NTU Nottingham Trent University
Department of Computer Science

Python Programming for Real World Data Analytics and GUI Development

SOFT40161 - Introduction to Computer Programming

Submission of the Coursework

Name: Natan Majchrzak

NTU ID: N1428754

The Full description of the Coursework is here - SOFT40161-Coursework (Click here)

Start your answer here-

GitHub Link to access your activity.

Please add your Link here: https://github.com/NM1428/Pima-Diabetes-Analysis-System..git

Section 1: Control Structures (12 marks)

Explanation and Documentation

Dataset Overview

The dataset used in this analysis is the Pima Indians Diabetes Dataset, originally used to test the ADAP learning algorithm by Smith et al. (1988). The data captures physiological measurements from a high-risk population in Arizona. The dataset was retrieved from the kaggle repository (National Institute of Diabetes and Digestive and Kidney Diseases).

Real-World Problem

Diabetes is a chronic public health issue that affects how the body turns food into energy. In a clinical setting, identifying high risk individuals and anomalies is critical for early intervention and personalised treatment. This analysis uses computational control structures to isolate these groups for further study.

Diabetes management relies on identifying high-risk physiological markers from a clinical informatics perspective. The primary challenge is to isolate "at risk" patient clusters and identify medical anomalies such as patients diagnosed with diabetes despite having low-range glucose levels, which may indicate unique clinical phenotypes or data entry errors.

Methodology: Control structures

- PD.read_csv was used to import the data into a structured dataframe.
- Iteration (for loop): iterrows() function was utilised to traverse every row of the dataset individually
- Conditional logic (if statement): used conditionals to filter for high risk patients to calculate their average BMI.
  - Used nested if statements to identify patients who were diagnosed with diabetes but presented with low glucose levels, which is an unusual medical finding.
- List structure: storing the indices of specific patients, allowing for easy data retrieval at the end of the script.

In [1]:
```
import pandas as pd

#Loading the dataset
file = "https://raw.githubusercontent.com/plotly/datasets/master/diabetes.csv"
pid_df = pd.read_csv(file)
#testing to make sure the dataset was loaded correctly.
#print(pid_df)

#Initializing variables
high_risk_threshold = 140
low_risk_threshold = 100
bmi_sum = 0
high_risk_patients = 0
anomaly_list = [] #empty list to store anomalies in.

for index, row in pid_df.iterrows():
    current_glucose = row["Glucose"]
    current_bmi = row["BMI"]
    current_outcome = row["Outcome"]
    #Average calculation
    if current_glucose > high_risk_threshold:
        bmi_sum += current_bmi
        high_risk_patients += 1
    #Anomaly detection
    if current_outcome == 1:
        #saves all anomalies into a list, this way all anomalies (outliers) can
        if current_glucose <= low_risk_threshold:
            anomaly_list.append(index)

if high_risk_patients > 0:
    average = bmi_sum / high_risk_patients
    print("Total amount of patients in the high risk threshold: ", high_risk_pat

print("The anomalies detected (indicies): ", anomaly_list) #index numbers of the
print(pid_df.loc[anomaly_list, ["Glucose", "BMI", "Outcome"]]) #displaying the a
```

```
Total amount of patients in the high risk threshold:  192
Average BMI of the high risk patients:  34.92708333333334
The anomalies detected (indicies):  [6, 15, 38, 70, 109, 125, 218, 254, 298, 349,
400, 429, 502, 510, 540, 542, 638, 659, 709, 719]
     Glucose   BMI  Outcome
6         78  31.0        1
15       100  30.0        1
38        90  38.2        1
70       100  32.9        1
109       95  37.4        1
125       88  55.0        1
218       85  29.0        1
254       92  27.6        1
298      100  36.6        1
349        0  41.0        1
400       95  32.0        1
429       95  35.0        1
502        0  39.0        1
510       84  29.7        1
540      100  39.4        1
542       90  34.9        1
638       97  40.9        1
659       80  34.2        1
709       93  38.0        1
719       97  35.6        1
```

Reflection on results

Analysis results:

- High risk patient insights: The average BMI of high risk patients was calculated to be approximately 34.93 indicating a correlation between weight and elevated glucose levels.
- Anomalies: Identified 20 anomaly patients who have been diagnosed with diabetes despite maintaining relatively low glucsose levels at the time of testing.

Reflection and Limitations:

- While the itterows() aproach is highly readable and perfect for demonstrating control structures it will be computationally slow on a large database with millions of rows.
- A Significant limitation noted in this dataset is the presence of 0 values in columns like BMI and Glucose, which It represnets missing data rather than the real measurments.

- To improve the accuracy of the average BMI calculation, future iterations of this code should a data handling section to remove or impute the zeros in the critical columns before the loop begins.

Section 2: Functions and Modules (16 marks)

Explanation and Documentation

The primary goal of this section is the implementation of modular functions using multiple external libraries to perform a comprehensive data analysis of the Pima Indians Diabetes dataset.

Splitting the code into multiple reusable functions is essential for maintaining a high standard of code quality.

Each function has a single, clear responsibility – loading, auditing, or visualising making the code easier to maintain and test. By using parameters, these functions can be applied to other datasets with minimal modification. Error handling has been implemented with the try-except blocks and conditional checks. This ensures that if the external source is unavailable or the data is missing, the program provides clear feedback rather than crashing.

External Modules:

Four specialised external modules were used.

1. Pandas: used for robust data ingestion and creating a correlation matrix for the analysis.
2. NumPy: Leveraged for high-performance numerical auditing. I used np.sum() to identify hidden missing values (zeros) in specific important columns where a zero value is medically invalid.
3. Matplotlib: Used to configure the visual environment, setting the figure size for the charts.
4. Seaborn: Used for visualisation to generate a color coded heatmap of feature correlations.

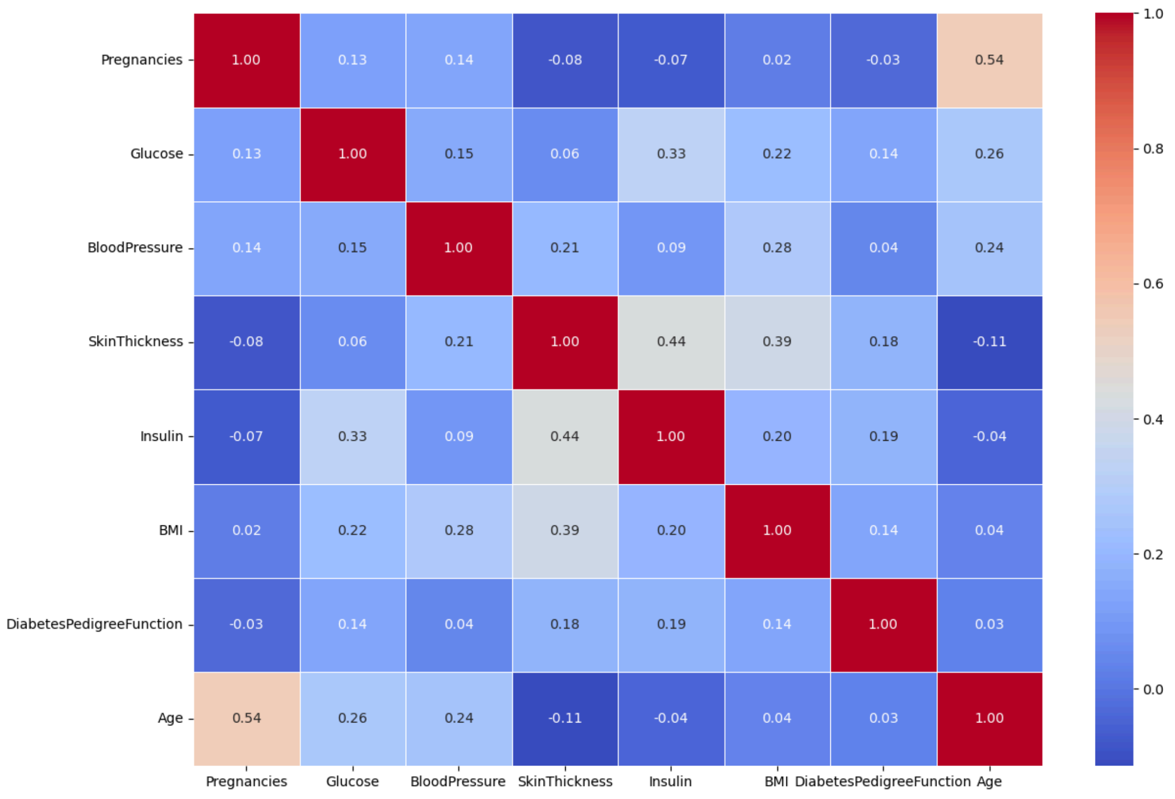
Functions:

- load_dataframe(file_path): safely imports the dataset into a dataframe, using a try-except block to make sure the data source is successfully reached.
- data_summary(df, check_cols): this function provides a descriptive analysis and identifies critical data gaps using numpy logical operators, it also includes a validation check to ensure the data exists before attempting analysis.

- visualise_correlations(df): by using Seaborn and Matplotlib this function identifies relationships between the features, for example the link between SkinThickness and Insulin is 0.44.

In [2]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

file = "https://raw.githubusercontent.com/plotly/datasets/master/diabetes.csv"

cols_to_check = ['Glucose', 'Insulin', 'BMI', 'BloodPressure', 'SkinThickness']

#loads the file into a dataframe (uses pandas).
def load_dataframe(file_path):
    try:
        df = pd.read_csv(file_path) #loads the dataframe
        return df
    except FileNotFoundError:
        print("Error file not found!")
        return None

#Looking at the dataframes data and using numpy to check for entries with 0 as t
def data_summary(df, check_cols):
    #checking to make sure there is a dataframe to analyse
    if df is None:
        print("Error no data to analyse!")

    print(df.head())
    print(f"dataframes shape: {df.shape}")
    print(df.describe())

    #using numpy to check for zero entires inside specific columns
    if check_cols is not None:
        for col in check_cols:
            zero_in_cols = np.sum(df[col] == 0) #numpy operator
            print(f"{col}: {zero_in_cols} zeros found")

#Plotting data to visualise correlations to identify which features are most lik
def visualise_correlations(df):
    #defining the figure size
    plt.figure(figsize=(14, 10))

    #dropping the outcome colummn to focus on the features against eachother
    correlation_matrix = df.drop('Outcome', axis = 1).corr()
    #plotting the numbers in a heatmap where red is negative and blue is positiv
    sns.heatmap(correlation_matrix, annot = True, cmap = 'coolwarm', fmt = ".2f"


#loading the dataframe
df = load_dataframe(file)
#looking at basic information on the dataframe and checking for zeros in specifi
data_summary(df, cols_to_check)
visualise_correlations(df)
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6   
1            1       85             66             29        0  26.6   
2            8      183             64              0        0  23.3   
3            1       89             66             23       94  28.1   
4            0      137             40             35      168  43.1   

   DiabetesPedigreeFunction  Age  Outcome  
0                     0.627   50        1  
1                     0.351   31        0  
2                     0.672   32        1  
3                     0.167   21        0  
4                     2.288   33        1  
dataframes shape: (768, 9)
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000   
mean      3.845052  120.894531      69.105469      20.536458   79.799479   
std       3.369578   31.972618      19.355807      15.952218  115.244002   
min       0.000000    0.000000       0.000000       0.000000    0.000000   
25%       1.000000   99.000000      62.000000       0.000000    0.000000   
50%       3.000000  117.000000      72.000000      23.000000   30.500000   
75%       6.000000  140.250000      80.000000      32.000000  127.250000   
max      17.000000  199.000000     122.000000      99.000000  846.000000   

              BMI  DiabetesPedigreeFunction         Age     Outcome  
count  768.000000                768.000000  768.000000  768.000000  
mean    31.992578                  0.471876   33.240885    0.348958  
std      7.884160                  0.331329   11.760232    0.476951  
min      0.000000                  0.078000   21.000000    0.000000  
25%     27.300000                  0.243750   24.000000    0.000000  
50%     32.000000                  0.372500   29.000000    0.000000  
75%     36.600000                  0.626250   41.000000    1.000000  
max     67.100000                  2.420000   81.000000    1.000000  
Glucose: 5 zeros found
Insulin: 374 zeros found
BMI: 11 zeros found
BloodPressure: 35 zeros found
SkinThickness: 227 zeros found
```



Reflection on results

Analysis results

Using the data_summary function, an audit of hidden missing values was performed. The audit revealed a high frequency of zero entries in columns which were physiologically impossible, such as SkinThickness. These zeros act as noise in the data, identifying them using numpy suggests that future predictive models would require data imputation or removal of these rows to maintain accuracy. The heatmap produced by the visualise_correlations function produced a heatmap that identifies how physiological features move in relation to one another. A moderate correlation exists between insulin and SkinThickness which may relate to body composition and metabolic health, on the other hand features like DiabetesPedigreeFunction show very low correlation with Age, indicating that genetic hereditary scores in this dataset do not significantly change as the person gets older.

Section 3: Data Handling with Pandas (16 marks)

Explanation and Documentation

Tool rationale

The data pre-processing for this dataset relies on the NumPy and Pandas libraries. As noted by Harris et al. (2020), NumPy provides a powerful, compact, and expressive syntax for operating on data vectors and matrices. This "array programming" paradigm is essential for clinical data analytics because it allows for efficient handling of large-scale arrays through an interoperability layer that connects data cleaning with downstream statistical analysis. By using NumPy as the foundation, we ensure that the imputation and outlier capping processes are both computationally efficient and aligned with the standards used in global scientific research.

Data cleaning and inconsistency management:

The objective of this section was to implement a professional data cleaning pipeline to ensure the Pima Indians Diabetes dataset was accurate and reliable for deeper analysis.

During data ingestion the na_values parameter was used to identify non-standard symbols, which were then converted into formal NaN values to ensure consistent behaviour across Pandas operations. Duplicate entries were removed to prevent data skewing. A critical step was to handle the identified zeros in the medical columns. In this dataset, a zero in these specific fields represents missing data therefore, the zeros were converted into null values, ensuring they do not pull down the statistical averages. To preserve the sample size of this relatively small dataset, I chose to fill NaN values rather than dropping rows. The median was used of each column because it is a robust measure of central tendency that is not influenced by extreme outliers.

Physiological data often contains extreme values that can distort trends. To handle this, the Interquartile Range (IQR). Calculating bounds, the statistical boundaries were defined using the formula q1 - 1.5 * IQR and q3 + 1.5 * IQR. The outliers at these boundaries were capped.

The final step involved a descriptive statistical analysis to extract meaningful insights from the cleaned data. By grouping the data by Outcome (diabetic vs healthy), the mean and standard deviation were calculated for the six critical health indicators.

Error handling was implemented in the data loading function to manage FileNotFoundError, validation checks were implemented into each function (if df is None). This ensures the pipeline only continues if the data was successfully loaded, preventing system crashes.

Logic has been separated into reusable functions with descriptive variable names, making the code easy to follow.

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

file = "https://raw.githubusercontent.com/plotly/datasets/master/diabetes.csv"

numerical_cols = ['Pregnancies', 'Glucose','BloodPressure', 'SkinThickness', 'In
medical_0_error_columns = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin
```

```
#loads the file into a dataframe and standardises all missing values
def load_and_clean_data(file_path):
    missing_symbols = ['N/A', 'na', '-', '?',' '] #defining the non standard mis
    try:
        df = pd.read_csv(file_path, na_values=missing_symbols) #loads the datafr
    except FileNotFoundError:
        print("Error file not found!")

    return df #returns the dataframe

#drops all duplicates in each column
def drop_duplicates(df):
    if df is None: return None
    df_cleaned = df.drop_duplicates()

    return df_cleaned

def handle_0_errors(df, cols_to_fix):
    if df is None: return None
    for col in cols_to_fix:
        #replacing the 0s with nans
        df[col] = df[col].replace(0, np.nan)

    return df

def handle_missing_values(df, cols_to_handle):
    if df is None: return None
    #imputing the missing values (nans) with the median values becuase this is a
    for col in cols_to_handle:
        median = df[col].median()
        df[col] = df[col].fillna(median)

    return df

#uses the IQR method to cap outliers
def clean_outliers(df, numerical_columns):
    if df is None: return None
    #IQR method to handle outliers by capping them.
    for col in numerical_columns:
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)
        IQR = q3 - q1
        upper_bound = q3 + 1.5*IQR
        lower_bound = q1 - 1.5*IQR

        #replaced by numpy
        #df.loc[df[col] > upper_bound, col] = upper_bound
        #df.loc[df[col] < lower_bound, col] = lower_bound

        #forcing the values outside the bounds to equal the bounds
        df[col] = np.clip(df[col], lower_bound, upper_bound)

        print("\n")
        print(f"outliers have been capped for column: {col}")

    return df

def aggregate_health_metrics(df):
    # grouping data to find a deeper insight
    if df is None: return None

    #grouping by outcome to see how SkinThickness varies between groups
    summary = df.groupby('Outcome').agg({
        'SkinThickness': ['mean','std'], # mean and standard deviation
        'Insulin': ['mean','std'],
        'BloodPressure': ['mean','std'],
        'BMI': ['mean','std'],
        'Glucose': ['mean', 'std'],
        'DiabetesPedigreeFunction': ['mean', 'std']

    })
    return summary.round(2) # 2 decimal place for better readability




df = load_and_clean_data(file)
#print(df)
#print("\n")
print(f"{df.shape}")
df = drop_duplicates(df)
print(f"{df.shape}")
df = handle_0_errors(df, medical_0_error_columns)
df = handle_missing_values(df, medical_0_error_columns)
df = clean_outliers(df, numerical_cols)

summary_health_metrics = aggregate_health_metrics(df)
display(summary_health_metrics)
```

```
(768, 9)
(768, 9)


outliers have been capped for column: Pregnancies


outliers have been capped for column: Glucose


outliers have been capped for column: BloodPressure


outliers have been capped for column: SkinThickness


outliers have been capped for column: Insulin


outliers have been capped for column: BMI


outliers have been capped for column: DiabetesPedigreeFunction


outliers have been capped for column: Age
```

	SkinThickness		Insulin		BloodPressure		BMI		Glucose		Diabete
	mean	std	mean	std	mean	std	mean	std	mean	std	
Outcome											
0	27.67	7.6	123.15	8.00	70.91	11.54	30.87	6.44	110.68	24.71	
1	31.10	6.6	127.57	6.89	75.06	11.54	35.24	6.14	142.13	29.57	

Reflection on results

The table below summarises the key differences between the two groups after performing median imputation and outlier capping.

Metric (mean)	Outcome 0 (Healthy)	Outcome 1 (Diabetic)	Observed Trend
Glucose	110.68	142.13	A clear 28 % increase in average glucose levels for diabetic patients.
BMI	30.87	35.24	Diabetic patients show a significantly higher average BMI.
SkinThickness	27.67	31.10	Higher skinfold thickness correlates with the diabetic outcome.
Insulin	123.15	127.57	While higher in diabetics, the gap is narrow, likely due to median imputation stabilizing the variance.
BloodPressure	70.91	75.06	moderate increase in the diabetic group.
Diabetes Pedigree Function	0.42	0.53	Higher hereditary risk score in diabetic patients.

The standard deviation for insulin is lower in the diabetic group (6.89) than in the healthy group (8.00). This suggests that after cleaning and imputation, the diabetic group's insulin levels in this dataset are more tightly clustered around the median, whereas the healthy group shows a more natural physiological variation.

Section 4: Data Visualization (16 marks)

Explanation and Documentation

Visual communication and logic The objective of this section was to use different types of visualisation methods to reveal hidden trends, distributions and correlations within the cleaned dataset. Six different distinct types of plots were generated (histogram, boxplot, violin plot, interactive scatter plot, regression plot and a heatmap) to provide a multidimensional view of the data. The Viridis and Magma colour palettes were chosen, as these are colour-blind inclusive, making these plots accessible to everyone. Each plot had a figure size of 15, 8 and included a title and axis labels to enhance clarity and readability.

Visualisation breakdown and insights

Distribution analysis – The histogram + KDE plot visualises the spread of Glucose levels. It reveals that while the healthy group (Outcome 0) has a normal distribution centred around 100, the diabetic group (Outcome 1) is significantly right-shifted with a much wider range, confirming high glucose as a key diagnostic factor.

Statistical spread – The boxplot visualises insulin levels across outcomes. This plot demonstrates the stability of the dataset following the median imputation and outlier capping performed.

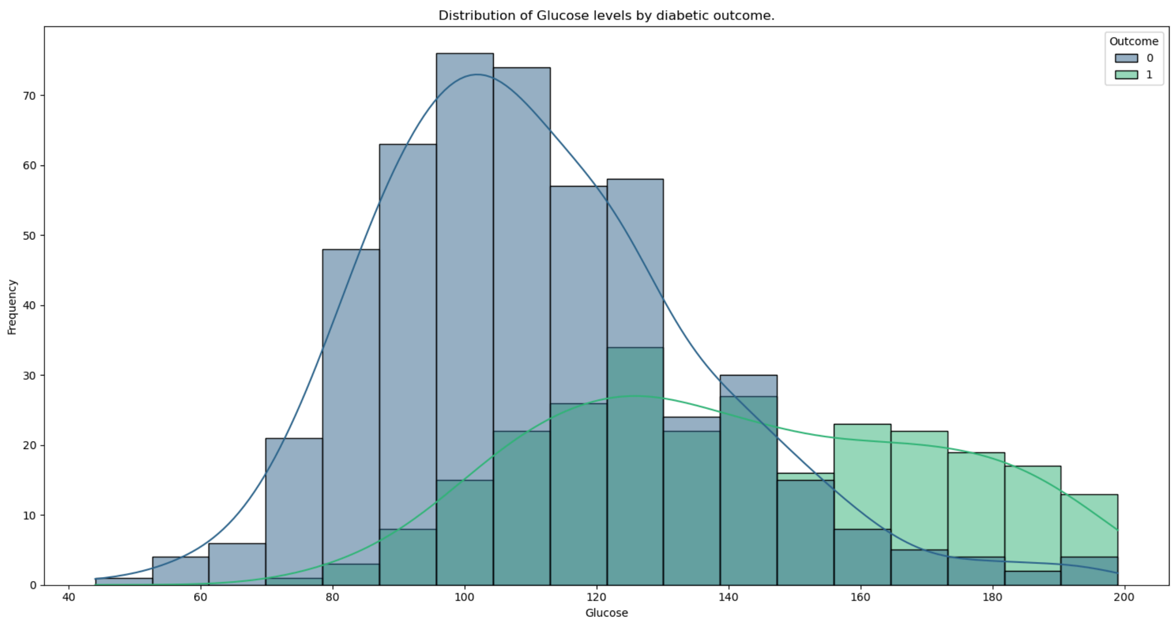
Genetic Density - The Diabetes Pedigree Function violin plot shows the probability density of hereditary risk. The diabetic group shows a thicker density at higher values, indicating a strong genetic link to the disease.

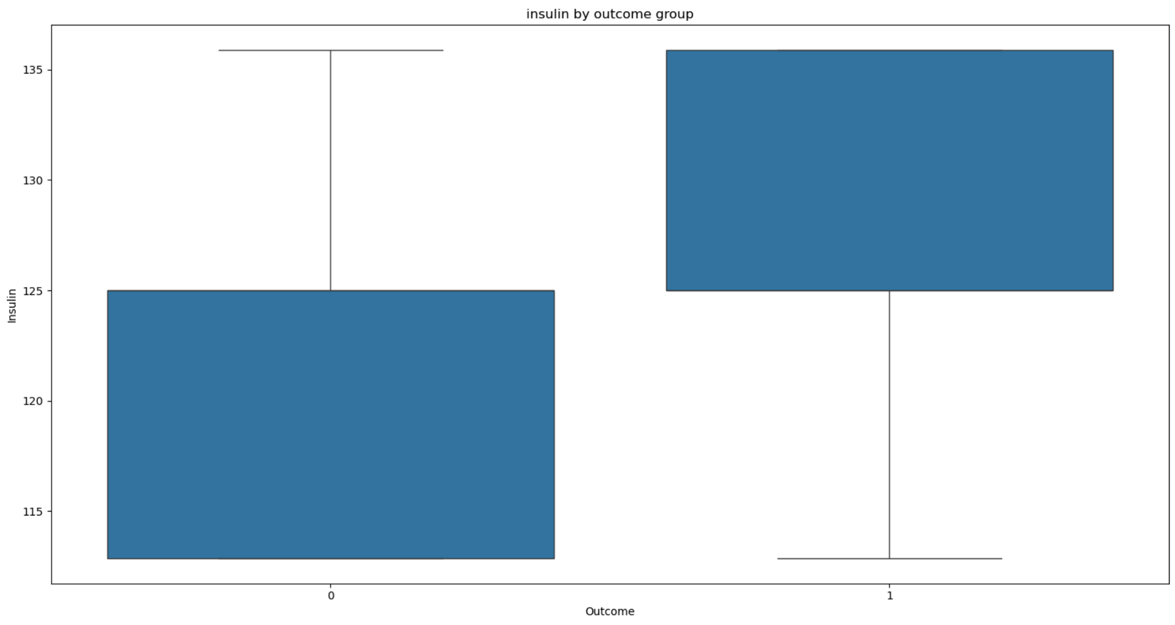
Trend analysis – The regression plot was used to explore the relationship between age and BMI. The red regression line indicates a slight positive correlation, showing that body mass index tends to increase as patients age within this population.

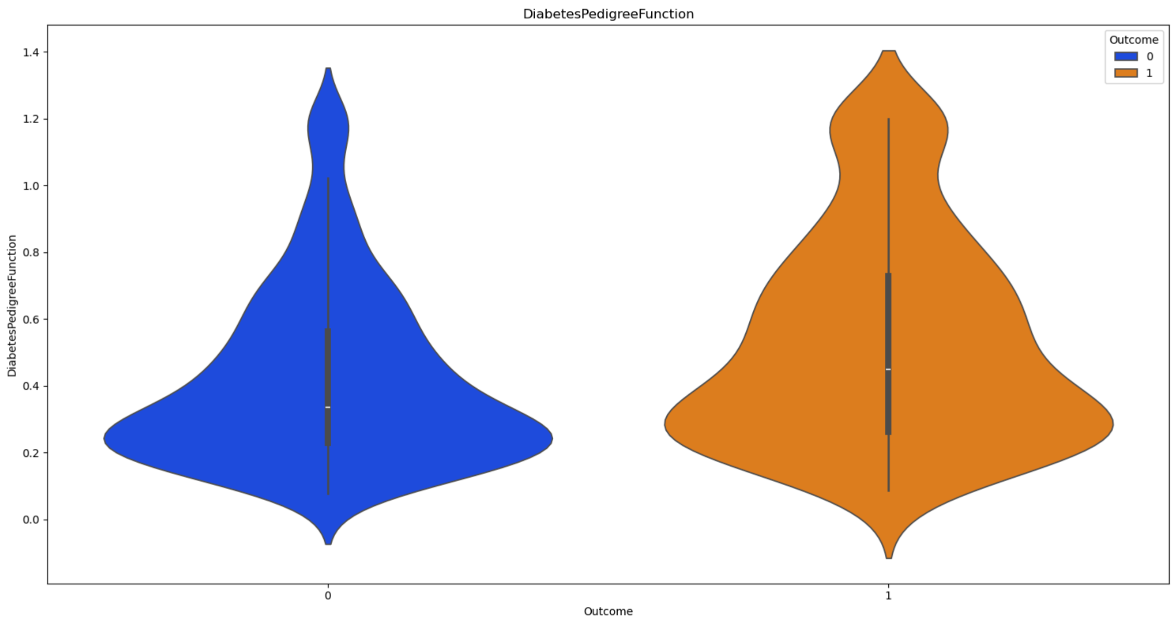
Correlation Analysis – The heatmap summarises how all the variables interact. It highlights that glucose (0.49) and BMI (0.31) have the highest positive correlations with the final diabetic outcome.

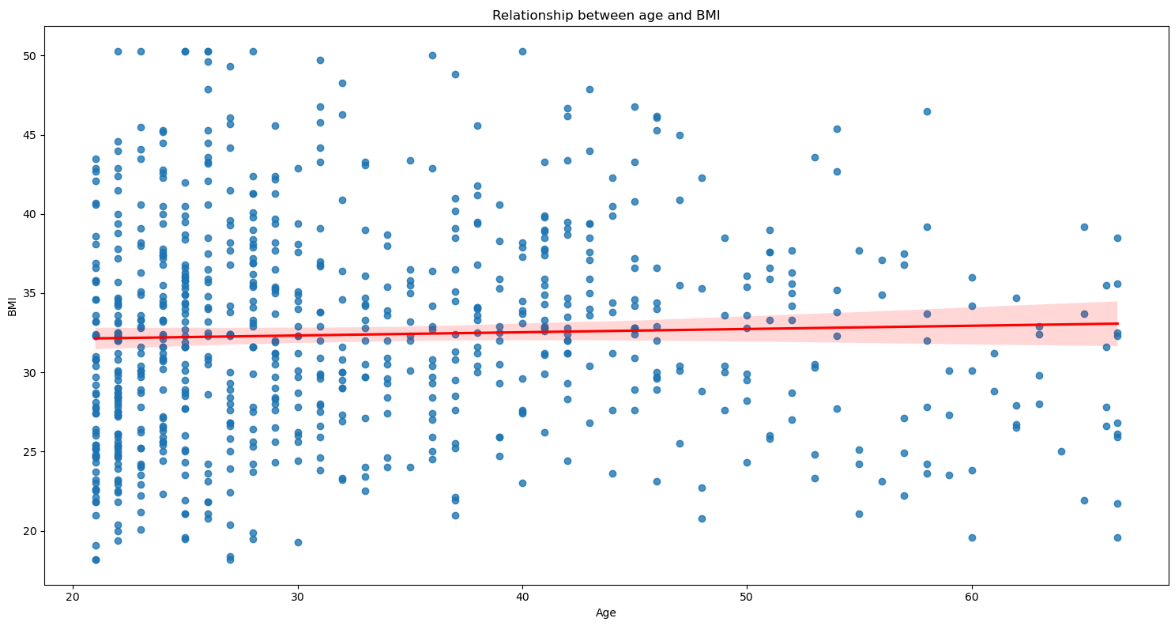
Interactive exploration – An interactive scatter plot of BMI vs glucose was created which allows users to hover over individual points to see exact values, effectively visualising the cluster of diabetic outcomes at high BMI/glucose intersections.

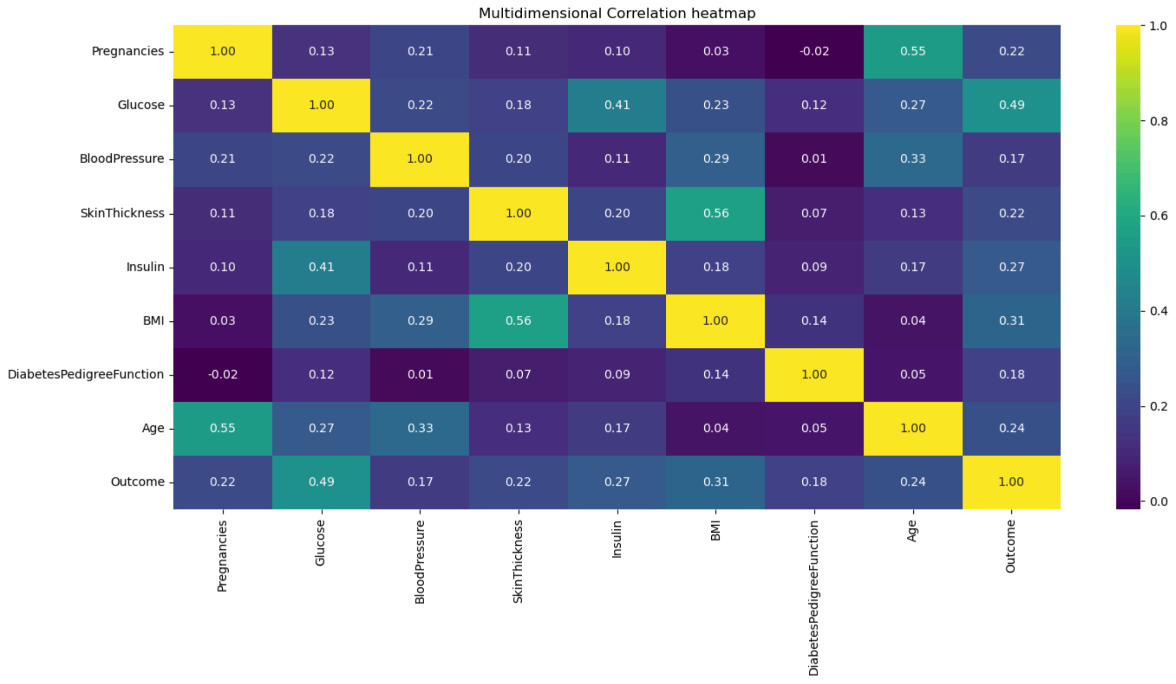
In [4]:
```
# To make sure the plots get displayed inline and not in seperate windows like i
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

def histogram(df):
    # Histogram + KDE
    # Generates a distribution plot for glucose levels. Showing how glucose leve
    plt.figure(figsize = (15, 8))
    sns.histplot(data = df,x = 'Glucose', kde = True, hue = 'Outcome', palette =
    plt.title("Distribution of Glucose levels by diabetic outcome.")
    plt.xlabel("Glucose")
    plt.ylabel("Frequency")
    plt.tight_layout()
    plt.show()

def boxplot(df):
    #boxplot
    # Generates a boxplot to compare the spread and median of insulin levels bet
    plt.figure(figsize = (15, 8))
    sns.boxplot(x = 'Outcome', y = 'Insulin', data = df)
    plt.title("insulin by outcome group")
    plt.tight_layout()
    plt.show()

def violin_plot(df):
    #Violin plot for the DiabetesPedigreeFunction.
    #combines a boxplot with kernel density plot to show data density.
    plt.figure(figsize = (15, 8))
    sns.violinplot(data = df, x = 'Outcome', y = 'DiabetesPedigreeFunction',hue
    plt.title("DiabetesPedigreeFunction")
    plt.tight_layout()
    plt.show()

def regression_plot(df):
    #Scatter plot with regression
    #creates a scatter plot with a linear regression line to analyse teh relatio
    plt.figure(figsize = (15, 8))
    sns.regplot(data = df, x = 'Age', y = 'BMI', line_kws = {'color': 'red'})
    plt.title("Relationship between age and BMI")
    plt.tight_layout()
    plt.show()

def heatmap(df):
    #Heatmap
    #Creates a correlation heatmap for the entire dataset
    plt.figure(figsize = (15, 8))
    corr = df.corr()
    sns.heatmap(corr, annot = True, fmt = '.2f', cmap = 'viridis')
    plt.title("Multidimensional Correlation heatmap")
    plt.tight_layout()
    plt.show()

def interactive_scatter(df, target = "notebook"): #target: "Notebook" to display
    #interactive scatter
    # Creates an interactive plotly scatter plot for BMI vs Glucose.
    fig = px.scatter(df, x = 'BMI', y = 'Glucose', color = 'Outcome', title = 'I
    #fig.update_layout(margin = dict(l=40, r = 40, t= 70, b = 60), autosize = Tr
    if target == "browser":
        fig.show(renderer = "browser")
    else:
        fig.show()

def visualisation(df):
    histogram(df)
    boxplot(df)
    violin_plot(df)
    regression_plot(df)
    heatmap(df)
    interactive_scatter(df)

visualisation(df)
```


Distribution of Glucose levels by diabetic outcome.
Outcome
0
1
Frequency
Glucose
0
10
20
30
40
50
60
70
40
60
80
100
120
140
160
180
200


insulin by outcome group
Insulin
Outcome
115
120
125
130
135
0
1


DiabetesPedigreeFunction
Outcome
0
1
DiabetesPedigreeFunction
Outcome
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
1


Relationship between age and BMI
BMI
Age
20
30
40
50
60
20
25
30
35
40
45
50


Multidimensional Correlation heatmap
Pregnancies 1.00 0.13 0.21 0.11 0.10 0.03 -0.02 0.55 0.22
Glucose 0.13 1.00 0.22 0.18 0.41 0.23 0.12 0.27 0.49
BloodPressure 0.21 0.22 1.00 0.20 0.11 0.29 0.01 0.33 0.17
SkinThickness 0.11 0.18 0.20 1.00 0.20 0.56 0.07 0.13 0.22
Insulin 0.10 0.41 0.11 0.20 1.00 0.18 0.09 0.17 0.27
BMI 0.03 0.23 0.29 0.56 0.18 1.00 0.14 0.04 0.31
DiabetesPedigreeFunction -0.02 0.12 0.01 0.07 0.09 0.14 1.00 0.05 0.18
Age 0.55 0.27 0.33 0.13 0.17 0.04 0.05 1.00 0.24
Outcome 0.22 0.49 0.17 0.22 0.27 0.31 0.18 0.24 1.00
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
1.0
0.8
0.6
0.4
0.2
0.0

Interactive scatter: BMI vs Glucose

Reflection on results

The visualisations provide clear physiological patterns within the dataset:

- The histogram and violin plot demonstrate that glucose and diabetes pedigree function are powerful discriminators. The right shift in the diabetics' glucose distribution and the higher density at upper pedigree scores visually suggest these are primary risk indicators.
- The boxplot confirms the insulin levels for both groups are statistically stable, though the diabetic group maintains a slightly higher median and quartile range.
- The heatmap and regression plot quantify the relationships between variables. The heatmap identifies glucose as the strongest correlate to a diabetic outcome, while the regression plot visualises a subtle, consistent upward trend of BMI as patients age.
- The interactive scatter plot reveals that most diabetic outcomes are concentrated where both BMI and glucose levels are high, suggesting that the nature of the disease is multifactorial.

The visual data confirms that while glucose is a dominant factor, diabetes risk is a composite of genetic pedigree, age and body mass.

Section 5: GUI Development (16 marks)

Explanation and Documentation

GUI Architecture and Rationale

A single-page application architecture was developed using the Tkinter library, adhering to the event-driven programming model established by Lundh's (1999) principles of geometry management, where specialised views, such as the summary report, are dynamically injected into a single root container. The clear_frame() utility function was created to manage this container; it ensures every time a user navigates to a new section, the page is wiped clean of previous widgets. This approach prevents overlapping text or buttons and gives it a professional appearance.

To ensure compatibility between the data analysis and the interface, plt.switch_backend('Tkagg') was included specifically within the visualisation menu. This allows the GUI to launch each plot from section 4 whilst keeping the section 4 visualisation plots neatly embedded within the Jupyter Notebook.

Features

The GUI provides 3 distinct options to facilitate interaction with the Pima Indian Diabetes dataset:

- Show summary: This feature captures standard console output using io.StringIO() and contextlib.redirect_stdout. It then renders this data in a ScrolledText widget. Following the technical standards set by Shipman (2013), the widget state is set to disabled after the data is inserted. This prevents the accidental modification of the data by the user while retaining the ability to scroll and read.
- Subsets menu: A dedicated Subsets menu has been implemented, which allows users to isolate diabetic patients or those with high BMI and immediately view the statistical profile of that specific group.
- Visualisation gallery: This gallery maps GUI button events to custom plotting functions developed in section 4. The mapping of buttons to specific event handlers is a core component of the callback architecture described by Lundh (1999). It supports both static Seaborn/matplotlib charts and interactive plotly scatter plots.

Usability

Usability was a key focus of this design to ensure it functions as a tool for researchers.

- Navigation: Every sub-page includes a back button to the previous page; this ensures a cyclical navigation flow, preventing the user from getting stuck in a sub-view.
- Visual standardisation: The interface uses a standard color and font system documented by Shipman (2013). Using standard colors such as "lightgreen" and standard fonts such as "Arial" ensures the interface remains visually consistent across different operating systems.

A global try-except and local try-except logic protects the user from application crashes.

In [5]:
```
import tkinter as tk
from tkinter import scrolledtext
import io
import contextlib

def run_gui(df, medical_cols):
    try:
        root = tk.Tk() # initialises the main window of the GUI, root is the par
        root.title("Diabetes Data Analysis System")
        root.geometry("1920x1080") # window size

        main_container = tk.Frame(root)
        main_container.pack(fill = "both", expand = True)

        def clear_frame():
            # A utility function used to remove elements from the screen and mem
            for widget in main_container.winfo_children():
                widget.destroy()

        def show_main_menu():
            clear_frame()

            label = tk.Label(main_container, text = "Main Menu", font = ("Arial"
            label.pack(pady = 20)
            #main menu buttons
            show_summary_button = tk.Button(main_container, text = "Show summary

            select_visualisation_button = tk.Button(main_container, text = "Visu

            select_subset_button = tk.Button(main_container, text = "Data subset
            #button to leave the gui
            exit_button = tk.Button(main_container, text = "Exit", font = ("Aria

        def show_summary():
            clear_frame() # gets rid of the main menu buttons to make sure the p
            tk.Label(main_container, text = "Pima Indian Diabetes Dataset summar

            f = io.StringIO() # Creating a virtual file that can hold text
            with contextlib.redirect_stdout(f): # everything will be outputed in
              data_summary(df, medical_cols)

            output_text = f.getvalue()

            #Displaying the text in the GUI, Creating a large text box, pasting
            text = scrolledtext.ScrolledText(main_container, width = 100, height
            text.insert(tk.INSERT, output_text)
            text.config(state = "disabled") # user cannot change the displayed i
            text.pack(padx = 10, pady = 20)
            #a return button to the main menu
            tk.Button(main_container, text = "Back to main menu", command = show

        def select_visualisation_method():
            plt.switch_backend('TkAgg') # Making sure that the visualisation plo
            clear_frame()
            tk.Label(main_container, text = "Pima Indian Diabetes Dataset visuli
            #buttons for each plot type
            tk.Button(main_container, text = "Glucose distribution", width = 30,
            tk.Button(main_container, text = "Insulin Statistical spread", width
            tk.Button(main_container, text = "Genetic density", width = 30, comm
            tk.Button(main_container, text = "Trend analysis", width = 30, comma
            tk.Button(main_container, text = "Corelation analysis", width = 30,
            tk.Button(main_container, text = "Interactive exploration", width =

            tk.Button(main_container, text = "Back to main menu", command = show

        def subsets(df, choice):
            clear_frame()
            tk.Label(main_container, text = f"Subset: {choice}", font = ("Arial"
            try:
                if choice == "Diabetic group (outcome = 1)":
                    filtered_df = df[df['Outcome'] == 1]
                elif choice == "Non diabetic group (outcome = 0)":
                    filtered_df = df[df['Outcome'] == 0]
                elif choice == "High BMI":
                    filtered_df = df[df['BMI'] > 30]
                elif choice == "High BMI (Diabetic)":
                    filtered_df = df[(df['BMI'] > 30) & (df['Outcome'] == 1)]

                f = io.StringIO()
                with contextlib.redirect_stdout(f):
                    data_summary(filtered_df, check_cols = None)

                output_text = f.getvalue()

                text = scrolledtext.ScrolledText(main_container, width = 100, he
                text.insert(tk.INSERT, output_text)
                text.config(state = "disabled")
                text.pack(padx = 10, pady = 20)

                tk.Button(main_container, text = "Back to subset selection", com

            except Exception as e:
                print("Invalid choice was made")

        def select_subset():
            clear_frame()
            tk.Label(main_container, text = "Pima Indian Diabetes Dataset Subset

            tk.Button(main_container, text = "Diabetic outcome group", width = 3
            tk.Button(main_container, text = "Non Diabetic outcome group", width
            tk.Button(main_container, text = "High BMI", width = 30, command = l
            tk.Button(main_container, text = "High BMI (Diabetic)", width = 30,

            tk.Button(main_container, text = "Back to main menu", command = show
        show_main_menu()
        root.mainloop()
    except Exception as e:
        print(f"GUI error: {e}")

run_gui(df, medical_0_error_columns)
```

Reflection on results

A single-page application architecture was implemented within Tkinter, it was a successful move from static code execution to a user-centric system. The architecture, centred around a main container and dynamic view injection, proved to be an effective strategy for creating a clean and organised user interface. The integration of Matplotlib visualisations within the Tkinter window provided a seamless experience, allowing users to move from data analysis to visual exploration without interruption. The features for viewing summary statistics, subsetting data and accessing teh visualisation gallery all functioned as intended.

Section 6: Conclusion: Version Control, Critical Appraisal, Documentation (24 marks)

Project report:

Introduction

This project focuses on performing data analytics on the Pima Indians Diabetes Dataset from the National Institute of Diabetes and Digestive and Kidney Diseases. Diabetes is a chronic condition that affects how the body processes blood sugar, posing a significant public health challenge. The primary objective of this work is to apply Python programming concepts, including control structures, modular functions, data cleaning with Pandas, and visualisation, to identfy high risk patients and anomalies. Furthermore, a graphical user interface (GUI) was developed to allow users to interact with and derive insights from the dataset.

Methodology

The project followed a structured data science workflow divided into distinct stages.

A foundational analysis (Section 1) was conducted using for loops and nested if statements to traverse the dataset. This allowed for the calculation of the average BMI of high-risk patients (defined as those with glucose levels >140) and the identification of medical anomalies, such as diabetic patients with unusually low glucose levels. To ensure code reusability and maintainability, the system was built using modular functions

(Section 2) supported by four external libraries: Pandas for data handling, Numpy for numerical auditing, and matplotlib/seaborn for visualisation.

The data cleaning process (Section 3) involved several critical steps. Non-standard missing symbols were converted to standard NaN values and duplicate entries were dropped if there were any. Hidden missing values such as the 0s in columns like BMI and Glucose were replaced using median imputation to preserve the dataset's size. In order to cap extreme values (outliers), the Interquartile Range (IQR) method was used to cap extreme values, preventing skewed analysis results.

Finally, a user-friendly interface (Section 5) was built using the Tkinter library. It includes features for displaying summary statistics and buttons to trigger specific visualisations dynamically, as well as buttons to look at specific subsets.

Findings and Visualisations

The analysis revealed several significant insights into the diabetic population within the dataset:

- Glucose correlation: High-risk patients identified in section 1 had an average BMI of 34.93, suggesting a strong link between weight and elevated glucose. Visualisation via histograms and heatmaps confirmed that glucose (0.49) and BMI (0.31) have the highest positive correlations with a diabetic outcome.
- Hereditary Risk: The Violin plot of the DiabetesPedigreeFunction showed a thicker density at higher values for the diabetic group, indicating that hereditary factors play a significant role in disease status.
- The interactive scatter plot demonstrated that the majority of diabetic outcomes cluster at the intersection of high BMI and high glucose levels.

Diabetic patients exhibited a 28% increase in average glucose levels compared to the healthy group and had a higher skinfold thickness and higher average BMI.

Version control with GitHub

Effective version control is a cornerstone of modern software development and data science projects, ensuring traceability, collaboration, and management of code changes. For this project GitHub was utilised as the primary platform for version control, providing a centralised repository for the codebase. The project repository demonstrates a commit history, reflecting incremental development and iterative refinement of the code. Regular commits were made, each accompanied by a description that clearly articulates the purpose of the changes. This practice facilitates easy tracking of modifications, seamless rollback to previous stable versions if necessary and supports collaborative development by providing a clear record of contributions.

Critical Appraisal

Critical appraisal of the developed solution involves evaluating its strengths, weaknesses, and potential areas for improvement through systematic methods. Several aspects of critical appraisal were considered:

- Algorithmic Efficiency: Section 1 utilised iterrows() for data traversal. While effective for demonstrating control structures, this approach has O(n) time complexity. For larger healthcare datasets, vectorised operations in Pandas would be significantly more performant, as they leverage compiled C code.
- Data Imputation Bias: While median imputation preserved the sample size (768), it artificially reduces variance. This compression of data could lead to an underestimation of risk. Future work can use K-Nearest Neighbours (KNN) imputation for more nuanced data recovery.
- Layout Complexity: A limitation of the current GUI is the reliance on the .pack() geometry manager. While functional for this single-container view, Shipman (2013) notes that the .grid() manager is significantly more powerful for complex layouts, as it treats the window as a two-dimensional table. A future improvement would involve refactoring the dashboard to use a grid-based coordinate system, allowing for side-by-side comparisons of different patient subsets.

Documentation

Comprehensive documentation is integrated directly within the Jupyter notebook using Markdown cells. Each section of the notebook includes detailed explanations of the methodology, rationale for design choices and interpretation of results. Code comments are used to clarify complex logic. This approach ensures that the notebook serves as a self-contained report, making the code understandable and reproducible for both technical and non-technical audiences.

Conclusion

Python programming fundamentals were successfully integrated with data analytics to explore a significant public health issue. By transitioning from basic control structures to a modular, GUI-integrated application, the work demonstrates the progression from raw code to a functional software solution. The findings highlight the critical roles of BMI and glucose as risk indicators. The critical appraisal identifies clear pathways for scaling the application for larger, more complex datasets.

References:

Harris, C.R. et al. (2020) 'Array programming with NumPy', Nature, 585(7825), pp. 357-362. doi: 10.1038/s41586-020-2649-2. (Accessed: 18 December 2025).

Lundh, F. (1999) An Introduction to Tkinter. PythonWare. Available at: http://www.pythonware.com/library/tkinter/introduction/ (Accessed: 01 January 2026).

National Institute of Diabetes and Digestive and Kidney Diseases Pima Indians Diabetes Database. Available at: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database (Accessed: 25 November 2025).

Shipman, J.W. (2013) Tkinter 8.5 reference: a GUI for Python. New Mexico Tech Computer Center. Available at: https://tkdocs.com/shipman/ (Accessed: 01 January 2026).

Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C. and Johannes, R.S. (1988) 'Using the ADAP learning algorithm to forecast the onset of diabetes mellitus', Proceedings of the Symposium on Computer Applications and Medical Care, p. 261. Available at: https://pmc.ncbi.nlm.nih.gov/articles/PMC2245318/ (Accessed: 25 November 2025).

Appendix:

(If any)