Pima Indians Diabetes Database

PROJECT REPORT

Submitted by

Ankit Singh- 201020407
Priyank Singh- 201020442

Dr. Shyama Prasad Mukherjee International Institute of Information Technology, Naya Raipur

November-2021

Abstract

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.The dataset consists of several medical predictor variables and one target variable, Outcome. Predictor variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.The data set is taken from Kaggle.

1. Introduction

Diabetes is a group of metabolic diseases in which there are high blood sugar levels over a prolonged period. Symptoms of high blood sugar include frequent urination, increased thirst, and increased hunger.To study the reason that leading to diabetes, a cluster of dataset about Pima Indian Diabetes was collected. It consists of 8 predictor variables and 1 response variable.
The variable names are as follows:

- Number of times pregnant.
- Plasma glucose concentration.
- Diastolic blood pressure.
- Triceps skinfold thickness.
- 2-Hour serum insulin.
- Body mass index.
- Diabetes pedigree function.
- Age (years).
- Class variable (0 or 1).

The dataset has observations from 768 patients, 9 variables were taken to fit a generalized linear model to predict the probability that individual females have diabetes.Then, using stepwise selection provided subgroups of characteristics with higher risk of diabetes.

2. Objectives

The objective of this project is:

a)Analyse the dataset.

b)Apply data preprocessing techniques on the dataset.

3. Observations

I) Data Imputation

Imputation is the process of replacing missing data with substituted values
We start with handling missing values in the dataset.

Load the dataset

```
import pandas as pd
import numpy as np
```

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_style("darkgrid")
df=pd.read_csv(r"C:\Users\ankit\Downloads\diabetes.csv")
print(df.head())
```

```
   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
```

```
df.shape
```

```
(768, 9)
```

Mark Missing Values

As we see that some columns contain ‘0’ instead of ‘NaN’ or missing values. So, we have to replace those zeros with ‘NaN’ in columns like Insulin, BMI, Glucose, Skin Thickness and

Blood Pressure as they cannot be zero. But other columns like the number of Pregnancies can be zero.

```
df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']] = df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']].replace(0,np.nan)
print(df.isnull().sum())
```

```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Now, we create a new data frame with Glucose, Blood Pressure, Skin Thickness, Insulin and BMI columns from the original data frame as they contain missing values.

```
df1 = df[["Glucose","BloodPressure","SkinThickness","Insulin","BMI"]]
df1.head(10)
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI
0	148.0	72.0	35.0	NaN	33.6
1	85.0	66.0	29.0	NaN	26.6
2	183.0	64.0	NaN	NaN	23.3
3	89.0	66.0	23.0	94.0	28.1
4	137.0	40.0	35.0	168.0	43.1
5	116.0	74.0	NaN	NaN	25.6
6	78.0	50.0	32.0	88.0	31.0
7	115.0	NaN	NaN	NaN	35.3
8	197.0	70.0	45.0	543.0	30.5
9	125.0	96.0	NaN	NaN	NaN

We will now impute missing values in the Insulin column. The reason for selecting this column is that it contains the highest number of missing values.

Impute Insulin column with Mean and Median

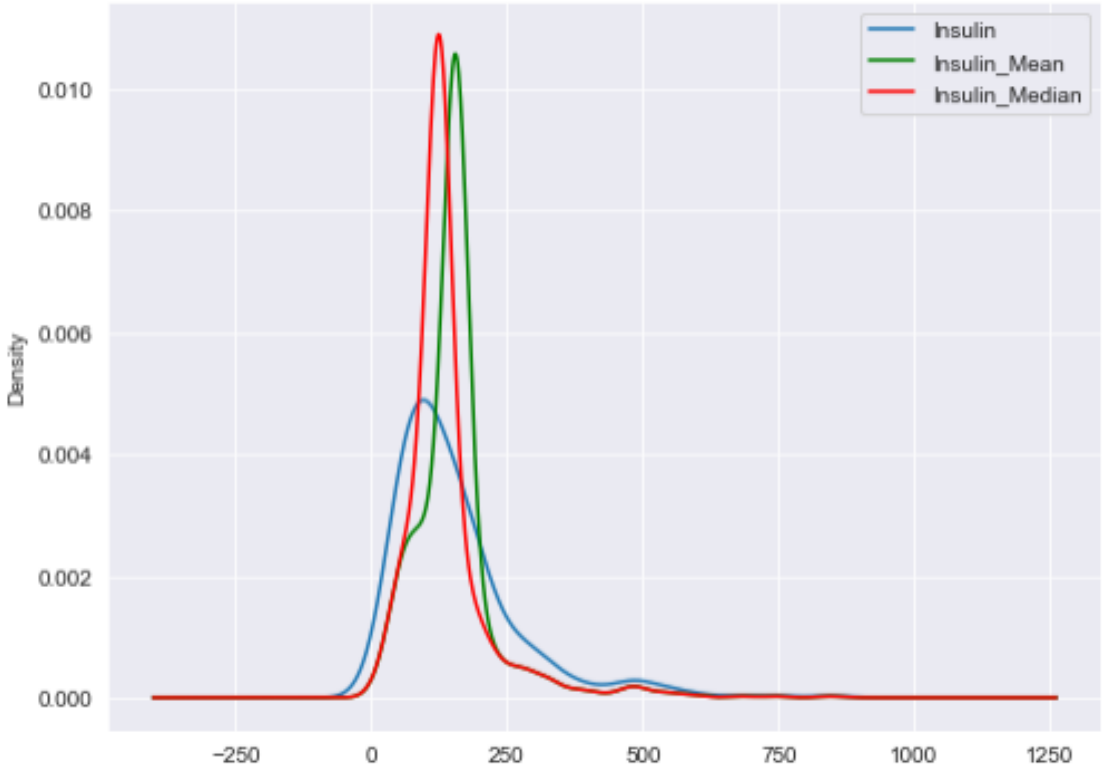
```
df1["Insulin_Mean"] = df1.Insulin.fillna(mean4)
df1["Insulin_Median"] = df1.Insulin.fillna(median4)
df1.head(20)
```

```
C:\Users\ankit\AppData\Local\Temp/ipykernel_14444/4293877484.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df1["Insulin_Mean"] = df1.Insulin.fillna(mean4)
C:\Users\ankit\AppData\Local\Temp/ipykernel_14444/4293877484.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df1["Insulin_Median"] = df1.Insulin.fillna(median4)
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Insulin_Mean	Insulin_Median
0	148.0	72.0	35.0	NaN	33.6	155.548223	125.0
1	85.0	66.0	29.0	NaN	26.6	155.548223	125.0
2	183.0	64.0	NaN	NaN	23.3	155.548223	125.0
3	89.0	66.0	23.0	94.0	28.1	94.000000	94.0
4	137.0	40.0	35.0	168.0	43.1	168.000000	168.0
5	116.0	74.0	NaN	NaN	25.6	155.548223	125.0
6	78.0	50.0	32.0	88.0	31.0	88.000000	88.0
7	115.0	NaN	NaN	NaN	35.3	155.548223	125.0
8	197.0	70.0	45.0	543.0	30.5	543.000000	543.0
9	125.0	96.0	NaN	NaN	NaN	155.548223	125.0
10	110.0	92.0	NaN	NaN	37.6	155.548223	125.0
11	168.0	74.0	NaN	NaN	38.0	155.548223	125.0
12	139.0	80.0	NaN	NaN	27.1	155.548223	125.0
13	189.0	60.0	23.0	846.0	30.1	846.000000	846.0
14	166.0	72.0	19.0	175.0	25.8	175.000000	175.0
15	100.0	NaN	NaN	NaN	30.0	155.548223	125.0
16	118.0	84.0	47.0	230.0	45.8	230.000000	230.0
17	107.0	74.0	NaN	NaN	29.6	155.548223	125.0
18	103.0	30.0	38.0	83.0	43.3	83.000000	83.0
19	115.0	70.0	30.0	96.0	34.6	96.000000	96.0


Insulin
Insulin_Mean
Insulin_Median
Density
0.010
0.008
0.006
0.004
0.002
0.000
−250
0
250
500
750
1000
1250

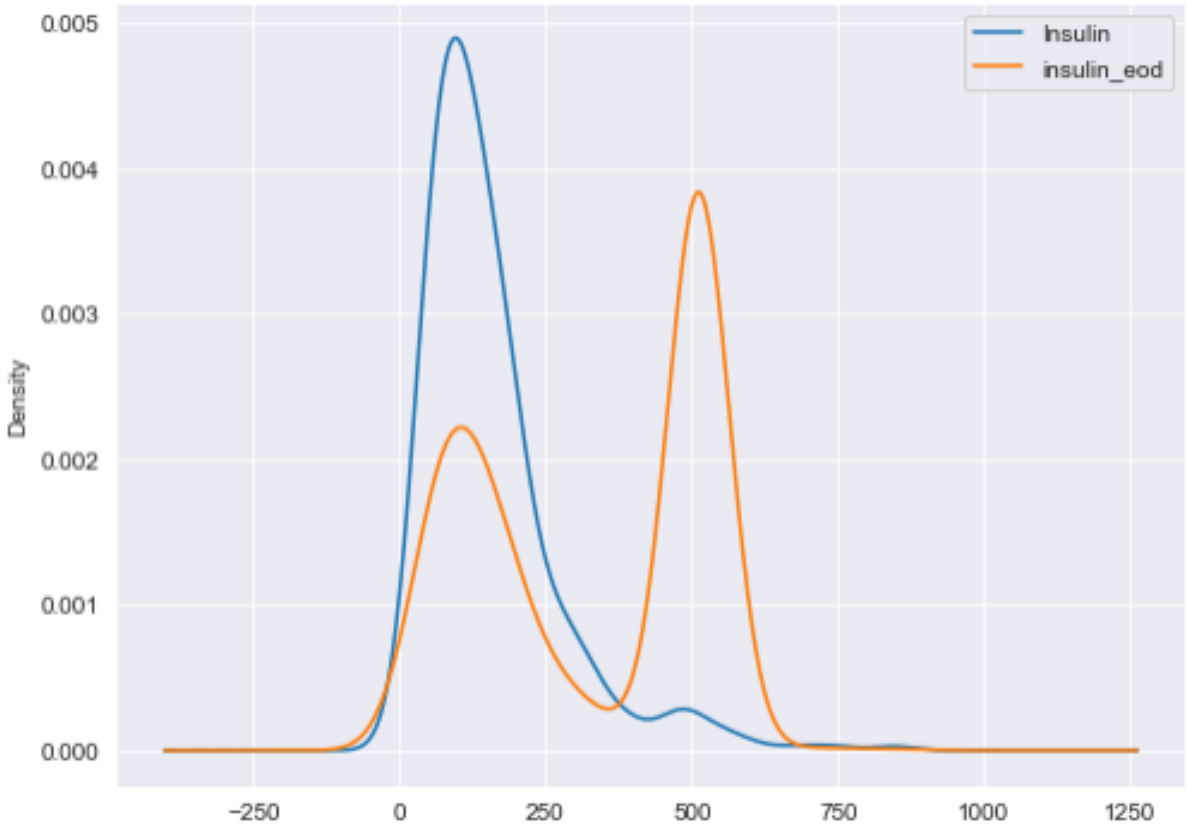
Impute with EOD(End of Distribution)

```
df1['insulin_eod'] = df1.Insulin.fillna(eod_value4)
df1.head(20)
```

```
C:\Users\ankit\AppData\Local\Temp/ipykernel_14444/1142793216.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df1['insulin_eod'] = df1.Insulin.fillna(eod_value4)
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Insulin_Mean	Insulin_Median	insulin_eod
0	148.0	72.0	35.0	NaN	33.6	155.548223	125.0	511.875789
1	85.0	66.0	29.0	NaN	26.6	155.548223	125.0	511.875789
2	183.0	64.0	NaN	NaN	23.3	155.548223	125.0	511.875789
3	89.0	66.0	23.0	94.0	28.1	94.000000	94.0	94.000000
4	137.0	40.0	35.0	168.0	43.1	168.000000	168.0	168.000000
5	116.0	74.0	NaN	NaN	25.6	155.548223	125.0	511.875789
6	78.0	50.0	32.0	88.0	31.0	88.000000	88.0	88.000000
7	115.0	NaN	NaN	NaN	35.3	155.548223	125.0	511.875789
8	197.0	70.0	45.0	543.0	30.5	543.000000	543.0	543.000000
9	125.0	96.0	NaN	NaN	NaN	155.548223	125.0	511.875789
10	110.0	92.0	NaN	NaN	37.6	155.548223	125.0	511.875789
11	168.0	74.0	NaN	NaN	38.0	155.548223	125.0	511.875789
12	139.0	80.0	NaN	NaN	27.1	155.548223	125.0	511.875789
13	189.0	60.0	23.0	846.0	30.1	846.000000	846.0	846.000000
14	166.0	72.0	19.0	175.0	25.8	175.000000	175.0	175.000000
15	100.0	NaN	NaN	NaN	30.0	155.548223	125.0	511.875789
16	118.0	84.0	47.0	230.0	45.8	230.000000	230.0	230.000000
17	107.0	74.0	NaN	NaN	29.6	155.548223	125.0	511.875789
18	103.0	30.0	38.0	83.0	43.3	83.000000	83.0	83.000000
19	115.0	70.0	30.0	96.0	34.6	96.000000	96.0	96.000000



Remove Rows With Missing Values

```
df2 = df.copy(deep=True)
df2
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	NaN	33.6	0.627	50	1
1	1	85.0	66.0	29.0	NaN	26.6	0.351	31	0
2	8	183.0	64.0	NaN	NaN	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101.0	76.0	48.0	180.0	32.9	0.171	63	0
764	2	122.0	70.0	27.0	NaN	36.8	0.340	27	0
765	5	121.0	72.0	23.0	112.0	26.2	0.245	30	0
766	1	126.0	60.0	NaN	NaN	30.1	0.349	47	1
767	1	93.0	70.0	31.0	NaN	30.4	0.315	23	0

768 rows × 9 columns

```
df2.dropna()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1
6	3	78.0	50.0	32.0	88.0	31.0	0.248	26	1
8	2	197.0	70.0	45.0	543.0	30.5	0.158	53	1
13	1	189.0	60.0	23.0	846.0	30.1	0.398	59	1
...	...	...	...	...	...	...	...	...	...
753	0	181.0	88.0	44.0	510.0	43.3	0.222	26	1
755	1	128.0	88.0	39.0	110.0	36.5	1.057	37	1
760	2	88.0	58.0	26.0	16.0	28.4	0.766	22	0
763	10	101.0	76.0	48.0	180.0	32.9	0.171	63	0
765	5	121.0	72.0	23.0	112.0	26.2	0.245	30	0

392 rows × 9 columns

II) Encoding Categorical Data

•A dataset can contain numerical, categorical, date time, and mixed variables.

•Models based on statistical algorithms, such as machine learning and deep learning, work with numbers.

•A mechanism is needed to convert categorical data to its numeric counterpart so that the data can be used to build statistical models.

•The techniques used to convert numeric data into categorical data are called categorical data encoding schemes.

Types of categorical data type -

• One Hot Encoding

• Label Encoding

• Frequency Encoding

• Ordinal Encoding

• Mean Encoding

In our dataset, there are no categorical variables, hence encoding is not required.

III) Data Discretization

The process of converting continuous numeric values into discrete intervals is called discretization or binning.

Examples: price, age, weight, etc.

Advantages:

• Helpful to handle Outliers.

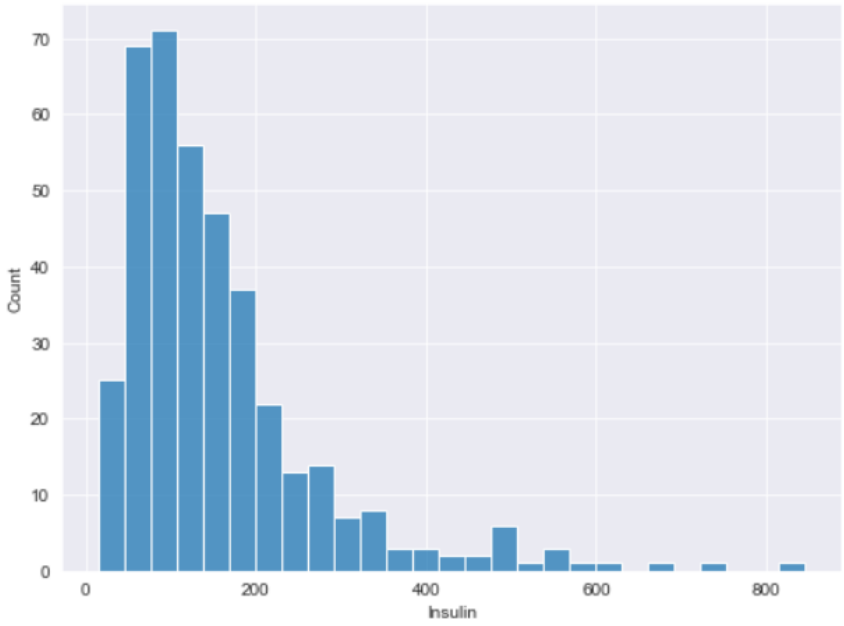
•With discretization, the outliers can be placed into tail intervals along with the remaining inlier values that occur at tails.

• Discretization is particularly helpful in cases where you have skewed distribution of data.

Graph before Discretization

```
sns.histplot(df1['Insulin'])
```

```
<AxesSubplot:xlabel='Insulin', ylabel='Count'>
```



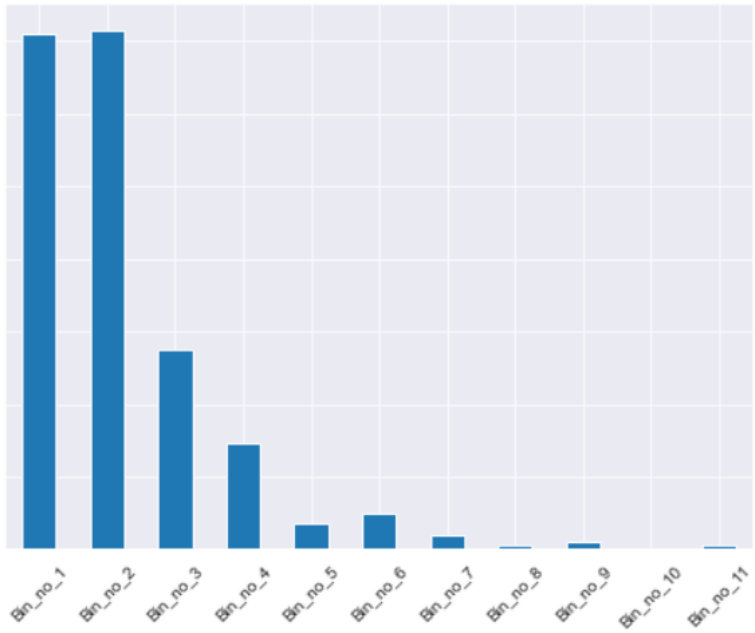
Equal Width Discretization -

• The most common type of discretization approach. Also called as fixed width discretization.

• The width or the size of all the intervals remains the same. An interval is also called a bin.

• Equal width binning divides the numerical predictor into k categories of equal width, where k is chosen by the client or analyst.

Graph after Equal Width Discretization -



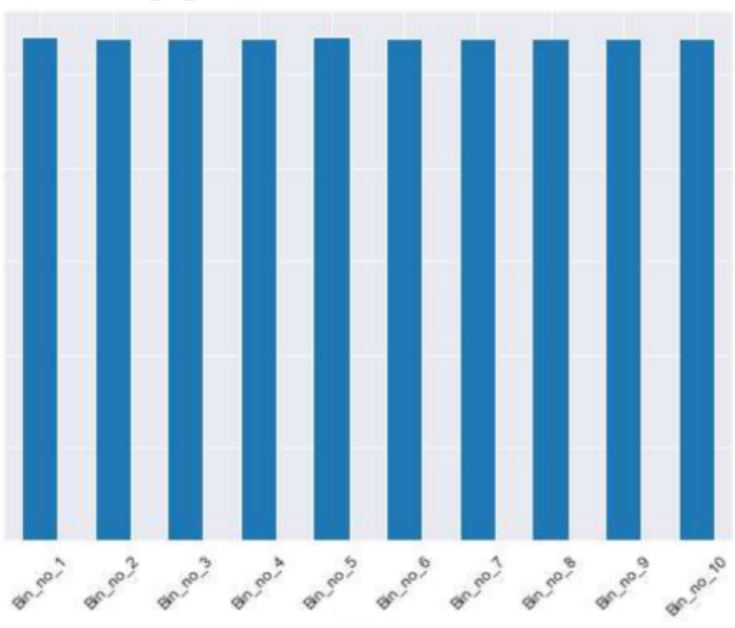
Equal Frequency Discretization -

• In equal frequency discretization, the bin width is adjusted automatically in such a way that each bin contains exactly the same number of records or has the same frequency.

• In equal frequency discretization, the bin interval may not be the same.

• Is an unsupervised discretization technique.

After Equal Frequency Discretization -



IV) Outlier Handling

• An outlier is an observation that lies an abnormal distance from other values in a random sample from a population.

• There are four main techniques to handle outliers:

• 1. You can totally remove the outliers from the dataset.

• 2. You can treat outliers as missing values, and then apply any data imputation technique.

• 3. You can apply discretization techniques to the dataset that will include the outlier along with other data points at the tail.

• 4. You can cap or censor the outliers and replace them with maximum and minimum values that can be found via several techniques.

Outlier trimming -

• Outlier trimming refers to simply removing the outliers beyond a certain threshold value.

• One of the main advantages of outlier trimming is it is extremely quick and doesn't distort the data.

• There are several ways to find the thresholds for outlier trimming.

Outlier Capping -

• The outliers are capped at certain minimum and maximum values.

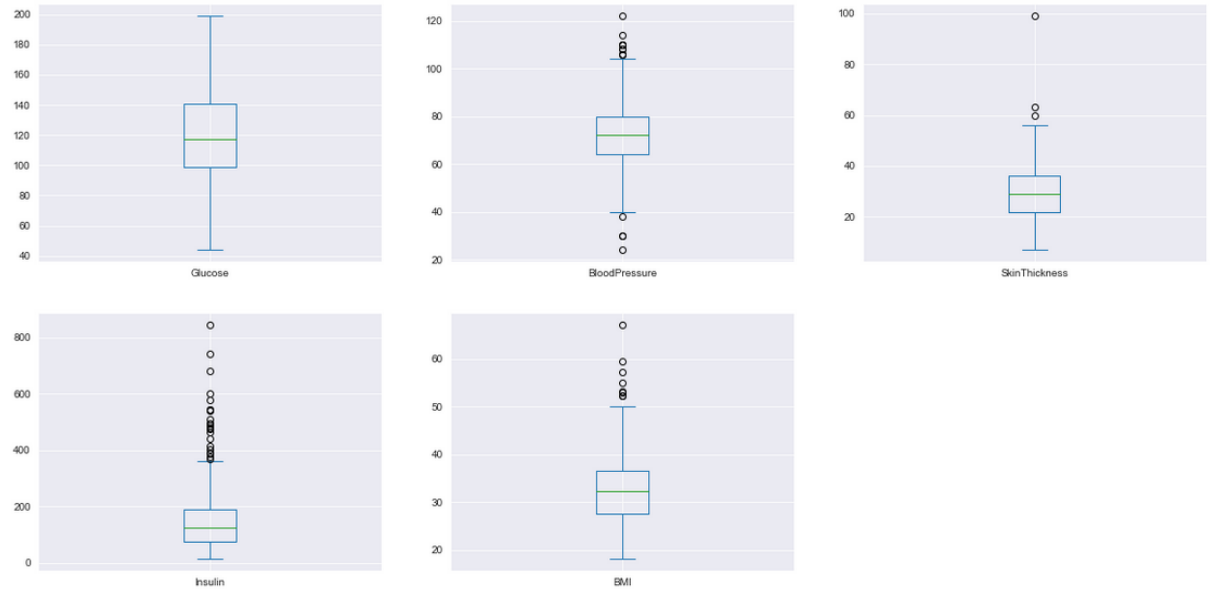
• The rows containing the outliers are not removed from the dataset.

• We will again use the InterQuartile Range technique to find the lower and upper limit for the outliers.

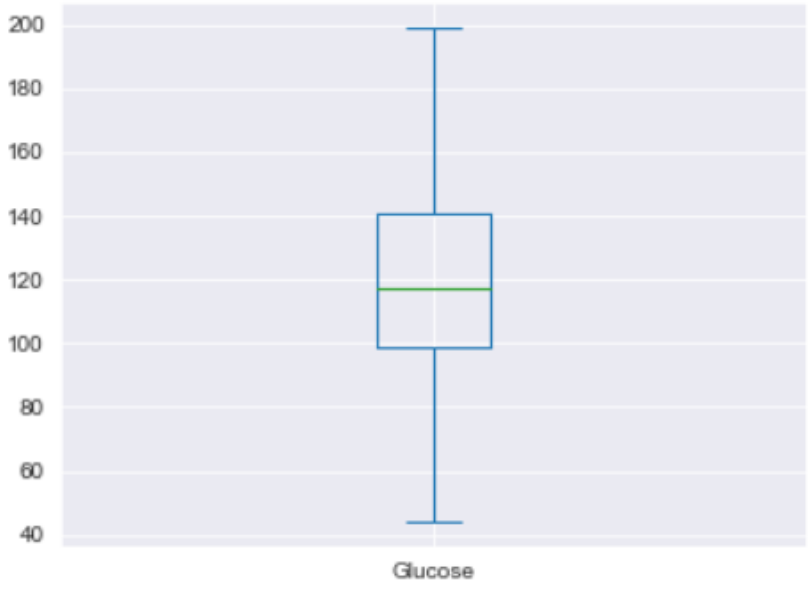
We draw the box plots of all attributes which show the outlier values in those columns.

```
df1.plot(kind= 'box' , subplots=True, layout=(3,3), sharex=False, sharey=False, figsize=(20,15))
```

```
Glucose             AxesSubplot(0.125,0.657941;0.227941x0.222059)
BloodPressure    AxesSubplot(0.398529,0.657941;0.227941x0.222059)
SkinThickness    AxesSubplot(0.672059,0.657941;0.227941x0.222059)
Insulin             AxesSubplot(0.125,0.391471;0.227941x0.222059)
BMI              AxesSubplot(0.398529,0.391471;0.227941x0.222059)
dtype: object
```



After applying various outlier handling techniques, the box plot of each of the columns looks like the following box plot of Glucose column:



V) Feature Selection

• Feature selection is also known as Variable selection or Attribute selection.

• Feature selection is a process where you automatically select those features in your data that contribute most to the prediction variable or output in which you are interested.

• Feature Selection can enhance the performance of a machine learning model as well.

• Three benefits of performing feature selection before modeling your data are:

• Reduces Overfitting: Less redundant data means less opportunity to make decisions based on noise.

• Improves Accuracy: Less misleading data means modeling accuracy improves.

• Reduces Training Time: Less data means that algorithms train faster.

Correlation between the different characteristics. Closer to 1 better is the correlation.

```
corr_matrix = df.corr(method='pearson')
corr_matrix
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.128135	0.214178	0.100239	0.082171	0.021719	-0.033523	0.544341	0.221898
Glucose	0.128135	1.000000	0.223192	0.228043	0.581186	0.232771	0.137246	0.267136	0.494650
BloodPressure	0.214178	0.223192	1.000000	0.226839	0.098272	0.289230	-0.002805	0.330107	0.170589
SkinThickness	0.100239	0.228043	0.226839	1.000000	0.184888	0.648214	0.115016	0.166816	0.259491
Insulin	0.082171	0.581186	0.098272	0.184888	1.000000	0.228050	0.130395	0.220261	0.303454
BMI	0.021719	0.232771	0.289230	0.648214	0.228050	1.000000	0.155382	0.025841	0.313680
DiabetesPedigreeFunction	-0.033523	0.137246	-0.002805	0.115016	0.130395	0.155382	1.000000	0.033561	0.173844
Age	0.544341	0.267136	0.330107	0.166816	0.220261	0.025841	0.033561	1.000000	0.238356
Outcome	0.221898	0.494650	0.170589	0.259491	0.303454	0.313680	0.173844	0.238356	1.000000

There is no strong correlation between the features. The 'strongest' ones are the following (as expected):

- Age x pregnancies (0.68) - Older women tend to have higher number of pregnancies
- Glucose and insulin (0.58)
- Glucose x outcome (0.52) - Women that have higher level of glucose tend to have higher level of insulin and have DM
- Skin fold thickness x BMI (0.66) - Women with higher skin fold thickness value have higher BMI (and probably are overweight/obese)

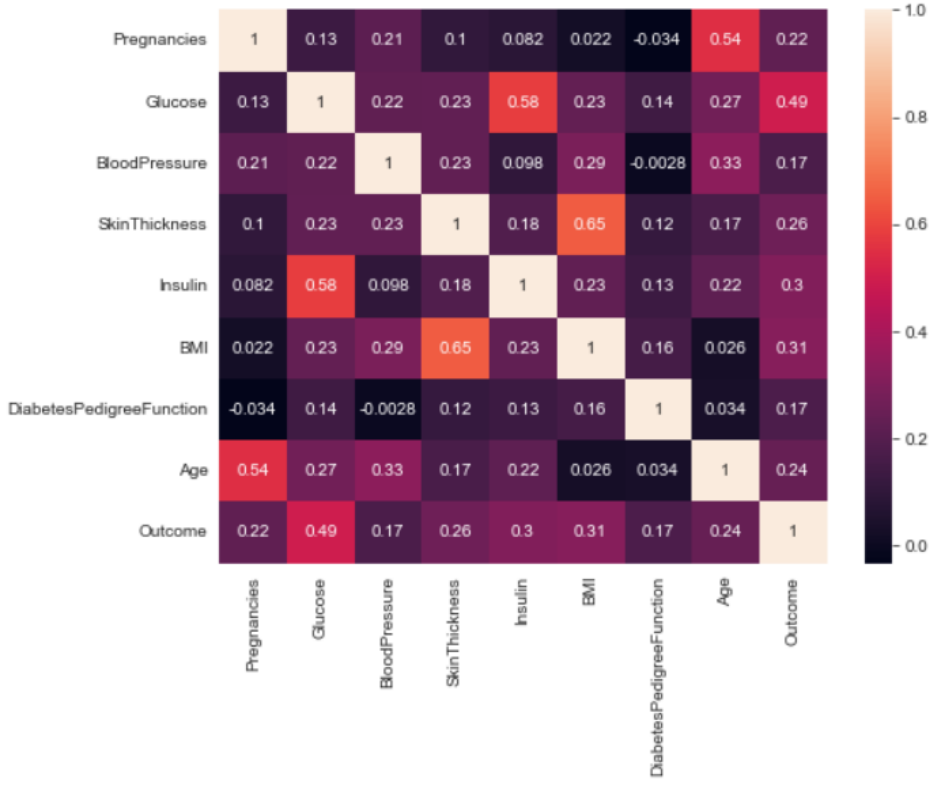
Negative correlation:

- BMI x Pregnancies (-0.025)
- Blood Pressure x Diabetes Pedigree Function (-0.016)

Heatmap makes it easy to identify which features are most related to the target variable, we will plot a heatmap of correlated features using the seaborn library.

```
sns.heatmap(corr_matrix, annot = True)
```

```
<AxesSubplot:>
```



4. Conclusion

Hence, we can see that in this project we used various data preprocessing techniques to remove the noise from the previous original dataset and also organised the data into meaningful intervals which are better suited for machine learning models and for understanding the data in general.

5. Acknowledgement

Both of the group members thank everyone for their best wishes and helping us to make this project a success. We are very grateful to one's who dedicated their valuable time to guide us throughout this project. We would like to acknowledge IIIT-NR for providing the necessary resources and facilities to implement the project successfully.

References

https://www.kaggle.com/uciml/pima-indians-diabetes-database