Naive Bayes Classifier

Bayes' Theorem

Bayes' Theorem is a fundamental concept in probability theory and statistics that describes the likelihood of an event based on prior knowledge of conditions related to the event. It provides a way to update the probability of a hypothesis (event) given new evidence.

The formula for **Bayes' Theorem** is:

```
P(A|B) = [P(B|A) * P(A)] / P(B)
```

Where:

- `P(A|B)` is the **posterior probability**: the probability of event A occurring given that B is true.
- `P(B|A)` is the **likelihood**: the probability of event B occurring given that A is true.
- `P(A)` is the **prior probability**: the initial probability of event A before considering B.
- `P(B)` is the **evidence**: the probability of event B.

Naive Bayes Classifier

The **Naive Bayes classifier** is a probabilistic machine learning model based on Bayes' Theorem. It's called "naive" because it assumes that the features are conditionally independent of each other given the class label, which rarely holds in real-life situations. Despite this naive assumption, Naive Bayes works surprisingly well in many complex real-world tasks, especially in text classification, spam filtering, and sentiment analysis.

Bayes' Theorem in Naive Bayes Classification

In a classification problem, Bayes' Theorem is used to compute the probability of a given class `C` given a set of features `X = (x1, x2, ..., xn)`:

```
P(C|X) = [P(X|C) * P(C)] / P(X)
```

Since `P(X)` is the same for all classes and does not affect the classification, we focus on maximizing the numerator:

```
P(C|X) ∝ P(C) * P(X|C)
```

Now, under the **Naive assumption** (i.e., all features are conditionally independent):

```
P(X|C) = P(x1|C) * P(x2|C) * ... * P(xn|C)
```

Thus, the classifier predicts the class that maximizes:

```
P(C) * P(x1|C) * P(x2|C) * ... * P(xn|C)
```

Steps in Naive Bayes Classification:

1. **Calculate Prior Probabilities (`P(C)`)**: This is the proportion of each class in the training dataset.
2. **Calculate Likelihood (`P(xi|C)`)**: This is the likelihood of each feature given each class.
3. **Predict Class**: For a new example, calculate the posterior probability for each class, then predict the class with the highest probability.

Assumptions of Naive Bayes

1. **Conditional Independence**: The key assumption is that the features are independent of each other given the class label. This is often not true in practice, but Naive Bayes performs well despite this limitation.
2. **Feature Contribution**: All features contribute equally to the outcome.

Types of Naive Bayes Classifiers

1. **Gaussian Naive Bayes**:

    - Assumes that the continuous values associated with each feature follow a Gaussian (normal) distribution.
    - Used for continuous data.
    - **Example**: Predicting if a person will purchase a product based on age and salary.

    Formula:

    ```
    P(x|C) = (1 / sqrt(2 * π * σ^2)) * exp(-(x - μ)^2 / (2 * σ^2))
    ```
    Where `μ` is the mean and `σ` is the standard deviation of the feature.

2. **Multinomial Naive Bayes**:

    - Used for discrete features like word counts in text classification (e.g., spam detection, sentiment analysis).
    - **Example**: Classifying an email as spam or not spam based on the frequency of certain words.

3. **Bernoulli Naive Bayes**:

    - Used for binary or boolean features.
    - Assumes that features are binary (e.g., presence or absence of a word).
    - **Example**: Sentiment analysis with binary word presence (whether a word appears in the document or not).

Example of Naive Bayes Classifier

Let's consider a **spam classification** example using a **Multinomial Naive Bayes classifier**. Assume you have a dataset with emails labeled as either "spam" or "not spam" and a set of words that appear in these emails.

1. **Prior Probability (`P(C)`):**

- Suppose 30% of the emails in the dataset are labeled as "spam" and 70% as "not spam".
- `P(spam) = 0.30`
- `P(not spam) = 0.70`

2. **Likelihood (`P(X|C)`):**
    - Assume you have two words: "buy" and "free". The likelihood of these words given the class is calculated from the training data.
    - `P(buy|spam) = 0.4`, `P(free|spam) = 0.8`
    - `P(buy|not spam) = 0.1`, `P(free|not spam) = 0.05`
3. **Prediction:**
    - Given a new email containing the words "buy" and "free", we calculate the posterior probabilities:
        - `P(spam|buy, free) ∝ P(spam) * P(buy|spam) * P(free|spam)`
        - `P(not spam|buy, free) ∝ P(not spam) * P(buy|not spam) * P(free|not spam)`

    After calculating both probabilities, the class with the higher probability will be the predicted label for the email.

Advantages of Naive Bayes

- **Fast and Efficient**: It is computationally efficient and works well with large datasets.
- **Performs Well with Small Data**: It doesn’t require large training datasets to perform well.
- **Performs Well with High-Dimensional Data**: Particularly useful for text classification tasks, which often involve many features (words).

Disadvantages of Naive Bayes

- **Independence Assumption**: The assumption that features are conditionally independent is often unrealistic in many practical scenarios. While Naive Bayes performs well despite this, it can be suboptimal when features are highly correlated.
- **Zero Frequency Problem**: If a categorical feature in the test data has a value that was not observed in the training data, the model assigns zero probability to that event. This issue is typically handled by techniques like **Laplace Smoothing**.

Conclusion

Naive Bayes is an efficient, interpretable, and widely used classification algorithm, particularly for text classification tasks. Its simplicity, coupled with surprisingly good performance despite its naive assumptions, makes it a powerful tool in many situations.

Docs: https://scikit-learn.org/stable/modules/naive_bayes.html

Import of all the required libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Create a dataframe

In [2]:
```
dataframe = pd.read_csv("./diabetes.csv")
dataframe.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPed
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	

In [3]:
```
dataframe.columns
```

Out[3]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insu
lin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [4]:
```
dataframe.dtypes
```

Out[4]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

Missing values in a given dataset

In [5]:
```
dataframe.isnull().sum()
```

Out[5]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [6]:
```
dataframe.head(20)
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
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	
5	5	116	74	0	0	25.6	
6	3	78	50	32	88	31.0	
7	10	115	0	0	0	35.3	
8	2	197	70	45	543	30.5	
9	8	125	96	0	0	0.0	
10	4	110	92	0	0	37.6	
11	10	168	74	0	0	38.0	
12	10	139	80	0	0	27.1	
13	1	189	60	23	846	30.1	
14	5	166	72	19	175	25.8	
15	7	100	0	0	0	30.0	
16	0	118	84	47	230	45.8	
17	7	107	74	0	0	29.6	
18	1	103	30	38	83	43.3	
19	1	115	70	30	96	34.6	

- Data Imputation of 0's in every feature
- Size of the data

In [7]:
```
dataframe.shape
```

Out[7]:
```
(768, 9)
```

- Target Column: Outcome[0,1] (Binary Classification Task)
- As the target column is available in the dataset, supervised machine learning algorithm.
- Records: 768

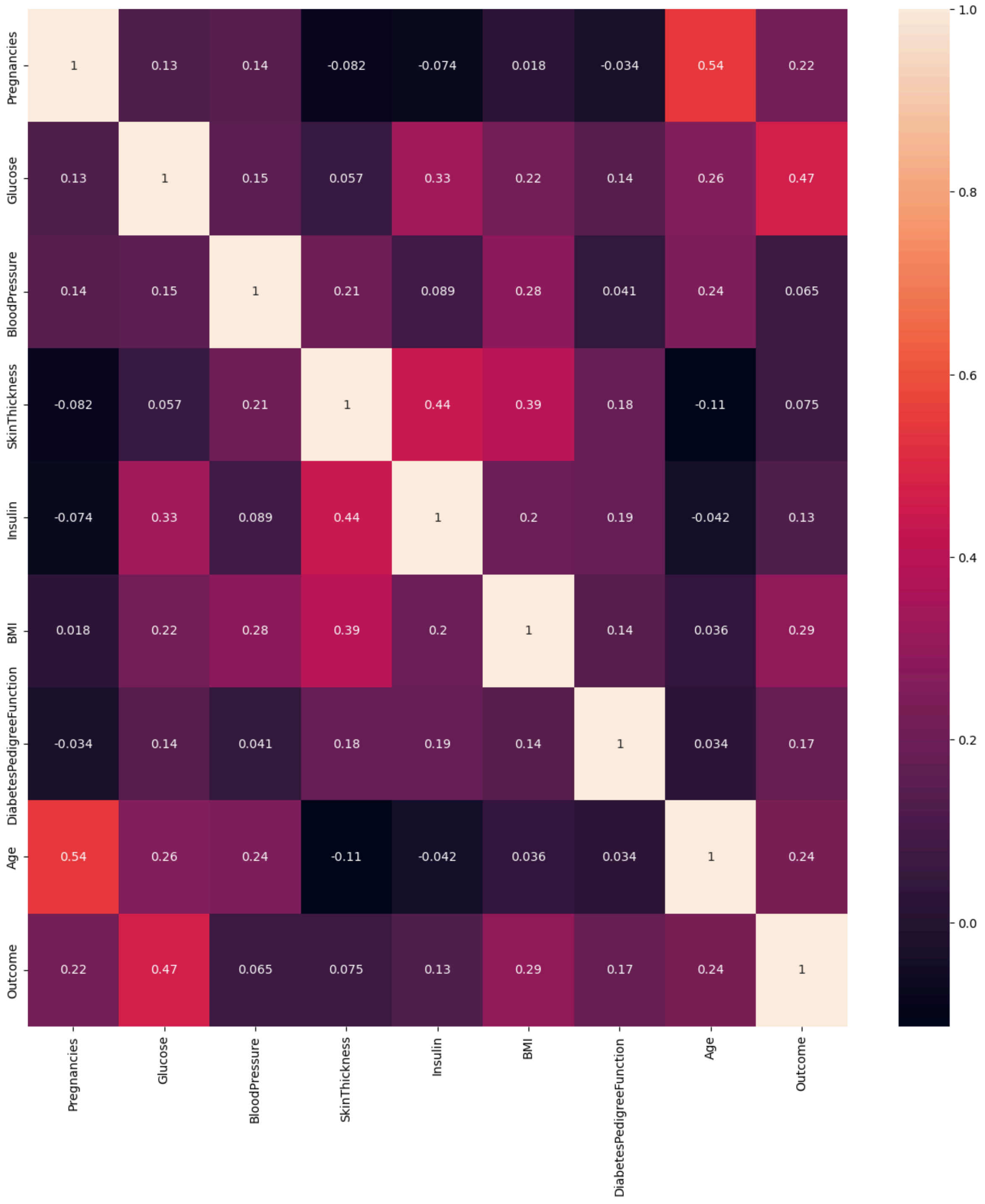
Correlation Coefficient

In [8]:
```
dataframe.corr()
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness
Pregnancies	1.000000	0.129459	0.141282	-0.081672
Glucose	0.129459	1.000000	0.152590	0.057328
BloodPressure	0.141282	0.152590	1.000000	0.207371
SkinThickness	-0.081672	0.057328	0.207371	1.000000
Insulin	-0.073535	0.331357	0.088933	0.436783
BMI	0.017683	0.221071	0.281805	0.392573
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928
Age	0.544341	0.263514	0.239528	-0.113970
Outcome	0.221898	0.466581	0.065068	0.074752

In [9]:
```
plt.figure(figsize=(15,15))
ax = sns.heatmap(dataframe.corr(), annot=True)
plt.savefig('correlation-coefficient.jpg')
plt.show()
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1	0.13	0.14	-0.082	-0.074	0.018	-0.034	0.54	0.22
Glucose	0.13	1	0.15	0.057	0.33	0.22	0.14	0.26	0.47
BloodPressure	0.14	0.15	1	0.21	0.089	0.28	0.041	0.24	0.065
SkinThickness	-0.082	0.057	0.21	1	0.44	0.39	0.18	-0.11	0.075
Insulin	-0.074	0.33	0.089	0.44	1	0.2	0.19	-0.042	0.13
BMI	0.018	0.22	0.28	0.39	0.2	1	0.14	0.036	0.29
DiabetesPedigreeFunction	-0.034	0.14	0.041	0.18	0.19	0.14	1	0.034	0.17
Age	0.54	0.26	0.24	-0.11	-0.042	0.036	0.034	1	0.24
Outcome	0.22	0.47	0.065	0.075	0.13	0.29	0.17	0.24	1

Descriptive Statistics of the given data

In [10]:
```
dataframe.describe()
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.0
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.9
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.8
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.3
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.0
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.6
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.1

Data Imputation

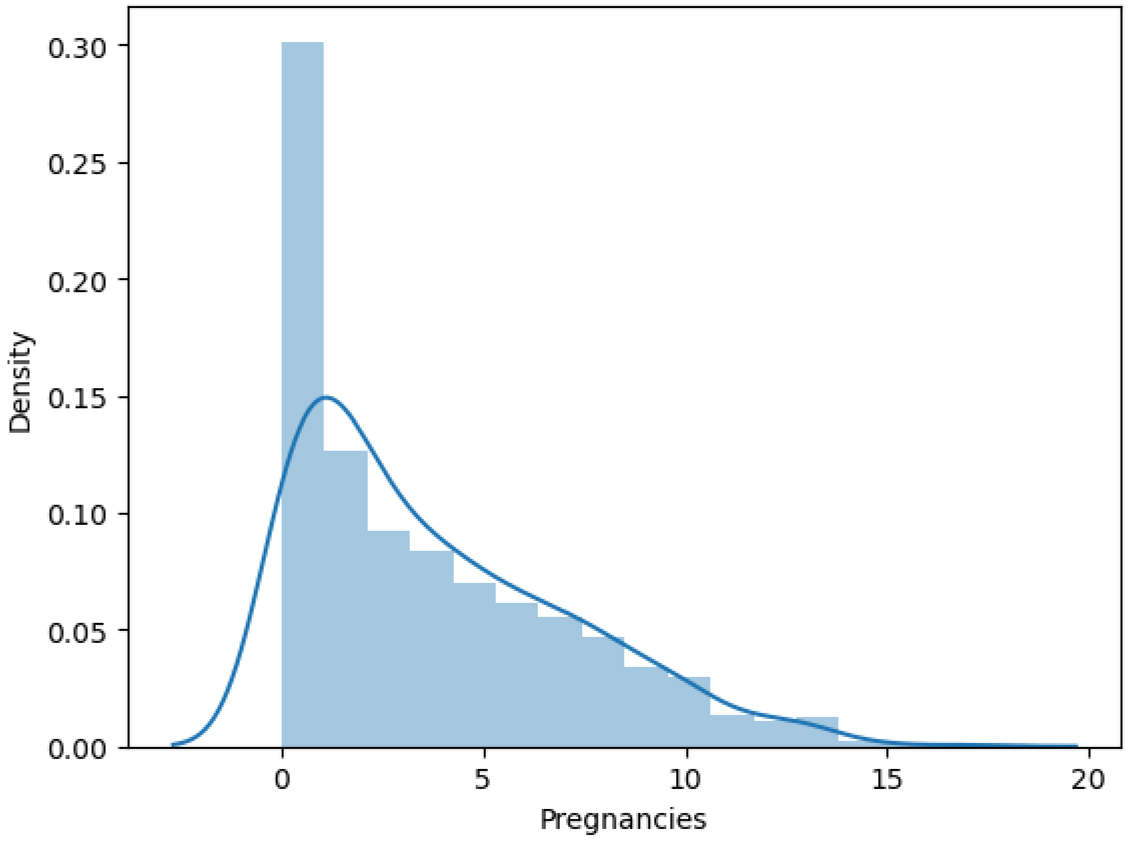
In [11]:

```python
# Pregnancies -> Median
sns.distplot(dataframe.Pregnancies)
```

```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_96515/325720374
0.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataframe.Pregnancies)
```

Out[11]:

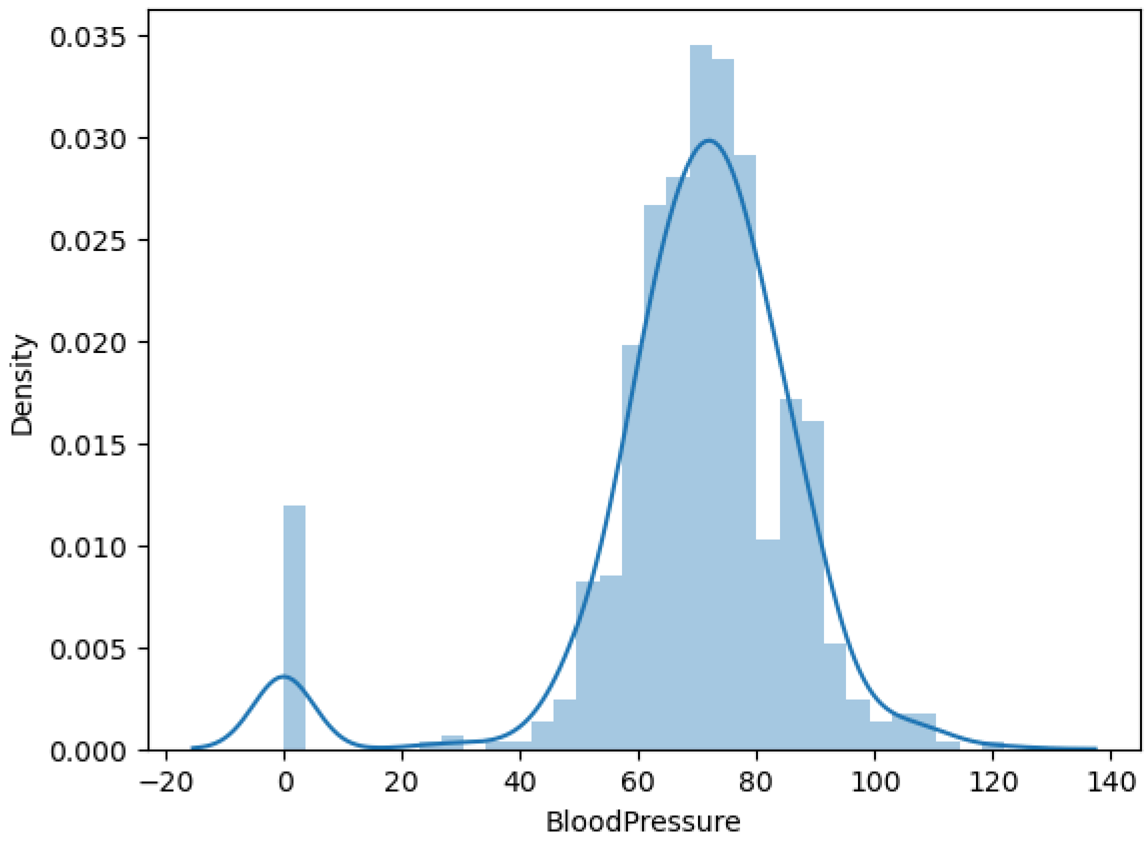
```
<Axes: xlabel='Pregnancies', ylabel='Density'>
```



```
In [12]: ## BP -> Mean
         sns.distplot(dataframe.BloodPressure)
```

```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_96515/89164806
8.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataframe.BloodPressure)
```

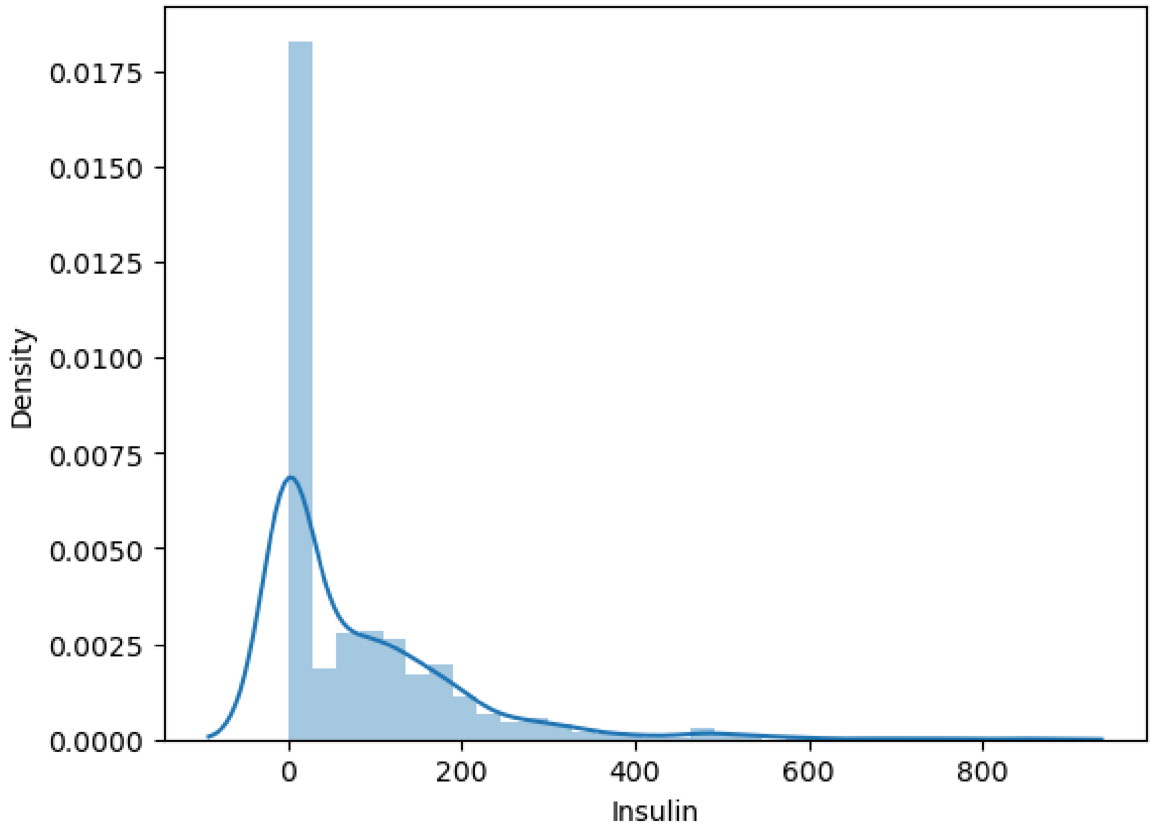
```
Out[12]: <Axes: xlabel='BloodPressure', ylabel='Density'>
```



In [13]:
```
# Insulin -> Median
sns.distplot(dataframe.Insulin)
```

```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_96515/257615224
7.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataframe.Insulin)
```

Out[13]:
```
<Axes: xlabel='Insulin', ylabel='Density'>
```



In [14]:
```
dataframe.columns
```

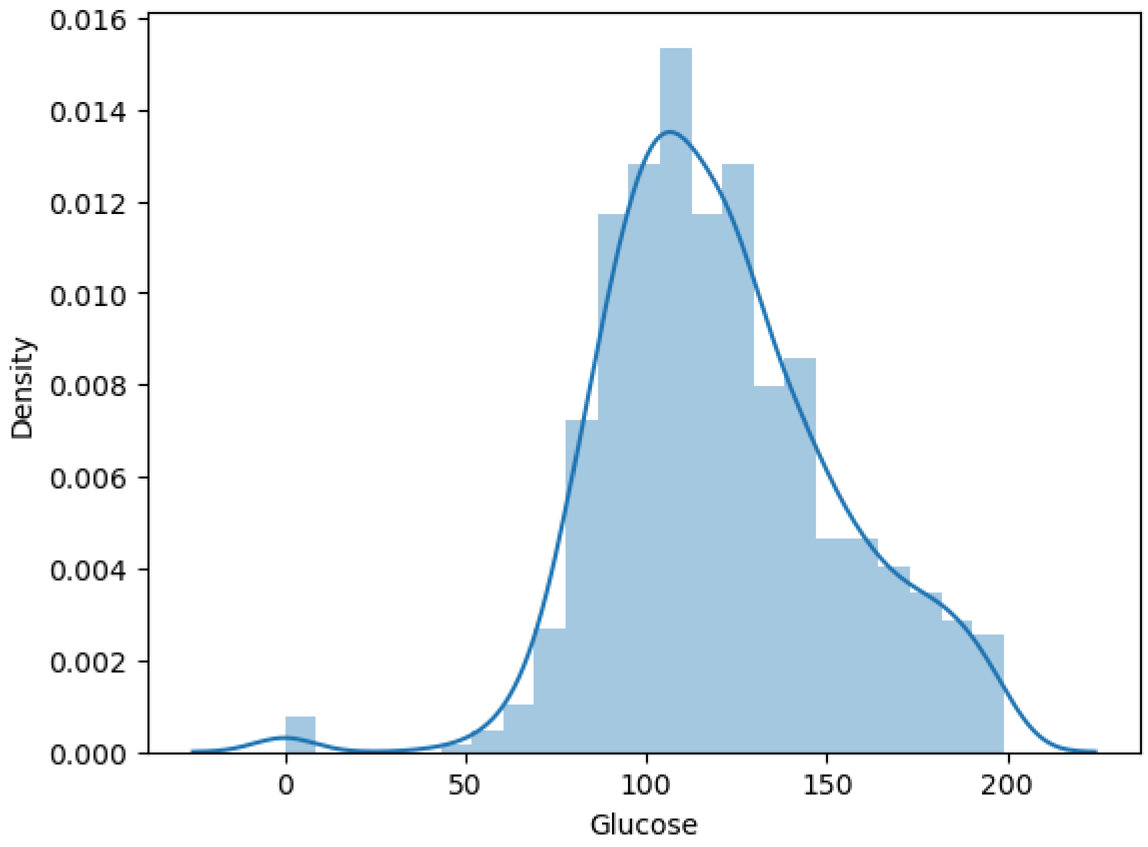
Out[14]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insu
lin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [15]:
```
## Insuline -> Right skewed distribution
dataframe['Insulin'] = dataframe['Insulin'].replace(0, dataframe['Insulin
```

In [16]:
```
sns.distplot(dataframe.Glucose)
```

```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_96515/223043267
7.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataframe.Glucose)
```

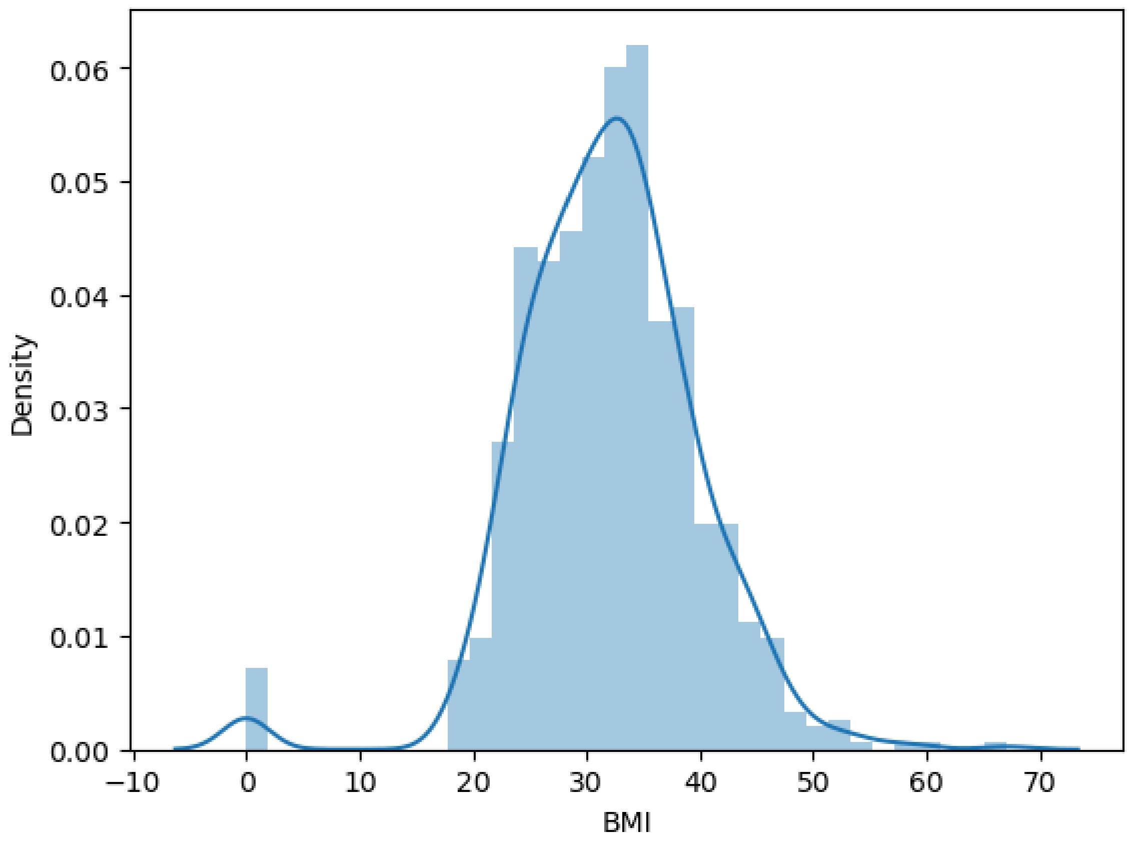
Out[16]:
```
<Axes: xlabel='Glucose', ylabel='Density'>
```



```
In [17]: sns.distplot(dataframe.BMI)
```

```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_96515/252098079
3.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataframe.BMI)
```

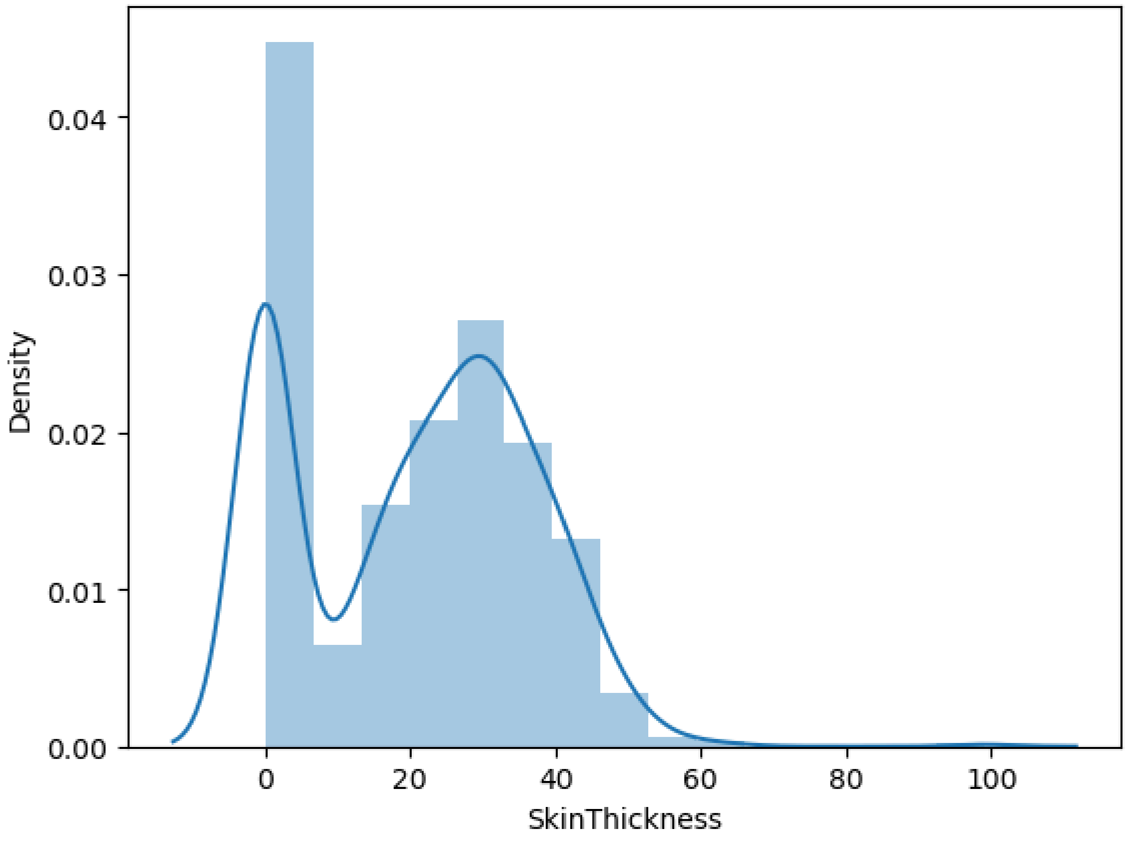
```
Out[17]: <Axes: xlabel='BMI', ylabel='Density'>
```



In [18]:
```python
sns.distplot(dataframe.SkinThickness)
```

```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_96515/386125304
5.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataframe.SkinThickness)
```

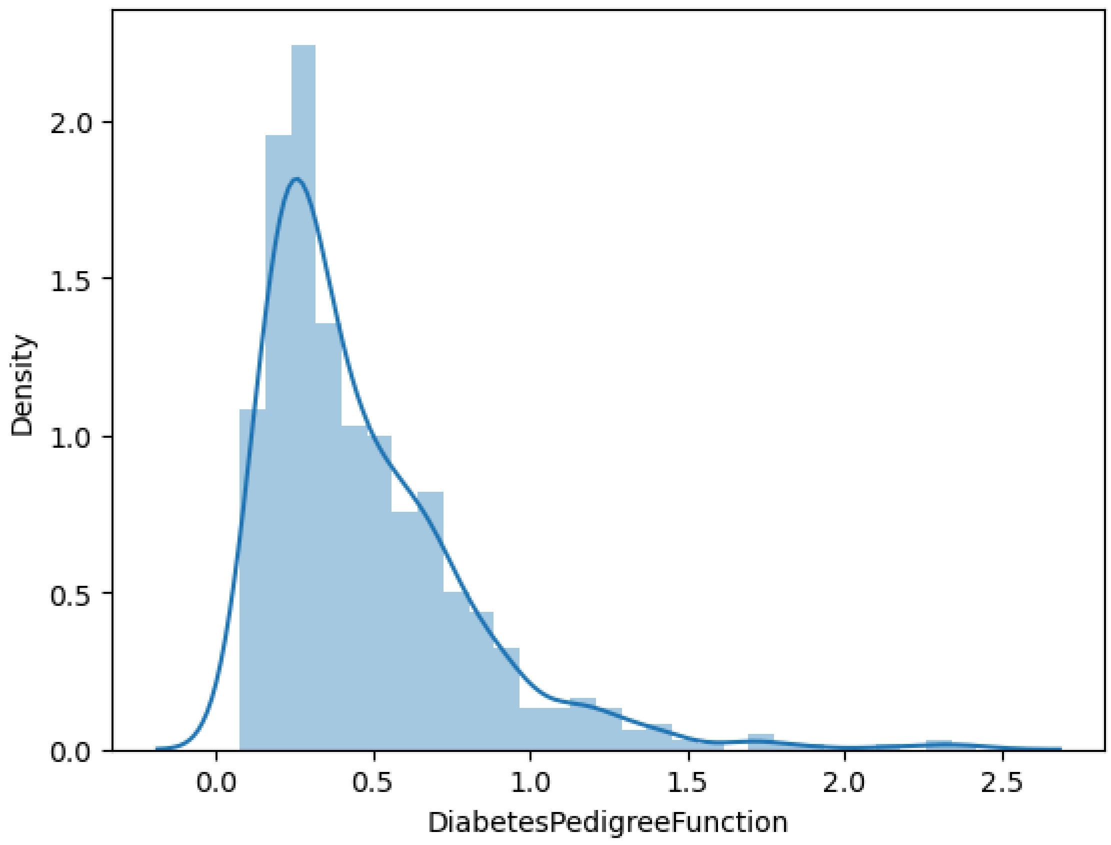
Out[18]:
```
<Axes: xlabel='SkinThickness', ylabel='Density'>
```



In [19]:
```
sns.distplot(dataframe.DiabetesPedigreeFunction)
```

```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_96515/264275873
4.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataframe.DiabetesPedigreeFunction)
```

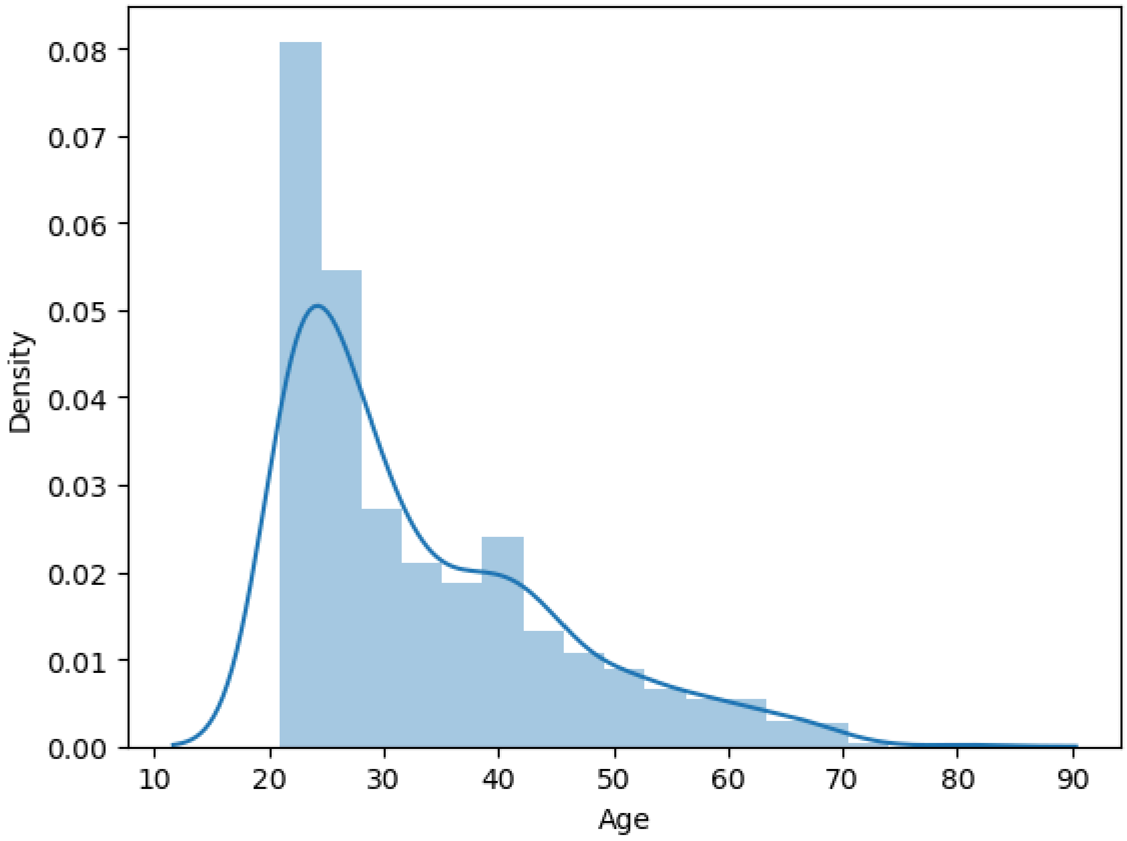
Out[19]:
```
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='Density'>
```



In [20]:
```
sns.distplot(dataframe.Age)
```

```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_96515/269143098
7.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataframe.Age)
```

Out[20]:
```
<Axes: xlabel='Age', ylabel='Density'>
```



In [21]:
```
dataframe['Pregnancies'] = dataframe['Pregnancies'].replace(0, dataframe[
dataframe['Glucose'] = dataframe['Glucose'].replace(0, dataframe['Glucose
dataframe['BloodPressure'] = dataframe['BloodPressure'].replace(0, datafr
dataframe['SkinThickness'] = dataframe['SkinThickness'].replace(0, datafr
dataframe['BMI'] = dataframe['BMI'].replace(0, dataframe['BMI'].mean())
dataframe['DiabetesPedigreeFunction'] = dataframe['DiabetesPedigreeFuncti
dataframe['Age'] = dataframe['Age'].replace(0, dataframe['Age'].median())
```

In [22]:
```
dataframe.head(20)
```

Out[22]:

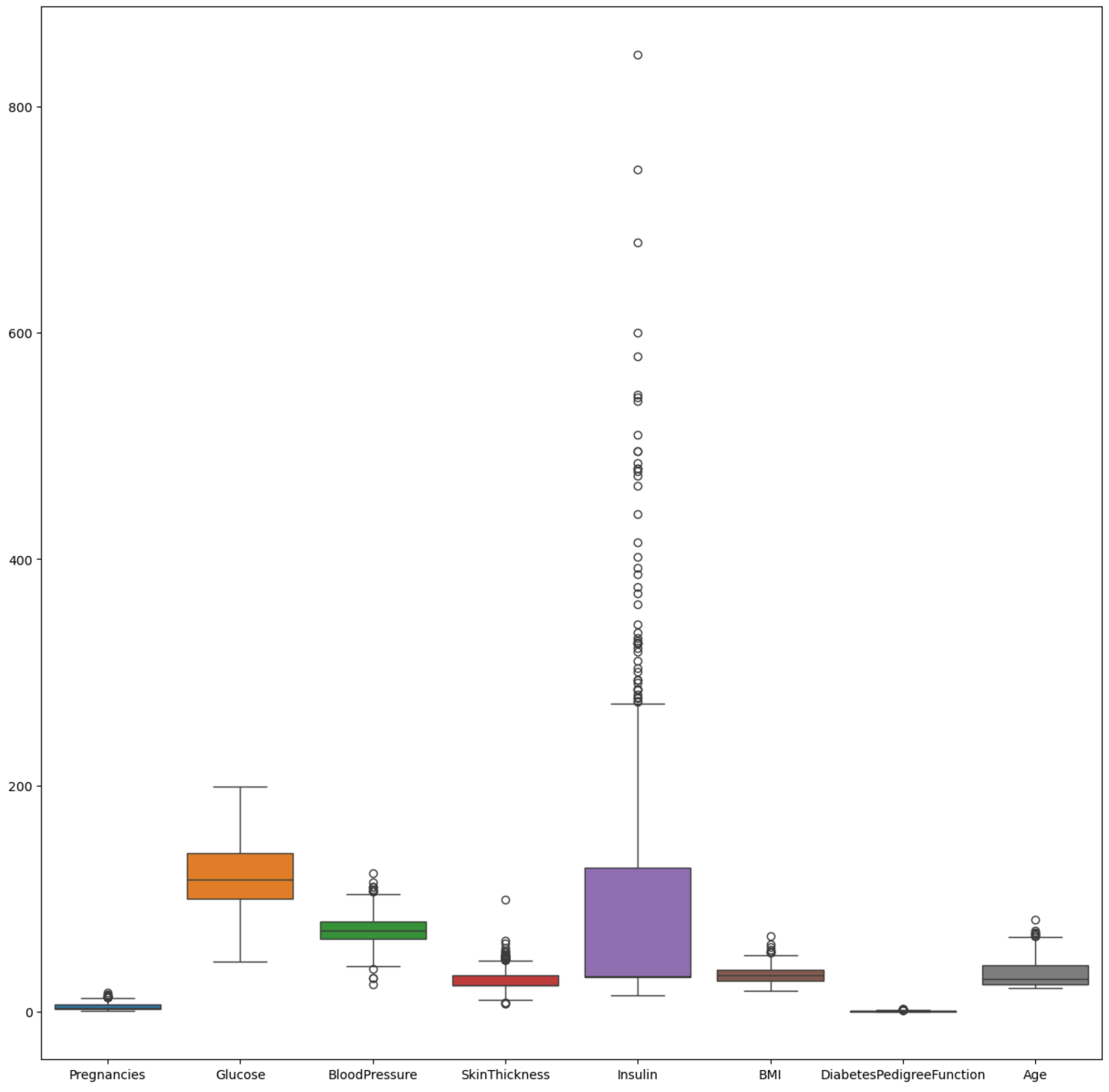
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabe
0	6	148.0	72.000000	35	30.5	33.600000	
1	1	85.0	66.000000	29	30.5	26.600000	
2	8	183.0	64.000000	23	30.5	23.300000	
3	1	89.0	66.000000	23	94.0	28.100000	
4	3	137.0	40.000000	35	168.0	43.100000	
5	5	116.0	74.000000	23	30.5	25.600000	
6	3	78.0	50.000000	32	88.0	31.000000	
7	10	115.0	69.105469	23	30.5	35.300000	
8	2	197.0	70.000000	45	543.0	30.500000	
9	8	125.0	96.000000	23	30.5	31.992578	
10	4	110.0	92.000000	23	30.5	37.600000	
11	10	168.0	74.000000	23	30.5	38.000000	
12	10	139.0	80.000000	23	30.5	27.100000	
13	1	189.0	60.000000	23	846.0	30.100000	
14	5	166.0	72.000000	19	175.0	25.800000	
15	7	100.0	69.105469	23	30.5	30.000000	
16	3	118.0	84.000000	47	230.0	45.800000	
17	7	107.0	74.000000	23	30.5	29.600000	
18	1	103.0	30.000000	38	83.0	43.300000	
19	1	115.0	70.000000	30	96.0	34.600000	

- Descriptive Statistics and it's significance
- Correlation Coefficient and it's significance
- Types of Distribution and it's significance
- Median is more robuts to outliers and why
- Data Imputation via Mean and Median(Numeric Data) => Symmetric -> Mean and Skewed -> Median, Categorical Data => Mode

In [23]:
```
## X -> input features y -> target value
X = dataframe.drop(columns='Outcome', axis=1)
y = dataframe['Outcome']
```

Outlier Detection -> Box Plot

In [24]:
```
fig, ax = plt.subplots(figsize = (15, 15))
sns.boxplot(data = X, ax=ax)
plt.savefig('boxPlot.jpg')
```



In [25]:
```
X.shape
```

Out[25]:
```
(768, 8)
```

In [26]:
```
y.shape
```

Out[26]:
```
(768,)
```

In [27]:
```
cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insu
for col in cols:
    Q1 = X[col].quantile(0.25)
    Q3 = X[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    mask = (X[col] >= lower_bound) & (X[col] <= upper_bound)
```

In [28]:
```
X_outlier_detection = X[mask]
y_outlier_detection = y[mask]
```

In [29]:
```
X_outlier_detection.shape
```

Out[29]:
```
(759, 8)
```

In [30]:
```
y_outlier_detection.shape
```

Out[30]:
```
(759,)
```

Standardization

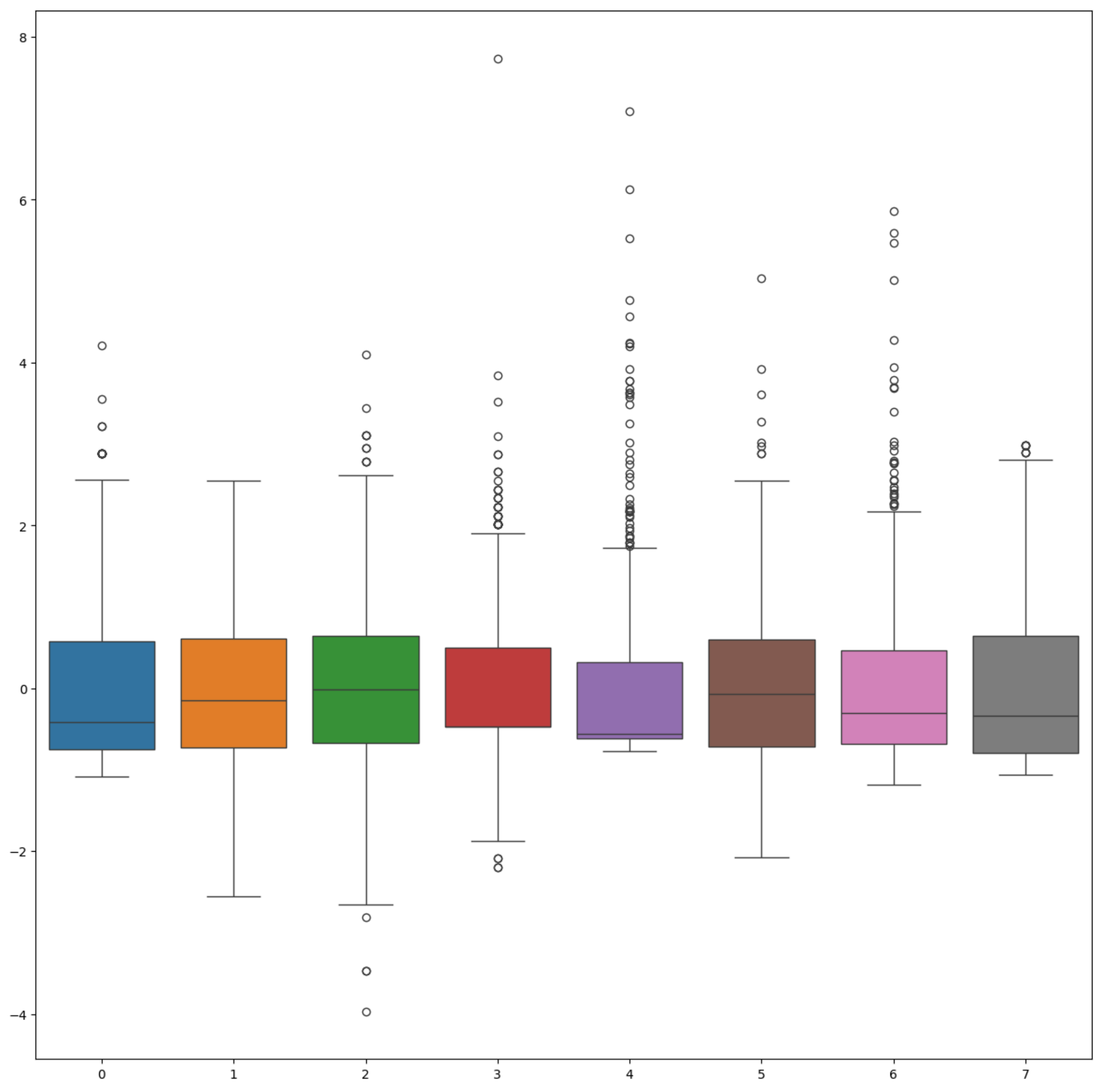
Standard Normal Form -> Mean = 0 and standard deviation = 1

In [31]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_outlier_detection)
```

In [32]:
```
X_scaled
```

Out[32]:
```
array([[ 0.57322173,  0.87008298, -0.01698412, ...,  0.16090077,
         0.46879263,  1.54828125],
       [-1.0797999 , -1.20656984, -0.51093456, ..., -0.85816238,
        -0.36177415, -0.16252742],
       [ 1.23443039,  2.02377899, -0.6755847 , ..., -1.33857787,
         0.60421113, -0.07248486],
       ...,
       [ 0.2426174 , -0.01991109, -0.01698412, ..., -0.91639456,
        -0.68075995, -0.25256998],
       [-1.0797999 ,  0.14490263, -1.00488499, ..., -0.3486308 ,
        -0.36779275,  1.27815356],
       [-1.0797999 , -0.9428679 , -0.18163427, ..., -0.30495667,
        -0.47010895, -0.88286791]])
```

In [33]:
```
fig, ax = plt.subplots(figsize = (15, 15))
sns.boxplot(data = X_scaled, ax=ax)
plt.savefig('boxPlot.jpg')
```



In [34]:
```
dataframe.columns
```

Out[34]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insu
lin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [35]:
```
cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insu
```

In [36]:
```
type(X_scaled)
```

Out[36]:
```
numpy.ndarray
```

In [37]:
```
X_scaled = pd.DataFrame(X_scaled, columns=cols)
X_scaled.describe()
```

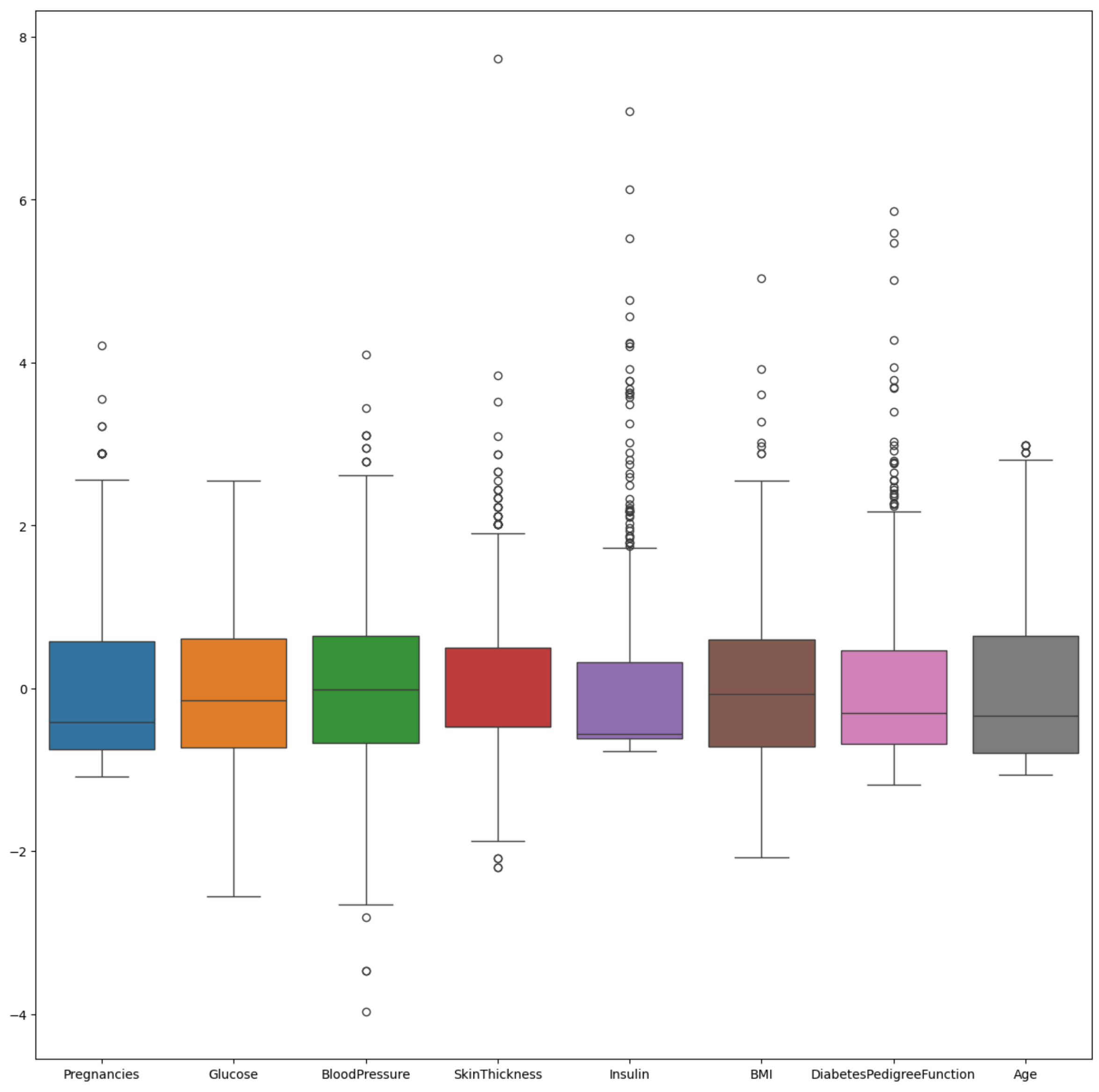
Out[37]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insuli
count	7.590000e+02	7.590000e+02	7.590000e+02	7.590000e+02	7.590000e+0
mean	1.029772e-16	-3.978665e-17	-3.042508e-17	-1.509552e-16	-4.329724e 1
std	1.000659e+00	1.000659e+00	1.000659e+00	1.000659e+00	1.000659e+0
min	-1.079800e+00	-2.558042e+00	-3.968588e+00	-2.200901e+00	-7.684941e 0
25%	-7.491956e-01	-7.286101e-01	-6.755847e-01	-4.729631e-01	-6.126688e 0
50%	-4.185912e-01	-1.517621e-01	-1.698412e-02	-4.729631e-01	-5.607270e 0
75%	5.732217e-01	6.063810e-01	6.416165e-01	4.990017e-01	3.222827e-0
max	4.209869e+00	2.551183e+00	4.099270e+00	7.734740e+00	7.088876e+0

- Approach 2 of quantiles to remove the outliers
- Handling of imbalanced data

In [38]:

```
fig, ax = plt.subplots(figsize = (15, 15))
sns.boxplot(data = X_scaled, ax=ax)
plt.savefig('boxPlot.jpg')
```



In [39]:
```
y_outlier_detection.shape
```

Out[39]:
```
(759,)
```

In [40]:
```
y_outlier_detection.value_counts()
```

Out[40]:
```
Outcome
0    493
1    266
Name: count, dtype: int64
```

Concluding:

- Detection of the outliers
- Normalization via StandardScaler Form & Why it is important(reduce the biasness in the model)

Approach 2: Quantiles

In [41]:
```
X_scaled.reset_index(drop=True, inplace=True)
y_outlier_detection.reset_index(drop=True, inplace=True)
```

In [42]:
```
q = X_scaled['Insulin'].quantile(.95)
mask = X_scaled['Insulin'] < q
dataNew = X_scaled[mask]
y_outlier_detection = y_outlier_detection[mask]
```

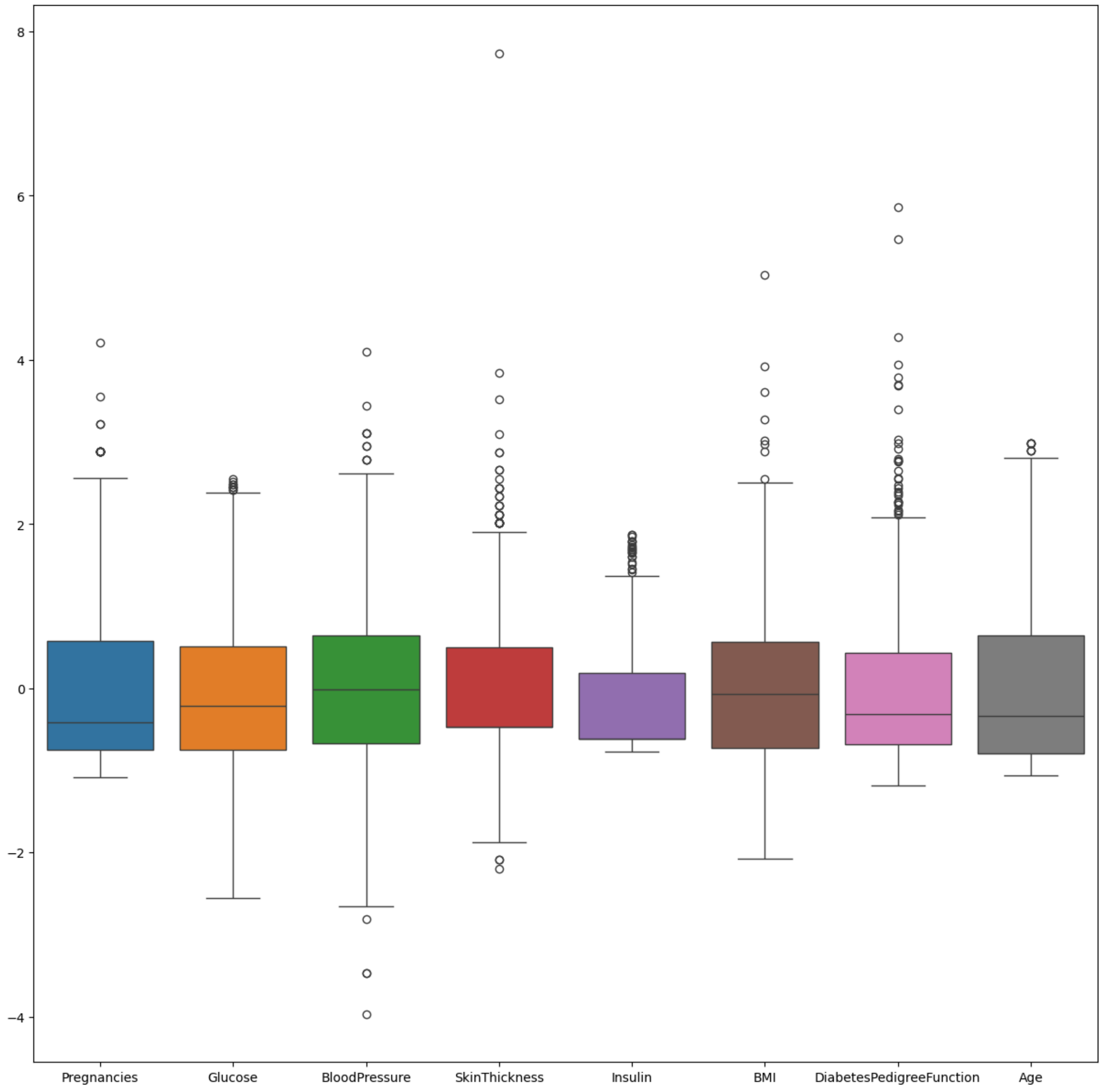
In [43]:
```
dataNew.shape
```

Out[43]:
```
(721, 8)
```

In [44]:
```
y_outlier_detection.shape
```

Out[44]:
```
(721,)
```

In [45]:
```
fig, ax = plt.subplots(figsize = (15, 15))
sns.boxplot(data = dataNew, ax=ax)
plt.savefig('boxPlot.jpg')
```



Model Training

Splitting of data into training and testing

In [46]:
```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(dataNew, y_outlier_de
```

In [47]:
```
X_train.shape
```

Out[47]:
```
(483, 8)
```

In [48]:
```
X_test.shape
```

Out[48]:
```
(238, 8)
```

Data Imbalancing

- Oversampling : Minority Class and increase that number to the majority class
- Undersampling : Majority class and decrease that number to the minority class
- SMOTE : Synthetic data and increase the number of samples to the majority class

In [49]:
```
y_train.value_counts()
```

Out[49]:
```
Outcome
0    318
1    165
Name: count, dtype: int64
```

SMOTE Technique

In [50]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_trai

# Check resampled class distribution
print("\nResampled class distribution:")
print(pd.Series(y_train_resampled).value_counts())
```

```
Resampled class distribution:
Outcome
0    318
1    318
Name: count, dtype: int64
```

In [51]:
```
from sklearn.linear_model import LogisticRegression
classification = LogisticRegression()
classification.fit(X_train_resampled, y_train_resampled)
```

Out[51]:
```
▾ LogisticRegression
LogisticRegression()
```

Model Predictions

In [52]:
```
y_predictions = classification.predict(X_test)
print(y_predictions)
```

```
[0 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 1 1 0 0 0
 0 1 0 1 0 1 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 1 0 0 1 0 0 0
 0 0 1 1 1 1 1 0 1 1 0 1 0 0 1 0 0 1 0 1 0 0 0 0 1 0 0 1 0 1 0 0 0 1 1 1 1
 0 0 1 1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 1 0 0 1 1 0 0 1 0 1
 1 0 0 0 0 1 1 1 1 0 1 0 0 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0
 1 1 1 1 0 0 1 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0
 0 0 1 0 1 1 0 0 1 0 0 1 1 1 1 1]
```

Model Evaluation

In [53]:
```python
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_predictions)
```

Out[53]:
```
0.7478991596638656
```

Healthcare: Recall is very important metric

In [54]:
```python
from sklearn.metrics import classification_report
target_names = ['Non-Diabetic', 'Diabetic']
print(classification_report(y_test, y_predictions, target_names=target_na
```

```
              precision    recall  f1-score   support

Non-Diabetic       0.85      0.76      0.80       159
    Diabetic       0.60      0.72      0.66        79

    accuracy                           0.75       238
   macro avg       0.72      0.74      0.73       238
weighted avg       0.76      0.75      0.75       238
```

In [55]:
```python
import pickle
pickle.dump(classification, open("classification_model.pkl", "wb"))
```

In [56]:
```python
classification_model = pickle.load(open("classification_model.pkl", "rb"))
classification_model.predict(X_test)
```

Out[56]:
```
array([0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1,
       1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1])
```

Model Training: KNNClassifier Model

In [57]:
```python
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.metrics import classification_report, confusion_matrix
knn = KNeighborsClassifier()
```

In [58]:
```python
knn.fit(X_train_resampled, y_train_resampled)
```

Out[58]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

Model Prediction

In [59]:
```python
y_prediction_knn = knn.predict(X_test)
y_prediction_knn
```

Out[59]:
```
array([0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1])
```

In [60]:
```python
print("Confusion Matrix")
print(confusion_matrix(y_test, y_prediction_knn))
```

```
Confusion Matrix
[[103  56]
 [ 19  60]]
```

In [61]:
```python
print("Classification Report")
print(classification_report(y_test, y_prediction_knn))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.84      0.65      0.73       159
           1       0.52      0.76      0.62        79

    accuracy                           0.68       238
   macro avg       0.68      0.70      0.67       238
weighted avg       0.74      0.68      0.69       238
```

Data Modeling: Implementation of Naive Bayes Classifier

In [62]:
```python
from sklearn.naive_bayes import GaussianNB
model_gaussian_naive_bayes = GaussianNB()
model_gaussian_naive_bayes.fit(X_train_resampled, y_train_resampled)
```

Out[62]:
```
▾ GaussianNB
GaussianNB()
```

In [63]:
```python
y_predict_gaussian_naive_bayes = model_gaussian_naive_bayes.predict(X_tes
print(y_predict_gaussian_naive_bayes)
```

```
[0 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 1 1 1 0 0
 0 1 0 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 0
 0 0 1 1 1 1 1 0 1 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 1 0 0 0 1 1 1 1
 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 1 0 0 1 0 1
 1 0 0 0 0 1 1 1 1 0 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0
 1 1 0 1 0 0 1 0 1 0 1 1 0 0 1 0 1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 1 0 1
 0 0 1 0 1 1 0 0 0 1 0 1 1 1 1 0]
```

In [64]:

```
print("Confusion Matrix")
print(confusion_matrix(y_test, y_predict_gaussian_naive_bayes))
```

```
Confusion Matrix
[[119  40]
 [ 27  52]]
```

In [65]:

```
print("Classification Report")
print(classification_report(y_test, y_predict_gaussian_naive_bayes))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.82      0.75      0.78       159
           1       0.57      0.66      0.61        79

    accuracy                           0.72       238
   macro avg       0.69      0.70      0.69       238
weighted avg       0.73      0.72      0.72       238
```

In [66]:

```
accuracy_score(y_test, y_predict_gaussian_naive_bayes)
```

Out[66]:

```
0.7184873949579832
```