LINEAR UNIVARIATE

IMPORTING HEADER FILES

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
df = pd.read_csv("Life Expectancy Data.csv")
df
```

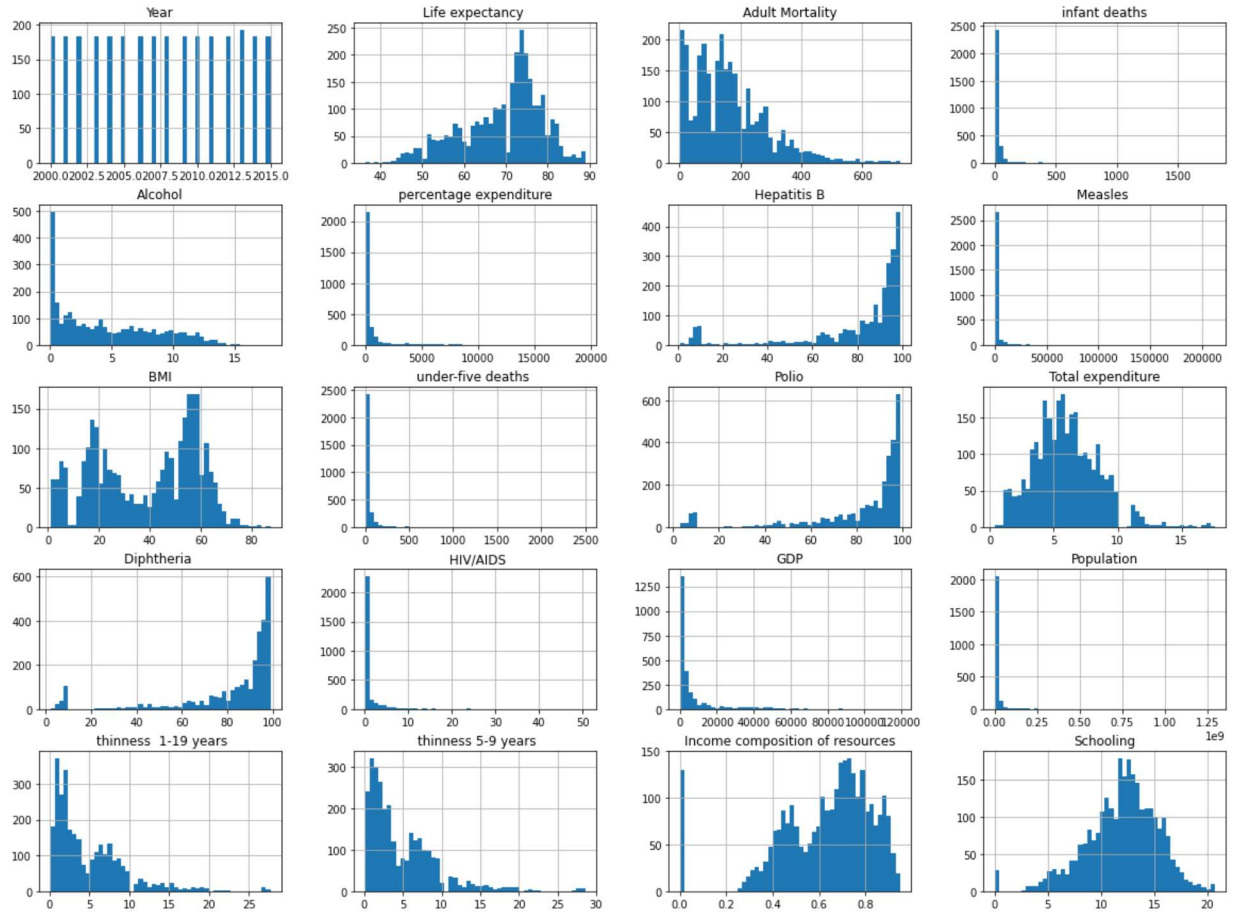
Out[1]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0
...	...	...	...	...	...	...	...	...	...
2933	Zimbabwe	2004	Developing	44.3	723.0	27	4.36	0.000000	68.0
2934	Zimbabwe	2003	Developing	44.5	715.0	26	4.06	0.000000	7.0
2935	Zimbabwe	2002	Developing	44.8	73.0	25	4.43	0.000000	73.0
2936	Zimbabwe	2001	Developing	45.3	686.0	25	1.72	0.000000	76.0
2937	Zimbabwe	2000	Developing	46.0	665.0	24	1.68	0.000000	79.0

2938 rows × 22 columns

CHECKING THE DISTRIBUTION OF THE DATA

In [2]:
```
import matplotlib
df.hist(bins=50, figsize=(20,15))
matplotlib.pyplot.show()
```



In [3]:
```
df.shape
```

Out[3]: (2938, 22)

In [4]:
```
df.describe()
```

Out[4]:

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis E
count	2938.000000	2928.000000	2928.000000	2938.000000	2744.000000	2938.000000	2385.00000
mean	2007.518720	69.224932	164.796448	30.303948	4.602861	738.251295	80.94046
std	4.613841	9.523867	124.292079	117.926501	4.052413	1987.914858	25.07001
min	2000.000000	36.300000	1.000000	0.000000	0.010000	0.000000	1.00000
25%	2004.000000	63.100000	74.000000	0.000000	0.877500	4.685343	77.00000
50%	2008.000000	72.100000	144.000000	3.000000	3.755000	64.912906	92.00000
75%	2012.000000	75.700000	228.000000	22.000000	7.702500	441.534144	97.00000
max	2015.000000	89.000000	723.000000	1800.000000	17.870000	19479.911610	99.00000

CHECKING FOR NULL VALUE ROWS

In [5]:
```
df.isna().sum()
```

Out[5]:
```
Country                            0
Year                               0
Status                             0
Life expectancy                   10
Adult Mortality                   10
infant deaths                      0
Alcohol                          194
percentage expenditure             0
Hepatitis B                      553
Measles                            0
 BMI                              34
under-five deaths                  0
Polio                             19
Total expenditure                226
Diphtheria                        19
 HIV/AIDS                          0
GDP                              448
Population                       652
 thinness  1-19 years             34
 thinness 5-9 years               34
Income composition of resources  167
Schooling                        163
dtype: int64
```

In [6]:
```
df.rename(columns={'Income composition of resources':'income', 'Life expectancy '
```

In [7]:
```
df.index[df["Life_expectancy"].isna()]
```

Out[7]:
```
Int64Index([624, 769, 1650, 1715, 1812, 1909, 1958, 2167, 2216, 2713], dtype='i
nt64')
```

DROPPING THE REQUIRED ROWS WITH NULL VALUES

In [8]:
```
df.drop(axis='rows', labels=df.index[df["Life_expectancy"].isna()], inplace=True)
df.drop(axis='rows', labels=df.index[df["income"].isna()], inplace=True)
```

In [9]:
```
df.isna().sum()
```

Out[9]:
```
Country                          0
Year                             0
Status                           0
Life_expectancy                  0
Adult Mortality                  0
infant deaths                    0
Alcohol                        184
percentage expenditure           0
Hepatitis B                    509
Measles                          0
 BMI                            32
under-five deaths                0
Polio                           19
Total expenditure              186
Diphtheria                      19
 HIV/AIDS                        0
GDP                            286
Population                     484
 thinness  1-19 years           32
 thinness 5-9 years             32
income                           0
Schooling                        0
dtype: int64
```

In [10]:
```
df = df.drop(['Country'], axis=1)
```

In [11]:
```
df.shape
```

Out[11]:
```
(2768, 21)
```

In [12]:
```
df
```

Out[12]:

	Year	Status	Life_expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles
0	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	115
1	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	49
2	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	43
3	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	278
4	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	301
...	...	...	...	...	...	...	...	...	..
2933	2004	Developing	44.3	723.0	27	4.36	0.000000	68.0	3
2934	2003	Developing	44.5	715.0	26	4.06	0.000000	7.0	99
2935	2002	Developing	44.8	73.0	25	4.43	0.000000	73.0	30
2936	2001	Developing	45.3	686.0	25	1.72	0.000000	76.0	52
2937	2000	Developing	46.0	665.0	24	1.68	0.000000	79.0	148

2768 rows × 21 columns

In [13]:
```
df["Status"].unique()
```

Out[13]:
```
array(['Developing', 'Developed'], dtype=object)
```

In [14]:
```
df['Status'] = df['Status'].replace('Developing',0).replace('Developed',1)
df["Status"].unique()
```

Out[14]:
```
array([0, 1], dtype=int64)
```

CHECKING THE CORRELATION OF FEATURES

In [15]:
```
correlation = df.corr()
correlation
```

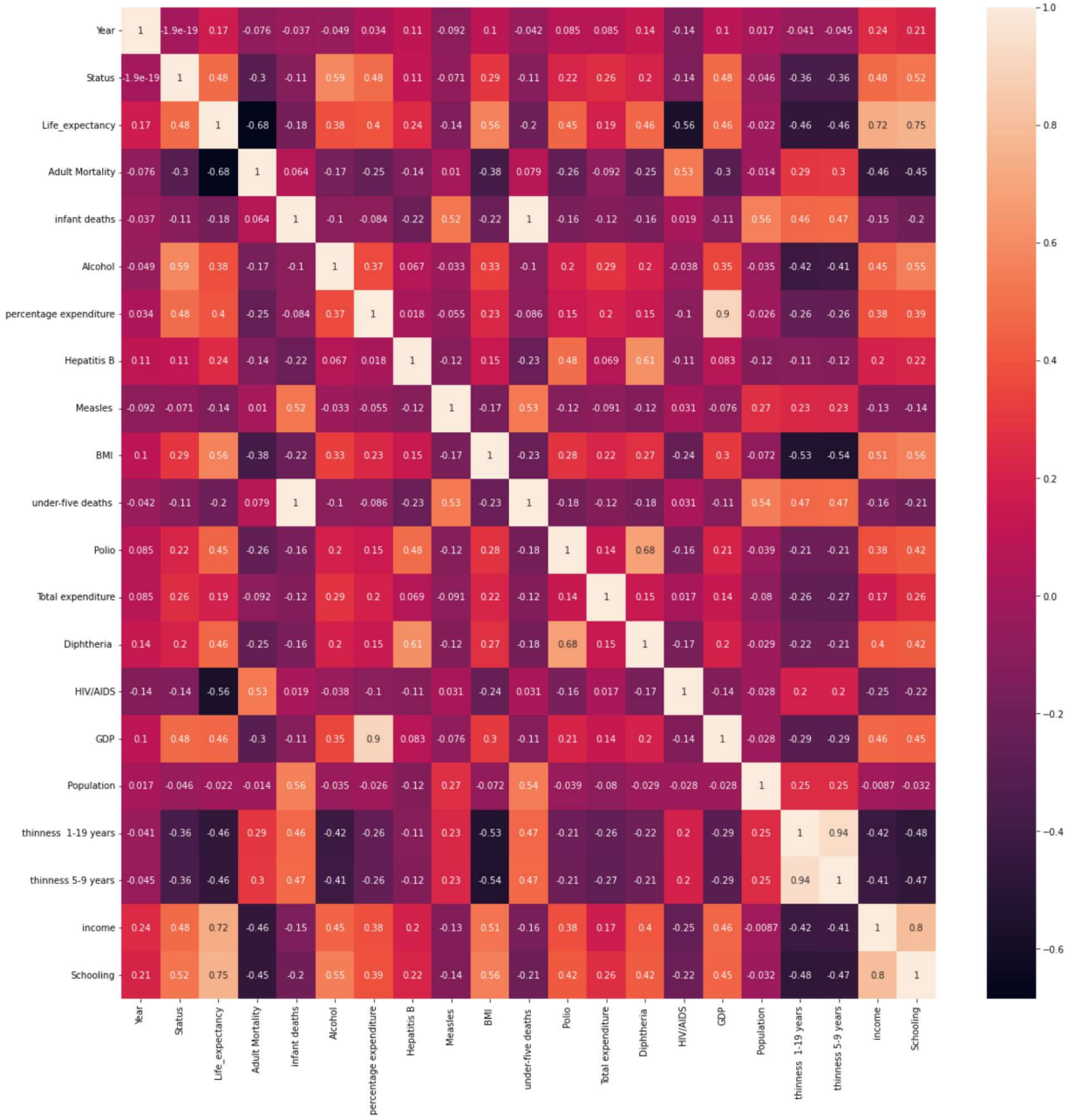
Out[15]:

	Year	Status	Life_expectancy	Adult Mortality	infant deaths	Alcohol	per exp
Year	1.000000e+00	-1.863462e-19	0.171042	-0.075688	-0.037298	-0.049335	
Status	-1.863462e-19	1.000000e+00	0.475475	-0.299422	-0.107295	0.585408	
Life_expectancy	1.710425e-01	4.754751e-01	1.000000	-0.684585	-0.179548	0.381469	
Adult Mortality	-7.568771e-02	-2.994216e-01	-0.684585	1.000000	0.063906	-0.171503	-(
infant deaths	-3.729756e-02	-1.072952e-01	-0.179548	0.063906	1.000000	-0.104269	-(
Alcohol	-4.933502e-02	5.854083e-01	0.381469	-0.171503	-0.104269	1.000000	
percentage expenditure	3.379037e-02	4.842085e-01	0.396084	-0.248033	-0.083809	0.365709	
Hepatitis B	1.102926e-01	1.121208e-01	0.240207	-0.143838	-0.217538	0.066802	
Measles	-9.210178e-02	-7.121249e-02	-0.141217	0.010355	0.521308	-0.032547	-(
BMI	1.017215e-01	2.915152e-01	0.563736	-0.375359	-0.219385	0.325384	
under-five deaths	-4.236229e-02	-1.094195e-01	-0.204858	0.078506	0.996618	-0.099810	-(
Polio	8.477105e-02	2.199571e-01	0.454795	-0.260643	-0.162037	0.201996	
Total expenditure	8.497195e-02	2.622620e-01	0.193112	-0.091961	-0.124206	0.287211	
Diphtheria	1.399374e-01	2.048133e-01	0.464119	-0.252511	-0.163022	0.201957	
HIV/AIDS	-1.368334e-01	-1.436871e-01	-0.563175	0.529330	0.018823	-0.038350	-(
GDP	1.037091e-01	4.780088e-01	0.461250	-0.295616	-0.108568	0.354148	
Population	1.719907e-02	-4.589220e-02	-0.021538	-0.013647	0.556781	-0.035376	-(
thinness 1-19 years	-4.056657e-02	-3.577153e-01	-0.464487	0.289868	0.463872	-0.417539	-(
thinness 5-9 years	-4.453943e-02	-3.560581e-01	-0.459313	0.296367	0.470227	-0.405254	-(
income	2.429532e-01	4.788258e-01	0.724776	-0.457626	-0.145018	0.450254	
Schooling	2.132653e-01	5.158267e-01	0.751975	-0.454612	-0.195202	0.548001	

21 rows × 21 columns

In [16]:
```
plt.figure(figsize=(20,20))
sn.heatmap(correlation, annot = True)

plt.show()
```

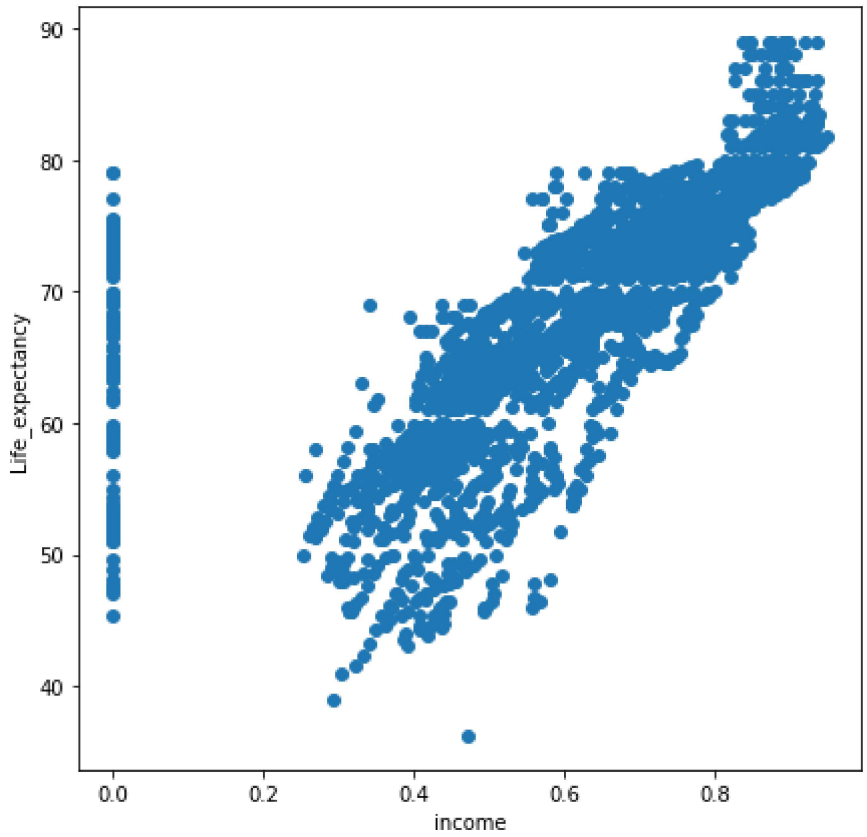


DROPPING UNRELATED COLUMNS

In [17]:
```
df = df.drop(['Year'], axis=1)
df = df.drop(['infant deaths'], axis=1)
df = df.drop(['Population'], axis=1)
```

In [18]:
```
plt.figure(figsize=(7,7))
plt.xlabel('income')
plt.ylabel('Life_expectancy')
plt.scatter(df.income,df.Life_expectancy)
```

Out[18]:
```
<matplotlib.collections.PathCollection at 0x1cfe43e6f70>
```

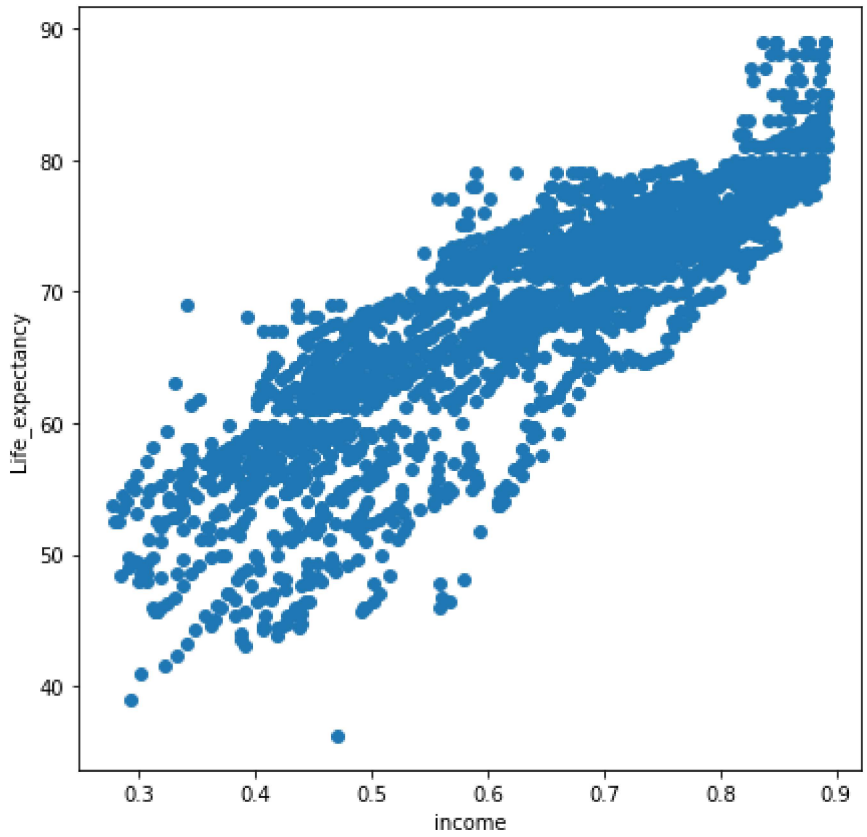


REMOVING OUTLIERS

In [19]:
```
max_threshold=df['income'].quantile(0.95)
min_threshold=df['income'].quantile(0.05)
df=df[(df['income']>min_threshold) & (df['income']<max_threshold)]
```

In [20]:
```
plt.figure(figsize=(7,7))
plt.xlabel('income')
plt.ylabel('Life_expectancy')
plt.scatter(df.income,df.Life_expectancy)
```

Out[20]:
```
<matplotlib.collections.PathCollection at 0x1cfe51db970>
```



SPLITTING TRAIN AND TEST DATA

In [21]:
```
train = df.iloc[:1800,:]
test = df.iloc[1800:,:]
```

In [22]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1800 entries, 0 to 2079
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Status                   1800 non-null   int64  
 1   Life_expectancy          1800 non-null   float64
 2   Adult Mortality          1800 non-null   float64
 3   Alcohol                  1691 non-null   float64
 4   percentage expenditure   1800 non-null   float64
 5   Hepatitis B              1523 non-null   float64
 6   Measles                  1800 non-null   int64  
 7    BMI                     1800 non-null   float64
 8   under-five deaths        1800 non-null   int64  
 9   Polio                    1798 non-null   float64
 10  Total expenditure        1689 non-null   float64
 11  Diphtheria               1798 non-null   float64
 12   HIV/AIDS                1800 non-null   float64
 13  GDP                      1645 non-null   float64
 14   thinness  1-19 years    1800 non-null   float64
 15   thinness 5-9 years      1800 non-null   float64
 16  income                   1800 non-null   float64
 17  Schooling                1800 non-null   float64
dtypes: float64(15), int64(3)
memory usage: 267.2 KB
```

In [23]:
```
test.shape
```

Out[23]:
```
(688, 18)
```

EXTRACTING THE INPUT AND TARGET VARIRABLES

In [24]:
```
x = train['income']
y = train['Life_expectancy']
```

BUILDING LINEAR MULTIVARIATE MODEL

In [25]:
```
def modeluni(x,y,l=0.001,itr=3000):
    m=0
    n=x.shape[0]
    c=0
    errlist = []
    for i in range(itr):
        h=m*x+c
        err=y-h
        cost=1/n*np.sum(np.square(err))
        errlist.append(cost)
        md=(2/n)*np.sum((err)*(-x))
        cd=(2/n)*np.sum(err*(-1))
        m=m-l*md
        c=c-l*cd
    return m,c,errlist
```

CALLING THE MODEL FUNCTION BY PASSING THE PARAMETERS

In [26]:
```
m,c,errlist = modeluni(x,y)
```

In [27]:
```
print("SLOPE : ",m)
```

```
SLOPE :  33.829017578590694
```

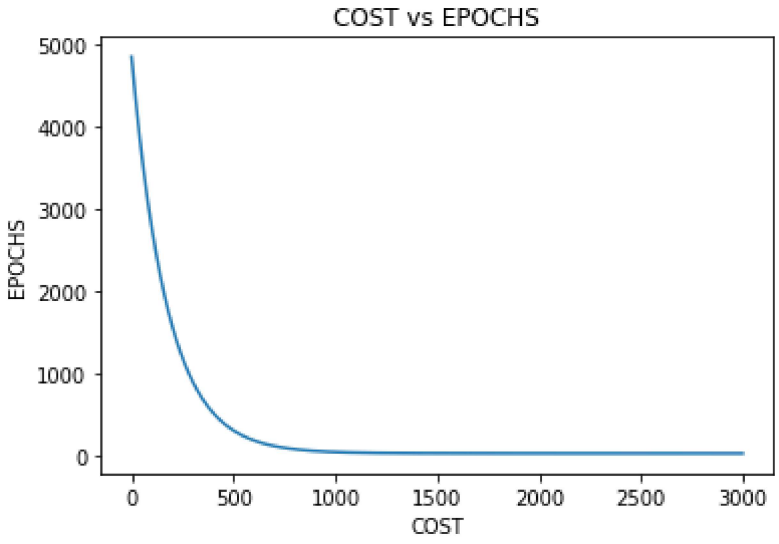
In [28]:
```
print("INTERCEPT : ",c)
```

```
INTERCEPT :  47.431961116793005
```

ERROR PLOT

In [29]:
```
plt.title('COST vs EPOCHS')
plt.xlabel('COST')
plt.ylabel('EPOCHS')
plt.plot(errlist)
```

Out[29]:
```
[<matplotlib.lines.Line2D at 0x1cfe5042910>]
```



In [30]:
```
errlist
```

Out[30]:
```
[4854.825588888888,
 4827.3597430209775,
 4800.050126254498,
 4772.89584991982,
 4745.896030402286,
 4719.0497891134555,
 4692.356252462516,
 4665.814551827862,
 4639.423823528817,
 4613.183208797536,
 4587.091853751061,
 4561.148909363529,
 4535.353531438548,
 4509.704880581724,
 4484.202122173346,
 4458.844426341228,
 4433.630967933703,
 4408.56092649277,
 4383.633486227396,
```

FINAL COST

In [31]:
```
errlist[-1]
```

Out[31]:
```
24.751446061068716
```

In [32]:
```
tgt_test=test['Life_expectancy']
```

PREDICTION

In [33]:
```
ypred=m*test['income']+c
```

In [34]:
```
ypred
```

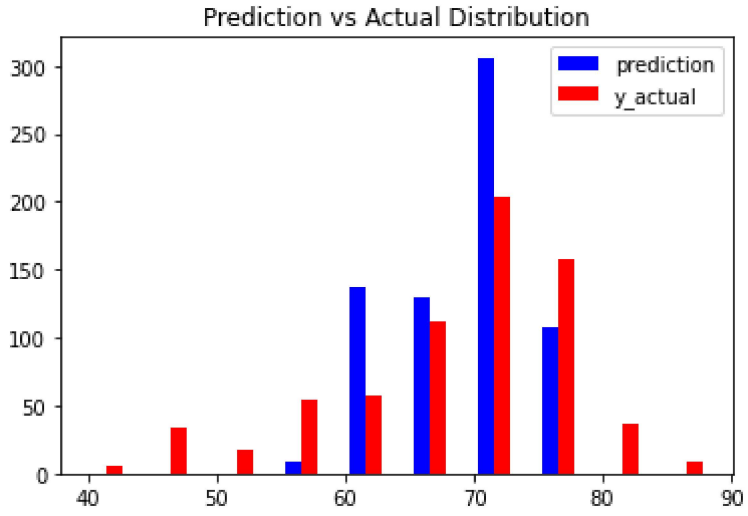
Out[34]:
```
2080    75.679191
2081    75.510046
2082    75.374730
2083    75.002610
2084    74.731978
          ...
2933    61.200371
2934    61.572490
2935    61.876952
2936    61.876952
2937    62.113755
Name: income, Length: 688, dtype: float64
```

In [35]:
```
tgt_test
```

Out[35]:
```
2080    76.6
2081    76.6
2082    76.6
2083    76.5
2084    76.4
        ...
2933    44.3
2934    44.5
2935    44.8
2936    45.3
2937    46.0
Name: Life_expectancy, Length: 688, dtype: float64
```

PREDICTION vs ACTUAL DISTRIBUTION

In [36]:
```
y_and_ypred = [ypred, tgt_test]
a = plt.subplot()
labels = ['prediction', 'y_actual']
colors = ['blue','red']
a.hist(y_and_ypred, bins = 10, rwidth=0.5,label=labels, color = colors)
a.legend(prop={'size': 10})
plt.title(label="Prediction vs Actual Distribution")
plt.show()
```



ERROR METRICS

In [37]:
```
mse=np.square(np.subtract(ypred,tgt_test)).mean()
mae = np.absolute(np.subtract(ypred,tgt_test)).mean()
```

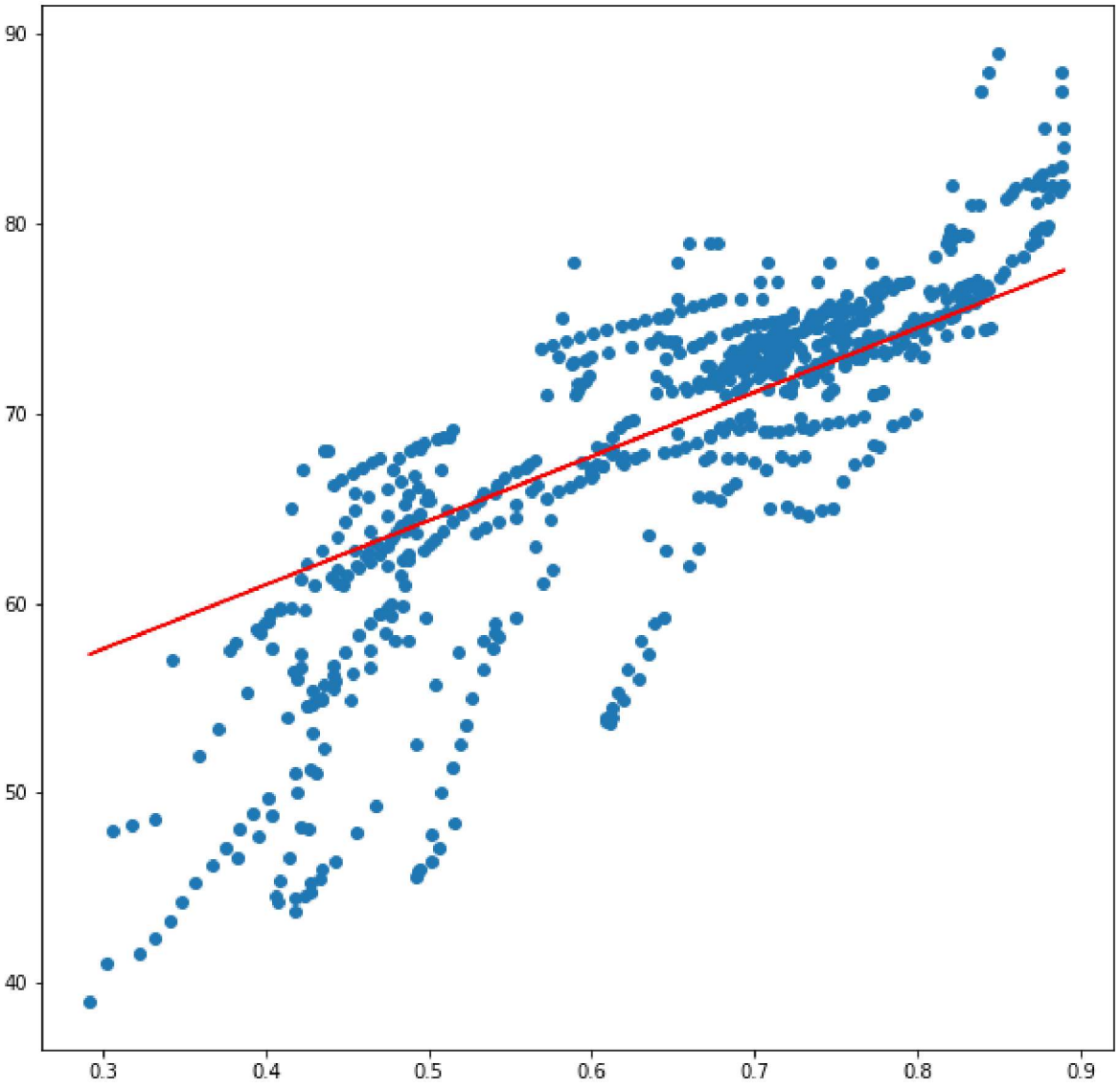
In [38]:
```
print("MEAN SQUARED ERROR : ",mse)
print("MEAN ABSOLUTE ERROR : ",mae)
```

```
MEAN SQUARED ERROR :  30.918932825837548
MEAN ABSOLUTE ERROR :  3.8226642664811674
```

MODEL PLOT

In [39]:
```
plt.figure(figsize=(10,10))
plt.scatter(test.income,test.Life_expectancy)
plt.plot(test.income,ypred,color='red')
```

Out[39]:
```
[<matplotlib.lines.Line2D at 0x1cfe48508e0>]
```



LINEAR MULTIVARIATE

IMPORTING HEADER FILES

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
df = pd.read_csv("Life Expectancy Data.csv")
df
```

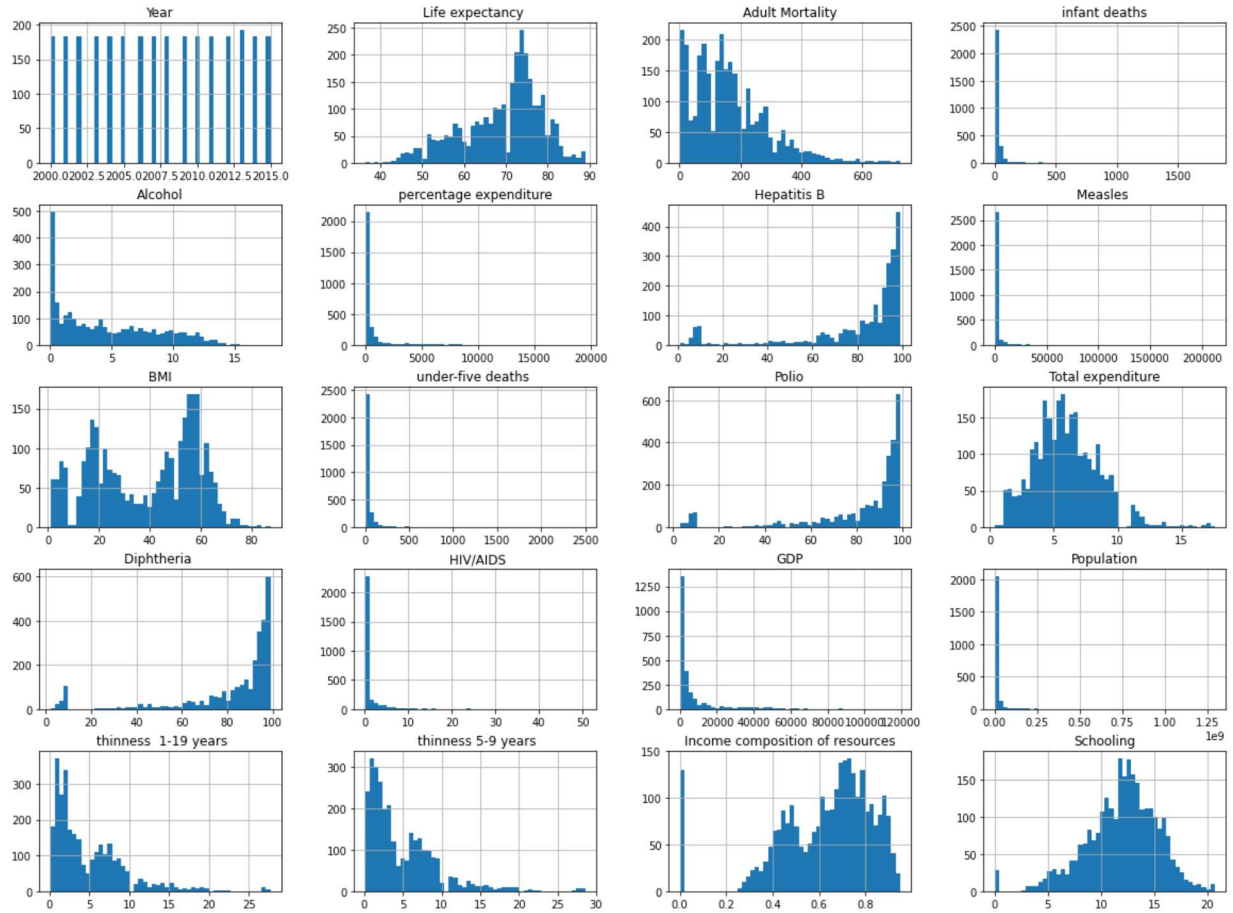
Out[1]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0
...	...	...	...	...	...	...	...	...	...
2933	Zimbabwe	2004	Developing	44.3	723.0	27	4.36	0.000000	68.0
2934	Zimbabwe	2003	Developing	44.5	715.0	26	4.06	0.000000	7.0
2935	Zimbabwe	2002	Developing	44.8	73.0	25	4.43	0.000000	73.0
2936	Zimbabwe	2001	Developing	45.3	686.0	25	1.72	0.000000	76.0
2937	Zimbabwe	2000	Developing	46.0	665.0	24	1.68	0.000000	79.0

2938 rows × 22 columns

CHECKING THE DISTRIBUTION OF THE DATA

In [2]:
```
import matplotlib
df.hist(bins=50, figsize=(20,15))
matplotlib.pyplot.show()
```



In [3]:
```
df.shape
```

Out[3]: (2938, 22)

In [4]:
```
df.describe()
```

Out[4]:

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis E
count	2938.000000	2928.000000	2928.000000	2938.000000	2744.000000	2938.000000	2385.00000
mean	2007.518720	69.224932	164.796448	30.303948	4.602861	738.251295	80.94046
std	4.613841	9.523867	124.292079	117.926501	4.052413	1987.914858	25.07001
min	2000.000000	36.300000	1.000000	0.000000	0.010000	0.000000	1.00000
25%	2004.000000	63.100000	74.000000	0.000000	0.877500	4.685343	77.00000
50%	2008.000000	72.100000	144.000000	3.000000	3.755000	64.912906	92.00000
75%	2012.000000	75.700000	228.000000	22.000000	7.702500	441.534144	97.00000
max	2015.000000	89.000000	723.000000	1800.000000	17.870000	19479.911610	99.00000

CHECKING FOR NULL VALUE ROWS

In [5]:
```
df.isna().sum()
```

Out[5]:
```
Country                            0
Year                               0
Status                             0
Life expectancy                   10
Adult Mortality                   10
infant deaths                      0
Alcohol                          194
percentage expenditure             0
Hepatitis B                      553
Measles                            0
 BMI                              34
under-five deaths                  0
Polio                             19
Total expenditure                226
Diphtheria                        19
 HIV/AIDS                          0
GDP                              448
Population                       652
 thinness  1-19 years             34
 thinness 5-9 years               34
Income composition of resources  167
Schooling                        163
dtype: int64
```

DROPPING THE REQUIRED ROWS WITH NULL VALUES

In [6]:
```
df.drop(axis='rows', labels=df.index[df["Life expectancy "].isna()], inplace=True
df.drop(axis='rows', labels=df.index[df["Income composition of resources"].isna()
df.drop(axis='rows', labels=df.index[df["Schooling"].isna()], inplace=True)
```

In [7]:
```
df.isna().sum()
```

Out[7]:
```
Country                            0
Year                               0
Status                             0
Life expectancy                    0
Adult Mortality                    0
infant deaths                      0
Alcohol                          184
percentage expenditure             0
Hepatitis B                      509
Measles                            0
 BMI                              32
under-five deaths                  0
Polio                             19
Total expenditure                186
Diphtheria                        19
 HIV/AIDS                          0
GDP                              286
Population                       484
 thinness  1-19 years             32
 thinness 5-9 years               32
Income composition of resources    0
Schooling                          0
dtype: int64
```

In [8]:
```
df
```

Out[8]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0
...	...	...	...	...	...	...	...	...	...
2933	Zimbabwe	2004	Developing	44.3	723.0	27	4.36	0.000000	68.0
2934	Zimbabwe	2003	Developing	44.5	715.0	26	4.06	0.000000	7.0
2935	Zimbabwe	2002	Developing	44.8	73.0	25	4.43	0.000000	73.0
2936	Zimbabwe	2001	Developing	45.3	686.0	25	1.72	0.000000	76.0
2937	Zimbabwe	2000	Developing	46.0	665.0	24	1.68	0.000000	79.0

2768 rows × 22 columns

In [9]:
```
df.shape
```

Out[9]:
```
(2768, 22)
```

In [10]:
```
df = df.drop(['Country'], axis=1)
df = df.drop(['Status'], axis=1)
```

In [11]:
```
df.shape
```

Out[11]:
```
(2768, 20)
```

RENAMING COLUMN NAMES

In [12]:
```
df.rename(columns={'Income composition of resources':'income', 'Life expectancy
```

CHECKING THE CORRELATION OF FEATURES

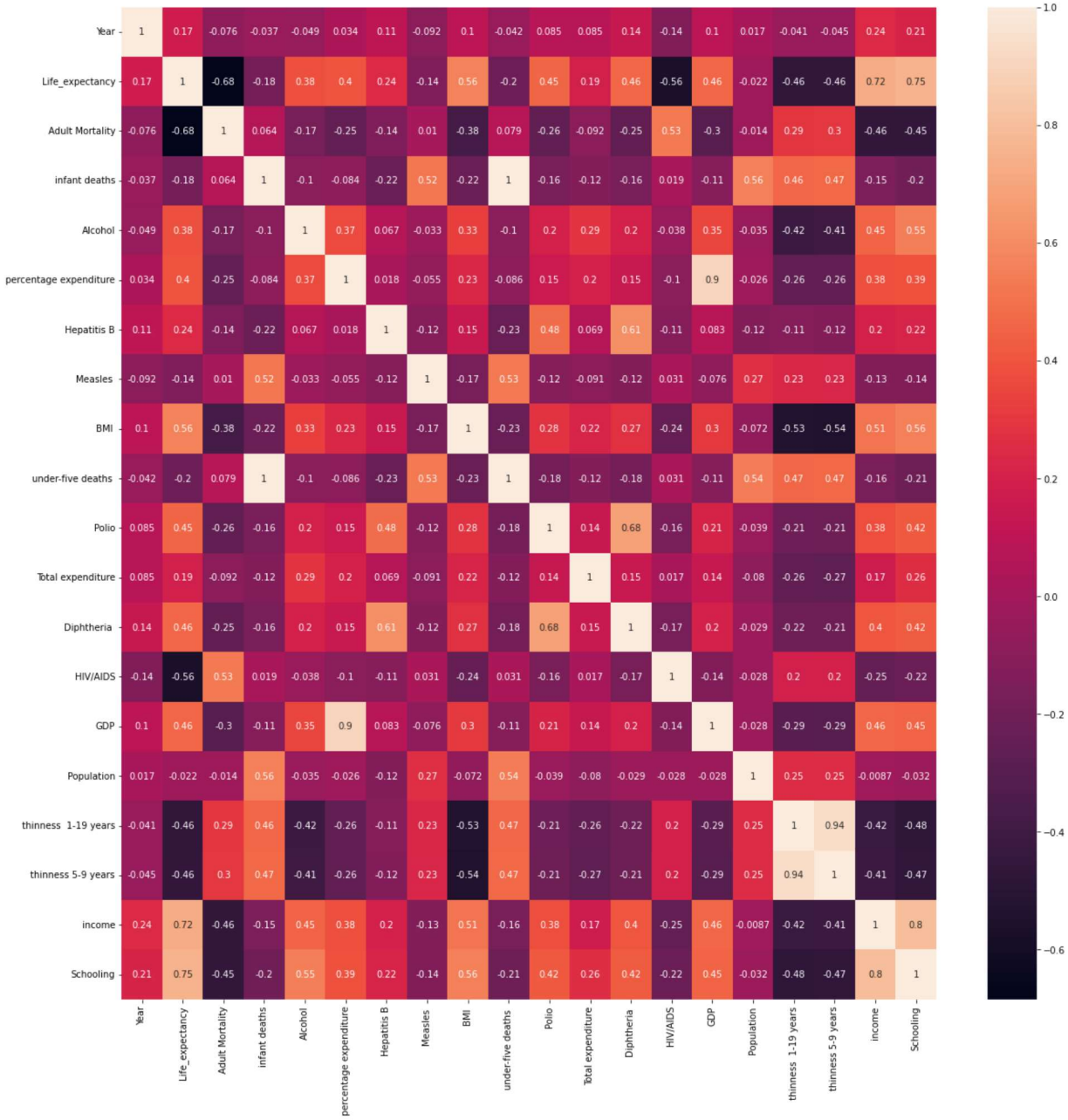
In [13]:
```
correlation = df.corr()
correlation
```

Out[13]:

	Year	Life_expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B
Year	1.000000	0.171042	-0.075688	-0.037298	-0.049335	0.033790	0.110293
Life_expectancy	0.171042	1.000000	-0.684585	-0.179548	0.381469	0.396084	0.240207
Adult Mortality	-0.075688	-0.684585	1.000000	0.063906	-0.171503	-0.248033	-0.143838
infant deaths	-0.037298	-0.179548	0.063906	1.000000	-0.104269	-0.083809	-0.217538
Alcohol	-0.049335	0.381469	-0.171503	-0.104269	1.000000	0.365709	0.066802
percentage expenditure	0.033790	0.396084	-0.248033	-0.083809	0.365709	1.000000	0.017548
Hepatitis B	0.110293	0.240207	-0.143838	-0.217538	0.066802	0.017548	1.000000
Measles	-0.092102	-0.141217	0.010355	0.521308	-0.032547	-0.055398	-0.116052
BMI	0.101722	0.563736	-0.375359	-0.219385	0.325384	0.234671	0.148289
under-five deaths	-0.042362	-0.204858	0.078506	0.996618	-0.099810	-0.085891	-0.227033
Polio	0.084771	0.454795	-0.260643	-0.162037	0.201996	0.151327	0.484866
Total expenditure	0.084972	0.193112	-0.091961	-0.124206	0.287211	0.199731	0.069019
Diphtheria	0.139937	0.464119	-0.252511	-0.163022	0.201957	0.145119	0.606706
HIV/AIDS	-0.136833	-0.563175	0.529330	0.018823	-0.038350	-0.100403	-0.107844
GDP	0.103709	0.461250	-0.295616	-0.108568	0.354148	0.899350	0.082908
Population	0.017199	-0.021538	-0.013647	0.556781	-0.035376	-0.025716	-0.123834
thinness 1-19 years	-0.040567	-0.464487	0.289868	0.463872	-0.417539	-0.260087	-0.113541
thinness 5-9 years	-0.044539	-0.459313	0.296367	0.470227	-0.405254	-0.262239	-0.121098
income	0.242953	0.724776	-0.457626	-0.145018	0.450254	0.382244	0.199141
Schooling	0.213265	0.751975	-0.454612	-0.195202	0.548001	0.391466	0.222895

In [14]:
```
plt.figure(figsize=(20,20))
sn.heatmap(correlation, annot = True)

plt.show()
```



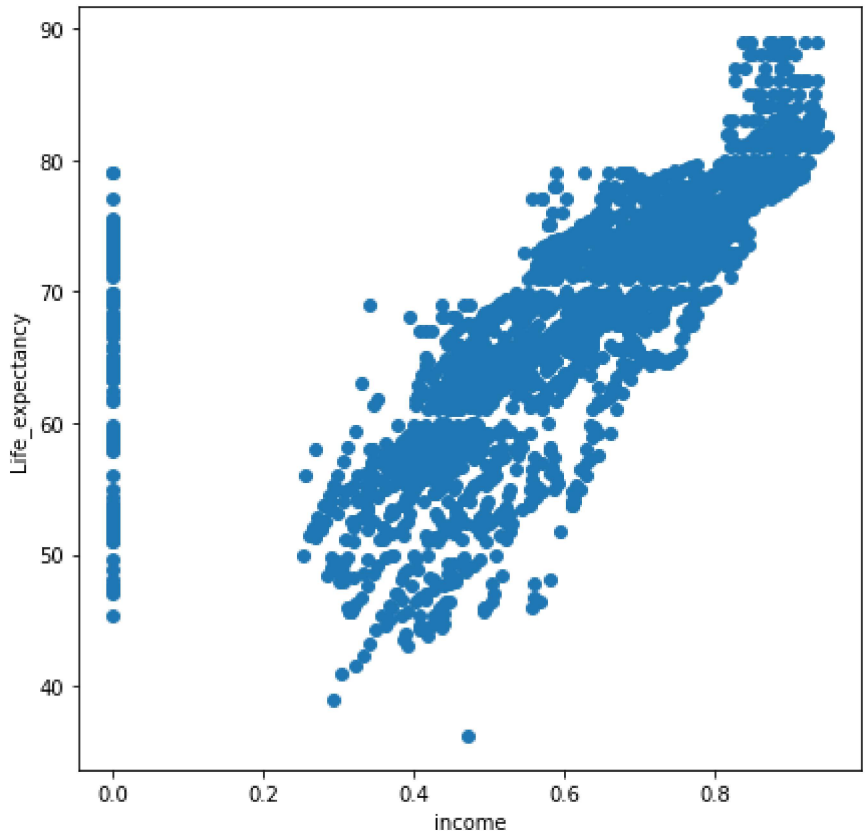
In [15]:
```
df = df
```

DROPPING UNRELATED COLUMNS

In [16]:
```
df = df.drop(['Year'], axis=1)
df = df.drop(['infant deaths'], axis=1)
df = df.drop(['Population'], axis=1)
```

In [17]:
```
plt.figure(figsize=(7,7))
plt.xlabel('income')
plt.ylabel('Life_expectancy')
plt.scatter(df.income,df.Life_expectancy)
```

Out[17]: <matplotlib.collections.PathCollection at 0x1d06c716e80>

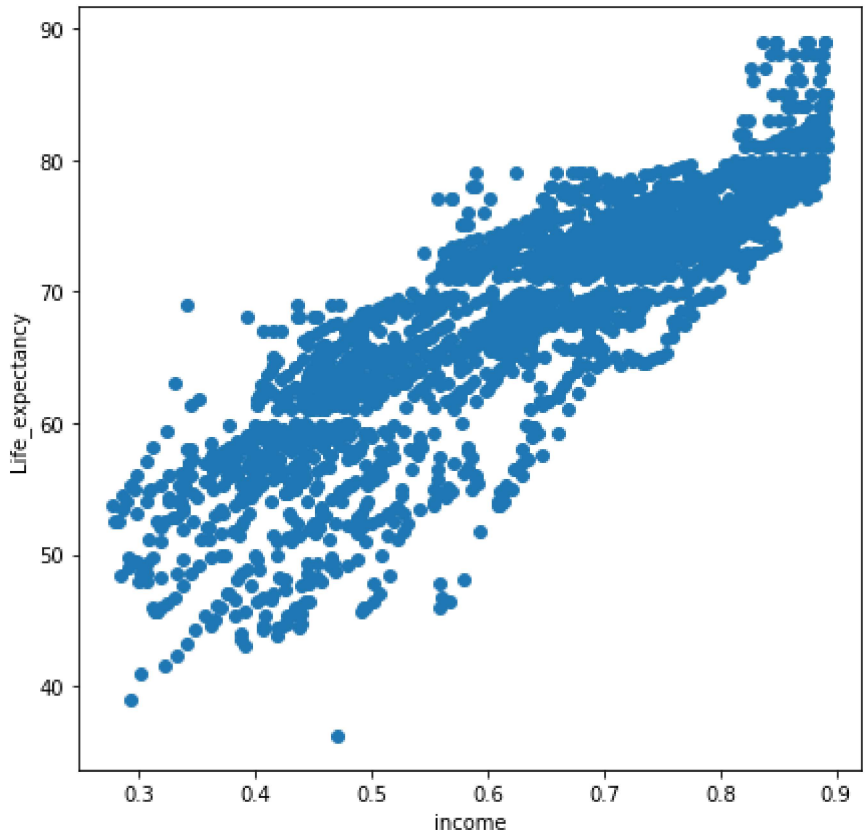


REMOVING OUTLIERS

In [18]:
```
max_threshold=df['income'].quantile(0.95)
min_threshold=df['income'].quantile(0.05)
df=df[(df['income']>min_threshold) & (df['income']<max_threshold)]
```

In [19]:
```
plt.figure(figsize=(7,7))
plt.xlabel('income')
plt.ylabel('Life_expectancy')
plt.scatter(df.income,df.Life_expectancy)
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x1d06bbb7910>
```



SPLITTING TRAIN AND TEST DATA

In [20]:
```
train = df.iloc[:1800,:]
test = df.iloc[1800:,:]
```

EXTRACTING THE INPUT AND TARGET VARIRABLES FROM TEST AND TRAIN

In [21]:
```
x_train = train[['income','Schooling']]
y_train = train['Life_expectancy']
x_test = test[['income','Schooling']]
y_test = test['Life_expectancy']
```

In [22]:
```
ones = np.ones((x_train.shape[0],1))
x_train = np.hstack((ones,x_train))
ones = np.ones((x_test.shape[0],1))
x_test = np.hstack((ones,x_test))
```

In [23]:
```
x_test.shape
```

Out[23]: (688, 3)

In [24]:
```
x_train.shape
```

Out[24]: (1800, 3)

BUILDING LINEAR MULTIVARIATE MODEL

In [25]:
```
def model(x,y,l=0.01,itr=5000):
    n=x.shape[1]
    theta=np.ones(n)
    err_list=[]

    for i in range (itr):
        h=np.dot(x,theta)
        err = h-y
        #cost function
        cost=(1/x.shape[0])*np.sum(np.square(err))
        err_list.append(cost)
        diff_cost = (np.dot(x.T,err)/x.shape[0])  #differentiated cost
        #theta = theta - learning rate * gradient
        theta=theta-l*diff_cost

    return theta,err_list,cost
```

CALLING THE MODEL FUNCTION BY PASSING THE PARAMETERS

In [26]:
```
theta,err_list,cost=model(x_train,y_train)
```

FINAL PARAMETERS

In [27]:
```
theta
```

Out[27]: array([38.7856549 , 14.1351858 , 1.77039865])

FINAL COST

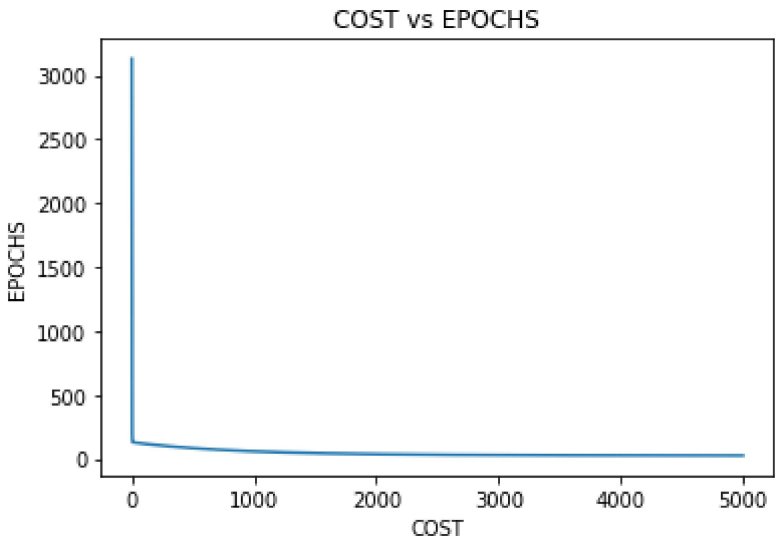
In [28]:
```
err_list[-1]
```

Out[28]:
```
26.069775955388444
```

ERROR PLOT

In [29]:
```
plt.title('COST vs EPOCHS')
plt.xlabel('COST')
plt.ylabel('EPOCHS')
plt.plot(err_list)
```

Out[29]:
```
[<matplotlib.lines.Line2D at 0x1d06be32790>]
```



In [30]:
```
err_list
```

Out[30]:
```
[3130.5907629594444,
 954.3467739574612,
 356.2355957499227,
 191.78796126262242,
 146.50912717803322,
 133.97742492811406,
 130.44458848534717,
 129.38470650107442,
 129.00448925987519,
 128.8111490939916,
 128.66927014934672,
 128.54164065666077,
 128.41803486497983,
 128.29564265375035,
 128.17369168275215,
 128.05196958432794,
 127.93041786879515,
 127.80902033157336,
 127.68777238799005,
```

In [31]:
```
y_pred = np.dot(x_test,theta)
mse = np.square(np.subtract(y_pred,y_test)).mean()
mae = np.absolute(np.subtract(y_pred,y_test)).mean()
```

In [32]:
```
y_pred
```

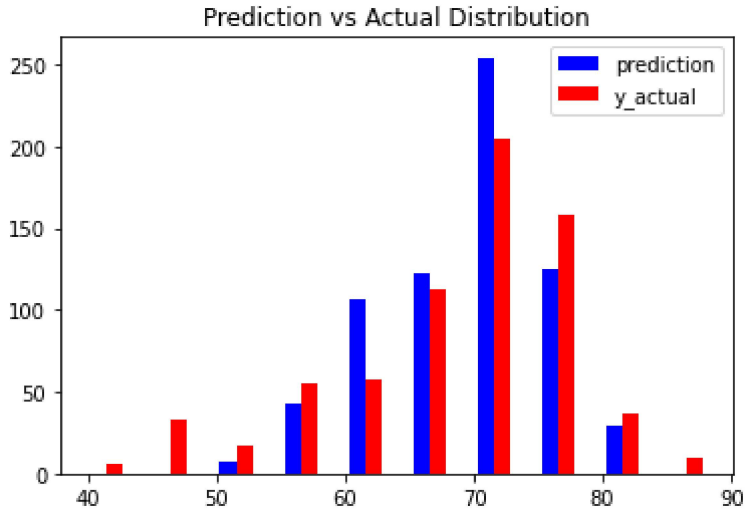
Out[32]:
```
array([74.84299649, 74.77232056, 74.36170009, 72.96693399, 71.79161332,
       72.52804315, 73.22206742, 76.09039326, 76.07625807, 76.03385252,
       77.13849726, 77.86079191, 78.02369658, 77.28726675, 75.83595992,
       74.57582813, 73.71218163, 73.03971017, 71.98488862, 71.12124212,
       70.12296131, 69.5070306 , 69.08227494, 76.72045915, 76.51514892,
       75.92748858, 75.29742269, 74.66735679, 73.96661497, 74.54014012,
       74.26415396, 73.9740326 , 73.8750863 , 73.61323532, 73.33724916,
       72.51600821, 71.72303764, 71.09297175, 70.22932525, 64.87460687,
       64.80393094, 64.76152539, 64.08905394, 63.4024473 , 62.90701571,
       63.16214914, 63.3890122 , 62.66      , 60.96097737, 59.84961507,
       58.54707773, 57.31521631, 56.38089388, 56.04836697, 54.96527505,
       72.36723872, 72.19761649, 72.35310354, 72.19019886, 71.98488862,
       71.68063208, 71.40464592, 71.12865975, 71.44033393, 70.54841705,
       70.12366139, 70.2582957 , 70.40706519, 70.5416995 , 70.76114492,
       72.50929066, 72.50929066, 72.46688511, 72.41034436, 72.39620918,
       72.38207399, 72.35380362, 72.28312769, 72.25485732, 72.15591102,
       72.11350546, 71.89406004, 71.44103401, 71.19331822, 70.95973761,
       71.54669786, 71.53256268, 71.51842749, 71.49015712, 71.41948119,
       71.37707563, 71.39121082, 71.34880526, 71.26399415, 70.83923849,
```

In [33]:
```
y_test
```

Out[33]:
```
2080    76.6
2081    76.6
2082    76.6
2083    76.5
2084    76.4
        ...
2933    44.3
2934    44.5
2935    44.8
2936    45.3
2937    46.0
Name: Life_expectancy, Length: 688, dtype: float64
```

PREDICTION vs ACTUAL DISTRIBUTION

In [34]:
```
y_and_ypred = [y_pred, y_test]
a = plt.subplot()
labels = ['prediction', 'y_actual']
colors = ['blue','red']
a.hist(y_and_ypred, bins = 10, rwidth=0.5,label=labels, color = colors)
a.legend(prop={'size': 10})
plt.title(label="Prediction vs Actual Distribution")
plt.show()
```



ERROR METRICS

In [35]:
```
print("MEAN SQUARED ERROR : ",mse)
print("MEAN ABSOLUTE ERROR : ",mae)
```

```
MEAN SQUARED ERROR :  35.06152562368472
MEAN ABSOLUTE ERROR :  4.186000706316932
```

CLOSED FORM

In [41]:
```
def normal_eq(X, Y):
    w = np.dot((np.linalg.inv(np.dot(X.T,X))),np.dot(X.T,Y))
    return w
```

In [42]:
```
theta_closed = normal_eq(x_train,y_train)
```

PARAMETERS OF CLOSED FORM

In [43]:
```
theta_closed
```

Out[43]:
```
array([36.63893343, 63.12822898, -0.69711119])
```

ERROR METRICS FOR CLOSED FORM

In [44]:
```
y_pred_closed = np.dot(x_test,theta_closed)
mse_closed = np.square(np.subtract(y_pred_closed,y_test)).mean()
mae_closed = np.absolute(np.subtract(y_pred_closed,y_test)).mean()
```

In [45]:
```
print("MEAN SQUARED ERROR IN CLOSED FORM : ",mse_closed)
print("MEAN ABSOLUTE ERROR IN CLOSED FORM : ",mae_closed)
```

```
MEAN SQUARED ERROR :  21.148842346679764
MEAN ABSOLUTE ERROR :  3.4288829518905097
```

LOGISTIC UNIVARIATE

IMPORTING HEADER FILES

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
df = pd.read_csv("diabetes.csv")
df
```

Out[1]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288
...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171
764	2	122	70	27	0	36.8	0.340
765	5	121	72	23	112	26.2	0.245
766	1	126	60	0	0	30.1	0.349
767	1	93	70	31	0	30.4	0.315

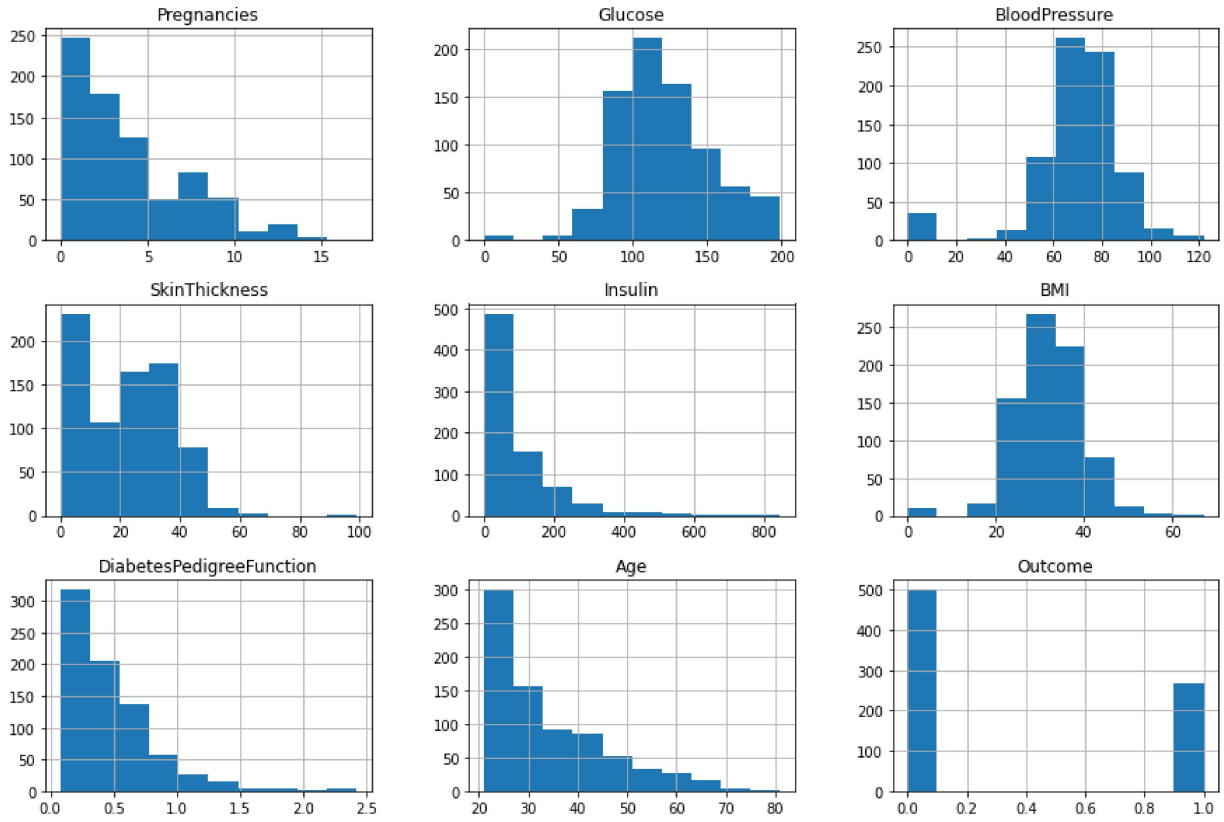
768 rows × 9 columns

In [2]:
```
df.shape
```

Out[2]:
```
(768, 9)
```

ANALYSING THE DATA

In [3]:
```
df.hist(figsize=(15,10))
plt.show()
```



In [4]:
```
df.describe()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

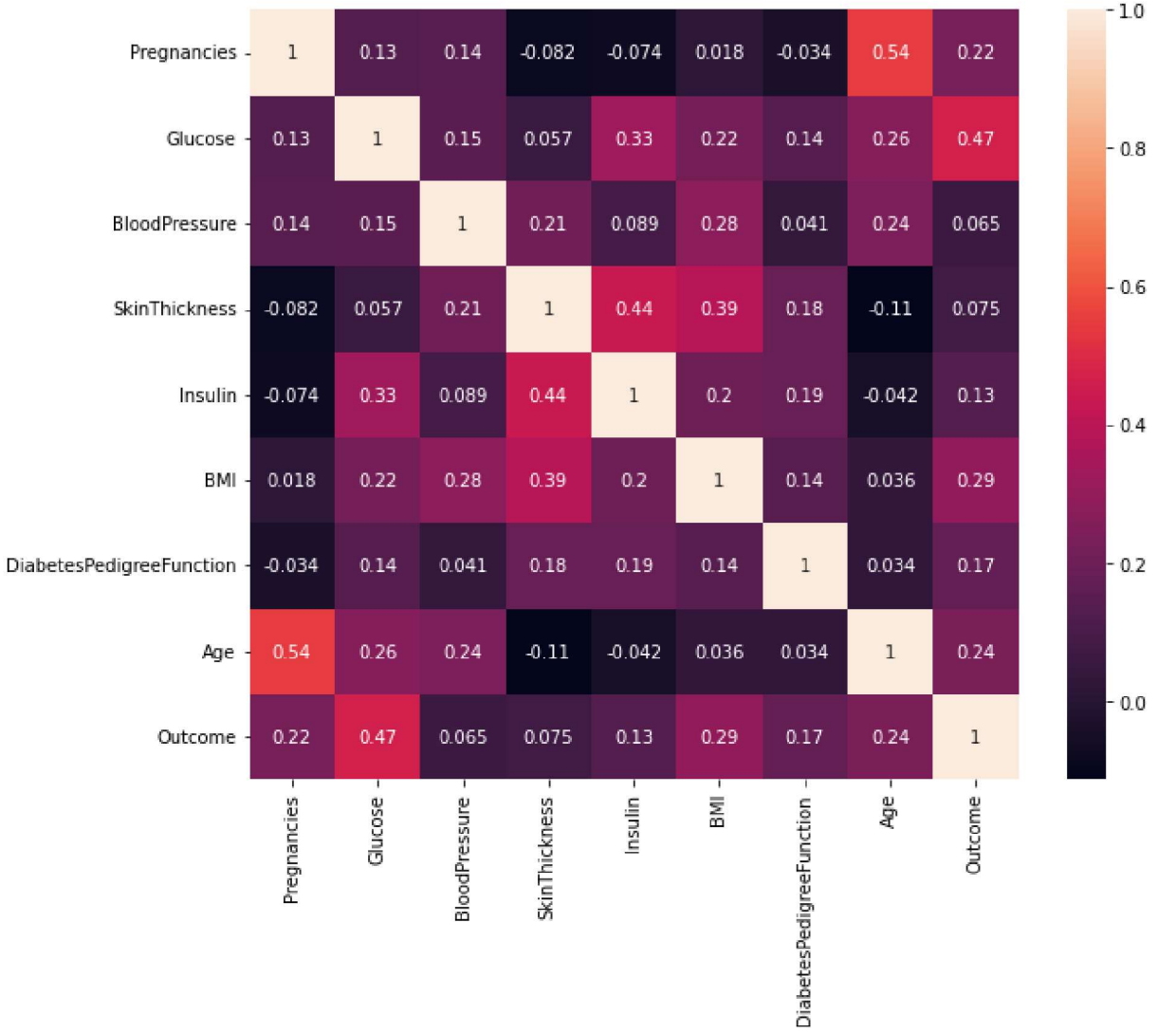
CHECKING THE CORRELATION OF DATA WITH THE TARGET

In [5]:
```
correlation = df.corr()
correlation
```

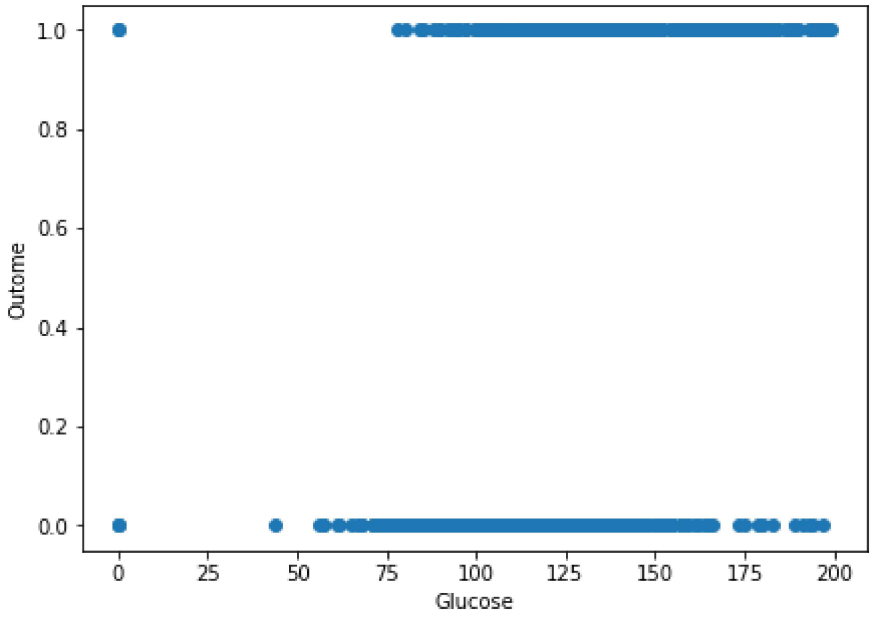
Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BI
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.01768
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.22107
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.28180
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.39257
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.19785
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.00000
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.14064
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.03624
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.29269

In [6]:
```
plt.figure(figsize=(10,8))
sn.heatmap(correlation, annot=True, square=True)
plt.show()
```



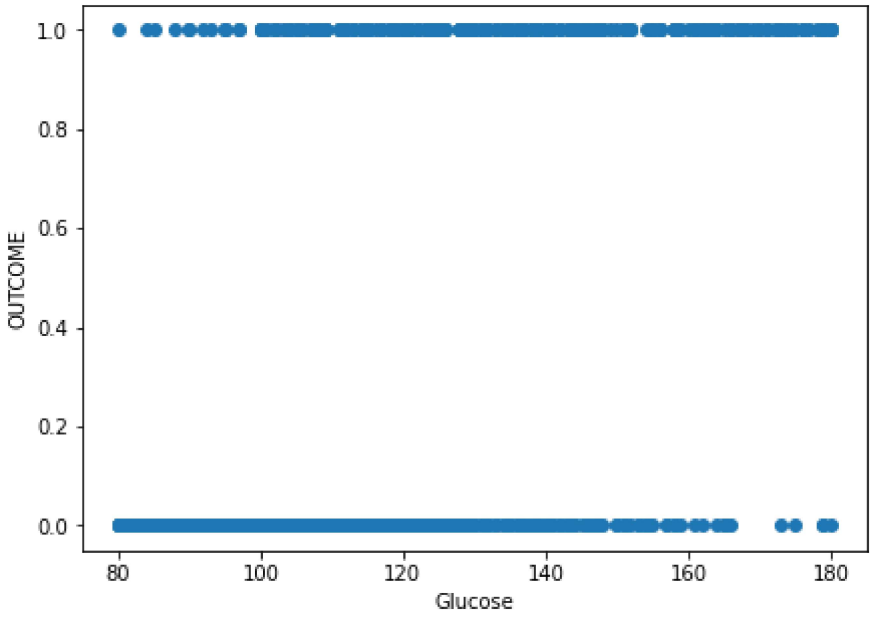
In [7]:
```
plt.figure(figsize=(7,5))
plt.xlabel('Glucose')
plt.ylabel('Outome')
plt.scatter(df.Glucose,df.Outcome)
plt.show()
```



REMOVING THE OUTLIERS

In [8]:
```
max_threshold=df['Glucose'].quantile(0.95)
min_threshold=df['Glucose'].quantile(0.05)
df=df[(df['Glucose']>min_threshold) & (df['Glucose']<max_threshold)]
```

In [9]:
```
plt.figure(figsize=(7,5))
plt.xlabel('Glucose')
plt.ylabel('OUTCOME')
plt.scatter(df.Glucose,df.Outcome)
plt.show()
```



In [10]:
```
df.shape
```

Out[10]: (686, 9)

SEPARTING THE TARGET

In [11]:
```
target=df[['Outcome']]
target
```

Out[11]:

	Outcome
0	1
1	0
3	0
4	1
5	0
...	...
763	0
764	0
765	0
766	1
767	0

686 rows × 1 columns

In [12]:
```
target.shape
```

Out[12]:
```
(686, 1)
```

In [13]:
```
df = df.drop(['Outcome'],axis=1)
df
```

Out[13]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288
5	5	116	74	0	0	25.6	0.201
...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171
764	2	122	70	27	0	36.8	0.340
765	5	121	72	23	112	26.2	0.245
766	1	126	60	0	0	30.1	0.349
767	1	93	70	31	0	30.4	0.315

686 rows × 8 columns

NORMALISING THE DATA

In [14]:
```
df=(df-df.mean())/df.std()
df
```

Out[14]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
0	0.640961	1.112488	0.149974	0.928586	-0.729675	0.200721	
1	-0.831575	-1.418545	-0.158518	0.547105	-0.729675	-0.676795	-
3	-0.831575	-1.257845	-0.158518	0.165624	0.156434	-0.488756	-
4	-1.126083	0.670562	-1.495318	0.928586	0.854009	1.391637	
5	0.346454	-0.173116	0.252805	-1.296721	-0.729675	-0.802155	-
...	...	...	...	...	...	...	
763	1.818991	-0.775743	0.355635	1.755129	0.967129	0.112970	
764	-0.537068	0.067935	0.047143	0.419945	-0.729675	0.601872	-
765	0.346454	0.027759	0.149974	0.165624	0.326114	-0.726939	-
766	-0.831575	0.228635	-0.467010	-1.296721	-0.729675	-0.238037	-
767	-0.831575	-1.097144	0.047143	0.674265	-0.729675	-0.200429	-

686 rows × 8 columns

TEST AND TRAIN SPLIT

In [15]:
```
train = df.iloc[:500,:]
test = df.iloc[500:,:]
target_train = target.iloc[:500,:]
target_test = target.iloc[500:,:]
```

In [16]:
```
target_train.shape
```

Out[16]: (500, 1)

In [17]:
```
target_test.shape
```

Out[17]: (186, 1)

EXTRACTING THE INPUT FEATURES AND TARGET FEATURE

In [18]:
```
x_train = train[['Glucose']]
y_train = target_train
x_test = test[['Glucose']]
y_test = target_test
```

In [19]:
```
x_train.shape
```

Out[19]: (500, 1)

In [20]:
```
y_train.shape
```

Out[20]: (500, 1)

In [21]:
```
x_test.shape
```

Out[21]: (186, 1)

In [22]:
```
y_test.shape
```

Out[22]: (186, 1)

DEFINING THE SIGMOID FUNCTION

In [23]:
```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

BUILDING THE LOGISTIC MULTIVARIATE MODEL

In [24]:
```
def model_log_uni(x,y,l=0.01,itr=10000):
        n = x.shape[0]                                      #rows (no. of samples)
        print("No. of rows in x : ", n)

        m = x.shape[1]                                      #columns (no. of feature
        print("No. of columns in x : ", m)

        x = np.hstack((np.ones((n,1)), x))                  #putting a column of all
        print("Updated Shape of x after adding a column of 1's: ",x.shape)

        theta = np.zeros((m+1,1))                           #increasing the size of
        print("Shape of theta : ",theta.shape)

        cost_list = []

        for i in range(itr):

            lines = np.dot(x,theta)
            ypred = sigmoid(lines)                          #converting the linear f
            error = ypred-y
                                                            #cost function
            cost = -(1/n)* np.sum(y*np.log(ypred) +(1-y)*np.log(1-ypred))

            theta = theta - l*((1/n) * np.dot(x.T,error))

            cost_list.append(cost)

        return theta,cost_list
```

CALLING THE LOGISTIC MODEL FUNCTION

In [25]:
```
theta,cost_list = model_log_uni(x_train,y_train)
```

```
No. of rows in x :  500
No. of columns in x :  1
Updated Shape of x after adding a column of 1's:  (500, 2)
Shape of theta :  (2, 1)
```

FINAL PARAMETERS

In [26]:
```
print("Final parameters after all the iterations of gradient descent :\n ",theta)
```

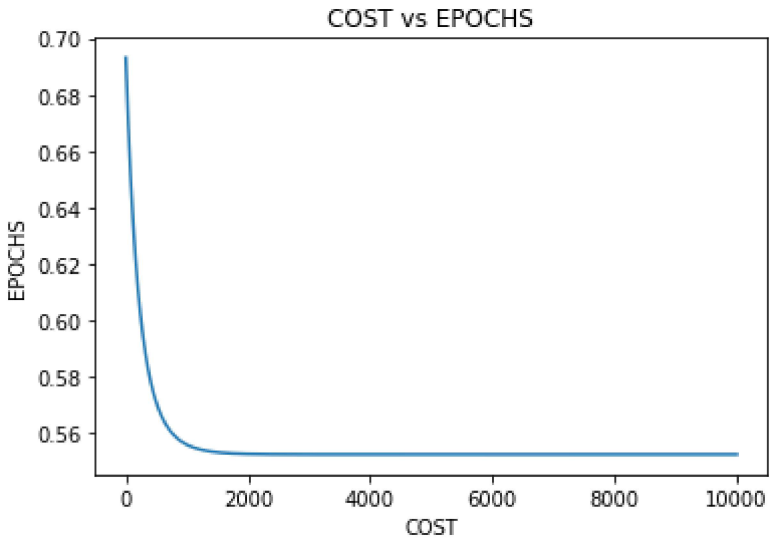
```
Final parameters after all the iterations of gradient descent :
  [[-0.77704266]
 [ 0.94396829]]
```

In [27]:
```
print("Final Cost : ",cost_list[-1])
```

```
Final Cost :  Outcome    0.552389
dtype: float64
```

ERROR PLOT

In [28]:
```
plt.title('COST vs EPOCHS')
plt.ylabel('EPOCHS')
plt.xlabel('COST')
plt.plot(cost_list)
plt.show()
```



CLASSIFYING THE PREDICTION >= 0.5 TO '1' AND <0.5 TO '0'

In [29]:
```
def pred_class(X, theta):
    prediction = []
    lines = np.dot(X,theta)
    y_pred = sigmoid(lines)
    for i in y_pred:
        if i>=0.5:
            prediction.append(1)
        else:
            prediction.append(0)
    return prediction
```

MAKING THE FIRST COLUMN OF 'x_test' AS ALL 1's

In [30]:
```
x_test = np.hstack((np.ones((x_test.shape[0],1)), x_test))
```

In [31]:
```
x_test.shape
```

Out[31]:
```
(186, 2)
```

In [32]:
```
y_pred = pred_class(x_test, theta)
```

PREDICTION ON x_test =>y_pred

In [33]:
```
y_pred
```

Out[33]:
```
[0,
 0,
 0,
 0,
 1,
 0,
 0,
 0,
 0,
 1,
 0,
 0,
 0,
 0,
 1,
 0,
 0,
 0,
 0,
```

In [34]:
```
print("TOP 10 ROWS OF PREDICTIONS : ")
y_test.head(10)
```

```
TOP 10 ROWS OF PREDICTIONS :
```

Out[34]:

	Outcome
564	0
565	0
566	0
567	0
568	0
569	1
571	0
572	0
573	0
574	0

EVALUATION METRICS

CHECKING ACCURACY

In [35]:
```
def accuracy_1(X, Y, theta):

    lines_1 = np.dot(X,theta)
    y_pred_1 = sigmoid(lines_1)
    y_pred_1 = y_pred_1 > 0.5

    y_pred_1 = np.array(y_pred_1, dtype='int64')
    acc = (1-np.sum(np.absolute(y_pred_1-Y))/Y.shape[0])*100

    print("Accuracy of the model is : ", round(acc,2), "%")
    return y_pred_1
```

In [36]:
```
A = accuracy_1(x_test, y_test, theta)
```

```
Accuracy of the model is :  Outcome    76.88
dtype: float64 %
```

In [37]:
```
A
```

Out[37]:
```
array([[0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
```

CHECKING F1-SCORE

In [38]:
```
from sklearn.metrics import f1_score
F1_Score = f1_score(y_test, y_pred,average = 'micro')
print("F1 Score of the model is : ",F1_Score)
```

```
F1 Score of the model is :  0.7688172043010751
```

LOGISTIC MULTIVARIATE

IMPORTING HEADER FILES

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
from collections import Counter
df = pd.read_csv("diabetes.csv")
df
```

Out[1]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288
...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171
764	2	122	70	27	0	36.8	0.340
765	5	121	72	23	112	26.2	0.245
766	1	126	60	0	0	30.1	0.349
767	1	93	70	31	0	30.4	0.315

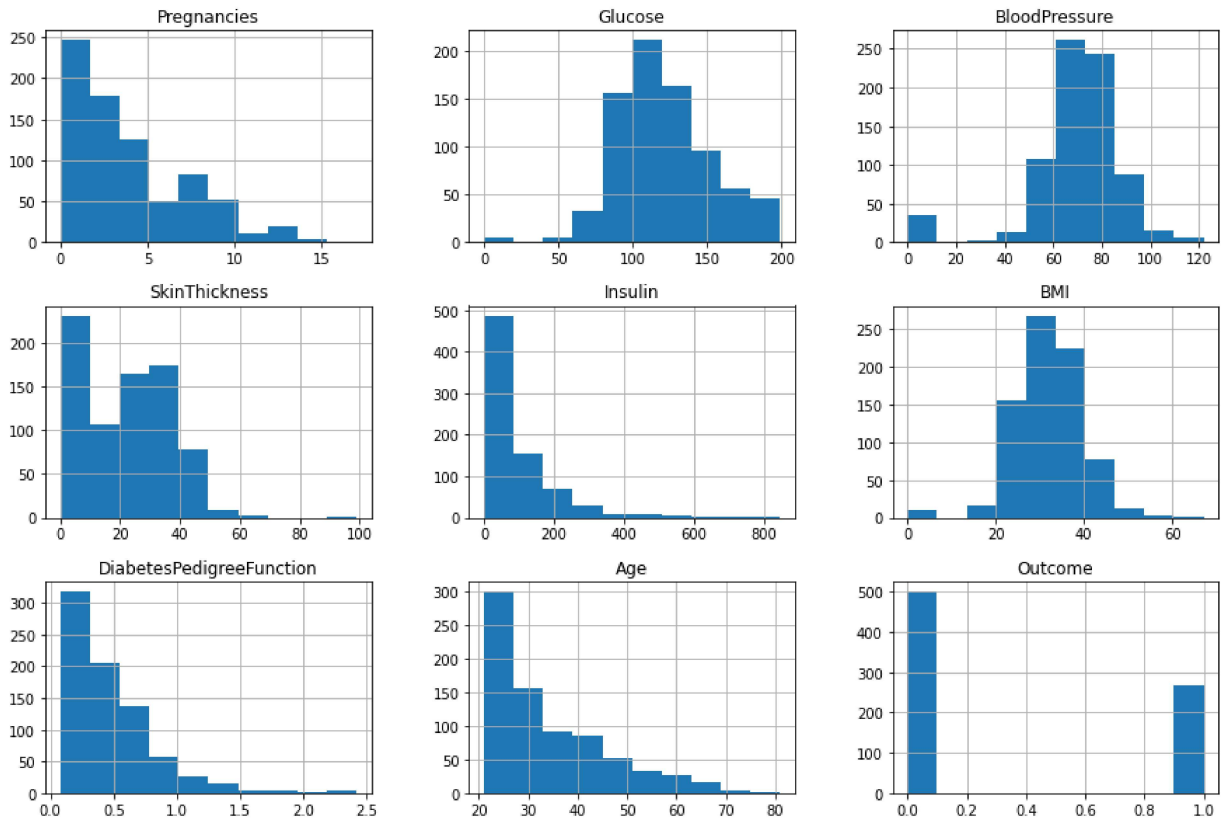
768 rows × 9 columns

In [2]:
```
df.shape
```

Out[2]:
```
(768, 9)
```

ANALYSING THE DATA

In [3]:
```
df.hist(figsize=(15,10))
plt.show()
```



In [4]:
```
df.describe()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

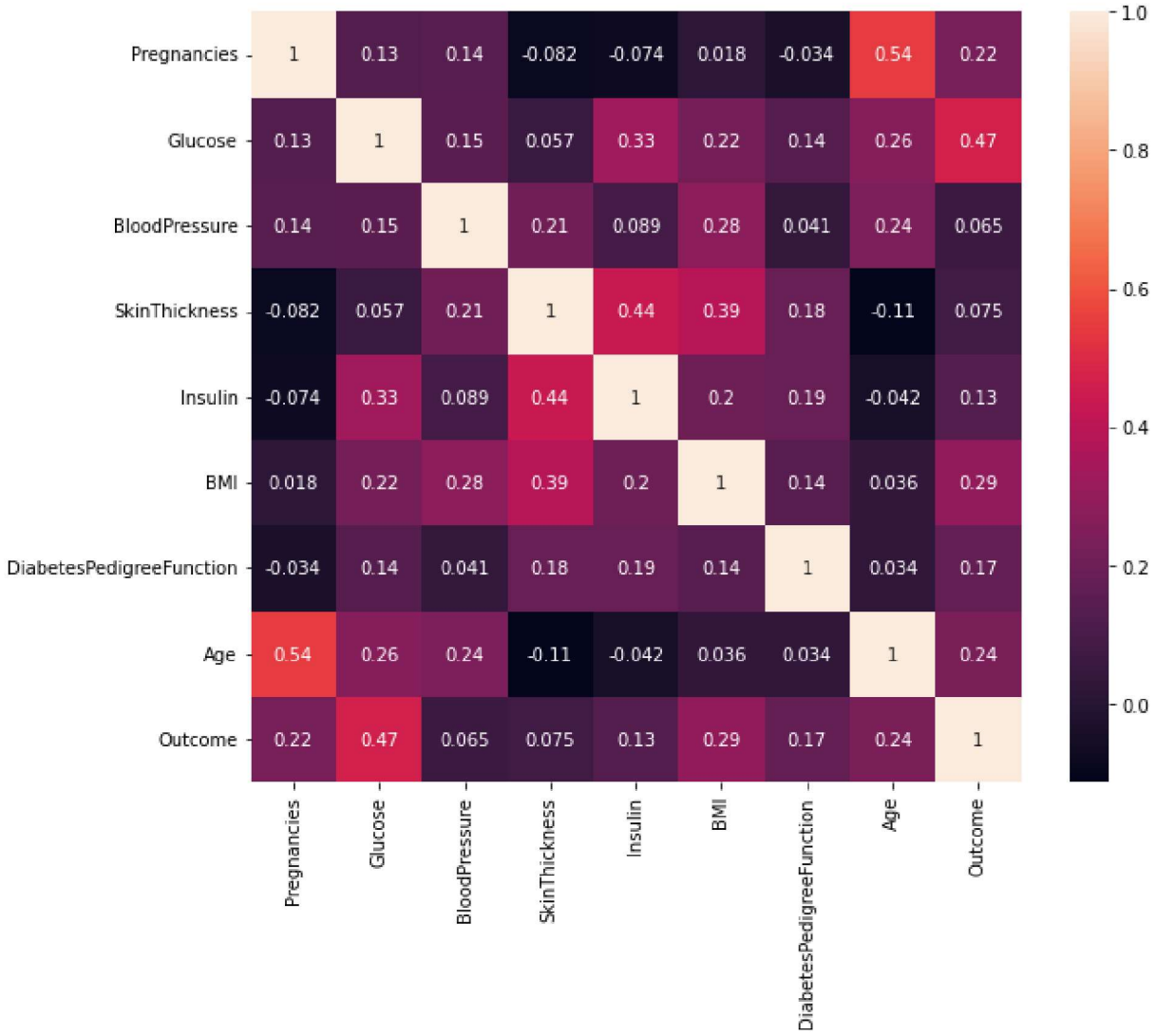
CHECKING THE CORRELATION

In [5]:
```
correlation = df.corr()
correlation
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BI
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.01768
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.22107
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.28180
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.39257
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.19785
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.00000
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.14064
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.03624
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.29269

In [6]:
```
plt.figure(figsize=(10,8))
sn.heatmap(correlation, annot=True, square=True)
plt.show()
```



SEPARATING THE TARGET BEFORE NORMALISING THE DATA

In [7]:
```
tgt = df[['Outcome']]
```

In [8]:
```
tgt
```

Out[8]:

	Outcome
0	1
1	0
2	1
3	0
4	1
...	...
763	0
764	0
765	0
766	1
767	0

768 rows × 1 columns

In [9]:
```
tgt.shape
```

Out[9]:
```
(768, 1)
```

In [10]:
```
df = df.drop(['Outcome'],axis=1)
df
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288
...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171
764	2	122	70	27	0	36.8	0.340
765	5	121	72	23	112	26.2	0.245
766	1	126	60	0	0	30.1	0.349
767	1	93	70	31	0	30.4	0.315

768 rows × 8 columns

NORMALISING THE DATA

In [11]:
```
df=(df-df.mean())/df.std()
df
```

Out[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
0	0.639530	0.847771	0.149543	0.906679	-0.692439	0.203880	
1	-0.844335	-1.122665	-0.160441	0.530556	-0.692439	-0.683976	-
2	1.233077	1.942458	-0.263769	-1.287373	-0.692439	-1.102537	
3	-0.844335	-0.997558	-0.160441	0.154433	0.123221	-0.493721	-
4	-1.141108	0.503727	-1.503707	0.906679	0.765337	1.408828	
...	...	...	...	...	...	...	
763	1.826623	-0.622237	0.356200	1.721613	0.869464	0.115094	-
764	-0.547562	0.034575	0.046215	0.405181	-0.692439	0.609757	-
765	0.342757	0.003299	0.149543	0.154433	0.279412	-0.734711	-
766	-0.844335	0.159683	-0.470426	-1.287373	-0.692439	-0.240048	-
767	-0.844335	-0.872451	0.046215	0.655930	-0.692439	-0.201997	-

768 rows × 8 columns

TEST AND TRAIN SPLIT

In [12]:
```
train = df.iloc[:500,:]
test = df.iloc[500:,:]
tgt_train = tgt.iloc[:500,:]
tgt_test = tgt.iloc[500:,:]
```

In [13]:
```
tgt_train.shape
```

Out[13]: (500, 1)

In [14]:
```
tgt_test.shape
```

Out[14]: (268, 1)

EXTRACTING THE INPUT FEATURES AND TARGET FEATURE

In [15]:
```
x_train = train[['Pregnancies','Glucose','BMI','DiabetesPedigreeFunction','Age']]
y_train = tgt_train
x_test = test[['Pregnancies','Glucose','BMI','DiabetesPedigreeFunction','Age']]
y_test = tgt_test
```

In [16]:
```
x_train.shape
```

Out[16]: (500, 5)

In [17]:
```
y_train.shape
```

Out[17]: (500, 1)

In [18]:
```
x_test.shape
```

Out[18]: (268, 5)

In [19]:
```
y_test.shape
```

Out[19]: (268, 1)

DEFINING THE SIGMOID FUNCTION

In [20]:
```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

BUILDING THE LOGISTIC MULTIVARIATE MODEL

In [21]:
```
def model_log_mul(x,y,l=0.01,itr=10000):            #l=learning rate(alpha)

    n = x.shape[0]                                  #rows (no. of samples)
    print("No. of rows in x : ", n)

    m = x.shape[1]                                  #columns (no. of features)
    print("No. of columns in x : ", m)

    x = np.hstack((np.ones((n,1)), x))              #putting a column of all one
    print("Updated Shape of x after adding a column of 1's: ",x.shape)

    theta = np.zeros((m+1,1))                       #increasing the size of para
    print("Shape of theta : ",theta.shape)

    cost_list = []

    for i in range(itr):

        lines = np.dot(x,theta)
        ypred = sigmoid(lines)                      #converting the linear funct
        error = ypred-y

        cost = -(1/n)* np.sum(y*np.log(ypred) +(1-y)*np.log(1-ypred))

        theta = theta - l*((1/n) * np.dot(x.T,error))

        cost_list.append(cost)

    return theta,cost_list
```

CALLING THE LOGISTIC MODEL FUNCTION

In [22]:
```
theta,cost_list = model_log_mul(x_train,y_train)
```

```
No. of rows in x :  500
No. of columns in x :  5
Updated Shape of x after adding a column of 1's:  (500, 6)
Shape of theta :  (6, 1)
```

FINAL PARAMETERS

In [23]:
```
print("Final parameters after all the iterations of gradient descent :\n ",theta)
```

```
Final parameters after all the iterations of gradient descent :
  [[-0.78885872]
 [ 0.38595702]
 [ 0.94696388]
 [ 0.64221071]
 [ 0.28904309]
 [ 0.03635859]]
```

In [24]:
```
theta.shape
```

Out[24]:
```
(6, 1)
```

In [25]:
```
cost_list
```

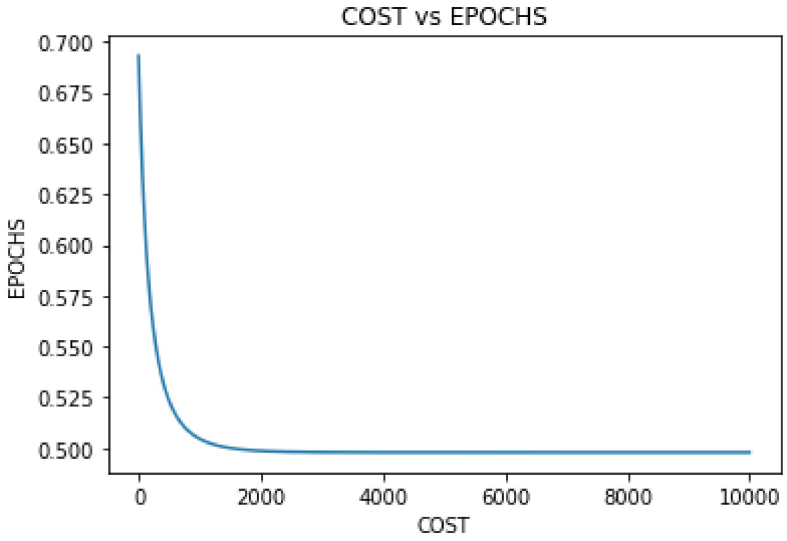
Out[25]:
```
[Outcome    0.693147
 dtype: float64,
 Outcome    0.691981
 dtype: float64,
 Outcome    0.690825
 dtype: float64,
 Outcome    0.689677
 dtype: float64,
 Outcome    0.688538
 dtype: float64,
 Outcome    0.687408
 dtype: float64,
 Outcome    0.686286
 dtype: float64,
 Outcome    0.685174
 dtype: float64,
 Outcome    0.68407
 dtype: float64,
 Outcome    0.682974
```

In [26]:
```
print("Final Cost : ",cost_list[-1])
```

```
Final Cost :  Outcome    0.49786
dtype: float64
```

ERROR PLOT

In [27]:
```
plt.title('COST vs EPOCHS')
plt.ylabel('EPOCHS')
plt.xlabel('COST')
plt.plot(cost_list)
plt.show()
```



CLASSIFYING THE PREDICTION >= 0.5 TO '1' AND <0.5 TO '0'

In [28]:
```
def pred_class(X, theta):
    prediction = []
    lines = np.dot(X,theta)
    y_pred = sigmoid(lines)
    for i in y_pred:
        if i>=0.5:
            prediction.append(1)
        else:
            prediction.append(0)
    return prediction
```

MAKING THE FIRST COLUMN OF 'x_test' AS ALL 1's

In [29]:
```
x_test = np.hstack((np.ones((x_test.shape[0],1)), x_test))
```

In [30]:
```
x_test.shape
```

Out[30]: (268, 6)

In [31]:
```
y_pred = pred_class(x_test, theta)
```

PREDICTION ON x_test =>y_pred

In [32]:
```
y_pred
```

Out[32]:
```
[0,
 0,
 0,
 0,
 0,
 0,
 0,
 1,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 1,
 1,
 1,
 0,
```

In [33]:
```
y_test.head(10)
```

Out[33]:

	Outcome
500	0
501	0
502	1
503	0
504	0
505	0
506	1
507	0
508	0
509	0

EVALUATION METRICS

CHECKING ACCURACY

In [34]:
```
def accuracy_1(X, Y, theta):

    lines_1 = np.dot(X,theta)
    y_pred_1 = sigmoid(lines_1)
    y_pred_1 = y_pred_1 > 0.5

    y_pred_1 = np.array(y_pred_1, dtype='int64')
    acc = (1-np.sum(np.absolute(y_pred_1-Y))/Y.shape[0])*100

    print("Accuracy of the model is : ", round(acc,2), "%")
    return y_pred_1
```

In [35]:
```
A = accuracy_1(x_test, y_test, theta)
```

```
Accuracy of the model is :  Outcome    80.22
dtype: float64 %
```

In [36]:
```
A
```

Out[36]:
```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [1],
       [1],
       [0],
```

CHECKING F1-SCORE

In [37]:
```
from sklearn.metrics import f1_score
F1_Score = f1_score(y_test, y_pred,average = 'micro')
print("F1 Score of the model is : ",F1_Score)
```

```
F1 Score of the model is :  0.8022388059701493
```

NAIVE BAYES

IMPORTING HEADER FILES

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn
df = pd.read_csv("diabetes.csv")
df
```

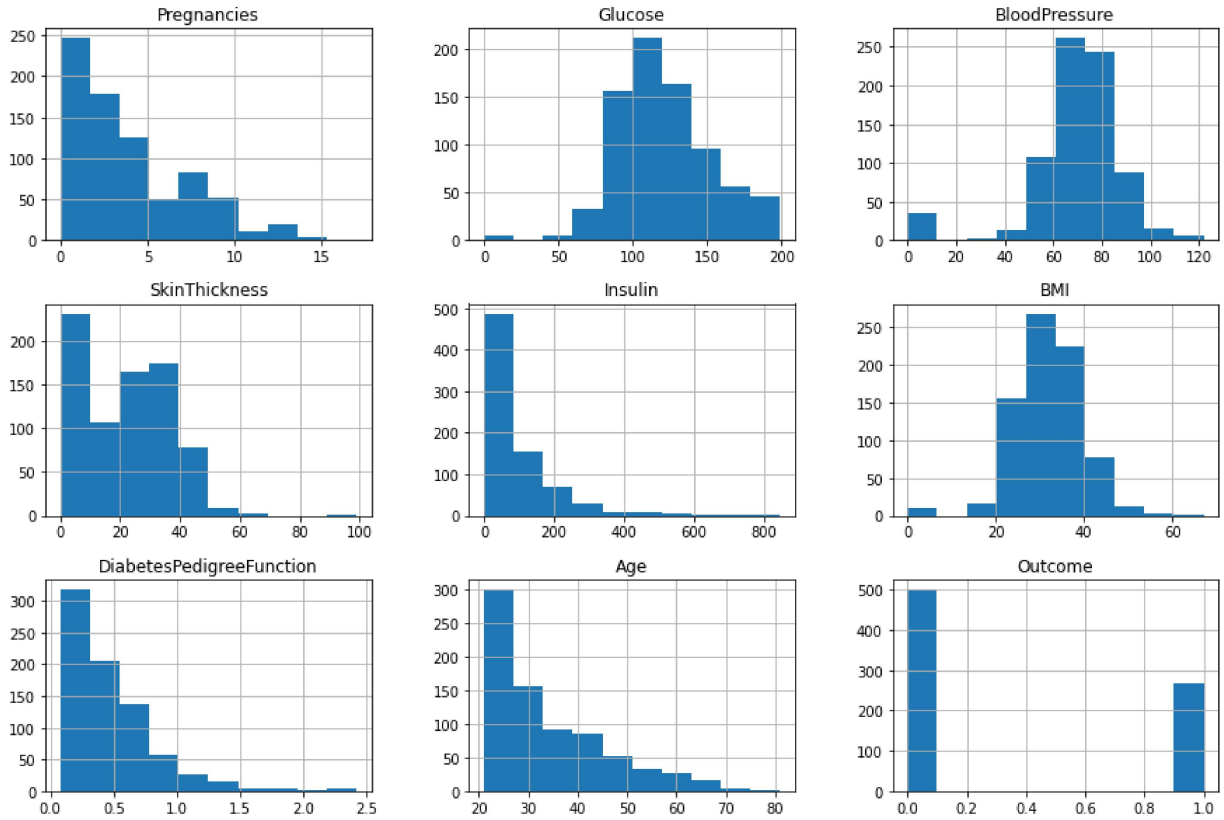
Out[1]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288
...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171
764	2	122	70	27	0	36.8	0.340
765	5	121	72	23	112	26.2	0.245
766	1	126	60	0	0	30.1	0.349
767	1	93	70	31	0	30.4	0.315

768 rows × 9 columns

VISUALIZING THE DATA

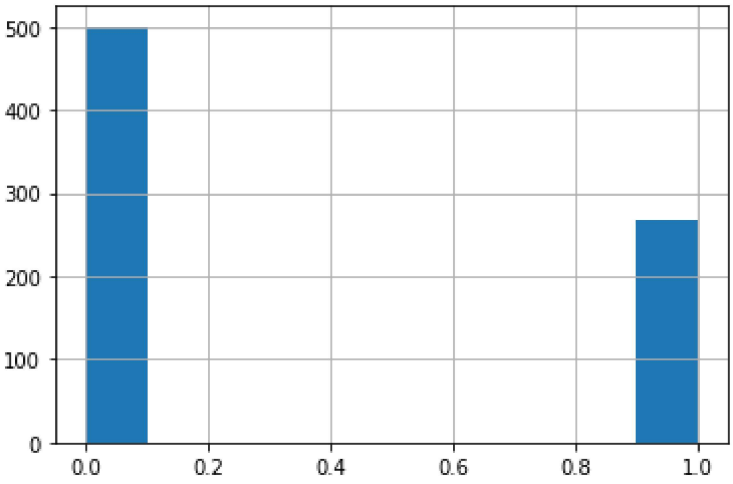
In [2]:
```
df.hist(figsize=(15,10))
plt.show()
```



CHECKING THE TARGET DATA

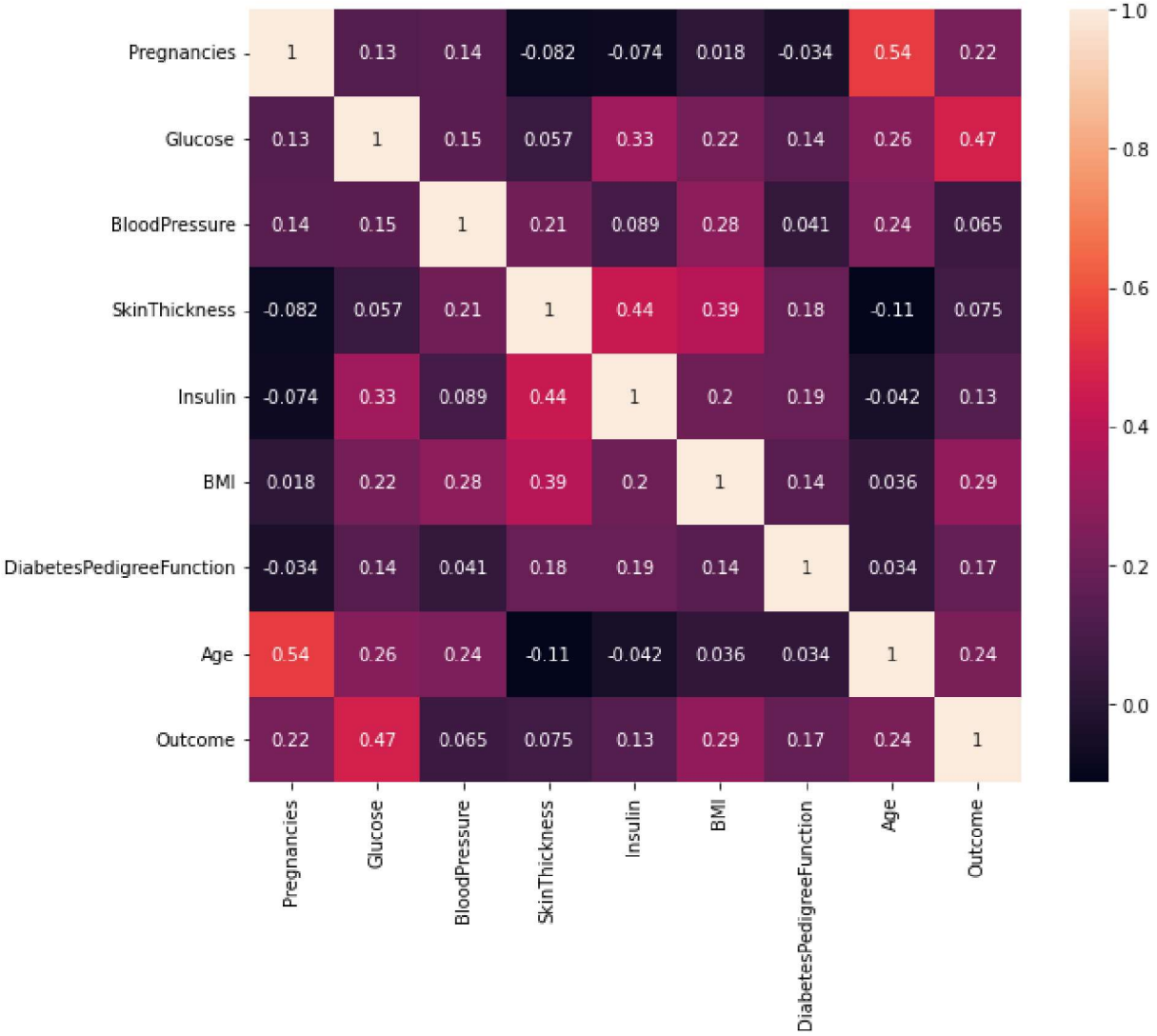
In [3]:
```
df["Outcome"].hist()
```

Out[3]: <AxesSubplot:>



CHECKING CORRELATION OF DATA

In [4]:
```
correlation = df.corr()
plt.figure(figsize=(10,8))
sn.heatmap(correlation, annot=True, square=True)
plt.show()
```



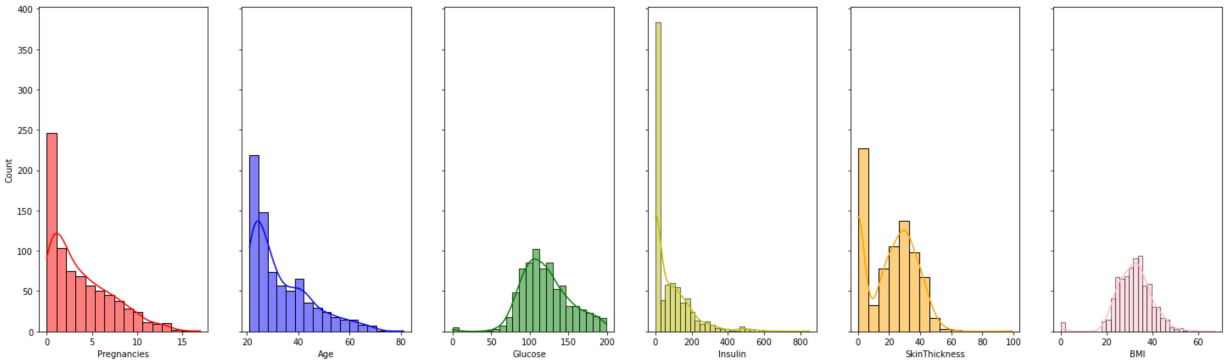
Plotting the gaussian curves to check which feature one resembles the most

In [5]:
```
fig, axes = plt.subplots(1, 6, figsize=(25, 7), sharey=True)
sn.histplot(df, ax=axes[0], x="Pregnancies", kde=True, color='r')
sn.histplot(df, ax=axes[1], x="Age", kde=True, color='b')

sn.histplot(df, ax=axes[2], x="Glucose", kde=True, color='g')
sn.histplot(df, ax=axes[3], x="Insulin", kde=True, color='y')


sn.histplot(df, ax=axes[4], x="SkinThickness", kde=True, color='orange')
sn.histplot(df, ax=axes[5], x="BMI", kde=True, color='pink')
```

Out[5]:
```
<AxesSubplot:xlabel='BMI', ylabel='Count'>
```



EXTRACTING THE FEATURES WITH PROPER GAUSSIAN CURVES

In [6]:
```
df = df[["Glucose","BMI","Outcome"]]
df
```

Out[6]:

	Glucose	BMI	Outcome
0	148	33.6	1
1	85	26.6	0
2	183	23.3	1
3	89	28.1	0
4	137	43.1	1
...	...	...	...
763	101	32.9	0
764	122	36.8	0
765	121	26.2	0
766	126	30.1	1
767	93	30.4	0

768 rows × 3 columns

Splitting training and testing

In [7]:
```
train = df.iloc[:500,:]
test = df.iloc[500:,:]
```

In [8]:
```
x_test = test[['Glucose','BMI']]
y_test = test[['Outcome']]
```

In [9]:
```
y_test.shape
```

Out[9]:
```
(268, 1)
```

CALCULATING PRIOR PROBABILITIES for "P(Y=0) and P(Y=1)"

In [10]:
```
def calculate_prior_prob(df, Y):
    classes = sorted(list(df[Y].unique()))
    prior = []
    for i in classes:
        prior.append(len(df[df[Y]==i])/len(df))
    return prior
```

CALCULATING POSTERIOR = LIKELIHOOD = P(X=x1|Y=y)P(X=x2|Y=y).....P(X=xn|Y=y)

USING GAUSSIAN

In [11]:
```
def calculate_likelihood(train_df, feature_name, feature_val, Y, label):
    feat = list(train_df.columns)

    train_df = train_df[train_df[Y]==label]

    mean, std = train_df[feature_name].mean(), train_df[feature_name].std()

    #gaussian formula
    p_x_given_y = (1 / (np.sqrt(2 * np.pi) * std)) *  np.exp(-((feature_val-mean)

    return p_x_given_y
```

CALCULATING POSTERIOR = LIKELIHOOD * PRIOR:- P(X=x1|Y=y)P(X=x2|Y=y).....P(X=xn|Y=y) * P(Y=y)

In [12]:
```
def naive_bayes(train_df,X,Y):

    #getting the features
    features = list(train_df.columns)[:-1]

    #prior
    prior = calculate_prior_prob(train_df,Y)

    y_pred = []

    for x in X:
        #likelihood
        labels = sorted(list(train_df[Y].unique()))
        likelihood = [1]*len(labels)

        for k in range(len(labels)):
            for i in range(len(features)):
                likelihood[k] *= calculate_likelihood(train_df, features[i], x[i]

        #posterior
        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]

        #classing the categories with maximum probabilities
        y_pred.append(np.argmax(post_prob))

    return np.array(y_pred)
```

In [13]:
```
ypred = naive_bayes(train, X=x_test.values, Y='Outcome')
```

PREDICTION

In [14]:
```
ypred
```

Out[14]:
```
array([0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 0], dtype=int64)
```

EVALUATION METRICS

F1 SCORE

In [15]:
```
from sklearn.metrics import f1_score
print("F1 Score : ",f1_score(y_test, ypred,average='micro'))
```

```
F1 Score :  0.783582089552239
```

In [16]:
```
y_test.shape
```

Out[16]:
```
(268, 1)
```

In [17]:
```
ypred = ypred.reshape(x_test.shape[0],1)
ypred.shape
```

Out[17]:
```
(268, 1)
```

ACCURACY

In [18]:
```
acc = (1 - np.sum(np.absolute(ypred - y_test))/268)*100
```

In [19]:
```
print("Accuracy : ", acc,"%")
```

```
Accuracy :  Outcome    78.358209
dtype: float64 %
```

Result Analysis

Metrics	Linear Regression (Univariate)	Linear Regression (Multi-variate)	Linear Regression (Closed-form)	Logistic Regression (Uni-variate)	Logistic Regression (Multi-variate)	Naïve Bayes Classifier
MSE	30.91893283	35.06152562	21.14884235	NA	NA	NA
MAE	3.822664266	4.186000706	3.428882952	NA	NA	NA
F1 Score	NA	NA	NA	0.768817204	0.802238806	0.78358209
Accuracy	NA	NA	NA	76.88	80.22	78.358209