

AI-Based Diabetes prediction system

BATCH MEMBER

510521106015:RUBA SHREE.J

Phase 3 Submission Document

PROJECT TITLE:Diabetes Prediction System

PHASE 3:Development Part 1

- TOPIC: *Start building the AI-Based Diabetes Prediction System by loading and preprocessing the dataset*



Diabetes Prediction System

Introduction:

- The development of a diabetes prediction system is a crucial step in leveraging technology to improve healthcare outcomes. With the increasing prevalence of diabetes worldwide, such a system holds immense promise in early detection and prevention. In this endeavor, we aim to harness the power of data analytics, machine learning, and medical expertise to create a robust predictive tool. This system will not only aid individuals in assessing their risk of diabetes but also assist healthcare providers in delivering personalized care and interventions. In this introduction, we will explore the significance of such a system, the underlying technology, and the potential benefits it can bring to individuals and the healthcare ecosystem.

Dataset:

Necessary Steps to follow:

1.Import Libraries

Start by importing the necessary libraries

Import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib import rcParams

from sklearn import model_selection

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score

from sklearn.metrics import f1_score, confusion_matrix, precision_recall_curve, roc_curve

from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.preprocessing import StandardScaler

import plotly.express as px

from plotly.subplots import make_subplots

import plotly.graph_objects as go

import warnings

warnings.filterwarnings(action='ignore')

2.Designing Utility Functions:

```
Def get_clf_eval(y_test, pred=None, pred_proba=None):  
    confusion = confusion_matrix(y_test, pred)  
    accuracy = accuracy_score(y_test, pred)  
    precision = precision_score(y_test, pred)  
    recall = recall_score(y_test, pred)  
    f1 = f1_score(y_test, pred)  
  
    roc_auc = roc_auc_score(y_test, pred_proba)  
  
    # ROC-AUC print  
    print('accuracy: {0:.4f}, precision: {1:.4f}, recall: {2:.4f},\  
    F1: {3:.4f}, AUC:{4:.4f}'.format(accuracy, precision, recall, f1, roc_auc))  
    return confusion
```

3. Reading and checking data

```
Diabetes_df = pd.read_csv("../input/pima-indians-diabetes-database/diabetes.csv")
```

[illegible]

Given Dataset:

	AGE	Diabetes Pedigree Function	Outcome
0	50.000000	0.627000	0.000000
1	31.000000	0.351000	1.000000
2	32.000000	0.672000	0.000000
3	21.000000	0.167000	1.000000

DiabetesPedigreeFunction has a long name. Change to DPF

```
diabetes_df.rename(columns  
={"DiabetesPedigreeFunction": "DPF"}, inplace=True)
```

4. Exploratory Data analysis (EDA):

Perform EDA to understand your data better. This include checking for missing values, exploring the data's statistics, and visualizing it to identify patterns.

PROGRAM:

INPUT:

```
import missingno as msno  
msno.matrix(diabetes_df)
```

OUTPUT:

<AxesSubplot:>

1

DPF

Age

Outcome

768.

5. Checking Target Imbalance:

```
Colors = ['gold', 'mediumturquoise']
```

```
labels = ['0','1']
```

```
values =
```

```
diabetes_df['Outcome'].value_counts()/diabetes_df['Outcome'].shape[0]
```

```
# Use `hole` to create a donut-like pie chart
```

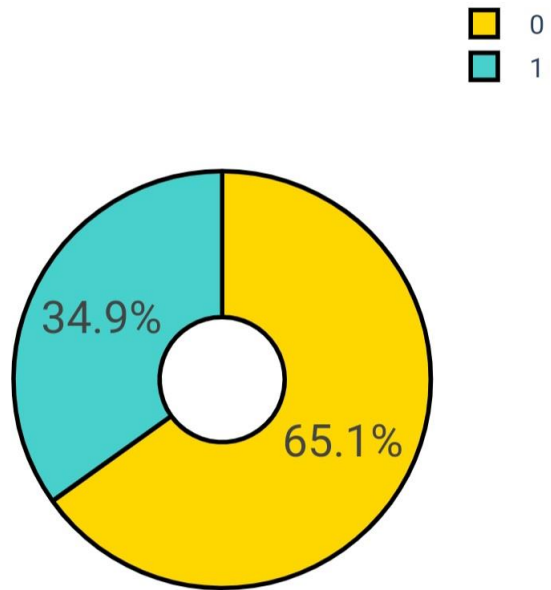
```
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.3)])
```

```
fig.update_traces(hoverinfo='label+percent', textinfo='percent',  
textfont_size=20,
```

```
                    marker=dict(colors=colors, line=dict(color='#000000', width=2)))
```

```
fig.update_layout(  
    title_text="Outcome")
```

OUTCOME:



6.Checking statistics:

```
Def highlight_min(s, props=""):
```

```
    return np.where(s == np.nanmin(s.values), props, "")
```

```
diabetes_df.describe().style.apply(highlight_min,  
props='color:Black;background-color:Grey', axis=0)
```

Count	768.000000
Mean	0.348958
Std	0.476951
Min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
Max	1.000000

7. Checking and removing outliers:

Input:

```
feature_names = [cname for cname in diabetes_df.loc[:,:'Age'].columns]
```

```
rcParams['figure.figsize'] = 40,60
```

```
sns.set(font_scale = 3)
```

```
sns.set_style("white")
```

```
sns.set_palette("bright")
```

```
plt.subplots_adjust(hspace=0.5)
```

```
i = 1;
```

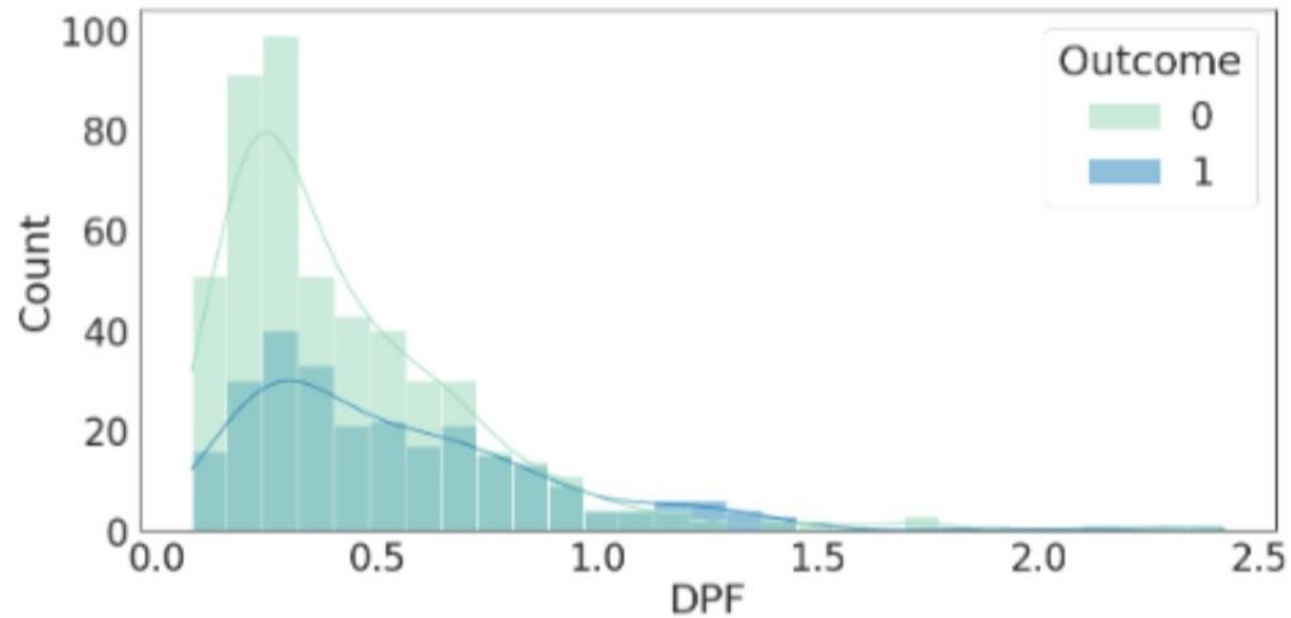
```
for name in feature_names:
```

```
    plt.subplot(5,2,i)
```

```
    sns.histplot(data=diabetes_df, x=name, hue="Outcome", kde=True, palette="YlGnBu")
```

```
    i = i + 1
```

Output:



PREPROCESSING:

Input:

#Transform the data to integer

```
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.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 int64
- 1 HighBP 253680 non-null int64
- 2 HighChol 253680 non-null int64
- 3 CholCheck 253680 non-null int64
- 4 BMI 253680 non-null int64
- 5 Smoker 253680 non-null int64
- 6 Stroke 253680 non-null int64
- 7 HeartDiseaseorAttack 253680 non-null int64
- 8 PhysActivity 253680 non-null int64
- 9 Fruits 253680 non-null int64
- 10 Veggies 253680 non-null int64
- 11 HvyAlcoholConsump 253680 non-null int64
- 12 AnyHealthcare 253680 non-null int64
- 13 NoDocbcCost 253680 non-null int64
- 14 GenHlth 253680 non-null int64
- 15 MentHlth 253680 non-null int64
- 16 PhysHlth 253680 non-null int64
- 17 DiffWalk 253680 non-null int64

Check null values:

Input:

```
data.isnull().sum()
```

Output:

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	

EDA:

Input:

#using heatmap to understand correlation better in dataset data

#Heatmap of correlation

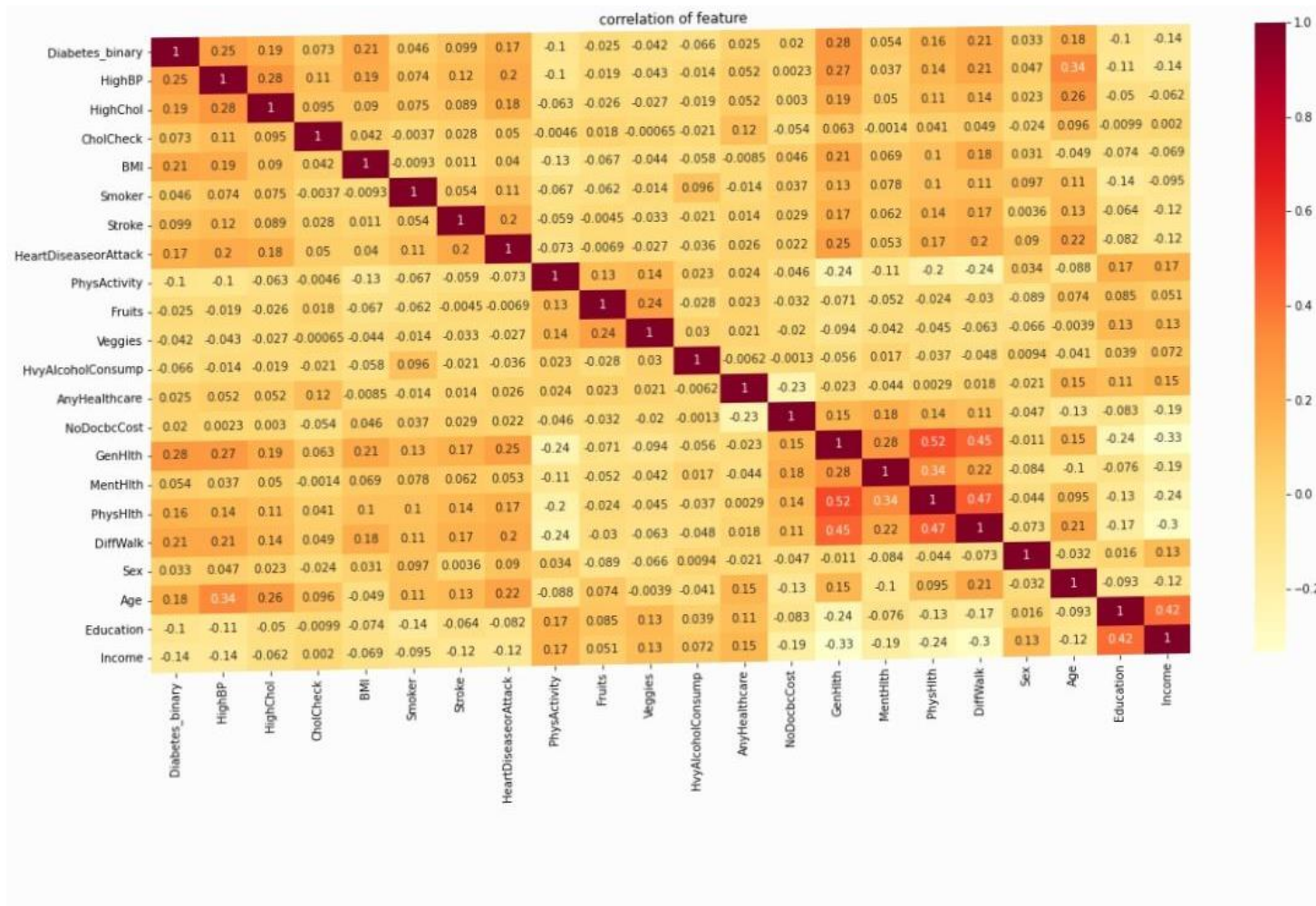
plt.figure(figsize = (20,10))

sns.heatmap(data.corr(),annot=True , cmap ='YlOrRd')

plt.title("correlation of feature")

Output:

text(0.5, 1.0, 'correlation of feature')



Correlation heatmap show relation between columns:

(GenHlth ,PhysHlth),(PhysHlth , DiffWalk),(GenHlth ,DiffWalk)are highly correleted with each other => positive relation

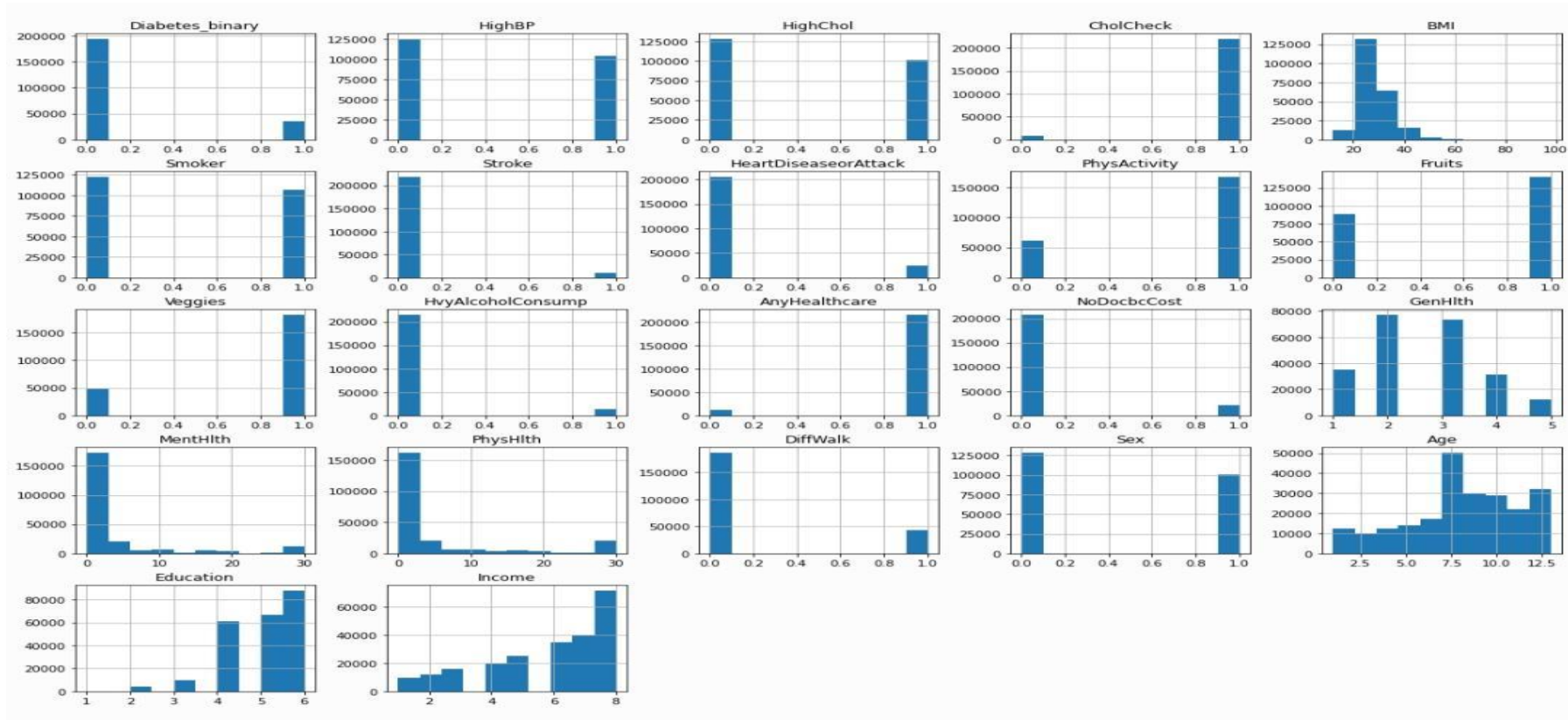
(GenHlth ,Income) , (DiffWalk , Income) are highly correleted with each other => Nagnative relation

Input:

#using histogram to understand dataset data better

```
data.hist(figsize=(20,15));
```

Output:

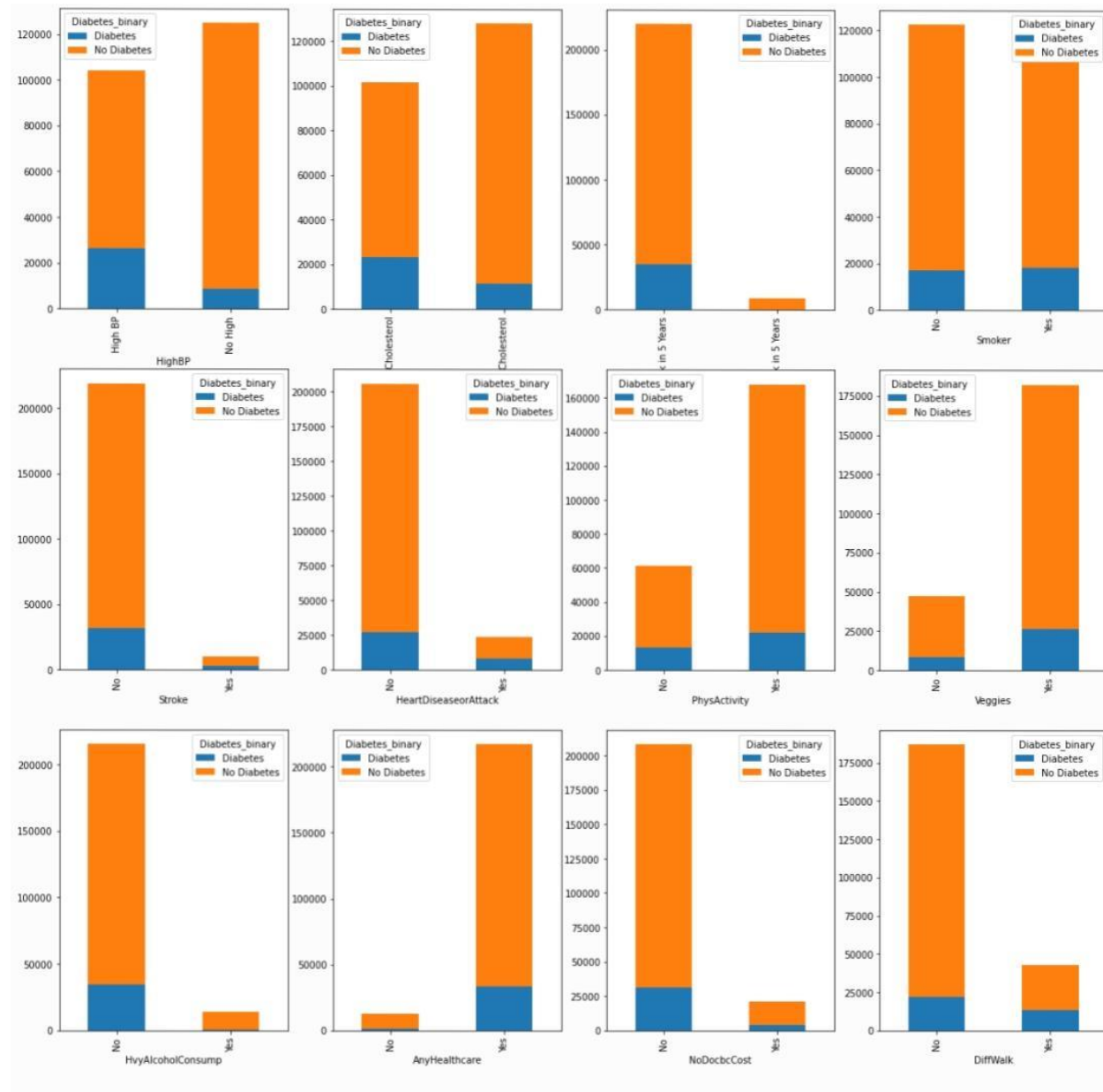


Visualization Of [Yes – NO] Columns and their relation with the target:

Input:

```
Cols = ['HighBP', 'HighChol', 'CholCheck','Smoker',  
        'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Veggies',  
        'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'DiffWalk']  
def create_plot_pivot(data2, x_column):  
    """ Create a pivot table for satisfaction versus another rating for easy plotting. """  
    _df_plot = data2.groupby([x_column, 'Diabetes_binary']).size() \  
    .reset_index().pivot(columns='Diabetes_binary', index=x_column, values=0)  
    return _df_plot  
fig, ax = plt.subplots(3, 4, figsize=(20,20))  
axe = ax.ravel()  
c = len(cols)  
for i in range(c):  
    create_plot_pivot(data2, cols[i]).plot(kind='bar',stacked=True, ax=axe[i])  
    axe[i].set_xlabel(cols[i])
```

Output:



Let's view our target values "Diabetes_binary"

Input:

#average of column Daibetes_binary

0 for non-Diabetic person and 1 for Diabetic person

data2["Diabetes_binary"].value_counts()

Output:

No Diabetes 194377

Diabetes 35097

Name: Diabetes_binary, dtype: int64

Input:

#checking the value count of Diabetes_binary_str by using countplot

figure1, plot1 = plt.subplots(1,2,figsize=(10,8))

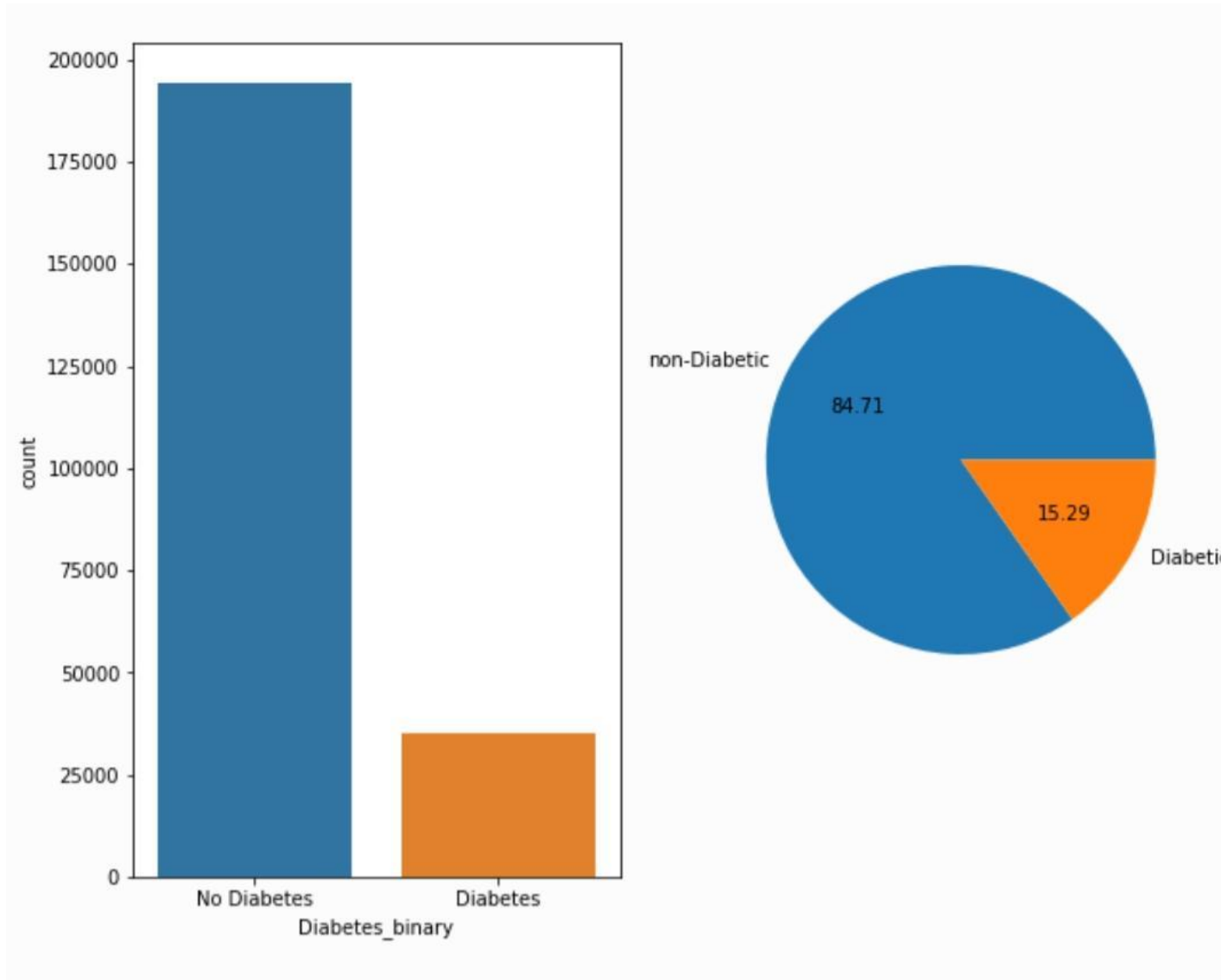
sns.countplot(data2['Diabetes_binary'],ax=plot1[0])

#checking diabetic and non diabetic pepoles average by pie

labels=["non-Diabetic","Diabetic"]

plt.pie(data2["Diabetes_binary"].value_counts(), labels =labels ,autopct='%.02f');

Output:



Conclusion:

- The loading and preprocessing phase plays a critical role in the development of an AI-based diabetes prediction system. It serves as the foundation for accurate and reliable model training and evaluation. Proper data loading ensures that relevant datasets are acquired, while effective preprocessing techniques, such as data cleaning, normalization, and feature engineering, help in enhancing the quality and relevance of the data. The success of the entire system depends on the careful execution of these steps, ultimately leading to a more robust and effective diabetes prediction model. It is essential to continuously refine and optimize the loading and preprocessing processes to keep the AI system up-to-date with the latest data and scientific advancements in the field of diabetes prediction.