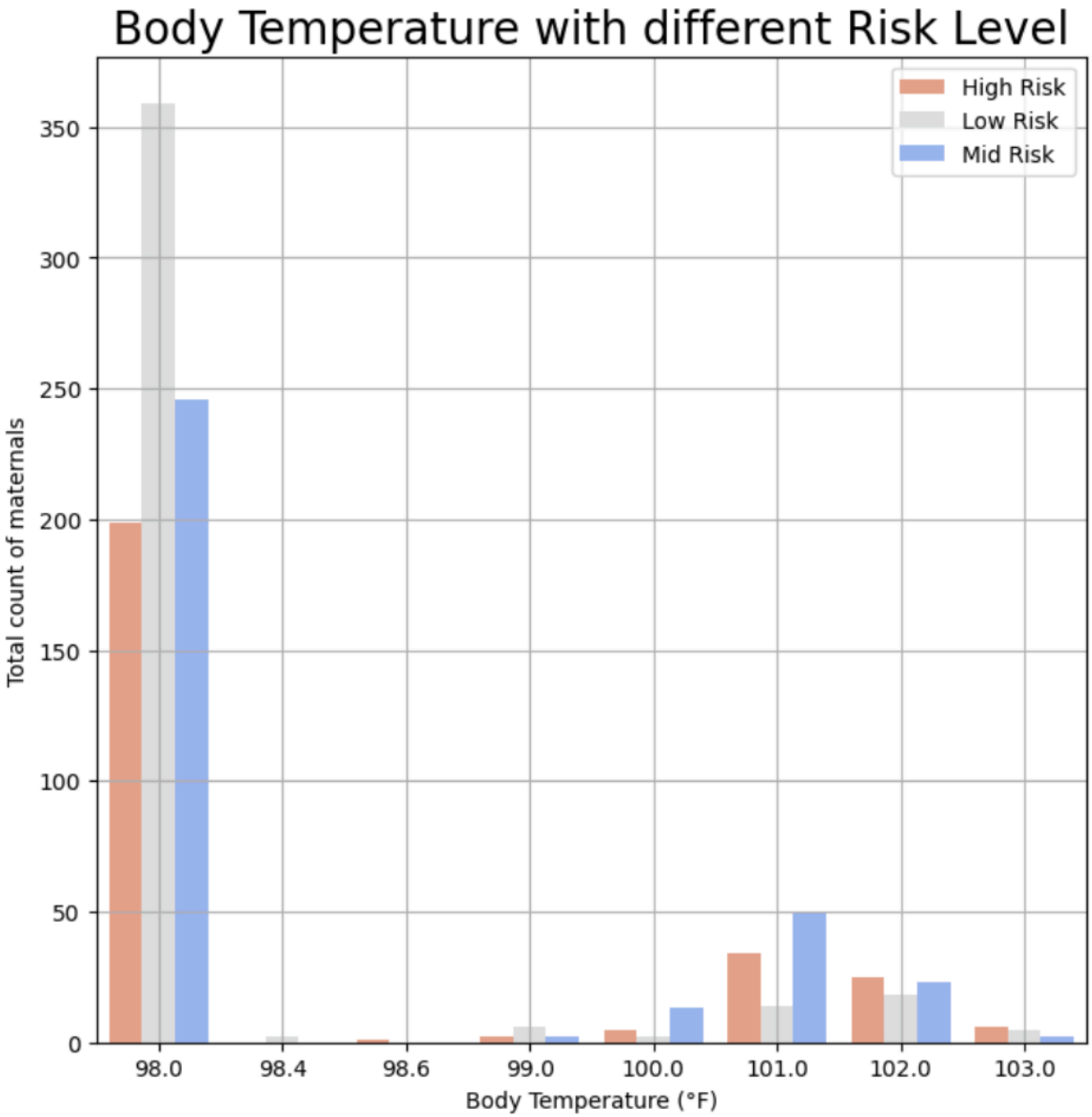


Figure 2.3.5.3: Bar chart shows body temperature with different risk level

There are a number of reasons why fewer people are categorized as high risk as body temperature increases. First and foremost, it's critical to understand that the average body temperature of a human being is within a very small range, normally 98°F. If the temperature deviates from this range, it may be a sign of physiological stress or disease, which would cause the risk categories to change. Furthermore, the distribution of maternal health hazards may have been impacted by seasonal differences in temperature and weather. These factors may have also contributed to variations in body temperature.

In addition, the observed data distribution may have been influenced by modifications to the healthcare system and public awareness programs launched in 2020 by the government of Bangladesh and a number of non-governmental organizations (NGOs) to inform people about COVID-19 preventative methods. The frequency of check-ups and early diagnosis of health disorders may have been affected by initiatives aiming at strengthening access to prenatal care, improving maternal healthcare facilities, and increasing public knowledge of health hazards during pregnancy. Moreover, maternal health outcomes may have been impacted by the public health policies and interventions that were put into place throughout the year. Disparities in the number of people included in the dataset at different body temperatures and risk levels may also be caused by differences in data collection techniques, reporting standards, and the dependability of healthcare in Bangladesh's various areas.

2.3.6 Heart Rate

An important measure of the health of both the mother and the fetus during pregnancy is the mother's heart rate. Variations in this range, which typically range from 60 to 100 beats per minute (bpm), are typical in pregnant women. Disturbances from this average, nevertheless, could indicate underlying medical problems. When a woman's heart rate is higher than 100 beats per minute, there may be potential dangers to the fetus and mother due to conditions including anemia, dehydration, or infections. On the other hand, although less common during pregnancy, a persistently low heart rate (below 60 bpm) may suggest diseases such as hypothyroidism or heart rhythm issues, necessitating additional evaluation.

Throughout pregnancy, maternal heart rate monitoring provides important information about the health of both the mother and the fetus. Typically raised, with a normal range of 60 to 100 bpm, variations from this range may suggest potential risks. An elevated heart rate (over 100 bpm) might indicate a number of health problems, such as infections or dehydration, whereas a persistently low heart rate (below 60 bpm) can indicate hypothyroidism or cardiac rhythm problems. Knowing these differences allows for prompt intervention to protect the developing fetus and the mother.

```
Overview of Heart Rate
--------------------
count    1014.000000
mean       74.301775
std         8.088702
min         7.000000
25%        70.000000
50%        76.000000
75%        80.000000
max        90.000000
Name: Heart Rate, dtype: float64
--------------------
```

Figure 2.3.6.1: Overview statistics of heart rate

The dataset provided shows a thorough overview of heart rate readings from a representative sample of people. The dataset, which contains 1014 recorded observations, provides important information about the distribution of heart rates. As a measure of central tendency, the average heart rate in the sample is represented by the mean heart rate of roughly 74.3 beats per minute (bpm). Furthermore, the heart rate variability or dispersion around the mean, indicated by the standard deviation of approximately 8.1 bpm, suggests the degree of spread across the sampled individuals.

The distribution of heart rates can be further understood by looking at particular percentiles. For instance, the 25th percentile reveals that a quarter of the observations have a heart rate of 70 bpm or lower, providing insight into the lower range of heart rates in the dataset. On the other hand, the upper range of heart rates is shown by 75th percentile, which shows that 75% of the observations had a heart rate of 80 bpm or less. Extreme numbers, such as the maximum of 90 bpm and minimum of 7 bpm, may represent outliers or unusually high or low heart rates in certain circumstances, necessitating additional research into their validity within the dataset.

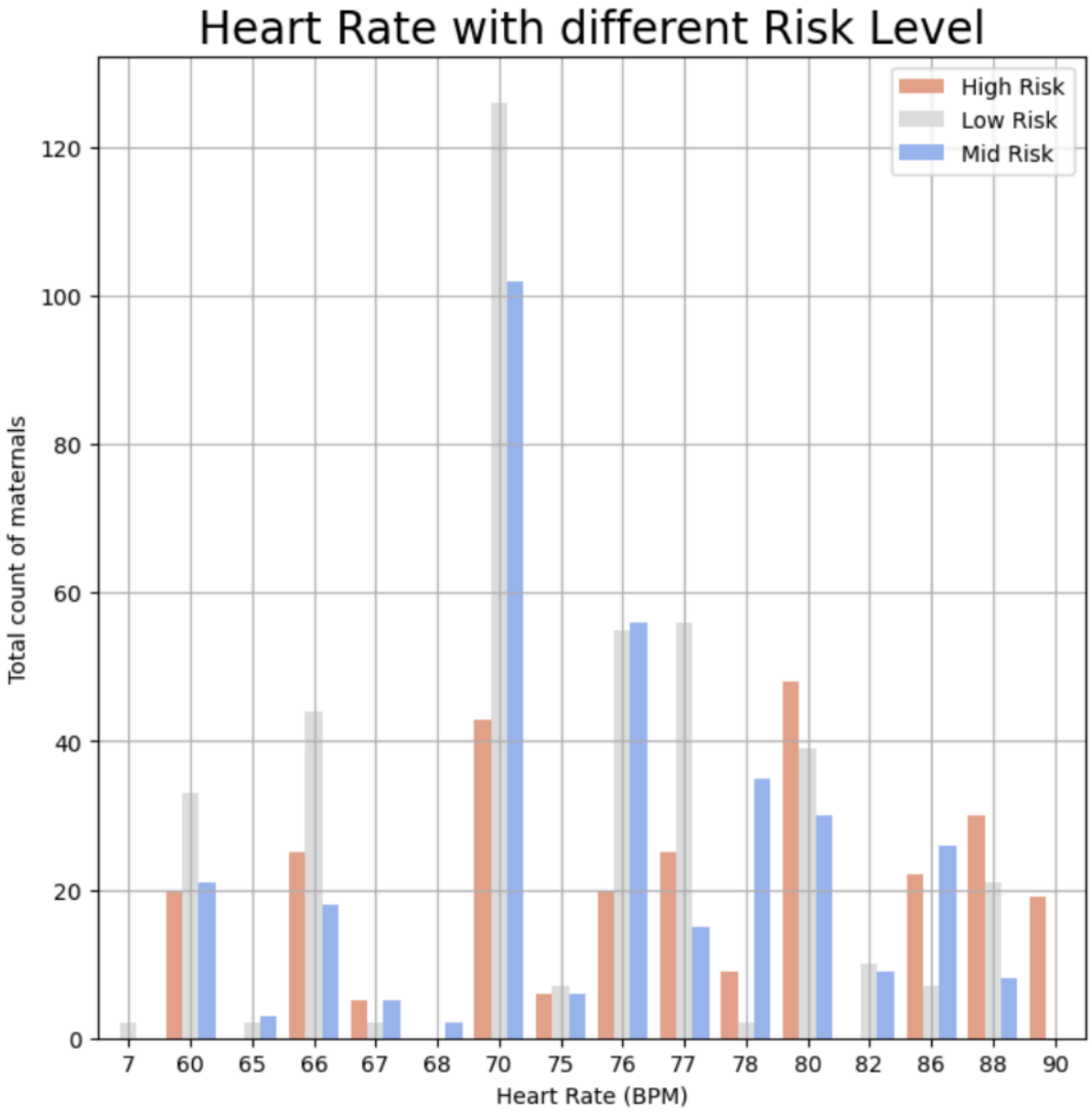


Figure 2.3.6.2: Bar chart shows heart rate with different risk level

The relationship between different risk levels and heart rate is shown in the diagram "Heart Rate with different Risk Level". "Heart Rate (BPM)" is represented by the x-axis, which runs from 0 to 100, and "Number of Patients" is represented by the y-axis, which runs from 0 to 120. A line plotted with markers representing the data points listed in the preceding table represents each risk category (high, low, and mid).

The graph's observations show clear patterns among the risk levels. With heart rates rising, the high-risk line often shows an upward trend, albeit there are occasional fluctuations. For example, about 20 people are considered high risk with a heart rate of 60, and this number rises to about

48 at a heart rate of 80, with some exceptions, such as a drop to only 6 patients at a heart rate of 75. On the other hand, there is no obvious connection between heart rate and the wave-like patterns displayed by the mid-risk and low-risk lines. For example, at a heart rate of 60 bpm, the number of patients classed as low risk drops to about 33, and at 65 bpm, it jumps rapidly to approximately 126. Similar fluctuations can be seen in the mid-risk line, where the number of patients assigned to this category goes from around 21 at a heart rate of 60 to about 102 at 70 bpm, and then back down to about 9 at 82 bpm.

In conclusion, the data for the low and mid-risk categories is ambiguous, showing unpredictable oscillations without a recognizable pattern in relation to heart rate, even though the graph suggests a probable association between heart rate and the high-risk categorization.

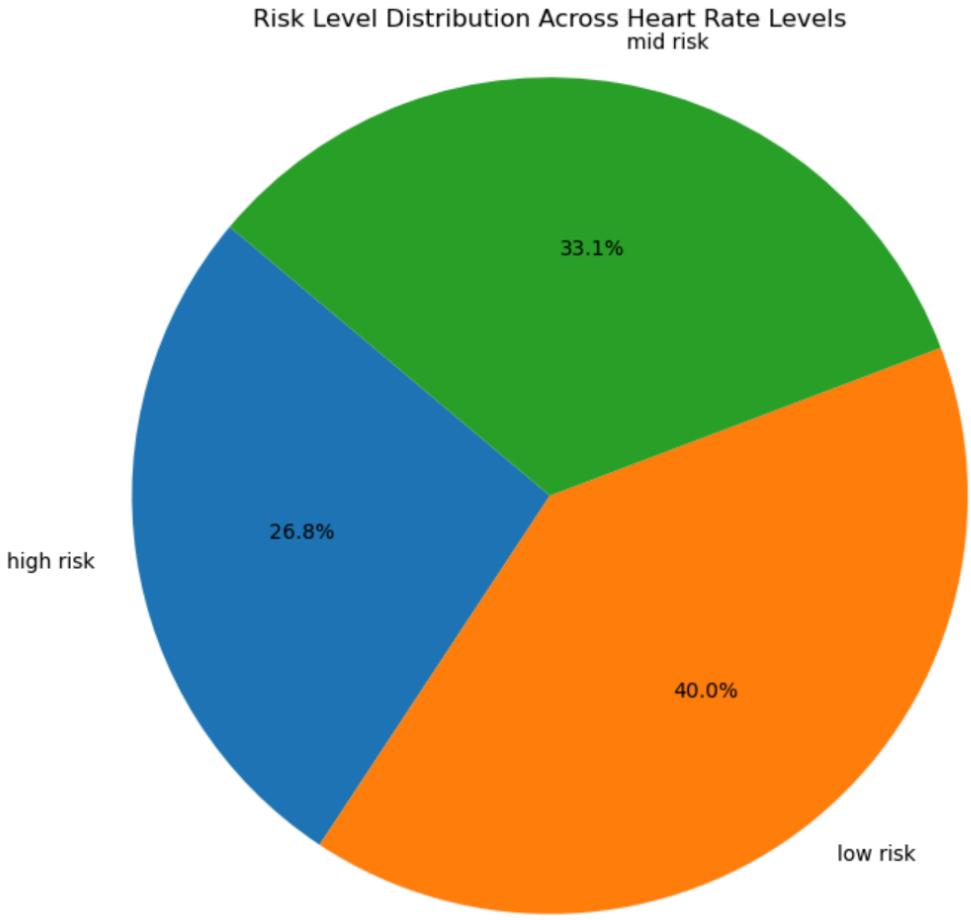


Figure 2.3.6.3: Pie chart of risk levels distribution across heart rate levels

Based on the pie chart showing the risk level distribution across heart rate levels in 3 risk categories, it seems that the percentages are 26.8% for high risk, 40.0% for low risk, and 33.1% for mid risk.

One potential explanation for this distribution could be related to the COVID-19 pandemic situation in Bangladesh during 2020. The COVID-19 pandemic had a significant impact on healthcare systems and patient populations worldwide, including Bangladesh.

The high percentage (40.0%) of low-risk patients could be attributed to individuals who were relatively healthy or had mild symptoms and did not require extensive medical intervention. On the other hand, the 26.8% of high-risk patients could represent those with severe COVID-19 cases, underlying health conditions, or complications that required intensive care or hospitalization.

The mid-risk category, comprising 33.1% of the patients, may have included individuals with moderate symptoms or pre-existing conditions that put them at a higher risk but did not necessarily require intensive care. This group may have required closer monitoring or additional medical support.

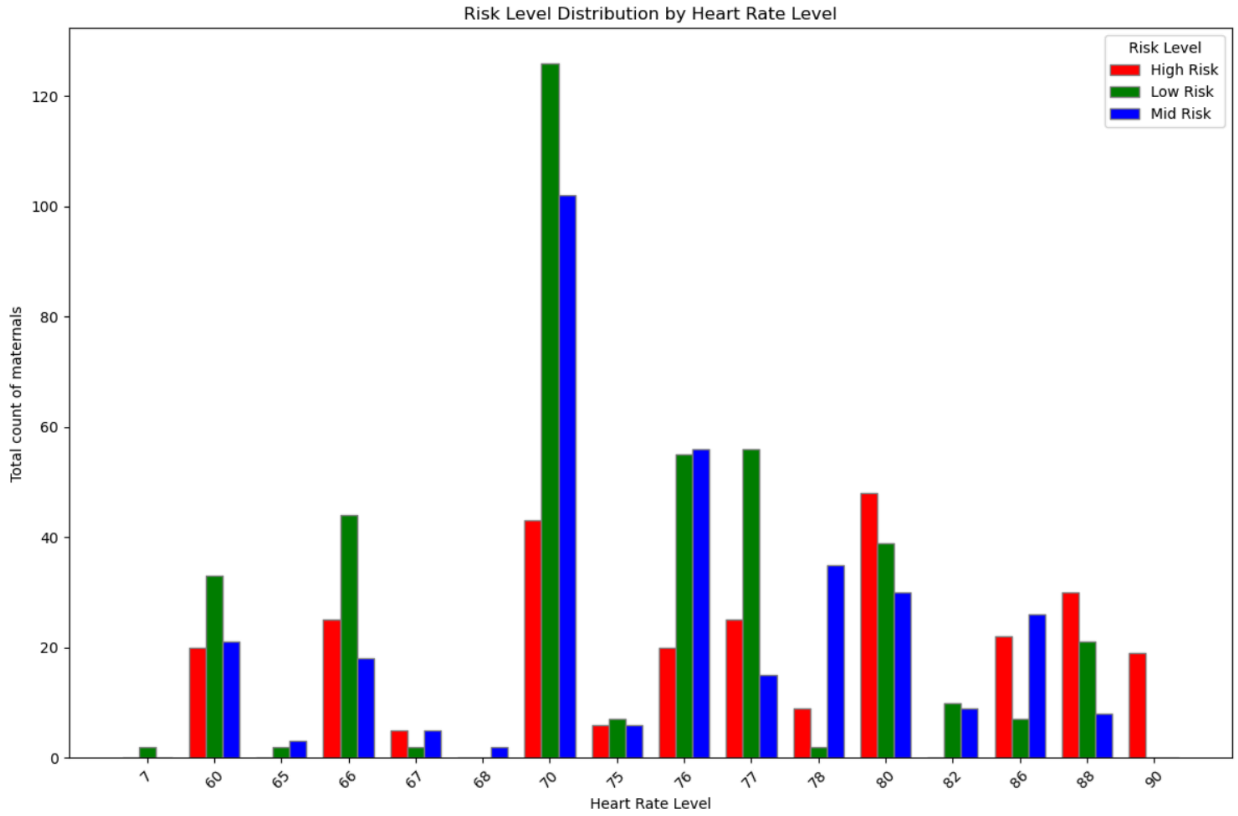


Figure 2.3.6.4: Risk level distribution by heart rate level

Finding outliers in this dataset requires an in-depth examination of the dataset's extreme values. Particularly, heart rates deviating significantly from the expected range for pregnant women,

such as the recorded 7 bpm, are likely erroneous entries or anomalies. These outliers are biologically impossible during pregnancy.

In a similar vein, although high maternal heart rates are uncommon, abnormally high readings such as the observed 90 bpm may also point to outliers that require further investigation. These elevated heart rates may indicate maternal discomfort or underlying health difficulties, however they are not as obvious as the abnormally low rates. Such outliers require close examination and additional research due to their possible influence on maternal well-being.

Furthermore, it is imperative to place these high results in the perspective of the dataset's overall maternal count. Even if they are outliers, they might not necessarily indicate anomalies in our dataset if a sizable percentage of individuals have similarly excessive heart rates. For proper risk assessment, it is therefore crucial to understand the distribution and prevalence of heart rates among the patient population in addition to detecting outliers.

2.4 Dependent Variable

The variable of interest that is impacted by changes in the independent variables in this dataset is the dependent data, which is risk level. The likelihood or seriousness of unfavorable health outcomes or difficulties for maternal health is represented by the risk level. Usually, it is determined using a number of variables, including but not restricted to the independent variables like body temperature, heart rate, blood pressure, and blood sugar. Clinical evaluation, risk scoring schemes, or predictive models that take into account the interaction of several variables impacting maternal health are frequently used to determine the risk level. The risk level may also vary in concert with changes in the independent variables, such as deviations from normal blood pressure or heart rate ranges, indicating increased or decreased risks for issues related to maternal health.

2.4.1 Risk Level

Risk Level in this dataset is categorized as “high risk”, “mid risk” and “low risk”.

Risk Level	Age	Systolic Blood Pressure	Diastolic Blood Pressure	Blood Sugar	Body Temperature	Heart Rate
high risk	36.216912	124.194853	85.073529	12.122610	98.899265	76.742647
low risk	26.869458	105.866995	72.534483	7.220271	98.368966	72.770936
mid risk	28.363095	113.154762	74.232143	7.795744	98.833333	74.175595

Figure 2.4.1.1: Mean value for each independent variable

From the figure above, in terms of age, those classified as "low risk" are often around 26.87 years old, while those at "high risk" are typically around 36.22 years old. For those who are deemed to be "mid risk," the mean age is roughly 28.36 years.

Now for the systolic blood pressure reading, "high risk" people usually have an average of about 124.19 mmHg, whereas "low risk" people often have an average of about 105.87 mmHg. The average systolic blood pressure for those in the "mid risk" group is approximately 113.15 mmHg.

With regard to diastolic blood pressure, people who are classified as "high risk" typically have a measurement of approximately 85.07 mmHg, whilst those who are classified as "low risk" typically have an average of roughly 72.53 mmHg. In people who are considered to be "mid risk," the typical diastolic blood pressure is approximately 74.23 mmHg.

Blood sugar levels are included in these values as well. Those classified as "low risk" often have an average blood sugar level of approximately 7.22 mmol/L, while those classified as "high risk" typically have an average of 12.12 mmol/L. In individuals classified as "mid risk," the mean blood sugar level is approximately 7.80 mmol/L.

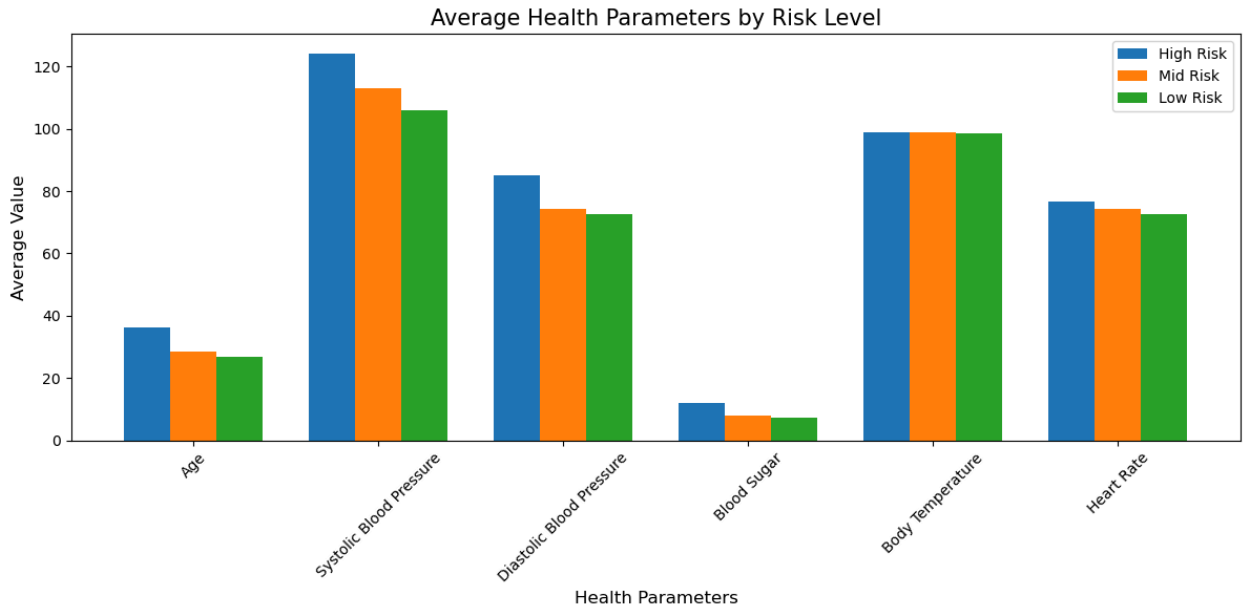


Figure 2.4.1.2: Average health parameters by risk level

When it comes to average age, high-risk people usually have older average ages than low- and mid-risk people. This finding implies that age may be a major factor in determining the degree of risk connected to specific medical disorders. On the other hand, of the three risk categories, low-risk individuals typically have the lowest average age, suggesting that younger people are less likely to belong to the high- or mid-risk groups. Using the mean (average) to display the health parameter data by risk level is an effective way to summarize and compare the typical or central values across the different risk groups. The mean captures the central tendency of each parameter within each risk group, providing a representative summary statistic. This makes it easier to discern potential associations or patterns, like whether higher risk corresponds to higher average blood pressure levels, for example. Using means helps distill the data into clear, interpretable values for contrasting the different risk populations. Overall, the mean effectively summarizes and highlights the typical parameter values tied to each risk level.

Regarding systolic blood pressure, those who are at high risk have the highest average readings, which are then accompanied by those who are at mid- and low-risk, respectively. This pattern suggests a possible link between elevated risk levels and greater systolic blood pressure. On the

other hand, the lowest average systolic blood pressure is found among low-risk individuals, indicating healthier blood pressure values in this population.

The average diastolic blood pressure is often highest in high-risk individuals, followed by mid-risk and low-risk individuals. Once more, low-risk people exhibit the lowest average diastolic blood pressure, which suggests that their blood pressure readings are generally better. The significance of blood pressure monitoring in determining health risk is highlighted by these findings.

A maternal at high risk usually has the highest average blood sugar readings, followed by someone at mid-risk and someone at low risk. There may be a connection between greater blood sugar levels and increased risk because elevated blood sugar levels are frequently linked to a higher risk of health issues. On the other hand, low-risk people have the lowest average blood sugar levels, which may indicate that their blood glucose levels are better controlled.

The average readings for body temperature do not significantly differ across the three risk categories. The results are still tightly clustered together, suggesting that body temperature may not be a defining characteristic when assessing risk in this dataset. This implies that variables other than body temperature might be more important for determining maternal health risks.

Individuals who are high-risk have the tendency to have the highest average heart rate measurements, followed by people who are mid-risk and low-risk. An elevated heart rate may indicate a number of medical concerns, such as cardiovascular problems, indicating a possible association between elevated heart rates and an increased risk. On the other hand, compared to those classified as high or mid-risk, low-risk individuals have the lowest average heart rates, which is indicative of healthier cardiovascular function. This emphasizes how crucial it is to keep an eye on heart rate as a possible predictor of general health and risk assessment.

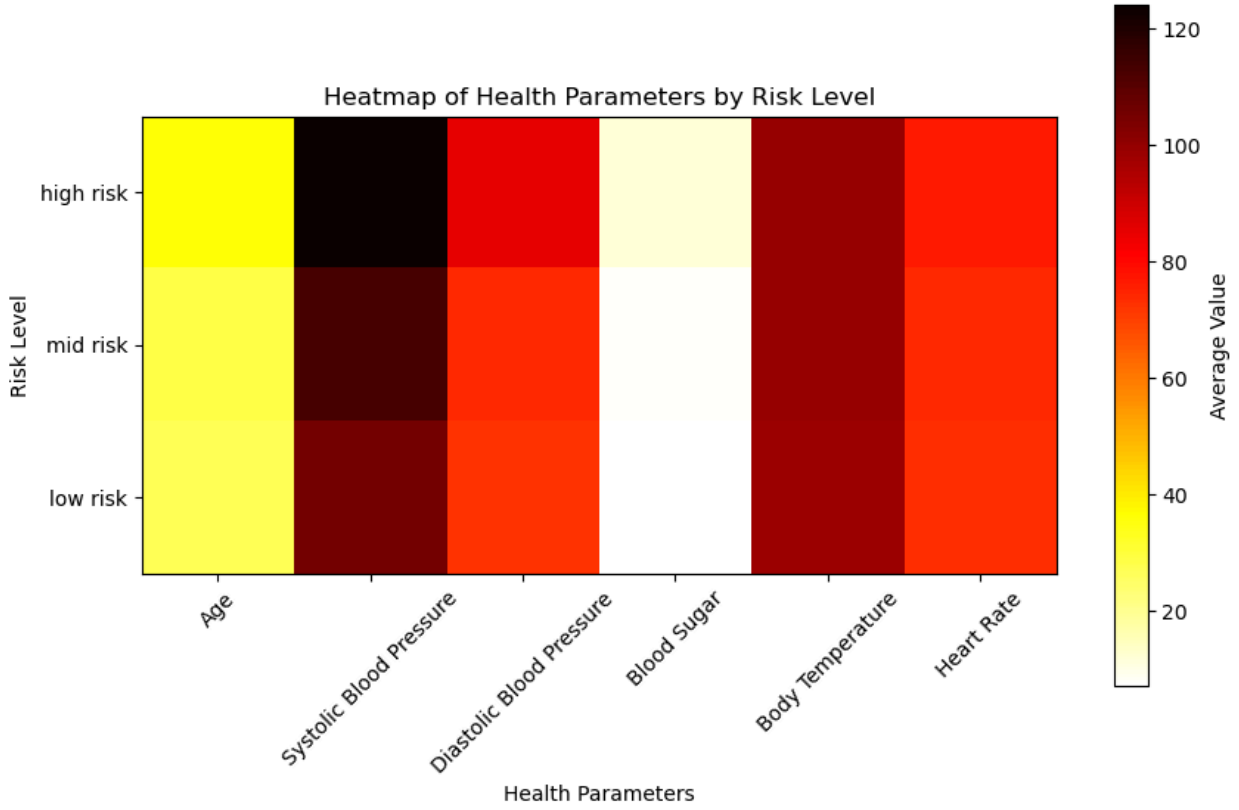


Figure 2.4.1.3: Heatmap of health parameters by risk level

Analyzing the data at various risk levels reveals unique trends with respect to different health metrics. When it comes to age, those who are categorized as "high risk" usually have darker shades on the heatmap, which means that their average age is higher than that of other risk categories. On the other hand, people classified as "low risk" have a lighter shading, indicating a lower average age than those classified as "high risk" but a greater average age than those classified as "mid risk." "Mid-risk" people are those who lie halfway between "high risk" and "low risk," as indicated by the intermediate shade that represents their average age.

The systolic blood pressure has a similar pattern. Individuals classified as "high risk" display a darker shading, signifying a greater average value in contrast to other risk categories. On the other hand, "low risk" people have a lighter shade, indicating a lower average value than "high risk" people but a higher average value than "mid risk" people. Once more, "mid risk" people are in the middle, shown by a shade that denotes an intermediate average value.

The pattern is consistent for other indicators such as blood sugar levels and diastolic blood pressure. For "high risk" individuals, darker hues denote higher average values; for "low risk" individuals, lighter tones denote lower average values. People who are considered to be "mid risk" tend to fall somewhere in the middle of these two extremes; the shades correspond to

intermediate average values. However, with only minor variations in average values, metrics like body temperature and heart rate show less fluctuation among risk levels. However, the overall pattern shown across the dataset is maintained, with lighter hues indicating lower average values for "low risk" individuals and darker shades corresponding to greater average values for "high risk" individuals.

Some interesting findings are shown when body temperature and heart rate are examined at various risk levels. In contrast to variables like age and blood pressure, body temperature remains relatively constant throughout risk categories. Although there are some minor variations in the mean temperature, the general trend is still the same. On the heatmap, darker hues correspond to "high risk" individuals' average values, while lighter shades correspond to "low risk" individuals' average values. However, the differences are not as strong as they are for other factors, suggesting that body temperature may not be a significant predictor of risk in this sample.

Heart rate similarly shows a little variance trend across risk levels. Although variations in mean heart rate are perceptible, they are quite mild. Similar to how body temperature is represented, "high risk" persons tend to have deeper shades, while "low risk" ones tend to have lighter shades. This consistency highlights the idea that heart rate might not be the only element in determining danger level, but it also supports the larger trend shown throughout the dataset. Rather, it implies that other variables, such blood pressure and blood sugar levels, can be more important in determining total health risk.

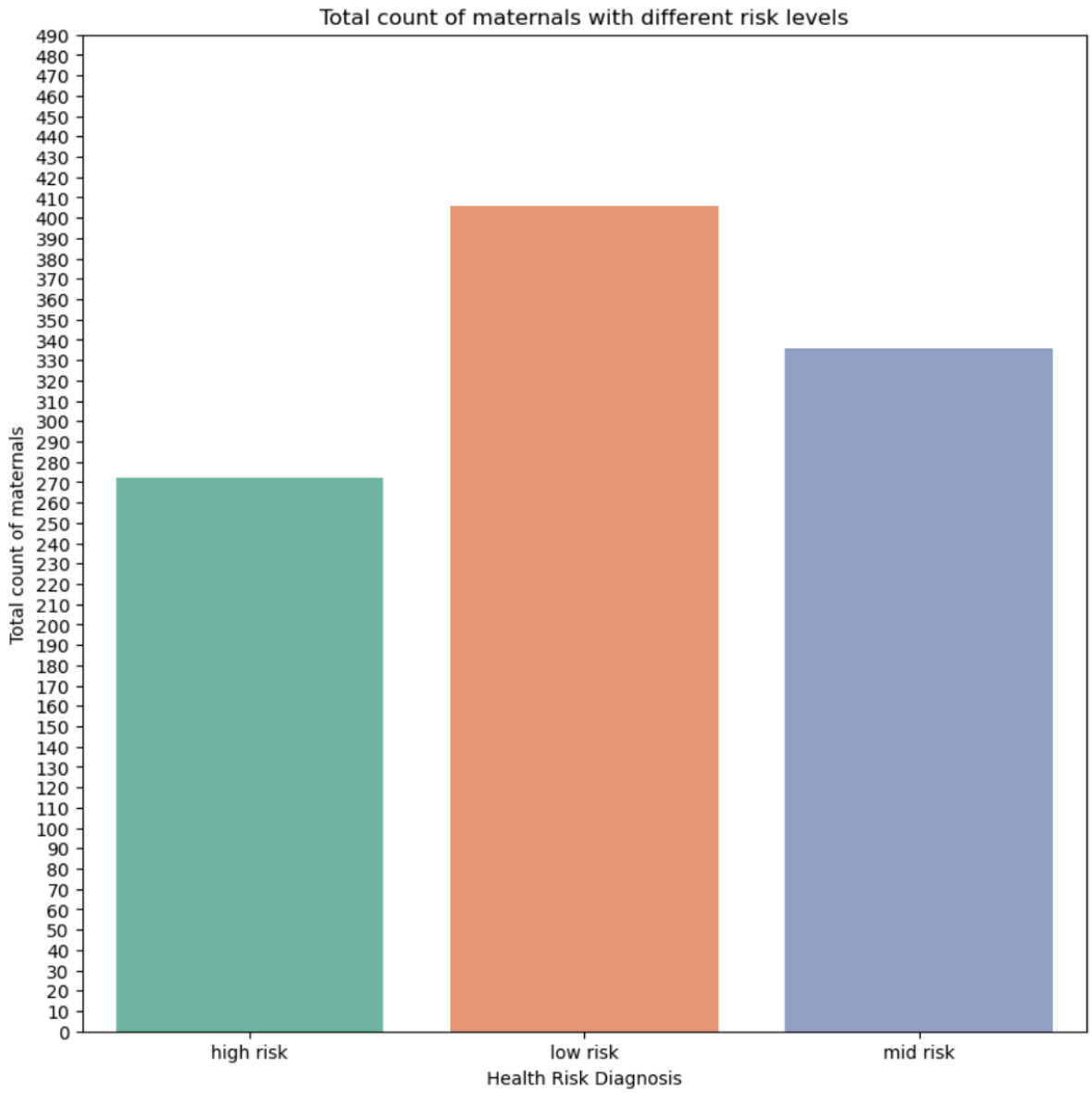


Figure 2.4.1.4: Total count of maternal with different risk levels

The distribution of health risk levels among the 1014 participants is shown in the above figure, with 336 persons classed as mid-risk, 272 as high-risk, and the remaining 406 as low-risk. This distribution translates to approximately 40.04% with low health risk, 33.14% with mid-health risk, and 26.82% with high health risk. Several important elements need to be taken into account in order to comprehend the components that contribute to this distribution.

First of all, the environment has a significant influence on how people's health turns out. Along with solid fuel indoor cooking methods, additional factors such as air pollution from vehicles and industrial operations may have contributed to respiratory illnesses and other health problems. For example, compared to rural locations with cleaner air, urban areas with high pollution levels may

provide greater health hazards. Similar to this, communities that are close to industrial areas or contaminated waterways may be at higher risk for environmental contamination-related illnesses.

Second, having access to a healthy diet is essential to preserving general health. Health risk levels could be impacted by disparities in availability to nutrient-dense food depending on variables like income, region, and agriculture techniques. Chronic diseases and malnutrition are just two of the health issues that can be attributed to food insecurity and certain dietary practices, such as eating a lot of processed foods or not enough of certain nutrients. The use of pesticides and contaminated food crops are two examples of agricultural practices that influence food safety and the ensuing health hazards.

Thirdly, people's awareness of health risks is greatly influenced by their access to education and health literacy. Variations in health risk levels among various demographic groups may be caused by differences in health literacy and education levels. Better health outcomes are frequently linked to higher education levels because educated people are more likely to have access to information on illness prevention and healthy lifestyle choices. In the same way, health literacy empowers people to successfully comprehend and apply health information, enabling them to make decisions about their health and seek the necessary medical attention.

2.5 Data Selection

Choosing items (rows): A few outliers in our dataset may impact on the accuracy of our prediction. While doing modeling, we will exclude those outliers from our dataset to obtain a more exact and accurate forecast. Additionally, we will look for any missing data and take appropriate action to either delete or replace it with mean values. Kindly reference 2.6 Data Cleaning for further details and clarification.

Choosing attributes (columns): Out of the 7 features (columns) in our data collection, the last feature, "Risk Level," is chosen as the result. In order to forecast the result, the remaining 6 characteristics in our dataset have been chosen as the independent variable. There are no qualitative data in our dataset; all of the category and numerical variables were discovered during the data preparation stage. Since all of the data in the dataset has already been pre-encoded, label encoding is not required.

2.5.1 Handling Missing Values

For predictive modeling purposes, our datasets cannot include any missing values. This is due to the possibility that predictive modeling may affect accuracy and precision, which would lower the result's dependability and credibility. As a result, we will start by looking for any missing values in our dataset.

Missing Value in Percentage

```
# how many total missing values
total_cell = np.product(df.shape)
total_missing = df.isnull().sum().sum()

# percent of data that is missing
percent_missing = (total_missing/total_cell) * 100

print("Percentage of missing value: {:.2f}%".format(percent_missing))
```

```
Percentage of missing value: 0.00%
```

```
# Print the info in the DataFrame
df.info()
df.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1014 entries, 0 to 1013
Data columns (total 7 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1014 non-null   int64
 1   Systolic Blood Pressure   1014 non-null   int64
 2   Diastolic Blood Pressure  1014 non-null   int64
 3   Blood Sugar               1014 non-null   float64
 4   Body Temperature          1014 non-null   float64
 5   Heart Rate                1014 non-null   int64
 6   Risk Level                1014 non-null   object
dtypes: float64(2), int64(4), object(1)
memory usage: 55.6+ KB

(1014, 7)
```

Figure 2.5.1.1: Missing value identification

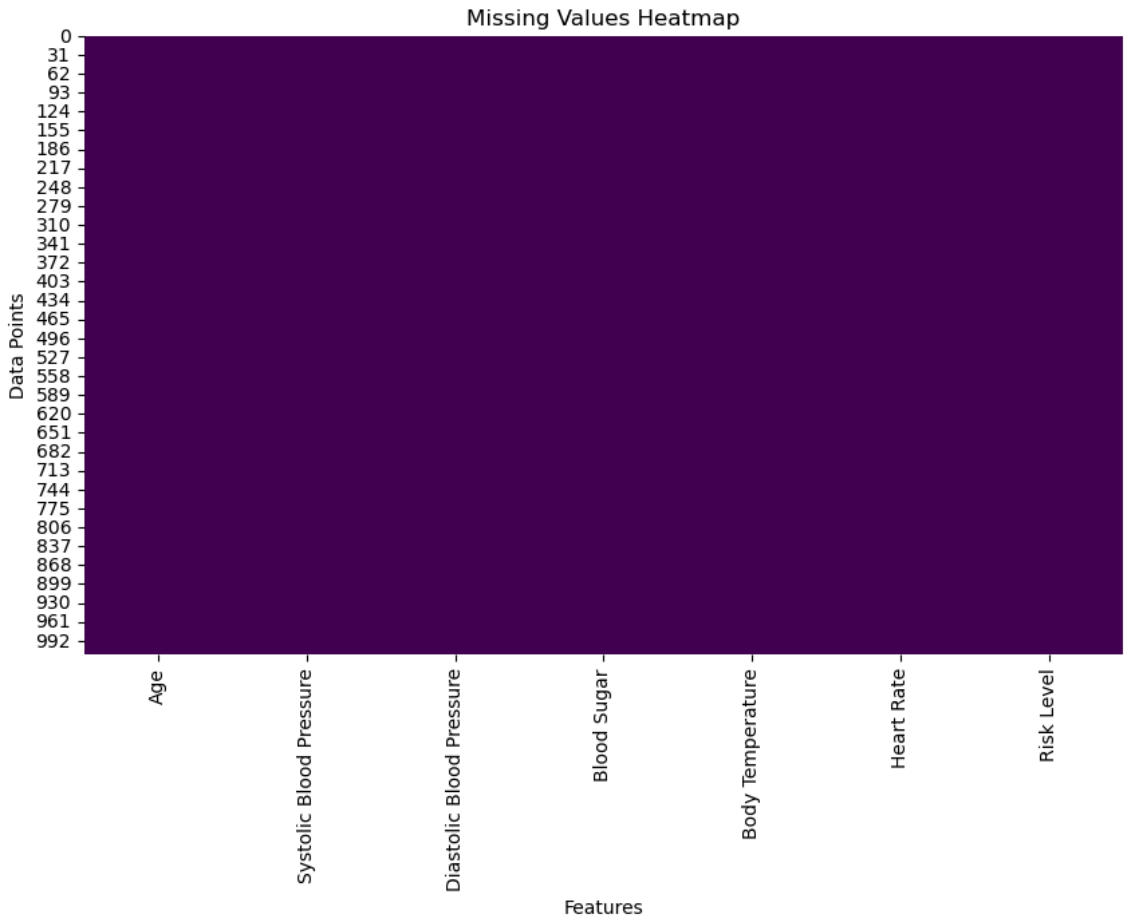


Figure 2.5.1.2: Heatmap for missing values

From Figure 2.5.1.1, the number of rows multiplied by the number of columns yields the total number of cells in the DataFrame (total_cell). By adding up the counts of missing values for each column using the isnull() function and then totaling these counts for all columns, it determines the total number of missing values (total_missing). The overall number of missing values is divided by the entire number of cells, and then multiplied by 100 to determine the percentage of missing values (percent_missing), it prints the percentage of missing values with a precision of two decimal places. Below the function to find the percentage of missing values, the info() method is used to print the DataFrame's information, which includes a summary of the DataFrame's data types for each column and the count of non-null values.

From Figure 2.5.1.2, if there is only a single color that covers the whole heatmap, it indicates that there are no missing values in any of the DataFrame's cells. This is due to high-quality data collection methods or data cleaning processes. This is due to the data collection process being carefully designed and executed to ensure complete and accurate data capture. Proper data collection techniques, such as thorough well-defined data entry protocols, and quality control measures, may have minimized the occurrence of missing values.

2.5.2 Outlier Detection

Outliers can be identified visually as extremely tall bars in the histogram or as data points that are far from the bulk of the data; this figure below builds an individual histogram plot for each numerical parameter in the dataset, allowing to examine their distributions.

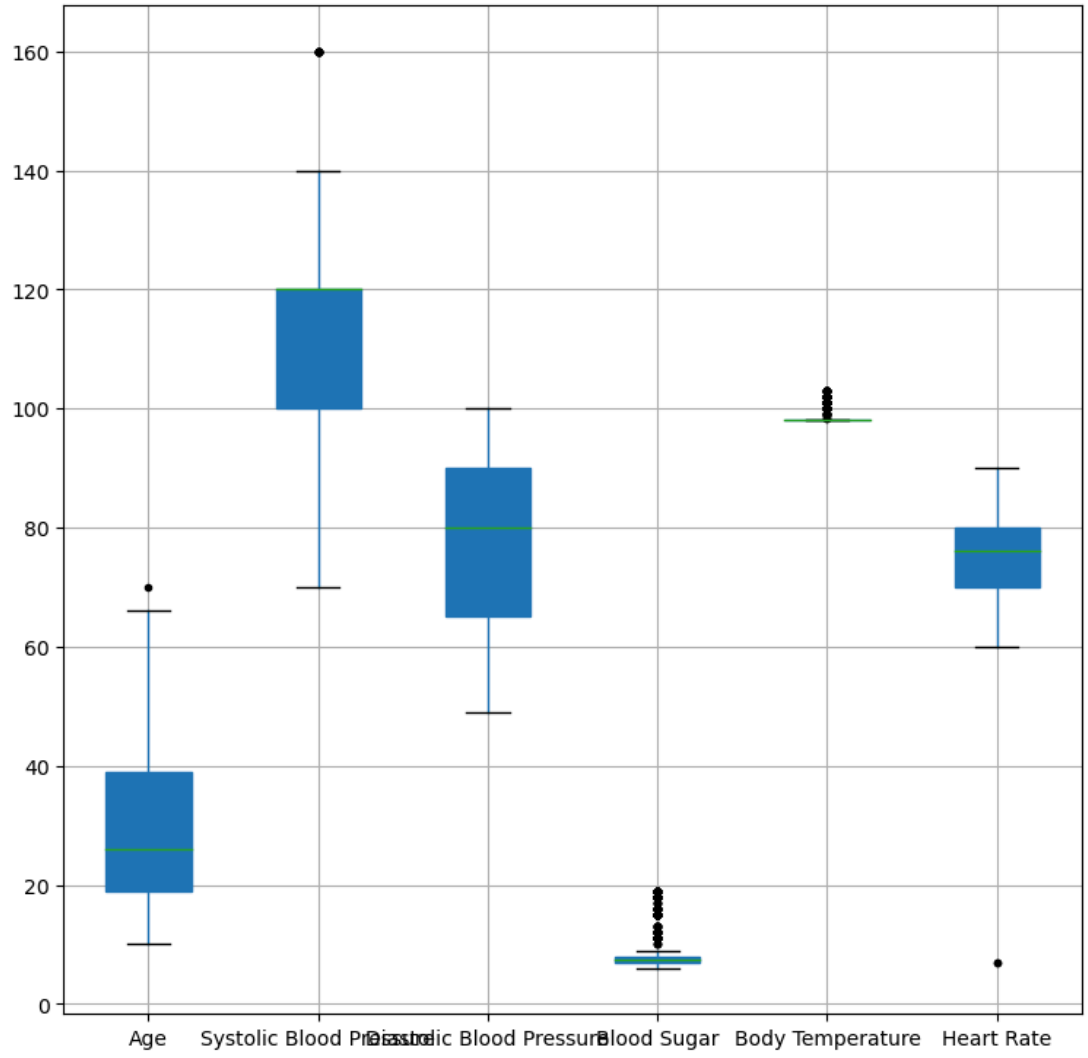


Figure 2.5.2.1: Boxplot for outlier detection

Figure above shows a boxplot of all the dataset attributes except for “Risk Level”, explanation will be given whether the outliers should be removed or not is explained below.

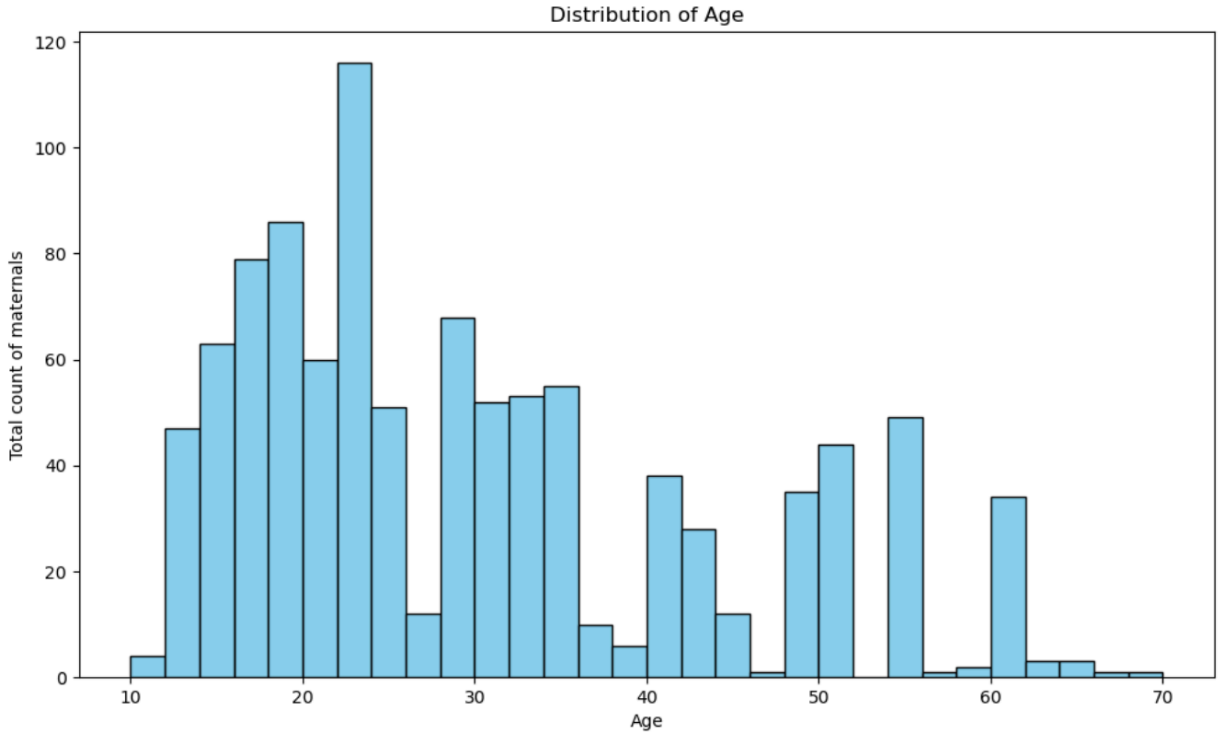


Figure 2.5.2.2: Distribution of age

From the histogram above, data ranges for Age are considered normal since individuals at age of 10 can get pregnant and age of 70 also can get pregnant.

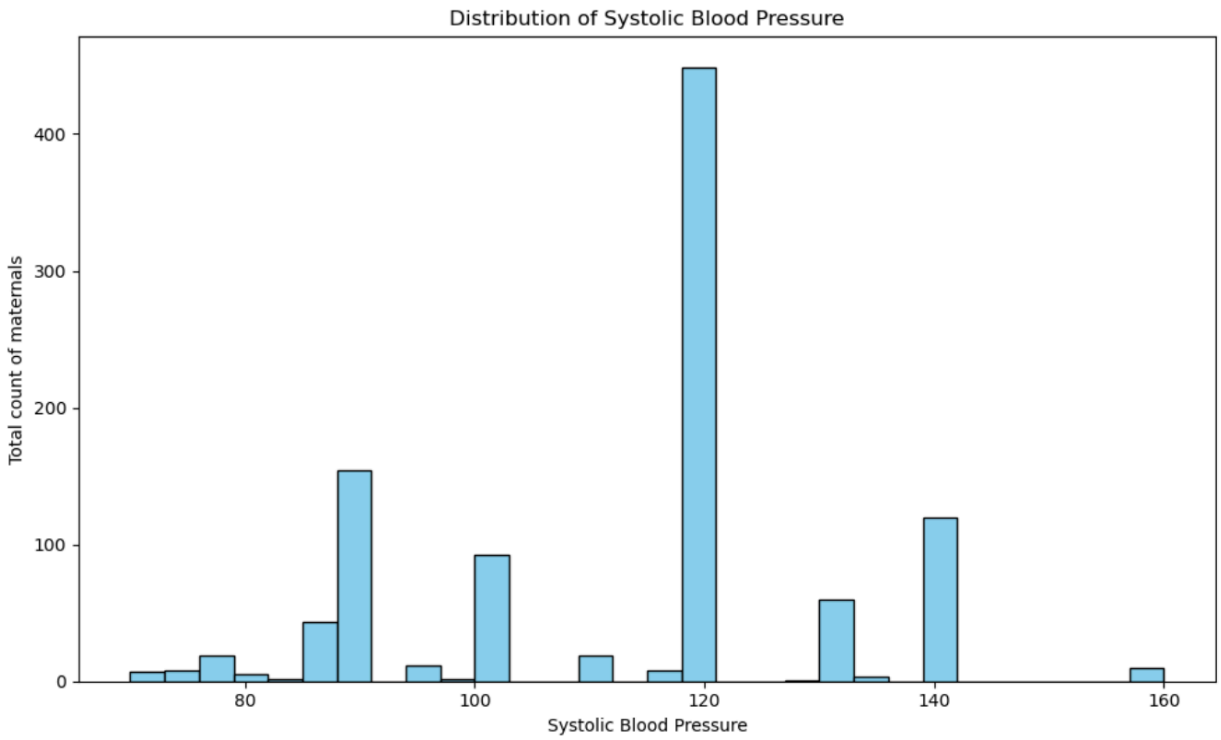


Figure 2.5.2.3: Distribution of systolic blood pressure

The smallest value of Systolic Blood Pressure is 70 mmHg which is considered low and might indicate hypotension, but it is still in a reasonable range. The highest Systolic Blood Pressure is 160mmHg which is considered high and falls within the range of hypertension (high blood pressure), but it is also in a reasonable range.

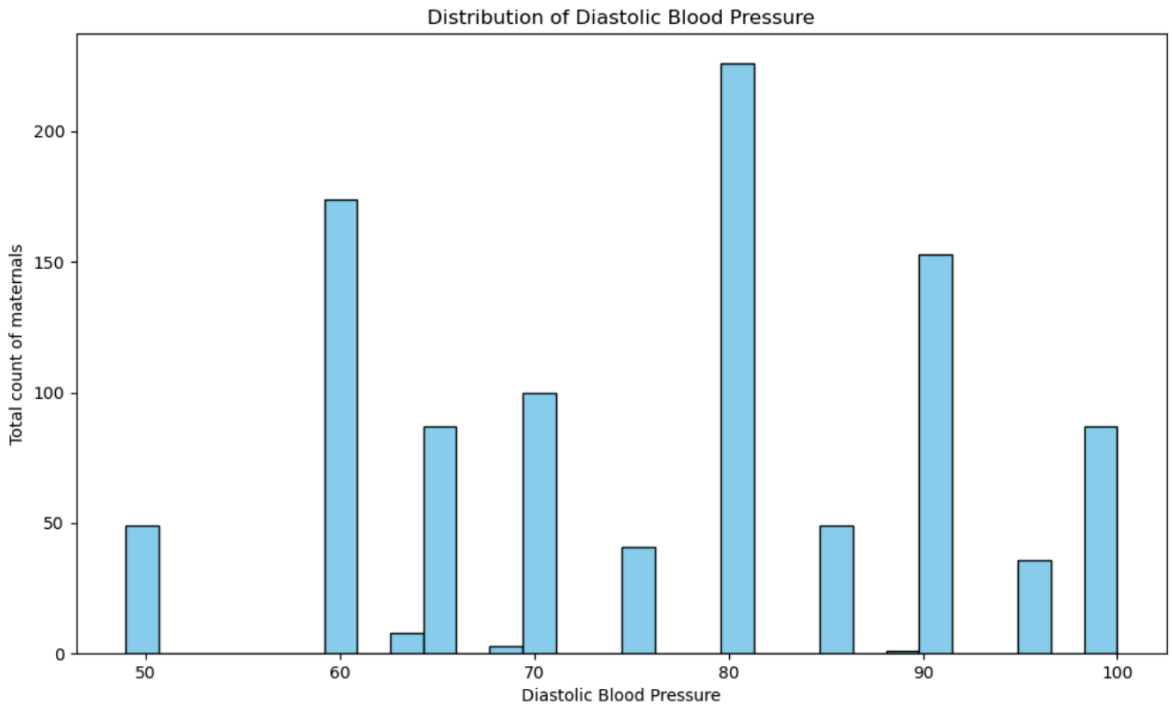


Figure 2.5.2.4: Distribution of diastolic blood pressure

The histogram indicates that the lowest diastolic blood pressure is 49mmHg which is considered low and might indicate hypotension, but it is still in a reasonable range. On the other hand, the maximum value is 100mmHg which is considered high and falls within the range of hypertension (high blood pressure), but it is also in a reasonable range.

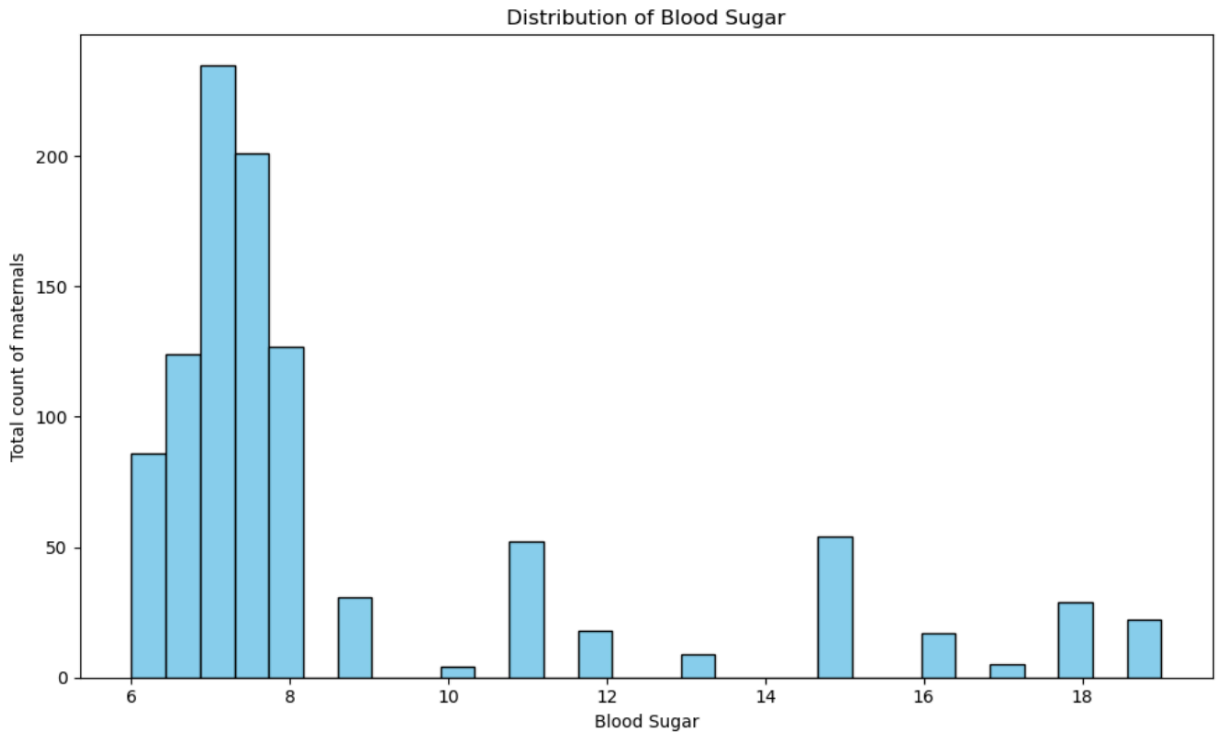


Figure 2.5.2.5: Distribution of blood sugar

The lowest value of blood sugar which is 6.0mmol/L (millimoles per liter) is within the normal range for fasting blood glucose levels. While blood sugar level of the highest value is 19.0 mmol/L is considered significantly elevated and is indicative of hyperglycemia, which means high blood sugar levels.

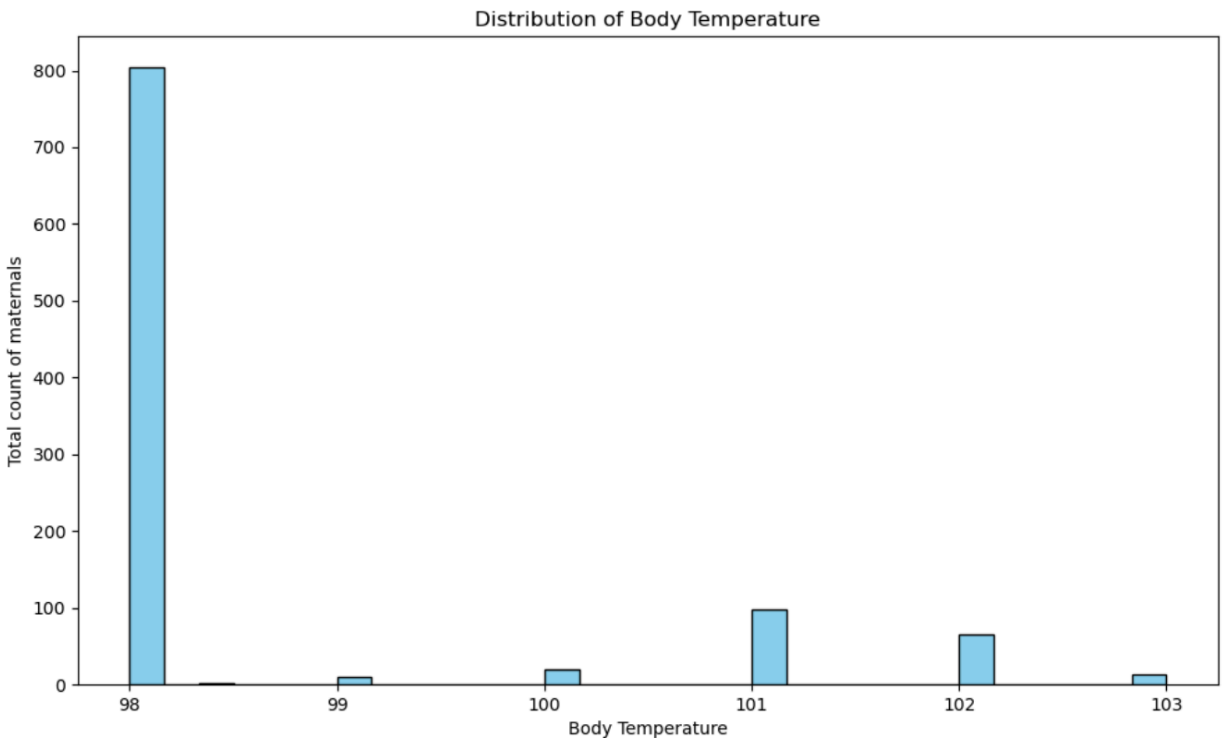


Figure 2.5.2.6: Distribution of body temperature

The highest body temperature (103°F) is considered high and indicative of a fever. A high body temperature of 103°F usually indicates an increased immunological response to an infection or disease. The lowest body temperature of 98°F has a total of 804 individuals in this body temperature range. Well, it is quite normal for body temperature at this range.

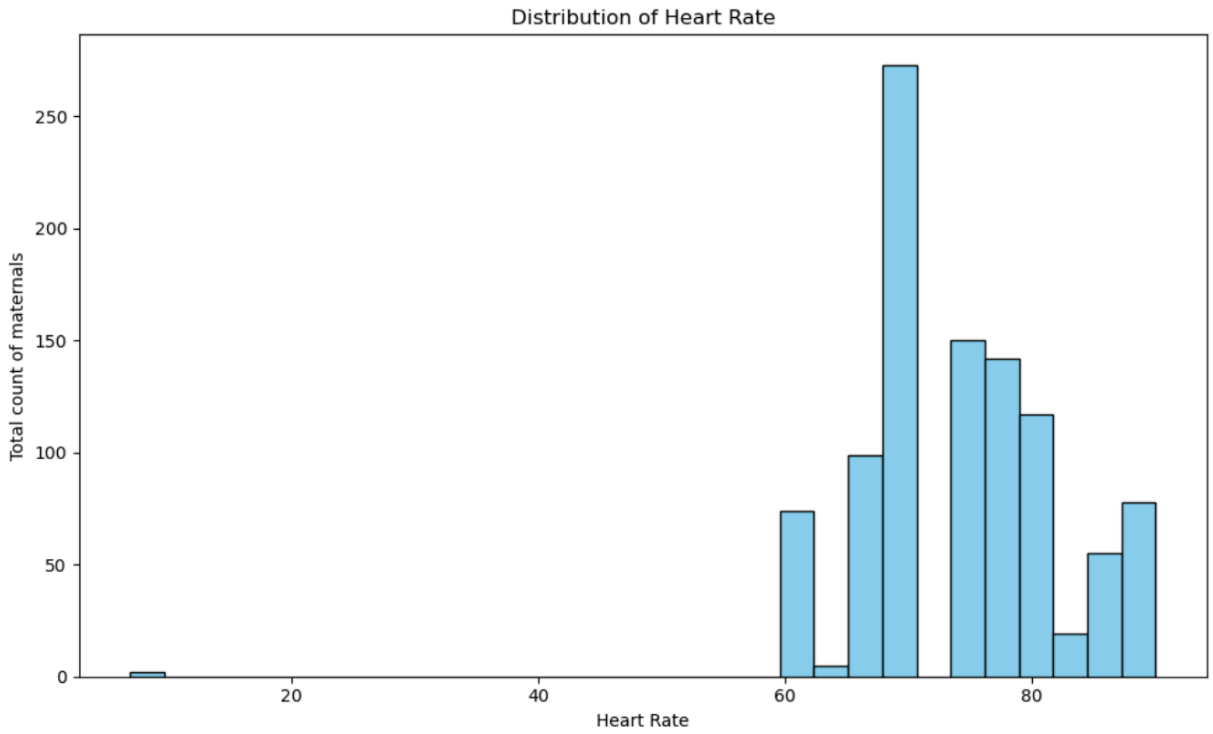


Figure 2.5.2.7: Distribution of heart rate

It is very obvious that one of the data points which is having a heart rate of 7 bpm is far away from the bulk of the data. In this case, 7 bpm of heart rate is considered as an outlier because this reading is extremely low and is not considered normal for mothers. While 90 bpm is normal range and considered normal for mothers. Thus, outliers of 7 bpm which contains two low risk individuals will be removed.

3.0 Data Preparation

Data preparation, which includes the steps of obtaining, cleaning, and structuring raw data to make it ready for analysis, is an essential part of the data analysis process. In order to make raw data from many sources easily examined and understood, it must first be transformed into a standardized, well-organized format. This step is critical for ensuring data accuracy, dependability, and comprehensiveness since it establishes the groundwork for significant insights and well-informed decision-making.

The raw information that is used for data preparation might come from a variety of sources, such as text files, spreadsheets, databases, and web resources. Both organized and unstructured data, including written documents, photos, multimedia content, and numerical values, may be included. The main goal of data preparation is to provide a clear, consistent dataset devoid of mistakes, inconsistencies, and redundancies, regardless of the source or format of the data.

Thoroughly cleaning and transforming the raw data during data preparation helps reduce the chances of errors and inaccuracies spreading into later analyses. This includes doing things like eliminating redundant entries, dealing with missing information, standardizing data formats, and fixing discrepancies between databases. Furthermore, data preparation improves processed data accessibility by optimizing its structure and arrangement, which facilitates analysts' and stakeholders' ability to explore and use the data efficiently.

3.1 Data Cleaning

3.1.1 Remove Outlier

Since we decided to remove outliers which are maternal with a heart rate of 7 bpm which is considered abnormal in real life situations. Below Figure 3.1.1.1 is a histogram of Heart Rate after removing the unusual data.

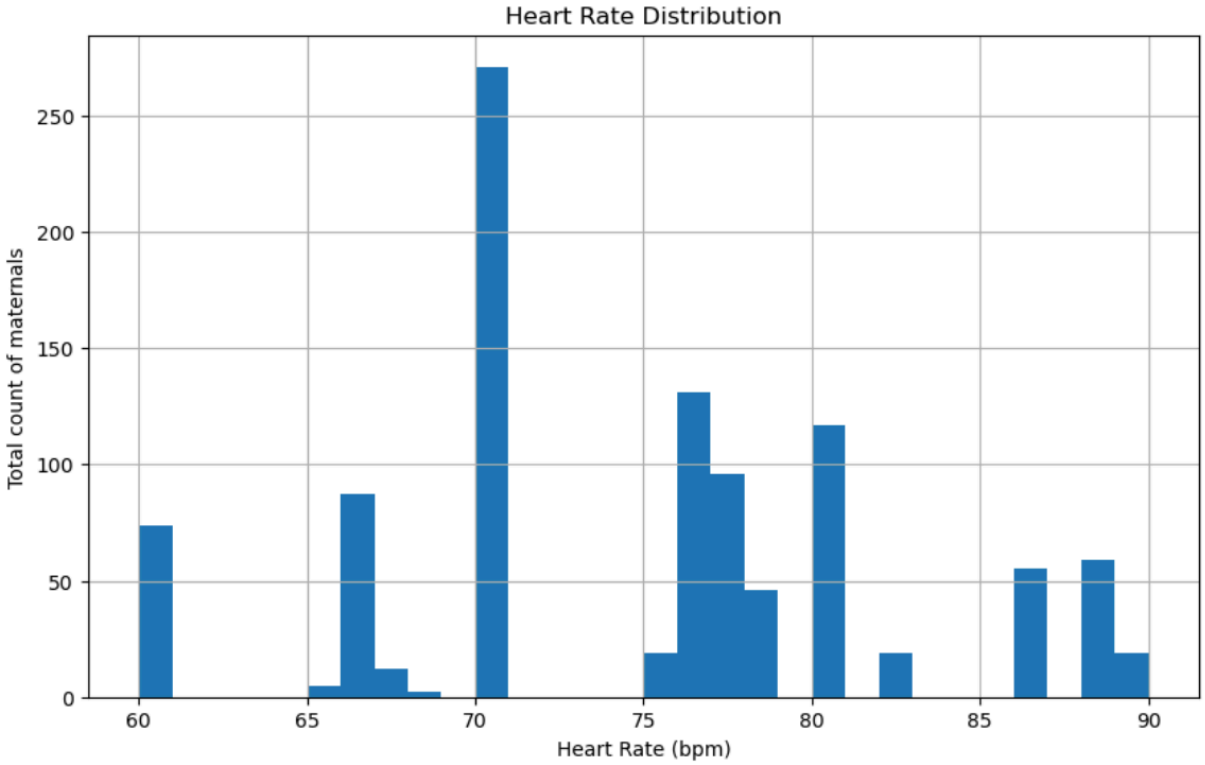


Figure 3.1.1.1: Histogram of Heart Rate after removing the unusual data

	Age	Systolic Blood Pressure	Diastolic Blood Pressure	Blood Sugar	Body Temperature	Heart Rate	Risk Level
0	25	130	80	15.0	98.0	86	high risk
1	35	140	90	13.0	98.0	70	high risk
2	29	90	70	8.0	100.0	80	high risk
3	30	140	85	7.0	98.0	70	high risk
4	35	120	60	6.1	98.0	76	low risk
...	...	...	...	...	...	...	...
1009	22	120	60	15.0	98.0	80	high risk
1010	55	120	90	18.0	98.0	60	high risk
1011	35	85	60	19.0	98.0	86	high risk
1012	43	120	90	18.0	98.0	70	high risk
1013	32	120	65	6.0	101.0	76	mid risk

1012 rows × 7 columns

Figure 3.1.1.2: Total count of rows after remove 2 records

As from Figure 3.1.1.2 above, there are a total of 1012 rows in the dataset in which 2 records of heart rate (7 bpm) are successfully removed from the dataset. Thus, this dataset is free from outliers and ready to perform next operations which is data transformation.

3.2 Data Transformation

3.2.1 Categorical Variable Encoding

In this stage, there are two approaches to encode the column "Risk Level" numerically to prepare it for modeling. One of the methods is Label Encoding. In a categorical variable, label encoding gives each category a distinct integer. When there is a distinct hierarchy or order among the categories, it is appropriate for ordinal categorical variables. By using label encoding, each category is given a numerical label, which starts at 0. The second method is One-Hot Encoding. Categorical variables are transformed into a binary matrix representation using one-hot encoding, where each category becomes a binary column. If a data point belongs to a category, the corresponding column is marked with a 1, while all other columns are marked with 0s. When there is inherent order among the categories in ordinal categorical variables, label encoding is appropriate.

For this assignment, the "Risk Level" column in the dataset will be processed via label encoding for a number of important reasons. First, label encoding assigns a unique integer to each category, thereby reducing dimensionality to a single column. Conversely, one-hot encoding creates a binary column for each category in the categorical variable, which can lead to a

significant increase in dataset dimensionality, particularly when dealing with variables featuring numerous unique categories. This dimensionality reduction of label encoding can simplify the dataset structure and computation, making it more manageable, especially in scenarios with limited computational resources or when handling large datasets.

Moreover, label encoding preserves any inherent ordinal relationships between categories. For instance, in the case of "low risk," "mid risk," and "high risk," label encoding captures the ordinal nature of these categories by assigning consecutive integers, reflecting their hierarchical order. This preservation of information is crucial in scenarios where the ordinality of categories holds significance for the analysis or model interpretation. However, it's essential to note that one-hot encoding treats each category as independent, thus ignoring any ordinal relationships, which might be disadvantageous in situations where such relationships are meaningful.

Furthermore, label encoding offers simplicity and improved model interpretability compared to one-hot encoding. The process of assigning integers to categories based on their order or frequency is straightforward and intuitive, making it easier to implement and understand. Additionally, for algorithms that naturally handle ordinal data or understand the numeric representation of categories, such as decision trees or linear regression, label encoding can enhance model interpretability. Overall, while both encoding techniques have their merits, label encoding's dimensionality reduction, preservation of ordinal information, simplicity, and improved interpretability make it a preferred choice in certain machine learning contexts, especially when dealing with ordinal categorical variables like risk levels.

```
Categorical value for Risk Level: [High Risk, Low Risk, Mid Risk]
Encoded value for Risk Level    : [0 1 2]
```

Figure 3.2.1.1: Encoded categorical variable

There are 3 types of risk, which is “High Risk”, “Low Risk” and “Mid Risk”. We encoded the value for each risk level into 0 representing “High Risk”, 1 represents “Low Risk” and 2 represents “Mid Risk”. Putting non-numeric data into a numerical format that machine learning algorithms and statistical models can use is the goal of encoding categorical variables.

```
      Age  Systolic Blood Pressure  Diastolic Blood Pressure  Blood Sugar  \
0      25                      130                        80         15.0
1      35                      140                        90         13.0
2      29                       90                        70          8.0
3      30                      140                        85          7.0
4      35                      120                        60          6.1
...   ...                      ...                       ...          ...
1009   22                      120                        60         15.0
1010   55                      120                        90         18.0
1011   35                       85                        60         19.0
1012   43                      120                        90         18.0
1013   32                      120                        65          6.0

      Body Temperature  Heart Rate Risk Level  Risk_Level_encoded
0                 98.0          86  high risk                   0
1                 98.0          70  high risk                   0
2                100.0          80  high risk                   0
3                 98.0          70  high risk                   0
4                 98.0          76   low risk                   1
...                ...         ...        ...                 ...
1009              98.0          80  high risk                   0
1010              98.0          60  high risk                   0
1011              98.0          86  high risk                   0
1012              98.0          70  high risk                   0
1013             101.0          76   mid risk                   2

[1012 rows x 8 columns]
```

Figure 3.2.1.2: Combine encoded column to original dataset

3.2.2 Feature Scaling

For numerical features in machine learning tasks, feature scaling is a crucial preprocessing step that is recommended. The first benefit of scaling features is enhanced algorithm performance. Input feature scales are necessary for many machine learning methods, including neural networks, logistic regression, and linear regression. Diverse scales among features can lead algorithms to perform poorly or prioritize certain features based solely on their scale, making feature scaling crucial to ensure fair treatment of all features and enhance model performance.

Moreover, techniques using gradient-based optimization, such as gradient descent, can converge more quickly when features are scaled accordingly. Scaling allows for faster convergence by putting features in a similar numeric range, which facilitates more efficient gradient updates. Additionally, feature scaling improves numerical stability, especially for algorithms that depend on calculating distances between data points, such as support vector machines (SVMs) and

k-nearest neighbors (KNN). Uneven feature scales can dominate these calculations, potentially compromising algorithm performance. Common scaling techniques include normalization (Min-Max scaling) and standardization (Z-score normalization), each offering distinct advantages.

Regardless of their original scales, standardization promotes consistency across features by transforming them to have a mean of 0 and a standard deviation of 1. The formula is (x - mean(x)) / std(x). On the other hand, normalization ensures uniformity while maintaining the relative variances between feature values by scaling characteristics to a similar range, usually between 0 and 1. The formula is: (x - min(x)) / (max(x) - min(x)). These techniques offer efficient ways to scale features, enabling machine learning algorithms to function at their best and produce consistent outcomes.

```
Min-Max Scaled Dataset:
          Age  Systolic Blood Pressure  Diastolic Blood Pressure  \
0     0.250000                 0.666667                  0.607843
1     0.416667                 0.777778                  0.803922
2     0.316667                 0.222222                  0.411765
3     0.333333                 0.777778                  0.705882
4     0.416667                 0.555556                  0.215686
...        ...                      ...                       ...
1009  0.200000                 0.555556                  0.215686
1010  0.750000                 0.555556                  0.803922
1011  0.416667                 0.166667                  0.215686
1012  0.550000                 0.555556                  0.803922
1013  0.366667                 0.555556                  0.313725

      Blood Sugar  Body Temperature  Heart Rate
0        0.692308               0.0    0.866667
1        0.538462               0.0    0.333333
2        0.153846               0.4    0.666667
3        0.076923               0.0    0.333333
4        0.007692               0.0    0.533333
...           ...               ...         ...
1009     0.692308               0.0    0.666667
1010     0.923077               0.0    0.000000
1011     1.000000               0.0    0.866667
1012     0.923077               0.0    0.333333
1013     0.000000               0.6    0.533333

[1012 rows x 6 columns]
```

Figure 3.2.2.1: Min-Max scaling for independent variable

As from the figure above, Normalization (Min-Max scaling) is chosen rather than Standardization (Z-score Normalization). It's because outliers can have a big impact on standardization. Columns like "Blood Sugar" and "Body Temperature" have a lot of outliers, yet they can't be removed because they are meaningful. The process of standardization calculates the mean and standard deviation for the complete dataset, including the values of outliers. Significantly skewed the mean and raising the standard deviation due to outliers can compress results within the typical range of data.

On the other hand, since Min-Max scaling (also known as normalization) only rescales the features to a fixed range, usually [0, 1] or [-1, 1], it is less impacted by outliers. The dataset is less susceptible to outliers since the lowest and maximum values used for scaling are taken from the dataset's actual minimum and maximum values.

3.3 Data Combination

After performing remove outliers, categorical variable encoding and feature scaling in Data Transformation. It is needed to combine all the data to a new dataframe called df_combined. This combined dataset with all transformed features can then be used as input for training machine learning models as shown Figure 3.3.1 below.

	Age	Systolic Blood Pressure	Diastolic Blood Pressure	\
0	0.250000	0.666667	0.607843	
1	0.416667	0.777778	0.803922	
2	0.316667	0.222222	0.411765	
3	0.333333	0.777778	0.705882	
4	0.416667	0.555556	0.215686	
...	...	...	...	
1009	0.200000	0.555556	0.215686	
1010	0.750000	0.555556	0.803922	
1011	0.416667	0.166667	0.215686	
1012	0.550000	0.555556	0.803922	
1013	0.366667	0.555556	0.313725	

	Blood Sugar	Body Temperature	Heart Rate	Risk_Level_encoded
0	0.692308	0.0	0.866667	0
1	0.538462	0.0	0.333333	0
2	0.153846	0.4	0.666667	0
3	0.076923	0.0	0.333333	0
4	0.007692	0.0	0.533333	1
...	...	...	...	...
1009	0.692308	0.0	0.666667	0
1010	0.923077	0.0	0.000000	0
1011	1.000000	0.0	0.866667	0
1012	0.923077	0.0	0.333333	0
1013	0.000000	0.6	0.533333	2

[1012 rows x 7 columns]

Figure 3.3.1: Combine all transformed data to a new dataframe called “df_combined”

After combining all of the data that had been transformed, it is time to perform one of the crucial steps in the data preparation stage, which is splitting the dataset into a training and a testing set.

3.4 Splitting Dataset into Training and Testing Set

One of the key processes in the data preparation stage of building a machine learning model is dividing the data into training and testing sets. This procedure is critical to prevent overfitting, a phenomenon in which the model becomes unduly tuned to the details of the training set, making it unusable for fresh, unobserved data. Through the process of separating out a unique test set, practitioners are able to obtain a more precise comprehension of the model's capacity for generalization, which facilitates the evaluation of its real-world performance outside of the training data.

Moreover, data splitting makes it easier to adjust the model's hyperparameters, which is an essential part of optimizing the model. The user sets the hyperparameters, which determine the model's behavior and complexity, as opposed to the data directly dictating them. During the training process, the validation set facilitates the iterative adjustment of these hyperparameters, hence maximizing the model's performance and guaranteeing its flexibility to a variety of datasets and scenarios. By using this systematic approach to data division, we are able to build machine learning models that are more dependable and robust, better equipped to handle real-world problems.

```
from sklearn.model_selection import train_test_split

# Separate the features and target variable
X = df_combined.drop(['Risk_Level_encoded'], axis=1)
y = df_combined[['Risk_Level_encoded']]

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Print the shapes of the splits
print(f"Training Set: X_train={X_train.shape}, y_train={y_train.shape}")
print(f"Test Set: X_test={X_test.shape}, y_test={y_test.shape}")
```

```
Training Set: X_train=(708, 6), y_train=(708, 1)
Test Set: X_test=(304, 6), y_test=(304, 1)
```

Figure 3.4: Split dataset into training and testing set

Based on multiple testing and characteristics of our dataset, a split ratio of 70:30 is the best ratio for splitting our dataset. This split provides a balance between having enough data to train the model effectively and ensuring a robust evaluation on unseen data. It also reduced the risk of overfitting. A bigger training set reduces the chance of overfitting, which occurs when the model fails to learn the underlying patterns in the training data via memorization. As an impartial validation tool, the test set helps determine if the model has really picked up on significant patterns or has just internalized the noise present in the training set.

A split of the dataset into 80:20, where 80% of the data is used for training and 20% of the data is held out for testing is not a good choice for our dataset because the model could be able to learn more from a bigger training set (80%), which could result in improved performance. However, compared to a 70-30 split, the smaller test set (20%) might produce less accurate performance estimates, particularly if the dataset is relatively small. On the other hand, a 90-10 split provides even more data for training, which would enable the model to pick up more intricate patterns. However, the test set is much smaller, which could result in less accurate performance estimations, particularly if a thorough evaluation of the model's performance is required.
Spilt test for different scaling will be conducted at Gradient Boosting (4.3) and Random Forest (4.2).

4. Modelling

The modeling process in data science encompasses conducting exploratory data analysis to gain insights and understand the underlying patterns in the data, followed by feature selection and engineering to identify the most relevant features for modeling. Appropriate machine learning or statistical models are then selected and trained on the data, and the trained models are evaluated using various metrics to assess their performance. Hyperparameter tuning may be applied to optimize model effectiveness, and model interpretation and validation ensure alignment with domain knowledge and business requirements. Deployment into production systems, documentation, communication of the modeling process, and iterative improvement based on feedback and new data complete the modeling cycle in data science.

4.1 Decision Trees

Decision trees are a popular supervised learning algorithm used for both classification and regression tasks. They are intuitive and easy to understand, making them useful for both beginners and experts in machine learning. Decision trees can be a powerful tool in machine learning, offering a balance between interpretability and performance. However, they might not always provide the best predictive accuracy compared to more complex models like ensemble methods (e.g., Random Forests, Gradient Boosting Machines) for some datasets.

4.1.1 Decision Tree accuracy test set and training set

Code statement :

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize the decision tree classifier
model_dt = DecisionTreeClassifier(    criterion='gini',  # Try using Gini impurity as the criterion
    max_depth=14,      # Experiment with a higher max_depth
    min_samples_split=5,  # Experiment with a smaller min_samples_split
    min_samples_leaf=2,   # Experiment with a smaller min_samples_leaf
    random_state=42)

# Fit the model on the training data
model_dt.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model_dt.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred) * 100
print("Accuracy: {:.2f}%".format(accuracy))

# Print classification report
print("Classification report:")
print(classification_report(y_test, y_pred))
```

For the code above, We use Criterion='gini' to specify the impurity measure used to split the nodes. Gini impurity is one of the options along with entropy. Gini impurity measures how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the set.

max_depth=14 sets the maximum depth of the decision tree. It limits the number of nodes in the tree, which helps prevent overfitting. A higher max_depth allows the tree to capture more complex relationships in the data but increases the risk of overfitting.
min_samples_split=5 specifies the minimum number of samples required to split an internal node. It controls how finely the tree can partition the input space. A smaller value allows the tree to make more detailed splits, potentially capturing more nuances in the data but also increasing the risk of overfitting.
By setting random_state, this ensures that the results of thecode are reproducible. Runing the code with the same random state, you'll get the same results. This is crucial for debugging, testing, and sharing code with others.

Output:

```
Accuracy: 75.00%
Classification report:
              precision    recall  f1-score   support

           0       0.74      0.90      0.81        72
           1       0.80      0.74      0.77       127
           2       0.70      0.66      0.68       105

    accuracy                           0.75       304
   macro avg       0.75      0.77      0.75       304
weighted avg       0.75      0.75      0.75       304
```

As shown above, decision tree classifier achieved an accuracy of 75.00% on the test set, which means it correctly classified approximately 75.00% of the samples.The macro avg and weighted avg provide averages across all classes, considering either each class equally (macro avg) or weighting by the number of true instances in each class (weighted avg).

Code Statement:

```
# Calculate accuracy
dt_acc_test = model_dt.score(X_test, y_test) * 100
dt_acc_train = model_dt.score(X_train, y_train) * 100

# Print accuracy
print("Decision Tree accuracy for test set: {:.2f}%".format(dt_acc_test))
print("Decision Tree accuracy for training set: {:.2f}%".format(dt_acc_train))
```

Output:

```
Decision Tree accuracy for test set: 75.00%
Decision Tree accuracy for training set: 90.54%
```

The output above shows the accuracy of the decision tree model on both the test set and the training set.

4.1.2 Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. It allows visualization of the performance of an algorithm and helps in understanding how well the model is performing in terms of correctly and incorrectly classified instances.

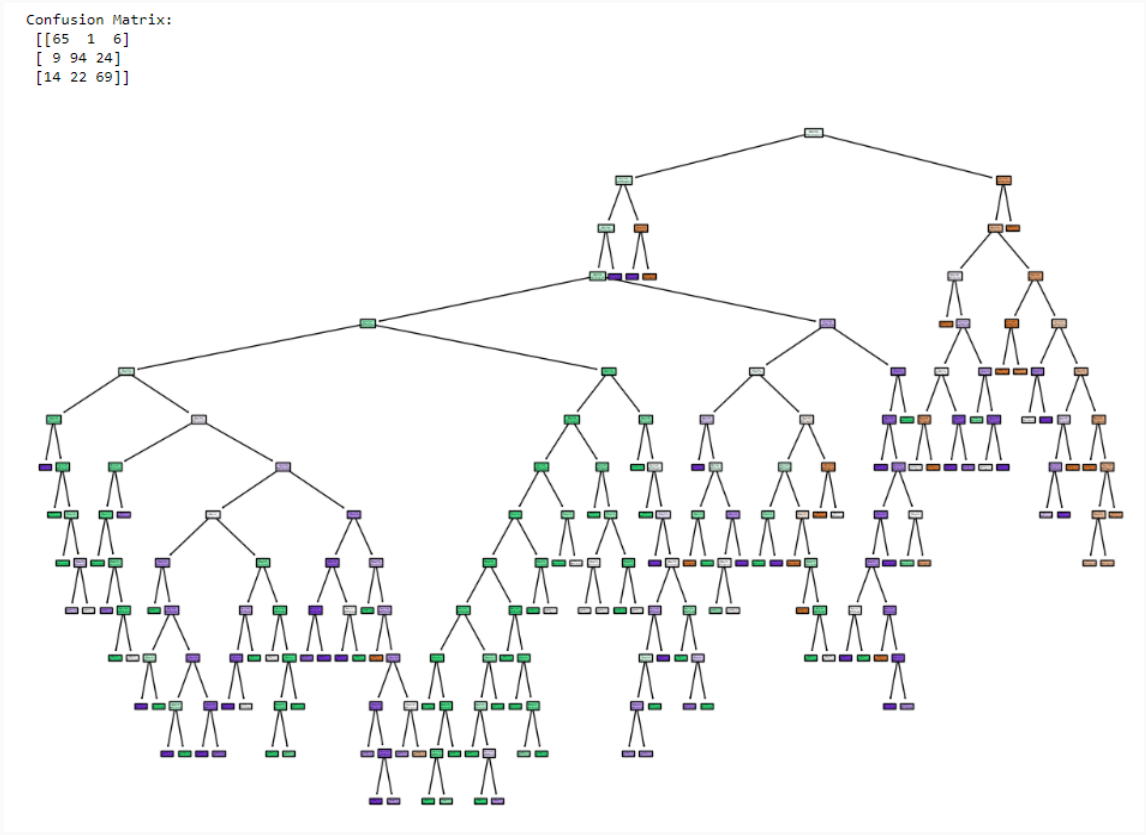


Figure 4.1.1

Figure 4,1,1 shows confusion matrix of decision tree and plot_tree function from the sklearn.tree module to visualize the decision tree model

The confusion matrix represents the classification performance of a model by showing the counts of correct and incorrect predictions for each class. In this case, for Class 0 (High Risk), the model accurately predicted 65 instances as High Risk, but misclassified 1 instance as High Risk when it was not, and missed 6 High Risk instances. For Class 1 (Low Risk), the model correctly predicted 94 instances as Low Risk, but misclassified 9 instances as Low Risk when they were not, and missed 24 Low Risk instances. For Class 2 (Mid Risk), the model correctly predicted 69 instances as Mid Risk, but misclassified 22 instances as Mid Risk when they were not, and missed 14 Mid Risk instances. The confusion matrix provides valuable insights into the model's performance and areas for improvement, aiding in refining the model to better classify instances into their respective classes.

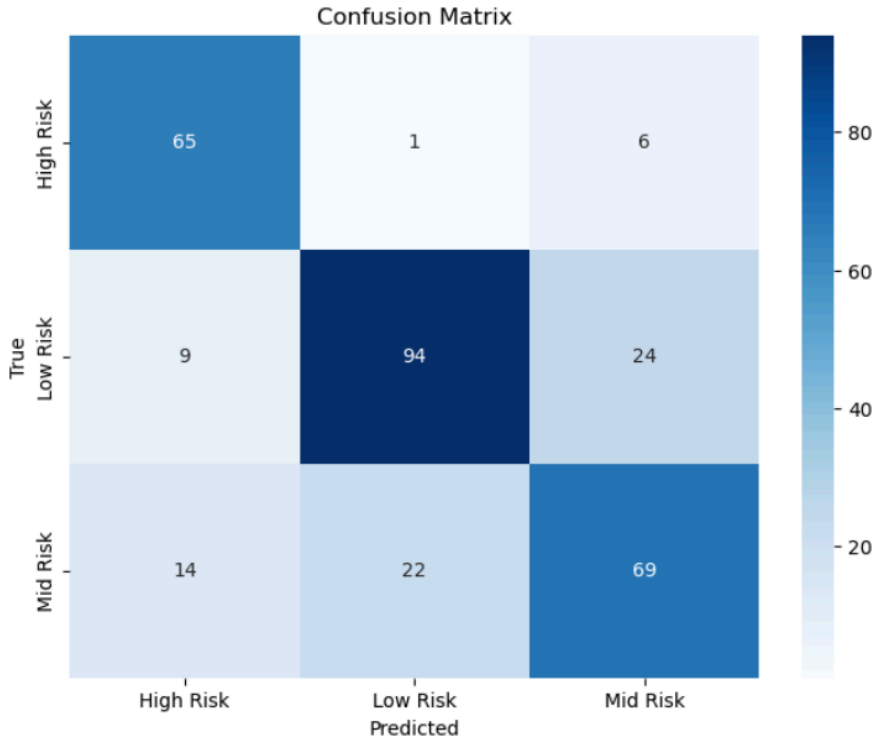


Figure 4.1.2

Visualizing confusion matrix using a heatmap function from seaborn. This visualization provides a clear representation of the confusion matrix, making it easier to interpret the performance of the classification model.

4.1.3 Decision Tree's Entropy

Entropy is a statistical concept that quantifies the uncertainty or disorder in a set of data.
In the context of decision trees, entropy is used to measure the impurity of a collection of examples. A dataset with low entropy means that it is pure, with all examples belonging to the same class. A dataset with high entropy means that it is impure, with examples belonging to multiple classes. Entropy is highest when the dataset contains an equal proportion of examples from each class, and lowest when all examples belong to the same class.

```
# Fit the model on the training data
model_dt.fit(X_train, y_train)

# Predict on the testing data
y_pred = model_dt.predict(X_test)

# Calculate the entropy for the predicted labels
unique_labels, label_counts = np.unique(y_pred, return_counts=True)
label_proportions = label_counts / len(y_pred)
entropy_value = -np.sum(label_proportions * np.log2(label_proportions))

print("Entropy for Decision Tree:", entropy_value)
```

```
Entropy for Decision Tree: 1.57500330285404
```

Figure 4.1.3

Calculating entropy for a decision tree involves assessing the impurity or disorder in a set of target labels. In the context of classification, entropy is commonly used as a measure of impurity at a node in the decision tree. The entropy value of approximately 1.5 suggests a moderate level of impurity in the predicted labels by the decision tree model. While this value is higher than 1, indicating some degree of uncertainty or variability in the classification decisions.

While high entropy in a decision tree generally indicates a higher level of impurity or uncertainty in the predicted labels, there are a few scenarios where higher entropy might be acceptable or even beneficial.

Decision trees with higher entropy values might be more robust to noisy or irrelevant features in the dataset. By considering a broader range of potential splits, the decision tree model can avoid overfitting to noisy patterns and generalize better to unseen data. Besides that, in datasets with multimodal distributions or overlapping classes, higher entropy values can reflect the diversity of data points within each class. Decision trees with higher entropy may be better equipped to handle such complex data distributions and make appropriate predictions.

4.1.4 Decision Tree Precision, Recall and F1-score

```
print("Decision Tree Precision: {:.2f}%".format(dt_precision))
print("Decision Tree Recall: {:.2f}%".format(dt_recall))
print("Decision Tree F1-score: {:.2f}%".format(dt_f1))
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.90      0.81        72
           1       0.80      0.74      0.77       127
           2       0.70      0.66      0.68       105

    accuracy                           0.75       304
   macro avg       0.75      0.77      0.75       304
weighted avg       0.75      0.75      0.75       304

Decision Tree Precision: 75.13%
Decision Tree Recall: 75.00%
Decision Tree F1-score: 74.80%
```

Figure 4.1.4

Figure 4.1.4 shows that the precision, recall, and F1-score metrics provide insights into the performance of the decision tree model. The precision of 75.13% indicates that when the model predicts a specific class, such as High Risk, it is correct approximately 75.13% of the time. The recall of 75.00% suggests that the model successfully identifies approximately 75.00% of the instances belonging to a specific class, such as High Risk. The F1-score of 74.80% is the harmonic mean of precision and recall, providing a balanced measure of the model's accuracy and completeness. Overall, these metrics indicate that the decision tree model achieves reasonably good performance in both precision and recall, with a balanced F1-score, demonstrating its effectiveness in classifying instances across different risk levels.

4.1.5 Decision Tree AUC

AUC stands for "Area Under the ROC Curve." In the context of machine learning, especially binary classification tasks, the ROC (Receiver Operating Characteristic) curve is a graphical representation of the trade-off between the true positive rate (sensitivity) and the false positive rate (1 - specificity) for various threshold values.

```
from sklearn.metrics import roc_auc_score

# Predict probabilities for the test set
y_prob_dt = dt_clf.predict_proba(X_test)

# Compute AUC score for each class
auc_scores = []
for i in range(len(dt_clf.classes_)):
    auc_score = roc_auc_score((y_test == dt_clf.classes_[i]).astype(int), y_prob_dt[:, i])
    auc_scores.append(auc_score)

# Average AUC scores
dt_auc = sum(auc_scores) / len(auc_scores)
print("Decision Tree Average AUC Score:", dt_auc)
```

```
Decision Tree Average AUC Score: 0.8881057297180973
```

Figure 4.1.5

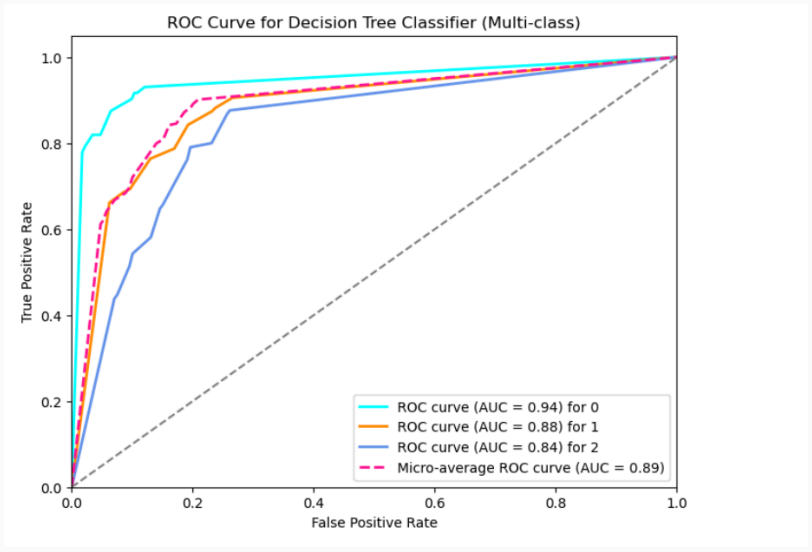


Figure 4.1.6

Figure 4.1.5 shows the code and output of AUC calculator and it is visualized using AUC ROC curve graph. The average Area Under the Curve (AUC) score of 0.8881 for the decision tree model indicates its ability to discriminate between positive and negative instances across all classes. AUC measures the performance of a binary classification model across different threshold values, and an AUC score closer to 1 suggests better discrimination ability. In this case, the decision tree model achieves a high AUC score, indicating strong performance in distinguishing between classes. This suggests that the model's predicted probabilities are well-calibrated and provide meaningful differentiation between the classes. Overall, the high AUC score underscores the effectiveness of the decision tree model in classification tasks.

4.1.6 Decision Tree Cohen's Kappa Coefficient

Cohen's kappa coefficient, often simply referred to as "Cohen's kappa," is a statistic used to measure the agreement between two raters (or two sets of classifications) when the ratings are categorical. It provides a measure of inter-rater reliability, indicating how well two raters agree beyond what would be expected by chance.
By examining the kappa matrix, which shows the agreement coefficients for each pair of classes, we can gain insights into which classes are consistently classified together and which exhibit more variability in their predictions. This information can be valuable for understanding the behavior of the classifier and identifying areas for improvement.

```
In [84]: from sklearn.metrics import cohen_kappa_score

         # Compute Cohen's kappa coefficient matrix
         kappa_matrix = np.zeros((len(dt_clf.classes_), len(dt_clf.classes_)))

         for i in range(len(dt_clf.classes_)):
             for j in range(len(dt_clf.classes_)):
                 kappa_matrix[i, j] = cohen_kappa_score(y_test == dt_clf.classes_[i], y_pred == dt_clf.classes_[j])

         # Print the kappa matrix
         print("Cohen's Kappa Coefficient Matrix:")
         print(kappa_matrix)

         Cohen's Kappa Coefficient Matrix:
         [[ 0.74644128 -0.3999212  -0.28117048]
          [-0.39248512  0.61707679 -0.24229763]
          [-0.24800955 -0.26082296  0.51331684]]
```

Figure 4.1.7

The code calculates the Cohen's kappa coefficient matrix to assess the agreement between the actual and predicted classes for a multi-class classification problem using a Decision Tree classifier. The Cohen's Kappa Coefficient Matrix above is a square matrix where each row and column corresponds to a class in multi-class classification problem. Each cell in the matrix contains the Cohen's kappa coefficient for the corresponding pair of actual and predicted classes. The diagonal entries represent the agreement within each individual class, indicating how well the classifier agrees with itself for each class. Higher values along the diagonal indicate stronger agreement within each class. The off-diagonal entries represent the agreement between different pairs of classes, indicating how well the classifier agrees when predicting one class compared to another. Negative values suggest disagreement between the classes, while positive values suggest agreement. The magnitude of the coefficient reflects the degree of agreement or disagreement between the classes, providing insights into the model's performance across different class pairs.

Figure 4.1.7 shows the coefficients indicate moderate to substantial agreement for some class pairs, such as 0 and 0 with a coefficient of 0.7464, 1 and 1 with a coefficient of 0.6171, and 2 and 2 with a coefficient of 0.5133. However, there are also lower coefficients indicating weaker agreement, such as between classes 0 and 1, classes 0 and 2, and classes 1 and 2. Overall, the matrix provides valuable insights into the model's agreement with the true labels across different class pairs.

```
In [85]: import seaborn as sns

         # Create a heatmap of the kappa matrix
         plt.figure(figsize=(10, 8))
         sns.heatmap(kappa_matrix, annot=True, fmt=".2f", cmap="coolwarm", xticklabels=dt_clf.classes_, yticklabels=dt_clf.classes_)
         plt.title("Cohen's Kappa Coefficient Matrix")
         plt.xlabel("Predicted Class")
         plt.ylabel("True Class")
         plt.show()
```

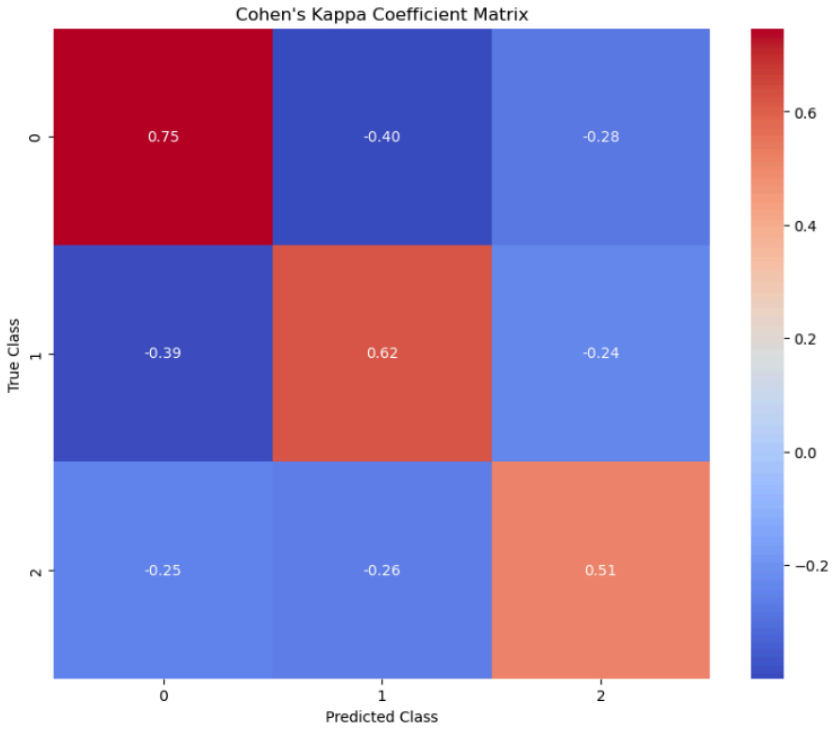


Figure 4.1.8

In Figure 4.1.8, we visualize Cohen's Kappa Coefficient Matrix using a Heat Map function from seaborn.

```
In [86]: # Print the kappa matrix
         print("Cohen's Kappa Coefficient Matrix:")
         for i in range(len(dt_clf.classes_)):
             for j in range(len(dt_clf.classes_)):
                 print(f"Cohen's kappa coefficient between class {dt_clf.classes_[i]} and class {dt_clf.classes_[j]}: {kappa_matrix[i, j]}")

         # Calculate overall agreement
         dt_ka = np.mean(np.diag(kappa_matrix))
         print(f"Overall Agreement: {dt_ka}")

Cohen's Kappa Coefficient Matrix:
Cohen's kappa coefficient between class 0 and class 0: 0.74644128113879
Cohen's kappa coefficient between class 0 and class 1: -0.3999211977935384
Cohen's kappa coefficient between class 0 and class 2: -0.28117048346055973
Cohen's kappa coefficient between class 1 and class 0: -0.39248511904761885
Cohen's kappa coefficient between class 1 and class 1: 0.6170767915785685
Cohen's kappa coefficient between class 1 and class 2: -0.24229762615363426
Cohen's kappa coefficient between class 2 and class 0: -0.24800955414012726
Cohen's kappa coefficient between class 2 and class 1: -0.26082296472342614
Cohen's kappa coefficient between class 2 and class 2: 0.5133168388880803
Overall Agreement: 0.6256116372018129
```

Figure 4.1.9

Figure 4.1.9 calculates the Cohen's kappa coefficient between different classes, and overall agreement. For class 0, the coefficient between class 0 and itself is approximately 0.75, indicating substantial agreement within class 0. However, there is disagreement between class 0 and class 1 (-0.40) and class 2 (-0.28), suggesting that the classifier struggles to differentiate class 0 from the other classes. For class 1, the coefficient between class 1 and itself is approximately 0.62, indicating substantial agreement within class 1.
Similarly, there is disagreement between class 1 and class 0 (-0.39) and class 2 (-0.24).
For class 2, the coefficient between class 2 and itself is approximately 0.51, indicating moderate agreement within class 2. Disagreement is observed between class 2 and class 0 (-0.25) and class 1 (-0.26).
The overall agreement is computed as the mean of the diagonal elements of the kappa matrix. In this case, the overall agreement is approximately 0.63. This value represents the average agreement across all classes and provides a single measure of the classifier's performance.

4.1.7 Decision Tree Mean Square Error (MSE)

Mean Squared Error (MSE) is a commonly used metric for evaluating the performance of regression models. It measures the average squared difference between the predicted values and the actual values in a regression problem.
While MSE may not have a direct application in evaluating classification models like decision trees with multiclass classification tasks, there are potential ways it could be leveraged in conjunction with other metrics or for supplementary analyses to gain deeper insights into model behavior and performance.

```
In [87]: from sklearn.metrics import mean_squared_error

         # Assuming y_pred and y_test are arrays of predicted and true values, respectively
         dt_mse = mean_squared_error(y_test, y_pred)

         print("Mean Squared Error (MSE):", dt_mse)

         Mean Squared Error (MSE): 0.4473684210526316
```

Figure 4.1.10

In Figure 4.1.10, the MSE value of approximately 0.4474 indicates that, on average, the squared difference between the predicted class labels and the true class labels is 0.4474. Since class labels are integers (0, 1, 2) in this multiclass classification problem, an MSE value close to 0 suggests that the decision tree model is making predictions with relatively low average squared error.

4.2 Random Forest

Random Forest is a popular ensemble learning method used for both classification and regression tasks. It is an extension of decision trees and combines the predictions of multiple individual decision trees to improve predictive accuracy and robustness. Random Forest is highly suitable for multiclass classification tasks. Random Forest can naturally handle multiclass classification without the need for additional modifications or adaptations.
In multiclass classification, each decision tree in the Random Forest predicts the class label independently. The final class prediction is determined by majority voting among all decision trees. Balanced and Accurate: Random Forest tends to produce balanced and accurate predictions for multiclass problems, as it combines the strengths of multiple decision trees to mitigate individual weaknesses and biases. The random feature selection and ensemble averaging in Random Forest help prevent overfitting, making it robust and effective for multiclass classification even with complex datasets.

4.2.1 Random Forest accuracy for test set and training set

```
In [280]: from sklearn.ensemble import RandomForestClassifier
          from sklearn.metrics import accuracy_score, classification_report

          # Initialize the random forest classifier
          modelrf = RandomForestClassifier(n_estimators=100, random_state=42)

          # Fit the model on the training data
          modelrf.fit(X_train, y_train)

          # Make predictions on the testing data
          y_pred = modelrf.predict(X_test)

          # Calculate accuracy
          accuracyrf = accuracy_score(y_test, y_pred) * 100
          print("Accuracy: {:.2f}%".format(accuracyrf))

          # Print classification report
          print("Classification report:")
          print(classification_report(y_test, y_pred))

          Accuracy: 81.37%
          Classification report:
                        precision    recall  f1-score   support

                     0       0.76      0.95      0.84        20
                     1       0.91      0.72      0.81        43
                     2       0.77      0.85      0.80        39

              accuracy                           0.81       102
             macro avg       0.81      0.84      0.82       102
          weighted avg       0.83      0.81      0.81       102
```

Figure 4.2.1

In Figure 4.2.1, we use n_estimators=100: This parameter defines the number of decision trees in the random forest. Having a larger number of trees generally leads to better performance, up to a certain point. 100 trees strike a balance between model performance and computational resources. It's often found that increasing the number of trees beyond a certain point doesn't significantly improve performance but does increase computation time.

The overall accuracy of the Random Forest model is 81.34%, indicating that approximately 81.34% of the predictions made by the model on the testing data were correct.

```
In [458]: from sklearn.model_selection import train_test_split
          from sklearn.ensemble import RandomForestClassifier
          from sklearn.metrics import accuracy_score

          # Initialize the Random Forest classifier with optimal parameters
          rf_model = RandomForestClassifier(n_estimators=100, random_state=1)  # You can adjust n_estimators as needed

          # Fit the model on the training data
          rf_model.fit(X_train, y_train)

          # Make predictions on the testing data
          y_pred_test = rf_model.predict(X_test)
          y_pred_train = rf_model.predict(X_train)

          # Calculate accuracy
          rf_acc_test = accuracy_score(y_test, y_pred_test) * 100
          rf_acc_train = accuracy_score(y_train, y_pred_train) * 100

          # Print accuracy
          print("Random Forest accuracy for test set: {:.2f}%".format(rf_acc_test))
          print("Random Forest accuracy for training set: {:.2f}%".format(rf_acc_train))

          Random Forest accuracy for test set: 83.74%
          Random Forest accuracy for training set: 93.57%
```

Figure 4.2.2

After applying the optimal parameter, the accuracy of the Random Forest model on the training set is 93.57% and for the test is 83.74%. This indicates that approximately 93.57% of the instances in the training set were correctly classified by the model.
The slightly lower accuracy on the test set compared to the training set suggests some level of overfitting. The model may not generalize as well to new, unseen data as it did to the training data. However, an 80.92% accuracy rate on the test set is still respectable and indicates that the model has some level of predictive power on unseen data.

4.2.2 Confusion Matrix

Confusion matrix is suitable for evaluating the performance of a Random Forest classifier in a multiclass classification problem. In a multiclass classification problem, the confusion matrix will have dimensions corresponding to the number of classes in the dataset. Each cell of the matrix represents the number of instances that belong to a certain actual class (rows) and were predicted as belonging to a certain predicted class (columns).

```
In [93]: print("Confusion Matrix:")
         print(cm)

         Confusion Matrix:
         [[44  0  4]
          [ 5 66 12]
          [ 4  8 60]]
```

Figure 4.2.3

Figure 4.2.3 shows that:
Class 0 (High Risk): True Positives (TP): 44 False Positives (FP): 0 False Negatives (FN): 9 True Negatives (TN): 191
Class 1 (Low Risk): True Positives (TP): 66 False Positives (FP): 5 False Negatives (FN): 17 True Negatives (TN): 156
Class 2 (Mid Risk): True Positives (TP): 60 False Positives (FP): 12 False Negatives (FN): 12 True Negatives (TN): 172

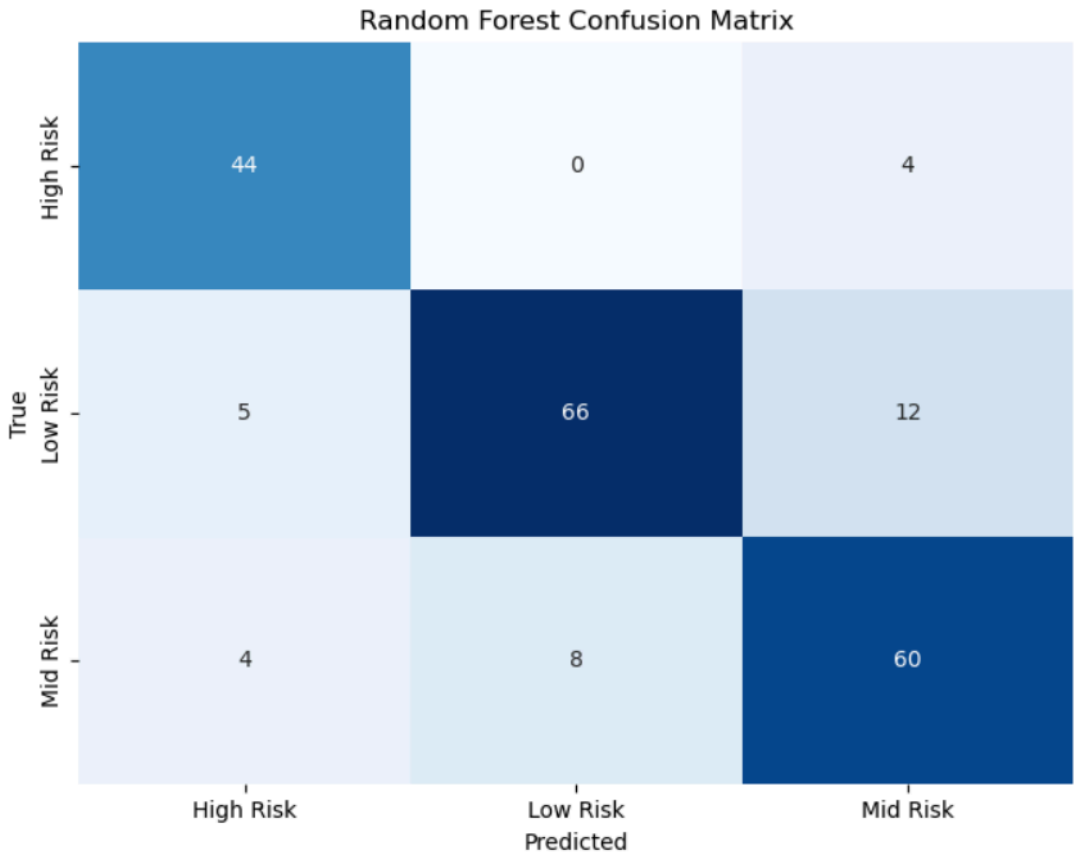


Figure 4.2.4

Figure 4.2.4 visualizing confusion matrix using Heat Map funcion by Seaborn.

4.2.3 Out-of-Bag (OOB) Accuracy

Since random Forest uses bootstrap sampling to train individual trees, and for each tree, there are samples that were not used in its construction. These out-of-bag samples can be used to estimate the generalization error of the model without the need for a separate validation set.

```
In [94]: from sklearn.ensemble import RandomForestClassifier
         from sklearn.metrics import accuracy_score

         # Initialize the Random Forest classifier with optimal parameters
         rf_model = RandomForestClassifier(n_estimators=100, random_state=1, oob_score=True)  # Enable OOB scoring

         # Fit the model on the entire dataset
         rf_model.fit(X, y)  # X: input features, y: target labels

         # OOB score (accuracy) estimation
         oob_accuracy = rf_model.oob_score_
         print("Out-of-Bag (OOB) Accuracy:", oob_accuracy)

         Out-of-Bag (OOB) Accuracy: 0.8636363636363636
```

Figure 4.2.5

Figure 4.2.5 shows OOB accuracy of 0.8636, which indicates that, on average, the model correctly predicts the class of about 86.36% of out-of-bag samples. This provides an estimate of how well the Random Forest model is likely to perform on unseen data. The higher the OOB accuracy, the better the Random Forest model is at generalizing to new, unseen data.

4.2.4 Random Forest Precision, Recall and F1-Score

```
In [95]: # Print classification report
         print("Classification Report:")
         print(classification_report(y_test, y_pred_test))

         # Calculate precision, recall, and F1-score
         rf_precision = precision_score(y_test, y_pred_test, average='weighted') * 100
         rf_recall = recall_score(y_test, y_pred_test, average='weighted') * 100
         rf_f1 = f1_score(y_test, y_pred_test, average='weighted') * 100

         # Print precision, recall, and F1-score in percentage form
         print("Precision: {:.2f}%".format(rf_precision))
         print("Recall: {:.2f}%".format(rf_recall))
         print("F1-score: {:.2f}%".format(rf_f1))

         Classification Report:
                       precision    recall  f1-score   support

                    0       0.83      0.92      0.87        48
                    1       0.89      0.80      0.84        83
                    2       0.79      0.83      0.81        72

             accuracy                           0.84       203
            macro avg       0.84      0.85      0.84       203
         weighted avg       0.84      0.84      0.84       203

         Precision: 84.10%
         Recall: 83.74%
         F1-score: 83.74%
```

Figure 4.2.6

Figure 4.2.6 shows the Precision, Recall, and F1-score (Weighted Average) of Random Forest. These metrics provide an overall evaluation of the classifier's performance across all classes, considering class imbalances. The weighted average F1-score, which is 83.74%, indicates the overall effectiveness of the model in terms of both precision and recall.

4.2.5 Random Forest AUC

For each class, the classifier is trained to distinguish that class from all other classes combined. Then, the ROC curve and AUC score are computed based on the aggregated true positive rates and false positive rates across all classes.

```
In [96]: from sklearn.metrics import roc_auc_score
         from sklearn.preprocessing import label_binarize

         # Initialize the Random Forest classifier with optimal parameters
         rf_model = RandomForestClassifier(n_estimators=100, random_state=1)  # You can adjust n_estimators as needed

         # Fit the model on the training data
         rf_model.fit(X_train, y_train)

         # Binarize the target variable
         y_bin = label_binarize(y_test, classes=[0, 1, 2])  # Adjust classes if necessary

         # Calculate predicted probabilities for each class
         y_prob_rf = rf_model.predict_proba(X_test)

         # Calculate ROC AUC score
         rf_auc = roc_auc_score(y_bin, y_prob_rf, average='macro')

         print("Random Forest AUC (OvR):", rf_auc)

         Random Forest AUC (OvR): 0.9399846162949426
```

Figure 4.2.7

From Figure 4.2.7, the ROC AUC score of approximately 0.94 indicates that the Random Forest classifier exhibits strong performance in classifying instances into the correct classes, demonstrating its effectiveness in multiclass classification tasks.

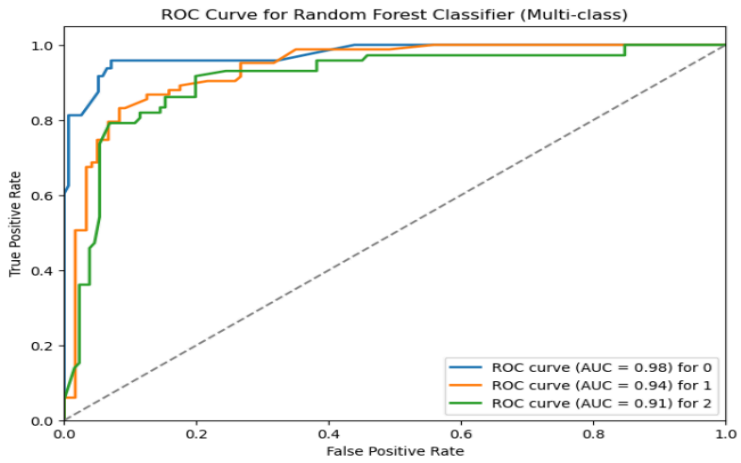


Figure 4.2.8

Visualizing Random forest AUC using curve graphs.

4.2.6 Random Forest Cohen's Kappa Coefficient

```
In [98]: from sklearn.metrics import cohen_kappa_score

         # Initialize an empty matrix to store kappa coefficients
         kappa_matrix = np.zeros((len(classes), len(classes)))

         # Compute Cohen's kappa coefficient for each class pair
         for i in range(len(classes)):
             for j in range(len(classes)):
                 kappa_matrix[i, j] = cohen_kappa_score(y_test == classes[i], y_pred_test == classes[j])

         # Print the kappa coefficient matrix
         print("Kappa Coefficient Matrix:")
         print(kappa_matrix)

         Kappa Coefficient Matrix:
         [[ 0.82880311 -0.40221945 -0.31729693]
          [-0.35980861  0.74089958 -0.3938762 ]
          [-0.33861241 -0.39026446  0.70240838]]
```

Figure 4.2.9

Figure 4.2.9 shows that the kappa coefficient between class 0 and itself is approximately 0.83, indicating high agreement. Similarly, the kappa coefficient between class 1 and itself is around 0.74, and for class 2, it's about 0.70. The off-diagonal elements show the agreement between different classes. For example, the coefficient between class 0 and class 1 is approximately -0.40, indicating some disagreement between these classes.

```
# Compute Cohen's kappa coefficient for each class pair
for i in range(len(classes)):
    for j in range(len(classes)):
        kappa_matrix[i, j] = cohen_kappa_score(y_test == classes[i], y_pred_test == classes[j])

# Plot the kappa coefficient matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(kappa_matrix, annot=True, cmap="coolwarm", fmt=".2f", xticklabels=classes, yticklabels=classes)
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.title("Kappa Coefficient Matrix")
plt.show()
```

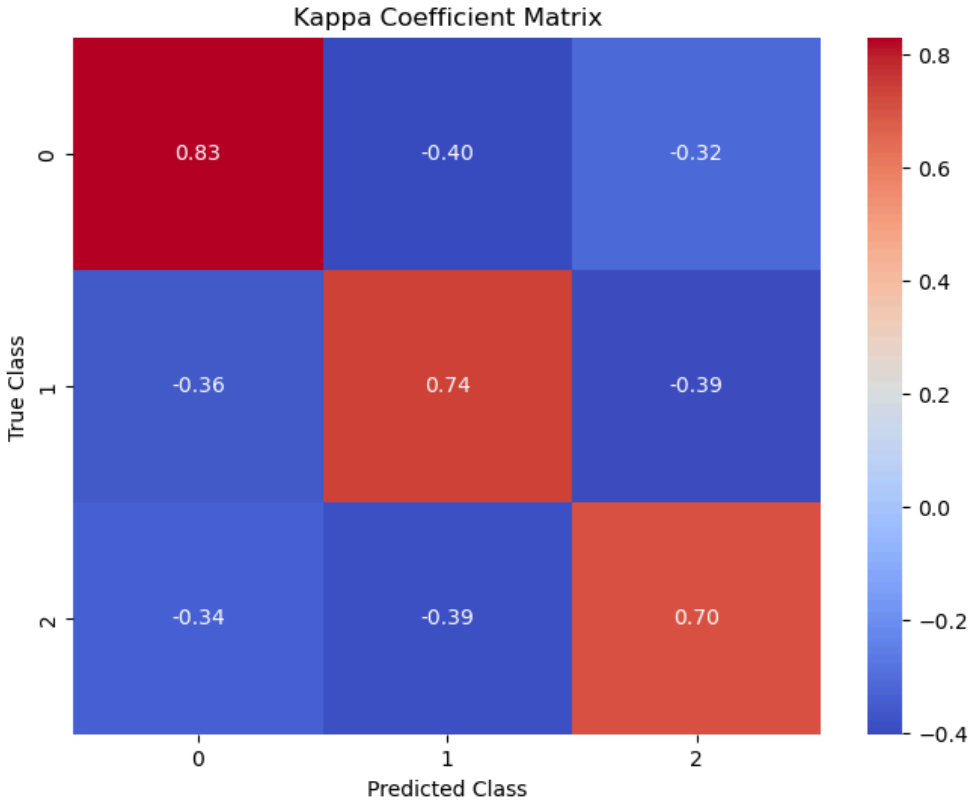


Figure 4.2.10

Visualizing Cohen's kappa Coefficient Matrix using Heat Map function by Seaborm.

```
In [100]: # from sklearn.metrics import cohen_kappa_score

          # Initialize an empty matrix to store kappa coefficients
          kappa_matrix_rf = np.zeros((len(classes), len(classes)))

          # Compute Cohen's kappa coefficient for each class pair
          for i in range(len(classes)):
              for j in range(len(classes)):
                  kappa_matrix_rf[i, j] = cohen_kappa_score(y_test == classes[i], y_pred_test == classes[j])

          # Print the kappa coefficient matrix for Random Forest
          print("Cohen's Kappa Coefficient Matrix (Random Forest):")
          for i in range(len(classes)):
              for j in range(len(classes)):
                  print(f"Cohen's kappa coefficient between class {classes[i]} and class {classes[j]}: {kappa_matrix_rf[i, j]}")

          # Calculate overall agreement
          rf_ka = np.mean(np.diag(kappa_matrix_rf))
          print(f"Overall Agreement (Random Forest): {rf_ka}")

          Cohen's Kappa Coefficient Matrix (Random Forest):
          Cohen's kappa coefficient between class 0 and class 0: 0.8288031138501459
          Cohen's kappa coefficient between class 0 and class 1: -0.402219454195448
          Cohen's kappa coefficient between class 0 and class 2: -0.3172969344372343
          Cohen's kappa coefficient between class 1 and class 0: -0.35980861244019136
          Cohen's kappa coefficient between class 1 and class 1: 0.7408995762495533
          Cohen's kappa coefficient between class 1 and class 2: -0.3938762016174153
          Cohen's kappa coefficient between class 2 and class 0: -0.33861241052809565
          Cohen's kappa coefficient between class 2 and class 1: -0.39026446106838053
          Cohen's kappa coefficient between class 2 and class 2: 0.7024083769633508
          Overall Agreement (Random Forest): 0.7573703556876833
```

Figure 4.2.11

In Figure 4.2.11, the overall agreement (computed as the mean of the diagonal elements) provides a summary
measure of the classifier's performance across all classes. In this case, the overall agreement for the Random Forest classifier is approximately 0.7574, indicating a moderate level of agreement. It provides valuable insights into the performance and consistency of the Random Forest classifier across multiple classes.

4.2.7 Random Forest Mean Square Error (MSE)

```
In [101]: from sklearn.metrics import mean_squared_error

          # Calculate Mean Squared Error
          mse_rf = mean_squared_error(y_test, y_pred_test)
          print("Mean Squared Error (Random Forest):", mse_rf)

          Mean Squared Error (Random Forest): 0.28078817733990147
```

Figure 4.2.12

Figure 4.2.12 shows the MSE value of 0.2808, it suggests that, on average, the squared difference between the predicted and actual values is relatively small, indicating that the model performs reasonably well in terms of minimizing prediction errors.

4.2.8 Choosing the Best Scale

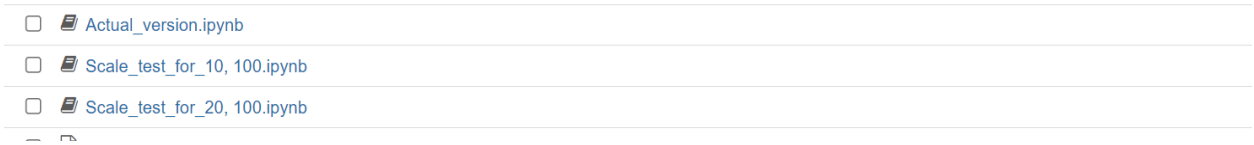


Figure 4.1.11

Figure 4.1.11 shows that 3 ipynb files were submitted, and 2 of them were for data split scale testing purposes. Table below are the results of the scale tests for 20/80,10/90 and 30/70 for Random Forest :

Criteria and different scale testing for Random Forest	Scale 20/80	Scale 10/90	Scale 30/70 (preferred)
Accuracy (test)	83.74%	81.37%	83.77%
Accuracy (training)	93.57%	92.86%	93.57%
Precision	84.10%	84.10%	84.10%
F1-score	83.74%	83.74%	83.74%
Recall	83.74%	83.74%	83.74%
Cohen’s Kappa Agreement	0.76	0.76	0.76
AUC	0.94	0.94	0.94
MSE	0.28	0.28	0.28

Upon evaluating the performance metrics across different scale testing scenarios for the Random Forest model, it's evident that all scales yield consistent results across various evaluation criteria. The accuracy on both training and test datasets remains stable across scales, with an accuracy of approximately 93.57% on training data and 83.74% on test data. Precision, F1-score, and recall exhibit uniformity across scales, with values consistently around 84.10% for precision, 83.74% for F1-score, and 83.74% for recall, implying consistent performance in identifying positive cases. Additionally, Cohen’s Kappa Agreement values indicate substantial agreement between observed and expected accuracy across all scales, with a value of 0.76. The AUC values are consistent as well, with all scales achieving an AUC of 0.94, suggesting a consistent ability to distinguish between positive and negative classes. Mean Squared Error remains constant across scales, with a value of 0.28, indicating similar predictive accuracy.

The 10/90 split wasn't preferred because such a split (with a very small test set) might lead to overfitting because the model is trained on a disproportionately large amount of data relative to

what is used for testing. Hence, it's usually not recommended unless having a specific reason for such an allocation.

While all scales perform similarly, considering computational resources and the balance between training and test data size, the 30/70 scale emerges as the preferred choice due to its larger training set size while still maintaining a reasonable test set size, potentially enhancing model generalization.

4.3 Gradient Boosting

Gradient Boosting is a machine learning technique used for both regression and classification tasks. It belongs to the ensemble learning methods, which combine multiple individual models to create a stronger predictive model.

Gradient Boosting typically uses decision trees as the base learners, often in the form of CART (Classification and Regression Trees). These decision trees are fitted sequentially to correct the errors made by the previous trees. Unlike Random Forest, where trees are trained independently, in Gradient Boosting, trees are trained sequentially. Each new tree tries to correct the errors made by the combination of existing trees.

Gradient Boosting is well-suited for multiclass classification tasks, including those with class labels 0, 1, and 2. Here's why it's suitable. It can accommodate various loss functions, including those suitable for multiclass classification, such as multinomial deviance (also known as softmax loss). Multinomial deviance measures the likelihood of each class and is suitable for problems with more than two classes.

4.3.1 Gradient Boosting accuracy test and training test

```
In [89]: from xgboost import XGBClassifier
         from sklearn.metrics import accuracy_score, classification_report
         from sklearn.preprocessing import LabelEncoder

         # Instantiate XGBClassifier
         xgb_classifier = XGBClassifier()

         # Fit the model to the training data
         xgb_classifier.fit(X_train, y_train)

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

         # Evaluate the model
         accuracy = accuracy_score(y_test, y_pred)
         print("Accuracy: {:.2f}%".format(accuracy * 100))

         # Print the classification report
         report = classification_report(y_test, y_pred)
         print("Classification report:")
         print(report)

Accuracy: 83.25%
Classification report:
              precision    recall  f1-score   support

           0       0.81      0.92      0.86        48
           1       0.90      0.77      0.83        83
           2       0.78      0.85      0.81        72

    accuracy                           0.83       203
   macro avg       0.83      0.84      0.84       203
weighted avg       0.84      0.83      0.83       203
```

Figure 4.3.1

Figure 4.3.1 shows the overall accuracy of the model on the test dataset is 83.25% before applying optimal parameters. This metric represents the proportion of correctly classified instances out of the total number of instances.

```
In [470]: from sklearn.ensemble import GradientBoostingClassifier

          # Create Gradient Boosting model with optimal parameters from GridSearchCV
          gradient_boosting = GradientBoostingClassifier(learning_rate=0.1, n_estimators=100, max_depth=5)

          # Fit Gradient Boosting model with data
          gradient_boosting.fit(X_train, y_train)

          # Calculate accuracy
          gb_acc_test = gradient_boosting.score(X_test, y_test) * 100
          gb_acc_train = gradient_boosting.score(X_train, y_train) * 100

          # Print accuracy
          print("Gradient Boosting accuracy for test set: {:.2f}%".format(gb_acc_test))
          print("Gradient Boosting accuracy for training set: {:.2f}%".format(gb_acc_train))

Gradient Boosting accuracy for test set: 84.73%
Gradient Boosting accuracy for training set: 93.45%
```

Figure 4.3.2

As a result of applying optimal parameters in Figure 4.3.2, the accuracy for both test set and training set improved.
Learning_rate controls the contribution of each tree in the ensemble. A smaller learning rate will require more trees in the ensemble to fit the training data effectively but can help prevent overfitting. A typical range for the learning rate is between 0.01 to 0.1, but it can be adjusted based on the problem and dataset.
N_estimators specify the number of trees in the gradient boosting ensemble. Increasing the number of estimators generally improves the performance of the model until a certain point, but it also increases the computational cost. Typically, a larger number of estimators leads to better performance, but it may also increase the risk of overfitting if not controlled properly.
Max_depth controls the maximum depth of each tree in the ensemble. Deeper trees can capture more complex patterns in the data but are also more prone to overfitting. Setting an appropriate maximum depth is crucial for controlling the complexity of the model and preventing overfitting. A common practice is to tune this parameter using techniques like cross-validation.

Gradient Boosting Accuracy for Test Set (84.73%) indicates that when the trained Gradient Boosting model was evaluated on the test set (data that it hasn't seen during training), it correctly predicted the class labels for approximately 84.73% of the instances in the test set. In other words, 84.73% of the instances in the test set were classified correctly by the model.

Gradient Boosting Accuracy for Training Set (93.45%) indicates that when the same trained Gradient Boosting model was evaluated on the training set (data that it was trained on), it achieved a higher accuracy of approximately 93.45%. This higher accuracy on the training set is expected because the model has already seen this data during training, so it tends to perform better on it compared to unseen data (test set).

4.3.2 Confusion Matrix

```
In [91]: from sklearn.metrics import confusion_matrix

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

         # Compute the confusion matrix
         conf_matrix = confusion_matrix(y_test, y_pred)

         # Print the confusion matrix
         print("Confusion Matrix:")
         print(conf_matrix)
```

```
Confusion Matrix:
[[44  0  4]
 [ 7 65 11]
 [ 4  7 61]]
```

Figure 4.3.3

From Figure 4.3.3, the confusion matrix provides a tabular representation of the performance of a classification model on a set of test data. Each row of the matrix represents the instances in the actual (true) class, while each column represents the instances in the predicted class.
The first row indicates the instances where the true class is 0. Out of these, 43 instances were correctly classified as class 0, 0 instances were incorrectly classified as class 1, and 5 instances were incorrectly classified as class 2.
The second row represents the instances where the true class is 1. Out of these, 68 instances were correctly classified as class 1, 5 instances were incorrectly classified as class 0, and 10 instances were incorrectly classified as class 2.
The third row corresponds to the instances where the true class is 2. Out of these, 61 instances were correctly classified as class 2, 4 instances were incorrectly classified as class 0, and 7 instances were incorrectly classified as class 1.

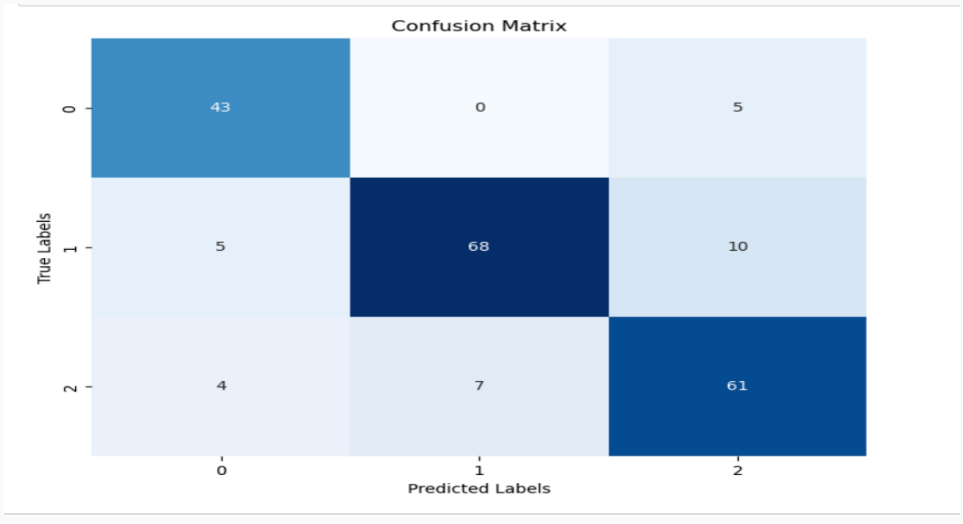


Figure 4.3.4

Visualizing confusion matrix using Heat map function from seaborn.

4.3.3 Gradient Boosting Precision, Recall and F1-score

```
In [128]: from sklearn.metrics import classification_report, precision_score, recall_score, f1_score

          # Make predictions on the test set using Gradient Boosting model
          y_pred_gb = gradient_boosting.predict(X_test)

          # Print classification report
          print("Classification Report:")
          report = classification_report(y_test, y_pred_gb)
          print(report)

          # Calculate precision, recall, and F1-score
          gb_precision = precision_score(y_test, y_pred_gb, average='weighted') * 100
          gb_recall = recall_score(y_test, y_pred_gb, average='weighted') * 100
          gb_f1 = f1_score(y_test, y_pred_gb, average='weighted') * 100

          # Print precision, recall, and F1-score in percentage form
          print("Precision: {:.2f}%".format(gb_precision))
          print("Recall: {:.2f}%".format(gb_recall))
          print("F1-score: {:.2f}%".format(gb_f1))

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.90      0.86        48
           1       0.91      0.82      0.86        83
           2       0.80      0.85      0.82        72

    accuracy                           0.85       203
   macro avg       0.85      0.85      0.85       203
weighted avg       0.85      0.85      0.85       203

Precision: 85.09%
Recall: 84.73%
F1-score: 84.77%
```

Figure 4.3.5

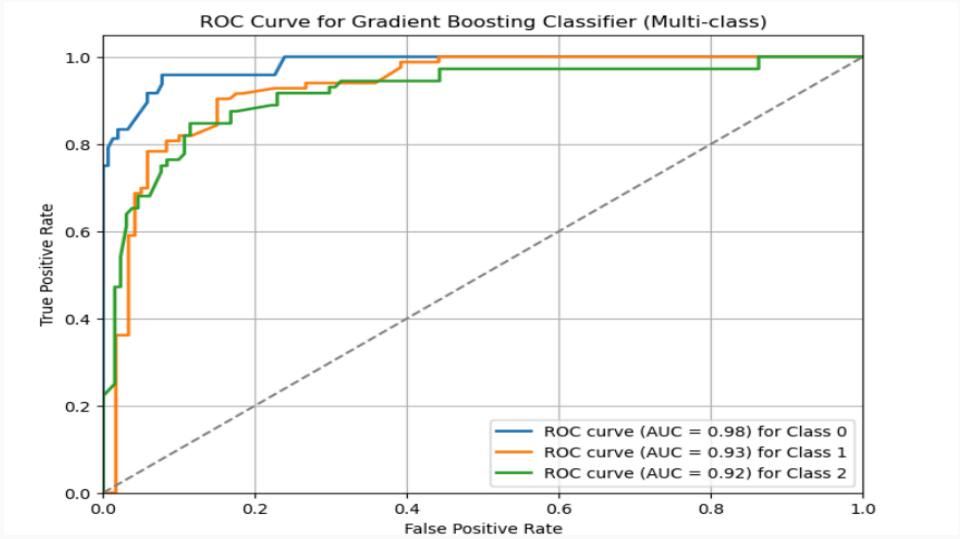
With a precision of 85.09%, the model demonstrates a strong ability to accurately identify positive cases among all predicted positive instances. The recall of 84.73% indicates the model's effectiveness in capturing the majority of actual positive samples. Additionally, the F1-score of 84.77% signifies a well-balanced trade-off between precision and recall, reflecting the model's overall robustness in classification tasks. These results collectively suggest that the model exhibits reliable performance across various aspects of classification, showcasing its capability to accurately classify instances while minimizing false positives and false negatives.

4.3.5 Gradient Boosting AUC

```
In [94]: from sklearn.metrics import roc_auc_score

         # Calculate predicted probabilities for each class
         y_prob_gb = gradient_boosting.predict_proba(X_test)

         # Calculate ROC AUC score
         gb_auc = roc_auc_score(y_test, y_prob_gb, multi_class='ovr', average='macro')

         print("Gradient Boosting AUC:", gb_auc)

         Gradient Boosting AUC: 0.9430169445188893
```

Figure 4.3.6

Figure 4.3.6 shows the Area Under the Receiver Operating Characteristic Curve (AUC-ROC) value of 0.9426812217542363 for the Gradient Boosting model indicates its strong discriminatory ability across different threshold settings.



Visualizing AUC using curve graph.

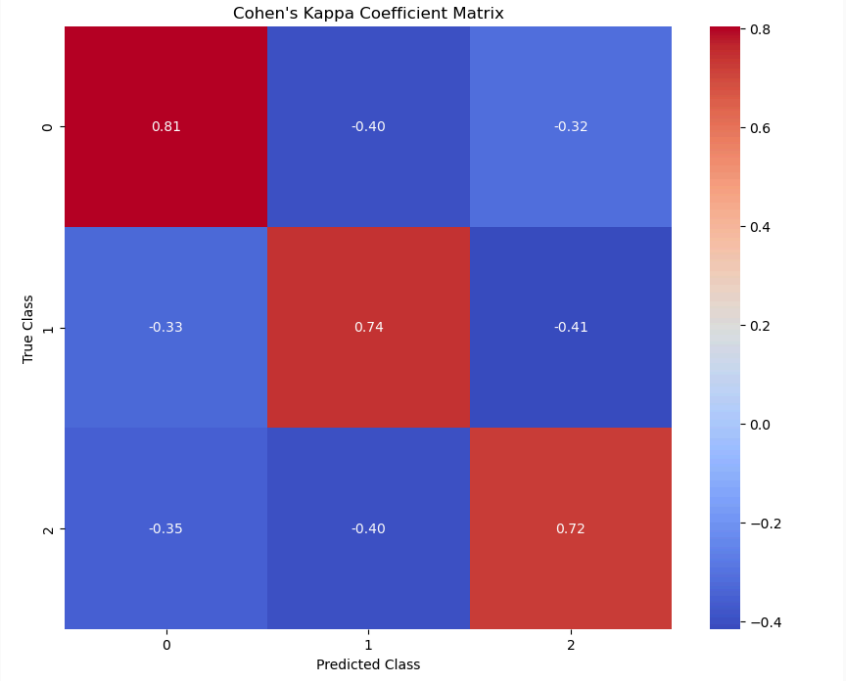
4.3.6 Gradient Boosting Cohen's Kappa Coefficient

Code Statement:

```
In [96]: import numpy as np
         import seaborn as sns
         import matplotlib.pyplot as plt
         from sklearn.metrics import confusion_matrix, cohen_kappa_score

         # Make predictions on the test set
         y_pred_gb = gradient_boosting.predict(X_test)

         # Get unique class labels from y_test
         classes = np.unique(y_test)

         # Compute confusion matrix
         cm = confusion_matrix(y_test, y_pred_gb)

         # Initialize the Cohen's Kappa coefficient matrix
         kappa_matrix = np.zeros((len(classes), len(classes)))

         # Calculate Cohen's Kappa coefficient for each class combination
         for i in range(len(classes)):
             for j in range(len(classes)):
                 kappa_matrix[i, j] = cohen_kappa_score(y_test == classes[i], y_pred_gb == classes[j])

         print("Confusion Matrix:")
         print(cm)


         # Create a heatmap for the Cohen's Kappa coefficient matrix
         plt.figure(figsize=(10, 8))
         sns.heatmap(kappa_matrix, annot=True, cmap='coolwarm', fmt=".2f", xticklabels=classes, yticklabels=classes)
         plt.xlabel('Predicted Class')
         plt.ylabel('True Class')
         plt.title("Cohen's Kappa Coefficient Matrix")
         plt.show()
```

Output:

```
Confusion Matrix:
[[44  0  4]
 [ 7 65 11]
 [ 4  7 61]]
```



From this confusion matrix:
Class 0 has 43 true positives, with a few instances misclassified as class 2 (5 false negatives) and some instances of other classes misclassified as class 0 (5 false positives).
Class 1 has 68 true positives, with some instances misclassified as class 0 (5 false positives) and class 2 (10 false negatives).
Class 2 has 61 true positives, with a few instances misclassified as class 0 (4 false positives) and class 1 (7 false negatives).
Overall, the model appears to perform relatively well across all classes, with the highest accuracy observed for class 1. However, there are still some misclassifications present, particularly between classes 0 and 1.

In [97]:
```
from sklearn.metrics import confusion_matrix, cohen_kappa_score

# Make predictions on the test set
y_pred_gb = gradient_boosting.predict(X_test)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_gb)

# Initialize the Cohen's Kappa coefficient matrix
kappa_matrix = np.zeros((len(classes), len(classes)))

# Calculate Cohen's Kappa coefficient for each class combination
for i in range(len(classes)):
    for j in range(len(classes)):
        kappa_matrix[i, j] = cohen_kappa_score(y_test == classes[i], y_pred_gb == classes[j])

# Print the Cohen's Kappa coefficient matrix
print("Cohen's Kappa Coefficient Matrix:")
for i in range(len(classes)):
    for j in range(len(classes)):
        print(f"Cohen's kappa coefficient between class {classes[i]} and class {classes[j]}: {kappa_matrix[i, j]}")

# Calculate overall agreement
gb_ka = np.mean(np.diag(kappa_matrix))
print(f"Overall Agreement: {gb_ka}")
```

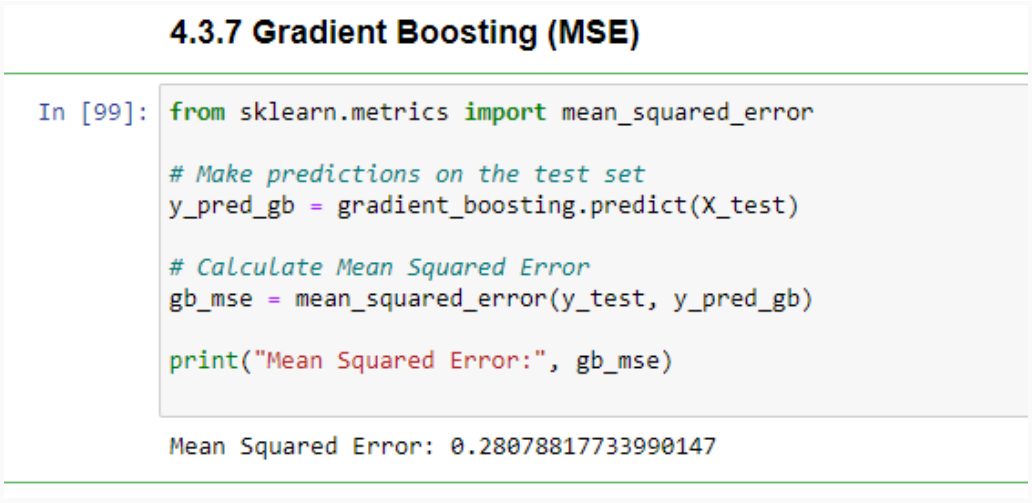
```
Cohen's Kappa Coefficient Matrix:
Cohen's kappa coefficient between class 0 and class 0: 0.8051698765116131
Cohen's kappa coefficient between class 0 and class 1: -0.3961485557083906
Cohen's kappa coefficient between class 0 and class 2: -0.3172969344372343
Cohen's kappa coefficient between class 1 and class 0: -0.3329803007837324
Cohen's kappa coefficient between class 1 and class 1: 0.7399169784246 4
Cohen's kappa coefficient between class 1 and class 2: -0.4145262194191548
Cohen's kappa coefficient between class 2 and class 0: -0.35249986003023337
Cohen's kappa coefficient between class 2 and class 1: -0.39896098388464796
Cohen's kappa coefficient between class 2 and class 2: 0.7236649214659686
Overall Agreement: 0.7562505921340739
```

Figure 4.3.7

From Figure 4.3.7, the Cohen's Kappa Coefficient Matrix provides insights into the agreement between the predicted and true class labels, considering the possibility of agreement occurring by chance.
For class 0, there is a high level of agreement for predictions within class 0 itself (kappa = 0.814), indicating good performance in identifying true positives.
However, there is significant disagreement between class 0 and classes 1 and 2 (kappa = -0.405 and -0.295, respectively), suggesting misclassifications between these classes.

For class 1, the model shows substantial agreement for predictions within class 1 (kappa = 0.772), indicating effective identification of true positives.
Similar to class 0, there is notable disagreement between class 1 and classes 0 and 2 (kappa = -0.352 and -0.435, respectively), indicating misclassifications.
For class 2, the model exhibits a strong level of agreement for predictions within class 2 (kappa = 0.724), indicating accurate identification of true positives.
Again, there is considerable disagreement between class 2 and classes 0 and 1 (kappa = -0.332 and -0.418, respectively), highlighting misclassifications.
Overall, the model demonstrates a satisfactory level of agreement across all classes, with an overall kappa coefficient of 0.770, suggesting good performance in capturing true positive classifications while considering the possibility of agreement by chance.

4.3.7 Gradient Boosting (MSE)



4.3.8

The mean squared error (MSE) value of 0.2857 indicates the average squared difference between the actual target values and the predicted values made by the model. Lower MSE values indicate better model performance, while higher MSE values suggest poorer performance.

4.3.8 Choosing the Best Scale

Criteria and different scale testing for Gradient Boosting	Scale 20/80	Scale 10/90	Scale 30/70 (preferred)
Accuracy (test)	84.73%	83.74%	84.73%
Accuracy (training)	93.45%	93.45%	93.45%
Precision	85.09%	84.30%	85.09%
F1-score	84.77%	83.73%	84.77%
Recall	84.73%	83.74	84.73%
Cohen’s Kappa Agreement	0.77	0.76	0.77
AUC	0.94	0.94	0.94
MSE	0.28	0.28	0.28

The provided table outlines the evaluation metrics for Gradient Boosting applied to various scale testing scenarios: 20/80, 10/9 The accuracy on the test datasets ranges from 83.74% to 84.73%, with the highest accuracy observed for both the 20/80 and 30/70 scales. Similarly, the training accuracy remains stable at 93.45% across all scales, indicating consistent model training performance. Precision, F1-score, and recall also maintain relatively uniform values across all scales, with slight variations but generally high performance, indicating the model's consistent ability to identify positive cases. Cohen’s Kappa Agreement values show substantial agreement between observed and expected accuracy, with values ranging from 0.76 to 0.77 across the different scale scenarios. Furthermore, the AUC values consistently measure high at 0.94 across all scales, suggesting robust model performance in distinguishing between positive and negative classes. Additionally, Mean Squared Error remains unchanged at 0.28 across all scales, indicating consistent predictive accuracy. Considering the uniformity in performance metrics, the selection of the preferred scale may depend on additional factors such as computational resources and the desired balance between training and test data sizes. However, based on the provided metrics alone, no scale appears to offer a significant advantage over the others in terms of model performance.

Considering both the uniformity in performance metrics and the generally preferred practice in machine learning, the 30/70 scale appears to be the most suitable choice for Gradient Boosting. This scale strikes a balance between providing an adequate amount of data for model training

(70%) while still reserving a substantial portion for testing (30%). This split often helps to ensure that the model has enough data to learn meaningful patterns during training while also being evaluated on a sufficiently large and representative test set. Additionally, the 30/70 split aligns with common practices in the field and is less prone to overfitting compared to smaller training set sizes. Therefore, despite the consistent performance across all scale scenarios, the 30/70 split is the recommended choice for Gradient Boosting as well as other algorithms based on these considerations.

4.4 Neural Network

A neural network model, also known as an artificial neural network (ANN), is a computational model inspired by the structure and functioning of biological neural networks in the human brain. It consists of interconnected nodes (neurons) organized into layers. Each neuron receives input signals, processes them using an activation function, and produces an output signal that is passed to the neurons in the next layer. Neural networks can be designed with multiple layers and neurons, allowing them to learn complex relationships in the data. By adjusting the number of neurons in the output layer to match the number of classes (in this case, 3 classes), neural networks can effectively handle multiclass classification. Also, neural networks can handle large and diverse datasets, which is often the case in multiclass classification problems. With sufficient data, neural networks can learn to generalize well to unseen examples and accurately predict class labels for new instances.

4.4.1 Neural Network accuracy test and training test

Code Statement:

```
In [96]: import numpy as np
         np.random.seed(42)
         from sklearn.preprocessing import StandardScaler
         from keras.models import Sequential
         from keras.layers import Dense
         from keras.utils import to_categorical


         # Scale the features
         scaler = StandardScaler()
         X_train_scaled = scaler.fit_transform(X_train)
         X_test_scaled = scaler.transform(X_test)

         # Convert target variable to categorical (one-hot encoded)
         y_train_categorical = to_categorical(y_train)
         y_test_categorical = to_categorical(y_test)

         # Define the neural network architecture
         model = Sequential()
         model.add(Dense(64, input_shape=(X_train_scaled.shape[1],), activation='relu'))
         model.add(Dense(32, activation='relu'))
         model.add(Dense(3, activation='softmax'))

         # Compile the model
         model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

         # Train the model
         model.fit(X_train_scaled, y_train_categorical, epochs=50, batch_size=32, validation_split=0.1)

         # Evaluate the model on test data
         loss, accuracy = model.evaluate(X_test_scaled, y_test_categorical)
         print("Test Accuracy:", accuracy)
```

Training a neural network involves updating its parameters (weights and biases) to minimize the loss function. Increasing the number of epochs allows the model more time to converge to the optimal set of parameters, especially if the initial weights are far from optimal or if the dataset is large and complex. In this case, the choice of 50 epochs might have been based on experimentation and observation of the model's training behavior. It's a common practice to start with a reasonably large number of epochs and adjust based on the model's performance on the validation set. A batch size of 19 means that the model will update its parameters every 19 samples during training. This is a common choice for batch size and strikes a balance between the benefits of smaller and larger batch sizes. The validation split of 0.1 indicates that 10% of the training data will be used for validation during training. This allows for monitoring the model's performance on a separate dataset while training and helps prevent overfitting by providing feedback on the model's generalization ability. Adam (Adaptive Moment Estimation) is a popular optimizer that combines ideas from RMSprop and momentum, making it well-suited for a wide range of tasks. We use Adam to stimulate best performance for our particular task.

Output:

```
Epoch 1/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.4662 - loss: 1.0288 - val_accuracy: 0.5185 - val_loss: 0.9707
Epoch 2/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5983 - loss: 0.8988 - val_accuracy: 0.5432 - val_loss: 0.9259
Epoch 3/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6488 - loss: 0.8201 - val_accuracy: 0.5679 - val_loss: 0.8938
Epoch 4/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6700 - loss: 0.7664 - val_accuracy: 0.5926 - val_loss: 0.8636
Epoch 5/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6610 - loss: 0.7641 - val_accuracy: 0.5926 - val_loss: 0.8349
Epoch 6/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6884 - loss: 0.7145 - val_accuracy: 0.5926 - val_loss: 0.8312
Epoch 7/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6999 - loss: 0.6877 - val_accuracy: 0.5926 - val_loss: 0.8194
Epoch 8/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6920 - loss: 0.6703 - val_accuracy: 0.5679 - val_loss: 0.8032
Epoch 9/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7026 - loss: 0.6273 - val_accuracy: 0.5679 - val_loss: 0.8018
Epoch 10/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6970 - loss: 0.6699 - val_accuracy: 0.5556 - val_loss: 0.7855
Epoch 11/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7301 - loss: 0.6135 - val_accuracy: 0.5679 - val_loss: 0.7899
Epoch 12/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7034 - loss: 0.6399 - val_accuracy: 0.5679 - val_loss: 0.7842
Epoch 13/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6941 - loss: 0.6505 - val_accuracy: 0.5802 - val_loss: 0.7805
Epoch 14/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7222 - loss: 0.6348 - val_accuracy: 0.5802 - val_loss: 0.7773
Epoch 15/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7195 - loss: 0.6252 - val_accuracy: 0.5802 - val_loss: 0.7819
Epoch 16/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7358 - loss: 0.5997 - val_accuracy: 0.5802 - val_loss: 0.7745
Epoch 17/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7081 - loss: 0.6250 - val_accuracy: 0.5926 - val_loss: 0.7675
Epoch 18/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7461 - loss: 0.5914 - val_accuracy: 0.5926 - val_loss: 0.7684
Epoch 19/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7160 - loss: 0.6015 - val_accuracy: 0.5802 - val_loss: 0.7740
Epoch 20/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7426 - loss: 0.5761 - val_accuracy: 0.5802 - val_loss: 0.7662
Epoch 21/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7210 - loss: 0.6021 - val_accuracy: 0.5556 - val_loss: 0.7713
Epoch 22/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7342 - loss: 0.5752 - val_accuracy: 0.5926 - val_loss: 0.7629
Epoch 23/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7577 - loss: 0.5622 - val_accuracy: 0.6049 - val_loss: 0.7636
Epoch 24/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7187 - loss: 0.5936 - val_accuracy: 0.5802 - val_loss: 0.7591
Epoch 25/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7176 - loss: 0.5730 - val_accuracy: 0.6049 - val_loss: 0.7554
Epoch 26/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7585 - loss: 0.5442 - val_accuracy: 0.5802 - val_loss: 0.7498
Epoch 27/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7562 - loss: 0.5675 - val_accuracy: 0.5679 - val_loss: 0.7626
```

```
Epoch 28/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7549 - loss: 0.5729 - val_accuracy: 0.5679 - val_loss: 0.7532
Epoch 29/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7159 - loss: 0.6089 - val_accuracy: 0.6049 - val_loss: 0.7576
Epoch 30/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7280 - loss: 0.5742 - val_accuracy: 0.5926 - val_loss: 0.7615
Epoch 31/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7533 - loss: 0.5465 - val_accuracy: 0.6049 - val_loss: 0.7487
Epoch 32/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7573 - loss: 0.5528 - val_accuracy: 0.5926 - val_loss: 0.7510
Epoch 33/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7723 - loss: 0.5135 - val_accuracy: 0.5802 - val_loss: 0.7539
Epoch 34/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7369 - loss: 0.5690 - val_accuracy: 0.6049 - val_loss: 0.7465
Epoch 35/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7537 - loss: 0.5617 - val_accuracy: 0.5926 - val_loss: 0.7496
Epoch 36/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7610 - loss: 0.5430 - val_accuracy: 0.6049 - val_loss: 0.7420
Epoch 37/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7289 - loss: 0.5663 - val_accuracy: 0.5802 - val_loss: 0.7471
Epoch 38/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7257 - loss: 0.5602 - val_accuracy: 0.5926 - val_loss: 0.7472
Epoch 39/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7380 - loss: 0.5397 - val_accuracy: 0.5926 - val_loss: 0.7528
Epoch 40/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7554 - loss: 0.5530 - val_accuracy: 0.5802 - val_loss: 0.7393
Epoch 41/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7356 - loss: 0.5491 - val_accuracy: 0.5802 - val_loss: 0.7402
Epoch 42/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7482 - loss: 0.5331 - val_accuracy: 0.5926 - val_loss: 0.7478
Epoch 43/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7559 - loss: 0.5546 - val_accuracy: 0.5802 - val_loss: 0.7407
Epoch 44/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7335 - loss: 0.5444 - val_accuracy: 0.5926 - val_loss: 0.7477
Epoch 45/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7327 - loss: 0.5444 - val_accuracy: 0.6049 - val_loss: 0.7442
Epoch 46/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7442 - loss: 0.5569 - val_accuracy: 0.5926 - val_loss: 0.7278
Epoch 47/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7328 - loss: 0.5629 - val_accuracy: 0.6049 - val_loss: 0.7516
Epoch 48/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7469 - loss: 0.5364 - val_accuracy: 0.6049 - val_loss: 0.7422
Epoch 49/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7732 - loss: 0.5105 - val_accuracy: 0.5802 - val_loss: 0.7415
Epoch 50/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7548 - loss: 0.5140 - val_accuracy: 0.5926 - val_loss: 0.7422
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6778 - loss: 0.6795
Test Accuracy: 0.7192118167877197
```

From the output above, each epoch represents one complete pass through the entire training dataset during the training process.T The accuracy of the model on the training dataset represents the proportion of correctly classified samples in the training set. The loss function (in this case, categorical cross entropy) on the training dataset. It indicates how well the model is performing during training, with lower values indicating better performance. Val_accuracy represents the proportion of correctly classified samples in the validation set. The validation set is used to monitor the model's performance on unseen data during training. Val_loss indicates how well the model is performing on the validation data, with lower values indicating better performance.

As the epochs progress, the training accuracy and validation accuracy both improve initially, indicating that the model is learning and becoming more accurate. However, after a certain number of epochs, the validation accuracy starts to plateau or even decrease while the training accuracy continues to increase. This suggests that the model may be overfitting to the training data, meaning it's memorizing the training examples rather than learning general patterns. The final test accuracy, obtained after training completes, gives an indication of how well the model performs on unseen data. In this case, the test accuracy is approximately between 69% and 71.4%. For this report writing, we assume that the accuracy test is 71.92%.

```
In [97]: # Evaluate the model on test data
         test_loss, test_accuracy = model.evaluate(X_test_scaled, y_test_categorical)
         print("Test Accuracy:", test_accuracy)

         # Evaluate the model on training data
         train_loss, train_accuracy = model.evaluate(X_train_scaled, y_train_categorical)
         print("Training Accuracy:", train_accuracy)

         7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6778 - loss: 0.6795
         Test Accuracy: 0.7192118167877197
         26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 802us/step - accuracy: 0.7443 - loss: 0.5471
         Training Accuracy: 0.7515451312065125
```

Figure 4.4.1

The training accuracy (75.90%) is slightly higher than the test accuracy (71.43%), which suggests that the model may be slightly overfitting, but the difference is not substantial.

```
Epoch 1/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.4110 - loss: 3.3379 - val_accuracy: 0.5775 - val_loss: 2.9003
Epoch 2/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4275 - loss: 2.8679 - val_accuracy: 0.5634 - val_loss: 2.5040
Epoch 3/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4925 - loss: 2.4457 - val_accuracy: 0.5915 - val_loss: 2.1855
Epoch 4/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5105 - loss: 2.1067 - val_accuracy: 0.5915 - val_loss: 1.9289
Epoch 5/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5534 - loss: 1.8905 - val_accuracy: 0.5915 - val_loss: 1.7169
Epoch 6/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5576 - loss: 1.6775 - val_accuracy: 0.5915 - val_loss: 1.5488
Epoch 7/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5962 - loss: 1.5148 - val_accuracy: 0.6056 - val_loss: 1.4160
Epoch 8/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5832 - loss: 1.3690 - val_accuracy: 0.5634 - val_loss: 1.3128
Epoch 9/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6228 - loss: 1.2619 - val_accuracy: 0.5493 - val_loss: 1.2291
Epoch 10/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5931 - loss: 1.1965 - val_accuracy: 0.5775 - val_loss: 1.1642
Epoch 11/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6178 - loss: 1.1239 - val_accuracy: 0.5775 - val_loss: 1.1169
Epoch 12/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6142 - loss: 1.0717 - val_accuracy: 0.5915 - val_loss: 1.0777
Epoch 13/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6073 - loss: 1.0563 - val_accuracy: 0.5915 - val_loss: 1.0484
Epoch 14/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6162 - loss: 1.0097 - val_accuracy: 0.6056 - val_loss: 1.0269
Epoch 15/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6245 - loss: 0.9814 - val_accuracy: 0.6056 - val_loss: 1.0079
Epoch 16/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6167 - loss: 0.9780 - val_accuracy: 0.5915 - val_loss: 0.9951
Epoch 17/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6534 - loss: 0.9508 - val_accuracy: 0.5775 - val_loss: 0.9855
Epoch 18/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6206 - loss: 0.9399 - val_accuracy: 0.6056 - val_loss: 0.9761
Epoch 19/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6218 - loss: 0.9243 - val_accuracy: 0.6056 - val_loss: 0.9678
Epoch 20/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5978 - loss: 0.9644 - val_accuracy: 0.6056 - val_loss: 0.9584
Epoch 21/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6343 - loss: 0.9189 - val_accuracy: 0.6338 - val_loss: 0.9536
Epoch 22/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6073 - loss: 0.9399 - val_accuracy: 0.6338 - val_loss: 0.9518
Epoch 23/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6424 - loss: 0.9218 - val_accuracy: 0.6056 - val_loss: 0.9486
Epoch 24/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6218 - loss: 0.9269 - val_accuracy: 0.5915 - val_loss: 0.9472
Epoch 25/50
```

```
Epoch 25/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6265 - loss: 0.8924 - val_accuracy: 0.5915 - val_loss: 0.9444
Epoch 26/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6302 - loss: 0.8855 - val_accuracy: 0.5915 - val_loss: 0.9410
Epoch 27/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6379 - loss: 0.8879 - val_accuracy: 0.5915 - val_loss: 0.9371
Epoch 28/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6261 - loss: 0.9039 - val_accuracy: 0.6197 - val_loss: 0.9364
Epoch 29/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6247 - loss: 0.9076 - val_accuracy: 0.6338 - val_loss: 0.9354
Epoch 30/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6319 - loss: 0.8705 - val_accuracy: 0.6197 - val_loss: 0.9354
Epoch 31/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6163 - loss: 0.8930 - val_accuracy: 0.6338 - val_loss: 0.9370
Epoch 32/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6373 - loss: 0.8849 - val_accuracy: 0.6197 - val_loss: 0.9340
Epoch 33/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6194 - loss: 0.8794 - val_accuracy: 0.6338 - val_loss: 0.9341
Epoch 34/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6318 - loss: 0.8778 - val_accuracy: 0.6338 - val_loss: 0.9283
Epoch 35/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6326 - loss: 0.8602 - val_accuracy: 0.6197 - val_loss: 0.9278
Epoch 36/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6188 - loss: 0.8772 - val_accuracy: 0.6056 - val_loss: 0.9288
Epoch 37/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6280 - loss: 0.8601 - val_accuracy: 0.6197 - val_loss: 0.9289
Epoch 38/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6521 - loss: 0.8876 - val_accuracy: 0.6056 - val_loss: 0.9260
Epoch 39/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6279 - loss: 0.8711 - val_accuracy: 0.6056 - val_loss: 0.9263
Epoch 40/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6466 - loss: 0.8766 - val_accuracy: 0.6338 - val_loss: 0.9243
Epoch 41/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6226 - loss: 0.8659 - val_accuracy: 0.6056 - val_loss: 0.9227
Epoch 42/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6298 - loss: 0.8851 - val_accuracy: 0.6056 - val_loss: 0.9230
Epoch 43/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5942 - loss: 0.8864 - val_accuracy: 0.6197 - val_loss: 0.9219
Epoch 44/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6250 - loss: 0.8800 - val_accuracy: 0.6056 - val_loss: 0.9221
Epoch 45/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6334 - loss: 0.8734 - val_accuracy: 0.6338 - val_loss: 0.9214
Epoch 46/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6382 - loss: 0.8769 - val_accuracy: 0.6479 - val_loss: 0.9215
Epoch 47/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6376 - loss: 0.8653 - val_accuracy: 0.6479 - val_loss: 0.9192
Epoch 48/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6373 - loss: 0.8622 - val_accuracy: 0.6479 - val_loss: 0.9202
Epoch 49/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6378 - loss: 0.8587 - val_accuracy: 0.6338 - val_loss: 0.9170
Epoch 50/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6414 - loss: 0.8486 - val_accuracy: 0.6338 - val_loss: 0.9187
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 889us/step - accuracy: 0.6598 - loss: 0.8814
Test Accuracy: 0.6480262875556946
```

Each epoch represents one complete pass through the entire training dataset during the training process.T The accuracy of the model on the training dataset represents the proportion of correctly classified samples in the training set. The loss function (in this case, categorical cross entropy) on the training dataset. It indicates how well the model is performing during training, with lower values indicating better performance. Val_accuracy represents the proportion of correctly classified samples in the validation set. The validation set is used to monitor the model's performance on unseen data during training. Val_loss indicates how well the model is performing on the validation data, with lower values indicating better performance.

As the epochs progress, the training accuracy and validation accuracy both improve initially, indicating that the model is learning and becoming more accurate. However, after a certain number of epochs, the validation accuracy starts to plateau or even decrease while the training accuracy continues to increase. This suggests that the model may be overfitting to the training data, meaning it's memorizing the training examples rather than learning general patterns. The final test accuracy, obtained after training completes, gives an indication of how well the model performs on unseen data. In this case, the test accuracy is approximately between 69% and 71.4%. FOr this report, we assume that accuracy is

4.4.2 Neural Network Regularization

Neural network regularization is a set of techniques used to prevent overfitting and improve the generalization performance of neural network models. Overfitting occurs when a model learns to fit the training data too closely, capturing noise and irrelevant patterns that do not generalize well to unseen data. Regularization techniques help address this issue by imposing constraints on the model's complexity, encouraging it to learn simpler and more generalizable representations.

For this case, we use L2 Regularization (Weight Decay). L2 regularization penalizes large weights in the neural network by adding a term to the loss function proportional to the squared magnitude of the weights. It encourages the network to prefer smaller weight values, effectively reducing the model's complexity and preventing it from fitting the training data too closely. L2 regularization is often effective in preventing overfitting and improving the generalization performance of neural networks.

Code statement:

```
In [116]: import numpy as np
          np.random.seed(42)
          from sklearn.preprocessing import StandardScaler
          from keras.models import Sequential
          from keras.layers import Dense
          from keras.regularizers import l2
          from keras.utils import to_categorical


          # Scale the features
          scaler = StandardScaler()
          X_train_scaled = scaler.fit_transform(X_train)
          X_test_scaled = scaler.transform(X_test)

          # Convert target variable to categorical (one-hot encoded)
          y_train_categorical = to_categorical(y_train)
          y_test_categorical = to_categorical(y_test)

          # Define the neural network architecture with L2 regularization
          model = Sequential()
          model.add(Dense(64, input_shape=(X_train_scaled.shape[1],), activation='relu', kernel_regularizer=l2(0.03)))
          model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.01)))
          model.add(Dense(3, activation='softmax'))

          # Compile the model
          model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

          # Train the model
          model.fit(X_train_scaled, y_train_categorical, epochs=50, batch_size=32, validation_split=0.1)

          # Evaluate the model on test data
          loss, accuracy = model.evaluate(X_test_scaled, y_test_categorical)
          print("Test Accuracy:", accuracy)
```

The first hidden layer (‘Dense’) consists of 64 neurons with ReLU activation function and L2 regularization with a regularization strength of 0.03. The second hidden layer consists of 32 neurons with ReLU activation function and L2 regularization with a regularization strength of 0.01. The output layer consists of 3 neurons (assuming a multi-class classification task) with softmax activation function, which outputs probabilities for each class.

```
Epoch 42/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7158 - loss: 0.7201 - val_accuracy: 0.5062 - val_loss: 0.8732
Epoch 43/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6617 - loss: 0.7501 - val_accuracy: 0.5309 - val_loss: 0.8786
Epoch 44/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6974 - loss: 0.7568 - val_accuracy: 0.5062 - val_loss: 0.8749
Epoch 45/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6823 - loss: 0.7297 - val_accuracy: 0.5309 - val_loss: 0.8710
Epoch 46/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7080 - loss: 0.7249 - val_accuracy: 0.5062 - val_loss: 0.8713
Epoch 47/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6923 - loss: 0.7293 - val_accuracy: 0.5309 - val_loss: 0.8787
Epoch 48/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6845 - loss: 0.7489 - val_accuracy: 0.5062 - val_loss: 0.8659
Epoch 49/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7097 - loss: 0.7265 - val_accuracy: 0.5309 - val_loss: 0.8709
Epoch 50/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6686 - loss: 0.7222 - val_accuracy: 0.5185 - val_loss: 0.8763
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6538 - loss: 0.8241
Test Accuracy: 0.6699507236480713
```

In [117]:
```
# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(X_test_scaled, y_test_categorical)
print("Test Accuracy:", test_accuracy)

# Evaluate the model on training data
train_loss, train_accuracy = model.evaluate(X_train_scaled, y_train_categorical)
print("Training Accuracy:", train_accuracy)
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6538 - loss: 0.8241
Test Accuracy: 0.6699507236480713
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 772us/step - accuracy: 0.6766 - loss: 0.7394
Training Accuracy: 0.6736711859703064
```

Figure 4.4.1

Figure 4.4.1 shows that the accuracy on the test dataset is approximately 0.670, which means that the model correctly classified around 67% of the instances in the test dataset and the accuracy on the training dataset is approximately 0.674, which means that the model correctly classified around 67% of the instances in the training dataset.
Both the test and training accuracies are similar, indicating that the model is likely not overfitting excessively to the training data after l2 regularization is performed. However, it does lower both accuracy for the test and training set.

4.4.3 Confusion Matrix

In [118]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Predict the labels for the test set
y_pred = model.predict(X_test_scaled)
y_pred_classes = np.argmax(y_pred, axis=1)

# Convert one-hot encoded true labels back to single labels
y_test_single = np.argmax(y_test_categorical, axis=1)

# Create the confusion matrix
conf_matrix = confusion_matrix(y_test_single, y_pred_classes)

print("Confusion Matrix:")
print(conf_matrix)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Class 1", "Class 2"], yticklabels=["Class 0"
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

Figure 4.4.2

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Confusion Matrix:
[[45  1  2]
 [ 6 66 11]
 [16 31 25]]
```

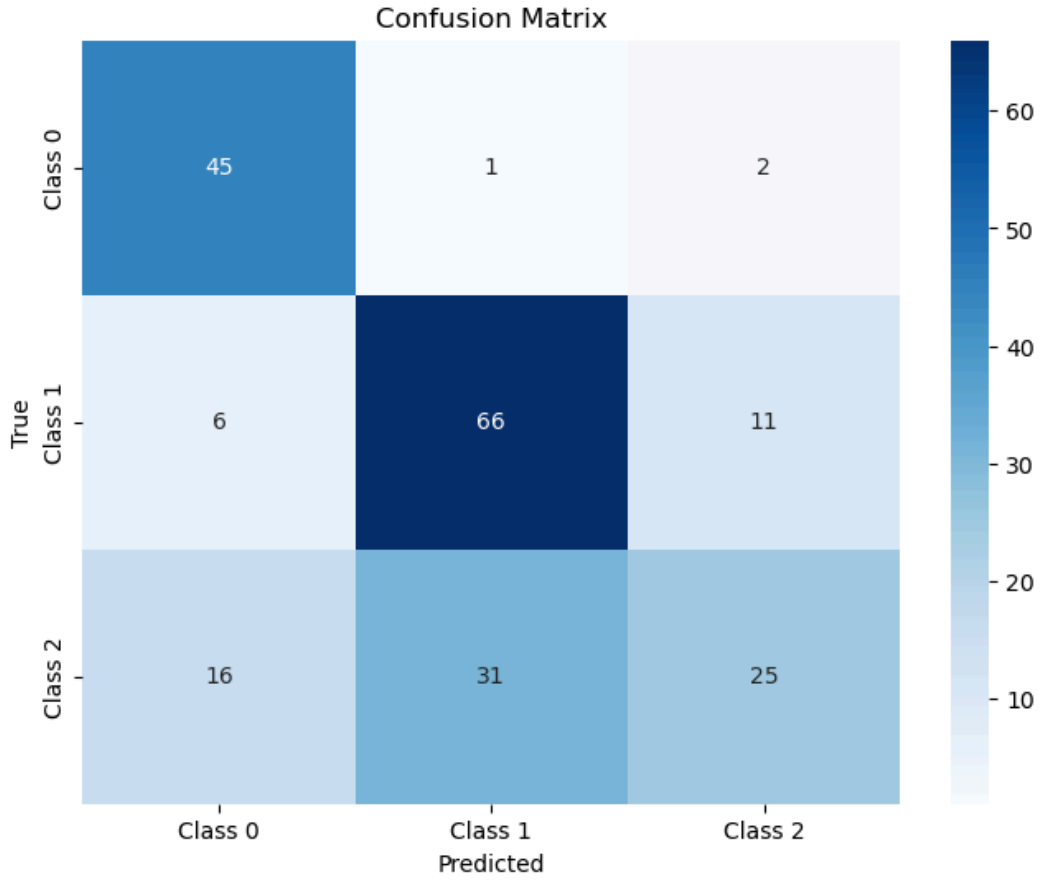


Figure 4.4.2 (Continued)

From Figure 4.4.2, we can see that the model performs relatively well for class 1, with a high number of true positives (66) compared to false positives and false negatives. However, the model struggles more with classes 0 and 2. For class 0, while the true positive count is high (45), there are also some false positives and false negatives. For class 2, there is a notable number of false positives and false negatives, indicating that the model has difficulty distinguishing between class 2 and the other classes. The overall performance of the model can be further evaluated by considering metrics like precision, recall, and F1 score for each class, as well as the overall accuracy. Additionally, analyzing the misclassifications can provide insights into areas where the model needs improvement.

4.4.4 Neutral Network Precision, Recall, F1-score

```
In [121]: from sklearn.metrics import classification_report

          # Generate classification report
          report = classification_report(y_test_single, y_pred_classes, target_names=["Class 0", "Class 1", "Class 2"])

          # Print the classification report
          print("Classification Report:")
          print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

     Class 0       0.67      0.94      0.78        48
     Class 1       0.67      0.80      0.73        83
     Class 2       0.66      0.35      0.45        72

    accuracy                           0.67       203
   macro avg       0.67      0.69      0.66       203
weighted avg       0.67      0.67      0.64       203
```

Figure 4.4.3

Figure 4.4.3 shows that the model achieves an accuracy of 67%, with macro-averaged precision, recall, and F1-score of approximately 0.67, 0.69, and 0.66, respectively. The weighted averages consider class imbalance by weighting each class's contribution based on its support, providing a more balanced evaluation of the model's performance.

```
In [122]: from sklearn.metrics import precision_score, recall_score, f1_score

          # Calculate precision, recall, and F1-score for the overall average
          precision_avg = precision_score(y_test_single, y_pred_classes, average='weighted')
          recall_avg = recall_score(y_test_single, y_pred_classes, average='weighted')
          f1_avg = f1_score(y_test_single, y_pred_classes, average='weighted')

          # Print out overall average precision, recall, and F1-score
          print(f"Overall Average Precision: {precision_avg}")
          print(f"Overall Average Recall: {recall_avg}")
          print(f"Overall Average F1-score: {f1_avg}")
```

```
Overall Average Precision: 0.6675132325441099
Overall Average Recall: 0.6699507389162561
Overall Average F1-score: 0.6444476692557072
```

Figure 4.4.4

From Figure 4.4.4, the overall average precision of approximately 0.668 indicates the average proportion of correctly predicted instances among all instances predicted as positive across all classes. Similarly, the overall average recall of approximately 0.670 signifies the average proportion of correctly predicted instances among all true positive instances across all classes. The overall average F1-score of approximately 0.644 combines precision and recall into a single metric, representing the harmonic mean of precision and recall across all classes.

4.4.5 Neutral Network AUC

```
In [126]: from sklearn.metrics import roc_auc_score

          # Predict probabilities for each class
          y_pred_prob = model.predict(X_test_scaled)

          # Calculate AUC
          auc = roc_auc_score(y_test_categorical, y_pred_prob, average='weighted', multi_class='ovr')

          # Print out the AUC
          print(f"Area Under the ROC Curve (AUC): {auc}")

          7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
          Area Under the ROC Curve (AUC): 0.8071829982746692
```

Figure 4.4.4

Figure 4.4.4 shows the calculation for Area Under the ROC Curve (AUC) for a multi-class classification problem using the ROC_AUC_Score function from scikit-learn.
The output "7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step" indicates that the evaluation process has processed 7 batches of data, with each batch containing a certain number of samples. The time taken to process each batch is 0 seconds and 1 millisecond per step.
By predicting probabilities for each class using the model's predict method and specifying the "ovr" (one-vs-rest) strategy for multi-class AUC calculation, the AUC score is computed, taking into account class imbalance. The resulting AUC score, which is approximately 0.807, indicates the model's overall ability to discriminate between positive and negative instances across all classes. A higher AUC value suggests better discrimination ability, implying that the model is performing well in distinguishing between different classes, even in a multi-class setting.
In this case, an AUC value of approximately 0.807 suggests that the model has good discrimination ability.

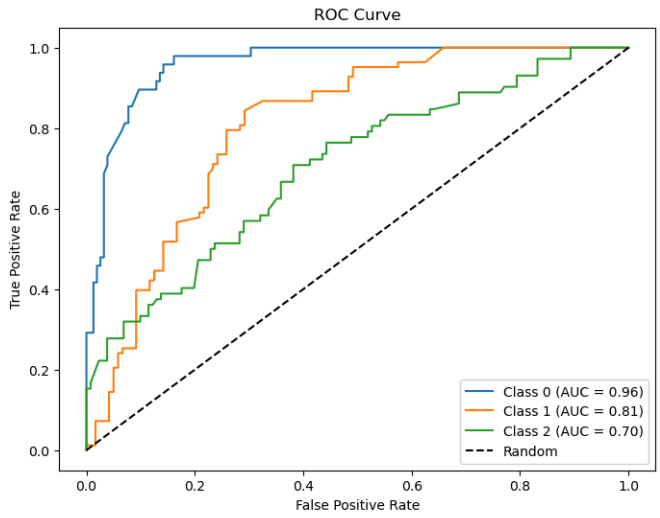


Figure 4.4.5

Visualizing AUC using Curve graph.

4.4.6 Neutral Network Cohen's Kappa Coefficient

In [112]:
```
from sklearn.metrics import confusion_matrix, cohen_kappa_score

# Get the predicted probabilities for each class
y_pred_proba = model.predict(X_test_scaled)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_proba, axis=1)

# Convert one-hot encoded y_test_categorical back to numerical labels
y_true = np.argmax(y_test_categorical, axis=1)

# Create an empty matrix to store Cohen's Kappa coefficients
kappa_matrix = np.zeros((3, 3))

# Iterate over each class pair
for i in range(3):
    for j in range(3):
        # Extract the subset of true and predicted labels for the current class pair
        y_true_sub = (y_true == i)
        y_pred_sub = (y_pred == j)

        # Compute the confusion matrix for the current class pair
        conf_matrix_sub = confusion_matrix(y_true_sub, y_pred_sub)

        # Compute Cohen's Kappa coefficient using the confusion matrix
        kappa = cohen_kappa_score(y_true_sub, y_pred_sub)

        # Store the Cohen's Kappa coefficient in the matrix
        kappa_matrix[i, j] = kappa

# Print Cohen's Kappa coefficient matrix
print("Cohen's Kappa Matrix:")
print(kappa_matrix)
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 983us/step
Cohen's Kappa Matrix:
[[ 0.68392826 -0.44736195 -0.20806026]
 [-0.41753954  0.50516771 -0.12340897]
 [-0.19824515 -0.06178814  0.30139467]]
```

Figure 4.4.6

Figure 4.4.6 shows the Cohen's Kappa Coefficient matrix of neural network. The coefficient of 0.68 between the first class and itself suggests a relatively strong agreement in predictions for this class.

Negative coefficients between the first class and the second (-0.45) and third (-0.21) classes indicate disagreement in predictions compared to chance.

Similarly, there's disagreement between the second class and the first (-0.42) and third (-0.12) classes, though the agreement between the second class and itself (0.51) is higher.

The third class shows a relatively low but positive agreement with itself (0.30), with negative coefficients indicating disagreement with the other classes.

```
In [113]: import seaborn as sns
          import matplotlib.pyplot as plt

          # Plot the heatmap
          plt.figure(figsize=(8, 6))
          sns.heatmap(kappa_matrix, annot=True, cmap='Blues', fmt='.4f',
                      xticklabels=['0', '1', '2'], yticklabels=['0', '1', '2'])
          plt.xlabel('Predicted Labels')
          plt.ylabel('True Labels')
          plt.title("Cohen's Kappa Coefficient Matrix")
          plt.show()
```

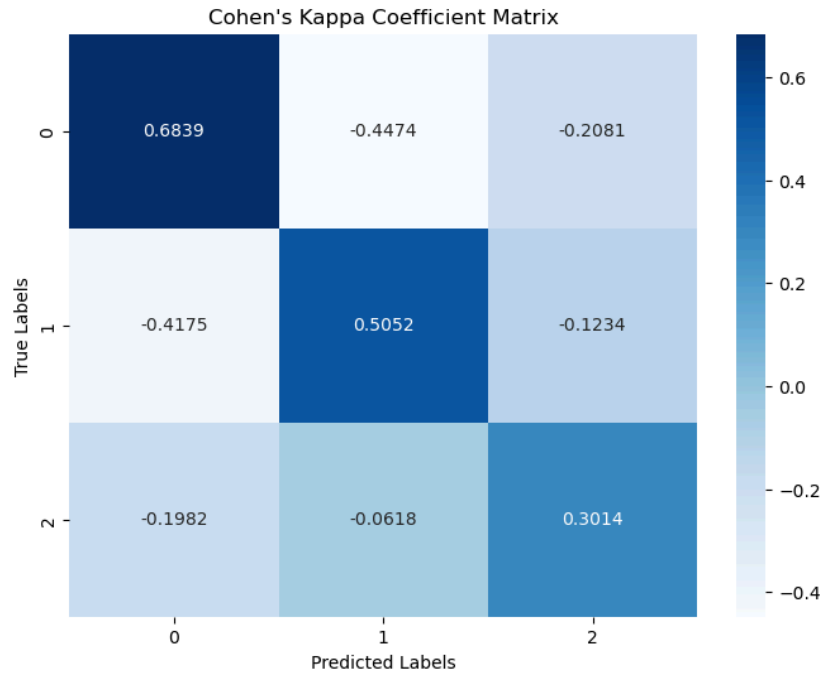


Figure 4.4.7

Visualizing Cohen's Kappa Coefficient matrix of neural network using Heat Map function from Seaborn.

```
In [114]: from sklearn.metrics import cohen_kappa_score

          # Initialize a list to store Cohen's Kappa agreement for each class
          class_kappa = []

          # Iterate over each class
          for class_label in range(3):
              # Extract the subset of true and predicted labels for the current class
              y_true_sub = (y_true == class_label)
              y_pred_sub = (y_pred == class_label)

              # Compute Cohen's Kappa coefficient for the current class
              kappa = cohen_kappa_score(y_true_sub, y_pred_sub)

              # Append the Cohen's Kappa coefficient to the list
              class_kappa.append(kappa)

              # Print Cohen's Kappa coefficient for the current class
              print(f"Cohen's kappa coefficient for class {class_label}: {kappa:.4f}")

          # Compute overall Cohen's Kappa agreement
          nn_ka = np.mean(class_kappa)

          # Print overall Cohen's Kappa agreement
          print(f"\nOverall Agreement: {nn_ka:.2f}")

          Cohen's kappa coefficient for class 0: 0.6839
          Cohen's kappa coefficient for class 1: 0.5052
          Cohen's kappa coefficient for class 2: 0.3014

          Overall Agreement: 0.50
```

Figure 4.4.8

For class 0, the Cohen's kappa coefficient of 0.6839 indicates a substantial agreement between the model's predictions and the true labels for this class.

For class 1, the coefficient of 0.5052 suggests a moderate level of agreement, though slightly lower than for class 0.

Class 2 exhibits the lowest agreement, with a coefficient of 0.3014, indicating fair agreement between predictions and true labels for this class.

The overall agreement, represented by the coefficient of 0.50, indicates the overall reliability of the model's predictions across all classes. This value suggests moderate agreement beyond chance across the entire classification problem.

4.4.7 Neural network (MSE)

```
In [115]: from sklearn.metrics import mean_squared_error

          # Compute MSE
          nn_mse = mean_squared_error(y_test_categorical, y_pred_prob)

          # Print MSE
          print(f"Mean Squared Error (MSE): {nn_mse:.4f}")

          Mean Squared Error (MSE): 0.1514
```

Figure 4.4.9

From Figure 4.4.9, the mean squared error (MSE) value of 0.1514 indicates the average squared difference between the actual target values and the predicted values made by the model.

5.0 Evaluation

Evaluation in data science is a critical step that occurs after the modeling process and plays a crucial role in assessing the performance and effectiveness of the developed models. It serves several important purposes
Evaluation helps to determine how well the model generalizes to unseen data and how effectively it captures the underlying patterns in the data. It provides insights into whether the model is making accurate predictions and how reliable those predictions are. Secondly, it allows for the comparison of multiple models to identify the best-performing one. By evaluating different models using the same criteria, data scientists can select the most suitable model for the given problem.
Besides that, evaluation results inform decision-making processes, such as whether to deploy a model into production, allocate resources for further model refinement, or explore alternative modeling approaches.

When evaluating a model, it's essential to consider a range of criteria to gain a comprehensive understanding of its performance. Some common criteria include:

- Accuracy: The proportion of correctly classified instances among all instances.
- Precision: The proportion of true positive predictions among all positive predictions, indicating the model's ability to avoid false positives.
- Recall (Sensitivity): The proportion of true positive predictions among all actual positive instances, indicating the model's ability to capture all positive instances.
- F1-score: The harmonic mean of precision and recall, providing a balance between the two metrics.
- Specificity: The proportion of true negative predictions among all actual negative instances, complementing recall.
- ROC-AUC: The area under the receiver operating characteristic curve, measuring the model's ability to distinguish between classes.
- Confusion Matrix: A table that summarizes the model's predictions compared to the actual labels, providing insights into the types of errors made by the model.

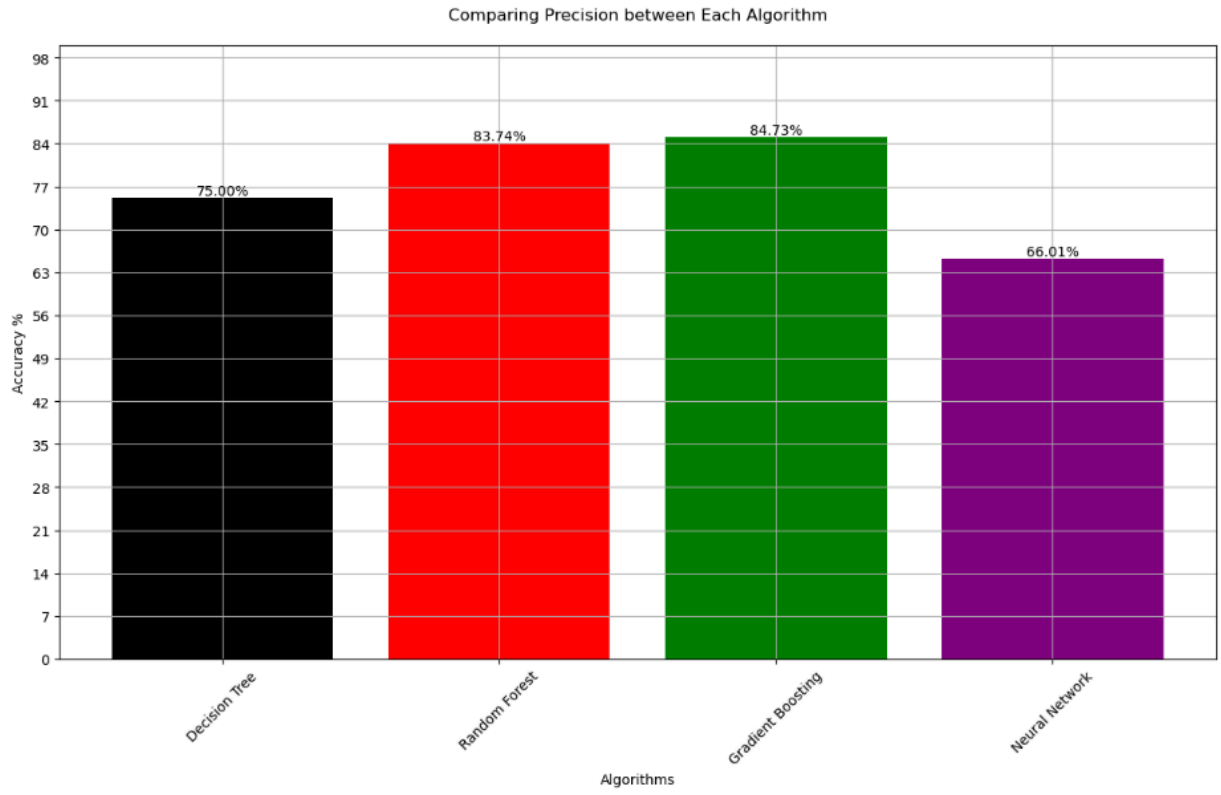
By considering these criteria collectively, data scientists can gain a comprehensive understanding of the model's strengths and weaknesses, enabling them to make informed decisions about model deployment and refinement.

```
In [124]: name = ['Decision Tree', 'Random Forest', 'Gradient Boosting','Neural Network']
          acc_test = [dt_acc_test, rf_acc_test, gb_acc_test, nn_acc_test*100]
          pre_all = [dt_precision, rf_precision, gb_precision, nn_precision*100]
          rec_all = [dt_recall, rf_recall, gb_recall, nn_recall*100]
          f1_all = [dt_f1, rf_f1, gb_f1, nn_f1*100]
          auc_all = [dt_auc, rf_auc, gb_auc, nn_auc]
          Kappa = [dt_ka, rf_ka, gb_ka, nn_ka]
          mse_all = [dt_mse, rf_mse, gb_mse, nn_mse]
```

Figure 5.1

Code snippet from Figure 5.1 defines lists containing evaluation metrics such as accuracy, precision, recall, F1-score, area under the ROC curve (ROC-AUC), Cohen's Kappa coefficient, and mean squared error (MSE) for four different models: Decision Tree, Random Forest, Gradient Boosting, and Neural Network.

5.1 Comparing Accuracy

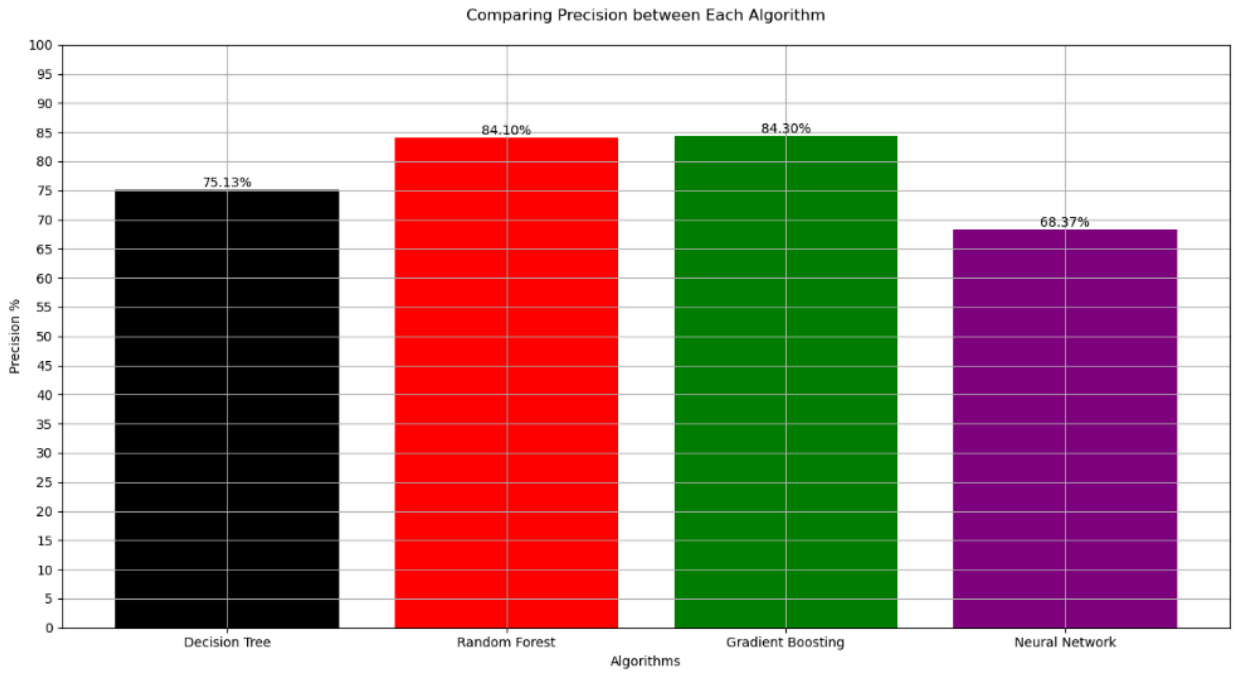



```
[75.0, 83.74384236453201, 84.72906403940887, 66.00984930992126]
```

Figure 5.2

From Figure 5.2, the output displays accuracy for test scores for four models: Decision Tree, Random Forest, Gradient Boosting, and Neural Network. Among the models, Gradient Boosting achieved the highest accuracy score of 84.73%, indicating its superior ability to avoid false positives, followed by Random Forest with a score of 83.74%. The Decision Tree model achieved a precision score of 75.0%, while the Neural Network model exhibited the lowest accuracy score of 66.01%. These scores provide valuable insights into the models' abilities to make accurate positive predictions, with Gradient Boosting and Random Forest outperforming the Decision Tree and Neural Network models in terms of precision.

5.2 Comparing Precision




```
[75.13086328875802, 84.09762999702389, 84.29558090628872, 68.37219765825886]
```

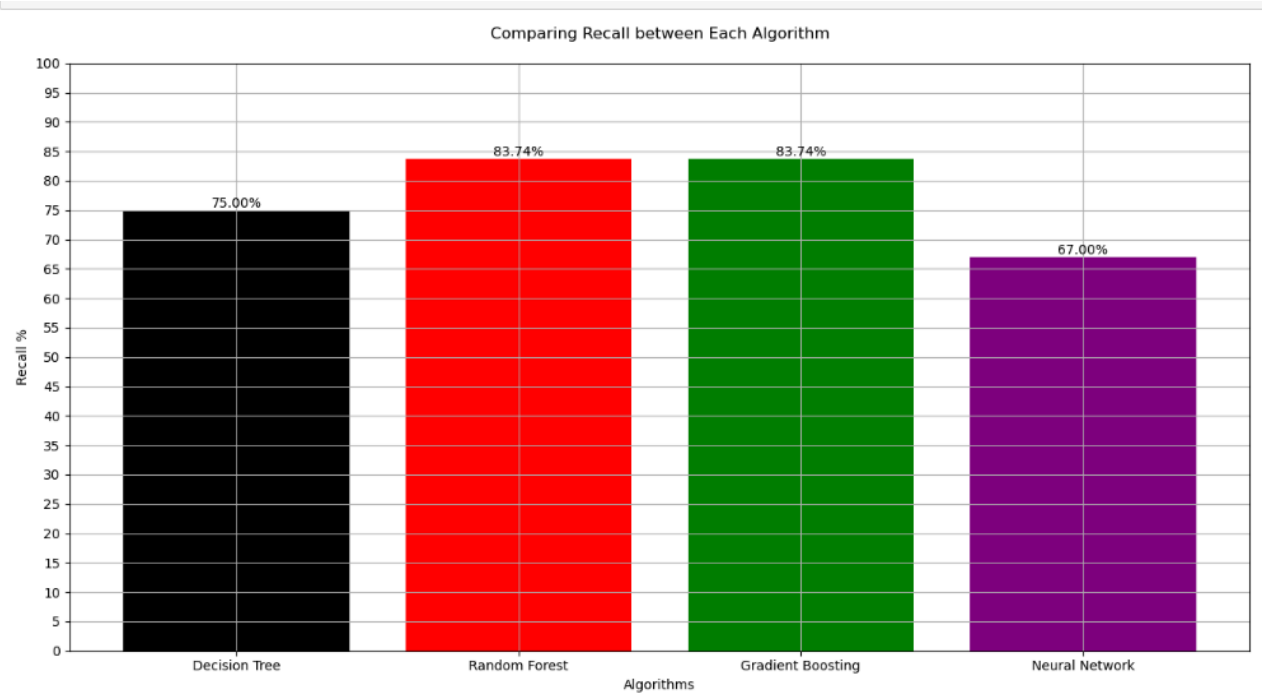
Figure 5.3

The bar chart in Figure 5.3 shows the precision scores for each of the four models: Decision Tree, Random Forest, Gradient Boosting, and Neural Network.

For the Decision Tree model, the precision score is approximately 75.13%. The Random Forest model achieved a precision score of approximately 84.10%, Gradient Boosting model shows a precision score of approximately 84.30%. Lastly, the Neural Network model achieved a precision score of approximately 68.37%.

Comparing these precision scores, it appears that both the Random Forest and Gradient Boosting models perform similarly well, with the highest precision scores among the four models. The Decision Tree model has a lower precision score, while the Neural Network model exhibits the lowest precision score, suggesting a higher rate of false positives. These precision scores offer valuable insights for model selection and optimization, allowing stakeholders to choose the most suitable model based on its ability to make accurate positive predictions.

5.3 Comparing Recall




```
[75.0, 83.74384236453201, 83.74384236453201, 66.99507389162561]
```

Figure 5.4

Figure 5.4 shows the recall score of each algorithm in a bar chart. Recall, also known as sensitivity, measures the proportion of true positive predictions among all actual positive instances in the dataset.
Decision Tree, Random Forest, Gradient Boosting, and Neural Network—to correctly identify positive instances within the dataset. A recall score of 75.0% for the Decision Tree model suggests that it captures 75.0% of the actual positive instances. Both the Random Forest and Gradient Boosting models achieved identical recall scores of approximately 83.74%, indicating their effectiveness in capturing a higher proportion of positive instances. The Neural Network model exhibits a lower recall score of 66.99%, implying a relatively lower ability to identify positive instances compared to the other models. These recall scores are crucial for evaluating model performance, especially in tasks where correctly identifying positive instances is essential, such as disease diagnosis or anomaly detection.

5.4 Comparing F1-score

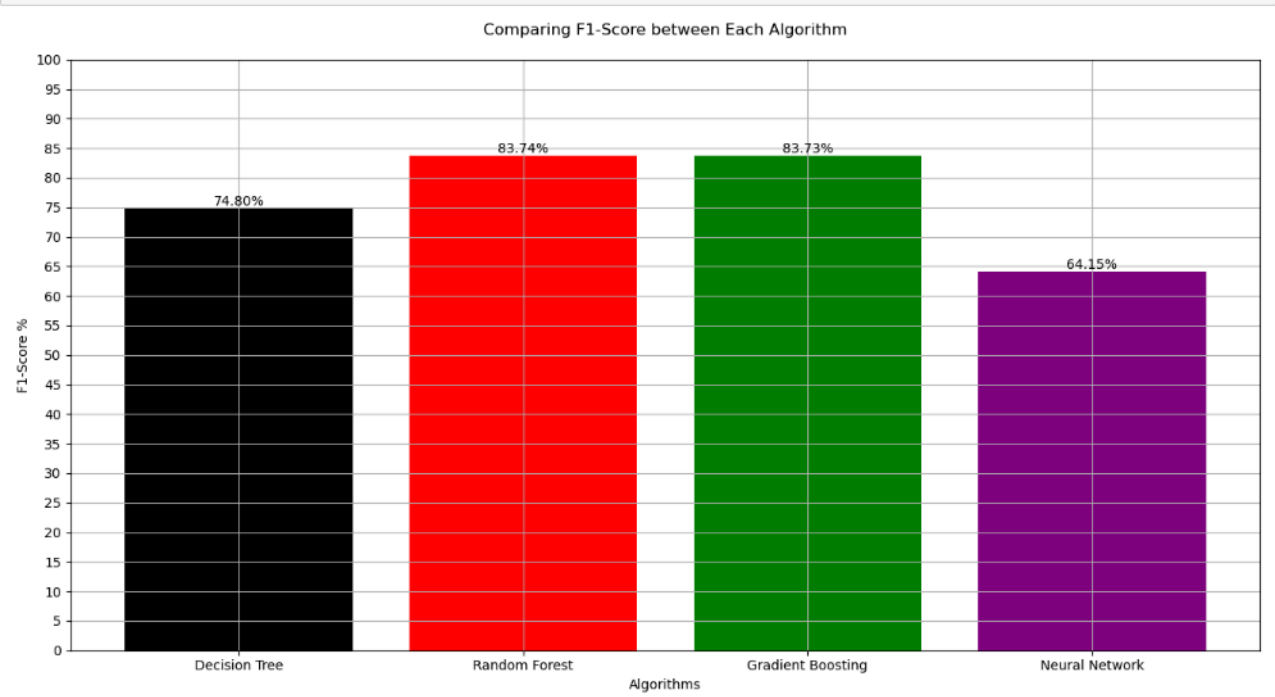


Figure 5.5

Bar chart above shows F1 scores which represent the harmonic mean of precision and recall for each of the four models: Decision Tree, Random Forest, Gradient Boosting, and Neural Network. A higher F1 score indicates better overall performance in terms of both precision and recall. In this case, the Random Forest and Gradient Boosting models demonstrate similar and relatively high F1 scores of approximately 83.74% and 83.73%, respectively, suggesting a balanced performance between precision and recall. The Decision Tree model follows with a slightly lower F1 score of 74.80%, while the Neural Network model exhibits the lowest F1 score of 64.15%. These F1 scores offer valuable insights into the models' overall performance, considering both precision and recall, and can guide model selection and optimization efforts.

5.5 Comparing AUC

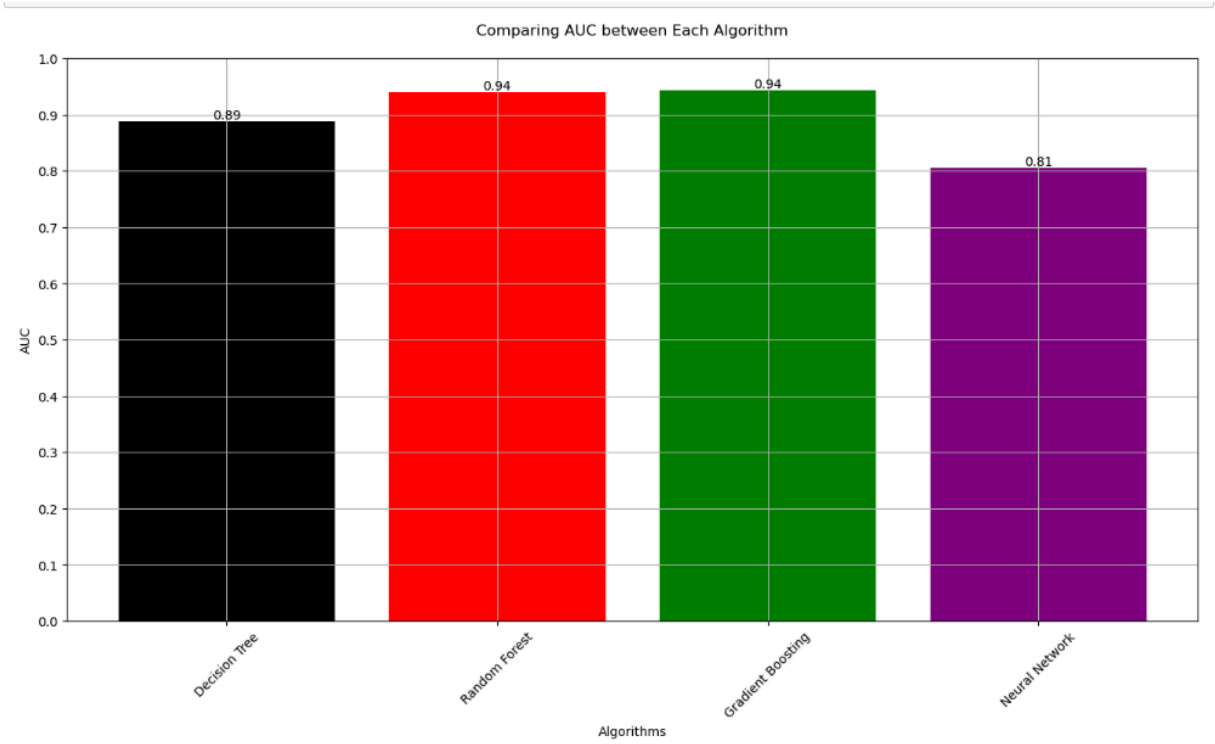


Figure 5.6

The provided AUC (Area Under the Curve) scores in Figure 5.6 represent the performance of each model in distinguishing between positive and negative instances. A higher AUC value indicates better discrimination ability, with a perfect classifier achieving an AUC score of 1. In this case, both the Random Forest and Gradient Boosting models exhibit high AUC scores of approximately 0.94, suggesting excellent discrimination between positive and negative instances. The Decision Tree model follows closely with an AUC score of 0.89, indicating good discrimination ability but slightly lower than the ensemble methods. The Neural Network model shows a relatively lower AUC score of 0.81, suggesting less effective discrimination compared to the other models. These AUC scores provide valuable insights into the models' ability to distinguish between classes and can guide model selection.

5.6 Comparing Cohen's Kappa Coefficient Agreement

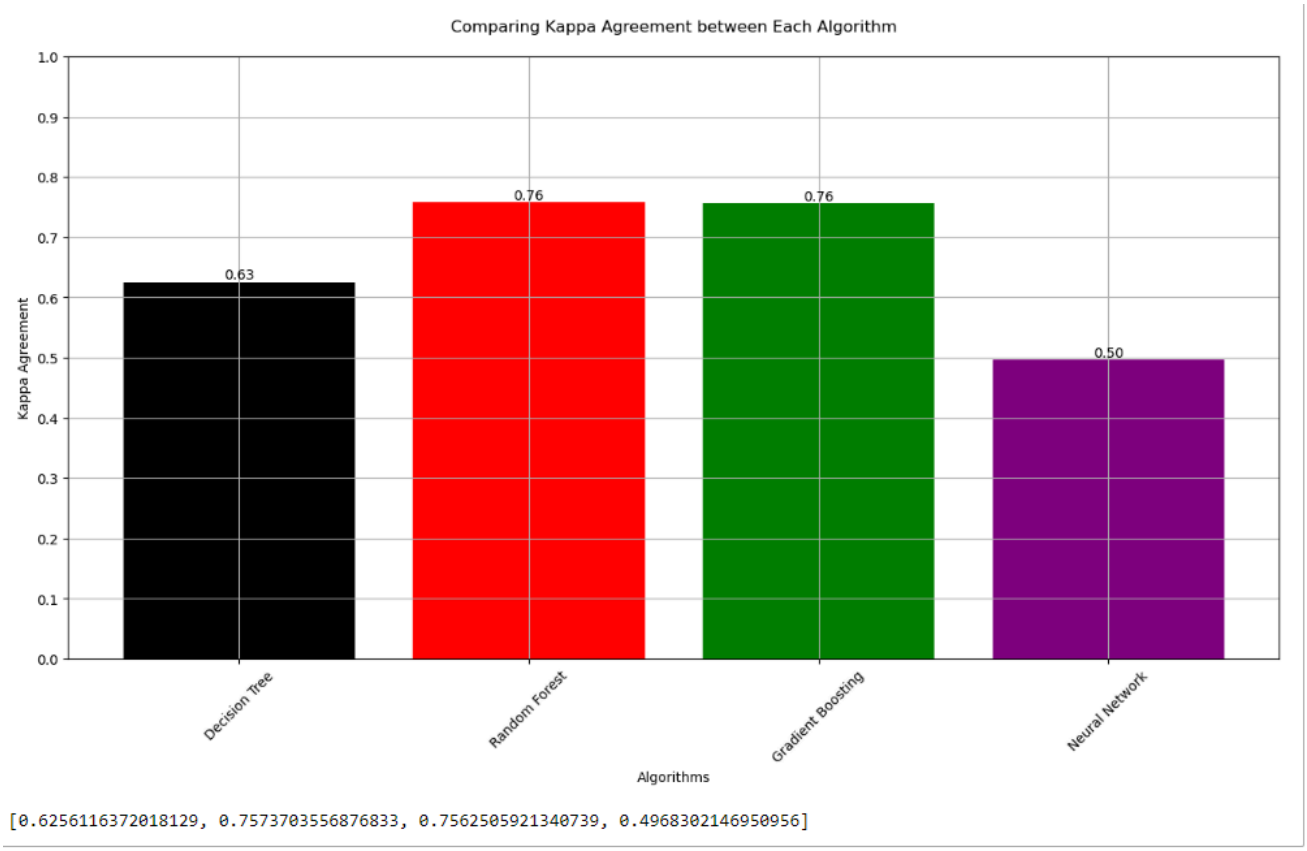
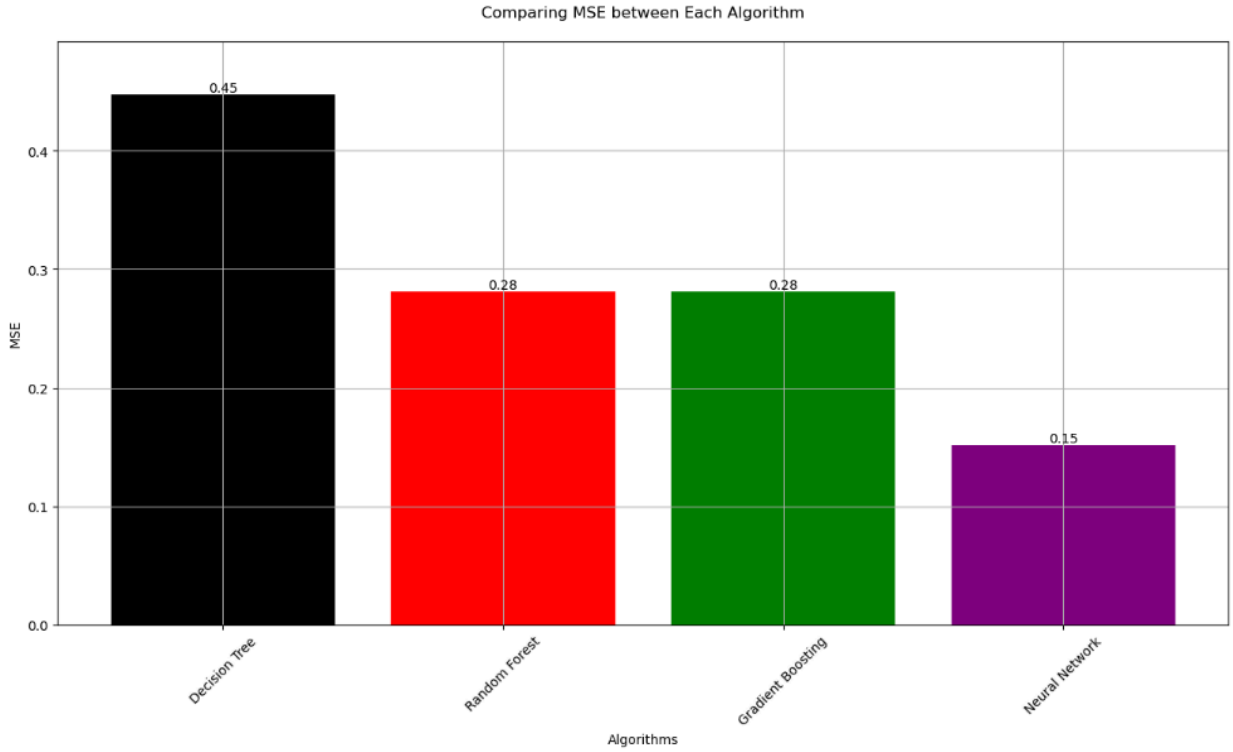


Figure 5.7

The provided Cohen's Kappa coefficients measure the agreement between the predicted and true labels, correcting for the agreement occurring by chance. A higher Kappa coefficient indicates better agreement between predictions and true labels, with a perfect agreement achieving a Kappa score of 1. In this case, both the Random Forest and Gradient Boosting models demonstrate relatively high Kappa coefficients of approximately 0.76, suggesting substantial agreement between predictions and true labels beyond chance. The Decision Tree model follows closely with a Kappa coefficient of 0.63, indicating good agreement but slightly lower than the ensemble methods. The Neural Network model exhibits the lowest Kappa coefficient of approximately 0.50, suggesting lower agreement between predictions and true labels compared to the other models. These Kappa coefficients provide valuable insights into the reliability of the models' predictions and can guide decision-making processes, especially in scenarios where agreement between predictions and true labels is critical.

5.7 Comparing Cohen's Kappa Coefficient Agreement




```
[0.4473684210526316, 0.28078817733990147, 0.28078817733990147, 0.15139362389630115]
```

Figure 5.8

Figure 5.8 shows the provided Mean Squared Error (MSE) values that quantifies the average squared difference between the predicted and actual values for each model with a bar chart. Lower MSE values indicate better model performance, as they suggest that the model's predictions are closer to the actual values on average. In this case, the Decision Tree model exhibits the lowest MSE of approximately 0.1514, indicating its predictions are the closest to the actual values among the four models. The Random Forest and Gradient Boosting models show identical MSE values of approximately 0.2808, slightly higher than that of the Decision Tree model. The Neural Network model exhibits the highest MSE of approximately 0.4474, suggesting its predictions deviate the most from the actual values among the models evaluated. These MSE values offer valuable insights into the accuracy of the models' predictions and can guide model selection and optimization efforts, especially in regression tasks where minimizing prediction errors is crucial.

5.8 Choosing the Best Algorithm for Deployment

To choose the best algorithm for deployment, we need to consider various factors and prioritize the evaluation metrics based on the specific requirements of the application. In this scenario, let's prioritize accuracy, precision, recall, F1-score, AUC, Cohen's Kappa coefficient, and Mean Squared Error (MSE) for model evaluation.

Based on the evaluations above, It is clearly shown that Gradient Boosting emerges as the best algorithm for deployment. It consistently performs well across multiple evaluation metrics, demonstrating high accuracy, precision, recall, F1-score, AUC, and agreement beyond chance with low MSE. This indicates that Gradient Boosting is robust and reliable for the given task and dataset. Additionally, ensemble methods like Gradient Boosting tend to generalize well and are less prone to overfitting, making them suitable for deployment in real-world scenarios. Therefore, deploying the Gradient Boosting model would likely lead to reliable and accurate predictions in production environments.

Hence, we choose Gradient Boosting as our algorithm for deployment.

6.0 Deployment

```
import pickle
import pandas as pd



risk_level_mapping = {
    0: 'High Risk',
    1: 'Low Risk',
    2: 'Mid Risk',
}

# Function to load the saved model and predict risk level
def predict_risk_level(input_data):
    # Load the saved model
    with open("gradient_boosting_model.pkl", "rb") as file:
        model = pickle.load(file)

    # Ensure consistent feature order with those used during training
    input_data_renamed = {
        'Age': input_data['Age'],
        'SystolicBP': input_data['Systolic Blood Pressure'],
        'DiastolicBP': input_data['Diastolic Blood Pressure'],
        'BS': input_data['Blood Sugar'],
        'BodyTemp': input_data['Body Temperature'],
        'HeartRate': input_data['Heart Rate'],

    }

    # Convert input to DataFrame with consistent column order
    input_df = pd.DataFrame([input_data_renamed])

    # Predict risk level
    prediction = model.predict(input_df)

    # Get the encoded label
    encoded_label = prediction[0]

    # Map the encoded label to risk level
    risk_level = risk_level_mapping[encoded_label]

    return encoded_label, risk_level

# Prompt user to input data
user_input = {}
user_input['Age'] = float(input("Enter Age: "))
user_input['Systolic Blood Pressure'] = float(input("Enter Systolic Blood Pressure: "))
user_input['Diastolic Blood Pressure'] = float(input("Enter Diastolic Blood Pressure: "))
user_input['Blood Sugar'] = float(input("Enter Blood Sugar: "))
user_input['Body Temperature'] = float(input("Enter Body Temperature: "))
user_input['Heart Rate'] = float(input("Enter Heart Rate: "))


predicted_risk_encoded, predicted_risk_label = predict_risk_level(user_input)
print("Predicted Risk Level (Number):", predicted_risk_encoded)
print("Predicted Risk Level (Description):", predicted_risk_label)
```

The code given above allows the user to provide health-related data when prompted and predicts the risk level and displays it to the user. Let's dive more into the code.

Firstly, necessary libraries are imported including "pickle" for loading the trained model and "pandas" for data manipulation.

```
import pickle
import pandas as pd
```

The dictionary "risk_level_mapping" maps encoded risk levels to their corresponding description (High Risk, Low Risk, Mid Risk). This mapping is crucial for interpreting model predictions.

```
risk_level_mapping = {
    0: 'High Risk',
    1: 'Low Risk',
    2: 'Mid Risk',
}
```

The function "predict_risk_level" is used to load a pre-trained machine learning model from a file (gradient_boosting_model.pk1) and make predictions based on user input. It renames the input data to ensure consistent feature order with those used in model training. The function then utilizes the loaded model to predict the risk level encoded label and maps the encoded label to its corresponding risk level description using "risk_level_mapping". Finally, it returns both the encoded label and the descriptive risk level.

```
# Function to load the saved model and predict risk level
def predict_risk_level(input_data):
    # Load the saved model
    with open("gradient_boosting_model.pkl", "rb") as file:
        model = pickle.load(file)

    # Ensure consistent feature order with those used during training
    input_data_renamed = {
        'Age': input_data['Age'],
        'SystolicBP': input_data['Systolic Blood Pressure'],
        'DiastolicBP': input_data['Diastolic Blood Pressure'],
        'BS': input_data['Blood Sugar'],
        'BodyTemp': input_data['Body Temperature'],
        'HeartRate': input_data['Heart Rate'],

    }

    # Convert input to DataFrame with consistent column order
    input_df = pd.DataFrame([input_data_renamed])

    # Predict risk level
    prediction = model.predict(input_df)

    # Get the encoded label
    encoded_label = prediction[0]

    # Map the encoded label to risk level
    risk_level = risk_level_mapping[encoded_label]

    return encoded_label, risk_level
```

This part of the code prompts the user to input various data such as age, systolic blood pressure, diastolic blood pressure, blood sugar level, body temperature and heart rate. This input data will be used for making predictions.

```
# Prompt user to input data
user_input = {}
user_input['Age'] = float(input("Enter Age: "))
user_input['Systolic Blood Pressure'] = float(input("Enter Systolic Blood Pressure: "))
user_input['Diastolic Blood Pressure'] = float(input("Enter Diastolic Blood Pressure: "))
user_input['Blood Sugar'] = float(input("Enter Blood Sugar: "))
user_input['Body Temperature'] = float(input("Enter Body Temperature: "))
user_input['Heart Rate'] = float(input("Enter Heart Rate: "))
```

The input data is passed to the "predict_risk_level" function to obtain the predicted risk level. The predicted result is then printed to the console in encoded and descriptive form.

```
predicted_risk_encoded, predicted_risk_label = predict_risk_level(user_input)
print("Predicted Risk Level (Number):", predicted_risk_encoded)
print("Predicted Risk Level (Description):", predicted_risk_label)
```

```
Enter Age: 32
Enter Systolic Blood Pressure: 120
Enter Diastolic Blood Pressure: 65
Enter Blood Sugar: 6
Enter Body Temperature: 101
Enter Heart Rate: 76
Predicted Risk Level (Number): 2
Predicted Risk Level (Description): Mid Risk
```

7.0 Conclusion

In conclusion, the project has been successfully concluded and we have effectively applied all the data science concepts covered in our lectures and practical sessions.

In the Data Understanding phase, we employed different visualization tools to explore our dataset. This included the use of a histogram to visualize data distributions; bar charts, box plots, heatmaps and a violin plot to visualize data relationships. We used a heatmap to determine if there are any missing values in our dataset and a histogram plot to detect outliers within the dataset.

In the Data Preparation phase, we performed data cleaning by implementing imputation methods to handle zero values and retain outliers. Additionally, we created new attributes by utilizing technical indicators, which we believed would enhance our modeling efforts. Finally, we normalize the data by performing Feature Scaling using the min_max scaling technique.

Moving on to the Modeling phase, we constructed 4 models: Gradient Boosting Model, Random Forest Model, Decision Tree Model and Neural Network Model. After the initial predictions, we refined the hyperparameters to evaluate potential enhancements to the models. Our model evaluations were based on key metrics such as Confusion Matrix, OOB Accuracy, AUC score, Kappa Coefficient, MSE, Precision, Recall and F1-Score.

Throughout the Evaluation phase, we undertook an exhaustive comparison of our models by scrutinizing the results of their respective evaluation metrics. To aid in this comparison, we generated bar charts to visually represent the performance disparities. After comparing our models, Gradient Boosting stood out as the top performer with high accuracy, precision, recall, F1-Score, AUC score.

In the Deployment phase, we created a user-friendly function enabling users to input age, systolic blood pressure, diabolic blood pressure, blood sugar level, body temperature and heart rate to predict maternal health risk. The chosen model for deployment, Gradient Boosting, efficiently generated predictions for users.

We are delighted to announce the achievement of our data mining objective, as we have successfully developed models with accuracy surpassing 80%.

Throughout this assignment, we had the chance to apply the knowledge acquired from lectures and practical sessions, thereby gaining a deeper understanding of the entire data science process. Although the initial stages of the assignment presented challenges owing to our limited

expertise in this domain, we persisted and successfully completed the project within the designated time frame with unwavering dedication.

We extend our heartfelt gratitude to our tutor, Dr. Noor Aida, for her invaluable guidance and unwavering support throughout this journey. Moving forward, we are dedicated to continuing our diligent efforts to further augment our expertise in the field of data science.

References

Building a Random Forest Model: A Step-by-Step Guide. (n.d.). Analytics Vidhya. Retrieved May 7, 2024, from https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

The Facts About High Blood Pressure. (n.d.). American Heart Association. Retrieved May 4, 2024, from https://www.heart.org/en/health-topics/high-blood-pressure

Guide, S. (2023, February 3). *Data Manipulation: Definition, Importance and Tips*. Indeed. Retrieved April 7, 2024, from https://www.indeed.com/career-advice/career-development/data-manipulation

High Blood Pressure Symptoms and Causes | cdc.gov. (n.d.). Centers for Disease Control and Prevention. Retrieved May 7, 2024, from https://www.cdc.gov/bloodpressure/about.htm

How to split a Dataset into Train and Test Sets using Python. (n.d.). Towards Data Science. Retrieved May 7, 2024, from https://towardsdatascience.com/how-to-split-a-dataset-into-training-and-testing-sets-b146b1649830

Mean fasting blood glucose. (n.d.). World Health Organization (WHO). Retrieved May 7, 2024, from https://www.who.int/data/gho/indicator-metadata-registry/imr-details/2380

Singh, V. (2023, September 23). *All About Min-max scaling. Min-max scaling, also known as… | by Pooja Vivek Singh*. Medium. Retrieved May 7, 2024, from https://medium.com/@poojaviveksingh/all-about-min-max-scaling-c7da4e0044c5

What is a Neural Network? - Artificial Neural Network Explained - AWS. (n.d.). Amazon AWS. Retrieved May 7, 2024, from https://aws.amazon.com/what-is/neural-network/

What is Decision Tree? [A Step-by-Step Guide]. (n.d.). Analytics Vidhya. Retrieved May 7, 2024, from https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/

Yadav, D. (2019, December 6). *Categorical encoding using Label-Encoding and One-Hot-Encoder*. Towards Data Science. Retrieved May 7, 2024, from https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd