AI Based Diabetes Prediction System

An AI-based diabetes prediction system is a computer program or application that leverages artificial intelligence (AI) techniques to predict the likelihood of an individual developing diabetes or to help manage the condition for those who are already diagnosed with diabetes. This technology is designed to provide valuable insights, early detection, and personalized recommendations for individuals at risk of or living with diabetes

Data loading:

Load the dataset into our programming environment. We can use libraries like Pandas in Python for this task.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in
the input directory

from subprocess import check_output
print(check_output(["ls", "../input"]).decode("utf8"))

# Any results you write to the current directory are saved as output.
```

```
demographic.csv
diet.csv
examination.csv
labs.csv
medications.csv
questionnaire.csv
```

Data exploration:

Explore the dataset to understand its structure and features. This step helps to gain insights into the data and decide which features are relevant for our prediction model.

```
df1 = pd.read_csv('../input/labs.csv')
df2 = pd.read_csv('../input/examination.csv')
df3 = pd.read_csv('../input/demographic.csv')
df4 = pd.read_csv('../input/diet.csv')
df5 = pd.read_csv('../input/questionnaire.csv')

df2.drop(['SEQN'], axis = 1, inplace=True)
df3.drop(['SEQN'], axis = 1, inplace=True)
df4.drop(['SEQN'], axis = 1, inplace=True)
df5.drop(['SEQN'], axis = 1, inplace=True)

df = pd.concat([df1, df2], axis=1, join='inner')
df = pd.concat([df, df3], axis=1, join='inner')
df = pd.concat([df, df4], axis=1, join='inner')
df = pd.concat([df, df5], axis=1, join='inner')

#sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
#sel.fit_transform(df)

df.describe()
```

	SEQN	URXUMA	URXUMS	URXUCR.x	URXCRS	URDACT	WTSAF2YR.x
count	9813.000000	8052.000000	8052.000000	8052.000000	8052.000000	8052.000000	3329.000000
mean	78644.559971	41.218854	41.218854	121.072529	10702.811525	41.905695	78917.195254
std	2938.592266	238.910226	238.910226	78.574882	6946.019595	276.261093	71088.020067
min	73557.000000	0.210000	0.210000	5.000000	442.000000	0.210000	0.000000
25%	76092.000000	4.500000	4.500000	60.000000	5304.000000	5.020000	33217.405018
50%	78643.000000	8.400000	8.400000	106.000000	9370.400000	7.780000	56397.702304
75%	81191.000000	17.625000	17.625000	163.000000	14409.200000	15.295000	99356.561999
max	83731.000000	9600.000000	9600.000000	659.000000	58255.600000	9000.000000	395978.465792

Dropping Rows and Columns with all NaN Values:

dropping rows and columns with all NaN values, renaming columns, selecting specific columns, and describing the DataFrame

```
from sklearn.feature_selection import VarianceThreshold

df.dropna(axis=1, how='all')
df.dropna(axis=0, how='all')

df = df.rename(columns = {'SEQN' : 'ID',
                          'RIAGENDR' : 'Gender',
                          'DMDYRSUS' : 'Years_in_US', # Nan -> american i guess
                          'INDFMPIR' : 'Family_income',
                          'LBXGH' : 'GlycoHemoglobin',
                          'BMXARMC' : 'ArmCircum',
                          'BMDAVSAD' : 'SaggitalAbdominal',
                          'MGDCGSZ' : 'GripStrength',
                          'DRABF' : 'Breast_fed'})

df = df.loc[:, ['ID', 'Gender', 'Years_in_US', 'Family_income','GlycoHemoglobin', 'ArmCirc
um',
                'SaggitalAbdominal', 'GripStrength', 'Breast_fed']]

df.describe()
```

	ID	Gender	Years_in_US	Family_income	GlycoHemoglobin	ArmCircum	SaggitalAbdomi
count	9813.000000	9813.000000	1837.000000	9051.000000	6643.000000	9301.000000	7218.000000
mean	78644.559971	1.509426	8.933043	2.253101	5.642556	28.485765	21.114034
std	2938.592266	0.499937	17.787060	1.635458	1.004850	7.961971	4.963949
min	73557.000000	1.000000	1.000000	0.000000	3.500000	10.400000	10.100000
25%	76092.000000	1.000000	3.000000	0.870000	5.200000	22.600000	17.300000
50%	78643.000000	2.000000	5.000000	1.710000	5.400000	29.300000	20.700000
75%	81191.000000	2.000000	7.000000	3.610000	5.800000	34.000000	24.400000
max	83731.000000	2.000000	99.000000	5.000000	17.500000	59.400000	40.100000

Data Processing:

1. **Handling Missing Values:**
    - Years_in_US: Imputed missing values with 0 for non-American.
    - GlycoHemoglobin, '*SaggitalAbdominal*', '*ArmCircum*', and '*GripStrength*': Filled missing values with their respective medians.
    - Family_income: Filled missing values using forward fill method.
    - Breast_fed: Filled missing values with 1.

```
#Family Income -> use ffill to fill na
df['Family_income'] = df['Family_income'].fillna(method='ffill')

#Breat_fed -> fill to 1
df['Breast_fed'] = df['Breast_fed'].fillna(value = 1)
```

```
from sklearn.feature_selection import VarianceThreshold

#year in us -> american : 0, not american : 1
df.dropna(axis=1, how='all')
df.dropna(axis=0, how='all')

#YEARS IN US NA처리
df['Years_in_US'] = df['Years_in_US'].apply(lambda x: x if x > 0 else 0)

#GlycoHemoglobin, Saggital Abdominal(median)
df['GlycoHemoglobin'] = df['GlycoHemoglobin'].fillna(df['GlycoHemoglobin'].median())
df['SaggitalAbdominal'] = df['SaggitalAbdominal'].fillna(df['SaggitalAbdominal'].median())
df['ArmCircum'] = df['ArmCircum'].fillna(df['ArmCircum'].median())
df['GripStrength'] = df['GripStrength'].fillna(df['GripStrength'].median())
```

2. **Feature Selection:**

   Used VarianceThreshold from scikit-learn to select features with variance above a certain threshold.

```
sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
sel.fit_transform(df)
```

```
df.describe()
```

Out[5]:

	ID	Gender	Years_in_US	Family_income	GlycoHemoglobin	ArmCircum	SaggitalAbdomi
count	9813.000000	9813.000000	9813.000000	9813.000000	9813.000000	9813.000000	9813.000000
mean	78644.559971	1.509426	1.672271	2.246973	5.564201	28.528248	21.004545
std	2938.592266	0.499937	8.446506	1.635495	0.834491	7.753571	4.261142
min	73557.000000	1.000000	0.000000	0.000000	3.500000	10.400000	10.100000
25%	76092.000000	1.000000	0.000000	0.870000	5.300000	23.200000	18.500000
50%	78643.000000	2.000000	0.000000	1.700000	5.400000	29.300000	20.700000
75%	81191.000000	2.000000	0.000000	3.600000	5.600000	33.800000	22.900000
max	83731.000000	2.000000	99.000000	5.000000	17.500000	59.400000	40.100000

Categorizing Based on GlycoHemoglobin:

creating a new column called 'Diabetes' based on the values in the 'GlycoHemoglobin' column. The code that provided sets the 'Diabetes' column values according to certain conditions in the 'GlycoHemoglobin' column:

- If 'GlycoHemoglobin' is less than 6.0, 'Diabetes' is set to 0.
- If 'GlycoHemoglobin' is between 6.0 and 6.4 (inclusive), 'Diabetes' is set to 1.
- If 'GlycoHemoglobin' is greater than or equal to 6.5, 'Diabetes' is set to 2.

```
df.loc[df['GlycoHemoglobin'] < 6.0, 'Diabetes'] = 0
df.loc[(df['GlycoHemoglobin'] >= 6.0) & (df['GlycoHemoglobin'] <= 6.4), 'Diabetes'] = 1
df.loc[df['GlycoHemoglobin'] >= 6.5, 'Diabetes'] = 2

df.head(10)
```

	ID	Gender	Years_in_US	Family_income	GlycoHemoglobin	ArmCircum	SaggitalAbdominal	GripStrength	Bre
0	73557	1	0.0	0.84	13.9	35.3	20.6	55.2	2.0
1	73558	1	0.0	1.78	9.1	34.7	24.4	61.5	2.0
2	73559	1	0.0	4.51	8.9	33.5	25.6	91.0	2.0
3	73560	1	0.0	2.52	5.4	21.0	14.9	32.2	2.0
4	73561	2	0.0	5.00	4.9	25.2	20.7	30.9	2.0
5	73562	1	0.0	4.79	5.5	41.8	29.1	53.1	2.0
6	73563	1	0.0	5.00	5.4	14.9	20.7	60.3	1.0
7	73564	2	0.0	5.00	5.5	38.0	26.7	45.9	2.0
8	73566	1	0.0	5.00	5.4	29.0	19.9	38.8	2.0
9	73567	2	0.0	0.48	5.2	27.5	20.0	43.5	2.0

der	Years_in_US	Family_income	GlycoHemoglobin	ArmCircum	SaggitalAbdominal	GripStrength	Breast_fed	Diabetes
	0.0	0.84	13.9	35.3	20.6	55.2	2.0	2.0
	0.0	1.78	9.1	34.7	24.4	61.5	2.0	2.0
	0.0	4.51	8.9	33.5	25.6	91.0	2.0	2.0
	0.0	2.52	5.4	21.0	14.9	32.2	2.0	0.0
	0.0	5.00	4.9	25.2	20.7	30.9	2.0	0.0
	0.0	4.79	5.5	41.8	29.1	53.1	2.0	0.0
	0.0	5.00	5.4	14.9	20.7	60.3	1.0	0.0
	0.0	5.00	5.5	38.0	26.7	45.9	2.0	0.0
	0.0	5.00	5.4	29.0	19.9	38.8	2.0	0.0
	0.0	0.48	5.2	27.5	20.0	43.5	2.0	0.0

Conclusion:

The AI diabetes phase 3 diabetes prediction project, our focus was on loading and preprocessing the dataset. This is a crucial first step in any machine learning project as it ensures that the data is in a suitable format for model training. Now that the dataset has been properly loaded and preprocessed, we are ready to move on to the next stages, which typically include model selection, training, and evaluation.

In conclusion, the successful loading and preprocessing of the diabetes dataset are vital steps in the development of the prediction model. They enable us to work with the data effectively and pave the way for the model's construction. In the subsequent phases, our focus will be on model building, training, and evaluation, with the aim of creating an accurate and valuable diabetes prediction system