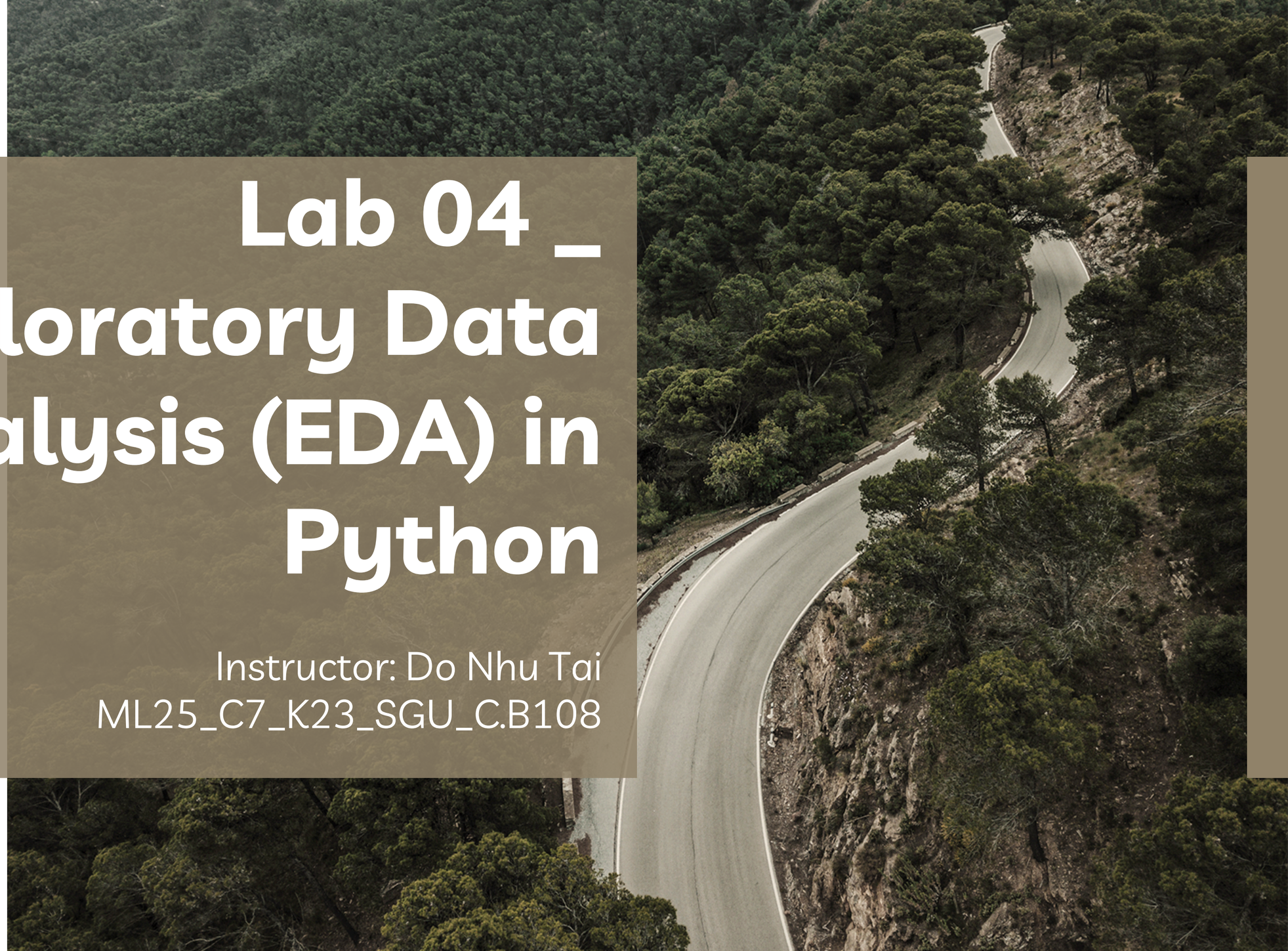


Lab 04 _ Exploratory Data Analysis (EDA) in Python

Instructor: Do Nhu Tai
ML25_C7_K23_SGU_C.B108

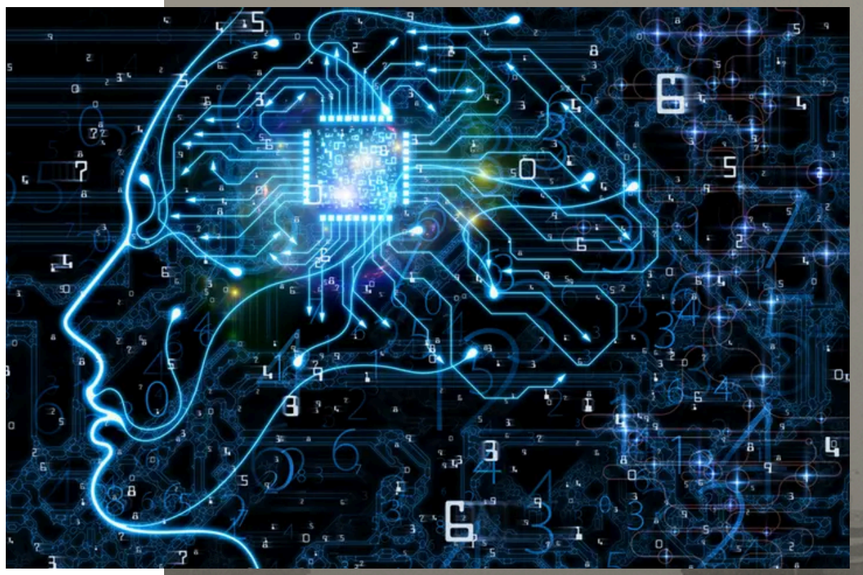
Task Assignment Table

Student ID	Full Name	Task
3123410386	Nguyen Duong Bao Tran (leader)	Slide Report, Survey, Code (Suspicious Zeros and Missing Values + Outlier Detection + Graphical EDA)
3123410371	Le Thi Thuy Tien	Slide Report, Read, Code (Univariate Analysis + Target Variable Distribution)
3123410385	Truong Thi Huyen Trang	Slide Report, Read, Code (Multivariate Analysis + Class-wise Summary)
3123410421	Giang Hao Tuong	Slide Report, Survey, Code (Missing Value Treatment + Heatmap)

Introduction

The dataset contains medical diagnostic measurements for 768 female patients of Pima Indian heritage.

The goal is to predict whether a patient has diabetes (binary outcome).



The features present in the dataset are:

Pregnancies

Number of times pregnant

Glucose

Plasma glucose concentration (mg/dL)

BloodPressure

Diastolic blood pressure (mm Hg)

SkinThickness

Triceps skinfold thickness (mm)

Insulin

2-Hour serum insulin (μU/mL)

BMI

Body mass index (weight in kg / (height in m)^2)

DiabetesPedigreeFunction

A function which scores likelihood of diabetes based on family history

Age

Age in years

The target variable is:

Outcome: Indicates whether the patient has diabetes (0 = No, 1 = Yes).

Data Import & Setup

- **Used Python with libraries:** pandas, numpy, matplotlib, seaborn.
- **Loaded the Pima Indians Diabetes dataset** from CSV file (768 rows × 9 columns).
- **Manually assigned column names** (Pregnancies, Glucose, BloodPressure, ..., Outcome).
- **Outcome** = target variable (0 = non-diabetic, 1 = diabetic)

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

1. Quick Overview of the Dataset

```
df.shape
```

```
(768, 9)
```

The dimensions of the dataset.

```
df.dtypes
```

```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

data type of each column in dataset

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

5 first rows of dataset

1. Quick Overview of the Dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

A concise summary of the dataset.

```
df.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

The number of missing values (NaN) in each column of the dataset.

1. Quick Overview of the Dataset

What are the stats of the variables?

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

1. Quick Overview of the Dataset

Are the classes (Outcome) balanced?

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

```
Outcome
0    500
1    268
Name: count, dtype: int64
```

Correlation Between Variables

```
df.corr()
```

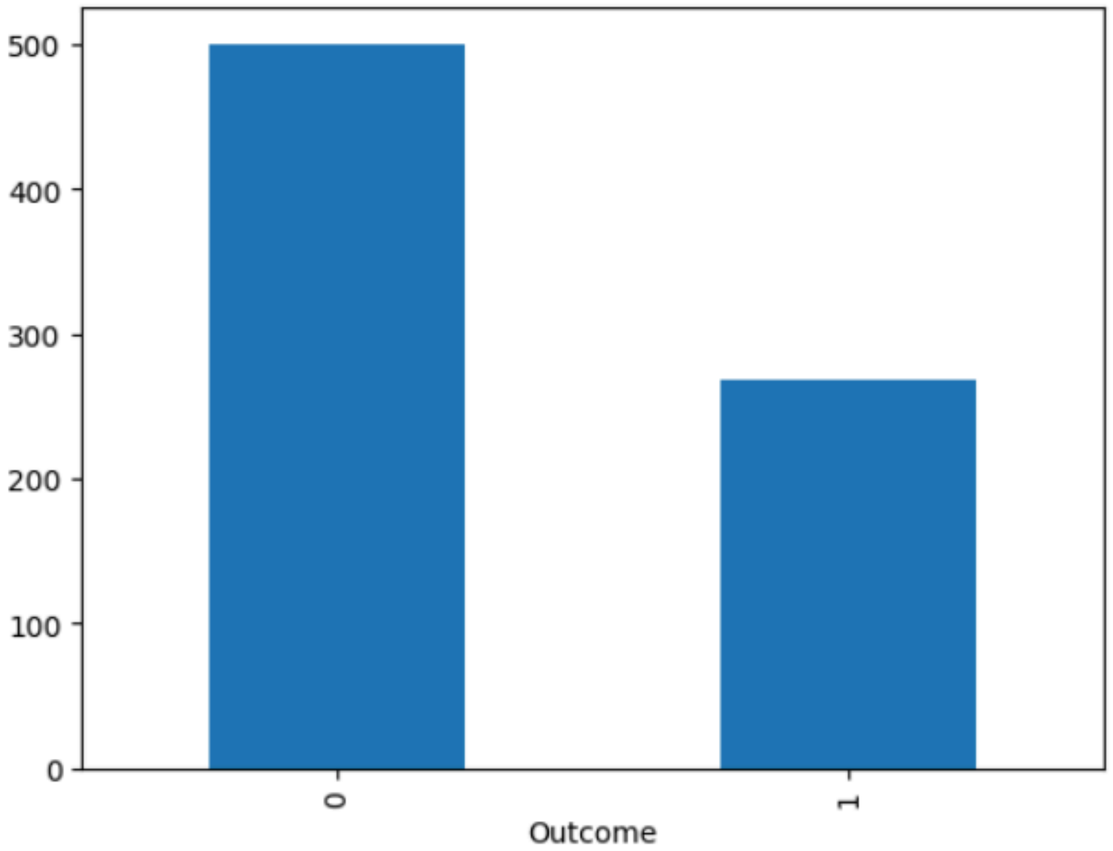
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242
iabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356

Graphical EDA

Using pandas

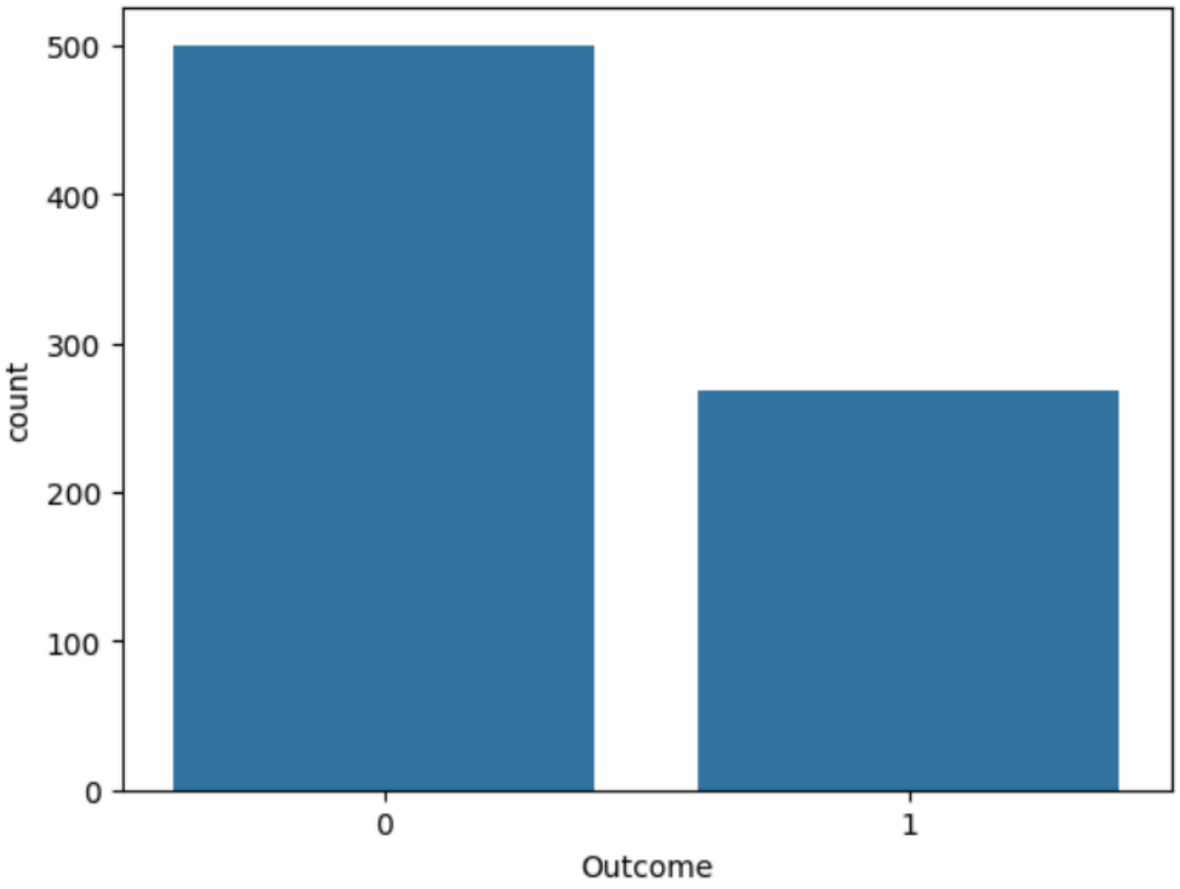
```
df["Outcome"].value_counts().plot.bar()
```

```
<Axes: xlabel='Outcome'>
```



Using seaborn

```
sns.countplot(x = df['Outcome']);
```

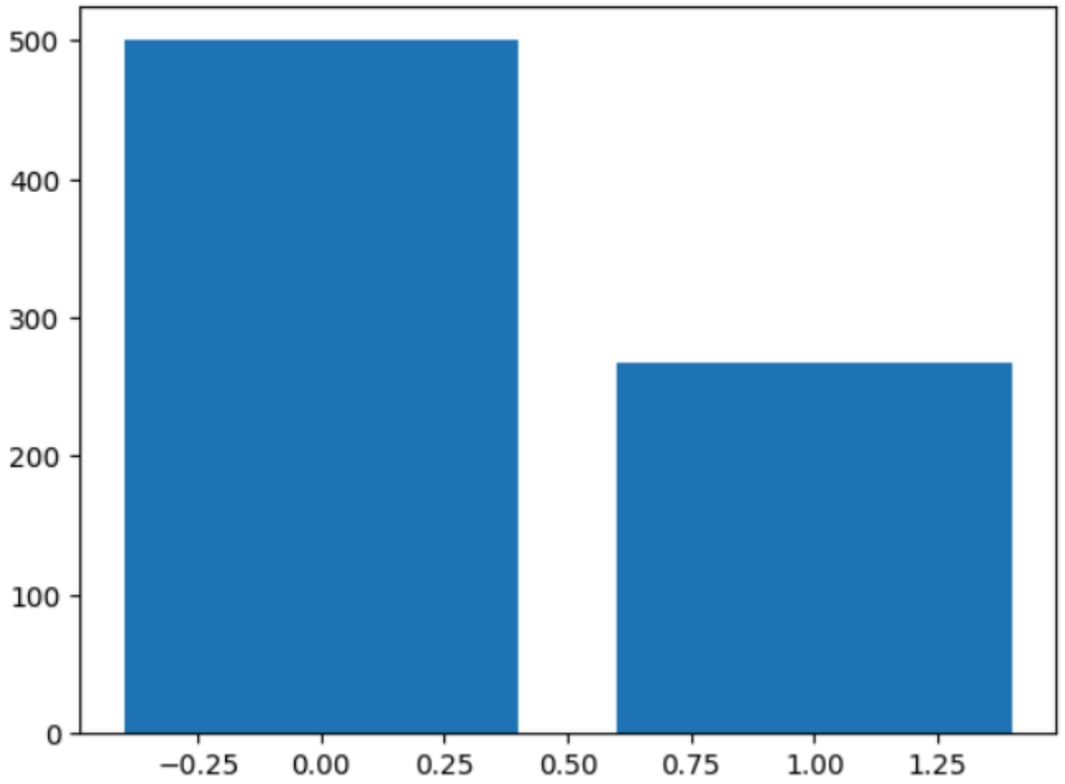


Graphical EDA

Using matplotlib

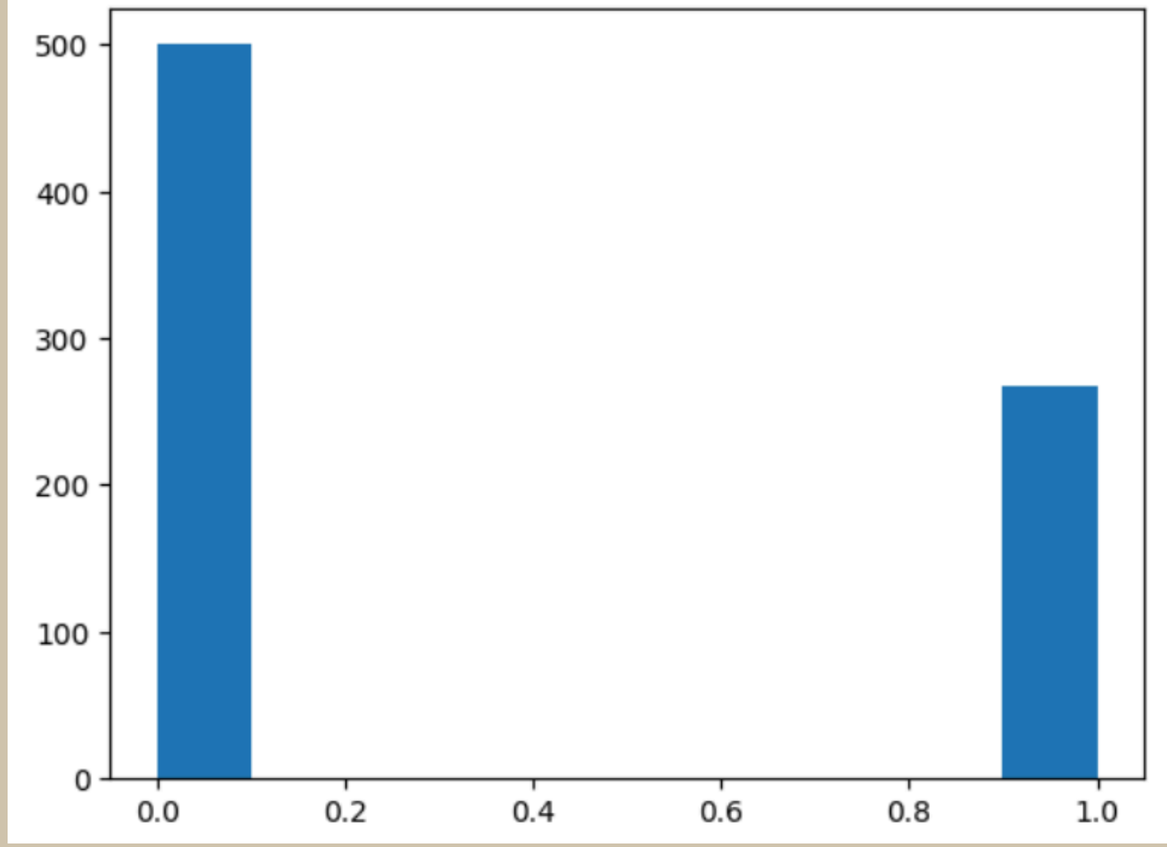
```
#we can use bar chart
counts = df["Outcome"].value_counts()
plt.bar(counts.index, counts.values)
```

```
<BarContainer object of 2 artists>
```




```
#one way using histogram :)
plt.hist(df['Outcome'])
```

```
(array([500.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 268.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <BarContainer object of 10 artists>)
```



What we can conclude:

- The classes are **not perfectly balanced** (500 patients without diabetes vs 268 patients with diabetes).

- This means classification accuracy alone may be misleading; additional metrics such as precision, recall, and F1-score should also be considered.

- No extreme imbalance handling (such as heavy oversampling or augmentation) is strictly required, but techniques like SMOTE or class weighting could improve model performance.

- ...

Comparison graphs

Scatterplots

What?

- A scatterplot uses a Cartesian coordinate system to display values of two variables for a set of data points.

Why?

- It allows us to visualize the relationship between two variables and how they separate across the target classes.

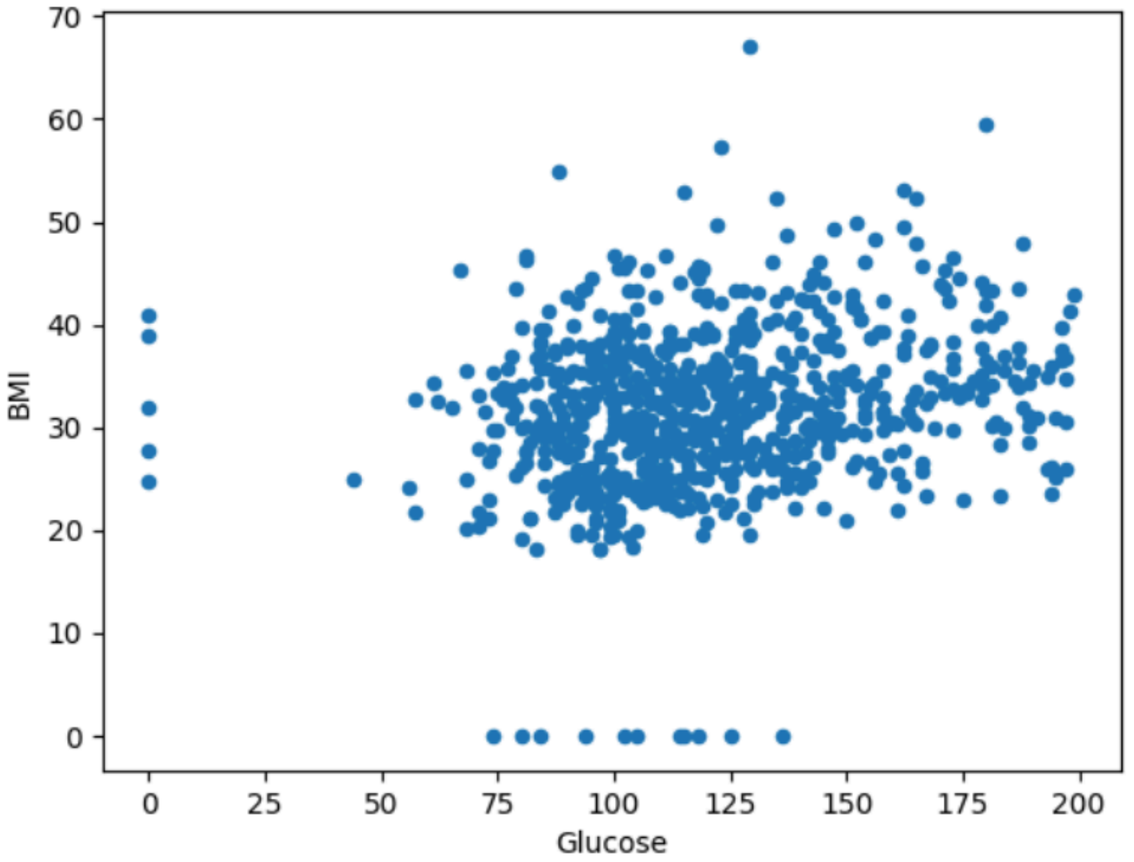
Example:

- Scatterplot between **Glucose** and **BMI**, colored by the target variable *Outcome*.

Scatterplot using pandas

```
df.plot.scatter('Glucose','BMI')
```

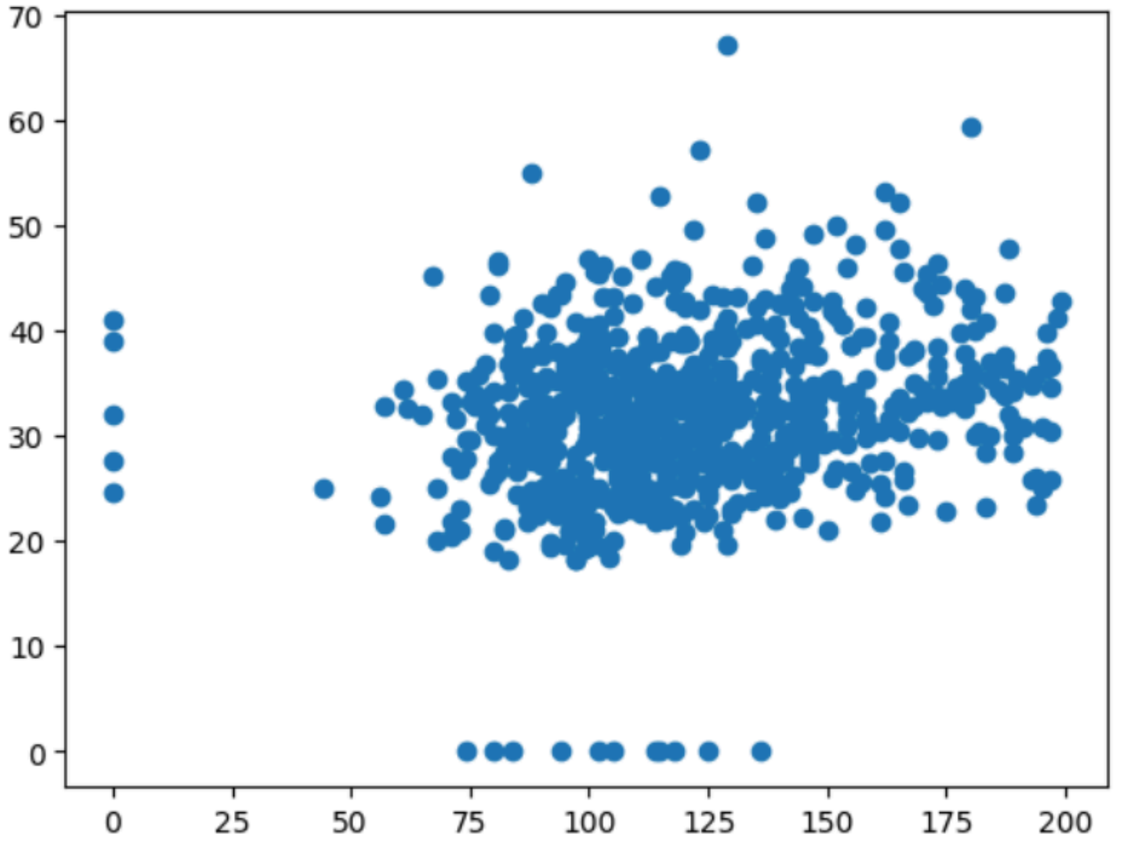
```
<Axes: xlabel='Glucose', ylabel='BMI'>
```



Scatterplot using matplotlib

```
plt.scatter('Glucose', 'BMI', data=df)
```

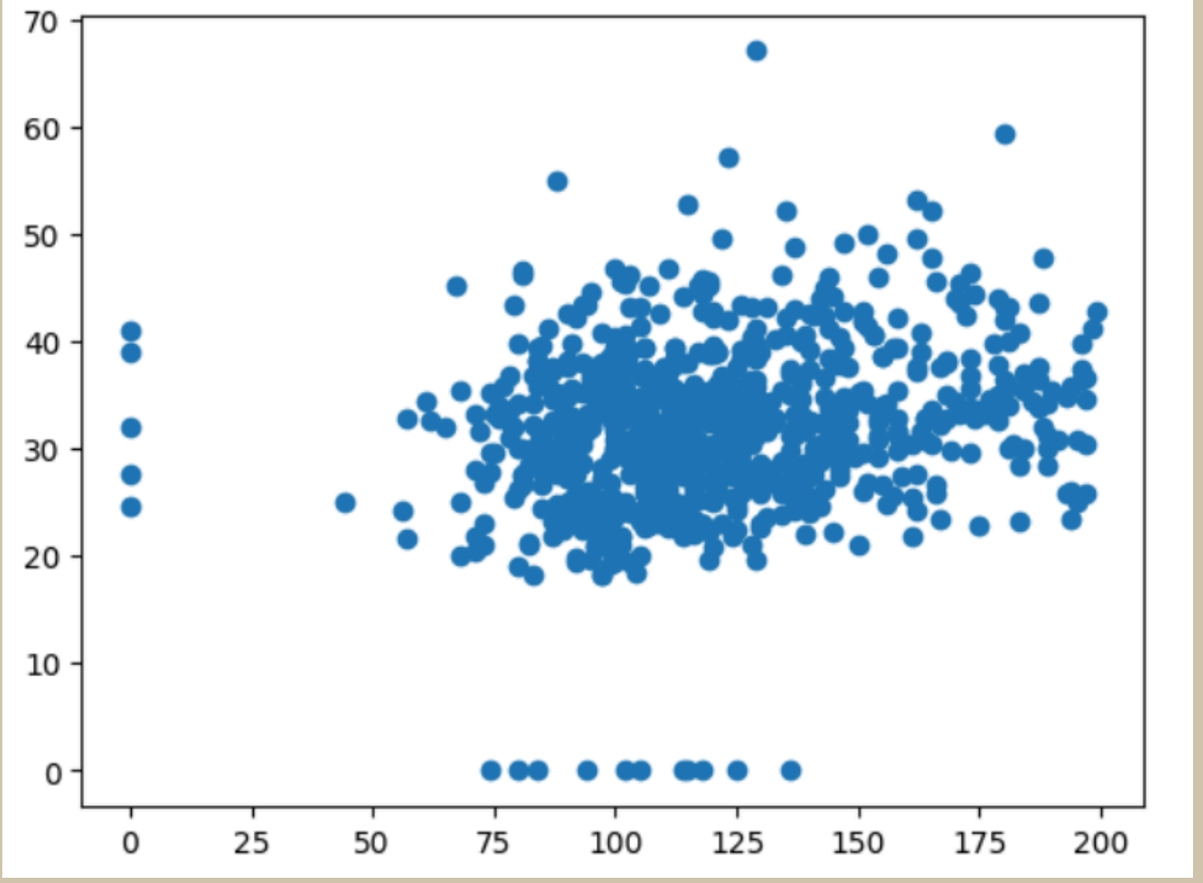
```
<matplotlib.collections.PathCollection at 0x23f0a9e5310>
```



Scatterplot using matplotlib (continue)

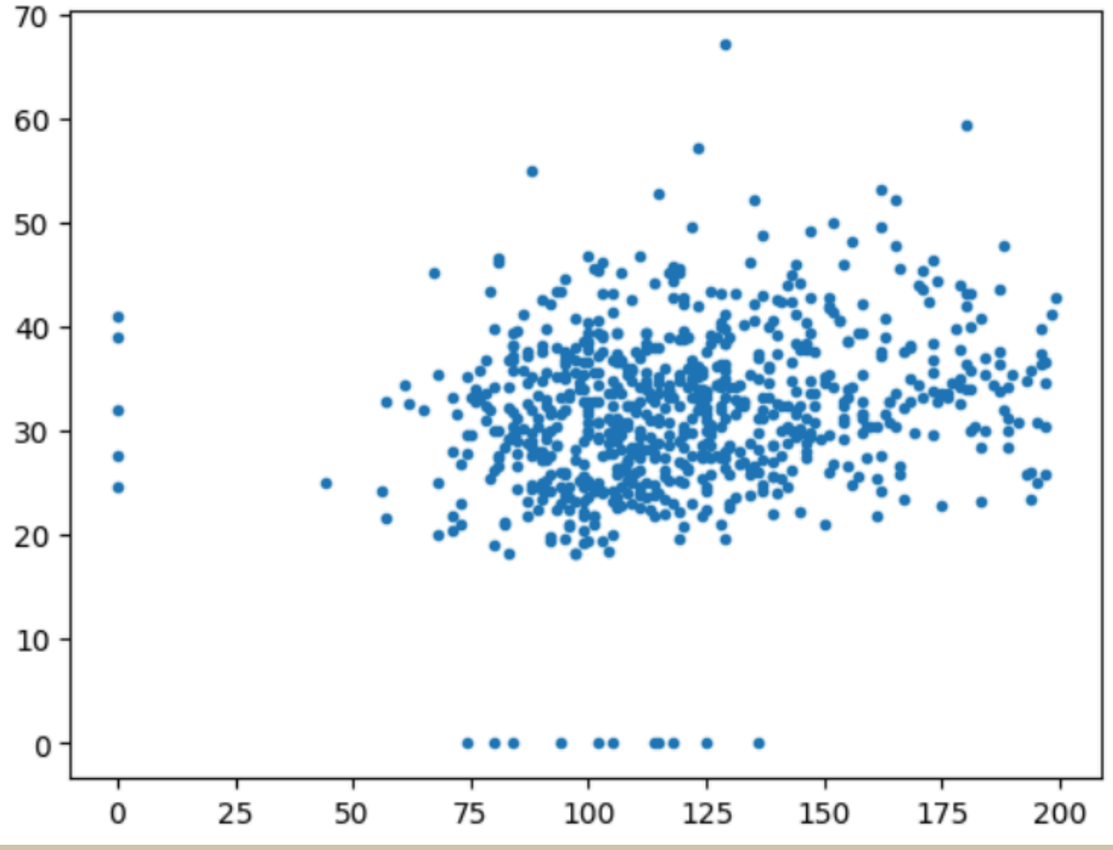
```
plt.scatter(df['Glucose'], df['BMI'])
```

```
<matplotlib.collections.PathCollection at 0x23f0ba75450>
```




```
#using plt
plt.plot('Glucose', 'BMI', data=df, marker='.',linestyle='none')
```

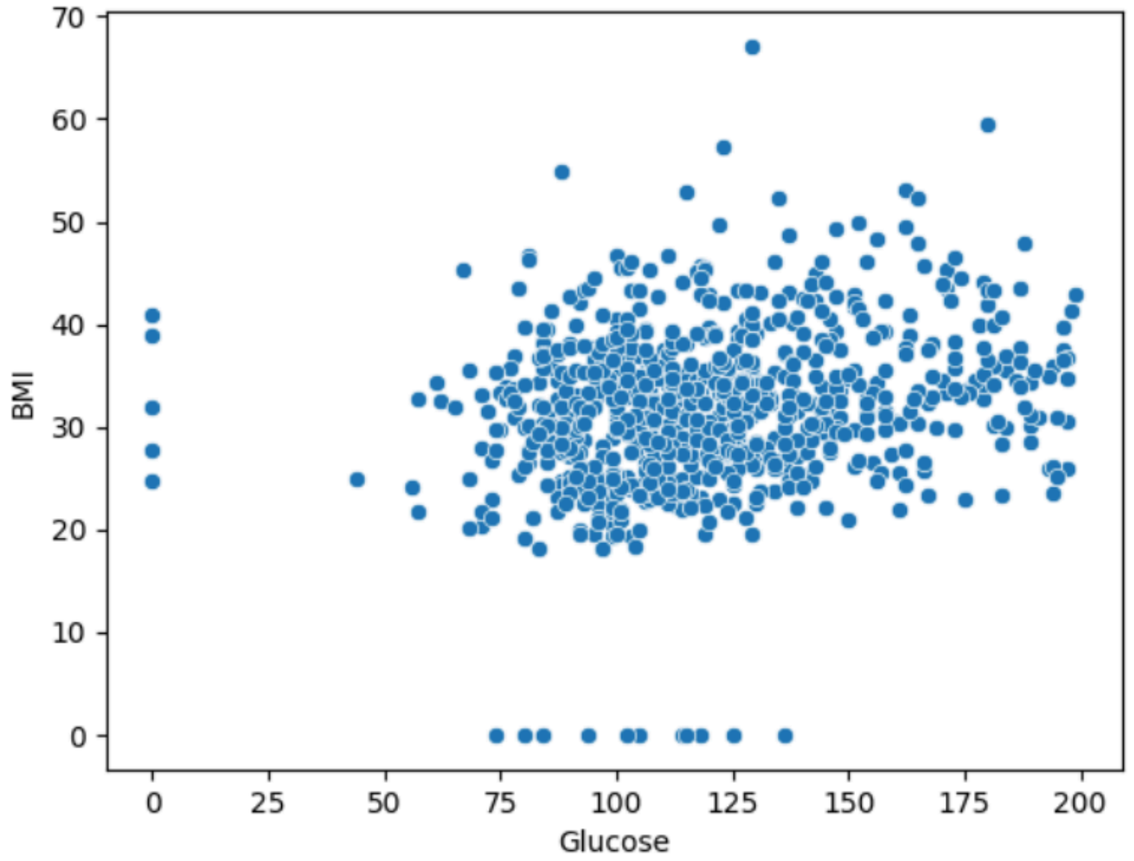
```
[<matplotlib.lines.Line2D at 0x23f0bafd950>]
```



Scatterplot using seaborn

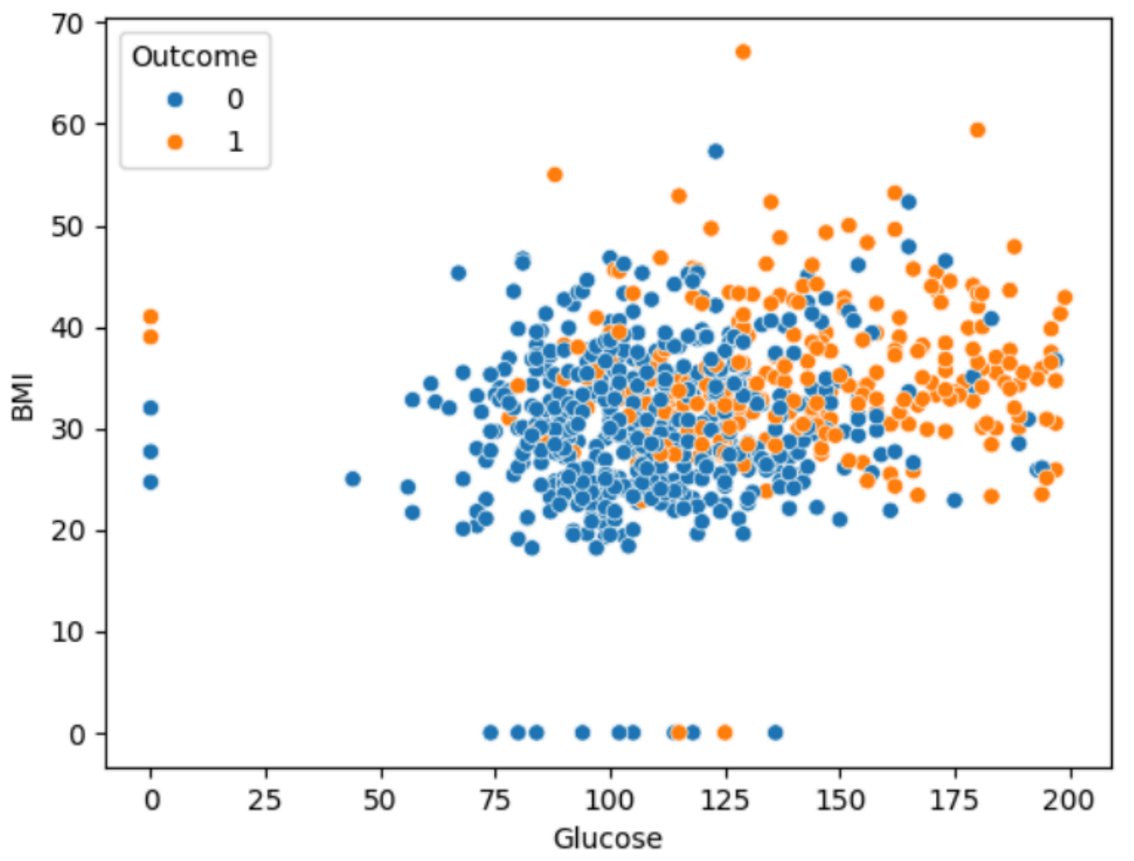
```
sns.scatterplot(x = df['Glucose'], y = df['BMI'])
#or sns.scatter('Glucose', 'BMI', data=df)
```

```
<Axes: xlabel='Glucose', ylabel='BMI'>
```




```
# With respect to the target class (Outcome)
sns.scatterplot(x = 'Glucose', y = 'BMI', hue=df['Outcome'], data=df)
```

```
<Axes: xlabel='Glucose', ylabel='BMI'>
```



What we can conclude:

- Patients with diabetes (**Outcome = 1**) tend to have **higher Glucose levels** compared to non-diabetic patients.

- BMI also shows higher median values in the diabetic group.

- Age plays a role: older patients are more likely to have diabetes.

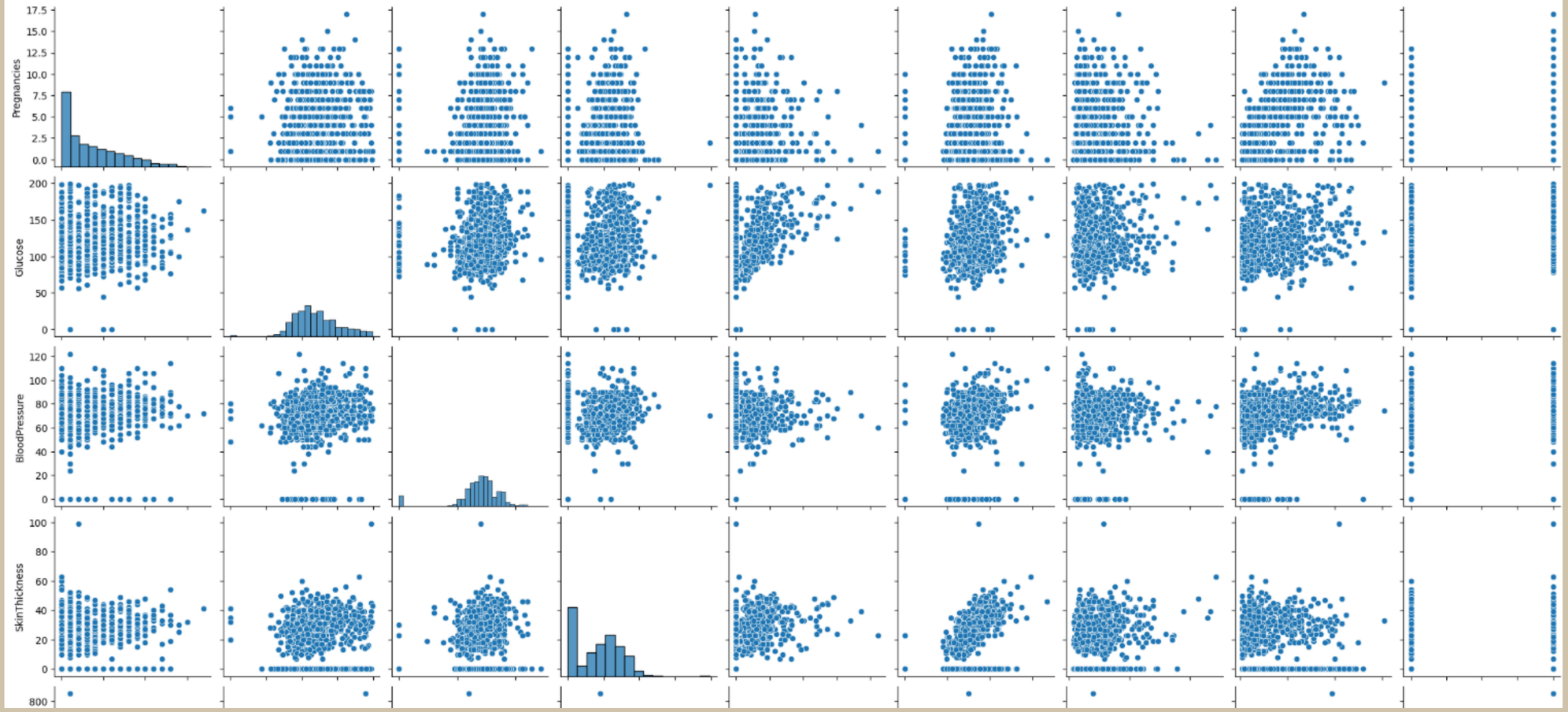
As seen above, different visualization tools can be used. From now on, only **Seaborn** plots will be shown, while you can explore other libraries on your own.

Correlogram

A correlogram (also known as a pairplot or correlation matrix) is used to analyze the relationship between each pair of numeric variables.

```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x23f087238c0>
```



```
sns.pairplot(df,hue="Outcome")
```

```
<seaborn.axisgrid.PairGrid at 0x23f102b82d0>
```

What we can conclude:

- Two main clusters can be observed when comparing **Glucose** and **BMI**.

- Patients with **Outcome = 0 (no diabetes)** generally have lower glucose levels and moderate BMI.

- Patients with **Outcome = 1 (diabetes)** tend to have higher glucose levels, and often higher BMI values as well.

- Age also contributes: older patients are more likely to fall into the diabetic cluster.

- Overall, the diabetic group shows distinct patterns in **Glucose**, **BMI**, and **Age**, while the non-diabetic group has lower values across these features.

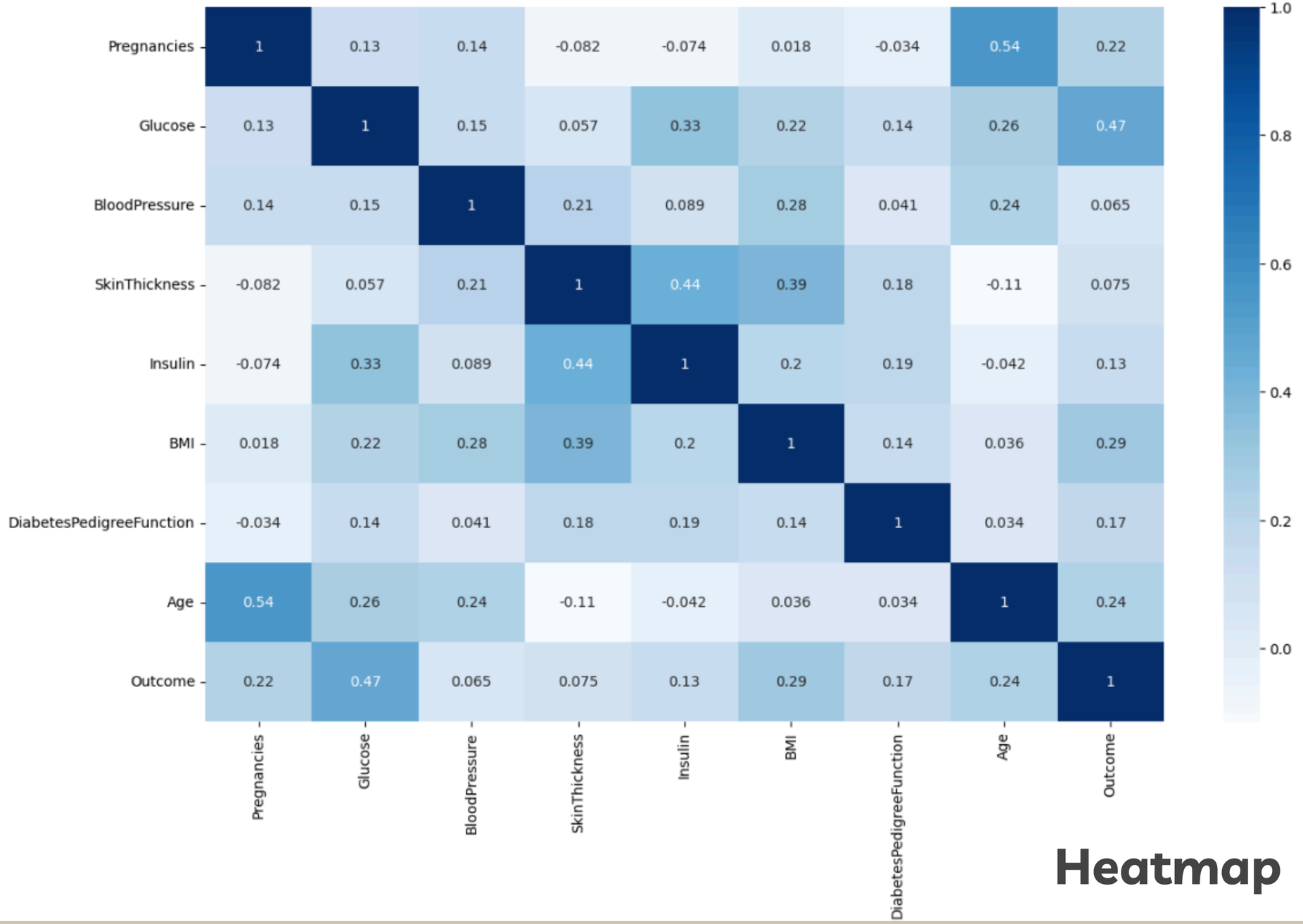
Heatmap

Further, we can visualize the correlation between numeric features using a **heatmap**.

A heatmap is a two-dimensional graphical representation of data where the individual values in a matrix are represented as colors.

In this context, the heatmap helps us quickly identify which features are strongly correlated with each other and with the target variable **Outcome**.

```
fig = plt.figure(figsize = (15,9))
sns.heatmap(df.corr(), cmap='Blues', annot = True);
```



Heatmap

Heatmap

From the above heatmap, we can observe the following correlations:

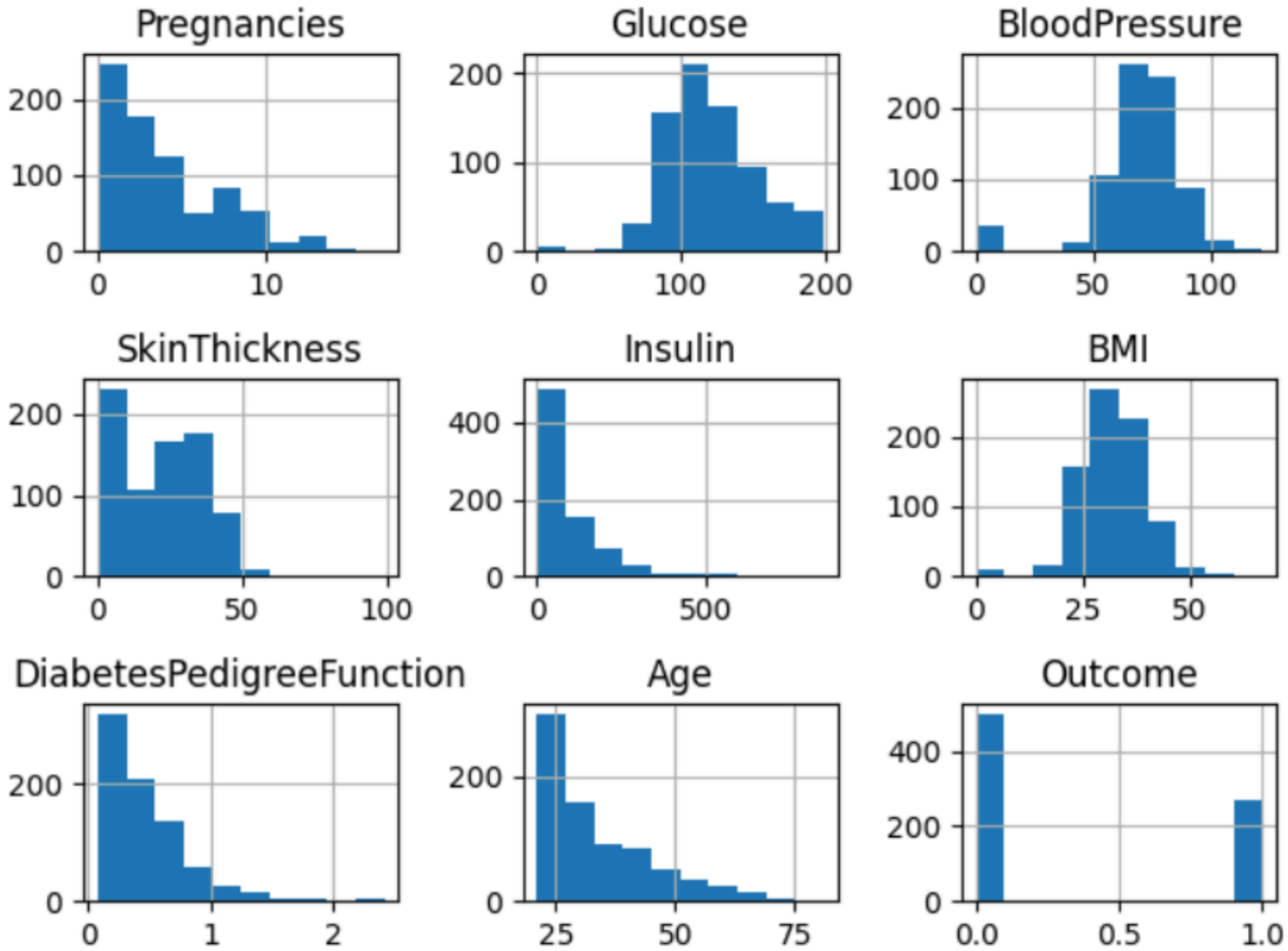
- **Glucose** has the strongest positive correlation with the target variable **Outcome**.

- **BMI** and **SkinThickness** show a moderate correlation, which makes sense since both relate to body composition.

- **Age** and **Pregnancies** are also positively correlated, as older women are more likely to have more pregnancies.

Histograms and distributions

```
df.hist()
plt.tight_layout()
plt.show()
```



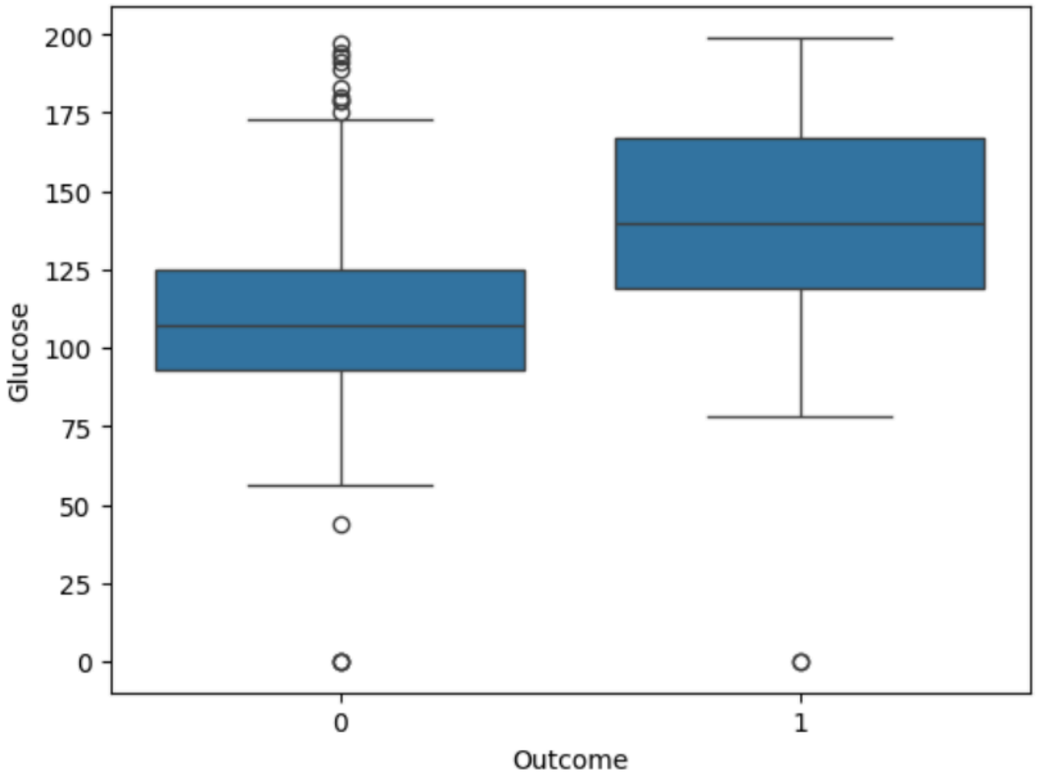
Most variables are skewed, not normally distributed.

Boxplots

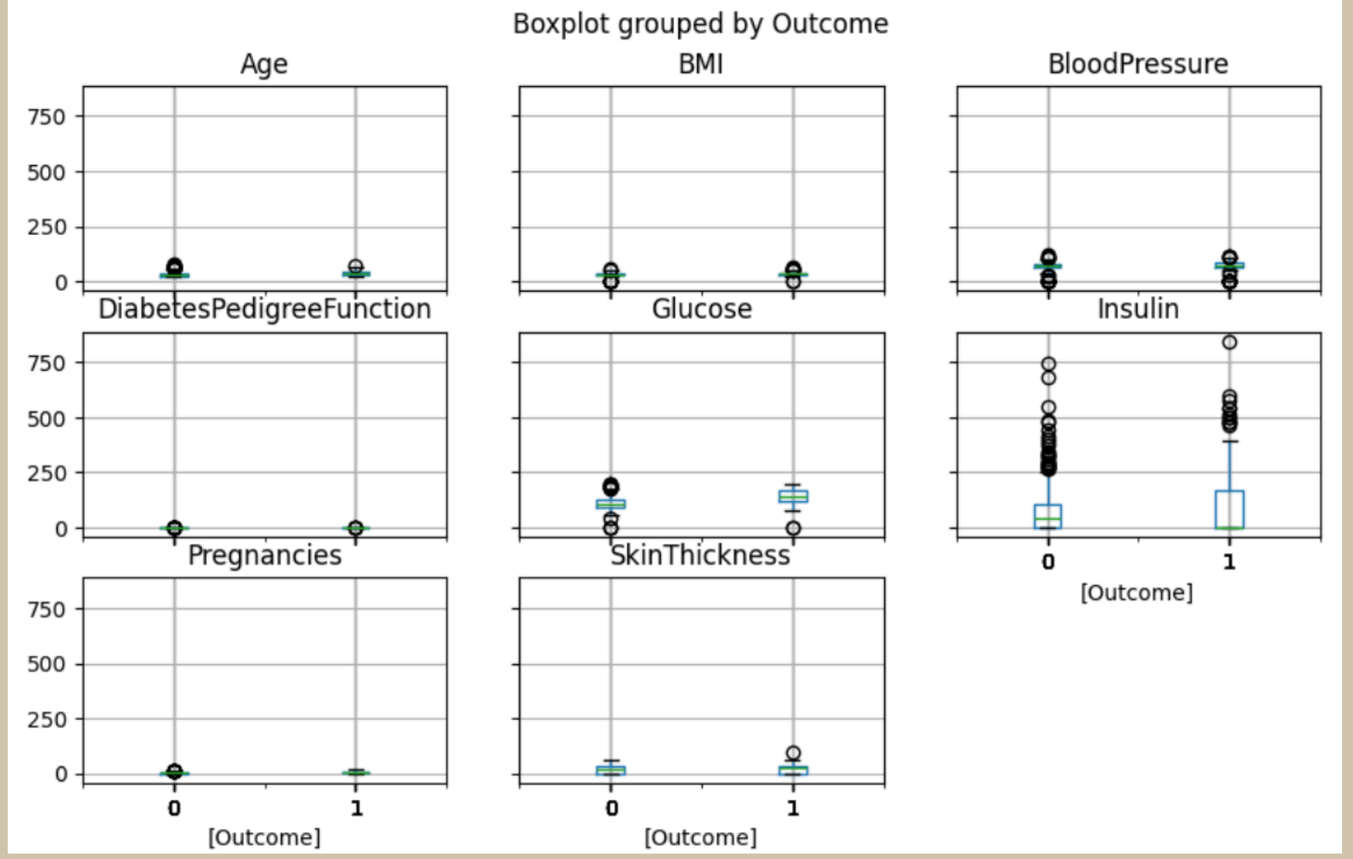
Boxplots and violins

```
sns.boxplot(x='Outcome', y='Glucose', data=df)
```

```
<Axes: xlabel='Outcome', ylabel='Glucose'>
```



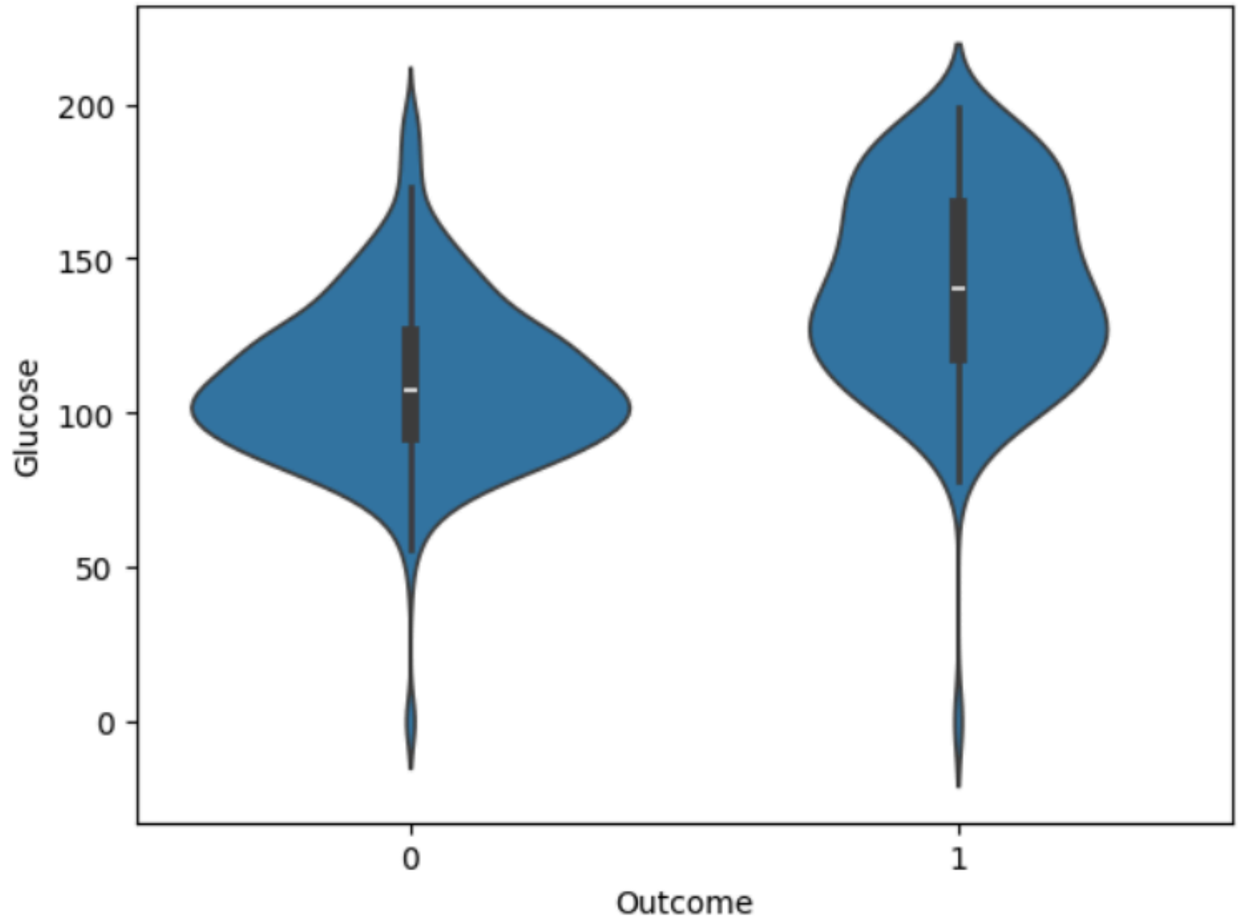

```
df.boxplot(by="Outcome", figsize=(10, 6));
```



Violins

```
sns.violinplot(x='Outcome', y='Glucose', data=df)
```

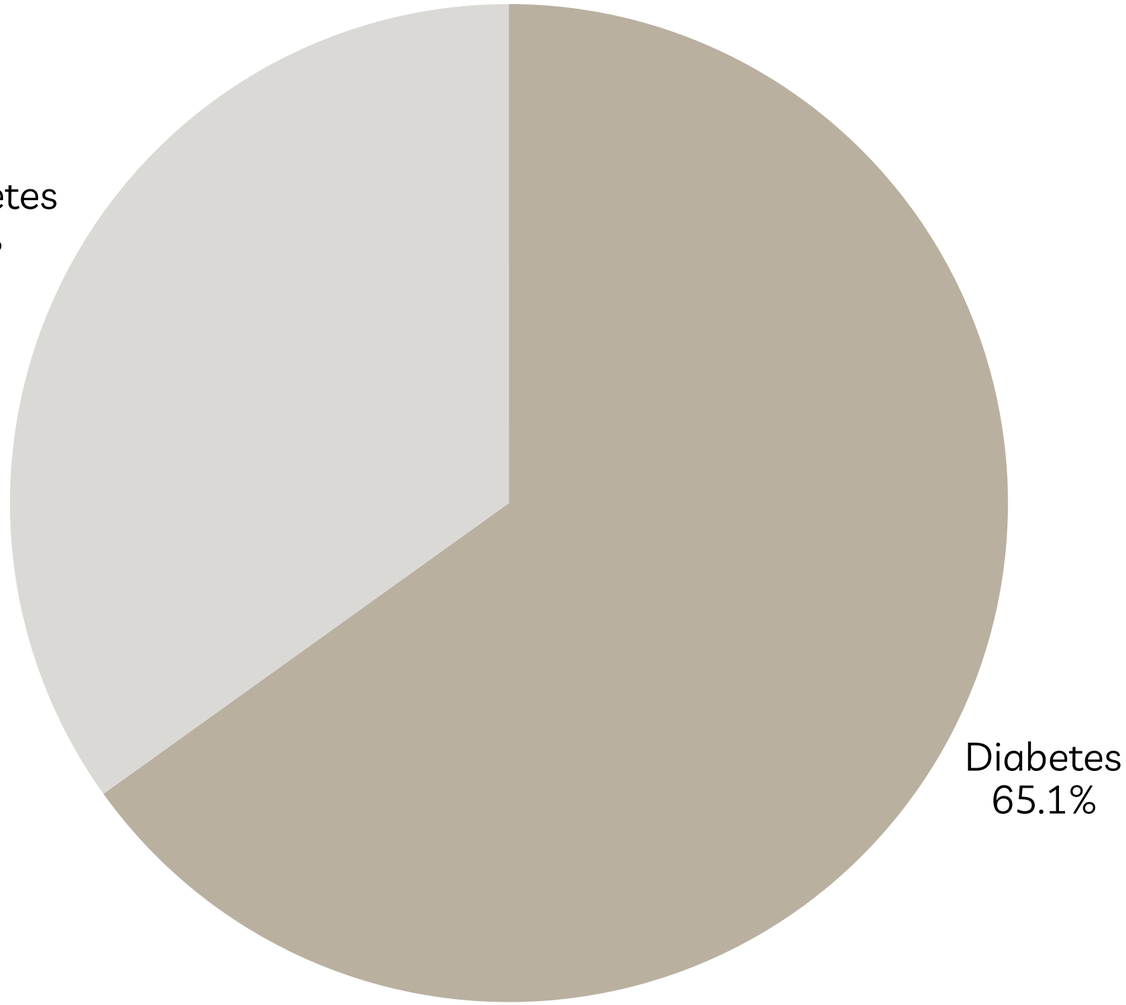
```
<Axes: xlabel='Outcome', ylabel='Glucose'>
```



2. Target Variable Distribution

Check class balance of the outcome variable (0 = no diabetes, 1 = diabetes)

Most patients are non-diabetic (65%), while diabetic cases account for 35%.



- **Label detected:** Outcome
- **Class distribution:**
  - Class 0 (No Diabetes): 500 samples (65.1%)
  - Class 1 (Diabetes): 268 samples (34.9%)

3. Suspicious Zeros and Missing Values

Problem: Some features (Glucose, BloodPressure, SkinThickness, Insulin, BMI) cannot realistically be 0.

- **Step 1:** Check suspicious zeros.
- **Step 2:** Replace suspicious zeros with NaN.

Suspicious Zeros Summary:

Feature	Zeros	Percentage (%)
Glucose	0	0.0
BloodPressure	0	0.0
SkinThickness	0	0.0
Insulin	0	0.0
BMI	0	0.0
Age	0	0.0

Missing Values After Replacement:

Feature	Missing Values
Glucose	5
BloodPressure	35
SkinThickness	227
Insulin	374
BMI	11

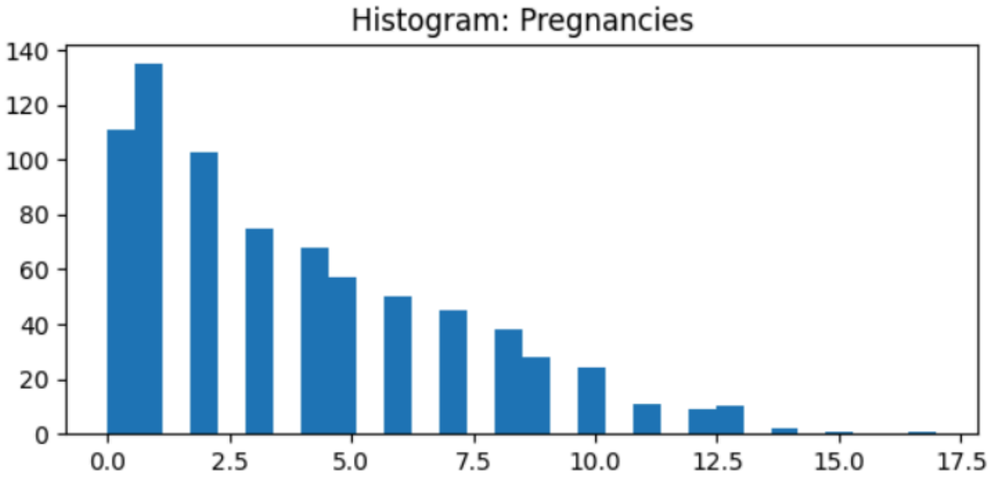
4. Univariate Analysis

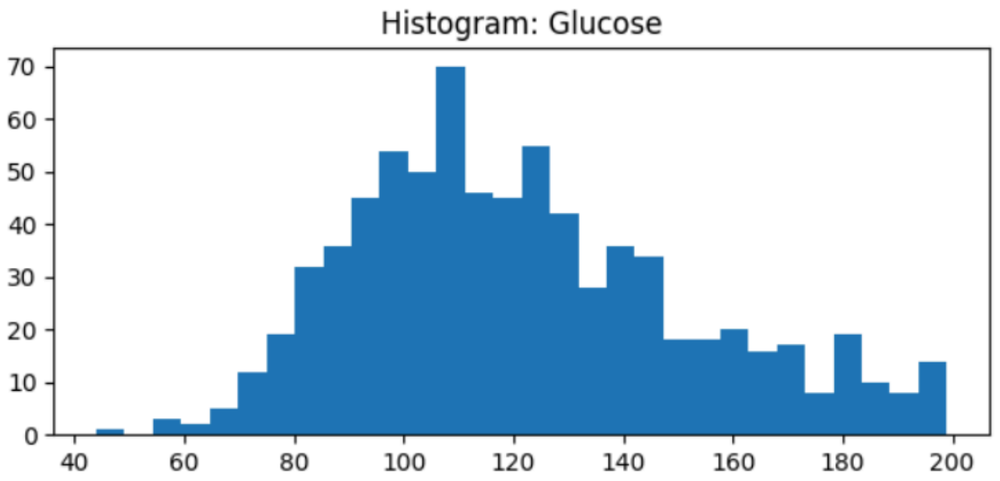
Histograms to observe feature distributions

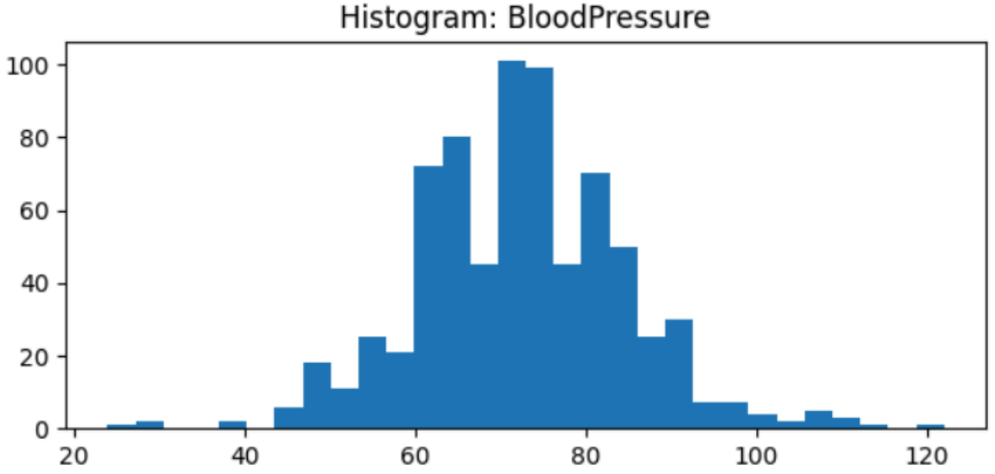
Descriptive stats including missing values

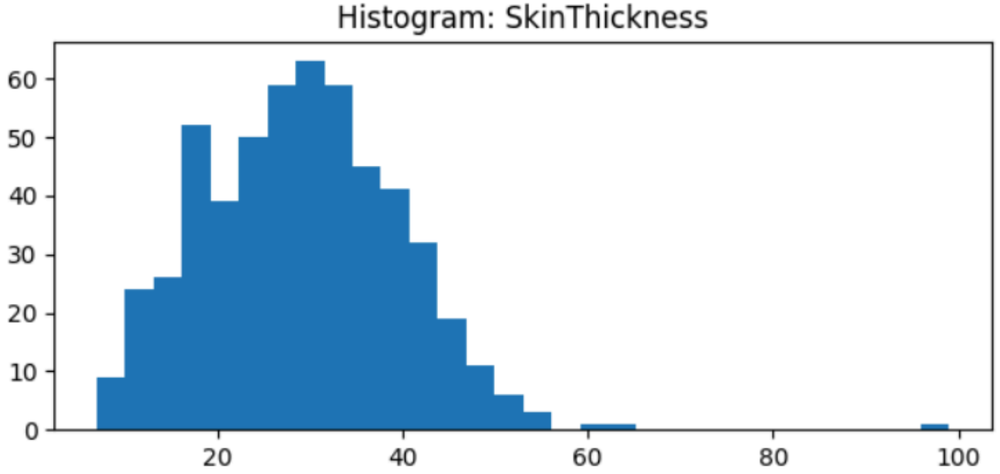
Boxplots to detect outliers

4. Univariate Analysis

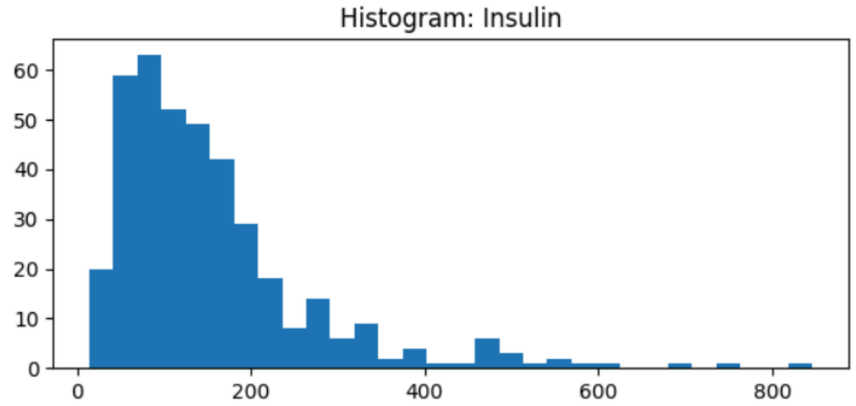


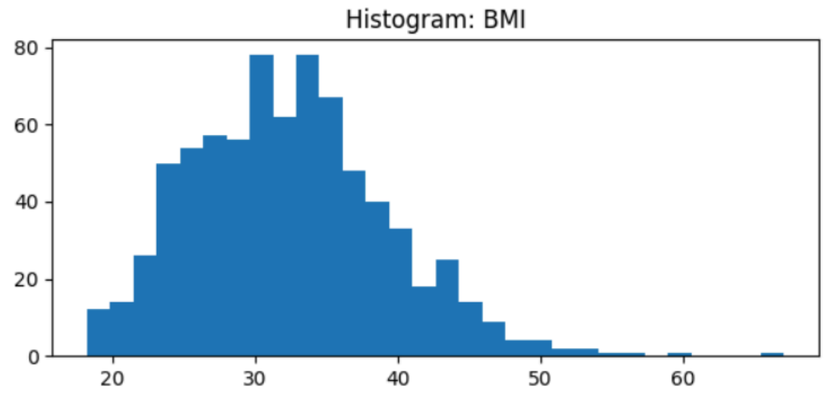


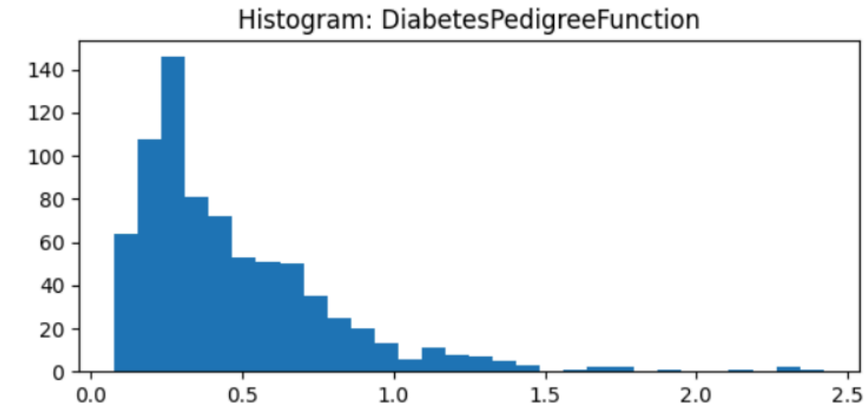


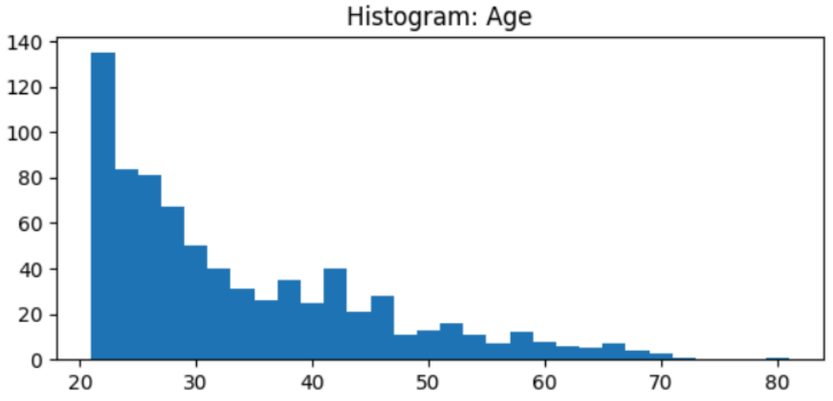


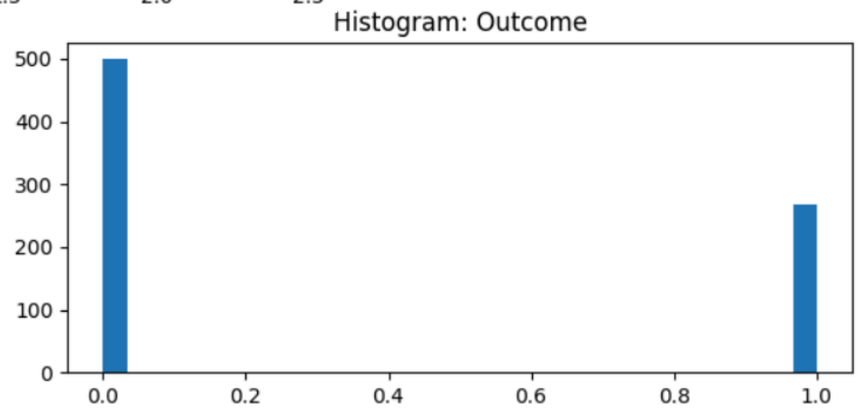
4. Univariate Analysis









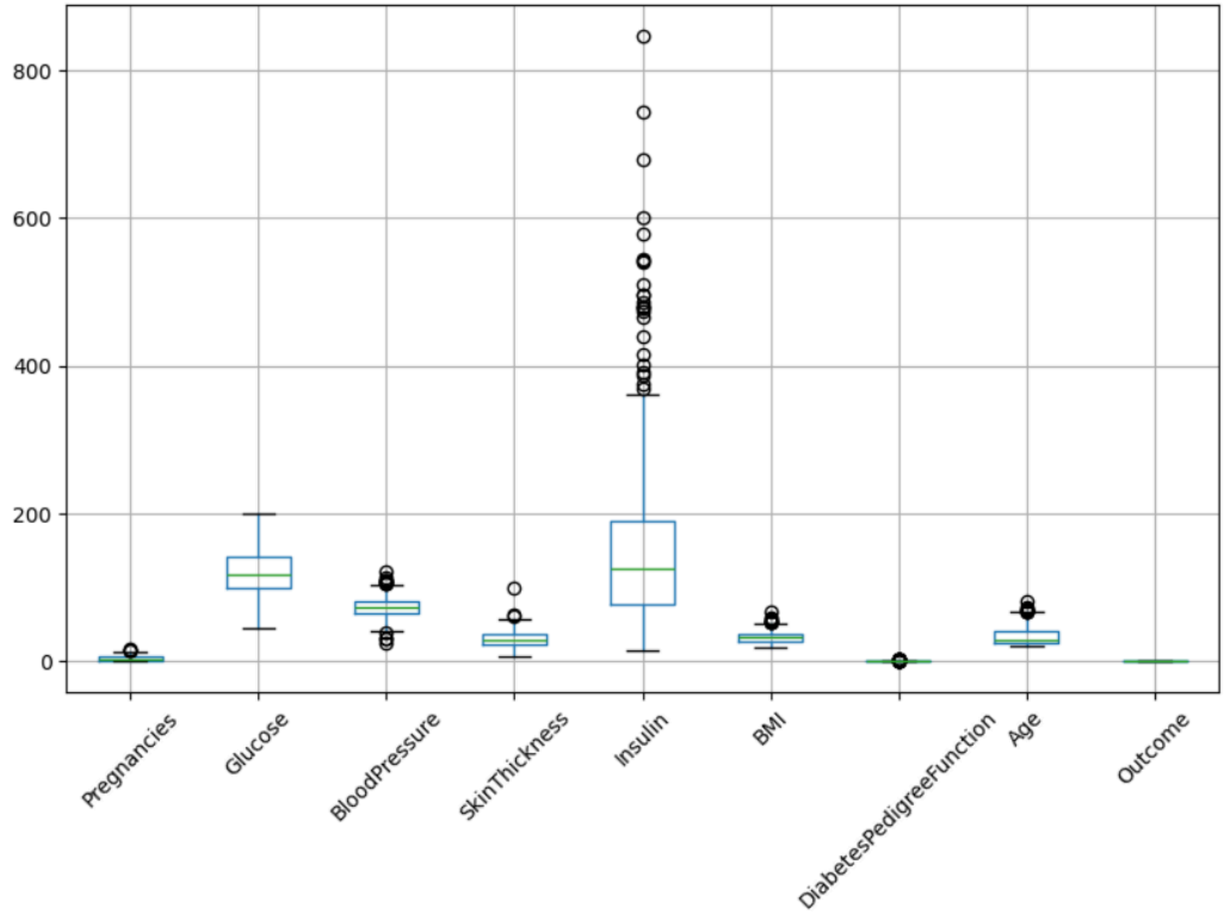


4. Univariate Analysis

Descriptive Statistics with Missing Values:

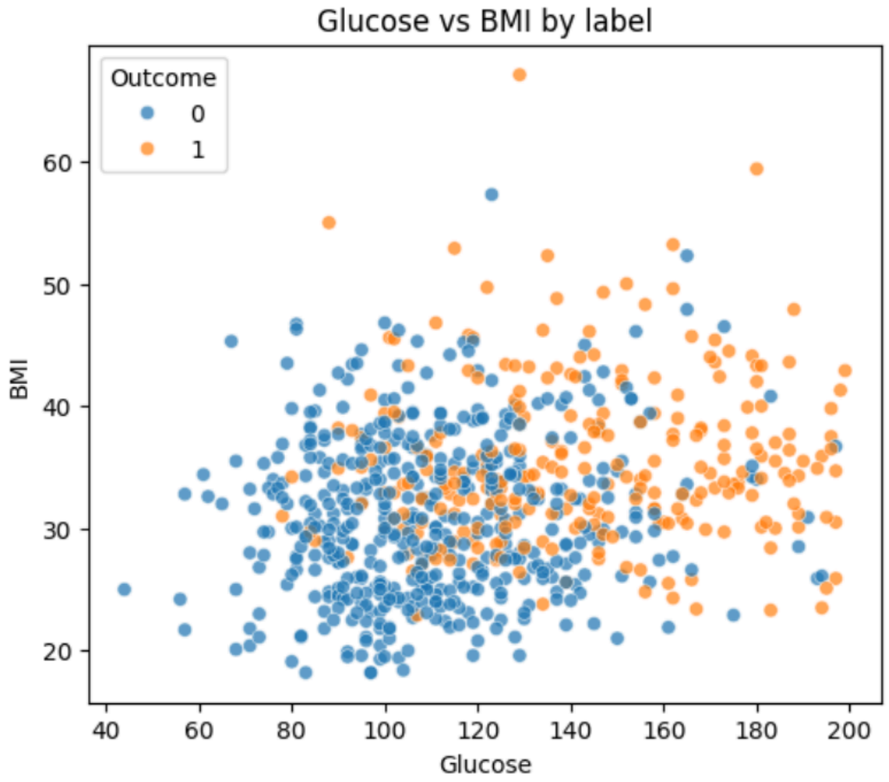
	count	mean	std	min	25%	50%	75%	max	missing	missing(%)
Pregnancies	768.0	3.85	3.37	0.00	1.00	3.00	6.00	17.00	0	0.00
Glucose	763.0	121.69	30.54	44.00	99.00	117.00	141.00	199.00	5	0.65
BloodPressure	733.0	72.41	12.38	24.00	64.00	72.00	80.00	122.00	35	4.56
SkinThickness	541.0	29.15	10.48	7.00	22.00	29.00	36.00	99.00	227	29.56
Insulin	394.0	155.55	118.78	14.00	76.25	125.00	190.00	846.00	374	48.70
BMI	757.0	32.46	6.92	18.20	27.50	32.30	36.60	67.10	11	1.43
DiabetesPedigreeFunction	768.0	0.47	0.33	0.08	0.24	0.37	0.63	2.42	0	0.00
Age	768.0	33.24	11.76	21.00	24.00	29.00	41.00	81.00	0	0.00
Outcome	768.0	0.35	0.48	0.00	0.00	0.00	1.00	1.00	0	0.00

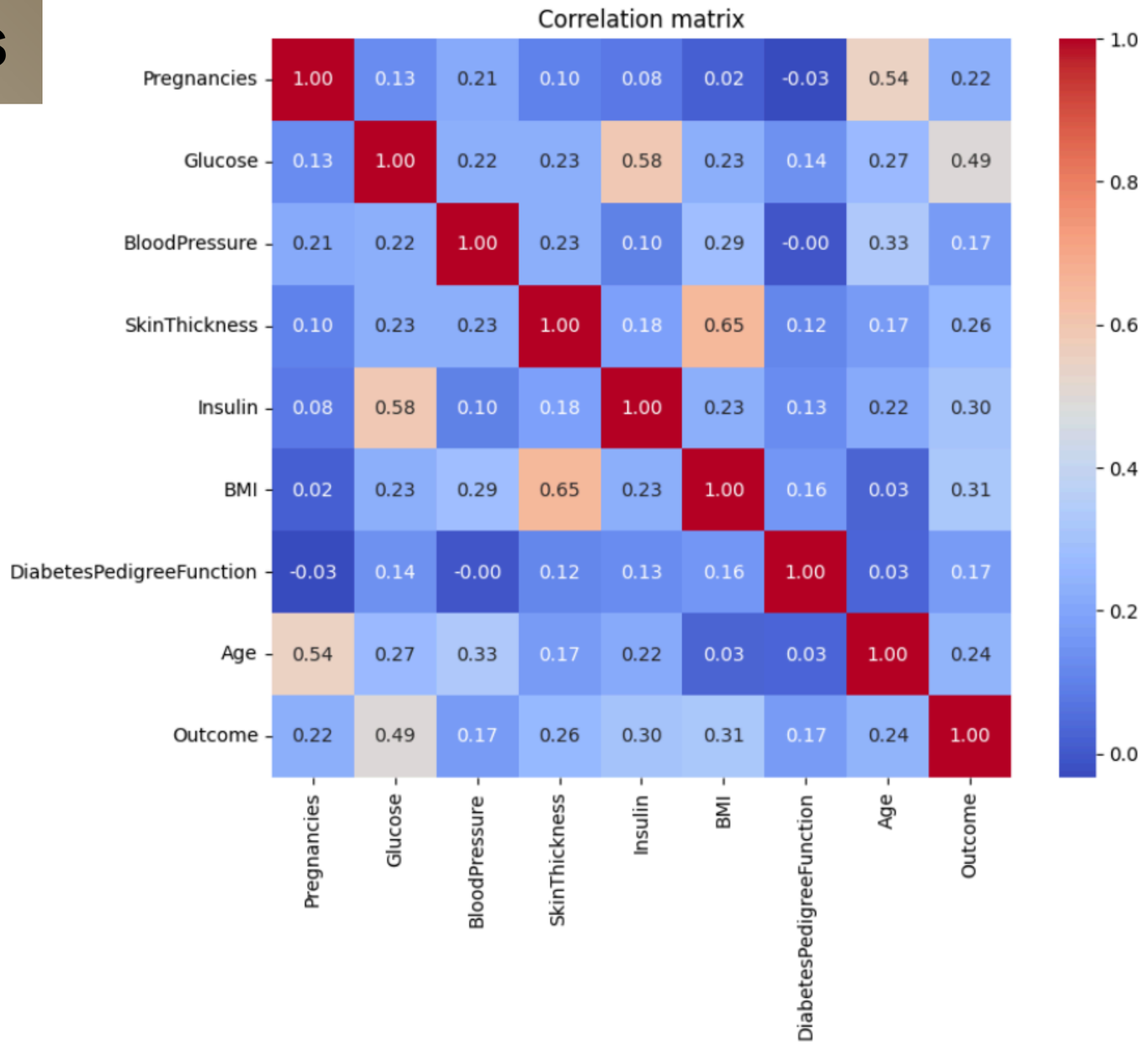
Boxplots - numeric:



5. Multivariate Analysis

- Correlation matrix
- Scatter plot of Glucose vs BMI by Outcome



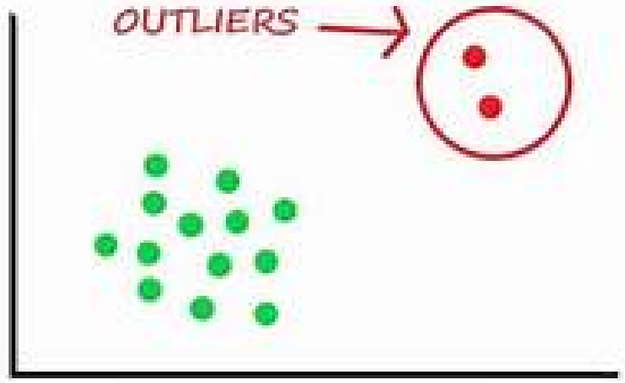


6. Outlier Detection

Use the Interquartile Range (IQR) method to detect outliers.

Outliers Detection (IQR method):

	outliers	outliers(%)
Pregnancies	4.0	0.52
Glucose	0.0	0.00
BloodPressure	14.0	1.82
SkinThickness	3.0	0.39
Insulin	24.0	3.12
BMI	8.0	1.04
DiabetesPedigreeFunction	29.0	3.78
Age	9.0	1.17
Outcome	0.0	0.00



7. Missing Value Treatment

The **Insulin** and **SkinThickness** columns had high missing data (nearly 50% and 30%).

The **Glucose**, **BloodPressure**, and **BMI** columns had only minor missing data (<5%).

The remaining columns had no missing data.

```python
# Missing value treatment suggestions (numeric)
print("Missing percent:")
print(df.isna().mean().round(3))

# Simple imputation example: median
df_imputed = df.copy()
for c in to_nan:
    df_imputed[c] = df_imputed[c].fillna(df_imputed[c].median())
```

```
Missing percent:
Pregnancies                 0.000
Glucose                     0.007
BloodPressure               0.046
SkinThickness               0.296
Insulin                     0.487
BMI                         0.014
DiabetesPedigreeFunction    0.000
Age                         0.000
Outcome                     0.000
dtype: float64
```

With little missing data → We can **impute missing values using the median**.

8. Class-wise Summary

Compare **median feature values** between **diabetic and non-diabetic** groups.

```
# Check class-wise summaries
if label_col:
    print(df_imputed.groupby(label_col)[num].median().T)
```

Outcome	0	1
Pregnancies	2.000	4.000
Glucose	107.500	140.000
BloodPressure	72.000	74.000
SkinThickness	29.000	29.000
Insulin	125.000	125.000
BMI	30.400	34.250
DiabetesPedigreeFunction	0.336	0.449
Age	27.000	36.000
Outcome	0.000	1.000

⟶ Diabetics show higher **glucose**, **BMI**, **age**, and **pregnancies**, while other features vary little.

9. Save Cleaned Dataset

Export the cleaned dataset for modeling.

```
# Save a cleaned copy
df_imputed.to_csv("diabetes_cleaned.csv", index=False, header=False)
print("Saved cleaned file: diabetes_cleaned.csv")
```

```
Saved cleaned file: diabetes_cleaned.csv
```

diabetes_cleaned.csv:

	A	B	C	D	E	F	G	H	I
1	6	148	72	35	125	33.6	0.627	50	1
2	1	85	66	29	125	26.6	0.351	31	0
3	8	183	64	29	125	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	29	125	25.6	0.201	30	0
7	3	78	50	32	88	31	0.248	26	1
8	10	115	72	29	125	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	29	125	32.3	0.232	54	1
11	4	110	92	29	125	37.6	0.191	30	0
12	10	168	74	29	125	38	0.537	34	1


THANK YOU FOR YOUR ATTENTION!