Empowering Diabetes Management

Summary Progress Report

Project Overview

The Diabetes Monitoring System Capstone Project aims to develop an integrated system that assesses the severity of diabetes based on user-input parameters and generates electronic health records (EHRs) through a conversational interface. The project involves data collection, analysis, feature engineering, machine learning for severity prediction, conversational interface development, and EHR generation.

Key Achievements

Data Collection and Preprocessing:

- Collected from diverse data sources, including clinical records and simulated user data.
- Chose to go with ***diabetes_012_health_indicators_BRFSS2015*** dataset.
- In app user data collection (taking care of privacy and other issues) for analysis.

Data Analysis:

- Conducted Exploratory Data Analysis (EDA) to understand the dataset.
- Performed correlation analysis.

Next Steps

Diabetes Severity Prediction:

- Develop a classification model using various algorithms.
- Train the model to predict diabetes severity based on selected features.
- Evaluate the model's performance using accuracy and classification reports.

Conversational Interface:

- Implement a conversational interface using natural language processing (NLP) techniques.
- Design the interface to collect user-specific data related to diabetes.

Electronic Health Record (EHR) Generation:

- Structured and formatted user data and severity predictions into standardized EHRs.
- Generated EHRs based on the information gathered through the conversation with the user.

Conclusion

The Diabetes Monitoring System Capstone Project has made substantial progress in developing a comprehensive system for diabetes management. The combination of data analysis, machine learning, and conversational interfaces provides a promising framework for improving diabetes monitoring and patient care. Further development and refinement are needed to create a robust and user-friendly solution.

The project's technical aspects, code snippets, and methodologies have been documented in detail in the technical progress report. This summary report provides a high-level overview of the achievements and outlines the next steps for the project's advancement.

Technical Progress Report

Introduction

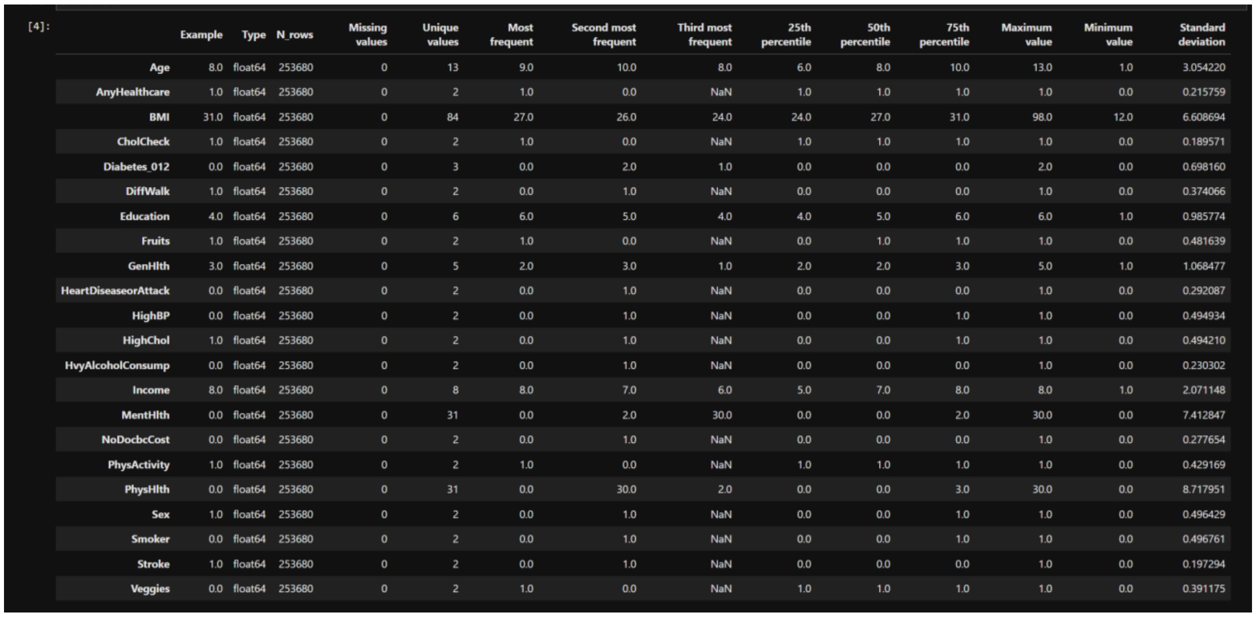
The goal of this capstone project is to build a diabetes monitoring system that assesses the status of diabetes based on user-input parameters and subsequently generates an electronic health record (EHR) through a conversational interface.

This part of the project involves data collection, analysis, feature engineering, and the development of algorithms to determine diabetes status.

Data collection and EDA

We used the "***diabetes_012_health_indicators_BRFSS2015.csv"*** dataset available in Kaggle (link), to train the classifier model. This is a clean dataset of 253,680 survey responses to the CDC's BRFSS2015. The target variable Diabetes_012 has 3 classes. 0 is for no diabetes or only during pregnancy, 1 is for prediabetes, and 2 is for diabetes.

Exploratory data analysis:



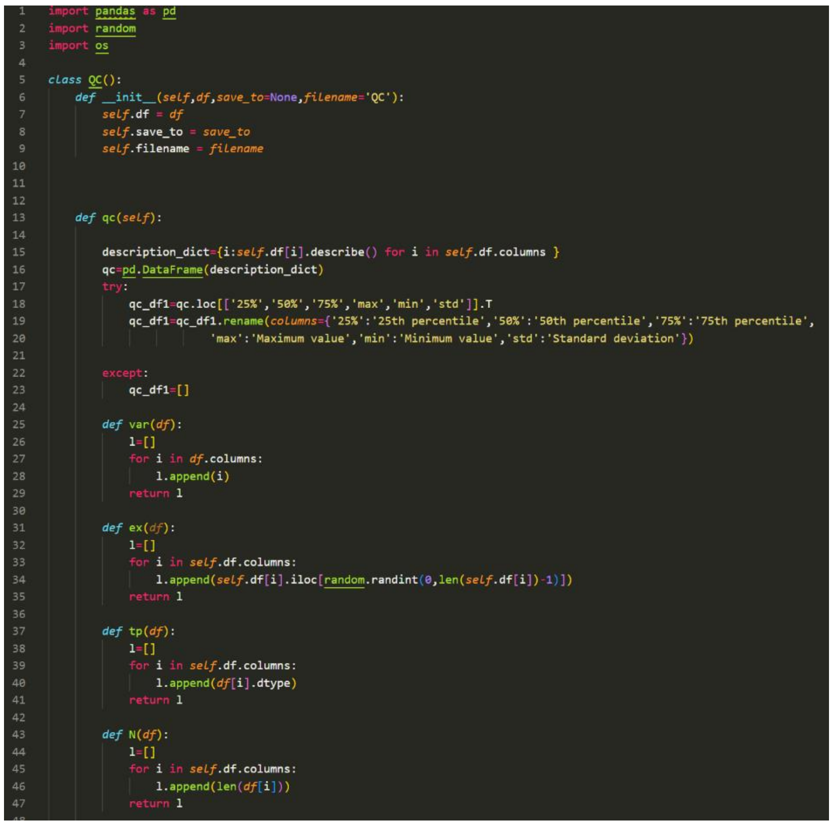
[4]:

	Example	Type	N_rows	Missing values	Unique values	Most frequent	Second most frequent	Third most frequent	25th percentile	50th percentile	75th percentile	Maximum value	Minimum value	Standard deviation
Age	8.0	float64	253680	0	13	9.0	10.0	8.0	6.0	8.0	10.0	13.0	1.0	3.054220
AnyHealthcare	1.0	float64	253680	0	2	1.0	0.0	NaN	1.0	1.0	1.0	1.0	0.0	0.215759
BMI	31.0	float64	253680	0	84	27.0	26.0	24.0	24.0	27.0	31.0	98.0	12.0	6.608694
CholCheck	1.0	float64	253680	0	2	1.0	0.0	NaN	1.0	1.0	1.0	1.0	0.0	0.189571
Diabetes_012	0.0	float64	253680	0	3	0.0	2.0	1.0	0.0	0.0	0.0	2.0	0.0	0.698160
DiffWalk	1.0	float64	253680	0	2	0.0	1.0	NaN	0.0	0.0	0.0	1.0	0.0	0.374066
Education	4.0	float64	253680	0	6	6.0	5.0	4.0	4.0	5.0	6.0	6.0	1.0	0.985774
Fruits	1.0	float64	253680	0	2	1.0	0.0	NaN	0.0	1.0	1.0	1.0	0.0	0.481639
GenHlth	3.0	float64	253680	0	5	2.0	3.0	1.0	2.0	2.0	3.0	5.0	1.0	1.068477
HeartDiseaseorAttack	0.0	float64	253680	0	2	0.0	1.0	NaN	0.0	0.0	0.0	1.0	0.0	0.292087
HighBP	0.0	float64	253680	0	2	0.0	1.0	NaN	0.0	0.0	1.0	1.0	0.0	0.494934
HighChol	1.0	float64	253680	0	2	0.0	1.0	NaN	0.0	0.0	1.0	1.0	0.0	0.494210
HvyAlcoholConsump	0.0	float64	253680	0	2	0.0	1.0	NaN	0.0	0.0	0.0	1.0	0.0	0.230302
Income	8.0	float64	253680	0	8	8.0	7.0	6.0	5.0	7.0	8.0	8.0	1.0	2.071148
MentHlth	0.0	float64	253680	0	31	0.0	2.0	30.0	0.0	0.0	2.0	30.0	0.0	7.412847
NoDocbcCost	0.0	float64	253680	0	2	0.0	1.0	NaN	0.0	0.0	0.0	1.0	0.0	0.277654
PhysActivity	1.0	float64	253680	0	2	1.0	0.0	NaN	1.0	1.0	1.0	1.0	0.0	0.429169
PhysHlth	0.0	float64	253680	0	31	0.0	30.0	2.0	0.0	0.0	3.0	30.0	0.0	8.717951
Sex	1.0	float64	253680	0	2	0.0	1.0	NaN	0.0	0.0	1.0	1.0	0.0	0.496429
Smoker	0.0	float64	253680	0	2	0.0	1.0	NaN	0.0	0.0	1.0	1.0	0.0	0.496761
Stroke	1.0	float64	253680	0	2	0.0	1.0	NaN	0.0	0.0	0.0	1.0	0.0	0.197294
Veggies	0.0	float64	253680	0	2	1.0	0.0	NaN	1.0	1.0	1.0	1.0	0.0	0.391175

Figure 1: EDA result

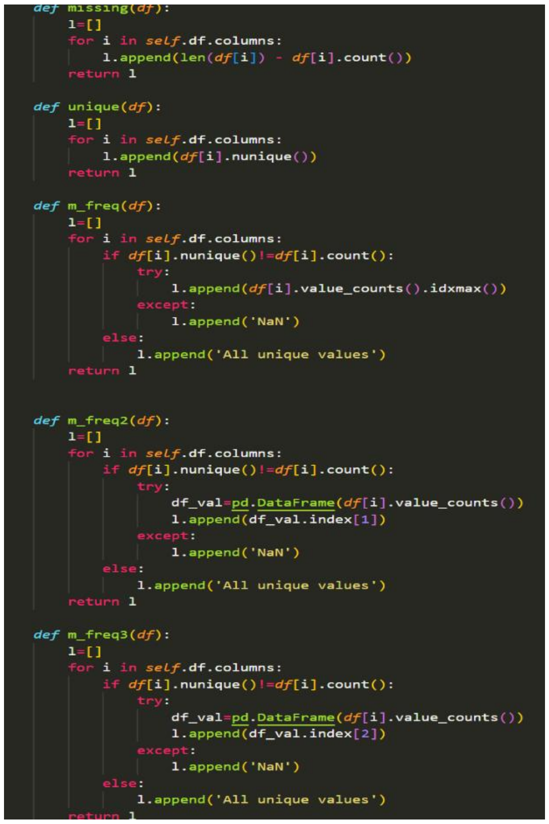
This dataset has 21 features that are used to determine the diabetes status of a person. From the above analysis, there are 253680 observations and none of the features has any missing value. This analysis also provides insight on number of unique values, maximum value, most frequent value and so on.

We built a custom package to perform a preliminary statistical analysis of the raw data. Following images show the code:



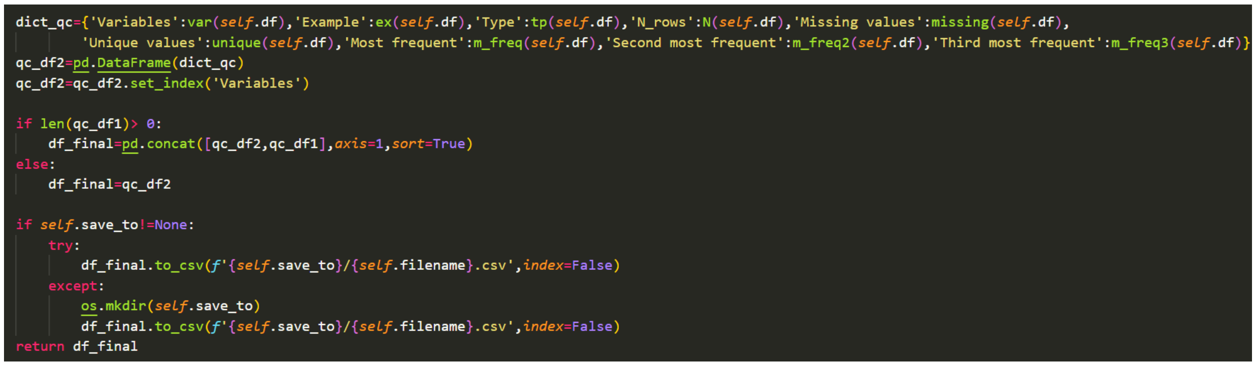
```
import pandas as pd
import random
import os

class QC():
    def __init__(self,df,save_to=None,filename='QC'):
        self.df = df
        self.save_to = save_to
        self.filename = filename



    def qc(self):

        description_dict={i:self.df[i].describe() for i in self.df.columns }
        qc=pd.DataFrame(description_dict)
        try:
            qc_df1=qc.loc[['25%','50%','75%','max','min','std']].T
            qc_df1=qc_df1.rename(columns={'25%':'25th percentile','50%':'50th percentile','75%':'75th percentile',
                        'max':'Maximum value','min':'Minimum value','std':'Standard deviation'})

        except:
            qc_df1=[]

        def var(df):
            l=[]
            for i in df.columns:
                l.append(i)
            return l

        def ex(df):
            l=[]
            for i in self.df.columns:
                l.append(self.df[i].iloc[random.randint(0,len(self.df[i])-1)])
            return l

        def tp(df):
            l=[]
            for i in self.df.columns:
                l.append(df[i].dtype)
            return l

        def N(df):
            l=[]
            for i in self.df.columns:
                l.append(len(df[i]))
            return l
```

(a)



```
def missing(df):
    l=[]
    for i in self.df.columns:
        l.append(len(df[i]) - df[i].count())
    return l

def unique(df):
    l=[]
    for i in self.df.columns:
        l.append(df[i].nunique())
    return l

def m_freq(df):
    l=[]
    for i in self.df.columns:
        if df[i].nunique()!=df[i].count():
            try:
                l.append(df[i].value_counts().idxmax())
            except:
                l.append('NaN')
        else:
            l.append('All unique values')
    return l


def m_freq2(df):
    l=[]
    for i in self.df.columns:
        if df[i].nunique()!=df[i].count():
            try:
                df_val=pd.DataFrame(df[i].value_counts())
                l.append(df_val.index[1])
            except:
                l.append('NaN')
        else:
            l.append('All unique values')
    return l

def m_freq3(df):
    l=[]
    for i in self.df.columns:
        if df[i].nunique()!=df[i].count():
            try:
                df_val=pd.DataFrame(df[i].value_counts())
                l.append(df_val.index[2])
            except:
                l.append('NaN')
        else:
            l.append('All unique values')
    return l
```

(b)

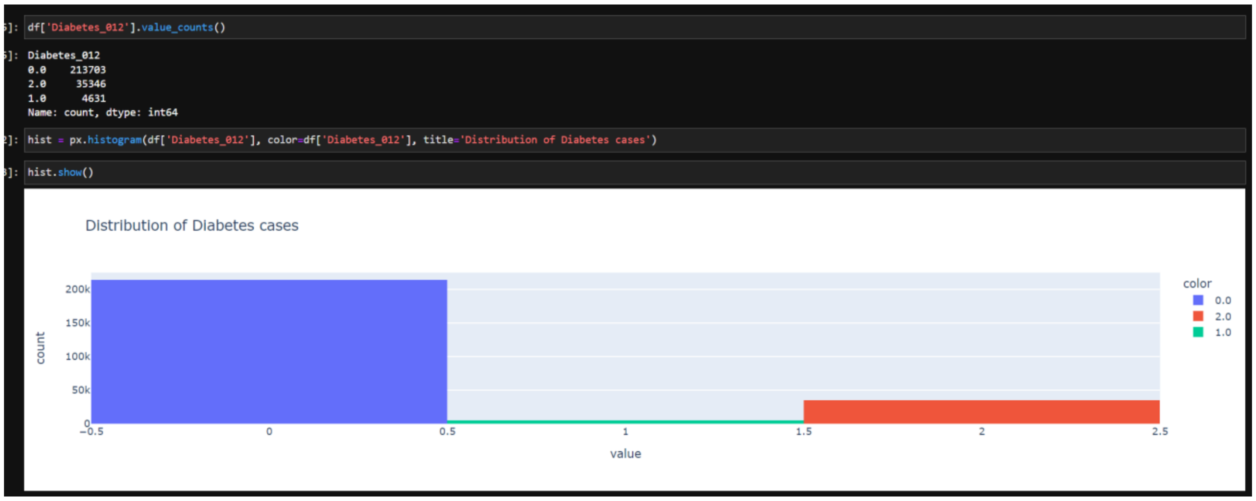


```
dict_qc={'Variables':var(self.df),'Example':ex(self.df),'Type':tp(self.df),'N_rows':N(self.df),'Missing values':missing(self.df),
        'Unique values':unique(self.df),'Most frequent':m_freq(self.df),'Second most frequent':m_freq2(self.df),'Third most frequent':m_freq3(self.df)}
qc_df2=pd.DataFrame(dict_qc)
qc_df2=qc_df2.set_index('Variables')

if len(qc_df1)> 0:
    df_final=pd.concat([qc_df2,qc_df1],axis=1,sort=True)
else:
    df_final=qc_df2

if self.save_to!=None:
    try:
        df_final.to_csv(f'{self.save_to}/{self.filename}.csv',index=False)
    except:
        os.mkdir(self.save_to)
        df_final.to_csv(f'{self.save_to}/{self.filename}.csv',index=False)
return df_final
```

(c)

Figure 2: (a), (b), (c) Custom class for analysis

Next step is to perform visualization. We used plotly library for this purpose.

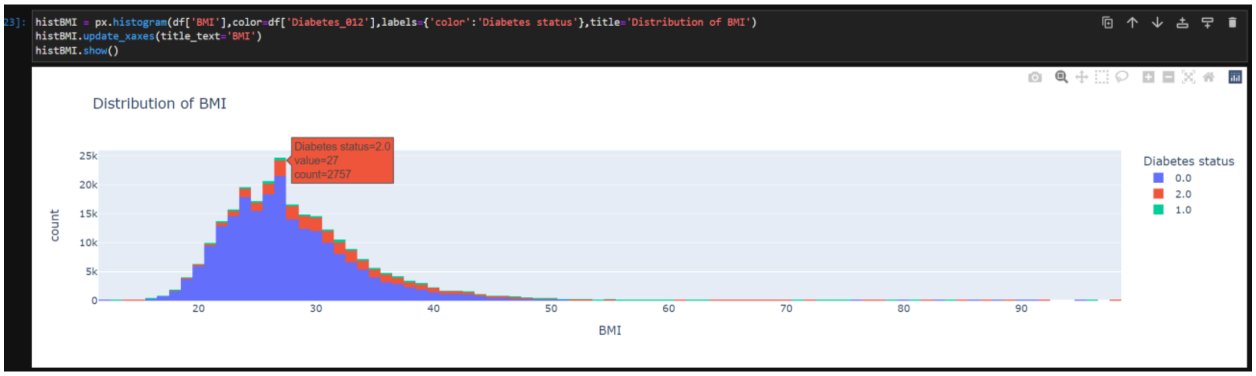


```
]: df['Diabetes_012'].value_counts()

]: Diabetes_012
   0.0    213703
   2.0     35346
   1.0      4631
   Name: count, dtype: int64

]: hist = px.histogram(df['Diabetes_012'], color=df['Diabetes_012'], title='Distribution of Diabetes cases')

]: hist.show()
```

Figure 3: Distribution of our target variable

Here we have a tabular as well as visual representation of how different cases of diabetes are distributed across the dataset. There is severe class imbalance, and this is one of the most important things to take care of before training any classifier model.

After target variable, we also analysis a few other variables as follows:

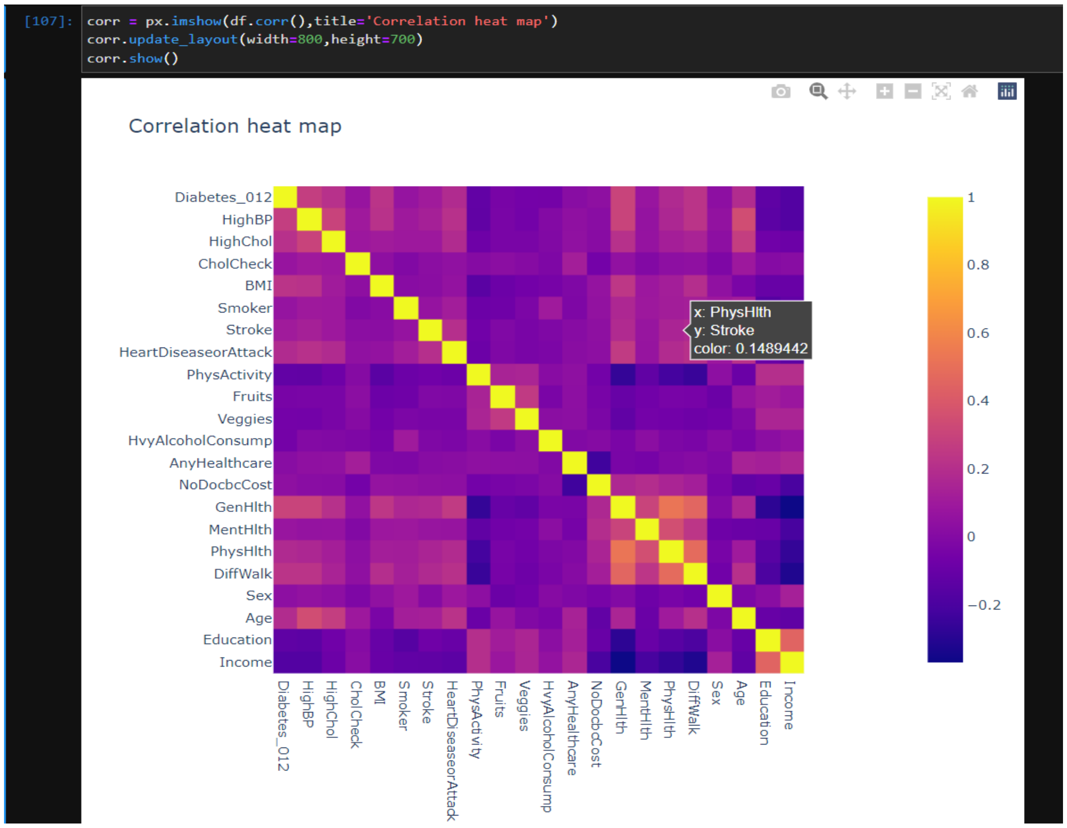


```
23]: histBMI = px.histogram(df['BMI'],color=df['Diabetes_012'],labels={'color':'Diabetes status'},title='Distribution of BMI')
     histBMI.update_xaxes(title_text='BMI')
     histBMI.show()
```

Figure 4: BMI analysis

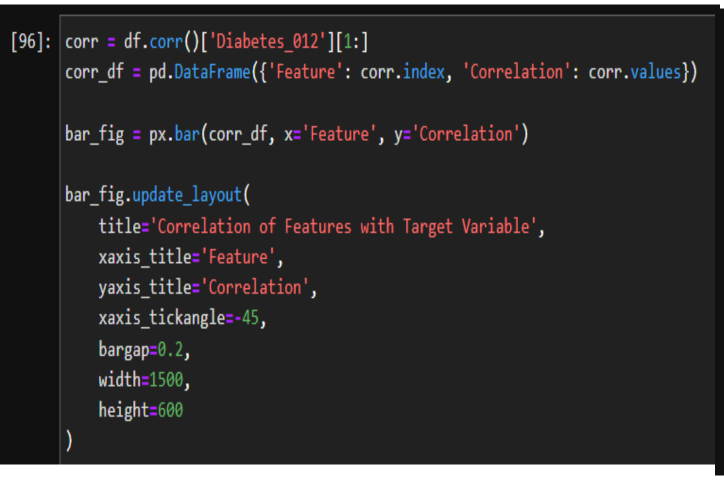
The above histogram shows distribution of BMI. We know from our analysis that there are 84 unique values.

We also visualize correlations between our Diabetes status indicator and all the 21 features. More details can be seen on hovering the cursor over various elements in the plot. We have a correlation heatmap for all the variables (including target). We even generated a bar plot that shows correlation of the target variable with all the features.

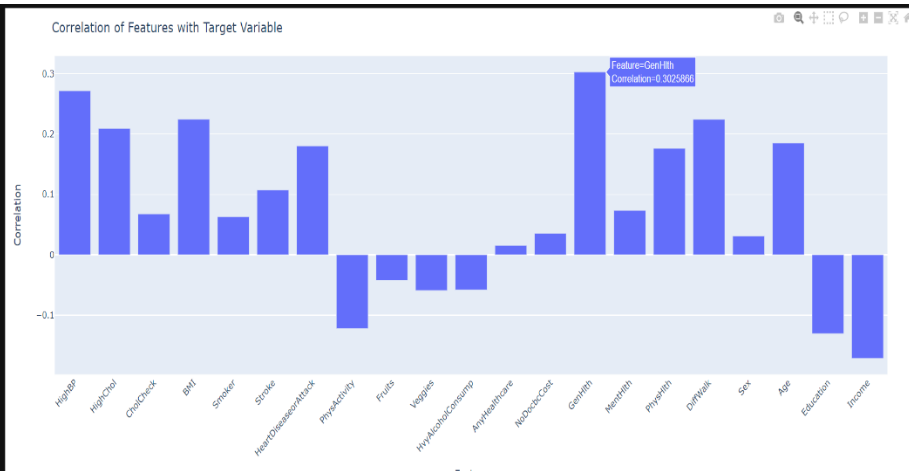


```
[107]: corr = px.imshow(df.corr(),title='Correlation heat map')
       corr.update_layout(width=800,height=700)
       corr.show()
```

Figure 5: Correlation heatmap for all variables (including target variable)



```
[96]: corr = df.corr()['Diabetes_012'][1:]
      corr_df = pd.DataFrame({'Feature': corr.index, 'Correlation': corr.values})

      bar_fig = px.bar(corr_df, x='Feature', y='Correlation')

      bar_fig.update_layout(
          title='Correlation of Features with Target Variable',
          xaxis_title='Feature',
          yaxis_title='Correlation',
          xaxis_tickangle=-45,
          bargap=0.2,
          width=1500,
          height=600
      )
```



(a) *(b)*

Figure 6: (a) Code snippet for bar plots; (b) Bar plot showing correlation between target variable and features