NHANES (National Health and Nutrition Examination Survey) Data Mining Challenge

Table of contents

1 SIT220 - Data Mining Challenge

Name : Khalisha Hanan Fakhira

Student ID : 224206393

Email : s224206393@deakin.edu.au

1.0.1 Undergraduate SIT220

2 Introduction

This report presents a data-driven analysis to explore key factors associated with diabetes diagnosis using the NHANES (National Health and Nutrition Examination Survey) datasets. The aim is to uncover patterns, evaluate feature significance, and identify influential variables related to diabetes risk through data cleaning, exploration, and modelling.

By performing Exploratory Data Analysis (EDA) and visualising relationships between nutritional intake, body measurements, and glycohemoglobin levels, this report highlights how different health indicators may relate to diabetes occurrence.

A Random Forest model is used as the primary predictive tool to evaluate the accuracy of the findings, ensuring both analytical depth and practical insights.

2.1 Importing Libraries and Datasets

To start, I import essential libraries such as NumPy, Pandas, Bokeh, Scikit-learn, SciPy, and Statsmodels for data manipulation, visualization, and statistical modelling. I then proceed to load the relevant datasets from the NHANES 2017–2023 collection for analysis.

```
# Basic Handling
import pandas as pd
import numpy as np
import panel as on
from IPython.display import Image
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import parallel_coordinates

```

```
# Bokeh Visualisation
from bokeh.plotting import figure, show, output_notebook
from bokeh.models import ColumnDataSource,  MultiLine, Select, HoverTool, DataTable, Table
from bokeh.layouts import row, column
from bokeh.palettes import Category10
from bokeh.models import Whisker
from bokeh.models import Div
from bokeh.plotting import figure, show
from bokeh.transform import dodge
from bokeh.plotting import figure, show
from bokeh.transform import linear_cmap
from bokeh.palettes import Blues256
from bokeh.plotting import figure
from bokeh.io import output_notebook
from bokeh.transform import factor_cmap
from bokeh.transform import transform
from bokeh.palettes import  Blues256
from bokeh.models import ColumnDataSource, Slider, CustomJS

# Statistical Test and build model
from scipy.stats import ttest_ind, pearsonr, chi2_contingency
from scipy.stats import f_oneway
import statsmodels.api as sm
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE
from statsmodels.miscmodels.ordinal_model import OrderedModel
from sklearn.metrics import classification_report, confusion_matrix

#PNG Bokeh
from bokeh.io.export import export_png
import chromedriver_autoinstaller
```

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from bokeh.io.webdriver import create_chromium_webdriver
```

```
# 2017-2020 datasets.
demo_17_20  = pd.read_sas("D:\Year 1\SIT220 Data Wrangling\HD\P_DEMO.XPT", format='xport')
diet_17_20  = pd.read_sas("D:\Year 1\SIT220 Data Wrangling\HD\P_DR1TOT.XPT", format='xport
exam_17_20  = pd.read_sas("D:\Year 1\SIT220 Data Wrangling\HD\P_BMX.XPT", format='xport')
glo_17_20   = pd.read_sas("D:\Year 1\SIT220 Data Wrangling\HD\P_GHB.XPT", format='xport')
diabetes_17_20 = pd.read_sas("D:\Year 1\SIT220 Data Wrangling\HD\P_DIQ.XPT", format='xport

# 2021-2023 datasets.
demo_21_23  = pd.read_sas("D:\Year 1\SIT220 Data Wrangling\HD\DEMO_L.XPT", format='xport')
diet_21_23  = pd.read_sas("D:\Year 1\SIT220 Data Wrangling\HD\DR1TOT_L.XPT", format='xport
exam_21_23  = pd.read_sas("D:\Year 1\SIT220 Data Wrangling\HD\BMX_L.XPT", format='xport')
glo_21_23   = pd.read_sas("D:\Year 1\SIT220 Data Wrangling\HD\GHB_L.XPT", format='xport')
diabetes_21_23 = pd.read_sas("D:\Year 1\SIT220 Data Wrangling\HD\DIQ_L.XPT", format='xport
```

```
<>:2: SyntaxWarning:

invalid escape sequence '\Y'

<>:3: SyntaxWarning:

invalid escape sequence '\Y'

<>:4: SyntaxWarning:

invalid escape sequence '\Y'

<>:5: SyntaxWarning:

invalid escape sequence '\Y'

<>:6: SyntaxWarning:

invalid escape sequence '\Y'

<>:9: SyntaxWarning:

invalid escape sequence '\Y'
```

```
<>:10: SyntaxWarning:

invalid escape sequence '\Y'

<>:11: SyntaxWarning:

invalid escape sequence '\Y'

<>:12: SyntaxWarning:

invalid escape sequence '\Y'

<>:13: SyntaxWarning:

invalid escape sequence '\Y'

<>:2: SyntaxWarning:

invalid escape sequence '\Y'

<>:3: SyntaxWarning:

invalid escape sequence '\Y'

<>:4: SyntaxWarning:

invalid escape sequence '\Y'

<>:5: SyntaxWarning:

invalid escape sequence '\Y'

<>:6: SyntaxWarning:

invalid escape sequence '\Y'

<>:9: SyntaxWarning:

invalid escape sequence '\Y'

<>:10: SyntaxWarning:
```

```
invalid escape sequence '\Y'

<>:11: SyntaxWarning:

invalid escape sequence '\Y'

<>:12: SyntaxWarning:

invalid escape sequence '\Y'

<>:13: SyntaxWarning:

invalid escape sequence '\Y'

C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\2865258276.py:2: SyntaxWarning:

invalid escape sequence '\Y'

C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\2865258276.py:3: SyntaxWarning:

invalid escape sequence '\Y'

C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\2865258276.py:4: SyntaxWarning:

invalid escape sequence '\Y'

C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\2865258276.py:5: SyntaxWarning:

invalid escape sequence '\Y'

C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\2865258276.py:6: SyntaxWarning:

invalid escape sequence '\Y'

C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\2865258276.py:9: SyntaxWarning:

invalid escape sequence '\Y'

C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\2865258276.py:10: SyntaxWarning:

invalid escape sequence '\Y'

C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\2865258276.py:11: SyntaxWarning:
```

```
invalid escape sequence '\Y'

C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\2865258276.py:12: SyntaxWarning:

invalid escape sequence '\Y'

C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\2865258276.py:13: SyntaxWarning:

invalid escape sequence '\Y'
```

2.2 Dataset Merging and Integration

Next, I combine multiple NHANES datasets from different domains demographics, diet, examination, global health, and diabetes spanning the years 2017 to 2023. These datasets are merged on the common identifier SEQN using inner joins to retain only participants with complete records across all domains. The resulting merged dataset contains 17608 rows and 259 columns for subsequent analysis.

```
#Combine the datasets
demo_all  = pd.concat([demo_17_20, demo_21_23], ignore_index=True)
diet_all  = pd.concat([diet_17_20, diet_21_23], ignore_index=True)
exam_all  = pd.concat([exam_17_20, exam_21_23], ignore_index=True)
glo_all   = pd.concat([glo_17_20, glo_21_23], ignore_index=True)
diabetes_all = pd.concat([diabetes_17_20, diabetes_21_23], ignore_index=True)

#Merge the datasets
# Merge them using SEQN as the key
merged_df = demo_all.merge(diet_all, on='SEQN', how='inner') \
                    .merge(exam_all, on='SEQN', how='inner') \
                    .merge(glo_all, on='SEQN', how='inner') \
                    .merge(diabetes_all, on='SEQN', how='inner')

print("Merged dataset shape:", merged_df.shape)
merged_df.head()
```

```
Merged dataset shape: (17608, 259)
```

	SEQN	SDDSRVYR	RIDSTATR	RIAGENDR	RIDAGEYR	RIDAGEMN	RIDRETH1	RIDR
0	109264.0	66.0	2.0	2.0	13.0	NaN	1.0	1.0
1	109266.0	66.0	2.0	2.0	29.0	NaN	5.0	6.0
2	109271.0	66.0	2.0	1.0	49.0	NaN	3.0	3.0
3	109273.0	66.0	2.0	1.0	36.0	NaN	3.0	3.0
4	109274.0	66.0	2.0	1.0	68.0	NaN	5.0	7.0

2.3 Variable Selection and Label Decoding

To focus the analysis on relevant attributes, I first selected a subset of variables critical to the study. These include demographic details (e.g., age, gender, education), dietary intake, and biomarkers such as Glycohemoglobin. Once selected, I applied a renaming dictionary to convert technical column names into more interpretable labels for ease of reference throughout the report. This enhances both readability and interpretability for broader audiences.

```
target_cols = [
    "DIQ010", "RIAGENDR", "RIDAGEYR", "RIDRETH1", "DMDEDUC2", "INDFMPIR",
    "DR1TKCAL", "DR1TPROT", "DR1TTFAT", "DR1TCARB", "DR1TSUGR", "DR1TFIBE", "DR1TSODI",
    "BMXBMI", "BMXWT", "BMXHT", "BMXWAIST", "LBXGLU", "LBXGH"
]

rename_dict = {
    "DIQ010": "Told_to_have_Diabetes",
    "RIAGENDR": "Gender",
    "RIDAGEYR": "Age",
    "RIDRETH1": "Race_Ethnicity",
    "DMDEDUC2": "Education_Level",
    "INDFMPIR": "Poverty_Income_Ratio",
    "DR1TKCAL": "Energy_Intake",
    "DR1TPROT": "Protein",
    "DR1TTFAT": "Total_Fat",
    "DR1TCARB": "Carbohydrates",
    "DR1TSUGR": "Total_Sugars",
    "DR1TFIBE": "Dietary_Fiber",
    "DR1TSODI": "Sodium",
    "BMXBMI": "BMI",
    "BMXWT": "Weight_kg",
    "BMXHT": "Height_cm",
    "BMXWAIST": "Waist_Circumference",
    "LBXGH": "Glycohemoglobin",
```

```
}

available_cols = [col for col in target_cols if col in merged_df.columns]
filtered_df = merged_df[available_cols]

available_renames = {k: v for k, v in rename_dict.items() if k in filtered_df.columns}
filtered_df.rename(columns=available_renames, inplace=True)

print("Decoded-columns Dataframe:")
display(filtered_df.head())
print("Shape:", filtered_df.shape)
```

```
Decoded-columns Dataframe:

C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\3523576568.py:32: SettingWithCopyWarning

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
```

	Told_to_have_Diabetes	Gender	Age	Race_Ethnicity	Education_Level	Poverty_Income_Ratio
0	2.0	2.0	13.0	1.0	NaN	0.83
1	2.0	2.0	29.0	5.0	5.0	5.00
2	2.0	1.0	49.0	3.0	2.0	NaN
3	2.0	1.0	36.0	3.0	4.0	0.83
4	1.0	1.0	68.0	5.0	4.0	1.20

```
Shape: (17608, 18)
```

2.4 Missing Values Overview

Before performing any statistical operations, I examined the dataset for missing values. Several columns, such as Education_Level, Carbohydrates, and other dietary intake features, were found to contain missing values ranging from approximately 13% to 17%. Fortunately, key categorical features like Gender, Race_Ethnicity, and the target variable Told_to_have_Diabetes had no missing data. Identifying this helps guide the imputation strategy applied in the next step of data wrangling.

```
missing_values = filtered_df.isnull().sum()
missing_percent = (missing_values / len(filtered_df)) * 100

missing_df = pd.DataFrame({'Missing Values': missing_values, 'Percent (%)': missing_percen
display(missing_df.sort_values(by='Percent (%)', ascending=False))
```

	Missing Values	Percent (%)
Education_Level	3000	17.037710
Total_Fat	2605	14.794412
Carbohydrates	2605	14.794412
Sodium	2605	14.794412
Protein	2605	14.794412
Energy_Intake	2605	14.794412
Dietary_Fiber	2605	14.794412
Total_Sugars	2605	14.794412
Poverty_Income_Ratio	2339	13.283735
Glycohemoglobin	1156	6.565198
Waist_Circumference	919	5.219219
BMI	303	1.720809
Weight_kg	265	1.504998
Height_cm	260	1.476602
Gender	0	0.000000
Told_to_have_Diabetes	0	0.000000
Race_Ethnicity	0	0.000000
Age	0	0.000000

2.5 Handling Missing Values

To address the missing values, I began by removing the Education_Level column, which had the highest proportion of null entries and was deemed less critical for this analysis. Following that, I selected only the numeric columns from the dataset to prepare them for imputation.

Using SimpleImputer with a mean strategy, I filled in the missing values for these numeric variables. This ensures the continuity of data while maintaining overall statistical integrity.

```
filtered_df.drop(columns=["Education_Level"], inplace=True)
```

```
C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\2630585956.py:1: SettingWithCopyWarning
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
```

```
# Select only numeric columns
num_cols = filtered_df.select_dtypes(include=["float64", "int64"]).columns

# Create imputer
imputer = SimpleImputer(strategy="mean")

# Apply imputation
filtered_df[num_cols] = imputer.fit_transform(filtered_df[num_cols])
```

```
C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\699305930.py:8: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
```

2.6 Validating Data Integrity

Following imputation, I verified that no missing values remained within the dataset, confirming that the data was now complete and ready for modeling. Subsequently, I reviewed the data types to ensure consistency and found that all selected variables were of numeric type, which is essential for most machine learning algorithms.

To finalise this step, I checked for and removed any duplicate rows. The dataset contained no such duplicates, ensuring that the data used in the next stages of analysis was clean and free from redundancy.

```
print("Still missing:", filtered_df.isnull().sum().sum())
```

```
Still missing: 0
```

```
filtered_df.dtypes
```

```
Told_to_have_Diabetes    float64
Gender                   float64
Age                      float64
Race_Ethnicity           float64
Poverty_Income_Ratio     float64
Energy_Intake            float64
Protein                  float64
Total_Fat                float64
Carbohydrates            float64
Total_Sugars             float64
Dietary_Fiber            float64
Sodium                   float64
BMI                      float64
Weight_kg                float64
Height_cm                float64
Waist_Circumference      float64
Glycohemoglobin          float64
dtype: object
```

```python
duplicates = filtered_df.duplicated().sum()
print("Duplicate rows:", duplicates)

# Remove them if any
filtered_df.drop_duplicates(inplace=True)
```

```
Duplicate rows: 0
```

```
C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\1394959270.py:5: SettingWithCopyWarning

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
```

2.7 Summary Statistics

Next, to gain a better understanding of the distribution of our variables, I generated summary statistics using the .describe() function. This provided a snapshot of the central tendency and dispersion for each feature, including the mean, standard deviation, minimum, and maximum values.

This step was particularly useful for identifying potential outliers or inconsistencies, and for confirming that the data is appropriately scaled for further analysis. I also reset the index to maintain a clean and sequential structure for the rows after the wrangling process.

```
filtered_df.describe()
```

	Told_to_have_Diabetes	Gender	Age	Race_Ethnicity	Poverty_Income_Ratio
count	17608.000000	17608.000000	17608.000000	17608.000000	1.760800e+04
mean	1.905157	1.524250	45.910495	3.199171	2.639578e+00
std	0.405254	0.499426	21.091800	1.144685	1.534644e+00
min	1.000000	1.000000	12.000000	1.000000	5.397605e-79
25%	2.000000	1.000000	27.000000	3.000000	1.330000e+00
50%	2.000000	2.000000	47.000000	3.000000	2.639578e+00
75%	2.000000	2.000000	64.000000	4.000000	3.960000e+00
max	9.000000	2.000000	80.000000	5.000000	5.000000e+00

```
filtered_df.reset_index(drop=True, inplace=True)
```

2.8 BMI Distribution

The histogram below illustrates the distribution of Body Mass Index (BMI) across the selected NHANES participants. An interactive slider has been integrated to allow dynamic filtering by maximum BMI value, enhancing interpretability.

The plot demonstrates a right-skewed distribution, with the majority of individuals concentrated in the 20–35 BMI range. This reflects common population-level patterns and highlights the prevalence of overweight and obesity within the dataset.

The interactive capabilities enable a more focused examination of specific BMI intervals, which can be useful when comparing subgroups or identifying potential outliers.

```
output_notebook()

# Prepare BMI values
bmi_values = filtered_df["BMI"].dropna()
hist, edges = np.histogram(bmi_values, bins=30)

# Create ColumnDataSource
source = ColumnDataSource(data=dict(
    top=hist,
```

```
    left=edges[:-1],
    right=edges[1:]
))

# Create histogram figure
p = figure(title="BMI Distribution (Interactive)",
           x_axis_label='BMI',
           y_axis_label='Frequency',
           tools="pan,wheel_zoom,box_zoom,reset")

quad = p.quad(top='top', bottom=0, left='left', right='right', source=source,
              fill_color="skyblue", line_color="white", alpha=0.8)

# Add hover (optional but nice)
from bokeh.models import HoverTool
hover = HoverTool(tooltips=[("Range", "@left - @right"), ("Count", "@top")])
p.add_tools(hover)

# Create slider
slider = Slider(start=10, end=50, value=50, step=1, title="Max BMI")

# JS callback to update histogram dynamically
callback = CustomJS(args=dict(source=source,
                              original=bmi_values,
                              slider=slider), code="""
    const max_bmi = slider.value;
    const data = source.data;

    // Get original BMI array and filter
    let values = Array.from(original).filter(v => v <= max_bmi);

    // Compute histogram
    let bins = 30;
    let minVal = Math.min(...values);
    let maxVal = Math.max(...values);
    let step = (maxVal - minVal) / bins;

    let counts = new Array(bins).fill(0);
    for (let v of values) {
        let idx = Math.min(Math.floor((v - minVal) / step), bins - 1);
        counts[idx]++;
```

```
    }

    let left = [], right = [];
    for (let i = 0; i < bins; i++) {
        left.push(minVal + i * step);
        right.push(minVal + (i + 1) * step);
    }

    data.top = counts;
    data.left = left;
    data.right = right;
    source.change.emit();
""")

slider.js_on_change("value", callback)

# Show interactive layout
show(column(p, slider))

```

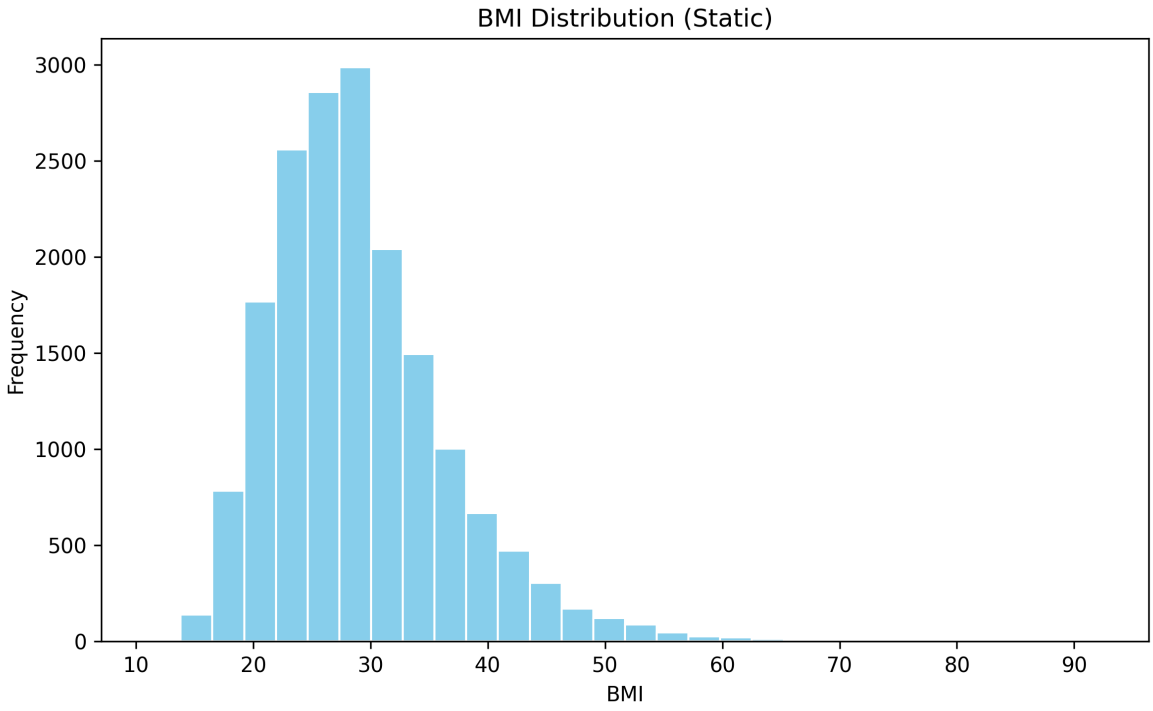
```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_lo
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_ex
```

```
plt.figure(figsize=(8,5))
plt.hist(bmi_values, bins=30, color="skyblue", edgecolor="white")
plt.title("BMI Distribution (Static)")
plt.xlabel("BMI")
plt.ylabel("Frequency")
plt.tight_layout()
plt.savefig("bmi_hist_static.png")
plt.close()

Image("bmi_hist_static.png")
```



2.9 Diabetes Prevalence

This interactive bar chart illustrates the distribution of individuals who have been told they have diabetes versus those who have not. The x-axis represents the diabetes diagnosis status (Yes or No), while the y-axis shows the number of individuals in each group.

The chart is built using the Bokeh library, which allows for interactivity such as hover tooltips and zoom controls. When hovering over each bar, users can view the exact count of individuals per category, enabling a clearer and more precise comparison.

From the visualisation, it is evident that the majority of individuals in the dataset reported not having been told they have diabetes, with a significantly smaller portion indicating a positive diagnosis. This disparity provides insight into the prevalence of diabetes within the population studied and can inform further exploration of potential risk factors.

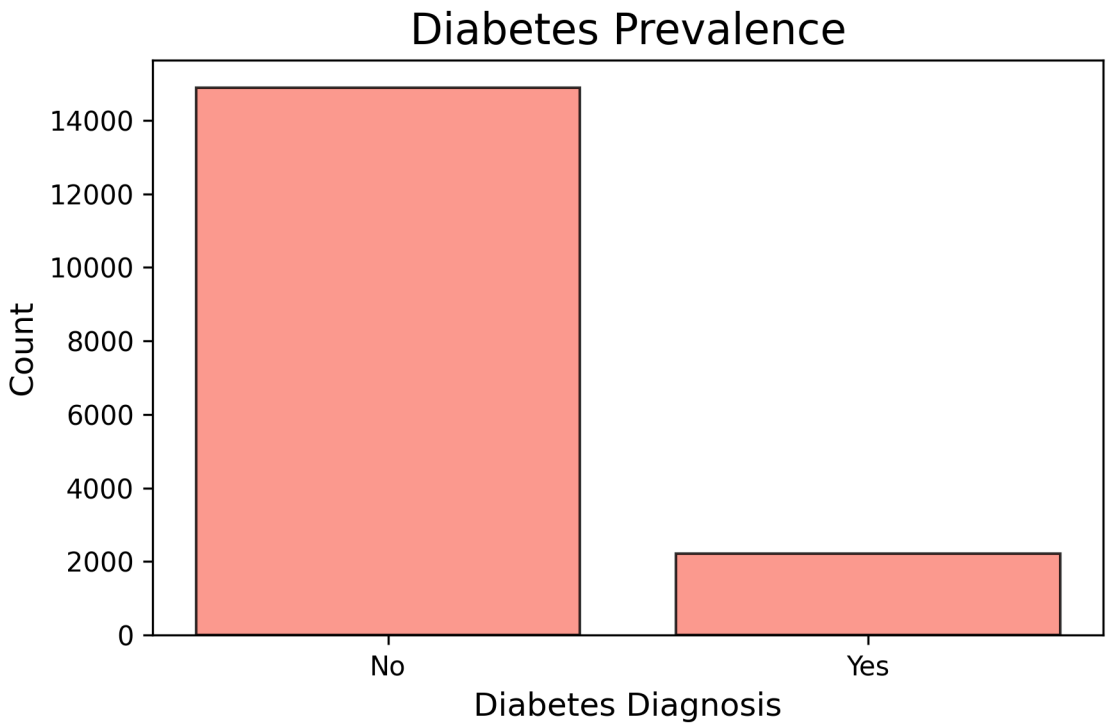
```
output_notebook()

diabetes_counts = filtered_df['Told_to_have_Diabetes'].value_counts().sort_index()
labels = ['No', 'Yes']
counts = [diabetes_counts.get(2, 0), diabetes_counts.get(1, 0)]  # NHANES codes: 1=Yes, 2=
```

```
source = ColumnDataSource(data=dict(
    status=labels,
    count=counts
))

p = figure(x_range=labels, title="Diabetes Prevalence",
           y_axis_label='Count', x_axis_label='Diabetes Diagnosis',
           tools="pan,wheel_zoom,box_zoom,reset,save")

p.vbar(x='status', top='count', width=0.5, source=source,
       fill_color="salmon", line_color="black", alpha=0.8)

hover = HoverTool(tooltips=[("Status", "@status"), ("Count", "@count")])
p.add_tools(hover)

p.title.text_font_size = "16pt"
p.xaxis.axis_label_text_font_size = "12pt"
p.yaxis.axis_label_text_font_size = "12pt"

# Show plot
show(p)
```

```
Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_lo

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_ex
```

```
diabetes_counts = filtered_df['Told_to_have_Diabetes'].value_counts().sort_index()
labels = ['No', 'Yes']
counts = [diabetes_counts.get(2, 0), diabetes_counts.get(1, 0)]

plt.figure(figsize=(6, 4))
plt.bar(labels, counts, color='salmon', edgecolor='black', alpha=0.8)
plt.title("Diabetes Prevalence", fontsize=16)
plt.xlabel("Diabetes Diagnosis", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.tight_layout()
```

```
plt.savefig("diabetes_prevalence_static.png")  # Save Diabetes plot
plt.close()

Image("diabetes_prevalence_static.png")
```



2.10 BMI Distribution by Diabetes Diagnosis

This visualisation presents a box plot comparing the Body Mass Index (BMI) distributions of individuals diagnosed with and without diabetes. The x-axis categorises respondents based on their diabetes diagnosis status ("Yes" or "No"), while the y-axis represents their respective BMI values.

Each box plot reflects the interquartile range (IQR) of BMI values within each group, with the lower and upper bounds indicating the 25th and 75th percentiles, respectively. The vertical lines ("whiskers") extend to 1.5 times the IQR, capturing the spread of most BMI values, while the central black segment denotes the median BMI.

This comparison enables a more nuanced understanding of how BMI levels differ between diabetic and non-diabetic populations. It also offers a clearer sense of central tendency and variability in each group.

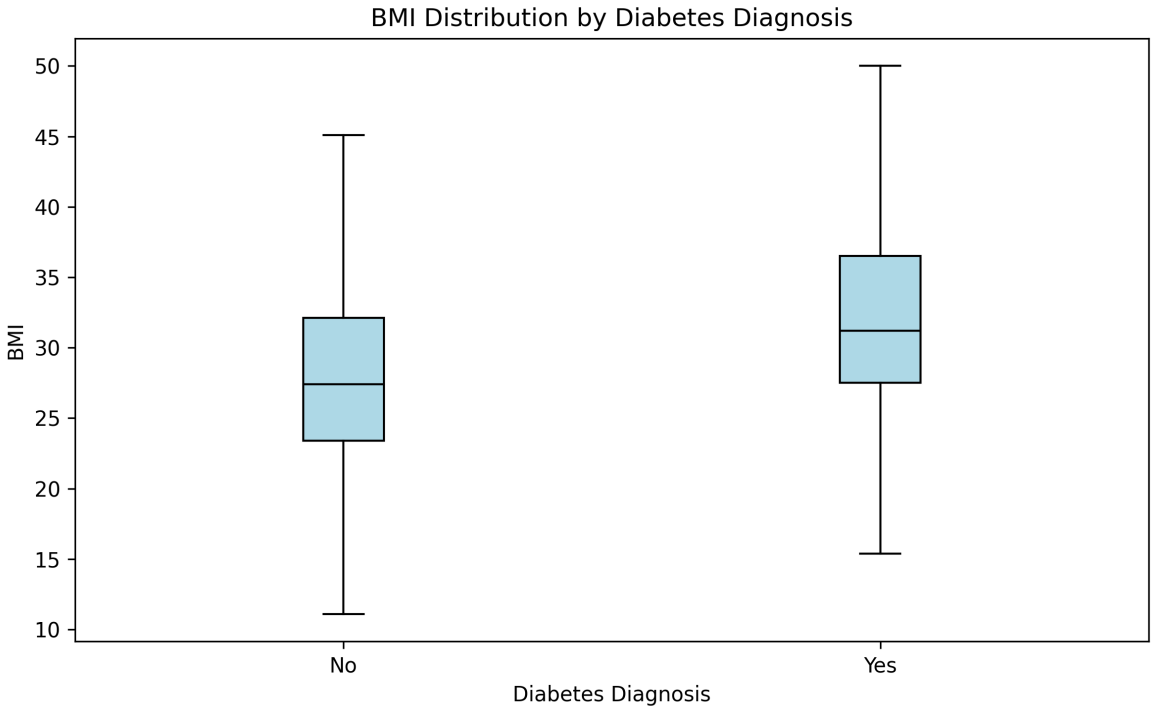
```
output_notebook()

filtered_df['Diabetes_Label'] = filtered_df['Told_to_have_Diabetes'].replace({1: 'Yes', 2:
filtered_df = filtered_df[filtered_df['Diabetes_Label'].isin(['Yes', 'No'])]

grouped = filtered_df.groupby('Diabetes_Label')['BMI']
stats_df = grouped.describe()[['25%', '50%', '75%']]
stats_df.columns = ['q1', 'q2', 'q3']
stats_df['iqr'] = stats_df['q3'] - stats_df['q1']
stats_df['upper'] = stats_df['q3'] + 1.5 * stats_df['iqr']
stats_df['lower'] = stats_df['q1'] - 1.5 * stats_df['iqr']
stats_df = stats_df.reset_index()

source = ColumnDataSource(stats_df)

p = figure(x_range=['No', 'Yes'], title="BMI Distribution by Diabetes Diagnosis",
           y_axis_label='BMI', x_axis_label='Diabetes Diagnosis', height=400,
           tools="pan,wheel_zoom,box_zoom,reset,save", toolbar_location="right")


p.vbar(x='Diabetes_Label', width=0.6, bottom='q1', top='q3', source=source,
       fill_color="lightblue", line_color="black")
p.segment(x0='Diabetes_Label', y0='q2', x1='Diabetes_Label', y1='q2', source=source,
          line_color="black", line_width=2)
p.add_layout(Whisker(source=source, base='Diabetes_Label', upper='upper', lower='lower'))

show(p)
```

```
Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_lo

C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\2012942645.py:3: SettingWithCopyWarning


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide

Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_ex
```

```python
grouped_data = [filtered_df[filtered_df['Diabetes_Label'] == label]['BMI'].dropna() for la

plt.figure(figsize=(8, 5))
plt.boxplot(grouped_data,
            labels=['No', 'Yes'],
            patch_artist=True,
            showfliers=False,
            boxprops=dict(facecolor='lightblue', color='black'),
            medianprops=dict(color='black'),
            whiskerprops=dict(color='black'),
            capprops=dict(color='black'))

plt.title("BMI Distribution by Diabetes Diagnosis")
plt.xlabel("Diabetes Diagnosis")
plt.ylabel("BMI")
plt.tight_layout()
plt.savefig("bmi_by_diabetes_static_nofliers.png")
plt.close()

Image("bmi_by_diabetes_static_nofliers.png")
```

```
C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\336479544.py:4: MatplotlibDeprecationWa

The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; supp
```



2.11 Age Summary by Diabetes Diagnosis

After that we made a table that displays descriptive statistics for age, grouped by diabetes diagnosis status. It provides insights into the age distribution among individuals with and without diabetes.

We can see that: - Individuals without diabetes have an average age of approximately 43.1 years, with the interquartile range (25th to 75th percentile) between 24.0 and 61.0 years. - In contrast, individuals with diabetes have a notably higher average age of approximately 62.5 years, with an interquartile range between 55.0 and 72.0 years.

The clear difference in central tendency and spread suggests that age is a significant factor associated with diabetes prevalence. Older individuals are considerably more likely to be diagnosed with diabetes compared to younger ones, which aligns with well-established medical findings.

```
age_summary = filtered_df.groupby("Diabetes_Label")["Age"].describe()
display(age_summary)
```

Diabetes_Label	count	mean	std	min	25%	50%	75%	max
No	14894.0	43.076138	20.924554	12.0	24.0	42.0	61.0	80.0
Yes	2213.0	62.472210	12.992931	12.0	55.0	64.0	72.0	80.0

2.12 Nutrient Intake Summary by Diabetes Diagnosis

To better understand dietary patterns across diabetes statuses, I aggregated the nutrient intake variables including Protein, Carbohydrates, Total Fat, Total Sugars, and Dietary Fiber by diabetes diagnosis groups (Yes/No). For each group, I computed the mean, standard deviation, minimum, and maximum values of these nutrients.

We can see that: - Protein, Carbohydrates, and Total Fat show slightly higher average intake among individuals without diabetes. - Total Sugars and Dietary Fiber show comparable mean values between groups, although the non-diabetic group has a slightly higher average sugar and fiber intake.

The standard deviations across all nutrients indicate considerable variability in individual intake levels, reflecting diverse dietary behaviors in both groups. This analysis provides useful insight into how nutrient consumption patterns differ by diabetes status, which may inform further investigation into dietary risk factors and interventions.

```
nutrients = ["Protein", "Carbohydrates", "Total_Fat", "Total_Sugars", "Dietary_Fiber"]
nutrient_summary = filtered_df.groupby("Diabetes_Label")[nutrients].agg(["mean", "std", "m
display(nutrient_summary.round(1))
```

	Protein				Carbohydrates				Total_Fat				Tota
Diabetes_Label	mean	std	min	max	mean	std	min	max	mean	std	min	max	mean
No	76.8	39.0	0.0	545.2	242.3	116.7	0.0	1586.2	85.3	45.0	0.0	568.0	104.7
Yes	74.6	35.5	0.0	314.2	218.3	100.7	0.0	1134.1	83.4	43.5	0.0	429.2	90.8

2.13 Feature Correlation Heatmap

After that, I tried to uncover relationships among the selected features. So, I constructed a correlation heatmap displaying Pearson correlation coefficients between each pair of continuous variables. This matrix provides a comprehensive overview of the linear association between features.

From the heatmap, we can see that: - Weight, BMI, and Waist Circumference show strong positive correlations with one another, which is expected due to their shared relevance in body composition. - Glycohemoglobin (a diabetes indicator) appears to correlate moderately with Waist Circumference and BMI, suggesting potential association with obesity-related metrics. - Most dietary variables, such as Carbohydrates, Total Sugars, and Total Fat have low to moderate correlations between themselves, indicating some overlap but not redundancy. - Age shows relatively low correlation with other features, implying it behaves independently within this dataset.

This correlation map helps identify risks of multicollinearity in modeling and informs feature selection strategies.

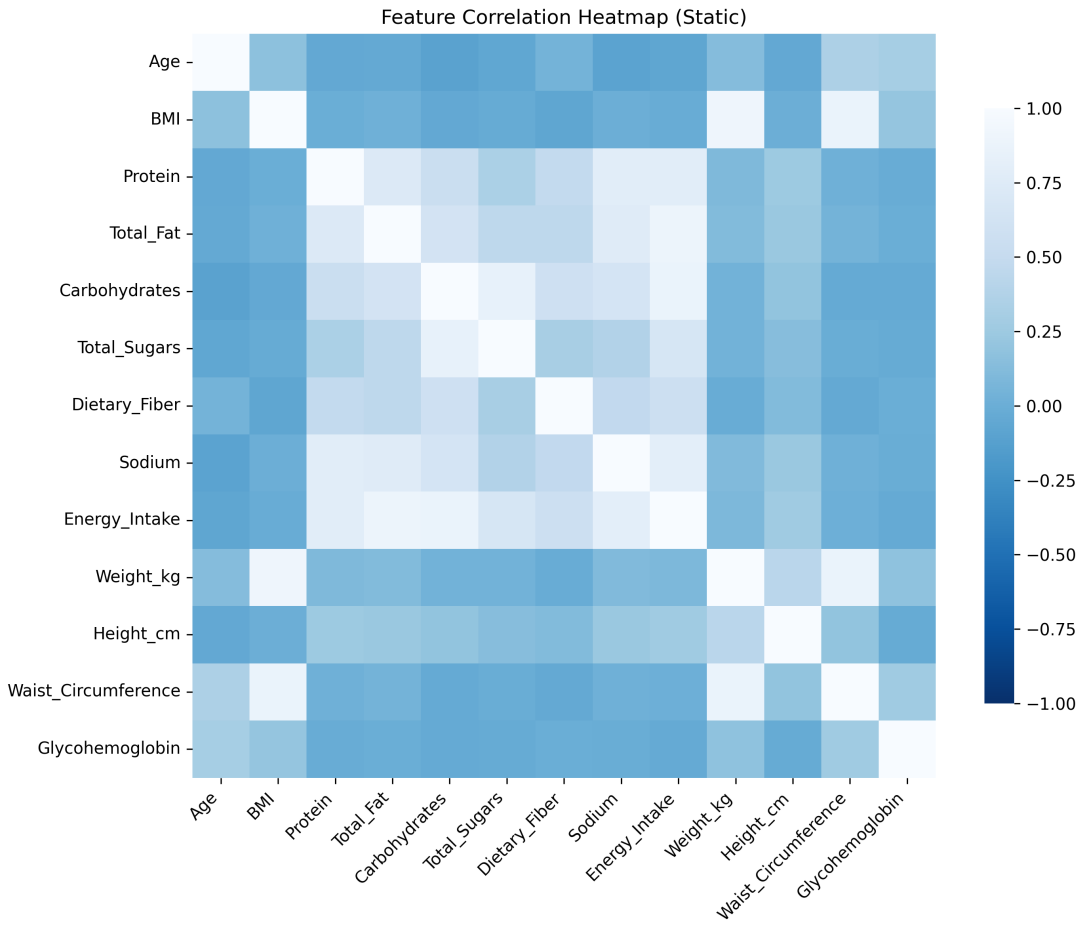
```
num_cols = [
    "Age", "BMI", "Protein", "Total_Fat", "Carbohydrates", "Total_Sugars", "Dietary_Fiber"
    "Sodium", "Energy_Intake", "Weight_kg", "Height_cm", "Waist_Circumference", "Glycohemo
]
filtered_num = filtered_df[num_cols]


corr_matrix = filtered_num.corr().round(2)
corr_data = corr_matrix.stack().reset_index()
corr_data.columns = ["x", "y", "value"]
source = ColumnDataSource(corr_data)

mapper = LinearColorMapper(palette=Blues256, low=-1, high=1)

p = figure(title="Feature Correlation Heatmap",
           x_range=list(corr_matrix.columns), y_range=list(reversed(corr_matrix.columns)),
           x_axis_location="above", width=800, height=800,
           toolbar_location='right', tools="hover,save,reset", tooltips=[("Var1", "@x"), (

p.rect(x="x", y="y", width=1, height=1, source=source,
       fill_color=transform('value', mapper), line_color=None)

color_bar = ColorBar(color_mapper=mapper, major_label_text_font_size="10px",
                     ticker=BasicTicker(desired_num_ticks=10),
                     formatter=PrintfTickFormatter(format="%.2f"),
                     label_standoff=8, border_line_color=None, location=(0, 0))

p.add_layout(color_bar, 'right')
p.xaxis.major_label_orientation = np.pi/4

```

```
show(p)
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_ex
```

```
num_cols = [
    "Age", "BMI", "Protein", "Total_Fat", "Carbohydrates", "Total_Sugars", "Dietary_Fiber"
    "Sodium", "Energy_Intake", "Weight_kg", "Height_cm", "Waist_Circumference", "Glycohemo
]
filtered_num = filtered_df[num_cols]
corr_matrix = filtered_num.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, cmap='Blues_r', vmin=-1, vmax=1, square=True, cbar_kws={"shrink":
plt.title("Feature Correlation Heatmap (Static)")
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.savefig("correlation_heatmap_static.png")
plt.close()
Image("correlation_heatmap_static.png")
```



2.14 Glycohemoglobin (%) by Diabetes Diagnosis

To assess how glycohemoglobin levels vary across diabetes status, I created a box plot comparing the distributions of glycohemoglobin (%) for individuals with and without diabetes. Glycohemoglobin, also known as HbA1c, reflects average blood glucose levels over the past three months and is a key indicator used in diagnosing and monitoring diabetes.

From the visualisation: - Individuals diagnosed with diabetes tend to exhibit significantly higher glycohemoglobin levels, with a visibly elevated median and broader range. - The non-diabetic group has a more compact distribution centered below 6%, indicating normal to borderline glycemic control. - The wider interquartile range and presence of high outliers among the diabetic group highlight substantial variability in glycemic control.

This plot visually reinforces the role of glycohemoglobin as a reliable biomarker distinguishing between diabetic and non-diabetic individuals, aligning with clinical expectations.

```
filtered_df["Diabetes_Label"] = filtered_df["Told_to_have_Diabetes"].replace({1: "Yes", 2:

grouped = filtered_df.groupby("Diabetes_Label")["Glycohemoglobin"].describe()
stats_df = grouped[["25%", "50%", "75%"]]
stats_df["iqr"] = stats_df["75%"] - stats_df["25%"]
stats_df["upper"] = stats_df["75%"] + 1.5 * stats_df["iqr"]
stats_df["lower"] = stats_df["25%"] - 1.5 * stats_df["iqr"]
stats_df = stats_df.reset_index()

source = ColumnDataSource(stats_df)

# Plot box
p = figure(x_range=stats_df["Diabetes_Label"], title="Glycohemoglobin (%) by Diabetes Diag
           height=400, tools="pan,wheel_zoom,box_zoom,reset,save", toolbar_location="right
           y_axis_label="Glycohemoglobin (%)", x_axis_label="Diabetes Status")


p.vbar(x='Diabetes_Label', width=0.6, bottom='25%', top='75%', source=source,
       fill_color=factor_cmap('Diabetes_Label', palette=["#aed6f1", "#f5b7b1"], factors=["
       line_color="black")

# Draw median and whiskers
p.segment(x0='Diabetes_Label', y0='upper', x1='Diabetes_Label', y1='75%', source=source, l
p.segment(x0='Diabetes_Label', y0='25%', x1='Diabetes_Label', y1='lower', source=source, l
p.segment(x0='Diabetes_Label', y0='50%', x1='Diabetes_Label', y1='50%', source=source, lin

p.add_layout(Whisker(source=source, base="Diabetes_Label", upper="upper", lower="lower"))

p.xaxis.axis_label_text_font_size = "12pt"
p.yaxis.axis_label_text_font_size = "12pt"

show(p)
```

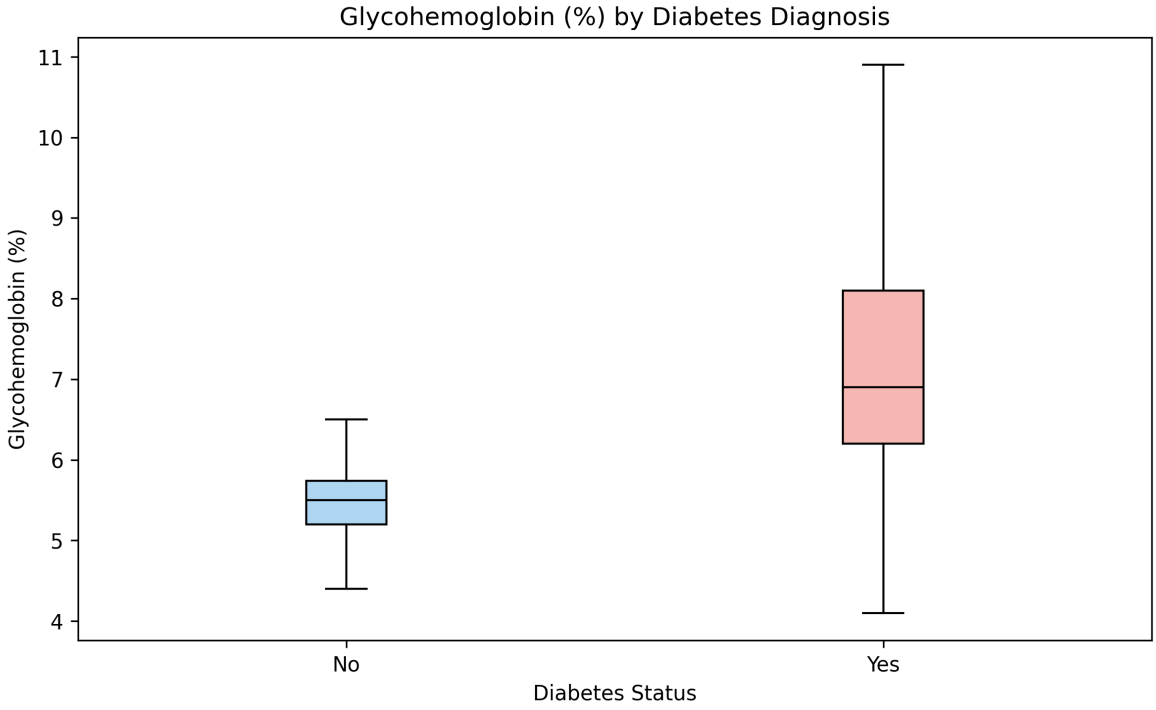
```
C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\678607615.py:5: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide

C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\678607615.py:6: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid

C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\678607615.py:7: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_ex
```

```
filtered_df["Diabetes_Label"] = filtered_df["Told_to_have_Diabetes"].replace({1: "Yes", 2:
filtered_df = filtered_df[filtered_df["Diabetes_Label"].isin(["Yes", "No"])]

grouped_data = [filtered_df[filtered_df["Diabetes_Label"] == label]["Glycohemoglobin"].dro
                for label in ["No", "Yes"]]

plt.figure(figsize=(8, 5))
box = plt.boxplot(grouped_data,
                  labels=["No", "Yes"],
                  patch_artist=True,
                  showfliers=False,
                  boxprops=dict(color="black"),
                  medianprops=dict(color="black"),
                  whiskerprops=dict(color="black"),
                  capprops=dict(color="black"))


colors = ["#aed6f1", "#f5b7b1"]
for patch, color in zip(box["boxes"], colors):
    patch.set_facecolor(color)

plt.title("Glycohemoglobin (%) by Diabetes Diagnosis")
```

```
plt.xlabel("Diabetes Status")
plt.ylabel("Glycohemoglobin (%)")
plt.tight_layout()
plt.savefig("glycohemoglobin_diabetes_static.png")
plt.close()


Image("glycohemoglobin_diabetes_static.png")
```

```
C:\Users\khalisha\AppData\Local\Temp\ipykernel_14184\3062432308.py:8: MatplotlibDeprecationWa

The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; supp
```



2.15 Waist Circumference vs BMI by Diabetes Status

This scatter plot explores the relationship between Waist Circumference (cm) and Body Mass Index (BMI) across individuals with and without diabetes. Each point represents an individual, coloured by their diabetes status: blue for non-diabetic and orange for diabetic.

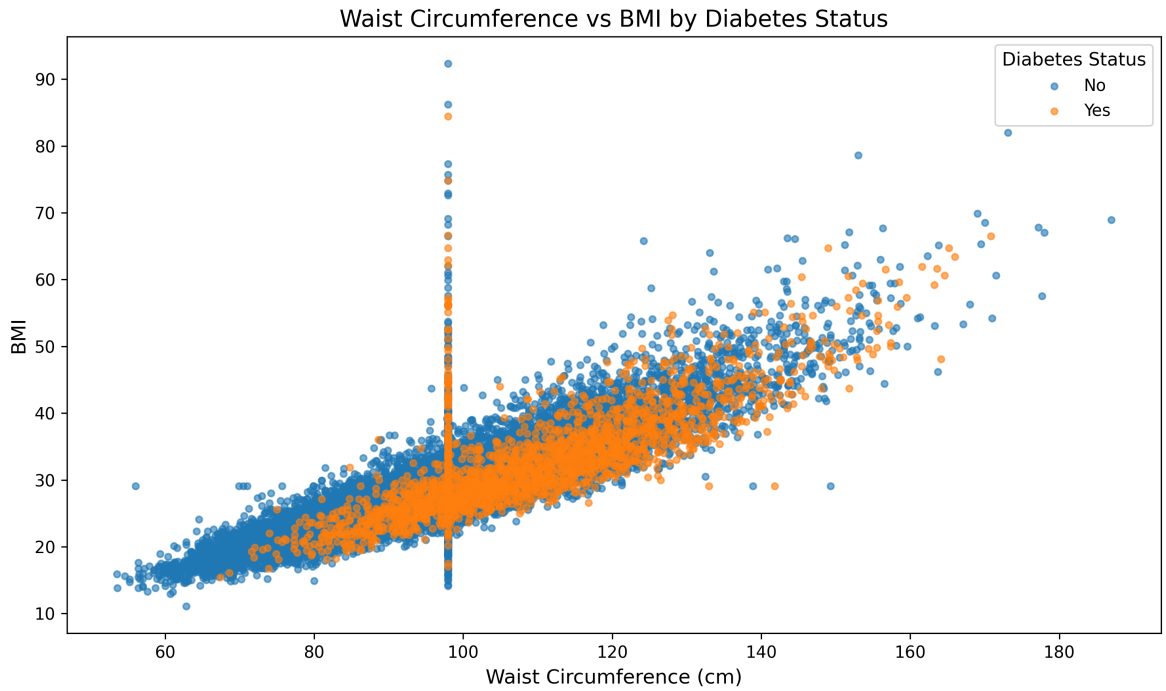
A strong positive correlation is visually evident as waist circumference increases, BMI tends to rise accordingly. This trend holds across both diabetes groups, although the diabetic group appears slightly more concentrated in higher ranges of both BMI and waist circumference.

The visual suggests that central obesity, indicated by greater waist circumference, is a notable factor associated with increased BMI regardless of diabetes status. However, the clustering of diabetic individuals in the upper quadrant implies that higher adiposity may play a role in the development or presence of diabetes, warranting further clinical investigation.

```
filtered_df["Diabetes_Label"] = filtered_df["Told_to_have_Diabetes"].replace({1: "Yes", 2:

source = ColumnDataSource(filtered_df)

p = figure(title="Waist Circumference vs BMI by Diabetes Status",
           x_axis_label="Waist Circumference (cm)",
           y_axis_label="BMI",
           tools="pan,wheel_zoom,box_zoom,reset,hover,save",
           height=400)
p.scatter(x='Waist_Circumference', y='BMI',
          source=source,
          color=factor_cmap('Diabetes_Label', palette=Category10[3], factors=["No", "Yes"]
          legend_field='Diabetes_Label',
          size=6, alpha=0.6)

p.hover.tooltips = [
    ("BMI", "@BMI{0.0}"),
    ("Waist Circumference", "@Waist_Circumference{0.0} cm"),
    ("Diabetes", "@Diabetes_Label")
]


p.legend.title = "Diabetes Status"
p.legend.location = "top_left"
p.legend.label_text_font_size = "10pt"
p.xaxis.axis_label_text_font_size = "12pt"
p.yaxis.axis_label_text_font_size = "12pt"

show(p)
```

```
Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_e
```

```
filtered_df["Diabetes_Label"] = filtered_df["Told_to_have_Diabetes"].replace({1: "Yes", 2:
filtered_df = filtered_df[filtered_df["Diabetes_Label"].isin(["Yes", "No"])]


plt.figure(figsize=(10, 6))

colors = {"No": "#1f77b4", "Yes": "#ff7f0e"}
for label in ["No", "Yes"]:
    subset = filtered_df[filtered_df["Diabetes_Label"] == label]
    plt.scatter(subset["Waist_Circumference"], subset["BMI"],
                c=colors[label], label=label, alpha=0.6, s=15)

plt.title("Waist Circumference vs BMI by Diabetes Status", fontsize=14)
plt.xlabel("Waist Circumference (cm)", fontsize=12)
plt.ylabel("BMI", fontsize=12)
plt.legend(title="Diabetes Status", title_fontsize=11, fontsize=10)
plt.tight_layout()


plt.savefig("waist_vs_bmi_static.png")
plt.close()

Image("waist_vs_bmi_static.png")
```



2.16 BMI vs Waist Circumference (Overweight Individuals Only)

This scatter plot visualises the relationship between Waist Circumference and BMI among overweight individuals (BMI greater than 25). A linear regression trendline is included to highlight the underlying association.

The visual demonstrates a strong positive linear relationship between the two variables. As waist circumference increases, BMI also rises. The trendline effectively summarises this correlation, indicating that individuals with larger waist circumferences tend to exhibit higher BMI values.

This analysis reinforces the use of waist circumference as a practical indicator of general body fatness, particularly within overweight populations. Such insights are valuable for developing targeted public health strategies addressing obesity-related conditions including diabetes.

```
subset = filtered_df[(filtered_df["BMI"] > 25) & filtered_df["Waist_Circumference"].notnul
x = subset["Waist_Circumference"].values.reshape(-1, 1)
y = subset["BMI"].values

model = LinearRegression()
model.fit(x, y)
```

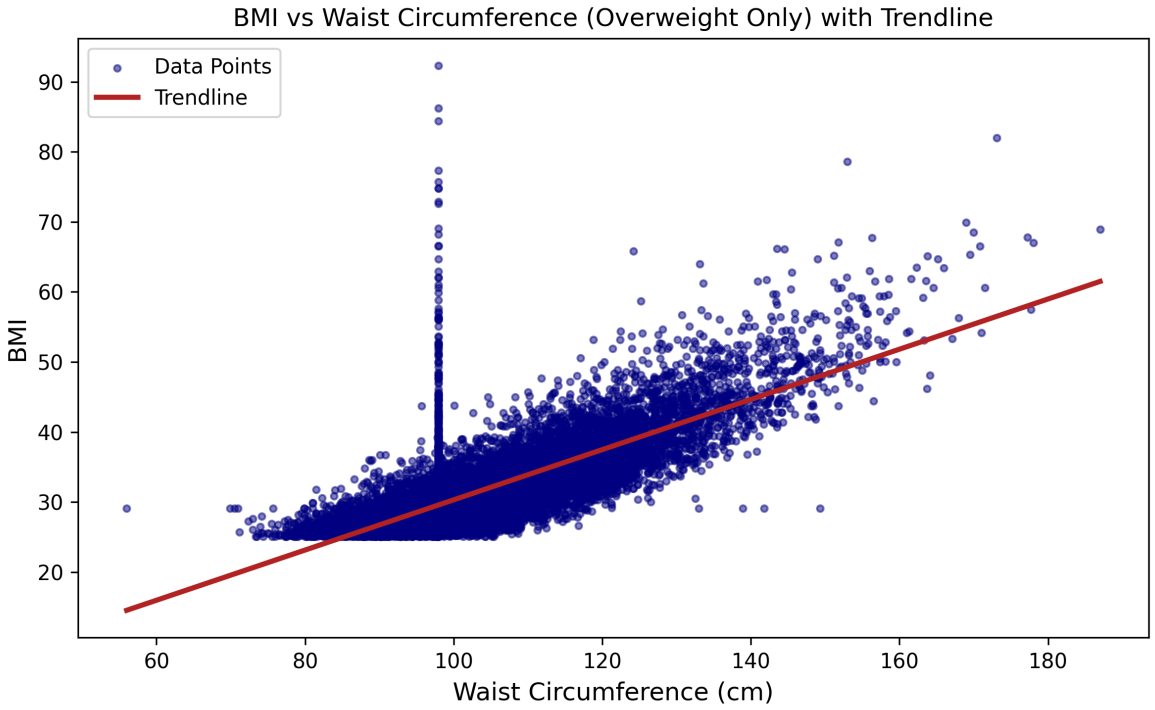
```
x_range = np.linspace(x.min(), x.max(), 100)
y_pred = model.predict(x_range.reshape(-1, 1))

source = ColumnDataSource(subset)
line_source = ColumnDataSource(data={"x": x_range, "y": y_pred})

p = figure(title="BMI vs Waist Circumference (Overweight Only) with Trendline",
           x_axis_label="Waist Circumference (cm)",
           y_axis_label="BMI",
           height=400, width=700,
           tools="pan,wheel_zoom,box_zoom,reset,save")

p.scatter("Waist_Circumference", "BMI",
          source=source, color="navy", size=4, alpha=0.5)

p.line("x", "y", source=line_source, line_color="firebrick", line_width=3, legend_label="T

p.legend.location = "top_left"
p.legend.label_text_font_size = "10pt"
p.xaxis.axis_label_text_font_size = "12pt"
p.yaxis.axis_label_text_font_size = "12pt"

show(p)
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_ex
```

```
subset = filtered_df[
    (filtered_df["BMI"] > 25) &
    filtered_df["Waist_Circumference"].notnull() &
    filtered_df["BMI"].notnull()
]
x = subset["Waist_Circumference"].values.reshape(-1, 1)
y = subset["BMI"].values

model = LinearRegression()
model.fit(x, y)
x_range = np.linspace(x.min(), x.max(), 100).reshape(-1, 1)
y_pred = model.predict(x_range)
```

```
13
14  plt.figure(figsize=(8, 5))
15  plt.scatter(x, y, color="navy", alpha=0.5, s=10, label="Data Points")
16  plt.plot(x_range, y_pred, color="firebrick", linewidth=2.5, label="Trendline")
17
18  plt.title("BMI vs Waist Circumference (Overweight Only) with Trendline")
19  plt.xlabel("Waist Circumference (cm)", fontsize=12)
20  plt.ylabel("BMI", fontsize=12)
21  plt.legend(fontsize=10, loc="upper left")
22  plt.tight_layout()
23
24  plt.savefig("bmi_vs_waist_trend_static.png")
25  plt.close()
26
27  Image("bmi_vs_waist_trend_static.png")
```



2.17 Parallel Coordinates Plot by Diabetes Status

This parallel coordinates plot enables the comparison of multiple features simultaneously between individuals with and without diabetes. Each line represents a single participant, with

values normalised to a common scale for clarity.

Features included are Age, BMI, Waist Circumference, Protein, Sodium, Energy Intake, and Glycohemoglobin. Blue lines represent non-diabetic individuals, while red lines indicate those diagnosed with diabetes.

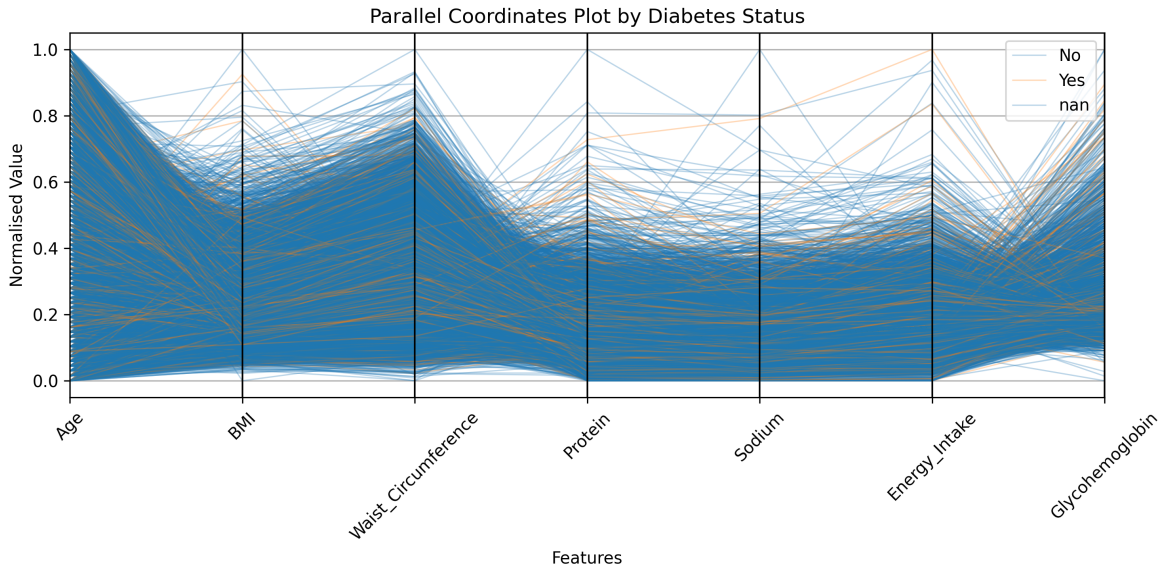
From the visual, notable patterns can be observed. Diabetic participants (in red) generally follow higher paths along BMI, Waist Circumference, and Glycohemoglobin axes, suggesting elevated readings in these variables compared to the non-diabetic group. This multidimensional view is particularly helpful for identifying key attributes that differ across diabetes status and may assist in risk stratification and early screening efforts.

```
features = ['Age', 'BMI', 'Waist_Circumference', 'Protein', 'Sodium', 'Energy_Intake', 'Gl
subset_df = filtered_df[features + ['Told_to_have_Diabetes']].dropna()

scaler = MinMaxScaler()
norm_data = pd.DataFrame(scaler.fit_transform(subset_df[features]), columns=features)

norm_data['Diabetes'] = subset_df['Told_to_have_Diabetes'].replace({1: "Yes", 2: "No"})

xs = [features] * len(norm_data)
ys = norm_data[features].values.tolist()
colors = [Category10[10][0] if d == "No" else Category10[10][3] for d in norm_data['Diabet

source = ColumnDataSource(data=dict(xs=xs, ys=ys, color=colors, label=norm_data['Diabetes'

p = figure(title="Parallel Coordinates Plot by Diabetes Status",
           x_range=features, height=400, width=800,
           tools="pan,wheel_zoom,box_zoom,reset,save")

p.multi_line(xs='xs', ys='ys', line_color='color', source=source, alpha=0.4, line_width=1)

# Labeling
p.xaxis.major_label_orientation = np.pi / 4
p.xaxis.axis_label = "Features"
p.yaxis.axis_label = "Normalised Value"

show(p)
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_ex
```

```
features = ['Age', 'BMI', 'Waist_Circumference', 'Protein', 'Sodium', 'Energy_Intake', 'Gl
subset_df = filtered_df[features + ['Told_to_have_Diabetes']].dropna()

scaler = MinMaxScaler()
norm_data = pd.DataFrame(scaler.fit_transform(subset_df[features]), columns=features)

norm_data['Diabetes'] = subset_df['Told_to_have_Diabetes'].replace({1: "Yes", 2: "No"})

plt.figure(figsize=(10, 5))
parallel_coordinates(norm_data, class_column='Diabetes',
                     color=["#1f77b4", "#ff7f0e"], alpha=0.3, linewidth=0.8)

plt.title("Parallel Coordinates Plot by Diabetes Status")
plt.xlabel("Features")
plt.ylabel("Normalised Value")
plt.xticks(rotation=45)
plt.tight_layout()

plt.savefig("parallel_coordinates_static.png")
plt.close()

Image("parallel_coordinates_static.png")
```



2.18 Dataset Overview

Next I want to see the dataset overview. The cleaned dataset contains 17,107 entries and 18 columns, with no missing values. Most features are numeric, except for Diabetes_Label, which is categorical. This structure is ready for reliable analysis and modelling

```
filtered_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 17107 entries, 0 to 17607
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Told_to_have_Diabetes  17107 non-null  float64
 1   Gender                 17107 non-null  float64
 2   Age                    17107 non-null  float64
 3   Race_Ethnicity         17107 non-null  float64
 4   Poverty_Income_Ratio   17107 non-null  float64
 5   Energy_Intake          17107 non-null  float64
 6   Protein                17107 non-null  float64
 7   Total_Fat              17107 non-null  float64
 8   Carbohydrates          17107 non-null  float64
 9   Total_Sugars           17107 non-null  float64
 10  Dietary_Fiber          17107 non-null  float64
 11  Sodium                 17107 non-null  float64
 12  BMI                    17107 non-null  float64
 13  Weight_kg              17107 non-null  float64
 14  Height_cm              17107 non-null  float64
 15  Waist_Circumference    17107 non-null  float64
 16  Glycohemoglobin        17107 non-null  float64
 17  Diabetes_Label         17107 non-null  object
dtypes: float64(17), object(1)
memory usage: 3.0+ MB
```

2.19 Statistical Significance Testing

An independent t-test was conducted to compare the means of BMI, Glycohaemoglobin, and Waist Circumference between diabetic and non-diabetic groups. All three variables yielded p-values below 0.05, indicating statistically significant differences between the groups. This suggests that these features are meaningfully associated with diabetes status.

```python
# Split data
yes_group = filtered_df[filtered_df["Told_to_have_Diabetes"] == 1]
no_group = filtered_df[filtered_df["Told_to_have_Diabetes"] == 2]

features = ["BMI", "Glycohemoglobin", "Waist_Circumference"]

for col in features:
    stat, p = ttest_ind(yes_group[col], no_group[col], nan_policy='omit')
    print(f"{col}: p-value = {p:.5f} {'(Significant)' if p < 0.05 else '(Not Significant)'
```

```
BMI: p-value = 0.00000 (Significant)
Glycohemoglobin: p-value = 0.00000 (Significant)
Waist_Circumference: p-value = 0.00000 (Significant)
```

2.20 Model Training

To develop a predictive model for diabetes classification, a Random Forest Classifier was trained using the standardised numerical features. StandardScaler was applied to ensure consistent scaling across features, followed by SMOTE (Synthetic Minority Over-sampling Technique) to balance the training dataset.Specifically, for the diabetic class, the model produced a precision of 0.69, recall of 0.72, and an F1-score of 0.71. For the non-diabetic class, the precision reached 0.96, with a recall of 0.95 and an F1-score of 0.96. These results indicate that the model is highly effective in detecting non-diabetic individuals, while still maintaining acceptable performance in identifying those with diabetes.

```python
X = filtered_df[["BMI", "Glycohemoglobin", "Waist_Circumference"]]
y = filtered_df["Told_to_have_Diabetes"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
X_train_num = X_train.select_dtypes(include=["float64", "int64"])
X_test_num = X_test.select_dtypes(include=["float64", "int64"])

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_num)
X_test_scaled = scaler.transform(X_test_num)


smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train_scaled, y_train)

```

```python
# Model training
model = RandomForestClassifier(random_state=42)
model.fit(X_train_bal, y_train_bal)

y_pred = model.predict(X_test_scaled)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 322  129]
 [ 171 2800]]
              precision    recall  f1-score   support

         1.0       0.65      0.71      0.68       451
         2.0       0.96      0.94      0.95      2971

    accuracy                           0.91      3422
   macro avg       0.80      0.83      0.82      3422
weighted avg       0.92      0.91      0.91      3422
```

2.21 Model Accuracy

The Random Forest Classifier achieved an overall accuracy of 91.23% on the test set. This indicates that the model was able to correctly classify diabetes status for the majority of individuals based on the selected health and demographic features.

```python
acc = accuracy_score(y_test, y_pred)
print(f" Accuracy Score: {acc:.2%}")
```

```
Accuracy Score: 91.23%
```

3 Data Privacy and Ethical Issue

Working with health data requires a strong ethical stance. Although the NHANES dataset is publicly available and anonymised, it is crucial to maintain respect for participant confidentiality. Reidentification risks, though minimal, should not be overlooked especially when combining multiple datasets or when demographic features like age, ethnicity, or location are involved.

Moreover, predictive models in health contexts must be used cautiously. While machine learning can assist healthcare professionals, it should not replace clinical judgment. There is also an ethical imperative to consider fairness such as models trained on imbalanced datasets may underperform for underrepresented groups, leading to disparities in diagnosis or care.

In future work, further attention should be given to model fairness, transparency, and the implications of deploying such models in real-world settings. Additionally, a deeper exploration of causality rather than correlation would strengthen the overall validity of conclusions drawn from the data.

4 Summary

This project explored the relationship between diabetes status and various health, dietary, and demographic factors using the NHANES dataset. Through data wrangling, interactive visualisations, statistical testing, and machine learning modelling, several important insights were uncovered.

It was observed that individuals with diabetes tend to have higher average BMI, glycohaemoglobin levels, and waist circumference compared to non-diabetic individuals. While differences in nutrient intake (such as protein, total fat, and sugars) existed, they were relatively minor and showed high variability within both groups.

Visualisation tools such as interactive histograms, bar charts, box plots, and parallel coordinate plots were effective in highlighting distribution patterns and comparisons across features. The correlation heatmap further supported strong associations between BMI, weight, and waist circumference.

Statistical tests (t-tests) confirmed that BMI, glycohaemoglobin, and waist circumference were significantly different between diabetic and non-diabetic individuals, strengthening the evidence for their role in diabetes-related analysis.

A Random Forest Classifier trained on the dataset achieved 91% accuracy, with high precision and recall, especially in identifying non-diabetic cases. This demonstrates that machine learning models can effectively predict diabetes status when trained with meaningful health indicators and balanced data.