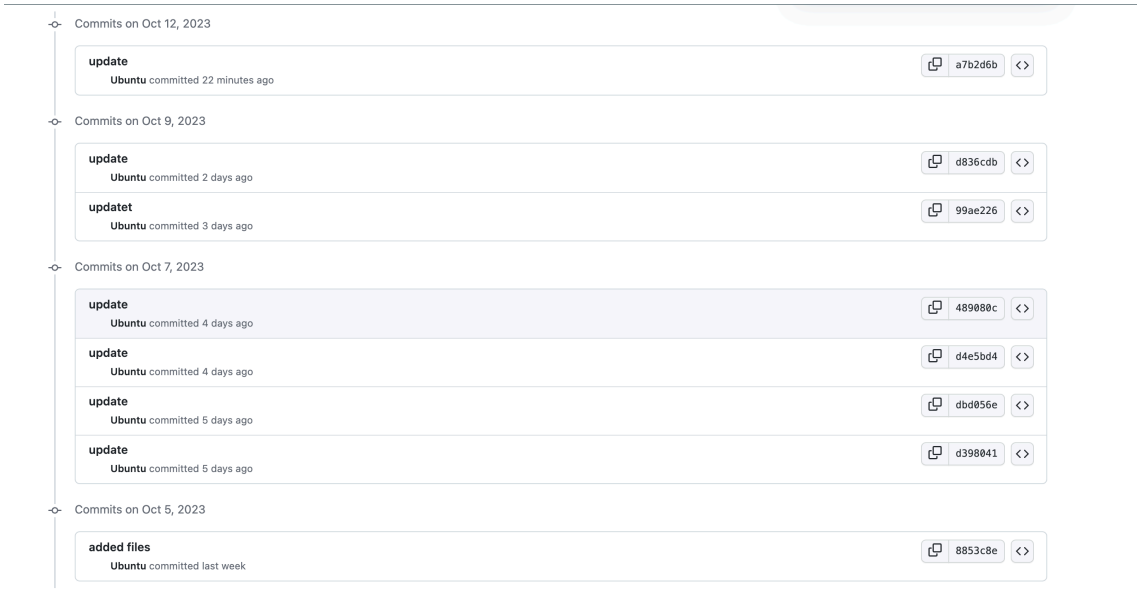
GitHub repository: https://github.com/NaomiHuang1/A3_722.git

GitHub repository: https://github.com/NaomiHuang1/A3_722.git

I pushed and committed my code from Jupyter Notebook to GitHub, below is the commit history.



Step 1. Business and/or Situation understanding

1.1 Identify the objectives of the business/situation.

In this project, we aim to respond to the United Nations' Sustainable development Goal of achieving universal health coverage for non-communicable diseases. Our goal is to use the dataset to delve deep into why people become vulnerable to diabetes and design strategies for its prevention.

It's evident that chronic diseases pose a significant challenge to global health, especially non-communicable diseases such as cardiovascular diseases, diabetes, cancer, and chronic respiratory diseases. These diseases claim 41 million lives annually, accounting for nearly three-quarters of global deaths (World Health Organization, 2023).

They inflict enormous effects on patients and place a heavy burden on societies and their families. Among these, diabetes particularly grabs our attention. The IDF Diabetes Atlas (2021) reveals that 10.5% of adults between 20 and 79 years grapple with diabetes. Alarmingly, half of them are in the dark about their condition. If we project into the future, the scenario gets grimmer. By 2045, we could see 1 out of every eight adults—a whopping 783 million people—struggling with diabetes. (International Diabetes Federation, 2023).

The Department of Health & Human Services (2004) notes that diabetes raises cholesterol and blood pressure levels, making patients more susceptible to cardiovascular diseases, Australia's leading cause of death (Department of Health & Human Services, 2004). Diabetes also affects blood vessels in the brain and harms the legs, eyes, and kidneys. Many body organs, including the digestive system, skin, reproductive organs, teeth, and immune system, are also impacted by diabetes. Given the huge influence of diabetes on patients' quality of life and the multifaceted harm, we recognize the importance of identifying the factors contributing to diabetes and urging timely prevention.

In this iteration, we have undertaken in-depth extra research to validate the importance of our research objectives. (Diabetes in Australia 2023) evident that diabetes has become a global epidemic in the 21st century, with Australia's healthcare system particularly affected. Over 300 Australians are diagnosed with diabetes daily, which means that there is one person becomes a diabetes patient every five minutes, which is surprising data. The risk of heart disease for individuals with diabetes is two to four times

greater than for those without the condition, and diabetes also stands as the seventh leading cause of death in Australia. Currently, it is the most rapidly growing chronic disease in the country, outstripping the growth rates of other chronic conditions like heart disease and cancer. The prevalence of all forms of diabetes is continually rising. In addition, relative to non-diabetics, those with diabetes face a 3.6 times higher chance of hospitalization from COVID-19 and a 2.3 times increased risk of mortality. (Diabetes in Australia 2023)

Therefore, we aspire to identify potential risks of diabetes by analyzing individuals' physical metrics and lifestyles and using this information for tailored prevention. Adopting this approach will enable us to manage better and control the incidence and mortality rates of non-communicable diseases, pushing us closer to achieving universal health coverage.

1.2 Assess the situation

In this project, we will assess the situation in six key aspects:

Resources: Central to the success of my project are robust resources. These encompass pertinent diabetes datasets and insights gleaned from authoritative diabetes-centric websites. Furthermore, meticulous time management is indispensable, ensuring each phase of the project receives its due attention.

Requirements: Upholding data privacy and adhering to ethical protocols are paramount,

especially when handling sensitive individual data. The authenticity and dependability of our data sources are foundational for deriving actionable insights. Thus, I have opted for Kaggle as the primary data reservoir and will implement rigorous data quality assessments. Through in-depth data analysis, our aim is to unearth patterns, correlations, and possible determinants to aid in diabetes prevention.

Assumptions: Within the project's ambit, we postulate that our data mirrors global demographics, suggesting that diabetes risk factors remain consistent across varied geographies. We also operate under the presumption that our Kaggle-sourced dataset is underpinned by scientific rigor and accuracy.

Constraints: Inherent challenges like data gaps or inconsistencies may emerge. Given the voluminous nature of data mining, holistic data verification at the project's onset could pose hurdles. Moreover, given the finite preparatory timeline, expertise and dataset availability might be restrictive.

Risks: One potential risk is that the expertise available for analysis could impact the project's quality and scope. Time, or the lack thereof, is another lurking risk, considering the project's demand for comprehensive completion and refinement.

Contingencies: In anticipation of potential pitfalls, I've devised a contingency strategy. An auxiliary dataset stands at the ready to bolster the data mining operation should our

primary dataset encounter unforeseen impediments.

1.3 Determine data mining objectives.

Our primary goal is to employ data mining techniques to thoroughly analyze the dataset that comprises features such as age, gender, body mass index (BMI), underlying diseases, and smoking history. We aim to understand patterns and relationships between these features and the prevalence of diabetes. We consider that recognizing the implications of these variables will be crucial in crafting more effective prevention and intervention strategies. Therefore, through data mining, we intend to explore those specific attributes, such as lifestyle choices like smoking or physiological markers like HbA1c levels, that significantly influence the likelihood of diabetes. We want to utilize this knowledge to identify individuals at risk for diabetes, then offer them effective prevention and early treatment, and provide a foundation for more effectively combating the prevalence of diabetes. By reducing the number of diabetes patients, we hope to decrease the number of deaths from non-communicable diseases, thereby increasing the chances of achieving universal health coverage.

1.4 Produce a project plan.

As we have eight steps, I plan to divide each step into three parts:

Step 1: Understand

Collect relevant materials for the project or related to each step.

Time allocation: 15% of the overall project time.

Step 2: Study

Devote time (30%) to study the methods and tools I will use for the project. This will help me gain a deeper understanding of the techniques and ensure that I apply them effectively.

Step 3: Implementation and Reporting

Allocate 55% of the overall project time to implementation and reporting.

Perform the data mining operations as planned during the implementation phase. Simultaneously work on the reporting aspect, which includes documenting my progress, findings, and insights throughout the project.

Step 2. Data understanding

2.1 Collect initial data

The search method I utilized is keyword-based, focusing on the words Healthy and Diabetes. However, the retrieved data did not fully involve enough factors that lead to diabetes. Some of the retrieved data are too small to support the project's objectives effectively. They are also insufficient for comprehensive and effective data mining, which is essential. Kaggle (Mustafa, 2023) states that this dataset is used to predict whether patients will develop diabetes based on their medical history and demographic information, which can help enable effective prevention and early treatment to enhance the overall public health level.

One of the main difficulties I encountered in the process of collecting data was

finding a suitable dataset. Many available datasets contained only one hundred or even fewer pieces of information. This small dataset size made it challenging to extract representative details. Furthermore, some datasets were overly simplistic, which limited the analysis of the data mining objective comprehensively. Therefore, finding a suitable dataset that balanced between size and richness proved to be a significant challenge for me.

During our previous iteration, we determined that the dataset's accuracy was satisfactory, yielding fairly precise results. This has proven beneficial for our data mining objectives. As a result, we intend to retain and utilize this dataset in the current iteration for continuity and consistent insights

2.2 Describe the data

The dataset that we collected is stored in the CSV format, where each row represents the record of a patient whose age ranges from 0-80, and the columns represent various features. This dataset has collected 100,000 records of the medical and demographic data (8 features) of diabetes patients and their diabetes status (positive or negative). The fields and their definitions are shown in Figure 1.

Field	Definition
gender	Gender refers to the biological sex of the individual, which can have an impact on their susceptibility to diabetes. There are three categories in it male ,female and other.
age	Age is an important factor as diabetes is more commonly diagnosed in older adults.Age ranges from 0-80 in our dataset.
hypertension	Hypertension is a medical condition in which the blood pressure in the arteries is persistently elevated. It has values a 0 or 1 where 0 indicates they don't have hypertension and for 1 it means they have hypertension.
heart_disease	Heart disease is another medical condition that is associated with an increased risk of developing diabetes. It has values a 0 or 1 where 0 indicates they don't have heart disease and for 1 it means they have heart disease.
smoking_history	Smoking history is also considered a risk factor for diabetes and can exacerbate the complications associated with diabetes.In our dataset we have 5 categories i.e not current,former,No Info,current,never and ever.
bmi	BMI (Body Mass Index) is a measure of body fat based on weight and height. Higher BMI values are linked to a higher risk of diabetes. The range of BMI in the dataset is from 10.16 to 71.55. BMI less than 18.5 is underweight, 18.5-24.9 is normal, 25-29.9 is overweight, and 30 or more is obese.
HbA1c_level	HbA1c (Hemoglobin A1c) level is a measure of a person's average blood sugar level over the past 2-3 months. Higher levels indicate a greater risk of developing diabetes. Mostly more than 6.5% of HbA1c Level indicates diabetes.
blood_glucose_level	Blood glucose level refers to the amount of glucose in the bloodstream at a given time. High blood glucose levels are a key indicator of diabetes.
diabetes	Diabetes is the target variable being predicted, with values of 1 indicating the presence of diabetes and 0 indicating the absence of diabetes.

Figure1: The dataset has nine fields which are gender（categorical）、age（numerical）、hypertension（categorical）、heart_disease（categorical）、smoking_history（categorical）、bmi（numerical）、HbA1c_level（numerical）、blood_glucose_level（numerical）、diabetes（categorical）.

2.3 Explore the data

This dataset includes age information for male and female patients from 0 to 80 years old, as well as other demographic information and their medical conditions. We can use this information to analyze the demographic data of different age groups, genders and the relationship between medical conditions such as heart disease, hypertension, and whether

they have diabetes. Through this analysis, we can determine which group of patients may have a higher risk of diabetes, which is very useful for developing personalized prevention plans. For example, as shown in Figure 2, this dataset is more focused on the analysis of the age group of patients aged 80, which may mean that the older may have a higher risk of diabetes. Figure 3 shows that the dataset contains more female patients, which may mean that gender may have a higher risk of diabetes. Furthermore, figure 4 shows that this dataset contains more records about never-smoking people, and fewer records of ever-smoking people, there are also lots of records which shows has no information of smoking history, this gives us hint that we need to remove these records that doesn't contain useful information. These figures only provide an initial observation of the dataset. Because they only display the proportion of specific groups within the dataset, it merely indicates a possible potentially higher risk of contracting diabetes. We need further analysis and data mining to arrive at a more accurate result.

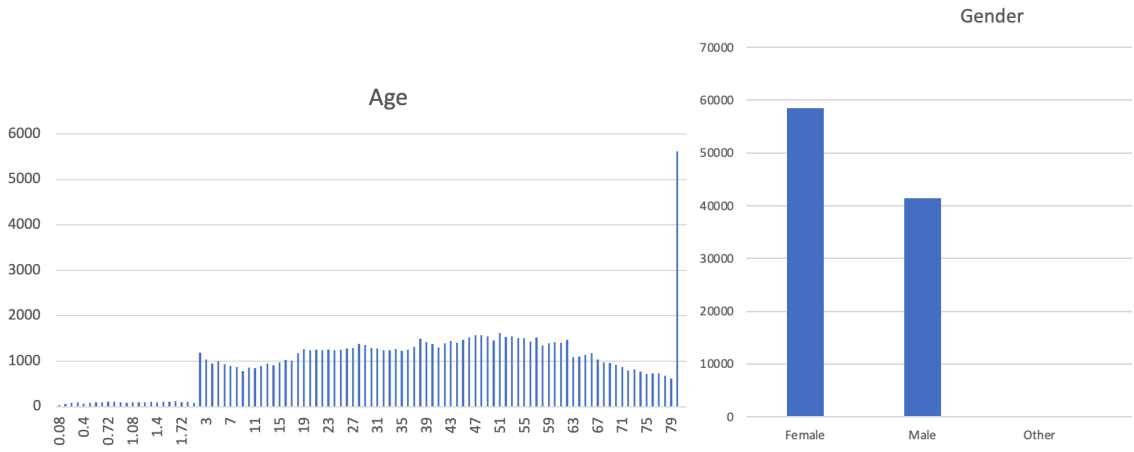


Figure 2 and figure 3.

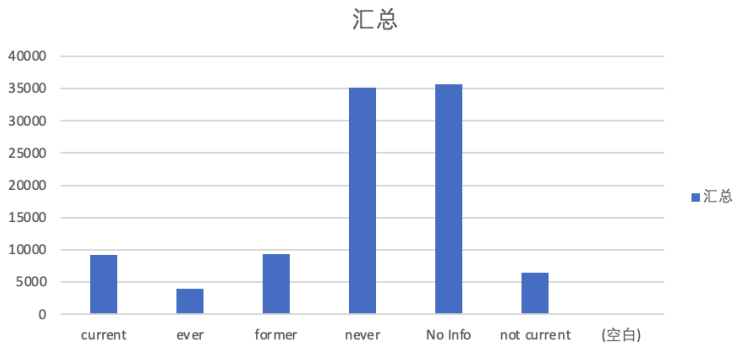


Figure4

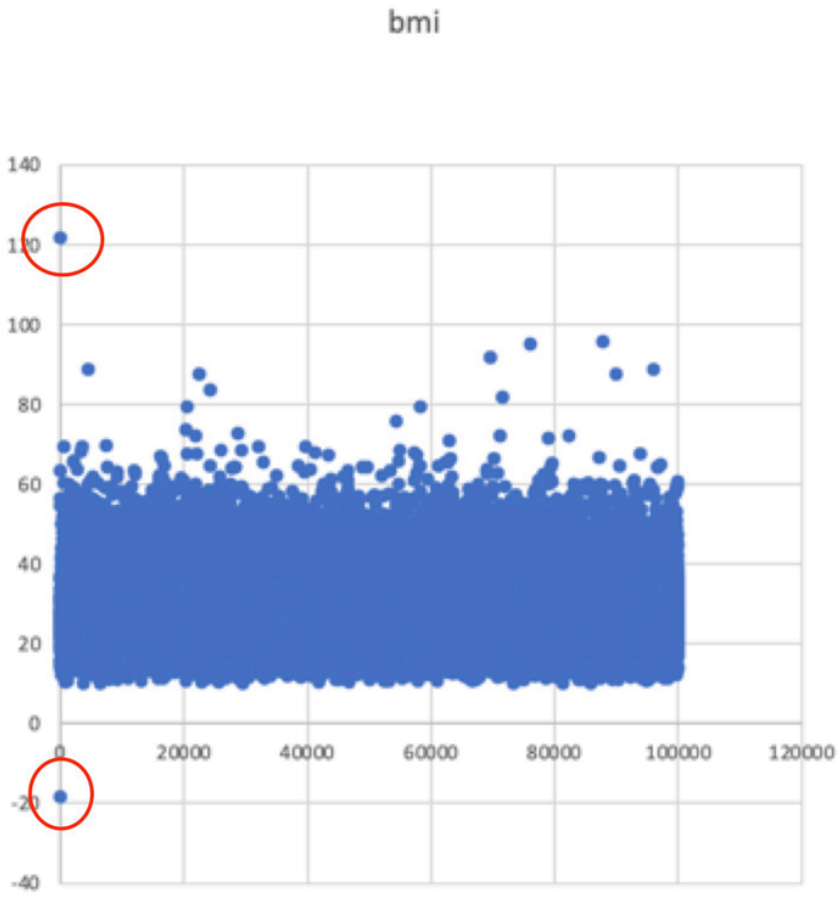
2.4 Verify the data quality

The dataset displays 1,590 missing rows out of a total of 100,000 rows. We use the COUNTBLANK function in Excel to count missing values. We noticed that the missing values were distributed across various features, with every column having some missing entries. Although there are over a thousand records with missing values, they account for only 1.59% of the total data, which leaves approximately 98.41% valid data. Therefore, these missing values are not expected to impact any specific factor or subsequent data mining tasks significantly. Nevertheless, we still plan to address and clean these missing values in the subsequent steps of the analysis.

	Missing Value
gender	164
age	157
hypertension	161
heart_disease	95
smoking_history	253
bmi	161
HbA1c_level	172
blood_glucose_level	243
diabetes	184
Total	1590

We use a scatter plot to observe if there are any evident outliers. We found that in the BMI column, there are shows that two values exceed the normal range.

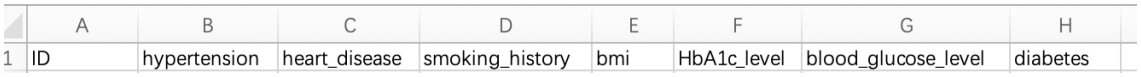
Overall, this dataset's missing values and outliers are within a reasonable range. In the following step, we can use some data cleaning methods to improve the data quality.



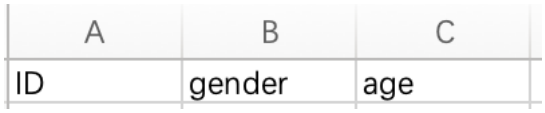
Drawing from our previous experience, we've found that this dataset is of high quality, with outcomes that align consistently with scientific research.

In order to do Step 3.1, 'Integrate various data sources,' we need to split the original dataset into two separate datasets. We added a new ID column to identify each record uniquely. Then, the two new datasets contain the columns ID, gender, and age, and ID, hypertension, smoking history, BMI, HbA1c level, blood glucose level, and diabetes

labels, respectively. We carried out this splitting directly in Excel and resulting in two new datasets. For the subsequent data preparation phase, I would use Python.



	A	B	C	D	E	F	G	H
1	ID	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes



A	B	C
ID	gender	age

After we split the dataset, we check the data again. Here are the outputs below.

```
root
 |-- ID: integer (nullable = true)
 |-- gender: string (nullable = true)
 |-- age: double (nullable = true)
```

```
root
 |-- ID: integer (nullable = true)
 |-- hypertension: integer (nullable = true)
 |-- heart_disease: integer (nullable = true)
 |-- smoking_history: string (nullable = true)
 |-- bmi: double (nullable = true)
 |-- HbA1c_level: double (nullable = true)
 |-- blood_glucose_level: integer (nullable = true)
 |-- diabetes: integer (nullable = true)
```

Step 3. Data preparation

3.1 Select the data

Our data mining goal is to analyze the factors that increase a patient's risk of developing diabetes based on their medical history and demographic information. As we need to explore all the features, the integrity and clarity of each column become important. The 'smoking history' column holds the potential risk of diabetes. However, this column has 'no info' entries, which do not contribute meaningful insights and could introduce

ambiguity in our analysis. Therefore, in the last iteration, we've decided to discard the rows containing "no info." However, in this iteration, we've decided to retain records labeled as 'no info' at this step. This is because we'll be using one-hot encoding later to format the data, which allows each category to be represented as a distinct feature. If we do not remove the 'no info' records at this stage, we can preserve other valuable information within these records. We're only missing data related to smoking history, while the remaining data in these records can still be useful for predicting our target. After completing the one-hot encoding, where 'no info' will have its own column, we plan to remove it then.

Diabetes is a metabolic disease characterized by high blood sugar levels, often accompanied by the presence of sugar in the urine (Diabetes , 2023). The "blood_glucose_level" and "HbA1c_level" are primary indicators for diagnosing diabetes. As for the diagnostic criteria for diabetes, an A1c level ≥ 6.5% (or ≥48 mmol/mol) is recognized as a diagnostic criterion for diabetes by the American Diabetes Association (ADA), the International Expert Committee (IEC), and the World Health Organization. The criteria also include using blood glucose level measurements taken during fasting (≥126 mg/dL [6.99 mmol/L]) and 2 hours after an oral glucose load (≥200 mg/dL [≥11.10 mmol/L]) for diagnosis, and this has been compared with the A1c standard in terms of sensitivity and specificity for detecting diabetes. (Classification and diagnosis of diabetes, 2023) Therefore, as "blood_glucose_level" and "HbA1c_level" serve as the basis for diagnosing diabetes and have a decisive impact on the results, I have decided to remove these two columns in order that we can better analyze and highlight other features.

```
df2 = df2.drop('blood_glucose_level', 'HbA1c_level')
df2.printSchema()
```

```
root
 |-- ID: integer (nullable = true)
 |-- hypertension: integer (nullable = true)
 |-- heart_disease: integer (nullable = true)
 |-- smoking_history: string (nullable = true)
 |-- bmi: double (nullable = true)
 |-- diabetes: integer (nullable = true)
```

3.2 Clean the data

There are a variety of methods for handling missing data. We can use the mean to substitute missing values for continuous variables, while the mode is often used for categorical ones. Alternatively, it can be replaced by a specific fixed value or randomly filled using existing data, among other techniques. (These missing values were randomly added by myself to meet this requirement.) Here is the number of missing value in each column:

```
+---+------+---+
| ID|gender|age|
+---+------+---+
|  0|   164|157|
+---+------+---+

+---+------------+-------------+---------------+---+--------+
| ID|hypertension|heart_disease|smoking_history|bmi|diabetes|
+---+------------+-------------+---------------+---+--------+
|  0|         161|           95|            253|160|     184|
+---+------------+-------------+---------------+---+--------+
```

We noticed that these missing values are distributed randomly (randomly added by myself), without any discernible patterns. We also noticed that each column has only less than 1% missing values, and the missing values essentially negate the analytical utility of the respective rows. In the last iteration, we decided to discard these entries directly.

However, we attempted to replace the missing values of continuous variables with the mean value in this iteration. We also imputed using the mode to replace the missing values for categorical data. As we use mean and mode to substitute missing values, the number of records will not change. Here is the output below.

```
avg_val = df1.agg(mean(df1['age'])).first()[0]
df1 = df1.fillna({'age': avg_val})
mode_val = df1.groupBy('gender').agg(count('gender').alias('count')).orderBy(desc('count')).first()[0]
df1 = df1.fillna({'gender': mode_val})

avg_val = df2.agg(mean(df2['hypertension'])).first()[0]
df2 = df2.fillna({'hypertension': avg_val})
avg_val = df2.agg(mean(df2['heart_disease'])).first()[0]
df2 = df2.fillna({'heart_disease': avg_val})
avg_val = df2.agg(mean(df2['bmi'])).first()[0]
df2 = df2.fillna({'bmi': avg_val})
avg_val = df2.agg(mean(df2['diabetes'])).first()[0]
df2 = df2.fillna({'diabetes': avg_val})
mode_val = df2.groupBy('smoking_history').agg(count('smoking_history').alias('count')).orderBy(desc('count')).first()[0
df2 = df2.fillna({'smoking_history': mode_val})

print("Number of entries: ", df1.count())
print("Number of entries: ", df2.count())
```

```
Number of entries:  100000
Number of entries:  100000
```

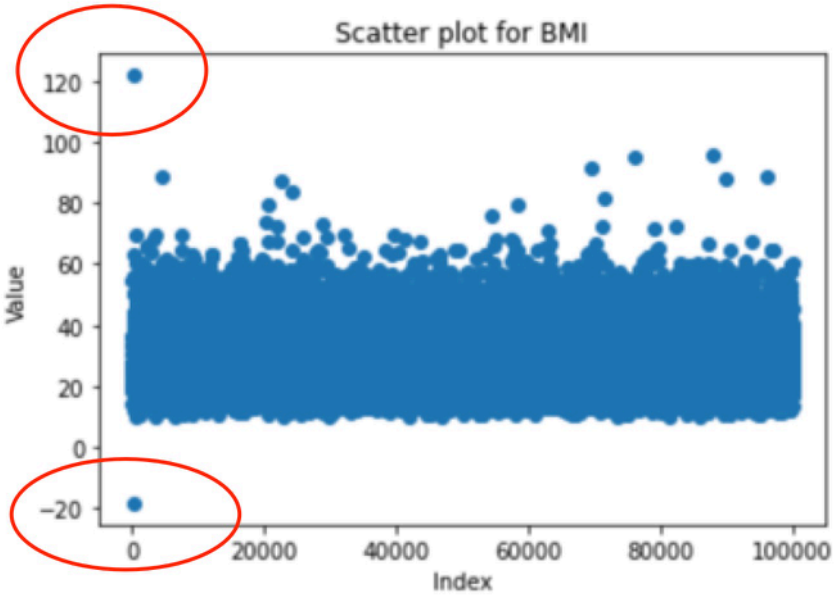
We can notice that all of the data are non-null, which shows we have substitute all the missing values.

```
+---+------+---+
| ID|gender|age|
+---+------+---+
|  0|     0|  0|
+---+------+---+
```

```
+---+------------+-------------+---------------+---+--------+
| ID|hypertension|heart_disease|smoking_history|bmi|diabetes|
+---+------------+-------------+---------------+---+--------+
|  0|           0|            0|              0|  0|       0|
+---+------------+-------------+---------------+---+--------+
```

In section 2.4, we find some outliers or extreme values in the 'bmi' column. To further investigate this, in this case, even though we are working with PySpark, we are going to use Pandas to convert and handle the data for visualization purposes.Below is the code and output.

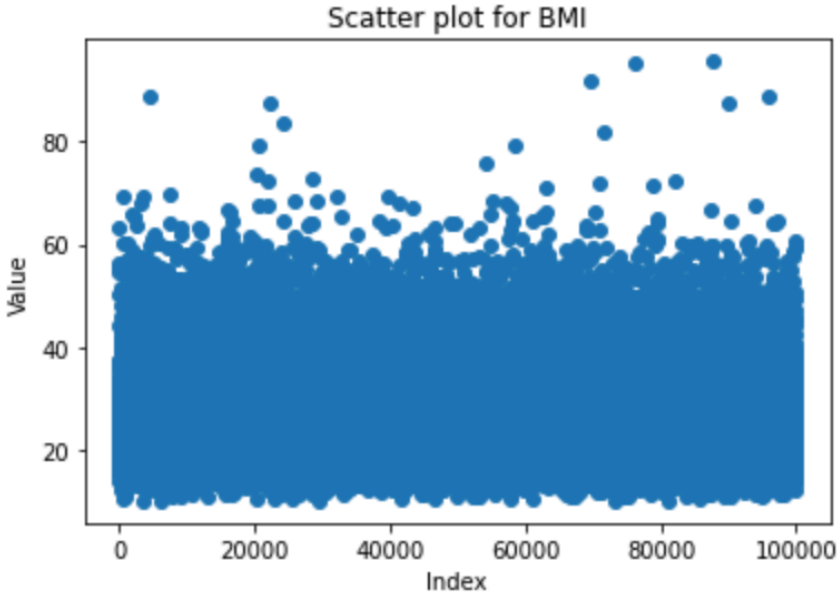
```
import pandas as pd
import matplotlib.pyplot as plt
df_pd = df2.toPandas()
plt.scatter(range(len(df_pd['bmi'])), df_pd['bmi'])
plt.title('Scatter plot for BMI')
plt.xlabel('Index')
plt.ylabel('Value')
plt.show()
```



From the visualization above, it's evident show that there's a data point below 0, and a data point above 100, which deviates from the typical range for BMI. This could be due to data entry errors, so we opted to remove it. Additionally, there are a few values exceeding a BMI of 80 and lower than 100. However, we chose to retain these because exceedingly high BMI values might influence the risk of diabetes, so leaving these outliers could be beneficial for subsequent analysis.

After we deleted the outliers, the scatter plot is shown below.

```
df2 = df2.filter((df2.bmi <= 100) & (df2.bmi >= 0))
df_pd = df2.toPandas()
plt.scatter(range(len(df_pd['bmi'])), df_pd['bmi'])
plt.title('Scatter plot for BMI')
plt.xlabel('Index')
plt.ylabel('Value')
plt.show()
```



3.3 Integrate various data sources

To facilitate subsequent analysis, we decided to integrate the data before we constructed new data. Before the data preparation, I divided the data into two separate datasets and added an 'ID' column. In this phase, we use the 'ID' column as a key for merging. Through this integration process, we established a new dataset where all features are collated within this single sheet, with the final column 'diabetes' serving as the target value. In data processing, joining is an operation that allows us to merge the data of two tables based on a common attribute, such as an ID. The most common types of joins include: inner join, which only returns rows that have matching entries in both

tables; left join, which returns all rows from the left table and appends the matching data from the right table; right join, which is the opposite of the left join; and full join, which returns all rows from both tables, filling in missing data from the other table. When we know that both tables have the same ID column and we are only interested in data under these common IDs, we opt for an inner join. This ensures the precision and consistency of the result set as it includes only data present in both original tables.

```
: df = df1.join(df2, 'id', 'inner')
  df.printSchema()
  print("Number of entries: ", df.count())
```

```
root
 |-- ID: integer (nullable = true)
 |-- gender: string (nullable = false)
 |-- age: double (nullable = false)
 |-- hypertension: integer (nullable = true)
 |-- heart_disease: integer (nullable = true)
 |-- smoking_history: string (nullable = false)
 |-- bmi: double (nullable = false)
 |-- diabetes: integer (nullable = true)

Number of entries:  99998
```

3.4 Construct the data

For BMI, we tend to focus on specific BMI ranges rather than individual BMI values. For instance, individuals with a BMI below 18 are categorized as "Underweight", then we can give this group of people some advice, such as increasing physical activity or maintaining a balanced diet. Instead of analyzing discrete values like 16.1 or 17.2, it's more practical to observe the patterns and potential health risks within these broader categories.

In this iteration, we did more research to support our view on building new categories. As stated in (Social Identity Theory 2023), when we conduct research or analysis, we aim

for conclusions that are statistically meaningful. (Social Identity Theory 2023) shows that summarizing based on the data or behavior of one person can lead to bias or misinformation. However, when we look at a larger group, individual variations or outliers have less impact on the overall results, making our conclusions more representative and reliable. If we make conclusions about a group of people which result in more generalizable and applicable to a broader range of situations and populations. Additionally, human behaviors and traits are varied and complex, with each individual having unique characteristics. Therefore, focusing on a single individual might overlook this complexity, while looking at a group can capture these variations more comprehensively. Thus, we added a new column named “Bmi class” to categorize BMI into six groups: "Underweight," "Normal weight," "Overweight," "Obesity I", "Obesity II", and "Obesity III". This categorization allows for a more intuitive and comprehensive comparison of the risk of diabetes across different weight categories, rather than merely concentrating on continuous BMI values.

```
bmi_bins = [0, 18.50, 24.99, 29.99, 34.99, 39.99, 100]
bmi_labels = ["Underweight", "Normal weight", "Overweight", "Obesity I", "Obesity II", "Obesity III"]

bmi_column_expr = when(col("bmi") < bmi_bins[0], None)
for i in range(len(bmi_bins) - 1):
    bmi_column_expr = bmi_column_expr.when((col("bmi") >= bmi_bins[i]) & (col("bmi") < bmi_bins[i + 1]), bmi_labels[i])

df = df.withColumn("Bmi class", bmi_column_expr)
```

Similarly for age, individuals within a specific age range tend to have similar physiological conditions. We segmented ages into distinct groups, which allowed us to analyze the disease risks associated with each age group, which is crucial for target identification. For different age groups, their health conditions are different. For instance, teenagers tend to be more robust, while the elderly are more prone to illnesses. By categorizing people based on their age, we can better identify which age group is more

susceptible to diabetes. If we detect an unusually high risk of disease within a particular age group, medical professionals can then formulate specific preventive measures tailored to that demographic. Thus, we added a new column named "Age group" to categorize BMI into eight groups: "Infants," "Toddlers," "Children," "Teenagers," "Young," "Middle-aged," "Older adults," and "seniors."

```
age_bins = [0, 2, 4, 13, 20, 30, 50, 65, 81]
age_labels = ["Infants", "Toddlers", "Children", "Teenagers", "Young", "Middle-aged", "Older adults", "Seniors"]

age_column_expr = when(col("age") < age_bins[0], None)
for i in range(len(age_bins) - 1):
    age_column_expr = age_column_expr.when((col("age") >= age_bins[i]) & (col("age") < age_bins[i + 1]), age_labels[i])

df = df.withColumn("Age group", age_column_expr)
```

We also constructed a new column named "Medical History Count" to count the number of medical histories of a patient. Each medical condition may alter the risk profile for developing other diseases. By quantifying the number of historical ailments an individual has, we can establish a unified metric for overall health complexity. This new feature simplifies the multifaceted nature of health conditions, allowing us to assess the patient's total health burden directly. For instance, patients with multiple diseases might be at an elevated risk for diabetes. In essence, considering the "Number of Medical Histories" provides a comprehensive health trajectory for patients, which may help in understanding and predicting diabetes.

```
medical_features = ['hypertension', 'heart_disease']

df = df.withColumn("Medical_History_Count", sum(col(feature) for feature in medical_features))
```

The information of the dataset is shown below, the last three columns were we constructed.

```
root
 |-- ID: integer (nullable = true)
 |-- gender: string (nullable = false)
 |-- age: double (nullable = false)
 |-- hypertension: integer (nullable = true)
 |-- heart_disease: integer (nullable = true)
 |-- smoking_history: string (nullable = false)
 |-- bmi: double (nullable = false)
 |-- diabetes: integer (nullable = true)
 |-- Age group: string (nullable = true)
 |-- Bmi class: string (nullable = true)
 |-- Medical_History_Count: integer (nullable = true)
```

3.5 Format the data as required

We used one-hot encoding to reformat our data. One-hot encoding is used to transform categorical variables into binary vectors. We'll apply this method to the categorical variables within our dataset. Compared to label encoding, one of the strengths of one-hot encoding is it can eliminate the implicit ordinal relationship that might arise from directly numbering categorical variables so that each category is distinctly represented. One-hot encoding enhances the model's accuracy, especially when the categorical variables are not ordinal; the results of one-hot encoding are also more intuitive and understandable. After each category is transformed into an individual feature, we can directly analyze how each single factor affects diabetes in subsequent steps. Although this encoding method might lead to a significant increase in data dimensions, the dimensions of our dataset remain within an acceptable range.

To transform categorical columns into binary format in PySpark, we first import the necessary modules, such as StringIndexer and OneHotEncoder. Next, we apply string indexing to each categorical column, assigning a unique numeric ID to each distinct string

value. We retain the original string values as labels for later use. Then, we employ one-hot encoding to convert the indexed columns into binary vectors, ensuring that each category generates a corresponding output column. By setting dropLast to False, we ensure that all categories have their output columns. We further transform the binary vectors into arrays using vector_to_array and create new columns for each unique category value based on the stored labels, converting the values to integers as needed. Finally, we remove intermediate columns like index, encoded vectors, and arrays, resulting in multiple binary columns named after the original column and the category values.

```
indexer = StringIndexer(inputCol="Age group", outputCol="Age group_index")
indexed_model = indexer.fit(df)
df = indexed_model.transform(df)
labels = indexed_model.labels
encoder = OneHotEncoder(inputCols=["Age group_index"], outputCols=["Age group_encoded"], dropLast=False)
df = encoder.fit(df).transform(df)
num_categories_age = df.select("Age group_index").distinct().count()
df = df.withColumn("Age group_array", vector_to_array("Age group_encoded"))
for i, label in enumerate(labels):
    df = df.withColumn(f"Age group_{label}", col("Age group_array")[i].cast("int"))

indexer = StringIndexer(inputCol="Bmi class", outputCol="Bmi class_index")
indexed_model = indexer.fit(df)
df = indexed_model.transform(df)
labels = indexed_model.labels
encoder = OneHotEncoder(inputCols=["Bmi class_index"], outputCols=["Bmi class_encoded"], dropLast=False)
df = encoder.fit(df).transform(df)
num_categories_bmi = df.select("Bmi class_index").distinct().count()
df = df.withColumn("Bmi class_array", vector_to_array("Bmi class_encoded"))
for i, label in enumerate(labels):
    df = df.withColumn(f"Bmi class_{label}", col("Bmi class_array")[i].cast("int"))
```

```
indexer = StringIndexer(inputCol="gender", outputCol="gender_index")
indexed_model = indexer.fit(df)
df = indexed_model.transform(df)
labels = indexed_model.labels
encoder = OneHotEncoder(inputCols=["gender_index"], outputCols=["gender_encoded"], dropLast=False)
df = encoder.fit(df).transform(df)
num_categories_gender = df.select("gender_index").distinct().count()
df = df.withColumn("gender_array", vector_to_array("gender_encoded"))
for i, label in enumerate(labels):
    df = df.withColumn(f"gender_{label}", col("gender_array")[i].cast("int"))

indexer = StringIndexer(inputCol="smoking_history", outputCol="smoking_history_index")
indexed_model = indexer.fit(df)
df = indexed_model.transform(df)
labels = indexed_model.labels
encoder = OneHotEncoder(inputCols=["smoking_history_index"], outputCols=["smoking_history_encoded"], dropLast=False)
df = encoder.fit(df).transform(df)
num_categories_smoking = df.select("smoking_history_index").distinct().count()
df = df.withColumn("smoking_history_array", vector_to_array("smoking_history_encoded"))
for i, label in enumerate(labels):
    df = df.withColumn(f"smoking_history_{label}", col("smoking_history_array")[i].cast("int"))
```

```
indexer = StringIndexer(inputCol="Medical_History_Count", outputCol="Medical_History_Count_index")
indexed_model = indexer.fit(df)
df = indexed_model.transform(df)
labels = indexed_model.labels
encoder = OneHotEncoder(inputCols=["Medical_History_Count_index"], outputCols=["Medical_History_Count_encoded"], dropLa
df = encoder.fit(df).transform(df)
num_categories_medical = df.select("Medical_History_Count_index").distinct().count()
df = df.withColumn("Medical_History_Count_array", vector_to_array("Medical_History_Count_encoded"))
for i, label in enumerate(labels):
    df = df.withColumn(f"Medical_History_Count_{label}", col("Medical_History_Count_array")[i].cast("int"))

columns_to_drop = ["Age group_index", "Bmi class_index", "gender_index", "smoking_history_index",
                   "Medical_History_Count_index", "Age group_encoded", "Bmi class_encoded",
                   "gender_encoded", "smoking_history_encoded", "Medical_History_Count_encoded",
                   "Age group_array", "Bmi class_array", "gender_array", "smoking_history_array",
                   "Medical_History_Count_array"]
df = df.drop(*columns_to_drop)

df.printSchema()
```

Then we got new columns to represent each category in the features.

```
root
 |-- ID: integer (nullable = true)
 |-- gender: string (nullable = false)
 |-- age: double (nullable = false)
 |-- hypertension: integer (nullable = true)
 |-- heart_disease: integer (nullable = true)
 |-- smoking_history: string (nullable = false)
 |-- bmi: double (nullable = false)
 |-- diabetes: integer (nullable = true)
 |-- Age group: string (nullable = true)
 |-- Bmi class: string (nullable = true)
 |-- Medical_History_Count: integer (nullable = true)
 |-- Age group_Middle-aged: integer (nullable = true)
 |-- Age group_Older adults: integer (nullable = true)
 |-- Age group_Seniors: integer (nullable = true)
 |-- Age group_Young: integer (nullable = true)
 |-- Age group_Children: integer (nullable = true)
 |-- Age group_Teenagers: integer (nullable = true)
 |-- Age group_Toddlers: integer (nullable = true)
 |-- Age group_Infants: integer (nullable = true)
 |-- Bmi class_Overweight: integer (nullable = true)
 |-- Bmi class_Normal weight: integer (nullable = true)
 |-- Bmi class_Obesity I: integer (nullable = true)
 |-- Bmi class_Underweight: integer (nullable = true)
 |-- Bmi class_Obesity II: integer (nullable = true)
 |-- Bmi class_Obesity III: integer (nullable = true)
 |-- gender_Female: integer (nullable = true)
 |-- gender_Male: integer (nullable = true)
 |-- gender_Other: integer (nullable = true)
 |-- smoking_history_No Info: integer (nullable = true)
 |-- smoking_history_never: integer (nullable = true)
 |-- smoking_history_former: integer (nullable = true)
 |-- smoking_history_current: integer (nullable = true)
 |-- smoking_history_not current: integer (nullable = true)
 |-- smoking_history_ever: integer (nullable = true)
 |-- Medical_History_Count_0: integer (nullable = true)
 |-- Medical_History_Count_1: integer (nullable = true)
 |-- Medical_History_Count_2: integer (nullable = true)
```

Step 4. Data transformation

4.1 Reduce the data

After our previous step of label encoding the categorical variables and generating new encoded columns, the original unencoded columns are no longer necessary for our

subsequent analysis. Therefore, we've opted to remove these original columns to achieve horizontal reduction so that we can streamline our dataset and make it more manageable and efficient for processing. We also decided to remove the ID column because ID is also unnecessary for our analysis. We have grouped age and bmi into new columns, and then the original continuous variable columns are no longer useful, so we will delete them as well.

We also deleted the smoking history of "no_info" at this stage. This makes the dataset contain only clear, actionable information, thus better enabling us to draw accurate and reliable conclusions on the linkages between smoking patterns and the likelihood of diabetes onset.

```
columns_to_keep = ['hypertension','heart_disease','diabetes','Age group_Middle-aged','Age group_Older adults',
                   'Age group_Seniors','Age group_Young','Age group_Children','Age group_Teenagers',
                   'Age group_Toddlers', 'Age group_Infants','Bmi class_Overweight','Bmi class_Normal weight',
                   'Bmi class_Obesity I', 'Bmi class_Underweight','Bmi class_Obesity II','Bmi class_Obesity III',
                   'gender_Female','gender_Male', 'gender_Other', 'smoking_history_never','smoking_history_former',
                   'smoking_history_current','smoking_history_not current','smoking_history_ever',
                   'Medical_History_Count_0','Medical_History_Count_1','Medical_History_Count_2'
]

df = df.select(columns_to_keep)
df.columns
```

Now, the dataset structure is shown below.

```
['hypertension',
 'heart_disease',
 'diabetes',
 'Age group_Middle-aged',
 'Age group_Older adults',
 'Age group_Seniors',
 'Age group_Young',
 'Age group_Children',
 'Age group_Teenagers',
 'Age group_Toddlers',
 'Age group_Infants',
 'Bmi class_Overweight',
 'Bmi class_Normal weight',
 'Bmi class_Obesity I',
 'Bmi class_Underweight',
 'Bmi class_Obesity II',
 'Bmi class_Obesity III',
 'gender_Female',
 'gender_Male',
 'gender_Other',
 'smoking_history_never',
 'smoking_history_former',
 'smoking_history_current',
 'smoking_history_not current',
 'smoking_history_ever',
 'Medical_History_Count_0',
 'Medical_History_Count_1',
 'Medical_History_Count_2']
```

Then, we used Random Forests for feature selection based on the importance. This is because the random forest is adept at capturing potential non-linear relationships and interactions between features present in our dataset. Random Forests have the ability to identify and prioritize features that exert the most influence on the target variable, and it offer a robust evaluation of feature importance. Hence, we chose Random Forests for feature selection, and it helped us delete those features that are least important for predicting the target. We first extract all columns except for the 'diabetes' column. This is done because 'diabetes' is our target variable, and the rest represent features that will be used to predict this target. To prepare the data for machine learning, the selected feature columns are transformed into a consolidated vector format using PySpark's VectorAssembler. This transformation is used for PySpark's ML algorithms, which predominantly expect input data to be in this vectorized format. Once the data is appropriately formatted, we can construct the RandomForest classifier. This classifier is then trained on our vectorized features, building a model that captures patterns and

relationships between the features and the 'diabetes' target. After the training is complete, one of the valuable outputs we can extract from this model is the importance of each feature. The code retrieves these importances, and sorts them in descending order to highlight the most significant features at the top, and finally prints them out. This sorted list provides insights into which features play a crucial role in predicting diabetes based on the data provided.

```
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import VectorAssembler

input_columns = [col for col in df.columns if col != 'diabetes']
assembler = VectorAssembler(inputCols=input_columns, outputCol="features")
data = assembler.transform(df)

x = data.select("features")
y = data.select("diabetes")

clf = RandomForestClassifier(numTrees=100, labelCol="diabetes", featuresCol="features")
model = clf.fit(data)

importances = model.featureImportances
feature_importance = dict(zip(input_columns, importances))

sorted_feature_importance = sorted(feature_importance.items(), key=lambda x: x[1], reverse=True)
for feature, importance in sorted_feature_importance:
    print(f"{feature}: {importance}")
```

Below is the output:

```
Medical_History_Count_0: 0.27285131491127146
Medical_History_Count_1: 0.17190562879308932
hypertension: 0.14287036949413354
Age group_Seniors: 0.12428758500456671
Bmi class_Obesity III: 0.11025609672989892
heart_disease: 0.04930370498940422
Medical_History_Count_2: 0.03401067423086855
Bmi class_Obesity II: 0.028589604031664353
Age group_Older adults: 0.0159494568173873
smoking_history_former: 0.014067004722848457
Age group_Young: 0.00964720189409095
Age group_Middle-aged: 0.008617858759677152
Bmi class_Normal weight: 0.005114516977826421
Bmi class_Overweight: 0.0027386843240689
gender_Male: 0.002413732969458538
Bmi class_Underweight: 0.0015323096841776157
Bmi class_Obesity I: 0.00139978709985935
gender_Female: 0.0011731650175771538
smoking_history_not current: 0.0008544345171869007
smoking_history_ever: 0.0007335540278995236
smoking_history_current: 0.0007195924203361026
smoking_history_never: 0.0005681593987859043
Age group_Teenagers: 0.00033005145007326066
Age group_Children: 6.551173384972096e-05
Age group_Toddlers: 0.0
Age group_Infants: 0.0
gender_Other: 0.0
```

The output highlights the importance of different features in predicting if one has

diabetes or not. We can notice Medical_History_Count_0 has the highest importance score of about 0.273, indicating that it is the most influential feature in the model. This is followed by Medical_History_Count_1 and hypertension with scores of approximately 0.172 and 0.143, respectively. These high scores mean that these features play a significant role in the model's predictions. Some features like Age group_Toddlers, Age group_Infants, and gender_Other have an importance score of 0.0. This means that during the training of this model, these features had no influence on the model's decision-making process, and they might not be valuable predictors for this specific dataset and problem. Therefore, we decided to drop these features from the dataset.

```
non_important_features = ['Age group_Toddlers', 'Age group_Infants', 'gender_Other']
for feature in non_important_features:
    df = df.drop(feature)
```

4.2 Project the data

We first show the distribution of the target value.

```
diabetes_counts = df.groupBy('diabetes').count().collect()
print({row['diabetes']: row['count'] for row in diabetes_counts})
```

```
{1: 8495, 0: 91503}
```

The result shows the imbalances in my target value. Balancing data is important. Our model may not learn well when our data has more of one class than another. This means our model might often guess the bigger class and miss the smaller one. If we balance the data, our model can learn every class better. This gives us better results. Balanced data makes the model work better in real situations. It helps the model understand the small class better and makes the model's guesses more stable. So, balancing the data is good

for a better and more trustworthy model. We first segmented the data into two groups: those with 'diabetes' value of 0 (df_high) and those with a value of 1 (df_low). It then calculates the fraction of records needed from df_high to match the count of df_low. We then used this fraction to sample a subset of df_high to create df_high_reduce, so that we can ensure both groups have comparable sizes. The two balanced subsets are then merged into df_reduced, resulting in a dataset with an approximately equal distribution of 'diabetes' values of 0 and 1

```
df_high = df.filter(col('diabetes') == 0)
df_low = df.filter(col('diabetes') == 1)
fraction = df_low.count() / df_high.count()
df_high_reduce = df_high.sample(False, fraction)
df_reduced = df_low.union(df_high_reduce)

reduced_diabetes_counts = df_reduced.groupBy('diabetes').count().collect()
print({row['diabetes']: row['count'] for row in reduced_diabetes_counts})
```

```
{1: 8495, 0: 8554}
```

Step 5. Data-mining method(s) selection

My data mining objective is mainly focused on analyzing factors influencing diabetes, with the column 'diabetes.' There are a few data mining methodologies that come to mind when we consider this. Classification stands out due to its numerous algorithms that have historically been employed for such predictive tasks. These algorithms include tree-based algorithms such as Decision Trees, Random Forests, and other algorithms such as SVM. The benefit of classification is its ability is not only segregate data based on the defined categories but also to highlight the weightage and significance of various features influencing these categories.

We also consider Clustering methods, such as K-Means or Hierarchical clustering, to segment the dataset into groups based on the similarities within the data. Although these techniques might uncover certain previously unseen patterns or groupings within the patient data, they won't directly label or categorize based on the diabetes outcome. So, it works well when we want to discover unlabeled groups or patterns in a dataset. For instance, if we want to find out if there are any groupings of people that differ in gender, age, and BMI, clustering would be suitable.

Regression, while primarily used for predicting continuous variables, also offers certain algorithms like Logistic Regression that cater to binary outcomes. Regression focuses on modeling the relationship between a dependent variable and one or more independent variables, particularly when dealing with continuous outcomes. For example, if our goal were to predict a specific frequency or degree of diabetes, not just categorize medical history and demographic information and predict their importance, then regression would be more suitable. However, our specific objective of predicting a binary outcome (has diabetes or doesn't have diabetes), we require more extra assumptions checks - such as linearity, independence, homoscedasticity, and normality. Failing to meet these assumptions can lead to biased, imprecise, or invalid predictions. Given our objective, these rigorous checks and requirements might introduce unnecessary complexities.

When comparing these methods for our diabetes-focused investigation, classification offers direct predictive capabilities with features' significance. On the other hand, classification directly categorizes data based on defined classes, making it more intuitive

for our binary outcome prediction. It provides categorical predictions without requiring the assumptions and complexities associated with a regression model, making it a more straightforward choice for our study. Moreover, I have previously undertaken steps for data balancing and feature selection. Balancing the data helps address the issue of class imbalance, enhancing the model's ability to recognize the minority class. Feature selection, on the other hand, enables me to concentrate on those features most pertinent to the target, reducing noise that unnecessary or redundant features might introduce. Both steps were taken to improve the model's performance and interpretability.

Step 6. Data-mining algorithm(s) selection

6.1 Conduct exploratory analysis and discuss

To gain a deeper understanding of how various algorithms perform in predicting diabetes, we carried out an exploratory analysis of multiple algorithms. As our objective is to analyze the factors influencing diabetes and make classification predictions, we first explored several primary algorithms in supervised learning.

As we used Pyspark, there are many tree-based models available for us to compare; each of them has its distinct characteristics and use cases. Decision trees are the most intuitive; they form a tree structure by recursively splitting the data, offering a clear understanding and a visual representation of data splits. . They split the data step by step, making it easy to see and understand how decisions are made. The Random forests

combine results from multiple decision trees, which ensures a more stable and accurate prediction. Random forests also effectively minimize the overfitting that is potentially seen in single-decision trees. This means they're less likely to get stuff wrong because they're cross-checking with multiple trees. It's like asking a group of friends for advice instead of just one. This group approach means they're not only better at guessing, but they also avoid some of the mistakes a single decision tree might make. Additionally, gradient-boosting trees (GBT) differentiate themselves by iteratively refining their predictions. Unlike other models, GBTs learn from the errors of previous iterations, thereby improving accuracy over time. This method allows GBTs to fit various loss functions and handle missing data efficiently. Furthermore, GBTs are resistant to overfitting as well, making them especially effective for complex datasets. Through iterative error correction and versatile adaptability, GBTs offer a reliable approach to data modeling.

We also considered logistic regression, which is a probability-based linear classifier. Although they can show the relationship between features and output, they might not be good at capturing non-linear patterns. Advanced algorithms like SVM and neural networks were also taken into account. However, due to their computational complexity and the need for interpretability, we decided to focus solely on tree-based models for our final selection.

We then use code to try each model, below is their output, which we will discuss in the next step.

```
Decision Tree - Accuracy: 0.7465
Decision Tree - 0 - Precision: 0.7566
Decision Tree - 0 - Recall: 0.7415
Decision Tree - 0 - F1 Score: 0.7490
Decision Tree - 1 - Precision: 0.7364
Decision Tree - 1 - Recall: 0.7517
Decision Tree - 1 - F1 Score: 0.7440
```

```
Random Forest - Accuracy: 0.7256
Random Forest - 0 - Precision: 0.7123
Random Forest - 0 - Recall: 0.7749
Random Forest - 0 - F1 Score: 0.7423
Random Forest - 1 - Precision: 0.7421
Random Forest - 1 - Recall: 0.6742
Random Forest - 1 - F1 Score: 0.7065
```

```
Gradient Boosted Trees - Accuracy: 0.7510
Gradient Boosted Trees - 0 - Precision: 0.7775
Gradient Boosted Trees - 0 - Recall: 0.7169
Gradient Boosted Trees - 0 - F1 Score: 0.7460
Gradient Boosted Trees - 1 - Precision: 0.7274
Gradient Boosted Trees - 1 - Recall: 0.7865
Gradient Boosted Trees - 1 - F1 Score: 0.7558
```

6.2 Select data-mining algorithms based on discussion

From the output, the Decision Tree model presents an accuracy of 74.65%. For label 0, precision stands at 0.7566, and recall at 0.7415, demonstrating a balanced performance in both metrics. For label 1, precision and recall hover around the 0.74 mark, implying consistent positive predictions. For the Random Forest model, it showcases an accuracy of 72.56%. For label 0, there's a noticeable difference between precision at 0.7123 and recall at 0.7749. This indicates a certain lean in the model's ability to predict label 0, with a potential overestimation. For label 1, the precision at 0.7421 and recall at 0.6742 further underscores this observation. The Gradient Boosted Trees (GBT) model emerges with an accuracy of 75.10%. In comparison to the Decision Tree, GBT has a slightly increased precision at 0.7775 for label 0 but a decreased recall at 0.7169. This may suggest that the GBT model exercises more caution when predicting label 0, thereby minimizing false positives. On the other hand, for label 1, both precision and recall are enhanced, making GBT the most balanced among the three in terms of prediction strategies for both labels.

All three models offer very close performances, with GBT slightly leading the pack.

Considering the importance of precision, recall, and the F1 score, these models provide fairly robust and balanced predictions. Therefore, we chose GBT as our model.

6.3 Build/Select appropriate model(s) and choose relevant parameter(s)

In our recent modeling endeavors, we utilized PySpark's CrossValidator to enhance the performance of our Gradient Boosted Trees (GBT) model. Specifically, we examined various depths by setting the maxDepth parameter to values like [2, 4, 6]. Additionally, we tried different iteration counts by adjusting the maxIter to [10, 20] and explored learning rates by tweaking the stepSize to [0.1, 0.01]. To validate the efficacy of each combination, we employed a 5-fold cross-validation, which ensured that our team could assess the performance implications of each parameter set. This detailed approach, albeit computationally demanding, but can provide us with the optimal parameters: a maxDepth of 4, maxIter of 20, and a stepSize of 0.1. Armed with these settings, our GBT model is now better equipped to generate reliable insights from the dataset.

```
paramGrid = (ParamGridBuilder()
             .addGrid(gbt_classifier.maxDepth, [2, 4, 6])
             .addGrid(gbt_classifier.maxIter, [10, 20])
             .addGrid(gbt_classifier.stepSize, [0.1, 0.05,0.01])
             .build())
```

We would use these parameters to train our models. This approach ensures our model delivers a strong performance without being prone to overfitting and provides a robust solution for the project.

```
best_max_depth = 4
best_max_iter = 20
best_step_size = 0.1

assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
train_data, test_data = df_reduced.randomSplit([0.8, 0.2], seed=42)

gbt_classifier = GBTClassifier(labelCol="diabetes",
                               featuresCol="features",
                               maxDepth=best_max_depth,
                               maxIter=best_max_iter,
                               stepSize=best_step_size)
pipeline = Pipeline(stages=[assembler, gbt_classifier])
```

The use of seed = 42 or any other specific seed value in data science and machine learning aims to ensure reproducibility. When we deal with algorithms that have a random component, such as random data splits or initializations, using a seed ensures that the randomness is consistent every time the code is run. By setting a seed, we make sure that our results can be reproduced exactly by ourselves.

Step 7. Data Mining

7.1 Create and justify test designs

During our data mining project, we opted for a 70/30 division for our training and testing datasets, a choice primarily influenced by prevalent industry standards and also considered accuracy. While many splits exist, a cursory internet search reveals that the 80/20 split is often the most recommended, closely followed by the 70/30 ratio. The rationale behind this preference underscores the significance of allocating ample data for training to ensure the model comprehensively learns the underlying patterns. Simultaneously, it emphasizes maintaining a substantial subset for testing, ensuring we can effectively gauge the model's ability to generalize to new, unseen data. By designating

80% of the data for training, we aspired for our model to delve deeper into the intricacies of the data, thereby potentially amplifying its predictive accuracy. The remaining 20% earmarked for testing offers a robust buffer so that it can reliably evaluate the model's performance. This setup yielded an accuracy of approximately 75.20%.

However, when we experimented with the 70/30 distribution, we observed a slightly improved accuracy of 75.84%. This marginal yet noteworthy increase made us use the 70/30 split for our project, as it seemed to strike a more optimal balance between training depth and validation breadth in the context of our specific dataset and objectives.

```
Gradient Boosted Trees - Accuracy: 0.7520
```

```
Gradient Boosted Trees - Accuracy: 0.7584
```

7.2 Conduct data mining – classify, regress, cluster, etc. (models must execute)

I run the model and make sure that the model can be executed. The data mining method is classification, so I used the GBT as my model. The model can be run correctly. Below is the code.

```
best_max_depth = 4
best_max_iter = 20
best_step_size = 0.1

assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")

train_data, test_data = df_reduced.randomSplit([0.8, 0.2], seed=42)

gbt_classifier = GBTClassifier(
    labelCol="diabetes",
    featuresCol="features",
    maxDepth=4,
    maxIter=20,
    stepSize=0.1
)

pipeline = Pipeline(stages=[assembler, gbt_classifier])
gbt_model = pipeline.fit(train_data)
predictions_gbt = gbt_model.transform(test_data)
```

7.3 Search for patterns

The first pattern is the result of the model. The GBT model was employed for the classification task, achieving an accuracy of 0.7584 on the test set. For class 0, the model correctly predicted it 70.57% of the time. In contrast, class 1 was predicted with a higher accuracy of 81.10%. This suggests that the model is somewhat better at correctly identifying instances of class 1 compared to class 0.

```
Gradient Boosted Trees - Accuracy: 0.7584
Gradient Boosted Trees - 0 - Precision: 0.7884
Gradient Boosted Trees - 0 - Recall: 0.7057
Gradient Boosted Trees - 0 - F1 Score: 0.7447
Gradient Boosted Trees - 1 - Precision: 0.7341
Gradient Boosted Trees - 1 - Recall: 0.8110
Gradient Boosted Trees - 1 - F1 Score: 0.7706
```

```
Accuracy for class 0: 0.7057
Accuracy for class 1: 0.8110
```

The Gradient Boosted Trees model showcases the significance of various features in predicting diabetes through its feature importance rankings. The standout feature is 'Age group_Seniors' with an importance score of 0.1403, emphasizing that there is a link between older age and heightened diabetes risk. The 'Medical_History_Count_0' at 0.1153 indicates that individuals with a limited medical history might be uniquely associated with the disease. The BMI classifications are evident with categories such as 'Obesity III', 'Normal weight', and other obesity levels showcasing their influence. In particular, 'Bmi class_Obesity III' holds a significant score of 0.1006, showing the health implications of one's weight. Other age brackets, namely 'Older adults' and 'Middle-aged', are also notably influential, reinforcing the correlation between age and diabetes

susceptibility. Smoking habits and gender further contribute to the model's predictive capabilities, albeit to a lesser extent.

Hence, the model's feature importance illustrates the intricate interplay between age, health history, weight, smoking tendencies, and diabetes risk. This pattern is essential for a comprehensive understanding of the myriad factors influencing diabetes prediction.

```
Feature Age group_Seniors: 0.1403
Feature Medical_History_Count_0: 0.1153
Feature Bmi class_Obesity III: 0.1006
Feature Age group_Older adults: 0.0941
Feature Age group_Middle-aged: 0.0785
Feature Bmi class_Normal weight: 0.0633
Feature Bmi class_Obesity II: 0.0626
Feature Bmi class_Obesity I: 0.0591
Feature Bmi class_Underweight: 0.0440
Feature Bmi class_Overweight: 0.0409
Feature smoking_history_current: 0.0337
Feature smoking_history_never: 0.0321
Feature smoking_history_former: 0.0298
Feature gender_Female: 0.0246
Feature Medical_History_Count_1: 0.0155
Feature heart_disease: 0.0104
Feature smoking_history_not current: 0.0098
Feature Age group_Teenagers: 0.0092
Feature Age group_Young: 0.0080
Feature Age group_Children: 0.0080
Feature hypertension: 0.0075
Feature smoking_history_ever: 0.0066
Feature gender_Male: 0.0035
Feature Medical_History_Count_2: 0.0027
```

Based on the results from "pattern2", we can discern the importance of each feature. However, these results lack interpretability; for instance, we cannot determine whether the senior group is more or less prone to diabetes. To enhance interpretability, we used the SHAP values in Python. SHAP (SHapley Additive exPlanations) offers a method for explaining machine learning model predictions. It allocates a value for each prediction, indicating how a particular feature value impacts that prediction. SHAP values assist in

understanding how the model's predictions for specific observations are determined by individual feature values. We first convert the Spark DataFrame to a Pandas DataFrame, then train a GBT model by using the same parameter we used in Pyspark, with Scikit-learn, and finally use the SHAP library to interpret and plot the model's feature importances. The reason we opt for this approach instead of directly using PySpark is due to the lack of native support for SHAP interpretation in PySpark. By transitioning to a single-node, in-memory computation with Pandas and Scikit-learn, we can leverage existing tools like SHAP to achieve our model interpretation goals. We utilized the shap.summary_plot() to visualize SHAP values for all observations, which we term our third pattern.

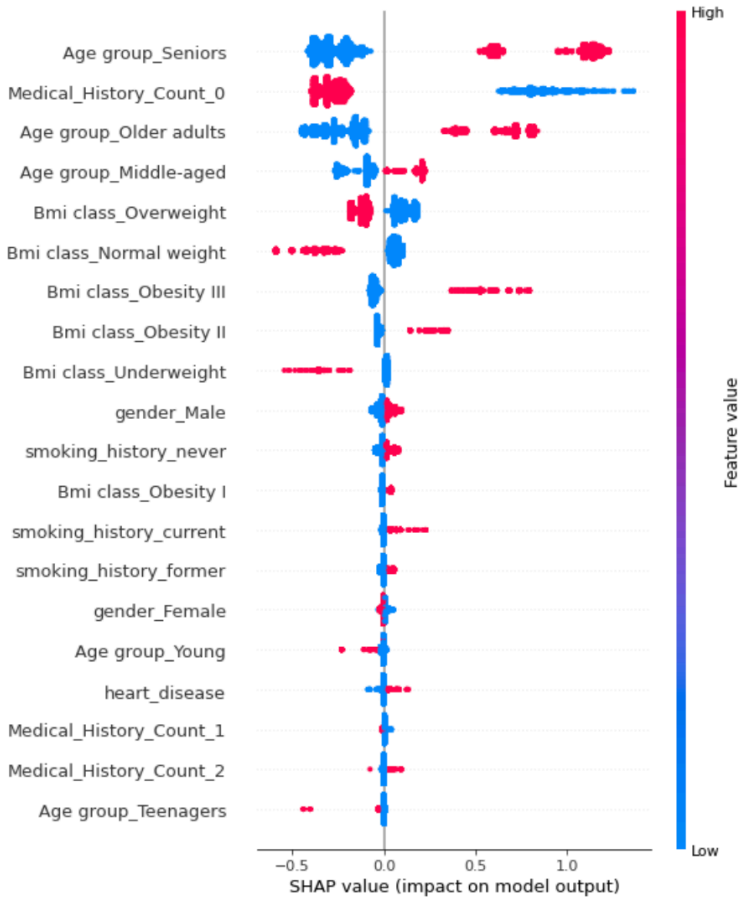
```
pandas_df = df_reduced.toPandas()

clf = GradientBoostingClassifier(max_depth=4, n_estimators=20, learning_rate=0.1)
X = pandas_df[feature_cols]
y = pandas_df['diabetes']
clf.fit(X, y)

explainer = shap.TreeExplainer(clf)
shap_values = explainer.shap_values(X)

shap.summary_plot(shap_values, X)
```

In the SHAP summary plot, colors (red and blue) denote the actual value of a specific feature. In the figure, for the 'Age_group_senior' feature, blue dots represent samples where "Age_group_senior" is false, while red dots indicate samples where "Age_group_senior" is true. All of the blue dots reside on the left side, which suggests that when "Age_group_senior" is false, the model tends to predict that they don't have diabetes. The red dots are concentrated on the right, implying that when an individual belongs to the "senior" category, the model is inclined to predict they have diabetes.

For the feature "Age_group_older", the majority of these blue dots are concentrated

on the left, which means that for the samples that are not in the "older" age group, the model tends to predict they do not have diabetes. Red dots signify samples where individuals are in the "older" age group. These dots are exclusively on the right, which means that the model predicts a diabetes risk for every sample in the "older" age group.

For the feature "Medical_History_Count_0", blue dots symbolize samples where "Medical_History_Count_0" is false, primarily clustering on the right side. This indicates that for most samples that don't have a medical history count of zero, the model leans toward predicting they have diabetes. Conversely, the red dots, representing samples with a medical history count of zero, are predominantly congregated on the left. This signifies that the model usually predicts a lower risk of diabetes for these samples.



Step 8. Interpretation

8.1 Study and discuss the mined patterns

From pattern 1, we realized that the Gradient Boosted Trees (GBT) model was suitable for this classification task because of its proficiency in handling diverse datasets and intrinsic capability to yield feature importance. Class 1 was predicted with a higher accuracy than Class 0. This suggests that the model is somewhat better at correctly identifying instances of class 1 compared to class 0.

From pattern 2 and pattern 3, we found that the 'Senior' and 'Older Adults' age groups in feature importance underscore age as a risk factor for diabetes. This corroborates many scientific investigations which have identified age as a primary contributor. The SHAP values further solidify the weight of age in predictions. The SHAP value provided granular insights; they are indispensable. It allows for the dissection of feature interactions and their bearing on the predictions, making the model's decision-making process more transparent. The "Medical History Count - 0" feature, with red dots densely packed on the left and blue dots on the right, suggests that individuals with no prior medical issues are generally not predicted by the model to have diabetes. This indicates a protective aspect of having fewer health complications, reinforcing the idea that multiple health issues might have a compounded effect, increasing the risk of diabetes. For the BMI features, the SHAP summary for "Normal Weight" and “underweight” shows red dots (indicating true) primarily on the left and blue dots (indicating false) mostly on the right. This suggests that individuals with normal weight are generally not

predicted to have diabetes by the model, while those outside this category show a higher risk of having diabetes. "Obesity 3" and "Obesity 2" shows significance in the feature importance list and it can highlight the known health narrative: obesity, especially in its extreme form, poses a heightened risk for diabetes. The SHAP summary for "overweight" predominantly positions blue dots (indicating false) on the right, suggesting that individuals who are not overweight are generally predicted by the model to have a higher risk of diabetes. In contrast, the red dots (indicating true) are densely packed on the left, implying that overweight individuals are typically not predicted to be diabetic. This is a counterintuitive finding. While being overweight might be a risk factor for diabetes in general, the model could be picking up on other concurrent features that, when combined with being overweight, decrease the likelihood of diabetes. For instance, an overweight individual with a robust exercise regimen might be at a lower risk than an individual with obesity and no physical activity. The behavior of other BMI-related features aligns well with general understanding, lending credibility to the model's reliability. We consider that the model's accuracy is not 100%; occasional misjudgments on certain features are understandable.

8.2 Visualize the data, results, models, and patterns

The figures are shown in the7.3, 8.1 and 8.3.

8.3 Interpret the results, models, and patterns

Firstly, the model displays an overall accuracy of 75% on the test data. This is a

relatively good score, suggesting that in 3 out of 4 instances, the model can correctly predict the occurrence or non-occurrence of diabetes. Precision and Recall are essential metrics for gauging the model's effectiveness. Analyzing the results reveals that for class 0 (no diabetes), the model's precision is at 78.84%. This indicates that when the model predicts an individual as non-diabetic, it is accurate 78.84% of the time. The Recall for this class stands at 70.57%, which means it rightly identifies 70.57% of all the non-diabetic individuals from the dataset. Conversely, for class 1 (diagnosed with diabetes), the precision is 73.41%, denoting that when the model foresees someone as diabetic, it gets it right 73.41% of the time. The Recall for this category is considerably higher at 81.10%. And the F1-score, which provides a balanced perspective between Precision and Recall, stands at 74.47% for class 0 and 77.06% for class 1. This score offers further assurance of the model's robust performance, particularly in situations where it's essential to harmonize Precision and Recall.

Moreover, we employed cross-validation to manifest the model's robustness further.

```
from sklearn.model_selection import cross_val_score, StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(clf, X, y, cv=skf)

for fold_num, score in enumerate(scores, 1):
    print(f"Fold {fold_num}: {score:.4f}")
print(f"\nAverage Score: {scores.mean():.4f}")

print(f"Standard deviation: {scores.std():.4f}")
```

As discerned from the five scores, the model's performance is relatively consistent, ranging between 0.7389 and 0.7612. This implies a lesser sensitivity of the model to different subsets of training data, which is a positive indicator. The average score is 0.7480, mirroring the previously mentioned test set accuracy of 0.7584. This reaffirms

the model's stability as its average performance in cross-validation is congruent with its performance on a standalone test set. Our model doesn't exhibit signs of high bias or high variance. High bias typically results in universally low cross-validation scores, while high variance would lead to a significant divergence among scores. In conclusion, these cross-validation results accentuate that the CatBoost classification model exhibits consistent performance on the training data and is anticipated to render similar performance on new, unseen data. The standard deviation represents the amount of variation or dispersion of the set of accuracy values obtained from the cross-validation folds. In simpler terms, it gives us a measure of how much the accuracy fluctuates between different folds of the cross-validation. A smaller standard deviation means that the accuracy values are closer to the mean accuracy of all the folds, suggesting that the model's performance is consistent across different subsets of the data. Conversely, a larger standard deviation indicates more variability in performance across the folds. Our standard deviation of 0.0073 is relatively small, suggesting that the model's performance is consistent across the different cross-validation splits, which indicates that the model is likely not overly sensitive to particular data splits and that its performance is stable.

```
Fold 1: 0.7454
Fold 2: 0.7612
Fold 3: 0.7457
Fold 4: 0.7483
Fold 5: 0.7389

Average Score: 0.7479
Standard deviation: 0.0073
```

We also used the AUC to analyze our result. The AUC (Area Under the Curve) value

measures the ability of a model to distinguish between the positive and negative classes. A value of 1 indicates perfect discrimination, while a value of 0.5 suggests no better discrimination than random guessing. In essence, a higher AUC value implies that the model has a better overall classification performance. It is especially useful in scenarios where the classes are imbalanced or when the cost of false positives and false negatives differ significantly.

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator

binary_evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction", labelCol="diabetes", metricName="are
auc = binary_evaluator.evaluate(predictions_gbt)
print(f"Area Under ROC (AUC): {auc:.4f}")
```

The AUC (Area Under the Curve) value for our model is 0.8312. This indicates that our model has a fairly good ability to distinguish between individuals with and without diabetes. Given our result of 0.8312, it suggests that the model's overall classification performance is quite satisfactory, particularly in discerning the differences between the two classes.

```
Area Under ROC (AUC): 0.8312
```

8.4 Assess and evaluate results, models, and patterns

Building on our in-depth analysis, our model consistently demonstrates robust performance across a spectrum of metrics, bolstering our confidence in its predictive capabilities. Delving into the insights derived from the feature importance (pattern2) and the SHAP values (pattern3), age distinctly stands out as a pivotal determinant in predicting diabetes, with the 'Senior' and 'Older Adults' age groups being particularly susceptible. This underscores a trend suggesting that as one age, the propensity to develop

diabetes escalates. Furthermore, a cleaner medical history, indicating fewer prior health complications, acts as a protective shield, reducing the likelihood of diabetes. The model also sheds light on the significant role of BMI, highlighting that individuals classified as obese are at an amplified risk. Our observations resonate with existing scientific literature and studies, such as the one cited by (myDr, 2020). While the model's efficacy is noteworthy, it's imperative to acknowledge its limitations, given it hasn't attained flawless accuracy. Hence, from a medical standpoint, our findings can be employed to identify groups that are at heightened risk and advocate for them to undergo routine health assessments. Nonetheless, for a conclusive diabetes diagnosis, it remains paramount to depend on comprehensive medical evaluations and diagnostic tests.

8.5 Iterate prior steps (1 – 7) as required

The content of this round can be viewed as an iterative process. In our recent data reduction efforts, we eliminated a greater number of irrelevant features, leading to enhanced model accuracy. During the process of parameter tuning, I made multiple attempts. Initially, I chose to adjust the parameters manually. However, I soon realized that this approach was time-consuming and challenging to evaluate comprehensively. Consequently, I turned to the RandomizedSearchCV method to assist in automatically selecting optimal parameters. Compared to manual tuning, RandomizedSearchCV saves a significant amount of time and explores a broader parameter space. By sampling randomly, it can assess parameter combinations more thoroughly, increasing the likelihood of identifying the best model configuration. Furthermore,

RandomizedSearchCV, combined with cross-validation, ensures robust model evaluations. Employing RandomizedSearchCV over manual parameter adjustments enhances efficiency and better assures model performance optimization.

These iterative efforts enhanced the model's performance and deepened my understanding of the data and features. After each modification, I assessed the model's performance and adjusted my strategy accordingly. Through these continuous endeavors, I garnered a more profound grasp of the entire data mining process and approach, furnishing invaluable experience for my subsequent undertakings.

Reference:

World Health Organization. (n.d.-a). *Second annual gathering of the global group of heads of state and government for the Prevention and control of NCDS*. World Health Organization. https://www.who.int/news-room/events/detail/2023/09/21/default-calendar/second-annual-gathering-of-the-global-group-of-heads-of-state-and-government-for-the-prevention-and-control-of-ncds

Facts & figures. International Diabetes Federation. (2023, August 24). https://idf.org/about-diabetes/diabetes-facts-figures/

Department of Health & Human Services. (2004, October 28). *Diabetes - long-term effects*. Better Health Channel. https://www.betterhealth.vic.gov.au/health/conditionsandtreatments/diabetes-long-term-effects

心脏病会引起糖尿病吗 (no date) 百度一下. Available at: https://m.baidu.com/bh/m/detail/vc_17950940328370966865 (Accessed: 16 September 2023).

(No date) *Classification and diagnosis of diabetes - diabetes in America - NCBI ...* Available at: https://www.ncbi.nlm.nih.gov/books/NBK568014/ (Accessed: 16 September 2023).

Diabetes (2023) *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Diabetes (Accessed: 17 September 2023).

myDr (2020) *Diabetes in older people*, *MyDr.com.au*. Available at: https://mydr.com.au/diabetes/diabetes-in-seniors/#:~:text=The%20rates%20of%20type%202,physical%20activity%20in%20older%20people. (Accessed: 20 September 2023).

Diabetes in Australia (2023) *Diabetes Australia*. Available at: https://www.diabetesaustralia.com.au/about-diabetes/diabetes-in-australia/ (Accessed: 07 October 2023).

Social Identity theory (no date) *Social Identity Theory - an overview | ScienceDirect Topics*. Available at: https://www.sciencedirect.com/topics/psychology/social-identity-theory (Accessed: 07 October 2023).

I acknowledge that the submitted work is my own original work in accordance with the University of Auckland guidelines and policies on academic integrity and copyright. (See: https://www.auckland.ac.nz/en/students/forms-policies-and-guidelines/student-policies-and-guidelines/academic-integrity-copyright.htmlLinks to an external site.).

I also acknowledge that I have appropriate permission to use the data that I have utilised in this project. (For example, if the data belongs to an organisation and the data has not been published in the public domain then the data must be approved by the rights holder.) This includes permission to upload the data file to Canvas. The University of Auckland bears no responsibility for the student's misuse of data.