Import Libraries

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

Reading data from csv

```
path = 'data.csv'
data = pd.read_csv(path)
data.head()
```

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI
0	1	39	4.0	0	0.0	0.0	0	0	0	195.0	106.0	70.0	26.97
1	0	46	2.0	0	0.0	0.0	0	0	0	250.0	121.0	81.0	28.73
2	1	48	1.0	1	20.0	0.0	0	0	0	245.0	127.5	80.0	25.34
3	0	61	3.0	1	30.0	0.0	0	1	0	225.0	150.0	95.0	28.58
4	0	46	3.0	1	23.0	0.0	0	0	0	285.0	130.0	84.0	23.10

Shape of the dataset

Data contains 4238 rows and 16 columns

```
data.shape
```

```
(4240, 16)
```

Information on the datatype of columns

Information about the datatype of each column contained in our data

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4240 entries, 0 to 4239
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   male             4240 non-null   int64
 1   age              4240 non-null   int64
 2   education        4135 non-null   float64
 3   currentSmoker    4240 non-null   int64
 4   cigsPerDay       4211 non-null   float64
 5   BPMeds           4187 non-null   float64
 6   prevalentStroke  4240 non-null   int64
 7   prevalentHyp     4240 non-null   int64
 8   diabetes         4240 non-null   int64
 9   totChol          4190 non-null   float64
 10  sysBP            4240 non-null   float64
 11  diaBP            4240 non-null   float64
 12  BMI              4221 non-null   float64
 13  heartRate        4239 non-null   float64
 14  glucose          3852 non-null   float64
 15  TenYearCHD       4240 non-null   int64
dtypes: float64(9), int64(7)
memory usage: 530.1 KB
```

Looking at the Columns in the dataset

```
data.columns
```

```
Index(['male', 'age', 'education', 'currentSmoker', 'cigsPerDay', 'BPMeds',
       'prevalentStroke', 'prevalentHyp', 'diabetes', 'totChol', 'sysBP',
       'diaBP', 'BMI', 'heartRate', 'glucose', 'TenYearCHD'],
      dtype='object')
```

Descibing the dataset

• Sex: male or female("M" or "F")

• Age: Age of the patient;(Continuous - Although the recorded ages have been truncated to whole numbers, the concept of age is continuous)Behavioral

• is_smoking: whether or not the patient is a current smoker ("YES" or "NO")

• Cigs Per Day: the number of cigarettes that the person smoked on average in one day.(can be considered continuous as one can have any number of cigarettes, even half a cigarette.)Medical(history)

• BP Meds: whether or not the patient was on blood pressure medication (Nominal)

• Prevalent Stroke: whether or not the patient had previously had a stroke (Nominal)

• Prevalent Hyp: whether or not the patient was hypertensive (Nominal)

• Diabetes: whether or not the patient had diabetes (Nominal)Medical(current)

• Tot Chol: total cholesterol level (Continuous)• Sys BP: systolic blood pressure (Continuous)• Dia BP: diastolic blood pressure (Continuous)• BMI: Body Mass Index (Continuous)

• Heart Rate: heart rate (Continuous - In medical research, variables such as heart rate though in fact discrete, yet are considered continuous because of large number of possible values.)

• Glucose: glucose level (Continuous)Predict variable (desired target)

• 10 year risk of coronary heart disease CHD(binary: “1”, means “Yes”, “0” means “No”)

Statistical Summaries for Numeric Columns

Feature Engineering

```
1 data.describe()
```

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	t
count	4240.000000	4240.000000	4135.000000	4240.000000	4211.000000	4187.000000	4240.000000	4240.000000	4240.000000	4190
mean	0.429245	49.580189	1.979444	0.494104	9.005937	0.029615	0.005896	0.310613	0.025708	236
std	0.495027	8.572942	1.019791	0.500024	11.922462	0.169544	0.076569	0.462799	0.158280	44
min	0.000000	32.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	107
25%	0.000000	42.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	206
50%	0.000000	49.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	234
75%	1.000000	56.000000	3.000000	1.000000	20.000000	0.000000	0.000000	1.000000	0.000000	263
max	1.000000	70.000000	4.000000	1.000000	70.000000	1.000000	1.000000	1.000000	1.000000	696

Remove Null Values from the data

Null values affects the performance and accuracy of the machine learning model and therefore it is important to identify them and deal with them accordingly.

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

```
male                 0
age                  0
education          105
currentSmoker        0
cigsPerDay          29
BPMeds              53
prevalentStroke      0
prevalentHyp         0
```

```
diabetes              0
totChol              50
sysBP                 0
diaBP                 0
BMI                  19
heartRate             1
glucose             388
TenYearCHD            0
dtype: int64
```

Replace null values with mean values for respective columns

We replaced columns with respective values so that we can still use those rows in our data

```
columns_with_na = ['education','cigsPerDay','BPMeds','totChol','BMI','heartRate','glucose']

for i in columns_with_na:
    data[i].fillna(data[i].mean(),inplace=True)

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

```
male               0
age                0
education          0
currentSmoker      0
cigsPerDay         0
BPMeds             0
prevalentStroke    0
prevalentHyp       0
diabetes           0
totChol            0
sysBP              0
diaBP              0
BMI                0
heartRate          0
glucose            0
TenYearCHD         0
dtype: int64
```

Check unique values for each column

In another attempt to clearly understand the data provided, we may decide to know unique values contained in each column. It can be seen below that we have six columns (***male,currentSmoker,prevalentStroke, prevalentHyp, diabetes,TenYearCHD***) which are binary (containing only 2 possible values.)

This might not be a very useful piece of information - But it helps us to know better how to treat each column in our dataset.

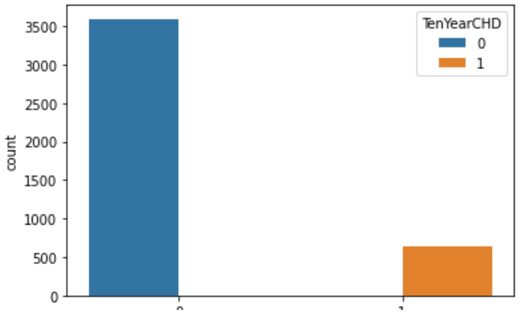
```
data.nunique()
```

```
male                  2
age                  39
education             5
currentSmoker         2
cigsPerDay           34
BPMeds                3
prevalentStroke       2
prevalentHyp          2
diabetes              2
totChol             249
sysBP               234
diaBP               146
BMI                1365
heartRate            74
glucose             144
TenYearCHD            2
dtype: int64
```

Exploratory Data Analysis

```
sns.countplot(data['TenYearCHD'],hue=data['TenYearCHD'])
plt.show()
```

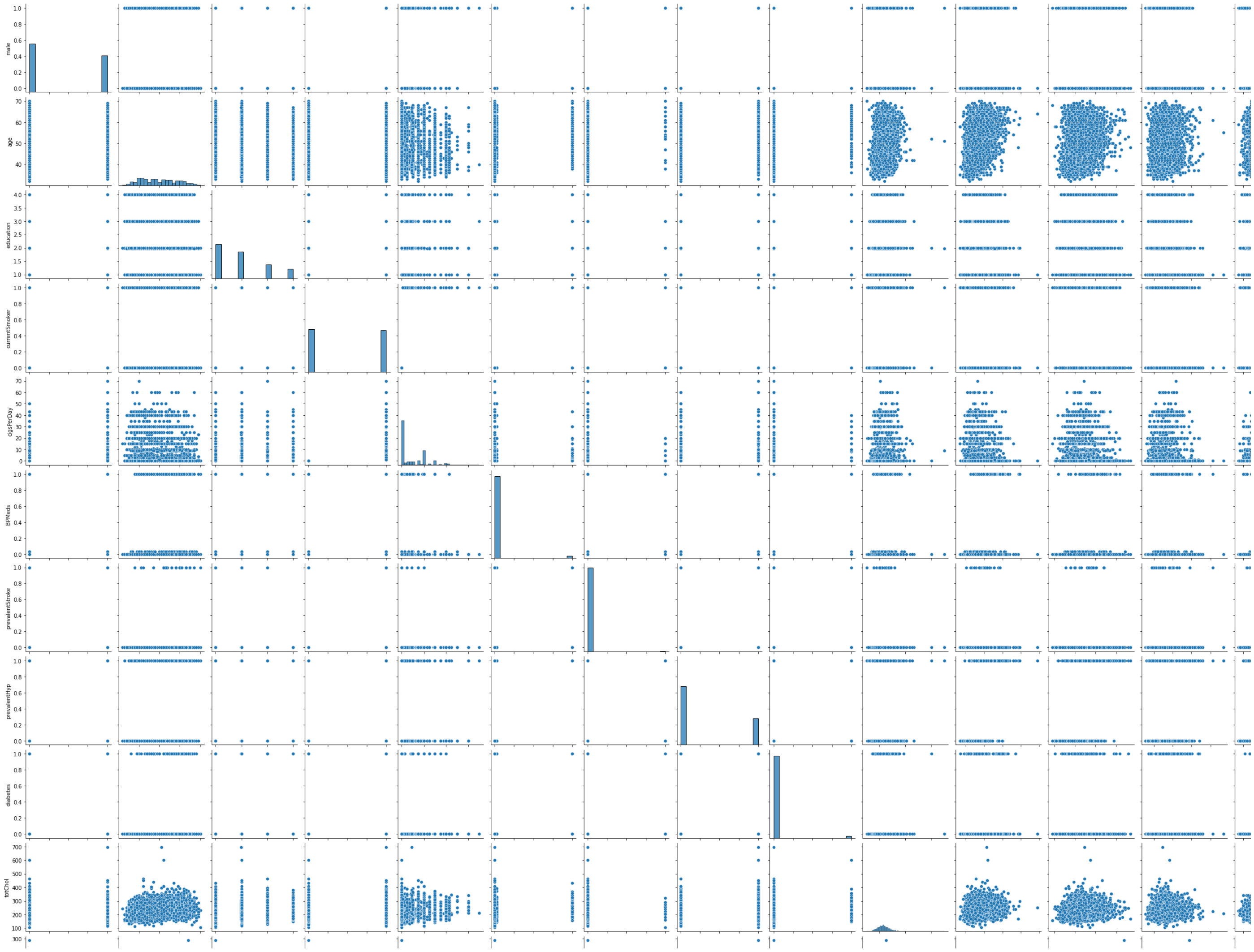
```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg
  warnings.warn(
```



It can be seen from the histogram above that, the dataset is highly unbalanced. It contains about 3500 examples with patients without the risk and only around 500 examples of patitents identified under risk.

Analyse relationship between pair of all features in the dataset

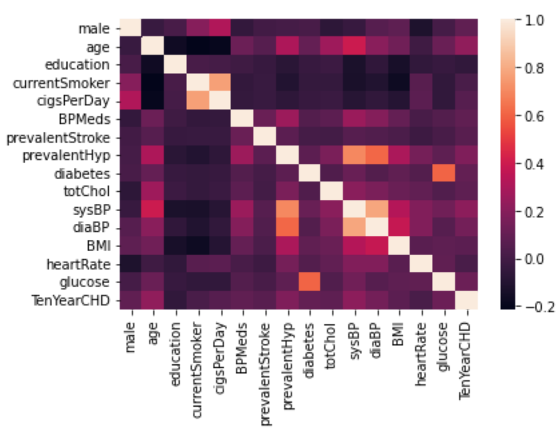
```
sns.pairplot(data)
plt.show()
```



Correlation Heat Maps

Correlation heat maps help us identify the features which are correlated, it is considered better to get rid of them, since they add no new information for the model to run

```
1 sns.heatmap(data.corr())
2 plt.show()
```



We use the information above to remove highly correlated features

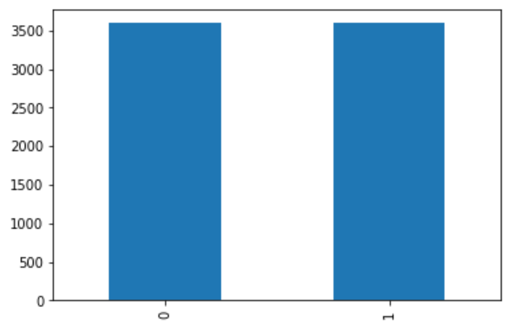
```
1 highly_correlated_features = ['currentSmoker','diaBP','prevalentHyp','diabetes']
2 data.drop(highly_correlated_features,axis=1,inplace=True)
```

Training on the dataset

```
X = data.drop('TenYearCHD',axis=1)
y = data['TenYearCHD']
```

Using SMOTE to balance the data

```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
X_ros, y_ros = smote.fit_resample(X, y)
ros_chd_plot=y_ros.value_counts().plot(kind='bar')
plt.show()
```



Train vs Test Splits

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X_ros,y_ros,test_size=0.2,random_state=42)
X_train.head()
```

	male	age	education	cigsPerDay	BPMeds	prevalentStroke	totChol	sysBP	BMI	heartRate	glucose
6256	0	50	2.463281	0.000000	0.0	0	269.146875	175.048958	24.151805	70.122396	82.000000
4668	1	44	2.000000	20.000000	0.0	0	194.588929	133.706715	21.010707	55.824500	74.350999
940	0	53	2.000000	0.000000	0.0	0	284.000000	167.500000	31.500000	88.000000	87.000000
1511	0	38	2.000000	0.000000	0.0	0	255.000000	125.000000	23.050000	72.000000	73.000000
6034	0	59	1.000000	0.731304	0.0	0	257.224352	144.753476	38.318474	74.149568	82.850432

Scale the data

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=pd.DataFrame(sc.fit_transform(X_train))
X_test=pd.DataFrame(sc.transform(X_test))
X_train.head()
```

	0	1	2	3	4	5	6	7	8	9	10
0	-0.777364	-0.155128	0.580691	-0.799126	-0.234228	-0.061958	0.640606	1.616807	-0.496426	-0.516189	-0.074275
1	1.286399	-0.877759	0.088383	0.855961	-0.234228	-0.061958	-1.002586	-0.124138	-1.297569	-1.752271	-0.340558
2	-0.777364	0.206188	0.088383	-0.799126	-0.234228	-0.061958	0.967956	1.298916	1.377747	1.029367	0.099789
3	-0.777364	-1.600390	0.088383	-0.799126	-0.234228	-0.061958	0.328821	-0.490783	-0.777443	-0.353866	-0.387590
4	-0.777364	0.928819	-0.974271	-0.738607	-0.234228	-0.061958	0.377844	0.341047	3.116813	-0.168032	-0.044669

Reset colmns (Scaling removed column indexes)

```
X_train.columns= X.columns
X_test.columns= X.columns
```

```

y_train.index= X_train.index
y_test.index= X_test.index

X_train.head()
```

	male	age	education	cigsPerDay	BPMeds	prevalentStroke	totChol	sysBP	BMI	heartRate	glucose
0	-0.777364	-0.155128	0.580691	-0.799126	-0.234228	-0.061958	0.640606	1.616807	-0.496426	-0.516189	-0.074275
1	1.286399	-0.877759	0.088383	0.855961	-0.234228	-0.061958	-1.002586	-0.124138	-1.297569	-1.752271	-0.340558
2	-0.777364	0.206188	0.088383	-0.799126	-0.234228	-0.061958	0.967956	1.298916	1.377747	1.029367	0.099789
3	-0.777364	-1.600390	0.088383	-0.799126	-0.234228	-0.061958	0.328821	-0.490783	-0.777443	-0.353866	-0.387590
4	-0.777364	0.928819	-0.974271	-0.738607	-0.234228	-0.061958	0.377844	0.341047	3.116813	-0.168032	-0.044669

Create a Logistic regression model

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train,y_train)
```

```
LogisticRegression()
```

```
pred = model.predict_proba(X_test)
```

```
pred.shape
```

```
(1439, 2)
```

```
def predictPremium(X_test):
    probabilities = model.predict_proba(X_test)
    print(probabilities)
    noDiabetes,diabetesProne = probabilities[0][0], probabilities[0][1]
    base = 5000
    return base * (1 + diabetesProne*1.8)

```

```
# predictPremium([1, 59, 1.0, 21.595, 0.0, 0, 210.765, 166.19, 27.02, 75.68, 71.5])
predictPremium( X_train.head(1))
```

```
[[0.4157976 0.5842024]]
10257.821635935868
```

Double-click (or enter) to edit

```
import pickle
with open('model.pkl','wb') as f:
    pickle.dump(model,f)
```