# Diabetes Disease Prediction

**Dataset CSV file:** diabetes_012_health_indicators_BRFSS2015.csv

**Group No.:** 18

**Group Members:**

1. VARINDER SINGH - 2021fc04070@wilp.bits-pilani.ac.in

2. BANDARU RAJA SEKHAR - 2021fc04074@wilp.bits-pilani.ac.in

3. MIKHIL. P.A. - 2021fc04326@wilp.bits-pilani.ac.in

## Importing Libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import warnings
```

## Importing Dataset

```
raw_df = pd.read_csv("diabetes_012_health_indicators_BRFSS2015.csv")
```

```
raw_df
```

| | Diabetes_012 | HighBP | HighChol | CholCheck | BMI | Smoker | Stroke | HeartDiseaseorAttack | PhysActivity | Fruits | ... | AnyHealthcare | NoDocbcCost | GenHlth | MentHlth |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | 1.0 | 1.0 | 1.0 | 40.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 5.0 | 18.0 |
| 1 | 0.0 | 0.0 | 0.0 | 0.0 | 25.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | ... | 0.0 | 1.0 | 3.0 | 0.0 |
| 2 | 0.0 | 1.0 | 1.0 | 1.0 | 28.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | ... | 1.0 | 1.0 | 5.0 | 30.0 |
| 3 | 0.0 | 1.0 | 0.0 | 1.0 | 27.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | ... | 1.0 | 0.0 | 2.0 | 0.0 |
| 4 | 0.0 | 1.0 | 1.0 | 1.0 | 24.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | ... | 1.0 | 0.0 | 2.0 | 3.0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 253675 | 0.0 | 1.0 | 1.0 | 1.0 | 45.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | ... | 1.0 | 0.0 | 3.0 | 0.0 |
| 253676 | 2.0 | 1.0 | 1.0 | 1.0 | 18.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 4.0 | 0.0 |
| 253677 | 0.0 | 0.0 | 0.0 | 1.0 | 28.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | ... | 1.0 | 0.0 | 1.0 | 0.0 |
| 253678 | 0.0 | 1.0 | 0.0 | 1.0 | 23.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | ... | 1.0 | 0.0 | 3.0 | 0.0 |
| 253679 | 2.0 | 1.0 | 1.0 | 1.0 | 25.0 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | ... | 1.0 | 0.0 | 2.0 | 0.0 |

253680 rows × 22 columns

```
raw_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253680 entries, 0 to 253679
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   Diabetes_012          253680 non-null  float64
 1   HighBP                253680 non-null  float64
 2   HighChol              253680 non-null  float64
 3   CholCheck             253680 non-null  float64
 4   BMI                   253680 non-null  float64
 5   Smoker                253680 non-null  float64
 6   Stroke                253680 non-null  float64
 7   HeartDiseaseorAttack  253680 non-null  float64
 8   PhysActivity          253680 non-null  float64
 9   Fruits                253680 non-null  float64
 10  Veggies               253680 non-null  float64
 11  HvyAlcoholConsump     253680 non-null  float64
 12  AnyHealthcare         253680 non-null  float64
 13  NoDocbcCost           253680 non-null  float64
 14  GenHlth               253680 non-null  float64
 15  MentHlth              253680 non-null  float64
 16  PhysHlth              253680 non-null  float64
 17  DiffWalk              253680 non-null  float64
 18  Sex                   253680 non-null  float64
 19  Age                   253680 non-null  float64
 20  Education             253680 non-null  float64
 21  Income                253680 non-null  float64
dtypes: float64(22)
memory usage: 42.6 MB
```

```
raw_df.describe()
```

| | Diabetes_012 | HighBP | HighChol | CholCheck | BMI | Smoker | Stroke | HeartDiseaseorAttack | PhysActivity | Fruits |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 253680.000000 | 253680.000000 | 253680.000000 | 253680.000000 | 253680.000000 | 253680.000000 | 253680.000000 | 253680.000000 | 253680.000000 | 253680. |
| mean | 0.296921 | 0.429001 | 0.424121 | 0.962670 | 28.382364 | 0.443169 | 0.040571 | 0.094186 | 0.756544 | 0.63425 |
| std | 0.698160 | 0.494934 | 0.494210 | 0.189571 | 6.608694 | 0.496761 | 0.197294 | 0.292087 | 0.429169 | 0.48163 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 12.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00000 |
| 25% | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 24.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.00000 |
| 50% | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 27.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 1.00000 |
| 75% | 0.000000 | 1.000000 | 1.000000 | 1.000000 | 31.000000 | 1.000000 | 0.000000 | 0.000000 | 1.000000 | 1.00000 |
| max | 2.000000 | 1.000000 | 1.000000 | 1.000000 | 98.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.00000 |

8 rows × 22 columns

Data Visualization and Exploration

### Print 2 rows for sanity check

```
raw_df.head(2)
```

| | Diabetes_012 | HighBP | HighChol | CholCheck | BMI | Smoker | Stroke | HeartDiseaseorAttack | PhysActivity | Fruits | ... | AnyHealthcare | NoDocbcCost | GenHlth | MentHlth | PhysHl |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | 1.0 | 1.0 | 1.0 | 40.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 5.0 | 18.0 | 15.0 |
| 1 | 0.0 | 0.0 | 0.0 | 0.0 | 25.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | ... | 0.0 | 1.0 | 3.0 | 0.0 | 0.0 |

2 rows × 22 columns

### Handling duplicates

```
raw_df.shape
```

```
(253680, 22)
```

```
print("Number of Duplicates before processing the dataset: ", raw_df.duplicated().sum())
```

```
Number of Duplicates before processing the dataset:  23899
```

```
raw_df = raw_df.drop_duplicates(keep='last')
raw_df.reset_index(inplace = True, drop = True)

print("Number of Duplicates after processing the dataset: ", raw_df.duplicated().sum())
```

```
Number of Duplicates after processing the dataset:  0
```

```
raw_df.shape
```

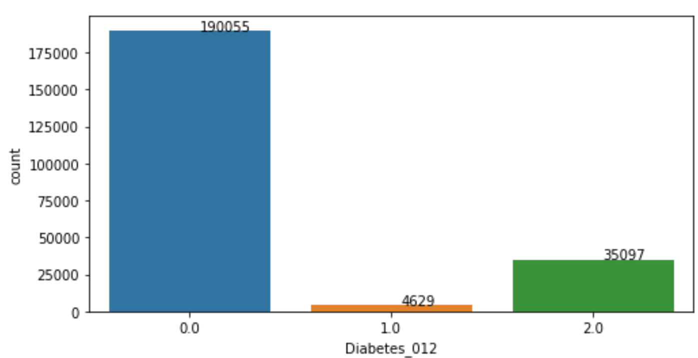
```
(229781, 22)
```

## Class imbalance

Imbalanced data refers to those types of datasets where the target class has an uneven distribution of observations, i.e one class label has a very high number of observations and the other has a very low number of observations.

There is an extreme class imbalance in the given diabetes dataset.

```
plt.figure(figsize=(8, 4))
ax = sns.countplot( x="Diabetes_012", data=raw_df )

for p in ax.patches:
   ax.annotate('{:}'.format(p.get_height()), (p.get_x()+0.45, p.get_height()+0.55))
```

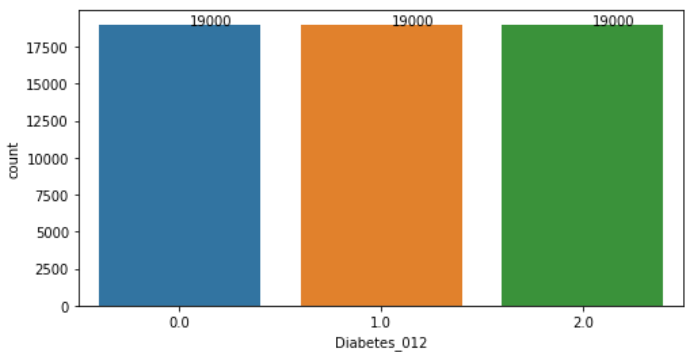


### Oversampling to balance the data

Random oversampling involves randomly duplicating examples from the minority class and adding them to the training dataset. This technique can be effective for those machine learning algorithms that are affected by a skewed distribution and where multiple duplicate examples for a given class can influence the fit of the model. It might be useful to tune the target class distribution.

```
shuffled_df = raw_df.sample( frac=1 ,random_state=42)

zero_df = shuffled_df.loc[shuffled_df['Diabetes_012'] == 0.0].sample(n=19000, replace=True)
one_df = shuffled_df.loc[shuffled_df['Diabetes_012'] == 1.0].sample(n=19000, replace=True)
two_df = shuffled_df.loc[shuffled_df['Diabetes_012'] == 2.0].sample(n=19000, replace=True)

balanced_df = pd.concat([zero_df, one_df, two_df])
balanced_df.reset_index(inplace = True, drop = True)
```

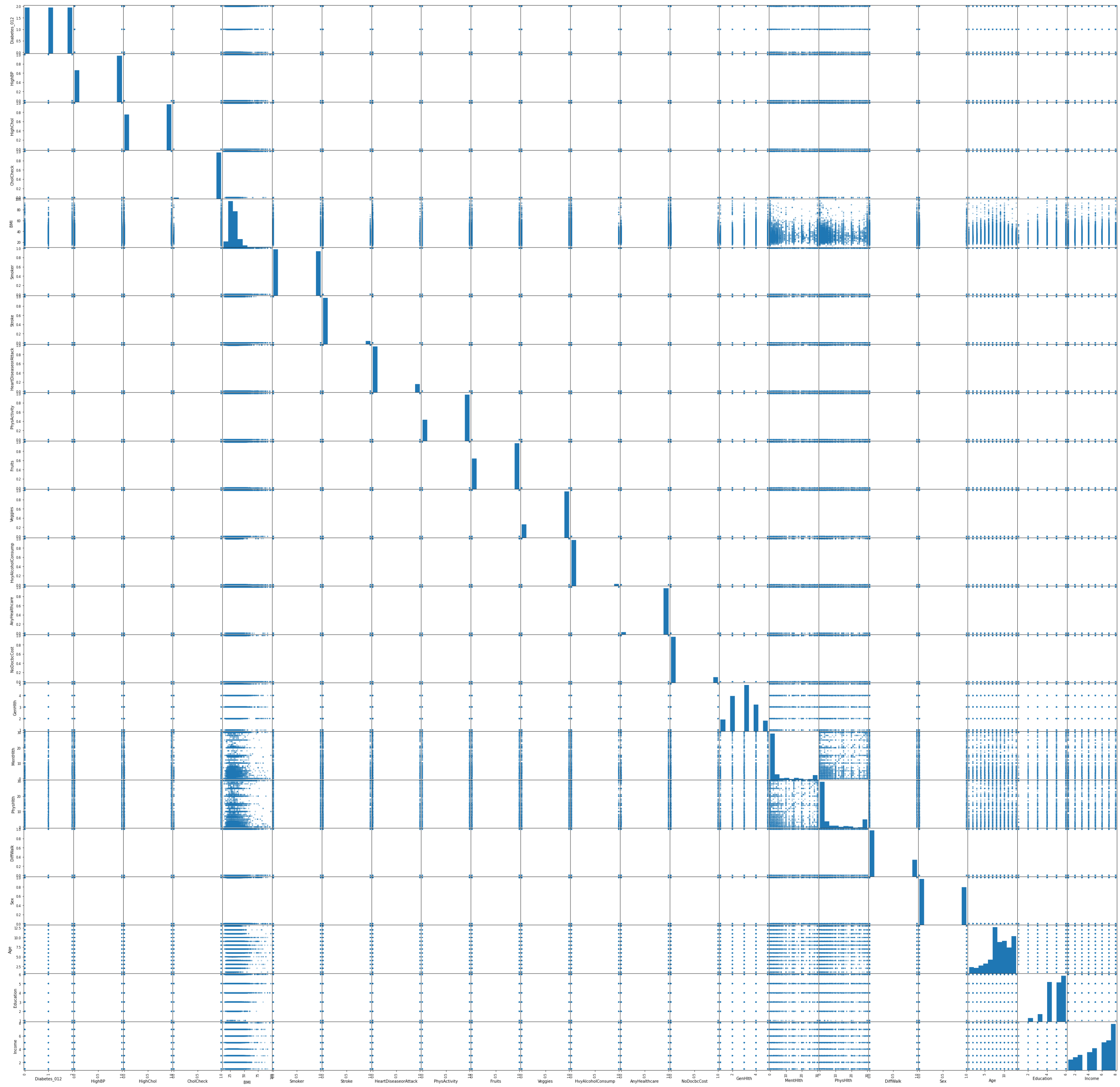
```
plt.figure(figsize=(8, 4))
ax = sns.countplot( x="Diabetes_012", data=balanced_df )

for p in ax.patches:
   ax.annotate('{:}'.format(p.get_height()), (p.get_x()+0.45, p.get_height()+0.55))
```




Here, we are sampling for class output of 0.0, 1.0 and 2.0. to 19000 individually class.

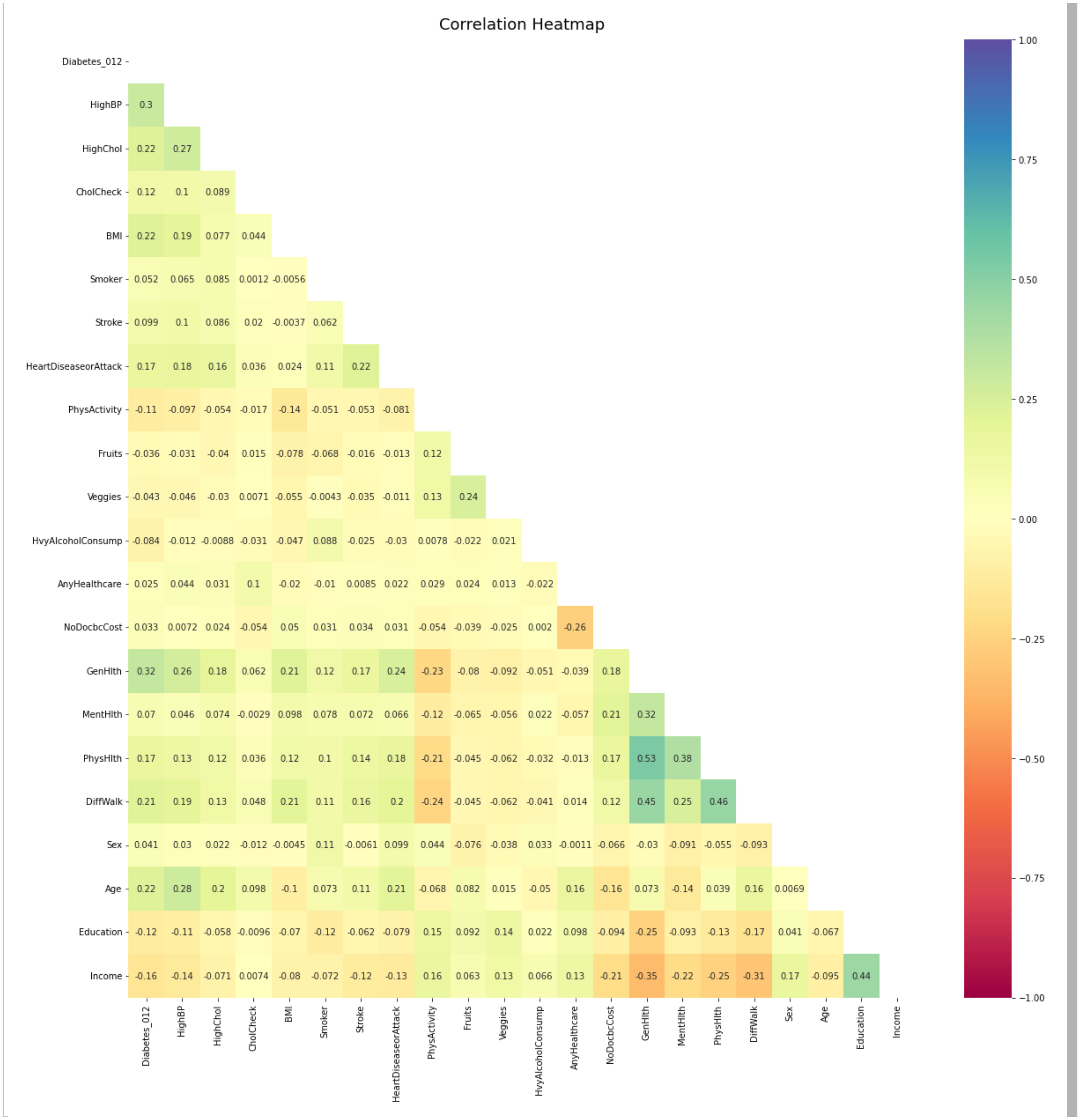
## Correlational Analysis

```
from pandas.plotting import scatter_matrix
scatter_matrix(balanced_df, figsize=(50, 50))
plt.show()
```




```
corr = balanced_df.corr()

plt.figure(figsize=(20, 20))
mask = np.triu( np.ones_like(corr) )
hm = sns.heatmap( corr, mask=mask, vmin=-1, vmax=1, annot=True, cmap='Spectral' )
hm.set_title('Correlation Heatmap', fontdict={'fontsize':18}, pad=12)
```

```
Text(0.5, 1.0, 'Correlation Heatmap')
```




There are no attributes which are highly correlated with each other; which can be considered to make changes to any feature selection. Therefore, we are not making any changes to feature selection. We are using all the column as provided.

### Data Pre-processing and cleaning

```
balanced_df.info()
```

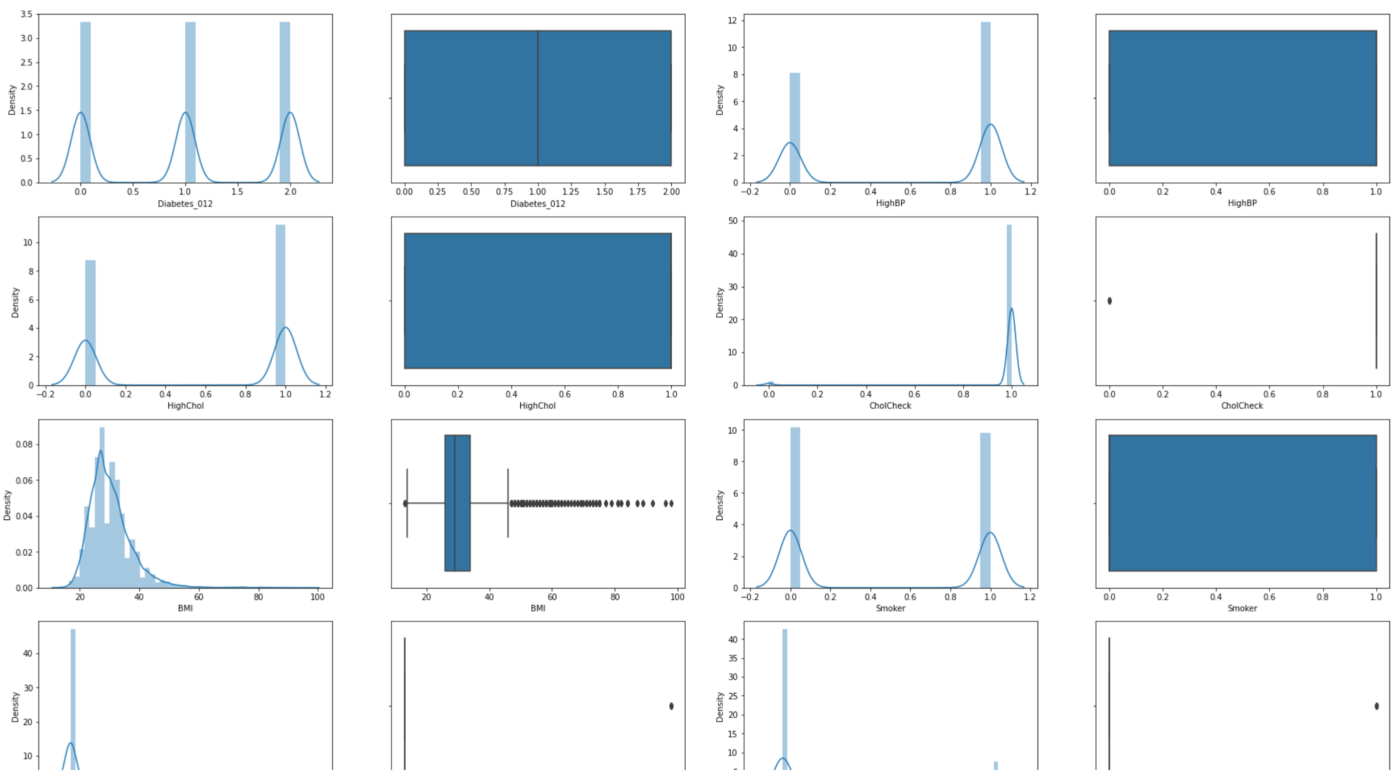
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57000 entries, 0 to 56999
Data columns (total 22 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
```

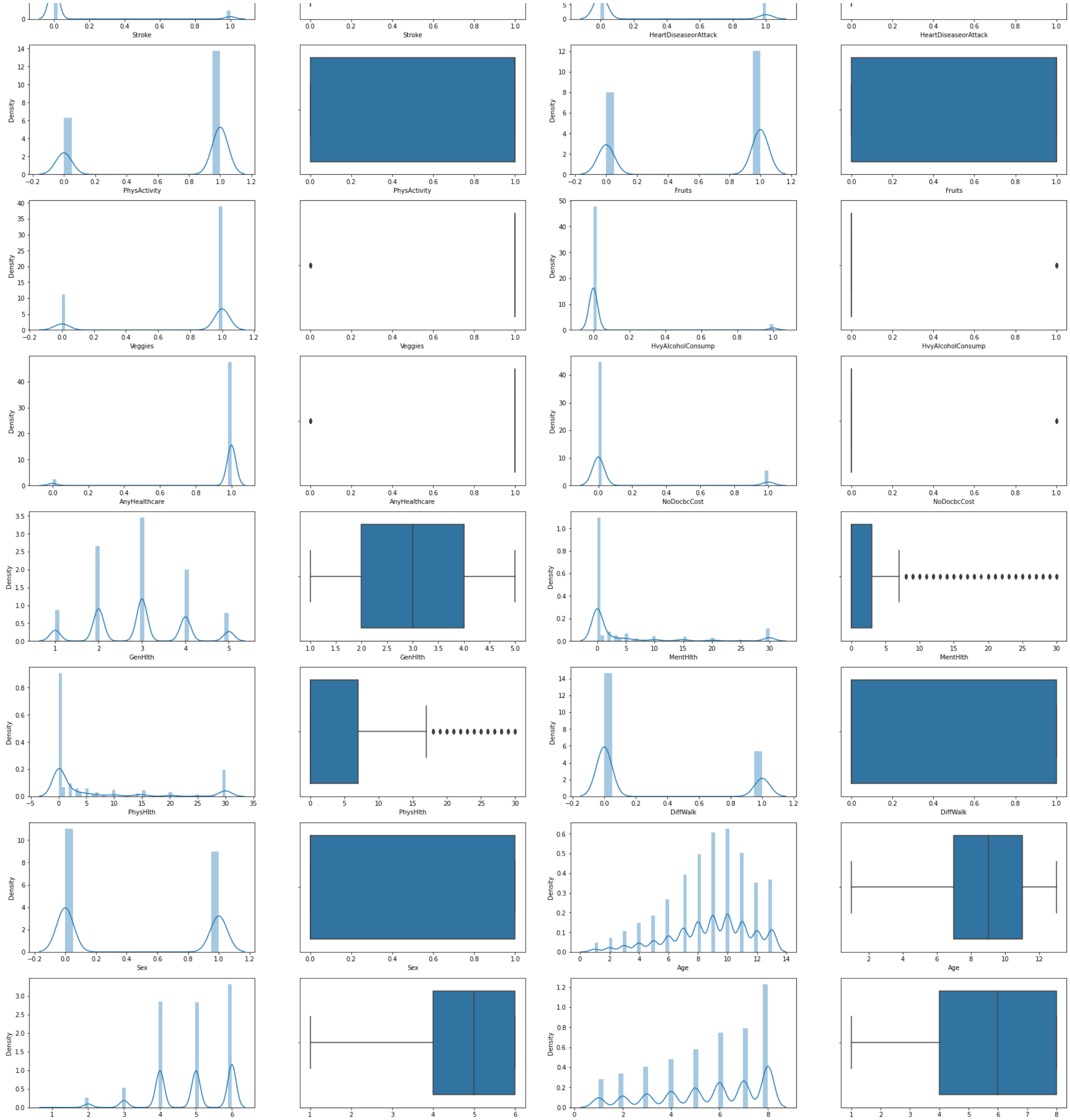
```
 0   Diabetes_012          57000 non-null  float64
 1   HighBP                57000 non-null  float64
 2   HighChol              57000 non-null  float64
 3   CholCheck             57000 non-null  float64
 4   BMI                   57000 non-null  float64
 5   Smoker                57000 non-null  float64
 6   Stroke                57000 non-null  float64
 7   HeartDiseaseorAttack  57000 non-null  float64
 8   PhysActivity          57000 non-null  float64
 9   Fruits                57000 non-null  float64
 10  Veggies               57000 non-null  float64
 11  HvyAlcoholConsump     57000 non-null  float64
 12  AnyHealthcare         57000 non-null  float64
 13  NoDocbcCost           57000 non-null  float64
 14  GenHlth               57000 non-null  float64
 15  MentHlth              57000 non-null  float64
 16  PhysHlth              57000 non-null  float64
 17  DiffWalk              57000 non-null  float64
 18  Sex                   57000 non-null  float64
 19  Age                   57000 non-null  float64
 20  Education             57000 non-null  float64
 21  Income                57000 non-null  float64
dtypes: float64(22)
memory usage: 9.6 MB
```

## Checking Data Distribution and Outlier Analysis on dataset

```
col = list(balanced_df.columns)
```

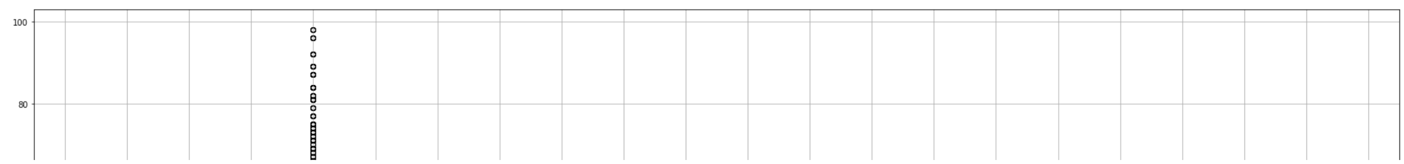
```
warnings.filterwarnings('ignore')

plt.figure(figsize=(30,50))
num = 1
for i in col:
    plt.subplot(11,4,num)
    sns.distplot(balanced_df[str(i)])
    num = num + 1
    plt.subplot(11,4,num)
    sns.boxplot(balanced_df[str(i)])
    num = num + 1
```






```
balanced_df.boxplot(figsize = (30,10), column = col)
```

```
<AxesSubplot:>
```




## Handling the Outliers

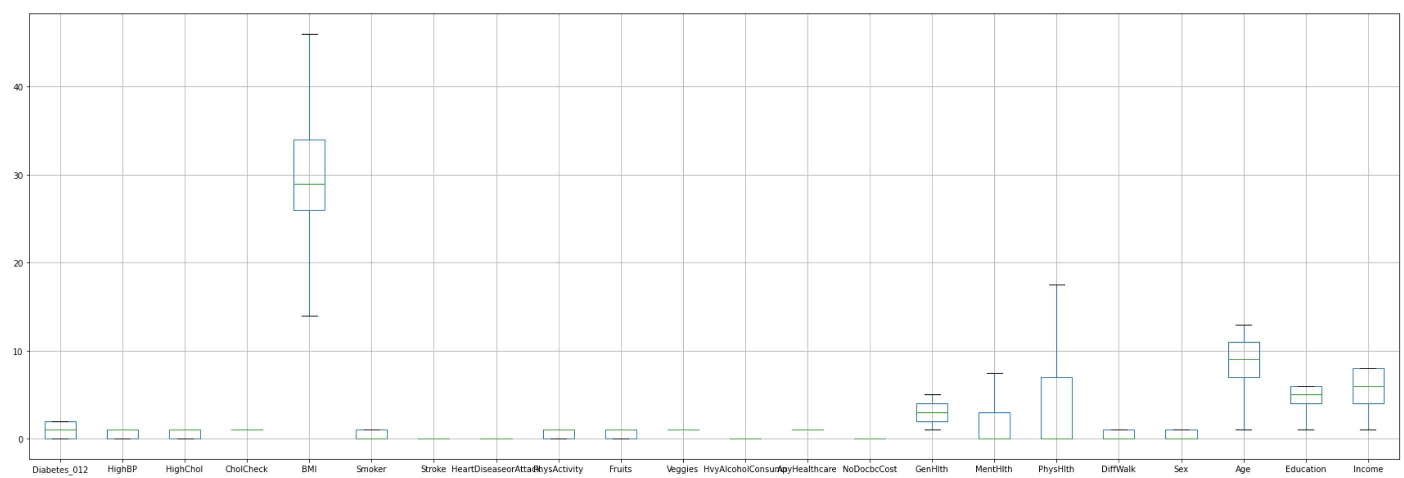
**Winsorization of Outliers** is the process of replacing the extreme values of statistical data in order to limit the effect of the outliers on the calculations or the results obtained by using that data.

```
for i in col:
    percentile25 = balanced_df[str(i)].quantile(0.25)
    percentile75 = balanced_df[str(i)].quantile(0.75)
    iqr = percentile75 - percentile25
    upper_limit = percentile75 + 1.5 * iqr
    lower_limit = percentile25 - 1.5 * iqr
    balanced_df[str(i)] = np.where(balanced_df[str(i)] >= upper_limit,
                                    upper_limit,
                                        np.where(balanced_df[str(i)] <= lower_limit,
                                        lower_limit,
                                            balanced_df[str(i)]
                                        )
                                    )
```

**After the processing the data, no outlier is present in dataset.**

```
balanced_df.boxplot(figsize = (30,10), column = col)
```

```
<AxesSubplot:>
```




## Standardising the dataset

Standard Scaler helps to get standardized distribution, with a zero mean and standard deviation of one (unit variance). It standardizes features by subtracting the mean value from the feature and then dividing the result by feature standard deviation.

```
balanced_df
```

| | Diabetes_012 | HighBP | HighChol | CholCheck | BMI | Smoker | Stroke | HeartDiseaseorAttack | PhysActivity | Fruits | ... | AnyHealthcare | NoDocbcCost | GenHlth | MentHlth | Pl |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | 1.0 | 1.0 | 1.0 | 33.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | ... | 1.0 | 0.0 | 1.0 | 0.0 | 0. |
| 1 | 0.0 | 1.0 | 1.0 | 1.0 | 29.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | ... | 1.0 | 0.0 | 3.0 | 0.0 | 0. |
| 2 | 0.0 | 1.0 | 1.0 | 1.0 | 46.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | ... | 1.0 | 0.0 | 5.0 | 0.0 | 17 |
| 3 | 0.0 | 0.0 | 0.0 | 1.0 | 18.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | ... | 1.0 | 0.0 | 2.0 | 5.0 | 0. |
| 4 | 0.0 | 0.0 | 1.0 | 1.0 | 33.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 4.0 | 0.0 | 10 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 56995 | 2.0 | 1.0 | 1.0 | 1.0 | 46.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 4.0 | 5.0 | 3. |
| 56996 | 2.0 | 1.0 | 1.0 | 1.0 | 23.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 3.0 | 0.0 | 0. |
| 56997 | 2.0 | 0.0 | 1.0 | 1.0 | 33.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 5.0 | 0.0 | 17 |
| 56998 | 2.0 | 1.0 | 1.0 | 1.0 | 34.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | ... | 1.0 | 0.0 | 3.0 | 0.0 | 8. |
| 56999 | 2.0 | 1.0 | 0.0 | 1.0 | 32.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | ... | 1.0 | 0.0 | 3.0 | 0.0 | 0. |

57000 rows × 22 columns

```
balanced_df.iloc[: , :]
```

| | Diabetes_012 | HighBP | HighChol | CholCheck | BMI | Smoker | Stroke | HeartDiseaseorAttack | PhysActivity | Fruits | ... | AnyHealthcare | NoDocbcCost | GenHlth | MentHlth | Pl |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | 1.0 | 1.0 | 1.0 | 33.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | ... | 1.0 | 0.0 | 1.0 | 0.0 | 0. |
| 1 | 0.0 | 1.0 | 1.0 | 1.0 | 29.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | ... | 1.0 | 0.0 | 3.0 | 0.0 | 0. |
| 2 | 0.0 | 1.0 | 1.0 | 1.0 | 46.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | ... | 1.0 | 0.0 | 5.0 | 0.0 | 17 |
| 3 | 0.0 | 0.0 | 0.0 | 1.0 | 18.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | ... | 1.0 | 0.0 | 2.0 | 5.0 | 0. |
| 4 | 0.0 | 0.0 | 1.0 | 1.0 | 33.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 4.0 | 0.0 | 10 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 56995 | 2.0 | 1.0 | 1.0 | 1.0 | 46.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 4.0 | 5.0 | 3. |
| 56996 | 2.0 | 1.0 | 1.0 | 1.0 | 23.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 3.0 | 0.0 | 0. |
| 56997 | 2.0 | 0.0 | 1.0 | 1.0 | 33.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 5.0 | 0.0 | 17 |
| 56998 | 2.0 | 1.0 | 1.0 | 1.0 | 34.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | ... | 1.0 | 0.0 | 3.0 | 0.0 | 8. |
| 56999 | 2.0 | 1.0 | 0.0 | 1.0 | 32.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | ... | 1.0 | 0.0 | 3.0 | 0.0 | 0. |

57000 rows × 22 columns

```
balanced_df.iloc[: , 1:]
```

| | HighBP | HighChol | CholCheck | BMI | Smoker | Stroke | HeartDiseaseorAttack | PhysActivity | Fruits | Veggies | ... | AnyHealthcare | NoDocbcCost | GenHlth | MentHlth | PhysHlt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | 1.0 | 1.0 | 33.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | ... | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| 1 | 1.0 | 1.0 | 1.0 | 29.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | ... | 1.0 | 0.0 | 3.0 | 0.0 | 0.0 |
| 2 | 1.0 | 1.0 | 1.0 | 46.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | ... | 1.0 | 0.0 | 5.0 | 0.0 | 17.5 |
| 3 | 0.0 | 0.0 | 1.0 | 18.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | | 1.0 | 0.0 | 2.0 | 5.0 | 0.0 |

```
#Standardising the data

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

df_standardised = sc.fit_transform( balanced_df.iloc[: , 1:] )
df_standardised
```

```
#Adding the column name to standardised dataset
df_standardised = pd.DataFrame(df_standardised,
                               columns = col[1:])
```

```
df_standardised
```

| | HighBP | HighChol | CholCheck | BMI | Smoker | Stroke | HeartDiseaseorAttack | PhysActivity | Fruits | Veggies | ... | AnyHealthcare | NoDocbcCost | GenHlth |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.826952 | 0.880645 | 0.0 | 0.476193 | 1.018665 | 0.0 | 0.0 | 0.677161 | -1.227974 | 0.0 | ... | 0.0 | 0.0 | -1.785175 |
| 1 | 0.826952 | 0.880645 | 0.0 | -0.158802 | -0.981677 | 0.0 | 0.0 | 0.677161 | 0.814350 | 0.0 | ... | 0.0 | 0.0 | 0.082197 |
| 2 | 0.826952 | 0.880645 | 0.0 | 2.539925 | 1.018665 | 0.0 | 0.0 | 0.677161 | 0.814350 | 0.0 | ... | 0.0 | 0.0 | 1.949569 |
| 3 | -1.209260 | -1.135532 | 0.0 | -1.905037 | -0.981677 | 0.0 | 0.0 | 0.677161 | 0.814350 | 0.0 | ... | 0.0 | 0.0 | -0.851489 |
| 4 | -1.209260 | 0.880645 | 0.0 | 0.476193 | 1.018665 | 0.0 | 0.0 | -1.476754 | -1.227974 | 0.0 | ... | 0.0 | 0.0 | 1.015883 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 56995 | 0.826952 | 0.880645 | 0.0 | 2.539925 | -0.981677 | 0.0 | 0.0 | -1.476754 | -1.227974 | 0.0 | ... | 0.0 | 0.0 | 1.015883 |
| 56996 | 0.826952 | 0.880645 | 0.0 | -1.111293 | -0.981677 | 0.0 | 0.0 | -1.476754 | -1.227974 | 0.0 | ... | 0.0 | 0.0 | 0.082197 |
| 56997 | -1.209260 | 0.880645 | 0.0 | 0.476193 | 1.018665 | 0.0 | 0.0 | -1.476754 | -1.227974 | 0.0 | ... | 0.0 | 0.0 | 1.949569 |
| 56998 | 0.826952 | 0.880645 | 0.0 | 0.634942 | -0.981677 | 0.0 | 0.0 | -1.476754 | 0.814350 | 0.0 | ... | 0.0 | 0.0 | 0.082197 |
| 56999 | 0.826952 | -1.135532 | 0.0 | 0.317444 | -0.981677 | 0.0 | 0.0 | 0.677161 | -1.227974 | 0.0 | ... | 0.0 | 0.0 | 0.082197 |

57000 rows × 21 columns

```
#Resetting the index
df_standardised = df_standardised.reset_index(drop = True)
raw_df['Diabetes_012'] = raw_df['Diabetes_012'].reset_index(drop = True)
```

```
#Adding target column to standardised dataset
df_standardised['Diabetes_012'] = raw_df['Diabetes_012']
df_standardised
```

| | HighBP | HighChol | CholCheck | BMI | Smoker | Stroke | HeartDiseaseorAttack | PhysActivity | Fruits | Veggies | ... | NoDocbcCost | GenHlth | MentHlth | Phy |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.826952 | 0.880645 | 0.0 | 0.476193 | 1.018665 | 0.0 | 0.0 | 0.677161 | -1.227974 | 0.0 | ... | 0.0 | -1.785175 | -0.624918 | -0.6 |
| 1 | 0.826952 | 0.880645 | 0.0 | -0.158802 | -0.981677 | 0.0 | 0.0 | 0.677161 | 0.814350 | 0.0 | ... | 0.0 | 0.082197 | -0.624918 | -0.6 |
| 2 | 0.826952 | 0.880645 | 0.0 | 2.539925 | 1.018665 | 0.0 | 0.0 | 0.677161 | 0.814350 | 0.0 | ... | 0.0 | 1.949569 | -0.624918 | 1.96 |
| 3 | -1.209260 | -1.135532 | 0.0 | -1.905037 | -0.981677 | 0.0 | 0.0 | 0.677161 | 0.814350 | 0.0 | ... | 0.0 | -0.851489 | 1.105408 | -0.6 |
| 4 | -1.209260 | 0.880645 | 0.0 | 0.476193 | 1.018665 | 0.0 | 0.0 | -1.476754 | -1.227974 | 0.0 | ... | 0.0 | 1.015883 | -0.624918 | 0.83 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 56995 | 0.826952 | 0.880645 | 0.0 | 2.539925 | -0.981677 | 0.0 | 0.0 | -1.476754 | -1.227974 | 0.0 | ... | 0.0 | 1.015883 | 1.105408 | -0.2 |
| 56996 | 0.826952 | 0.880645 | 0.0 | -1.111293 | -0.981677 | 0.0 | 0.0 | -1.476754 | -1.227974 | 0.0 | ... | 0.0 | 0.082197 | -0.624918 | -0.6 |
| 56997 | -1.209260 | 0.880645 | 0.0 | 0.476193 | 1.018665 | 0.0 | 0.0 | -1.476754 | -1.227974 | 0.0 | ... | 0.0 | 1.949569 | -0.624918 | 1.96 |
| 56998 | 0.826952 | 0.880645 | 0.0 | 0.634942 | -0.981677 | 0.0 | 0.0 | -1.476754 | 0.814350 | 0.0 | ... | 0.0 | 0.082197 | -0.624918 | 0.53 |
| 56999 | 0.826952 | -1.135532 | 0.0 | 0.317444 | -0.981677 | 0.0 | 0.0 | 0.677161 | -1.227974 | 0.0 | ... | 0.0 | 0.082197 | -0.624918 | -0.6 |

57000 rows × 22 columns

Model Building

```
df_standardised
```

| | HighBP | HighChol | CholCheck | BMI | Smoker | Stroke | HeartDiseaseorAttack | PhysActivity | Fruits | Veggies | ... | NoDocbcCost | GenHlth | MentHlth | Phy |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.826952 | 0.880645 | 0.0 | 0.476193 | 1.018665 | 0.0 | 0.0 | 0.677161 | -1.227974 | 0.0 | ... | 0.0 | -1.785175 | -0.624918 | -0.6 |
| 1 | 0.826952 | 0.880645 | 0.0 | -0.158802 | -0.981677 | 0.0 | 0.0 | 0.677161 | 0.814350 | 0.0 | ... | 0.0 | 0.082197 | -0.624918 | -0.6 |
| 2 | 0.826952 | 0.880645 | 0.0 | 2.539925 | 1.018665 | 0.0 | 0.0 | 0.677161 | 0.814350 | 0.0 | ... | 0.0 | 1.949569 | -0.624918 | 1.96 |
| 3 | -1.209260 | -1.135532 | 0.0 | -1.905037 | -0.981677 | 0.0 | 0.0 | 0.677161 | 0.814350 | 0.0 | ... | 0.0 | -0.851489 | 1.105408 | -0.6 |
| 4 | -1.209260 | 0.880645 | 0.0 | 0.476193 | 1.018665 | 0.0 | 0.0 | -1.476754 | -1.227974 | 0.0 | ... | 0.0 | 1.015883 | -0.624918 | 0.83 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 56995 | 0.826952 | 0.880645 | 0.0 | 2.539925 | -0.981677 | 0.0 | 0.0 | -1.476754 | -1.227974 | 0.0 | ... | 0.0 | 1.015883 | 1.105408 | -0.2 |
| 56996 | 0.826952 | 0.880645 | 0.0 | -1.111293 | -0.981677 | 0.0 | 0.0 | -1.476754 | -1.227974 | 0.0 | ... | 0.0 | 0.082197 | -0.624918 | -0.6 |
| 56997 | -1.209260 | 0.880645 | 0.0 | 0.476193 | 1.018665 | 0.0 | 0.0 | -1.476754 | -1.227974 | 0.0 | ... | 0.0 | 1.949569 | -0.624918 | 1.96 |
| 56998 | 0.826952 | 0.880645 | 0.0 | 0.634942 | -0.981677 | 0.0 | 0.0 | -1.476754 | 0.814350 | 0.0 | ... | 0.0 | 0.082197 | -0.624918 | 0.53 |
| 56999 | 0.826952 | -1.135532 | 0.0 | 0.317444 | -0.981677 | 0.0 | 0.0 | 0.677161 | -1.227974 | 0.0 | ... | 0.0 | 0.082197 | -0.624918 | -0.6 |

57000 rows × 22 columns

# Splitting the dataset into training and test sets as per below conditions:

Case 1 : Train = 90 % Test = 10%

Case 2 : Train = 50 % Test = 50%

```
X = df_standardised.iloc[:, :-1].values
Y = df_standardised.iloc[:, -1].values
```

```
from sklearn.model_selection import train_test_split

#Case 1 with test size 10%
X_train_case1, X_test_case1, Y_train_case1, Y_test_case1 = train_test_split(X, Y, test_size = 0.1, random_state = 0)

#Case 2 with test size 50%
X_train_case2, X_test_case2, Y_train_case2, Y_test_case2 = train_test_split(X, Y, test_size = 0.5, random_state = 0)
```

# Model Building

### Random Forest - Model building - Case 1

```
from sklearn.ensemble import RandomForestClassifier

reg_case1 = RandomForestClassifier(n_estimators=10, random_state=0)

# Case- 1
reg_case1.fit(X_train_case1,Y_train_case1)
```

```
RandomForestClassifier(n_estimators=10, random_state=0)
```

#### Predicting Random Forest | Case - 1

```
# Predicting
Y_pred_RandomForest_train_case1 = reg_case1.predict(X_train_case1)
Y_pred_RandomForest_test_case1 = reg_case1.predict(X_test_case1)
```

### Random Forest - Model building - Case 2

```
reg_case2 = RandomForestClassifier(n_estimators=10, random_state=0)

# Case- 2
reg_case2.fit(X_train_case2,Y_train_case2)
```

```
RandomForestClassifier(n_estimators=10, random_state=0)
```

#### Predicting Random Forest | Case-2

```
#Predicting
Y_pred_RandomForest_train_case2 = reg_case2.predict(X_train_case2)
Y_pred_RandomForest_test_case2 = reg_case2.predict(X_test_case2)
```

### KNN - Model building - Case 1

```
from sklearn.neighbors import KNeighborsClassifier

# Case-1
```

```
classifier_case1 = KNeighborsClassifier(n_neighbors=5, metric='euclidean', p=2)
classifier_case1.fit(X_train_case1, Y_train_case1)
```

```
KNeighborsClassifier(metric='euclidean')
```

#### Predicting KNN | Case - 1

```
Y_pred_KNN_train_case1 = classifier_case1.predict(X_train_case1)
Y_pred_KNN_test_case1 = classifier_case1.predict(X_test_case1)
```

### KNN - Model building - Case 2

```
# Case-2

classifier_case2 = KNeighborsClassifier(n_neighbors=5, metric='euclidean', p=2)
classifier_case2.fit(X_train_case2, Y_train_case2)
```

```
KNeighborsClassifier(metric='euclidean')
```

#### Predicting KNN | Case - 2

```
Y_pred_KNN_train_case2 = classifier_case2.predict(X_train_case2)
Y_pred_KNN_test_case2 = classifier_case2.predict(X_test_case2)
```

Performance Evaluation

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

def accuracy(y_pred, y_test):
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix : \n", cm, '\n')
    acc = accuracy_score(y_test, y_pred)*100
    print("Accuracy Score : {0:.2f}%".format(accuracy_score(y_test, y_pred)*100))
    print("Precision : {0:.4f}".format(precision_score(y_test, y_pred, average="weighted")))
    print("Recall : {0:.4f}".format(recall_score(y_test, y_pred, average="weighted")))
    print("F1 Score : {0:.4f}".format(f1_score(y_test, y_pred, average="weighted")))
    return acc
```

# Random Forest Performance

#### Prediction Performance Analysis with Case - 1 Test data

```
print("=============================================================\n   \
Random Forest | CASE - 1 | Prediction Evaluation metrics\n\
=============================================================\n")
print("+++++++++++++++++\nTest Set Accuracy: \n+++++++++++++++++")
acc_RandomForest_case1 = accuracy(Y_pred_RandomForest_test_case1, Y_test_case1)
```

```
=============================================================
   Random Forest | CASE - 1 | Prediction Evaluation metrics
=============================================================

+++++++++++++++++
```

```
Test Set Accuracy:
+++++++++++++++++
Confusion Matrix :
 [[4423   22  238]
 [ 114    1    9]
 [ 822   10   61]]

Accuracy Score : 78.68%
Precision : 0.7098
Recall : 0.7868
F1 Score : 0.7399
```

#### Prediction Performance Analysis with Case - 2 Test data

```
print("============================================================\n   \
Random Forest | CASE - 1 | Prediction Evaluation metrics\n\
============================================================\n")
print("+++++++++++++++++\nTest Set Accuracy: \n+++++++++++++++++")
acc_RandomForest_case2 = accuracy(Y_pred_RandomForest_test_case2, Y_test_case2)
```

```
============================================================
   Random Forest | CASE - 1 | Prediction Evaluation metrics
============================================================

+++++++++++++++++
Test Set Accuracy:
+++++++++++++++++
Confusion Matrix :
 [[21992   147  1243]
 [  600     6    45]
 [ 4214    32   221]]

Accuracy Score : 77.96%
Precision : 0.6968
Recall : 0.7796
F1 Score : 0.7309
```

# KNN Performance

#### Prediction Performance Analysis with Case - 1 Test data

```
print("============================================================\n   \
      KNN | CASE - 1 | Prediction Evaluation metrics\n\
============================================================\n")
print("+++++++++++++++++\nTest Set Accuracy: \n+++++++++++++++++")
acc_KNN_case1 = accuracy(Y_pred_KNN_test_case1, Y_test_case1)
```

```
============================================================
         KNN | CASE - 1 | Prediction Evaluation metrics
============================================================

+++++++++++++++++
Test Set Accuracy:
+++++++++++++++++
Confusion Matrix :
 [[4561    9  113]
 [ 120    0    4]
 [ 874    2   17]]

Accuracy Score : 80.32%
Precision : 0.6944
Recall : 0.8032
F1 Score : 0.7372
```

#### Prediction Performance Analysis with Case-2 Test data

```
print("==============================================================\n   \
      KNN | CASE - 2 | Prediction Evaluation metrics\n\
==============================================================\n")
print("++++++++++++++++++\nTest Set Accuracy: \n++++++++++++++++++")
acc_KNN_case2 = accuracy(Y_pred_KNN_test_case2, Y_test_case2)
```

```
==============================================================
         KNN | CASE - 2 | Prediction Evaluation metrics
==============================================================

++++++++++++++++++
Test Set Accuracy:
++++++++++++++++++
Confusion Matrix :
 [[22738    12   632]
 [  629     0    22]
 [ 4331     2   134]]

Accuracy Score : 80.25%
Precision : 0.7002
Recall : 0.8025
F1 Score : 0.7384
```

### Observations from Predictions Analysis:

1. **Model**

   KNN Model is slightly more accurate than Random Forest Model in diabetes classification.

2. **Test Case**

   Case-1 dataset is slightly more accurate than Case-2 dataset prediction.

```
warnings.filterwarnings('ignore')

!pip install tabulate

from tabulate import tabulate

data = [
    ["Random Forest Model", \
     "{0:.2f}%".format(accuracy_score(Y_train_case1, Y_pred_RandomForest_train_case1)*100), \
     "{0:.2f}%".format(accuracy_score(Y_test_case1, Y_pred_RandomForest_test_case1)*100), \
     "{0:.2f}%".format(accuracy_score(Y_train_case2, Y_pred_RandomForest_train_case2)*100), \
     "{0:.2f}%".format(accuracy_score(Y_test_case2, Y_pred_RandomForest_test_case2)*100) \
     ],
    ["KNN Model", \
     "{0:.2f}%".format(accuracy_score(Y_train_case1, Y_pred_KNN_train_case1)*100), \
     "{0:.2f}%".format(accuracy_score(Y_test_case1, Y_pred_KNN_test_case1)*100), \
     "{0:.2f}%".format(accuracy_score(Y_train_case2, Y_pred_KNN_train_case2)*100), \
     "{0:.2f}%".format(accuracy_score(Y_test_case2, Y_pred_KNN_test_case2)*100)
    ]
]

header = ["Name of the Model", \
          "Accuracy for Train Case-1", \
          "Accuracy for Test Case-1", \
          "Accuracy for Train Case-2", \
          "Accuracy for Test Case-2"]
```

```
Requirement already satisfied: tabulate in /opt/python/envs/default/lib/python3.8/site-packages (0.8.10)
WARNING: You are using pip version 21.3.1; however, version 22.2.2 is available.
You should consider upgrading via the '/opt/python/envs/default/bin/python -m pip install --upgrade pip' command.
```

```
print(tabulate(data, headers = header, tablefmt="rst"))
```

```
===================  ===========================  ==========================  ===========================  ==========================
Name of the Model    Accuracy for Train Case-1    Accuracy for Test Case-1    Accuracy for Train Case-2    Accuracy for Test Case-2
===================  ===========================  ==========================  ===========================  ==========================
Random Forest Model  90.91%                       78.68%                      92.42%                       77.96%
KNN Model            82.72%                       80.32%                      82.88%                       80.25%
===================  ===========================  ==========================  ===========================  ==========================
```

### Random Forest Model

1. Case 1 (Train = 90 % Test = 10%)

   Accuracy of Training set is 90.91% and Accuracy for Test set is 78.68%. We could say that the case-1 model is **overfit**.

2. Case 2 (Train = 50 % Test = 50%)

   Accuracy of Training set is 92.42% and Accuracy for Test set is 77.96%. We could say that the case-2 model is **overfit**.

### KNN Model

1. Case 1 (Train = 90 % Test = 10%)

   Accuracy of Training set is 82.72% and Accuracy for Test set is 80.32%. We could say that the case-1 model is **overfit**.

2. Case 2 (Train = 50 % Test = 50%)

   Accuracy of Training set is 82.88% and Accuracy for Test set is 80.25%. We could say that the case-2 model is **overfit**.