INTRODUCTIONTODATASCIENCE
CSE327

Diabetes Prediction Model

Report by
Vanshita Gupta (21ucc110)

Course Instructors
Dr Aloke Datta
Dr. Subrat Dash
Dr. Lal Upendra Pratap Singh

Department of Computer Science Engineering
The LNM Institute of Information Technology

LNMIIT
The LNM Institute of Information Technology

TABLE OF CONTENT

PROBLEM STATEMENT

We need to collect a dataset from the given website and perform the following steps:

1. Data pre-processing and it's visualisation.
2. Explain all the inferences we got from our data.
3. Explain what ML Classification Algorithms are being used and why.
4. Implementing those algorithms.
5. Output the result of the testing set and its visualisation.

All of the tasks are performed with the help of pre-existing Python Libraries such as:

- statsmodels
- matplotlib
- sklearn
- numpy
- mblearn
- pandas
- ydata_profiling

INTRODUCTION TO THE DATASET

The dataset having 330 features(columns) originates from research on diabetes disease, focusing on factors that impact diabetes and other chronic health conditions. The publisher has processed BRFSS data, transforming it into a format suitable for machine learning algorithms.

- ❖ Data Set Characteristics: Multivariate and Tabular
- ❖ Area: Health and Medicine
- ❖ Number of instances: 253680
- ❖ Number of Attributes: 21
- ❖ Attribute characteristics: Categorical, Integer
- ❖ Associated Tasks: Classification
- ❖ Missing Values: No

Attribute Description:

- **Diabetes_binary**: you have diabetes (0,1)
- **HighBP**: Adults who have been told they have high blood pressure by a doctor, nurse, or other health professional (0,1)
- **HighChol**: Have you EVER been told by a doctor, nurse or other health professional that your blood cholesterol is high? (0,1)
- **CholCheck**: Cholesterol check within the past five years (0,1)
- **BMI**: Body Mass Index (BMI)
- **Smoker**: Have you smoked at least 100 cigarettes in your entire life? [Note: 5 packs = 100 cigarettes] (0,1)
- **Stroke** : (Ever told) you had a stroke. (0,1)

- **HeartDiseaseorAttack**: Respondents that have ever reported having coronary heart disease (CHD) or myocardial infarction (MI) (0,1)
- **PhysActivity**: Adults who reported doing physical activity or exercise during the past 30 days other than their regular job (0,1)
- **Fruits**: Consume Fruit 1 or more times per day (0,1)
- **Veggies**: Consume Vegetables 1 or more times per day (0,1)
- **HvyAlcoholConsump**: Heavy drinkers (adult men having more than 14 drinks per week and adult women having more than 7 drinks per week)(0,1)
- **AnyHealthcare**: Do you have any kind of healthcare coverage, including health insurance, prepaid plans such as HMOs, or government plans such as Medicare, or Indian Health Service? (0,1)
- **NoDocbcCost**: Was there a time in the past 12 months when you needed to see a doctor but could not because of cost? (0,1)
- **GenHlth**: Would you say that in general, your health is: rate (1 ~ 5)
- **MentHlth**: Now thinking about your mental health, which includes stress, depression, and problems with emotions, for how many days during the past 30 days was your mental health not good? (0 ~ 30)
- **PhysHlth**: Now thinking about your physical health, which includes physical illness and injury, for how many days during the past 30 days was your physical health not good? (0 ~ 30)
- **DiffWalk**: Do you have serious difficulty walking or climbing stairs? (0,1)
- **Sex**: Indicate the sex of respondent (0,1) (Female or Male)
- **Age**: Fourteen-level age category (1 ~ 14)
- **Education**: What is the highest grade or year of school you completed? (1 ~ 6)
- **Income**: Is your annual household income from all sources: (If the respondent refuses at any income level, code "Refused.") (1 ~ 8)

Source of our dataset: https://archive.ics.uci.edu/dataset/891/cdc+diabetes+health+indicators

(All the above information about the dataset is taken from this link as well)

DATA EXPLORING

First, we import all the necessary libraries which will be required in our code.

```
In [2]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
        from ydata_profiling import ProfileReport
        import math

        from statsmodels.stats.outliers_influence import variance_inflation_factor
        from statsmodels.tools.tools import add_constant
        from statsmodels.stats.outliers_influence import variance_inflation_factor
        from statsmodels.tools.tools import add_constant
        from sklearn.feature_selection import SelectKBest
        from sklearn.feature_selection import chi2
        from sklearn.datasets import make_classification
        from sklearn.feature_selection import SelectKBest
        from sklearn.feature_selection import f_classif
        from sklearn.model_selection import train_test_split
        from imblearn.over_sampling import SMOTE

        from scikitplot.metrics import plot_roc_curve
        from sklearn.metrics import confusion_matrix, classification_report
        from sklearn.metrics import mean_absolute_error , mean_absolute_percentage_error , mean_squared_error , accuracy_score
        from mlxtend.plotting import plot_confusion_matrix
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.linear_model import LogisticRegression
        #%matplotlib inline

        import warnings
        warnings.filterwarnings("ignore")
```

Importing the dataset

```
In [3]: Orginal_data = pd.read_csv("diabetes_binary_health_indicators_BRFSS2015.csv" , sep = "," , encoding = 'utf-8')

In [4]: data = pd.read_csv("diabetes_binary_health_indicators_BRFSS2015.csv" , sep = "," , encoding = 'utf-8')
```

Viewing 5 elements of the dataset to understand

```
In [8]: print(data.head())
```

```
   Diabetes_binary  HighBP  HighChol  CholCheck   BMI  Smoker  Stroke  \
0              0.0     1.0       1.0        1.0  40.0     1.0     0.0
1              0.0     0.0       0.0        0.0  25.0     1.0     0.0
2              0.0     1.0       1.0        1.0  28.0     0.0     0.0
3              0.0     1.0       0.0        1.0  27.0     0.0     0.0
4              0.0     1.0       1.0        1.0  24.0     0.0     0.0

   HeartDiseaseorAttack  PhysActivity  Fruits  ...  AnyHealthcare  \
0                   0.0           0.0     0.0  ...            1.0
1                   0.0           1.0     0.0  ...            0.0
2                   0.0           0.0     1.0  ...            1.0
3                   0.0           1.0     1.0  ...            1.0
4                   0.0           1.0     1.0  ...            1.0

   NoDocbcCost  GenHlth  MentHlth  PhysHlth  DiffWalk  Sex   Age  Education  \
0          0.0      5.0      18.0      15.0       1.0  0.0   9.0        4.0
1          1.0      3.0       0.0       0.0       0.0  0.0   7.0        6.0
2          1.0      5.0      30.0      30.0       1.0  0.0   9.0        4.0
3          0.0      2.0       0.0       0.0       0.0  0.0  11.0        3.0
4          0.0      2.0       3.0       0.0       0.0  0.0  11.0        5.0

   Income
0     3.0
1     1.0
2     8.0
3     6.0
4     4.0

[5 rows x 22 columns]
```

Using the data.info( ), we find the information about our dataset, i.e., how many attributes are there, the datatype of each attribute, and whether is there any null value to it or not

```
In [7]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253680 entries, 0 to 253679
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Diabetes_binary       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
```

We also use data.describe( ), which is used to view basic statistical details about our dataset such as min-max values, standard deviation, mean, etc.

```
In [9]: data.describe()
```

Out[9]:

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	
count	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	2536
mean	0.139333	0.429001	0.424121	0.962670	28.382364	0.443169	0.040571	0.094186	0.756544	
std	0.346294	0.494934	0.494210	0.189571	6.608694	0.496761	0.197294	0.292087	0.429169	
min	0.000000	0.000000	0.000000	0.000000	12.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	1.000000	24.000000	0.000000	0.000000	0.000000	1.000000	
50%	0.000000	0.000000	0.000000	1.000000	27.000000	0.000000	0.000000	0.000000	1.000000	
75%	0.000000	1.000000	1.000000	1.000000	31.000000	1.000000	0.000000	0.000000	1.000000	
max	1.000000	1.000000	1.000000	1.000000	98.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 22 columns

Data Report (including a complete data report is not possible in this format)

```
In [5]: profile = ProfileReport(data)
        profile
```

Summarize dataset: 100% 67/67 [00:19<00:00, 4.33it/s, Completed]

Generate report structure: 100% 1/1 [00:04<00:00, 4.77s/it]

Render HTML: 100% 1/1 [00:01<00:00, 1.61s/it]

Pandas Profiling Report | Overview | Variables | Interactions | Correlations | Missing values | Sample | Duplicate rows

Overview

Overview | Alerts 9 | Reproduction

Dataset statistics

Number of variables	22
Number of observations	253680
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	11369
Duplicate rows (%)	4.5%
Total size in memory	42.6 MiB
Average record size in memory	176.0 B

Variable types

Categorical	16
Numeric	6

This help us to show the relation between features clearly during EDA

In [20]:
```
data2.Age[data2['Age'] == 1] = '18 to 24'
data2.Age[data2['Age'] == 2] = '25 to 29'
data2.Age[data2['Age'] == 3] = '30 to 34'
data2.Age[data2['Age'] == 4] = '35 to 39'
data2.Age[data2['Age'] == 5] = '40 to 44'
data2.Age[data2['Age'] == 6] = '45 to 49'
data2.Age[data2['Age'] == 7] = '50 to 54'
data2.Age[data2['Age'] == 8] = '55 to 59'
data2.Age[data2['Age'] == 9] = '60 to 64'
data2.Age[data2['Age'] == 10] = '65 to 69'
data2.Age[data2['Age'] == 11] = '70 to 74'
data2.Age[data2['Age'] == 12] = '75 to 79'
data2.Age[data2['Age'] == 13] = '80 or older'

data2.Diabetes_binary[data2['Diabetes_binary'] == 0] = 'No Diabetes'
data2.Diabetes_binary[data2['Diabetes_binary'] == 1] = 'Diabetes'

data2.HighBP[data2['HighBP'] == 0] = 'No High'
data2.HighBP[data2['HighBP'] == 1] = 'High BP'

data2.HighChol[data2['HighChol'] == 0] = 'No High Cholesterol'
data2.HighChol[data2['HighChol'] == 1] = 'High Cholesterol'

data2.CholCheck[data2['CholCheck'] == 0] = 'No Cholesterol Check in 5 Years'
data2.CholCheck[data2['CholCheck'] == 1] = 'Cholesterol Check in 5 Years'

data2.Smoker[data2['Smoker'] == 0] = 'No'
data2.Smoker[data2['Smoker'] == 1] = 'Yes'

data2.Stroke[data2['Stroke'] == 0] = 'No'
data2.Stroke[data2['Stroke'] == 1] = 'Yes'

data2.HeartDiseaseorAttack[data2['HeartDiseaseorAttack'] == 0] = 'No'
data2.HeartDiseaseorAttack[data2['HeartDiseaseorAttack'] == 1] = 'Yes'

data2.PhysActivity[data2['PhysActivity'] == 0] = 'No'
data2.PhysActivity[data2['PhysActivity'] == 1] = 'Yes'

data2.Fruits[data2['Fruits'] == 0] = 'No'
data2.Fruits[data2['Fruits'] == 1] = 'Yes'

data2.Veggies[data2['Veggies'] == 0] = 'No'
data2.Veggies[data2['Veggies'] == 1] = 'Yes'
```

INFERENCE

In our dataset, we can observe the following things:

- All of our data is numeric.
- Luckily, there aren't any missing values in our dataset.

So we can begin with our next part, EDA, to get inferences from the dataset.

EDA- Exploratory Data Analysis

Exploratory Data Analysis (EDA) in data science is a crucial phase that involves analyzing and visualizing data sets to understand their main characteristics, uncover patterns, and identify potential relationships between variables. EDA helps data scientists gain insights into the data, formulate hypotheses, and make informed decisions about subsequent analyses. Key components of EDA include

1. **Descriptive Statistics:**

   Central Tendency: Mean, median, mode.

   Dispersion: Range, variance, standard deviation.

2. **Data Visualization:**

   Histograms: Displaying the distribution of numerical data.

   Box Plots: Showing the summary of a set of data values.

   Scatter Plots: Examining relationships between two variables.

   Heatmaps: Displaying the correlation between variables.

3. **Data Cleaning:**

   Identifying and handling missing values.

   Addressing outliers that may affect the analysis.

4. **Distribution Analysis:**

   Checking if data follows a particular distribution (normal, skewed, etc.).

5. **Feature Engineering:**

   Creating new features based on existing ones to improve model performance.

6. **Correlation Analysis:**

   Examining relationships between variables to identify potential patterns.

7. **Dimensionality Reduction:**

   Reducing the number of features while retaining key information (e.g., PCA).

8. **Data Transformation:**

   Normalizing or scaling data to ensure uniformity.

9. **Pattern Recognition:**

   Identifying trends, seasonality, or other patterns in time-series data.

10. **Interactive Data Exploration:**

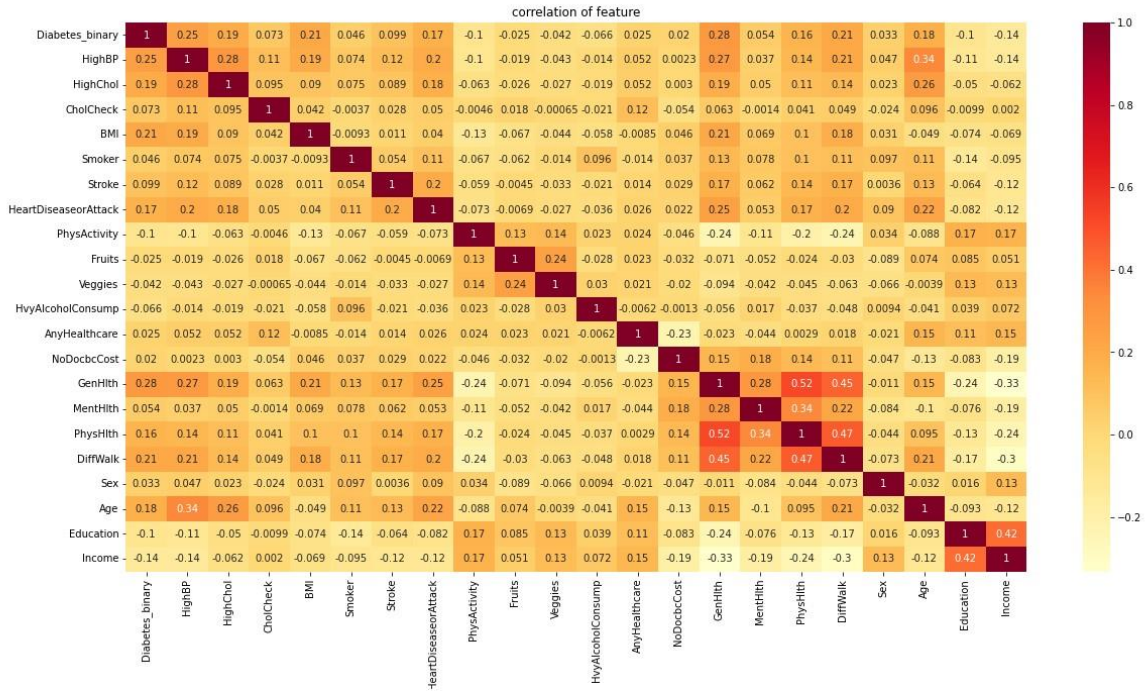
    Using tools like Tableau or Power BI for dynamic and interactive exploration.

The goal of EDA is to provide a foundation for more advanced analytics and modelling by revealing insights about the data's structure and characteristics. It is an iterative process, and findings from EDA may lead to further questions and refinements in the analysis approach.

#using heatmap to understand correlation better in dataset data
#Heatmap of correlation

```
In [21]: plt.figure(figsize = (20,10))
         sns.heatmap(data.corr(),annot=True , cmap ='YlOrRd' )
         plt.title("correlation of feature")
```

[25]: Text(0.5, 1.0, 'correlation of feature')



Correlation heatmap shows the relation between columns:

(GenHlth, PhysHlth),(PhysHlth, DiffWalk),(GenHlth, DiffWalk)are highly correlated with each other

=> POSITIVE

(GenHlth, Income), (DiffWalk, Income) are highly correlated with each other

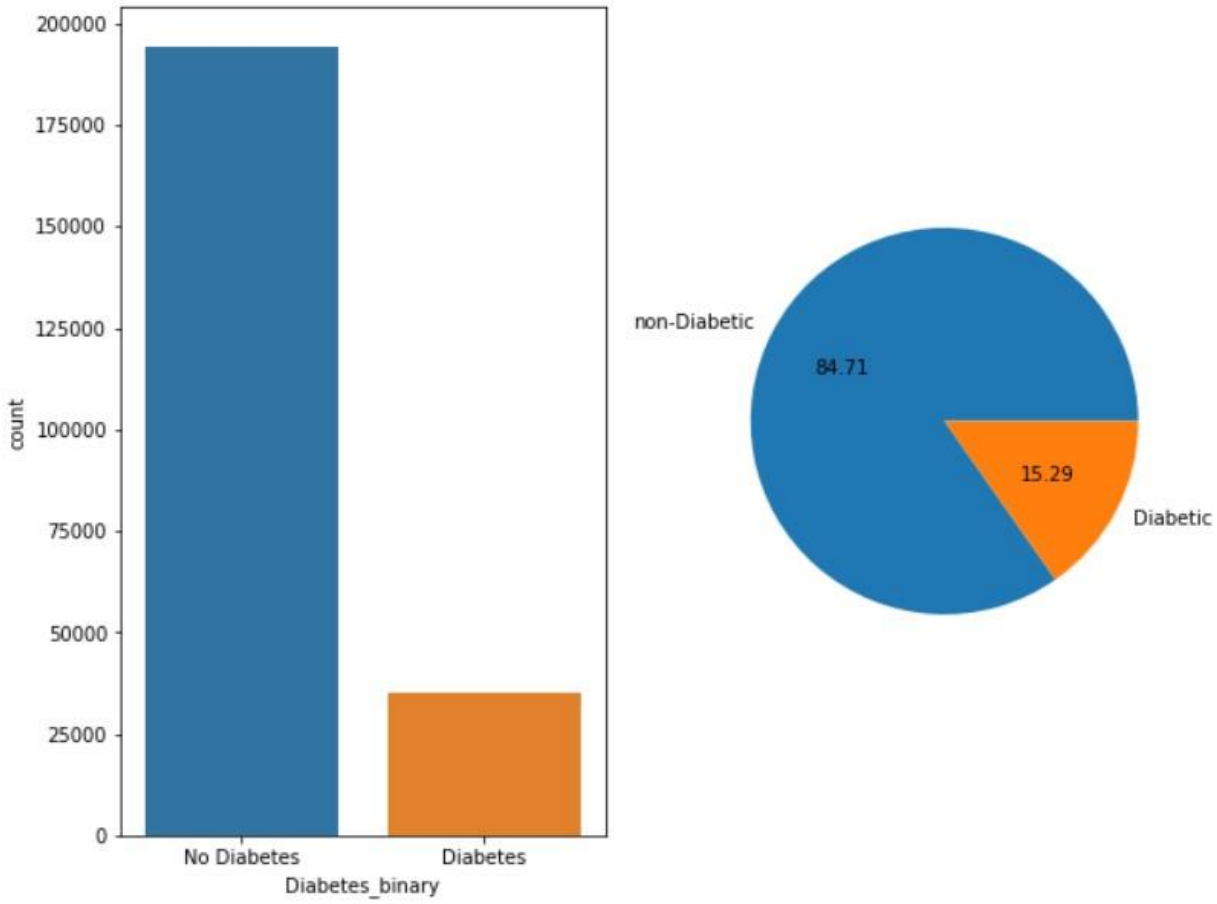
=> **NEGATIVE**

Let's view our target values "Daibetes_binary"

```
In [27]: data2["Diabetes_binary"].value_counts()

Out[27]: No Diabetes    194377
         Diabetes        35097
         Name: Diabetes_binary, dtype: int64
```

Checking Non diabetic and Diabetic people using pie charts and bar plot.

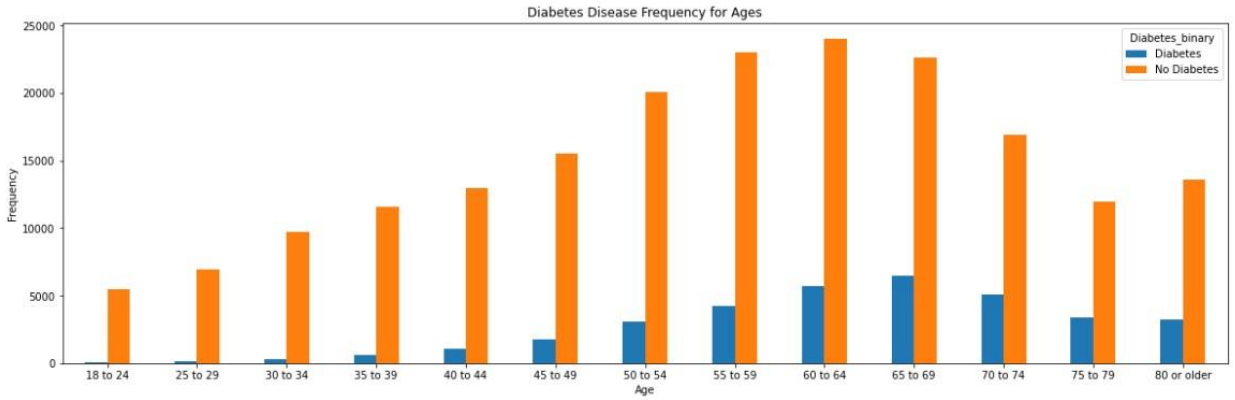
```
In [28]: figure1, plot1 = plt.subplots(1,2,figsize=(10,8))
         sns.countplot(data2['Diabetes_binary'],ax=plot1[0])

         #checking diabetic and non diabetic pepoles average by pie
         labels=["non-Diabetic","Diabetic"]
         plt.pie(data2["Diabetes_binary"].value_counts() , labels =labels ,autopct='%.02f' );
```



Therefore we infer that the number of healthy individuals exceeds those with diabetes.

- **The feature "Age" and its relation with the target:**

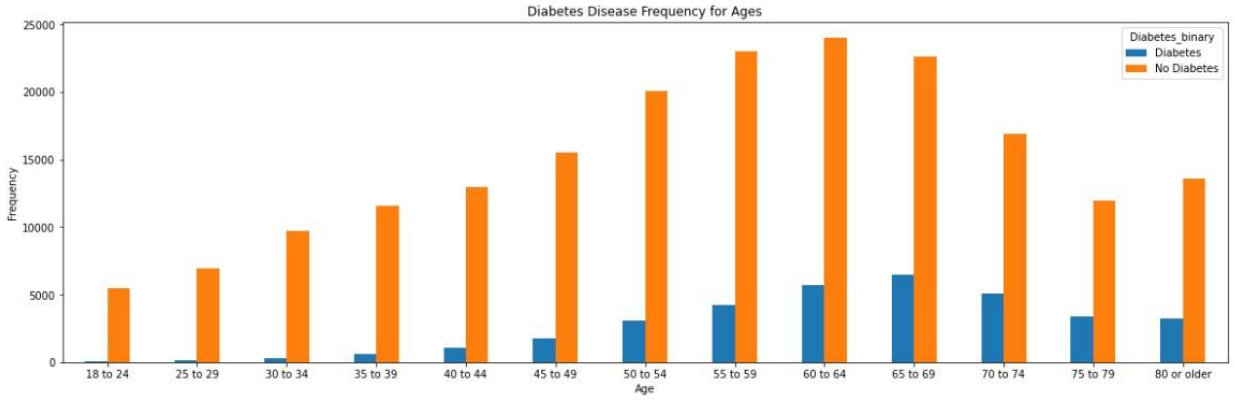
```
In [30]: pd.crosstab(data2.Age,data2.Diabetes_binary).plot(kind="bar",figsize=(20,6))
         plt.title('Diabetes Disease Frequency for Ages')
         plt.xlabel('Age')
         plt.xticks(rotation=0)
         plt.ylabel('Frequency')
         plt.show()
```



Observing the data, it becomes evident that with increasing age, the likelihood of diabetes also tends to rise. Consequently, we can infer that the median age of individuals with diabetes is higher than that of those without diabetes.

- The feature "Education" and its relation with the target:

```
In [31]: plt.figure(figsize=(10,6))


         sns.distplot(data.Education[data.Diabetes_binary == 0], color="y", label="No Diabetic" )
         sns.distplot(data.Education[data.Diabetes_binary == 1], color="m", label="Diabetic" )
         plt.title("Relation b/w Education and Diabetes")

         plt.legend()
```

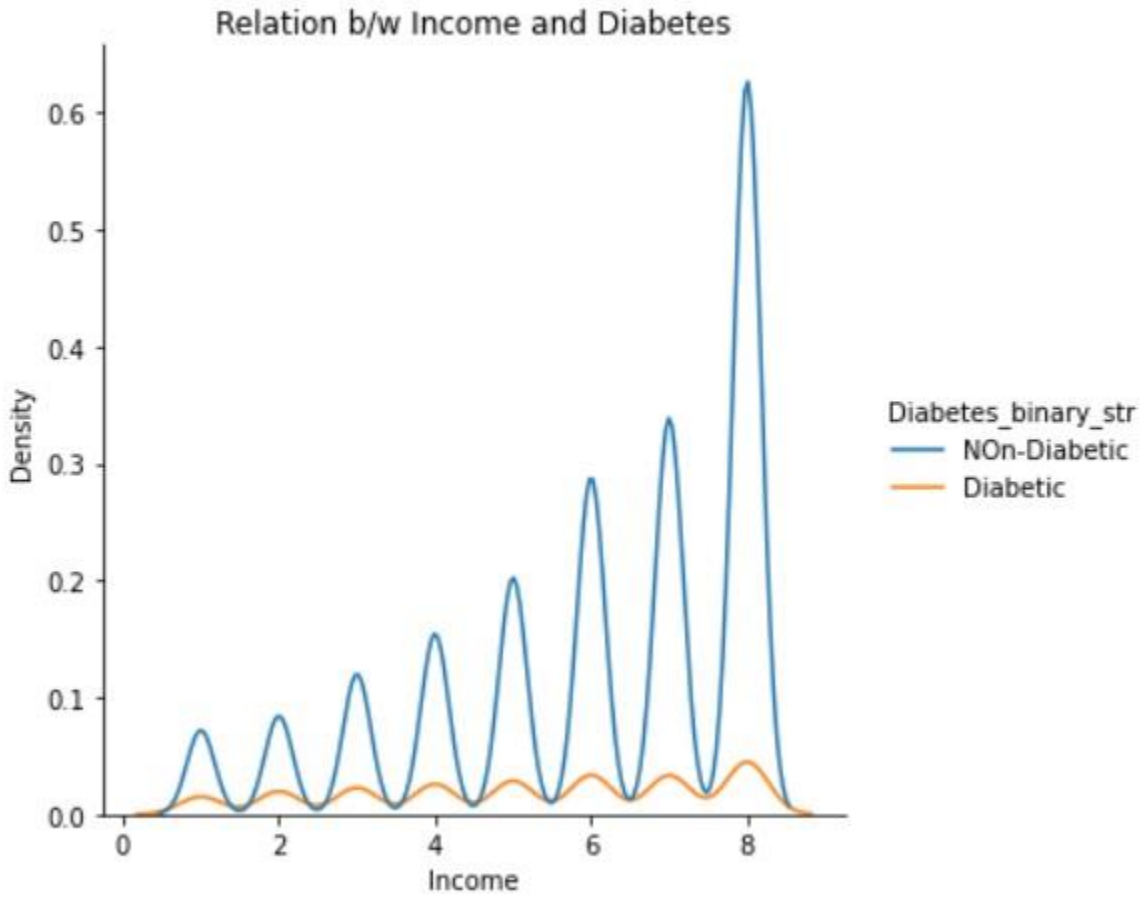


A significant portion of the population has attained a high level of education. Within this educated demographic, the prevalence of good health is higher compared to other education levels.

- **The feature "income" and its relation with the target:**

```
In [32]: plt.figure(figsize=(20,10))

         sns.displot(data=data, x="Income", hue="Diabetes_binary_str", kind="kde")
         plt.title("Relation b/w Income and Diabetes")
```

```
Text(0.5, 1.0, 'Relation b/w Income and Diabetes')
<Figure size 1440x720 with 0 Axes>
```



A majority of individuals boast high income levels. Within this affluent segment, the number of healthy individuals surpasses those in lower income brackets.

Upon Doing a similar comparison for other features:

1. It becomes apparent that the PhysHlth Group 0-5 significantly influences the occurrence of diabetes**.**
2. Similarly, the analysis indicates that the Menthlth Group 0-5 plays a role in influencing diabetes.
3. The impact of the PhysHlth Group 0-5 on diabetes is reiterated by the findings from the figures.
4. Notably, despite the lower prevalence of "5" and "4" in the GenHlth category, individuals with these ratings exhibit cases of diabetes**.**

LNMIIT
The LNM Institute of Information Technology

DATA PRE-PROCESSING

- After thoroughly analyzing the dataset through visuals(graph plots), we process our dataset a bit
- We convert the data to an integer.
- We check for any null values and unique values.
- This is used to check for any outliers in data.

Checking unique values in different variables

```
In [12]: data.isnull().sum()

Out[12]: Diabetes_binary         0
         HighBP                  0
         HighChol                0
         CholCheck               0
         BMI                     0
         Smoker                  0
         Stroke                  0
         HeartDiseaseorAttack    0
         PhysActivity            0
         Fruits                  0
         Veggies                 0
         HvyAlcoholConsump       0
         AnyHealthcare           0
         NoDocbcCost             0
         GenHlth                 0
         MentHlth                0
         PhysHlth                0
         DiffWalk                0
         Sex                     0
         Age                     0
         Education               0
         Income                  0
         dtype: int64
```

Checking Unique Values

In [58]:
```
unique_values = {}
for col in data.columns:
    unique_values[col] = data[col].value_counts().shape[0]

pd.DataFrame(unique_values, index=['unique value count']).transpose()
```

Out[58]:

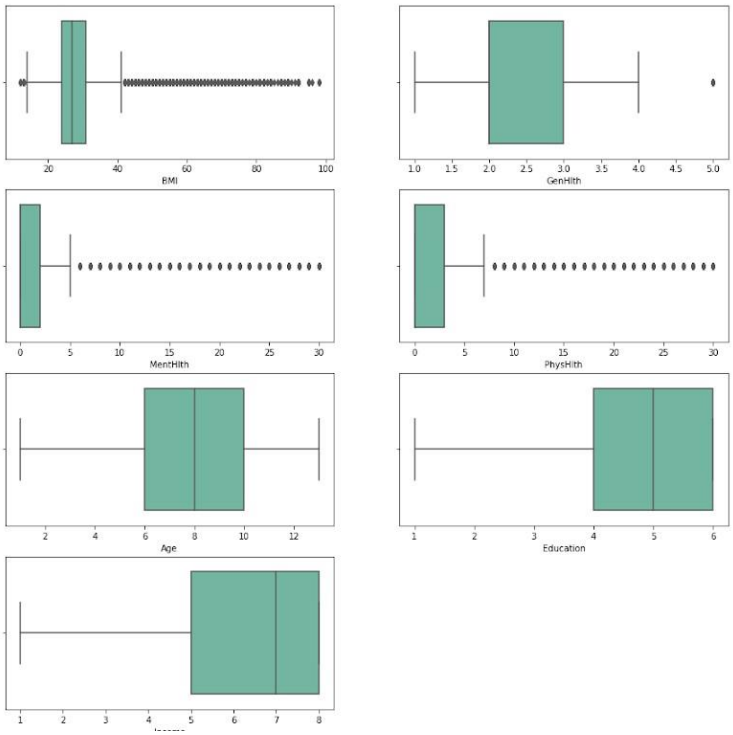
	unique value count
Diabetes_binary	2
HighBP	2
HighChol	2
CholCheck	2
BMI	84
Smoker	2
Stroke	2
HeartDiseaseorAttack	2
PhysActivity	2
Fruits	2
Veggies	2
HvyAlcoholConsump	2
AnyHealthcare	2
NoDocbcCost	2
GenHlth	5
MentHlth	31
PhysHlth	31
DiffWalk	2
Sex	2
Age	13
Education	6
Income	8

Transforming Data to Integer

LNMIIT
The LNM Institute of Information Technology

In [10]:
```
data["Diabetes_binary"] = data["Diabetes_binary"].astype(int)
data["HighBP"] = data["HighBP"].astype(int)
data["HighChol"] = data["HighChol"].astype(int)
data["CholCheck"] = data["CholCheck"].astype(int)
data["BMI"] = data["BMI"].astype(int)
data["Smoker"] = data["Smoker"].astype(int)
data["Stroke"] = data["Stroke"].astype(int)
data["HeartDiseaseorAttack"] = data["HeartDiseaseorAttack"].astype(int)
data["PhysActivity"] = data["PhysActivity"].astype(int)
data["Fruits"] = data["Fruits"].astype(int)
data["Veggies"] = data["Veggies"].astype(int)
data["HvyAlcoholConsump"] = data["HvyAlcoholConsump"].astype(int)
data["AnyHealthcare"] = data["AnyHealthcare"].astype(int)
data["NoDocbcCost"] = data["NoDocbcCost"].astype(int)
data["GenHlth"] = data["GenHlth"].astype(int)
data["MentHlth"] = data["MentHlth"].astype(int)
data["PhysHlth"] = data["PhysHlth"].astype(int)
data["DiffWalk"] = data["DiffWalk"].astype(int)
data["Sex"] = data["Sex"].astype(int)
data["Age"] = data["Age"].astype(int)
data["Education"] = data["Education"].astype(int)
data["Income"] =data["Income"].astype(int)
```

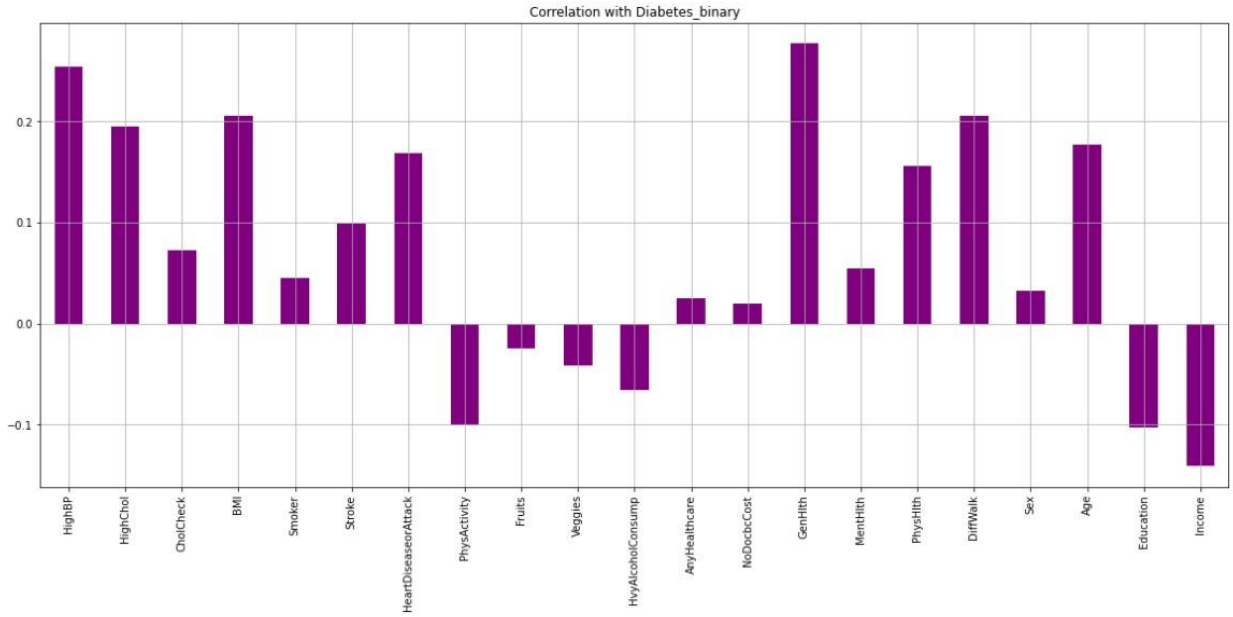
Checking for outliers



Feature selection

Feature selection is a crucial step in data science where relevant variables are chosen from a dataset to enhance model performance. Methods include a filter (correlation analysis, statistical tests), wrapper (RFE, forward selection, backward elimination), embedded (LASSO, tree-based methods), dimensionality reduction (PCA, t-SNE), and information gain/entropy techniques. The choice depends on the dataset and problem, aiming to balance dimensionality reduction with preserving valuable information for accurate modelling.

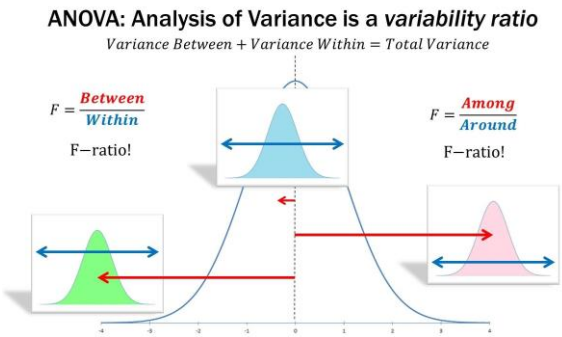
```
In [37]: data.drop('Diabetes_binary', axis=1).corrwith(data.Diabetes_binary).plot(kind='bar', grid=True, figsize=(20, 8)
         , title="Correlation with Diabetes_binary",color="Purple");
```



Diabetes_binary's relation with other columns Through bar Graph Result:

1. Fruits, AnyHealthcare, NoDocbccost and sex are least correlated with Diabetes_binary.
2. HighBP, high, BMI, smoker, stroke, HeartDiseaseorAttack, PhysActivity, Veggies, MentHlth, HvyAlcoholconsump, GenHlth, PhysHlth, Age, Education, Income and DiffWalk have a significant correlation with Diabetes_binary.

ANOVA TEST



Analysis of Variance (ANOVA) is a statistical method used to test differences between two or more means. It is similar to the t-test, but the t-test is generally used for comparing two means, while ANOVA is used when you have more than two means to compare.

ANOVA is based on comparing the variance (or variation) between the data samples to the variation within each particular sample. If the between-group variance is high and the within-group variance is low, this provides evidence that the means of the groups are significantly different.

ANOVA Terms:

- Factor: Independent variable in the analysis; one-way ANOVA has one factor, and two-way ANOVA has two.
- Levels: Different- groups or categories within a factor (e.g., 'low fat', 'medium fat', 'high fat').
- Response Variable- Dependent variable or measured outcome.
- Within-group Variance: Variance within each level of the factor.
- Between-group Variance: Variance between different levels of the factor.
- Grand Mean: Overall mean considering all data.
- Treatment Sums of Squares (SS): Between-group variability.
- Error Sums of Squares (SS): Within-group variability.
- Total Sums of Squares (SS): Sum of Treatment SS and Error SS; total variability.
- Degrees of Freedom (df): Number of values with freedom to vary when computing a statistic.
- Mean Square (MS): Average squared deviation; calculated by dividing the sum of squares by degrees of freedom.
- F-Ratio:Test statistic for ANOVAs; ratio of between-group variance to within-group variance.
- Null Hypothesis (H0): Hypothesis of no difference between group means.
- Alternative Hypothesis (H1): Hypothesis of a difference between at least two group means.
- P-value: Probability of obtaining a test statistic as extreme as observed, assuming the null hypothesis is true. If p-value < 0.05, the null hypothesis is rejected.
- Post-hoc tests: Follow-up tests after ANOVA to identify specific groups with different means. Examples: Tukey's HSD, Scheffe, Bonferroni.

ANOVA Formulas:

Sum of Squares Total (SST)

This represents the total variability in the data. It is the sum of the squared differences between each observation and the overall mean.

Formula:

SST = Σ(yi - y_mean)^2

Where:

- yi represents each individual data point
- y_mean represents the grand mean (mean of all observations)

Sum of Squares Within (SSW)

This represents the variability within each group or factor level. It is the sum of the squared differences between each observation and its group mean.

Formula:

SSW = Σ(yij - y_meani)^2

Where:

- yij represents each individual data point within a group
- y_meani represents the mean of the ith group

Sum of Squares Between (SSB)

This represents the variability between the groups. It is the sum of the squared differences between the group means and the grand mean, multiplied by the number of observations in each group.

Formula:

SSB = Σni(y_meani - y_mean)^2

Where:

- ni represents the number of observations in each group
- y_meani represents the mean of the ith group ● y_mean represents the grand mean

Degrees of Freedom

The degrees of freedom are the number of values that have the freedom to vary when calculating a statistic.

For within groups (dfW):

dfW = N - k

For between groups (dfB):

dfB = k - 1

For total (dfT):

dfT = N - 1

Where:

- N represents the total number of observations
- k represents the number of groups

Mean Squares

Mean squares are the sum of squares divided by the respective degrees of freedom.

Mean Squares Between (MSB):

MSB = SSB/dfB

Mean Squares Within (MSW):

MSW = SSW/dfW

F-Statistic

The F-statistic is used to test whether the variability between the groups is significantly greater than the variability within the groups.

Formula:

F = MSB / MSW

If the F-statistic is significantly higher than what would be expected by chance, we reject the null hypothesis that all group means are equal.

Implementing ANOVA test

In [39]:
```
X = Orginal_data.iloc[:,1:]
Y = Orginal_data.iloc[:,0]
```

In [40]:
```
# ANOVA feature selection for numeric input and categorical output


# define feature selection
fs = SelectKBest(score_func=f_classif, k=10)
# apply feature selection
X_selected = fs.fit_transform(X, Y)
print(X_selected.shape)
```

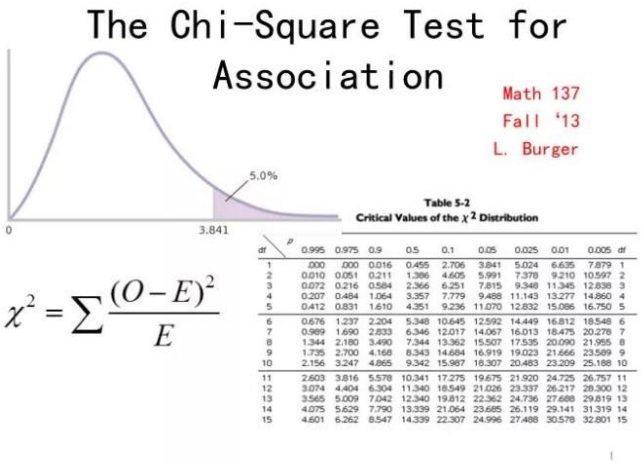
```
(253680, 10)
```

In [41]:
```
pd.DataFrame(X_selected).head(3)
```

Out[41]:

	0	1	2	3	4	5	6	7	8	9
0	1.0	1.0	40.0	0.0	5.0	15.0	1.0	9.0	4.0	3.0
1	0.0	0.0	25.0	0.0	3.0	0.0	0.0	7.0	6.0	1.0
2	1.0	1.0	28.0	0.0	5.0	30.0	1.0	9.0	4.0	8.0

CHI SQUARE TEST



df \ P	0.995	0.975	0.9	0.5	0.1	0.05	0.025	0.01	0.005	df
1	.000	.000	0.016	0.455	2.706	3.841	5.024	6.635	7.879	1
2	0.010	0.051	0.211	1.386	4.605	5.991	7.378	9.210	10.597	2
3	0.072	0.216	0.584	2.366	6.251	7.815	9.348	11.345	12.838	3
4	0.207	0.484	1.064	3.357	7.779	9.488	11.143	13.277	14.860	4
5	0.412	0.831	1.610	4.351	9.236	11.070	12.832	15.086	16.750	5
6	0.676	1.237	2.204	5.348	10.645	12.592	14.449	16.812	18.548	6
7	0.989	1.690	2.833	6.346	12.017	14.067	16.013	18.475	20.278	7
8	1.344	2.180	3.490	7.344	13.362	15.507	17.535	20.090	21.955	8
9	1.735	2.700	4.168	8.343	14.684	16.919	19.023	21.666	23.589	9
10	2.156	3.247	4.865	9.342	15.987	18.307	20.483	23.209	25.188	10
11	2.603	3.816	5.578	10.341	17.275	19.675	21.920	24.725	26.757	11
12	3.074	4.404	6.304	11.340	18.549	21.026	23.337	26.217	28.300	12
13	3.565	5.009	7.042	12.340	19.812	22.362	24.736	27.688	29.819	13
14	4.075	5.629	7.790	13.339	21.064	23.685	26.119	29.141	31.319	14
15	4.601	6.262	8.547	14.339	22.307	24.996	27.488	30.578	32.801	15

The chi-squared test is a hypothesis-testing method comparing observed and expected frequencies in experimental outcomes. In this test, a null hypothesis (H0) is assumed, and an alternative hypothesis (Ha) is defined. Sample data are used to determine whether to reject H0. The p-value, indicating the likelihood of sample results under the null hypothesis, is compared to α (commonly

0.05). If the p-value is less than α, H0 is rejected. The chi-squared value is then calculated as part of this testing process. **Formula:**

$$\chi^2 = \Sigma \frac{(O_i - E_i)^2}{E_i},$$

where χ^2 represents the chi-squared value, O_i represents the observed value, E_i represents the expected value (that is, the value expected from the null hypothesis), and the symbol Σ represents the summation of values for all i. One then looks up in a table the chi-squared value that corresponds to the chosen p-value and the number of degrees of freedom of the data (that is, the number of categories of the data minus one). If that value from the table is less than the chi-squared value calculated from the data, one can reject the null hypothesis.

Implementing Chi square test:

In [42]:
```
#apply SelectKBest class to extract top 10 best features
BestFeatures = SelectKBest(score_func=chi2, k=10)
fit = BestFeatures.fit(X,Y)

df_scores = pd.DataFrame(fit.scores_)
df_columns = pd.DataFrame(X.columns)

#concatenating two dataframes for better visualization
f_Scores = pd.concat([df_columns,df_scores],axis=1)              # feature scores
f_Scores.columns = ['Feature','Score']

f_Scores
```

Out[42]:

	Feature	Score
0	HighBP	10029.013935
1	HighChol	5859.710582
2	CholCheck	39.716825
3	BMI	18355.166400
4	Smoker	521.978858
5	Stroke	2725.225194
6	HeartDiseaseorAttack	7221.975378
7	PhysActivity	861.887532
8	Fruits	154.291404
9	Veggies	153.169215
10	HvyAlcoholConsump	779.424807
11	AnyHealthcare	3.280938
12	NoDocbcCost	229.542412
13	GenHlth	9938.507776
14	MentHlth	21029.632228
15	PhysHlth	133424.406534
16	DiffWalk	10059.506391
17	Sex	140.248274
18	Age	9276.141199
19	Education	756.035496
20	Income	4829.816361

Observing the above results, we can remove 4 features. Therefore "Fruits", "Veggies", "Sex", "CholCheck", and " AnyHealthcare" will not be with us because they contribute the least. Hence we will use these top 16 features.

Data Splitting

Data splitting into training and testing sets is a crucial step in the development and evaluation of machine learning models. The purpose is to assess how well a model generalizes to new, unseen data. Here's a detailed explanation of data splitting:

1. Objective:

- The primary goal is to train a machine learning model on a subset of the data (training set) and then evaluate its performance on another, distinct subset (testing set). This process helps estimate how well the model will perform on new, unseen data.

2. Dataset Division:

- The dataset is typically divided into two main subsets: the training set and the testing set.

3. Training Set:

- Purpose: The training set is used to train the machine learning model. The model learns the patterns, relationships, and features within this subset.
- Size: It often constitutes a larger portion of the dataset, usually around 70-80%, depending on the size and characteristics of the dataset.
- Randomization: The selection of instances for the training set should be random to avoid bias.

4. Testing Set:

- Purpose: The testing set is reserved to evaluate the model's performance. It simulates how well the model will generalize to new, unseen data.
- Size: It usually constitutes the remaining portion of the dataset (20-30%).
- Independence: The testing set should be independent of the training set to ensure that the model is not merely memorizing the training data but is capable of making predictions on new samples.

5. Randomization:

- Randomly shuffle the entire dataset before splitting to avoid any ordering effects that might exist in the raw data.

```
In [47]: Y.value_counts()

Out[47]: 0    194377
         1     35097
         Name: Diabetes_binary, dtype: int64

In [48]: from imblearn.under_sampling import NearMiss
         nm = NearMiss(version = 1 , n_neighbors = 10)

         x_sm,y_sm= nm.fit_resample(X,Y)

In [49]: y_sm.shape , x_sm.shape

Out[49]: ((70194,), (70194, 16))

In [50]: y_sm.value_counts()

Out[50]: 0    35097
         1    35097
         Name: Diabetes_binary, dtype: int64
```

6. Stratified Sampling:

- In situations where the dataset is imbalanced (e.g., one class significantly outnumbers another in classification problems), it is beneficial to use stratified sampling. This ensures that each class is proportionally represented in both the training and testing sets.

7. Implementation:

- Libraries like scikit-learn in Python provide functions (e.g., train_test_split) to easily split datasets into training and testing sets. The splitting can be based on a specified percentage or other criteria.

```
In [51]: X_train , X_test , Y_train , Y_test = train_test_split(x_sm,y_sm, test_size=0.3 , random_state=42)
```

8. Evaluation:

- Once the model is trained on the training set and tested on the testing set, performance metrics such as accuracy, precision, recall, and F1 score are calculated to assess the model's effectiveness.

LNMIIT
The LNM Institute of Information Technology

Data Scaling

Data scaling, also known as feature scaling or normalization, is a preprocessing step in machine learning that involves adjusting the scale of input features. The goal is to ensure that all features contribute equally to the learning process and that no single feature dominates due to its larger scale. When it comes to training and testing data, it's crucial to scale both sets consistently to maintain the integrity of the model. Here's a detailed explanation of data scaling in the context of training and testing sets:

1. Objective:

- The primary objective of data scaling is to bring all features to a similar scale, preventing certain features from dominating the learning process and improving the convergence of optimization algorithms.

2. Common Scaling Techniques:

    - Min-Max Scaling:
    - Scales the data to a specific range, often [0, 1], by subtracting the minimum value and dividing by the range (difference between maximum and minimum values).
    - Standardization (Z-score Normalization):
    - Transforms the data to have a mean of 0 and a standard deviation of 1. It involves subtracting the mean and dividing by the standard deviation. (Here we use this technique use fit_transform()).
    - Robust Scaling:
    - Similar to standardization but uses the median and interquartile range, making it more robust to outliers.
    - Normalization:
    - Adjusts the values in each row to have a unit norm, commonly used in scenarios where the magnitude of each instance is important.
-

3. Importance of Scaling:

- Scaling is critical for algorithms that rely on distance measures or gradient descent optimization. It ensures that the model treats all features equally, preventing bias toward features with larger scales.

4. Scaling in Training Data:

- Before training a machine learning model, the training data must be scaled to ensure that the model learns from features that are on a similar scale.

5. Scaling in Testing Data:

- The same scaling parameters (mean, standard deviation, min-max range, etc.) applied to the training data must be used to scale the testing data. This ensures consistency and prevents information leakage.

```
In [52]: from sklearn.preprocessing import StandardScaler
         scalar = StandardScaler()
         X_train = scalar.fit_transform(X_train)
         X_test = scalar.fit_transform(X_test)
```

ML Classifcation Algorithm

- After getting a cleaned dataset, we can now apply our prediction algorithms.
- In our project, instead of applying just one, we use 4 different classification algorithms to predict the results.
- The motivation to apply all these algorithms was that we wanted to compare their accuracy results to see which algorithm works better on our dataset.

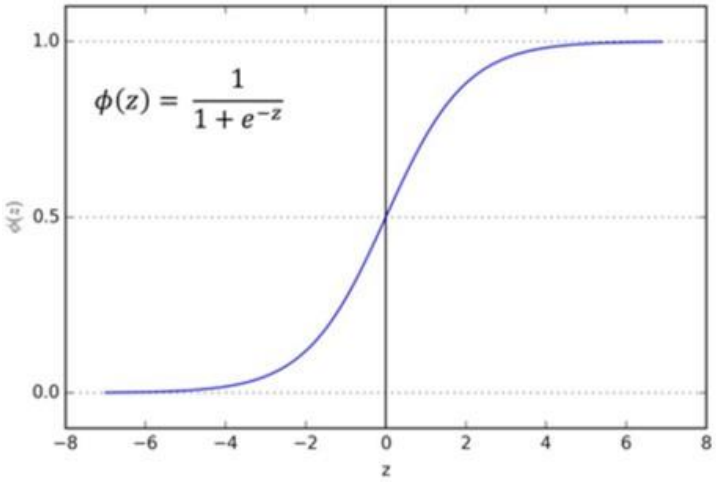
We applied the algorithms given below.

- Logistic regression
- Decision Tree Classifier
- KNN
- Support Vector Machine

- For each case, we've visualized the Confusion Matrix along with it.
- We've also displayed the accuracy percentage for each case too.

1) Logistic Regression

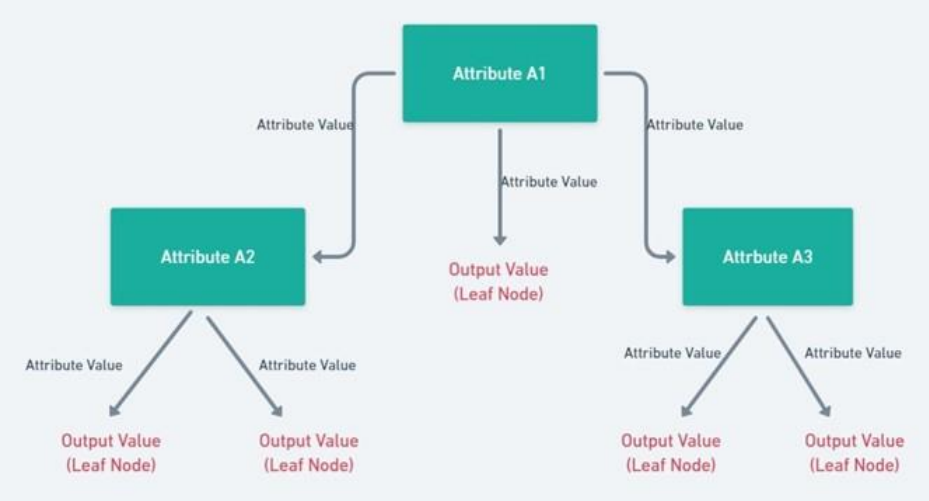
Logistic regression is a process of modeling the probability of a discrete outcome given an input variable. The most common logistic regression models a binary outcome; something that can take two values such as true/false, yes/no, and so on. For our problem which is a 2- Class, if the sigmoid function gives a value greater than threshold, then class 1 is selected, otherwise class 0.



This sigmoid function helps to scale the values between 0-1.

2) Decision Tree Classifier

A **decision tree** is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes. We've learnt that decision trees are made with the help of GINI Index, Entropy calculations etc which measure impurity, to try to determine what should be taken as the best split.



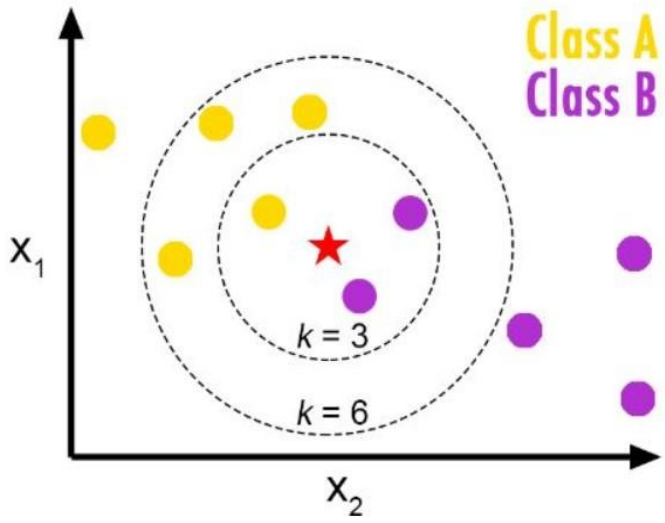
3) **KNN**

K-nearest neighbors is a non-parametric method used for classification and regression. It is one of the most easy ML technique used. It is a lazy learning model, with local approximation.

Basic Theory :

The basic logic behind KNN is to explore your neighborhood, assume the test datapoint to be similar to them and derive the output. In KNN, we look for k neighbors and come up with the prediction.

In case of KNN classification, a majority voting is applied over the k nearest datapoints whereas, in KNN regression, mean of k nearest datapoints is calculated as the output. As a rule of thumb, we selects odd numbers as k. KNN is a lazy learning model where the computations happens only runtime.



In the above diagram yellow and violet points corresponds to Class A and Class B in training data. The red star, points to the testdata which is to be classified. when k = 3, we predict Class B as the output and when K=6, we predict Class A as the output.

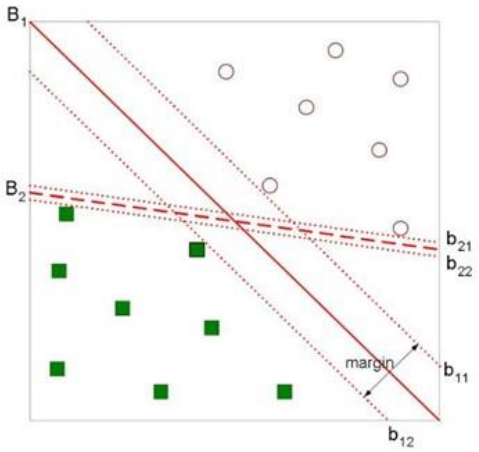
Loss function :

There is no training involved in KNN. During testing, k neighbors with minimum distance, will take part in classification /regression.

4) Support Vector Machine

Support Vector Machine uses a supervised learning algorithm. It is used to find a hyperplane that will separate the data classes. Now, there may be many hyperplanes which can separate the data, but SVM tries to find the best fit line for this separation.

This classifier generally works well when there is a clear line of separation between the classes, and the dataset is not large enough.



- Find hyperplane maximizes the margin => B1 is better than B2

IMPLEMENTATION

1) **Logistic Regression -**

In [54]:
```
lg = LogisticRegression(max_iter = 1500)
lg.fit(X_train , Y_train)
```

Out[54]:
```
LogisticRegression
LogisticRegression(max_iter=1500)
```

In [55]:
```
# make predictions on test set
y_pred=lg.predict(X_test)

print('Training set score: {:.4f}'.format(lg.score(X_train, Y_train)))

print('Test set score: {:.4f}'.format(lg.score(X_test, Y_test)))
```

```
Training set score: 0.8512
Test set score: 0.8472
```

In [56]:
```
#check MSE & RMSE
mse =mean_squared_error(Y_test, y_pred)
print('Mean Squared Error : '+ str(mse))
rmse = math.sqrt(mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error : '+ str(rmse))
```

```
Mean Squared Error : 0.152808775345458
Root Mean Squared Error : 0.3909076301960068
```

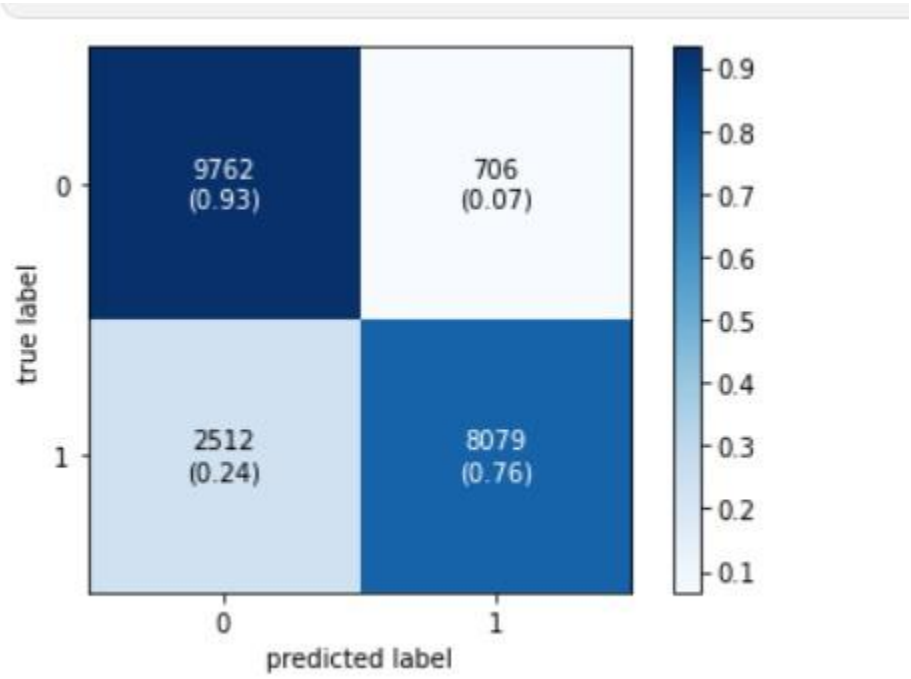
In [57]:
```
matrix = classification_report(Y_test,y_pred )
print(matrix)
```

```
              precision    recall  f1-score   support

           0       0.80      0.93      0.86     10468
           1       0.92      0.76      0.83     10591

    accuracy                           0.85     21059
   macro avg       0.86      0.85      0.85     21059
weighted avg       0.86      0.85      0.85     21059
```

From these results we can infer that Accuracy with Logistic Regression is 85%.

Confusion matrix for Logistic Regression.

```
In [58]: # calculating and plotting the confusion matrix
         cm1 = confusion_matrix(Y_test,y_pred)
         plot_confusion_matrix(conf_mat=cm1,show_absolute=True,
                               show_normed=True,
                               colorbar=True)
         plt.show()
```



2) **Decision Tree Classifier -**

```
In [59]: dt = DecisionTreeClassifier( max_depth= 12)
         dt.fit(X_train , Y_train)
```

Out[59]:

```
DecisionTreeClassifier(max_depth=12)
```

```
In [60]: # make predictions on test set
         y_pred=dt.predict(X_test)

         print('Training set score: {:.4f}'.format(dt.score(X_train, Y_train)))

         print('Test set score: {:.4f}'.format(dt.score(X_test, Y_test)))
```

```
Training set score: 0.8657
Test set score: 0.8479
```

```
In [61]: #check MSE & RMSE
         mse =mean_squared_error(Y_test, y_pred)
         print('Mean Squared Error : '+ str(mse))
         rmse = math.sqrt(mean_squared_error(Y_test, y_pred))
         print('Root Mean Squared Error : '+ str(rmse))
```

```
Mean Squared Error : 0.1520964908115295
Root Mean Squared Error : 0.3899955010144726
```

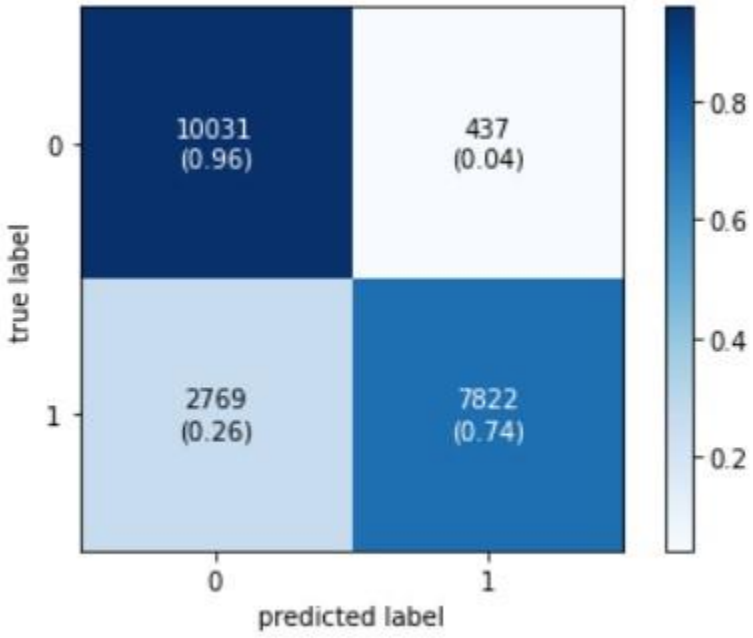
```
In [62]: matrix = classification_report(Y_test,y_pred )
         print(matrix)
```

```
              precision    recall  f1-score   support

           0       0.78      0.96      0.86     10468
           1       0.95      0.74      0.83     10591

    accuracy                           0.85     21059
   macro avg       0.87      0.85      0.85     21059
weighted avg       0.87      0.85      0.85     21059
```

From this we can clearly infer that accuracy with decision tree is 85% .

Confusion matrix for Decision tree -

```
In [63]: # calculating and plotting the confusion matrix
cm1 = confusion_matrix(Y_test,y_pred)
plot_confusion_matrix(conf_mat=cm1,show_absolute=True,
                                    show_normed=True,
                                    colorbar=True)
plt.show()
```



3) **KNN -**

In [64]:
```
knn = KNeighborsClassifier(n_neighbors= 6 )
knn.fit(X_train , Y_train)
```

Out[64]:
```
KNeighborsClassifier(n_neighbors=6)
```

In [65]:
```
# make predictions on test set
y_pred=knn.predict(X_test)

print('Training set score: {:.4f}'.format(knn.score(X_train, Y_train)))

print('Test set score: {:.4f}'.format(knn.score(X_test, Y_test)))
```

```
Training set score: 0.8423
Test set score: 0.8049
```

In [66]:
```
#check MSE & RMSE
mse =mean_squared_error(Y_test, y_pred)
print('Mean Squared Error : '+ str(mse))
rmse = math.sqrt(mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error : '+ str(rmse))
```

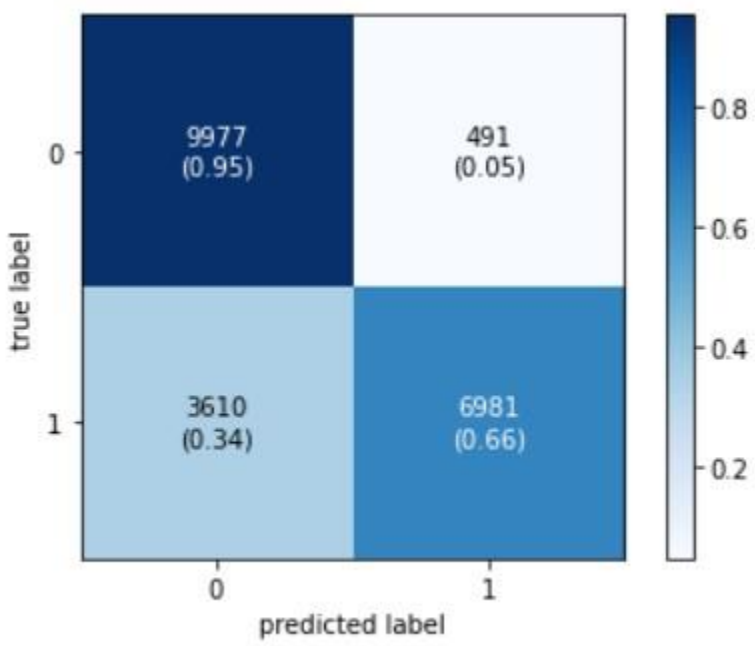
```
Mean Squared Error : 0.19507099102521488
Root Mean Squared Error : 0.44166841750935154
```

In [67]:
```
matrix = classification_report(Y_test,y_pred )
print(matrix)
```

```
              precision    recall  f1-score   support

           0       0.73      0.95      0.83     10468
           1       0.93      0.66      0.77     10591

    accuracy                           0.80     21059
   macro avg       0.83      0.81      0.80     21059
weighted avg       0.83      0.80      0.80     21059
```

From this we can infer that accuracy using KNN is 80% Confusion matrix for KNN-

In [68]:
```
# calculating and plotting the confusion matrix
cm1 = confusion_matrix(Y_test,y_pred)
plot_confusion_matrix(conf_mat=cm1,show_absolute=True,
                                  show_normed=True,
                                  colorbar=True)
plt.show()
```



4) **SVM -**

In [73]:
```
from sklearn.svm import SVC

# define the model
clf = SVC(kernel='rbf', C=1.0)

# train the model
clf.fit(X_train, Y_train)

y_pred=clf.predict(X_test)

print('Training set score: {:.4f}'.format(clf.score(X_train, Y_train)))

print('Test set score: {:.4f}'.format(clf.score(X_test, Y_test)))
```

```
Training set score: 0.8687
Test set score: 0.8603
```

In [74]:
```
#check MSE & RMSE
mse =mean_squared_error(Y_test, y_pred)
print('Mean Squared Error : '+ str(mse))
rmse = math.sqrt(mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error : '+ str(rmse))
```

```
Mean Squared Error : 0.13970273992117385
Root Mean Squared Error : 0.37376829710553816
```

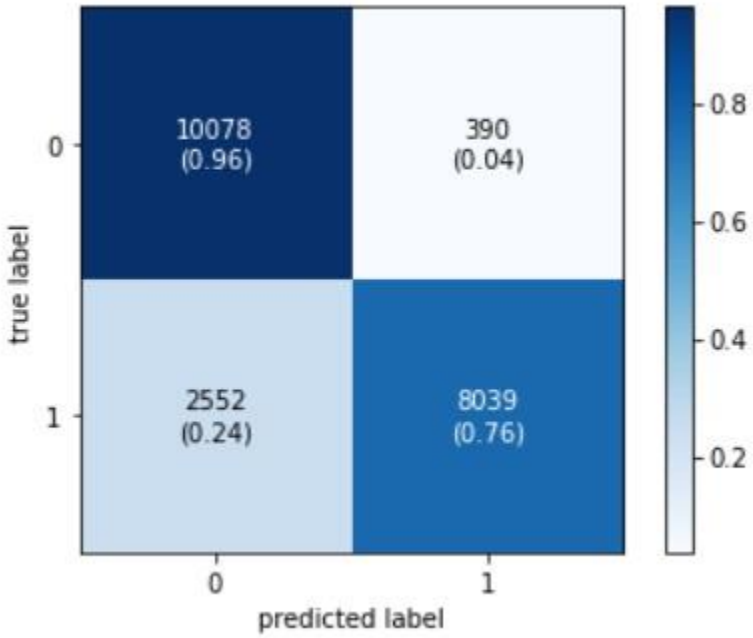
In [75]:
```
matrix = classification_report(Y_test,y_pred )
print(matrix)
```

```
              precision    recall  f1-score   support

           0       0.80      0.96      0.87     10468
           1       0.95      0.76      0.85     10591

    accuracy                           0.86     21059
   macro avg       0.88      0.86      0.86     21059
weighted avg       0.88      0.86      0.86     21059
```

From this we can infer that accuracy using SVM is 86 %

Confusion matrix for SVM -

In [76]:
```
# calculating and plotting the confusion matrix
cm1 = confusion_matrix(Y_test,y_pred)
plot_confusion_matrix(conf_mat=cm1,show_absolute=True,
                                  show_normed=True,
                                  colorbar=True)
plt.show()
```



Conclusion

To summarize our results from the implementations, we get the following

- From the results, it's clear that **SVM gives the highest accuracy of 86 % on our dataset.**
- SVMs have several advantages, such as the ability to handle high-dimensional data and the ability to perform well with small datasets. They also have the ability to model non-linear decision boundaries, which can be very useful in many applications.
- SVM works relatively well when there is a clear margin of separation between classes.

- SVM is more effective in high dimensional spaces and is relatively memory efficient.