In [1]:
```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import style
import seaborn as sns
```

In [2]:
```
# Reading the dataset
df=pd.read_csv('health care diabetes.csv')
```

In [3]:
```
# Exploring the dataset
df.head()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
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	

In [4]:
```
df.columns
```

Out[4]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [5]:
```
df.info()
# here we can see the datatype and null count
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [6]:
```
df.shape
```

Out[6]:
```
(768, 9)
```

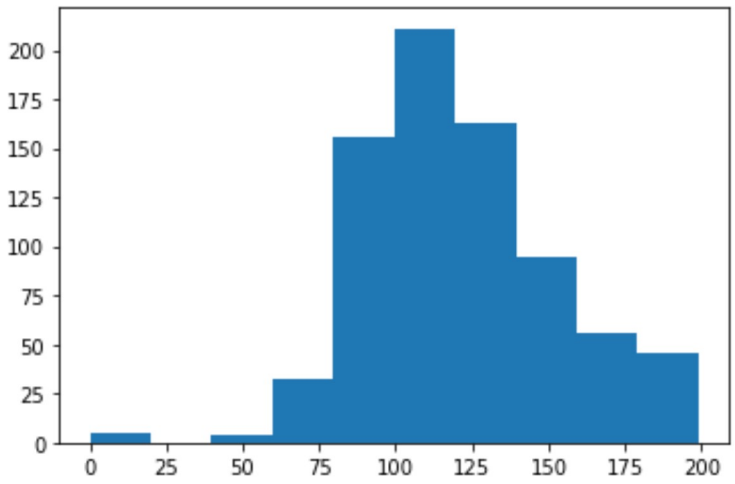
In [7]:
```
df.describe()
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPec
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	

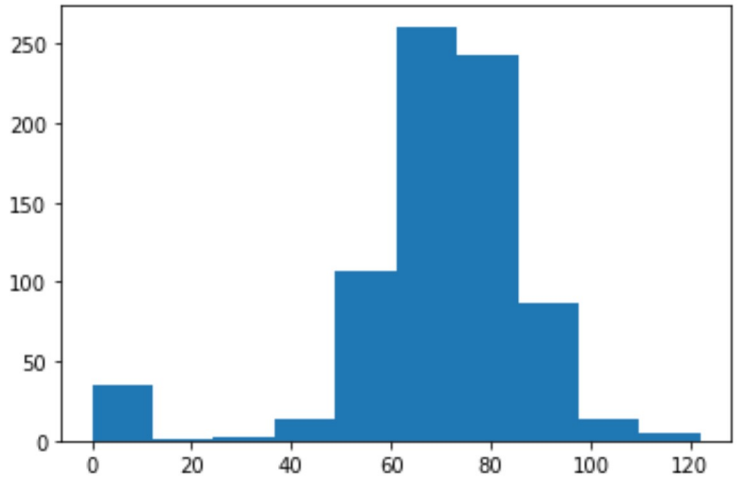
In [8]:
```
# Here we can see that there are 8 variables. Now we will create histograms of give
# (Glucose, BloodPressure, Skin Thikness, Insulin, BMI)
plt.hist(df['Glucose'])
```

Out[8]:
```
(array([  5.,   0.,   4.,  32., 156., 211., 163.,  95.,  56.,  46.]),
 array([  0. ,  19.9,  39.8,  59.7,  79.6,  99.5, 119.4, 139.3, 159.2,
        179.1, 199. ]),
 <BarContainer object of 10 artists>)
```



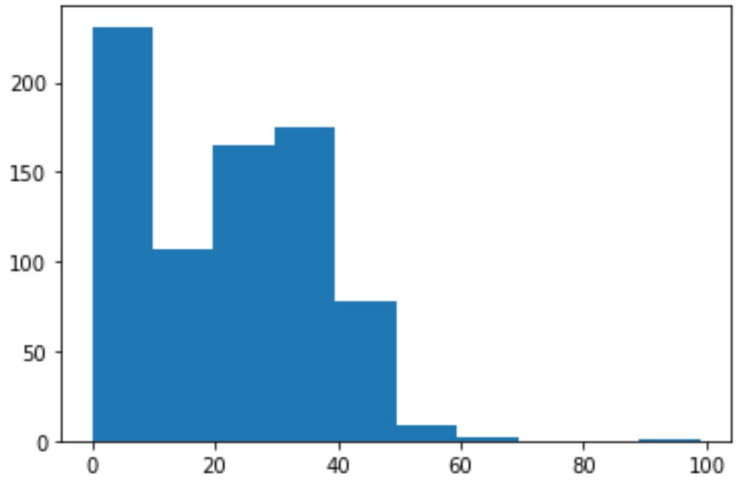
In [9]:
```
plt.hist(df['BloodPressure'])
```

Out[9]:
```
(array([ 35.,   1.,   2.,  13., 107., 261., 243.,  87.,  14.,   5.]),
 array([  0. ,  12.2,  24.4,  36.6,  48.8,  61. ,  73.2,  85.4,  97.6,
        109.8, 122. ]),
 <BarContainer object of 10 artists>)
```



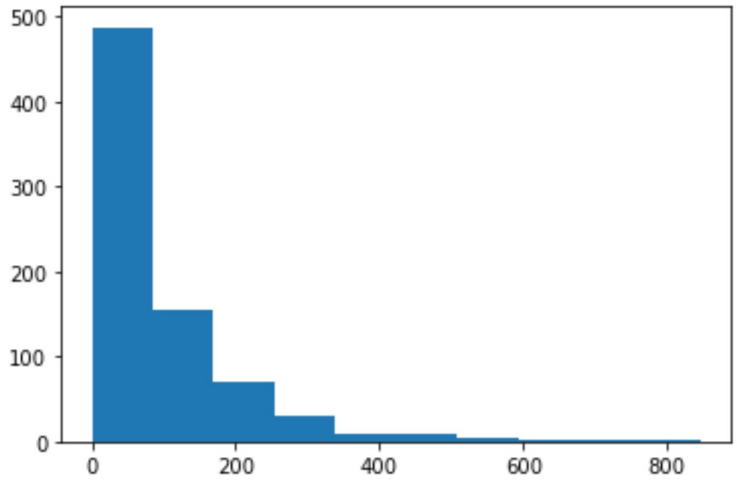
In [10]:
```
plt.hist(df['SkinThickness'])
```

Out[10]:
```
(array([231., 107., 165., 175.,  78.,   9.,   2.,   0.,   0.,   1.]),
 array([ 0. ,  9.9, 19.8, 29.7, 39.6, 49.5, 59.4, 69.3, 79.2, 89.1, 99. ]),
 <BarContainer object of 10 artists>)
```



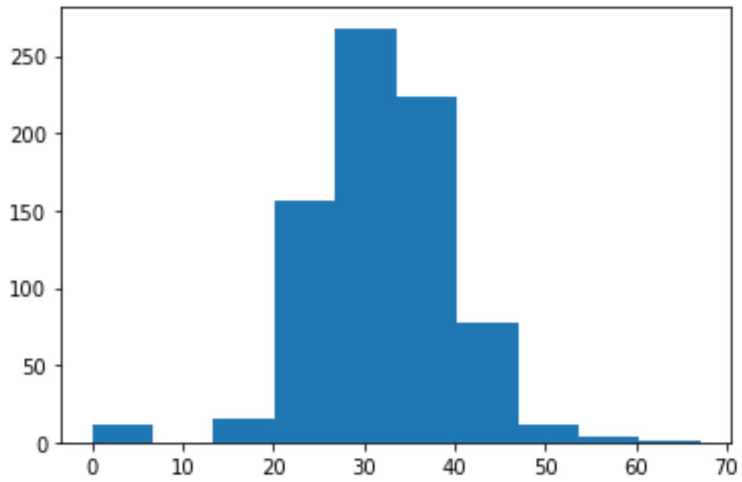
In [11]:
```
plt.hist(df['Insulin'])
```

Out[11]:
```
(array([487., 155.,  70.,  30.,   8.,   9.,   5.,   1.,   2.,   1.]),
 array([  0. ,  84.6, 169.2, 253.8, 338.4, 423. , 507.6, 592.2, 676.8,
        761.4, 846. ]),
 <BarContainer object of 10 artists>)
```



In [12]:
```
plt.hist(df['BMI'])
```

Out[12]:
```
(array([ 11.,   0.,  15., 156., 268., 224.,  78.,  12.,   3.,   1.]),
 array([ 0.  ,  6.71, 13.42, 20.13, 26.84, 33.55, 40.26, 46.97, 53.68,
        60.39, 67.1 ]),
 <BarContainer object of 10 artists>)
```



In [13]:
```
pd.isnull('df')
# there are no null values in dataset
```

Out[13]:
```
False
```

In [14]:
```
# now we'll break our dataset into two parts, diabitic and not diabitic
df_1=df[df['Outcome']==1]
df_0=df[df['Outcome']==0]
```

In [15]:
```
df_1.head()
```

Out[15]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
0	6	148	72	35	0	33.6	0.627	
2	8	183	64	0	0	23.3	0.672	
4	0	137	40	35	168	43.1	2.288	
6	3	78	50	32	88	31.0	0.248	
8	2	197	70	45	543	30.5	0.158	

In [16]:
```
df_0.head()
```

Out[16]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
1	1	85	66	29	0	26.6	0.351	
3	1	89	66	23	94	28.1	0.167	
5	5	116	74	0	0	25.6	0.201	
7	10	115	0	0	0	35.3	0.134	
10	4	110	92	0	0	37.6	0.191	

In [17]:
```
df_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 268 entries, 0 to 766
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               268 non-null    int64  
 1   Glucose                   268 non-null    int64  
 2   BloodPressure             268 non-null    int64  
 3   SkinThickness             268 non-null    int64  
 4   Insulin                   268 non-null    int64  
 5   BMI                       268 non-null    float64
 6   DiabetesPedigreeFunction  268 non-null    float64
 7   Age                       268 non-null    int64  
 8   Outcome                   268 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 20.9 KB
```

In [18]:
```
df_0.info()
# here we can see that 500 persons are non diabitic and 268 are diabitic
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 500 entries, 1 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               500 non-null    int64  
 1   Glucose                   500 non-null    int64  
 2   BloodPressure             500 non-null    int64  
 3   SkinThickness             500 non-null    int64  
 4   Insulin                   500 non-null    int64  
 5   BMI                       500 non-null    float64
 6   DiabetesPedigreeFunction  500 non-null    float64
 7   Age                       500 non-null    int64  
 8   Outcome                   500 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 39.1 KB
```

In [19]:
```
# Address the 'zero' values
df[df['Glucose']==0]
#here we can see that 5 entries with Glucose=0. we can replace these value with mea
#and for outcome=1 take mean of df_1
```

Out[19]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
75	1	0	48	20	0	24.7	0.140
182	1	0	74	20	23	27.7	0.299
342	1	0	68	35	0	32.0	0.389
349	5	0	80	32	0	41.0	0.346
502	6	0	68	41	0	39.0	0.727

In [20]:
```
df["Glucose"] = np.where((df["Glucose"] == 0) & (df["Outcome"]== 0), round(df_0["Gl
df["Glucose"] = np.where((df["Glucose"] == 0) & (df["Outcome"]== 1), round(df_1["Gl
```

In [21]:
```
# xy[(xy['Glucose']==0]) & (xy['Outcome']==1)]=df_1['Glucose'].mean()
# xy=pd.read_csv('health care diabetes.csv')
```

In [22]:
```
# xy[xy['Glucose']==0]
```

In [23]:
```
# xy[(xy['Glucose']==0) & (xy['Outcome']==0)].Glucose
# df.loc[df["gender"] == "male", "gender"] = 1
# xy.loc[(xy['Glucose']==0) & (xy['Outcome'==1]),'Glucose'] = df_1['Glucose'].mean(
# xy.loc[xy['Glucose']==0, xy['Glucose'] = 109]
# df["gender"] = np.where(df["gender"] == "female", 0, 1)
# xy["Glucose"] = np.where((xy["Glucose"] == 0) & (xy["Outcome"]== 0), round(df_0["
# xy["Glucose"] = np.where((xy["Glucose"] == 0) & (xy["Outcome"]== 1), round(df_1["
# xy["SkinThickness"] = np.where((xy["SkinThickness"] == 0) & (xy["Outcome"]== 1),
# xy["SkinThickness"] = np.where((xy["SkinThickness"] == 0) & (xy["Outcome"]== 0),

# round(df_1['Glucose'].mean(),2)
```

In [24]:
```
# xy.loc[(xy['Glucose']==0),'Glucose'] = 109
# xy[(xy['Glucose']==0) & (xy['Outcome']==0)]
# xy[xy['Glucose']==0]
```

In [25]:
```
# xy.head(10)
```

In [26]:
```
# df_0['Glucose'].mean()
```

In [27]:
```
df[df['BloodPressure']==0]
# here also we will replace the 0 values by taking mean BP for diabitic and non dia
```

Out[27]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
7	10	115.0	0	0	0	35.3	0.134
15	7	100.0	0	0	0	30.0	0.484
49	7	105.0	0	0	0	0.0	0.305
60	2	84.0	0	0	0	0.0	0.304
78	0	131.0	0	0	0	43.2	0.270
81	2	74.0	0	0	0	0.0	0.102
172	2	87.0	0	23	0	28.9	0.773
193	11	135.0	0	0	0	52.3	0.578
222	7	119.0	0	0	0	25.2	0.209
261	3	141.0	0	0	0	30.0	0.761
266	0	138.0	0	0	0	36.3	0.933
269	2	146.0	0	0	0	27.5	0.240
300	0	167.0	0	0	0	32.3	0.839
332	1	180.0	0	0	0	43.3	0.282
336	0	117.0	0	0	0	33.8	0.932
347	3	116.0	0	0	0	23.5	0.187
357	13	129.0	0	30	0	39.9	0.569
426	0	94.0	0	0	0	0.0	0.256
430	2	99.0	0	0	0	22.2	0.108
435	0	141.0	0	0	0	42.4	0.205
453	2	119.0	0	0	0	19.6	0.832
468	8	120.0	0	0	0	30.0	0.183
484	0	145.0	0	0	0	44.2	0.630
494	3	80.0	0	0	0	0.0	0.174
522	6	114.0	0	0	0	0.0	0.189
533	6	91.0	0	0	0	29.8	0.501
535	4	132.0	0	0	0	32.9	0.302
589	0	73.0	0	0	0	21.1	0.342
601	6	96.0	0	0	0	23.7	0.190
604	4	183.0	0	0	0	28.4	0.212
619	0	119.0	0	0	0	32.4	0.141
643	4	90.0	0	0	0	28.0	0.610
697	0	99.0	0	0	0	25.0	0.253

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
703	2	129.0	0	0	0	38.5	0.304
706	10	115.0	0	0	0	0.0	0.261

In [28]:
```
df["BloodPressure"] = np.where((df["BloodPressure"] == 0) & (df["Outcome"]== 0), ro
df["BloodPressure"] = np.where((df["BloodPressure"] == 0) & (df["Outcome"]== 1), ro
```

In [29]:
```
df[df['SkinThickness']==0]
# here we will replace the zero values with means of diabitic and non diabitic
```

Out[29]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
2	8	183.0	64.00	0	0	23.3	0.672
5	5	116.0	74.00	0	0	25.6	0.201
7	10	115.0	68.18	0	0	35.3	0.134
9	8	125.0	96.00	0	0	0.0	0.232
10	4	110.0	92.00	0	0	37.6	0.191
...	...	...	...	...	...	...	...
757	0	123.0	72.00	0	0	36.3	0.258
758	1	106.0	76.00	0	0	37.5	0.197
759	6	190.0	92.00	0	0	35.5	0.278
762	9	89.0	62.00	0	0	22.5	0.142
766	1	126.0	60.00	0	0	30.1	0.349

227 rows × 9 columns

In [30]:
```
df["SkinThickness"] = np.where((df["SkinThickness"] == 0) & (df["Outcome"]== 0), ro
df["SkinThickness"] = np.where((df["SkinThickness"] == 0) & (df["Outcome"]== 1), ro
```

In [31]:
```
df[df['Insulin']==0]
# here we will replace zero value with mean insulin values of diabitic and non diab
```

Out[31]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148.0	72.00	35.00	0	33.6	0.627
1	1	85.0	66.00	29.00	0	26.6	0.351
2	8	183.0	64.00	22.16	0	23.3	0.672
5	5	116.0	74.00	19.66	0	25.6	0.201
7	10	115.0	68.18	19.66	0	35.3	0.134
...	...	...	...	...	...	...	...
761	9	170.0	74.00	31.00	0	44.0	0.403
762	9	89.0	62.00	19.66	0	22.5	0.142
764	2	122.0	70.00	27.00	0	36.8	0.340
766	1	126.0	60.00	22.16	0	30.1	0.349
767	1	93.0	70.00	31.00	0	30.4	0.315

374 rows × 9 columns

In [32]:
```
df["Insulin"] = np.where((df["Insulin"] == 0) & (df["Outcome"]== 0), round(df_0["In
df["Insulin"] = np.where((df["Insulin"] == 0) & (df["Outcome"]== 1), round(df_1["In
```

In [33]:
```
df[df['BMI']==0]
# here we will replace zero values with mean of diabitic and non diabitic persons
```

Out[33]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
9	8	125.0	96.00	22.16	100.34	0.0	0.232
49	7	105.0	68.18	19.66	68.79	0.0	0.305
60	2	84.0	68.18	19.66	68.79	0.0	0.304
81	2	74.0	68.18	19.66	68.79	0.0	0.102
145	0	102.0	75.00	23.00	68.79	0.0	0.572
371	0	118.0	64.00	23.00	89.00	0.0	1.731
426	0	94.0	68.18	19.66	68.79	0.0	0.256
494	3	80.0	68.18	19.66	68.79	0.0	0.174
522	6	114.0	68.18	19.66	68.79	0.0	0.189
684	5	136.0	82.00	19.66	68.79	0.0	0.640
706	10	115.0	70.82	22.16	100.34	0.0	0.261

In [34]:
```
df["BMI"] = np.where((df["BMI"] == 0) & (df["Outcome"]== 0), round(df_0["BMI"].mean
df["BMI"] = np.where((df["BMI"] == 0) & (df["Outcome"]== 1), round(df_1["BMI"].mean
```

In [35]:
```
# Now we have treated missing values. Now we'll again check for zero values
df[df['Glucose']==0]
```

Out[35]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag

In [36]:
```
df[df['BloodPressure']==0]
```

Out[36]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag

In [37]:
```
df[df['SkinThickness']==0]
```

Out[37]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag

In [38]:
```
df[df['Insulin']==0]
```

Out[38]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag

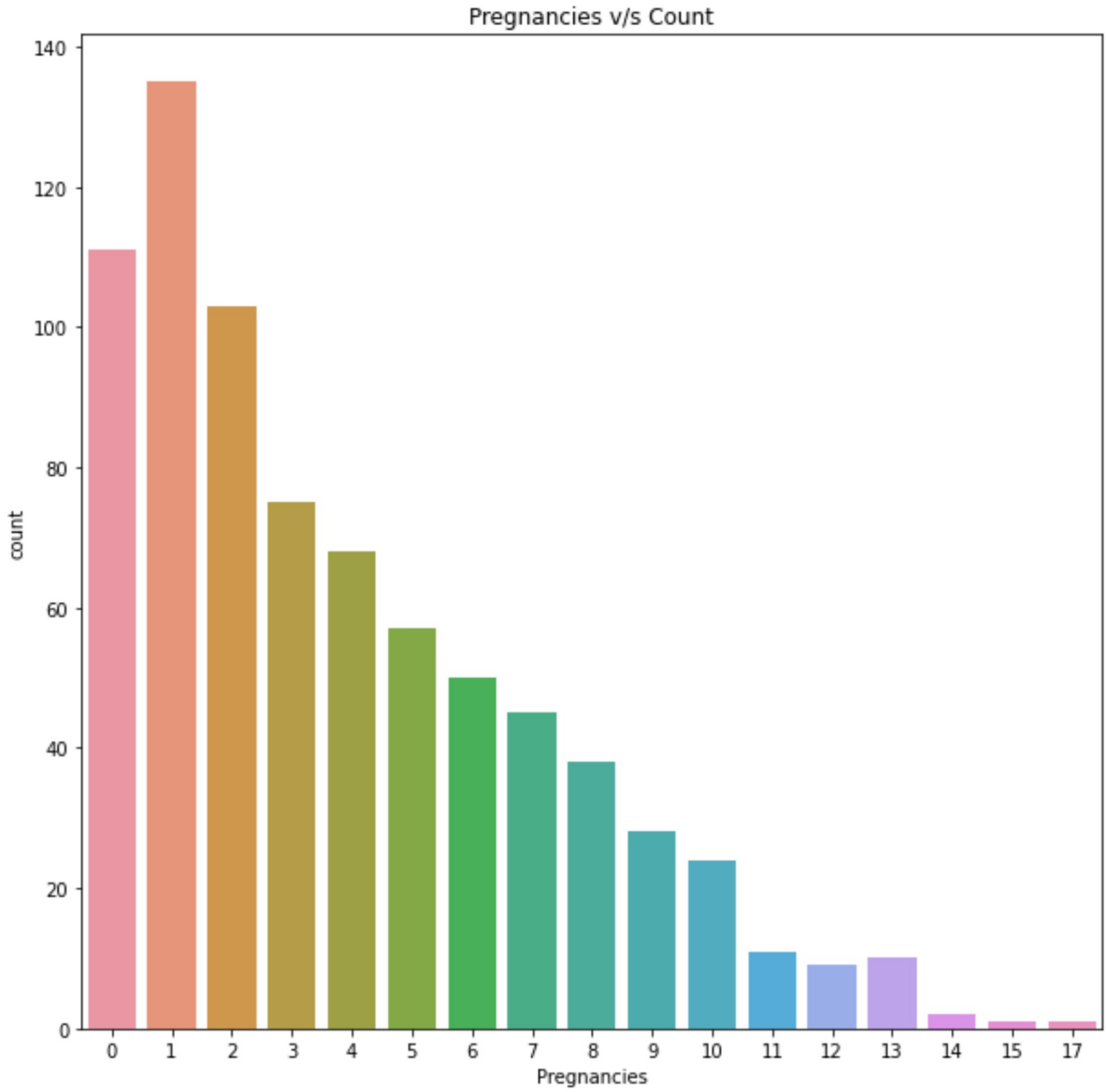
In [39]:
```
df[df['BMI']==0]
```

Out[39]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag

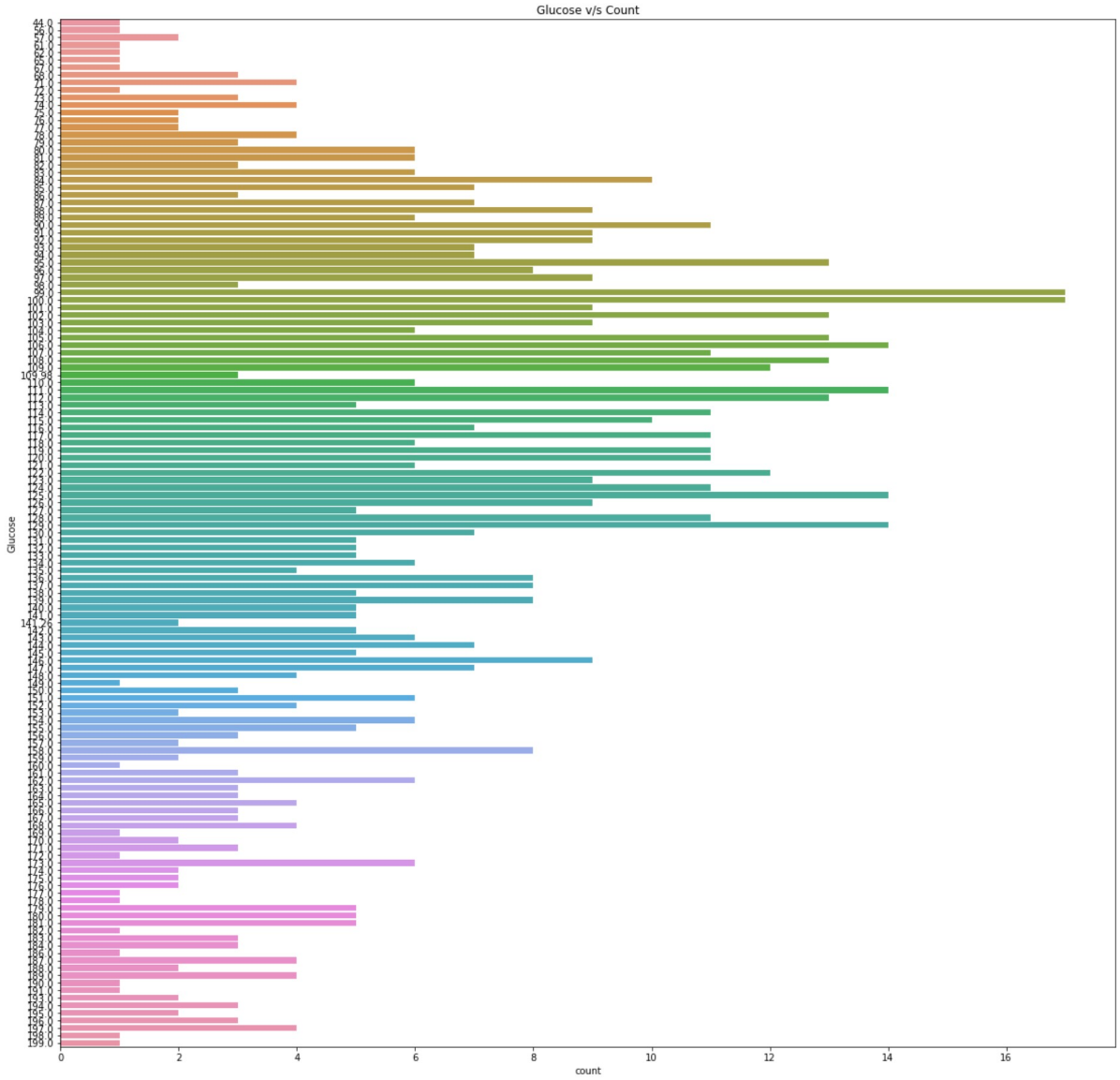
In [40]:
```
# now after treating missing values, we'll create count plot for each variable to s
plt.figure(figsize=(10,10))
plt.title('Pregnancies v/s Count')
sns.countplot(x='Pregnancies', data=df)
```

Out[40]:
```
<AxesSubplot:title={'center':'Pregnancies v/s Count'}, xlabel='Pregnancies', ylabel
='count'>
```



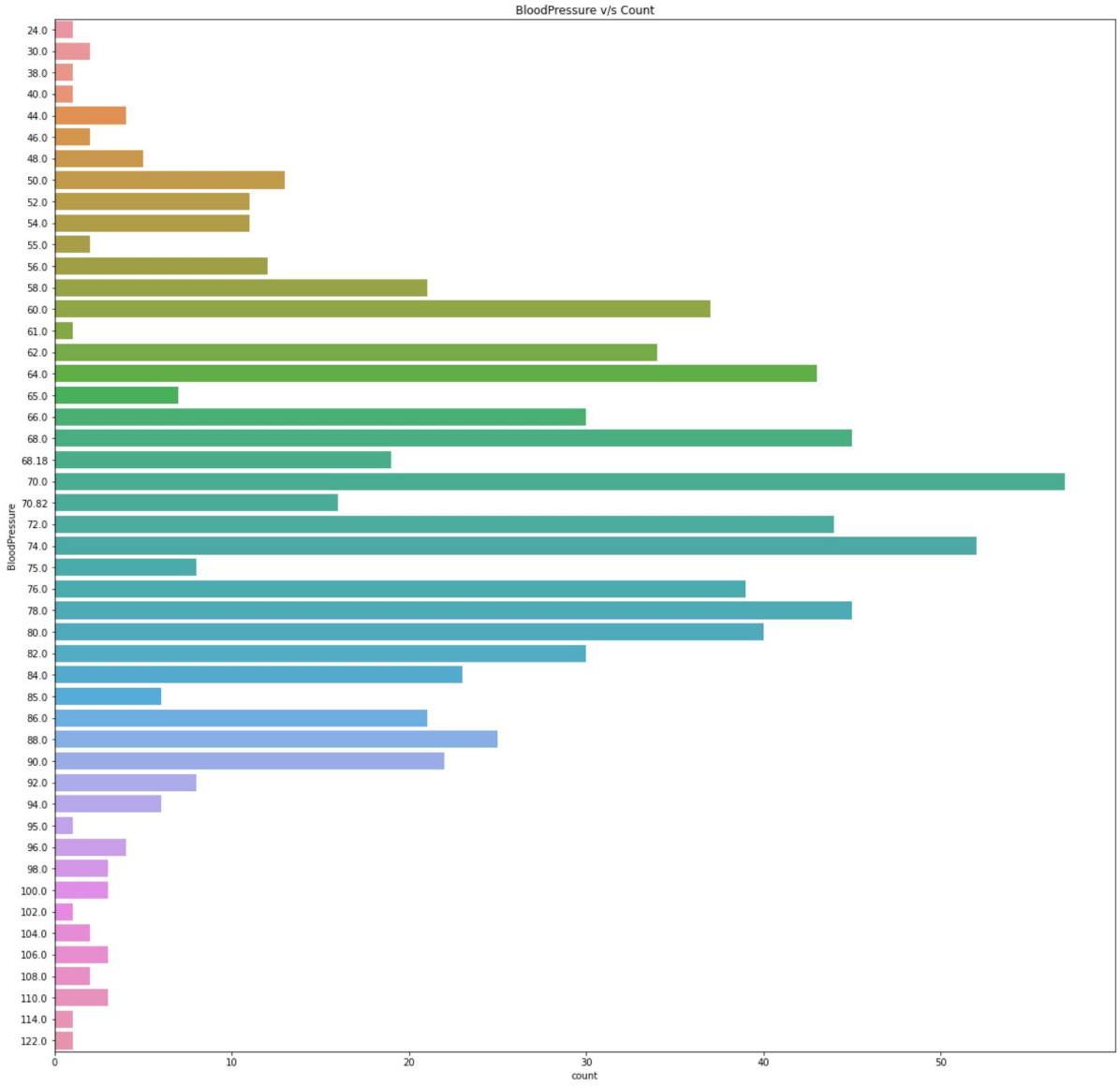
In [41]:
```
plt.figure(figsize=(20,20))
plt.title('Glucose v/s Count')
sns.countplot(y='Glucose', data=df)
```

Out[41]:
```
<AxesSubplot:title={'center':'Glucose v/s Count'}, xlabel='count', ylabel='Glucose'
>
```



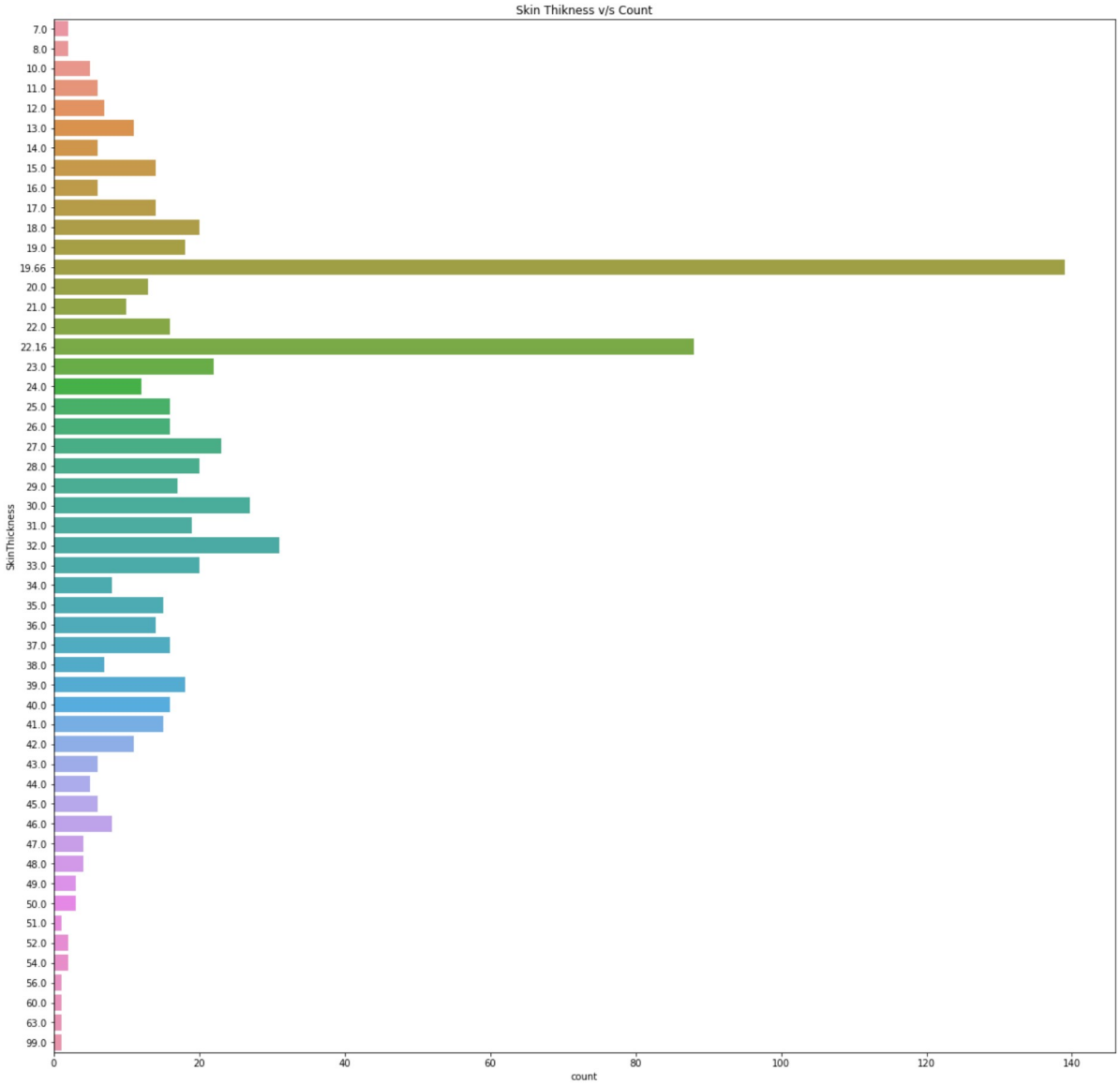
In [42]:
```
plt.figure(figsize=(20,20))
plt.title('BloodPressure v/s Count')
sns.countplot(y='BloodPressure', data=df)
```

Out[42]:
```
<AxesSubplot:title={'center':'BloodPressure v/s Count'}, xlabel='count', ylabel='Bl
oodPressure'>
```



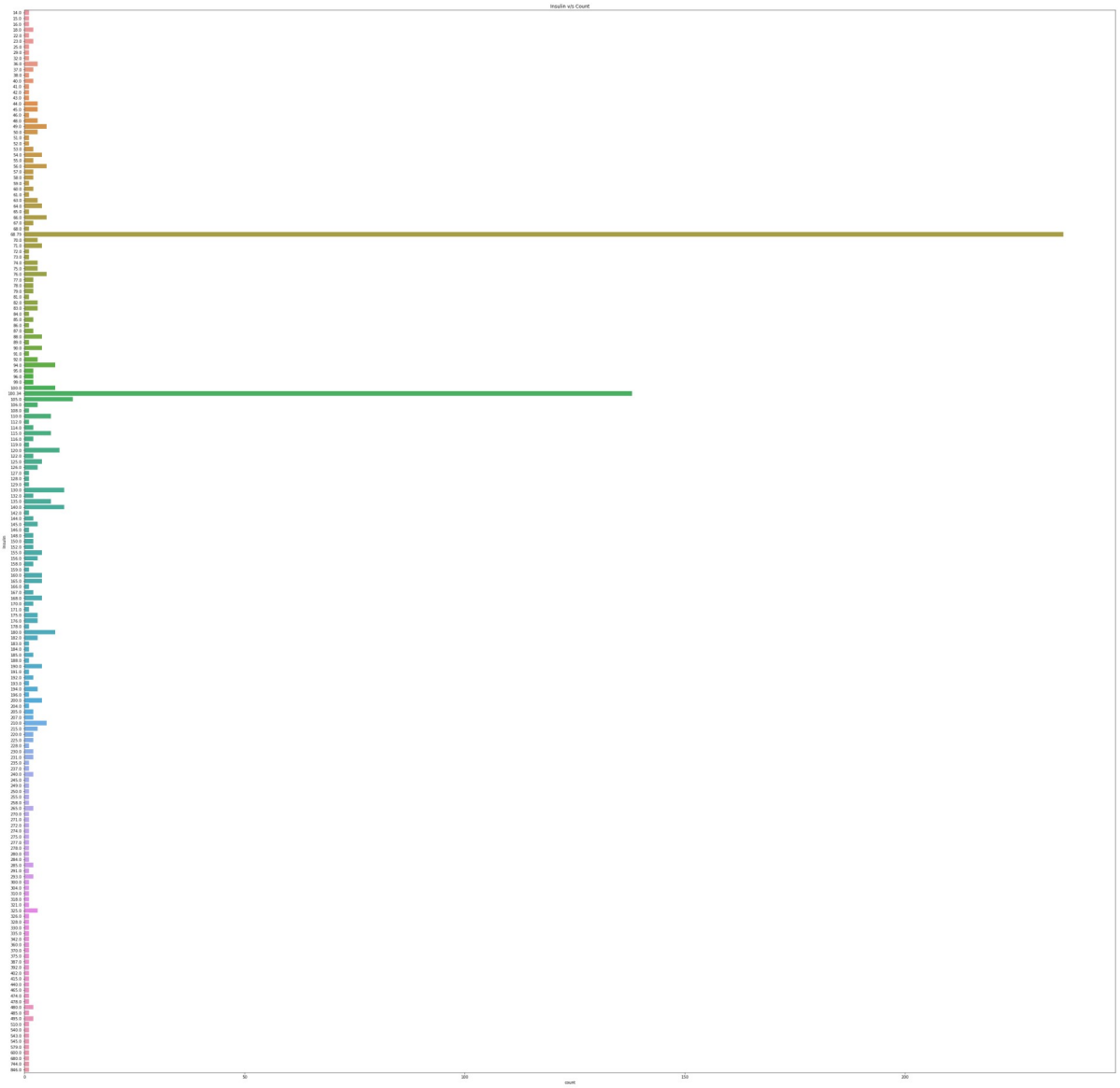
In [43]:
```
plt.figure(figsize=(20,20))
plt.title('Skin Thikness v/s Count')
sns.countplot(y='SkinThickness', data=df)
```

Out[43]:
```
<AxesSubplot:title={'center':'Skin Thikness v/s Count'}, xlabel='count', ylabel='Sk
inThickness'>
```



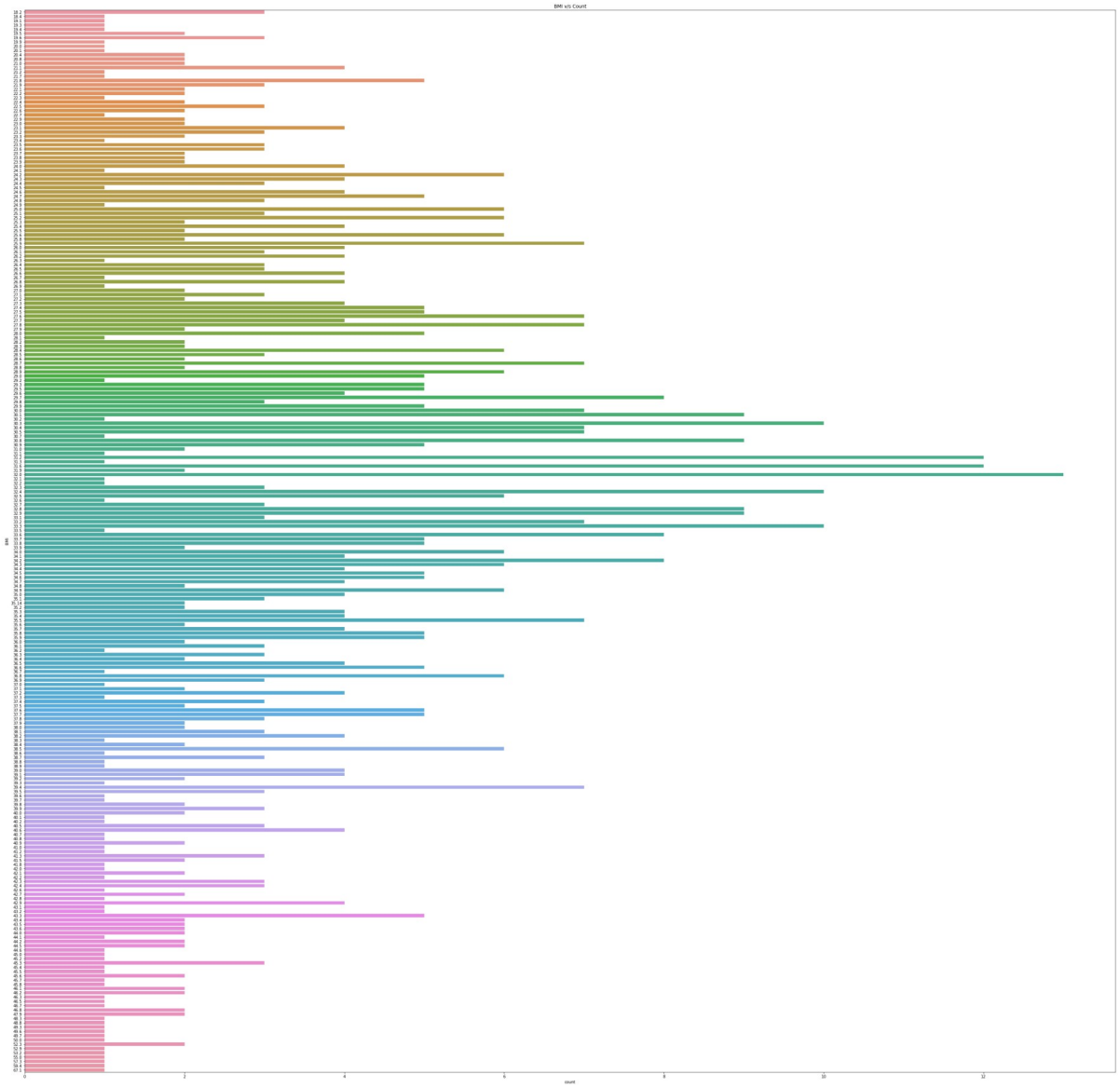
In [44]:
```
plt.figure(figsize=(50,50))
plt.title('Insulin v/s Count')
sns.countplot(y='Insulin', data=df)
```

Out[44]:
```
<AxesSubplot:title={'center':'Insulin v/s Count'}, xlabel='count', ylabel='Insulin'
>
```



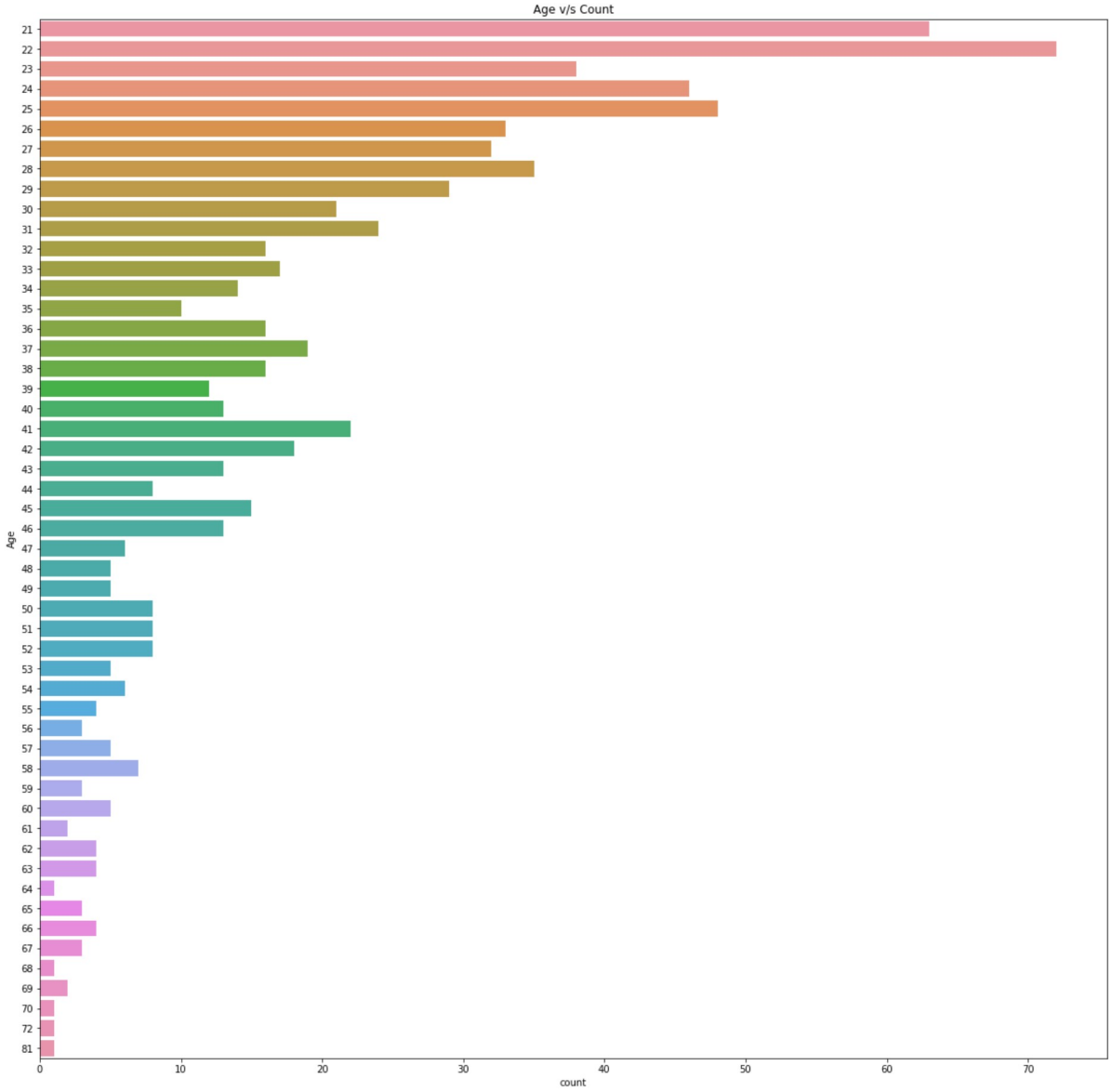
In [45]:
```
plt.figure(figsize=(50,50))
plt.title('BMI v/s Count')
sns.countplot(y='BMI', data=df)
```

Out[45]:
```
<AxesSubplot:title={'center':'BMI v/s Count'}, xlabel='count', ylabel='BMI'>
```



In [46]:
```
plt.figure(figsize=(20,20))
plt.title('Age v/s Count')
sns.countplot(y='Age', data=df)
```

Out[46]:
```
<AxesSubplot:title={'center':'Age v/s Count'}, xlabel='count', ylabel='Age'>
```



In [47]:
```
# checking the datatypes of variables
df.dtypes
```

Out[47]:
```
Pregnancies                 int64
Glucose                   float64
BloodPressure             float64
SkinThickness             float64
Insulin                   float64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

In [48]:
```
# Now we have treated the missing values. Now we will again break the dataset in di
# because zero values have been changed
df_diab=df[df['Outcome']==1]
df_non_diab=df[df['Outcome']==0]
```

In [49]:
```
df_diab.describe()
```

Out[49]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPec
count	268.000000	268.000000	268.000000	268.000000	268.000000	268.000000	
mean	4.865672	142.311642	75.052687	29.440597	152.003433	35.404776	
std	3.741239	29.488274	11.973412	9.875083	106.544926	6.590201	
min	0.000000	78.000000	30.000000	7.000000	14.000000	22.900000	
25%	1.750000	119.000000	68.000000	22.160000	100.340000	30.900000	
50%	4.000000	140.500000	74.000000	27.000000	100.340000	34.300000	
75%	8.000000	167.000000	82.000000	36.000000	167.250000	38.775000	
max	17.000000	199.000000	114.000000	99.000000	846.000000	67.100000	

In [50]:
```
df_non_diab.describe()
```

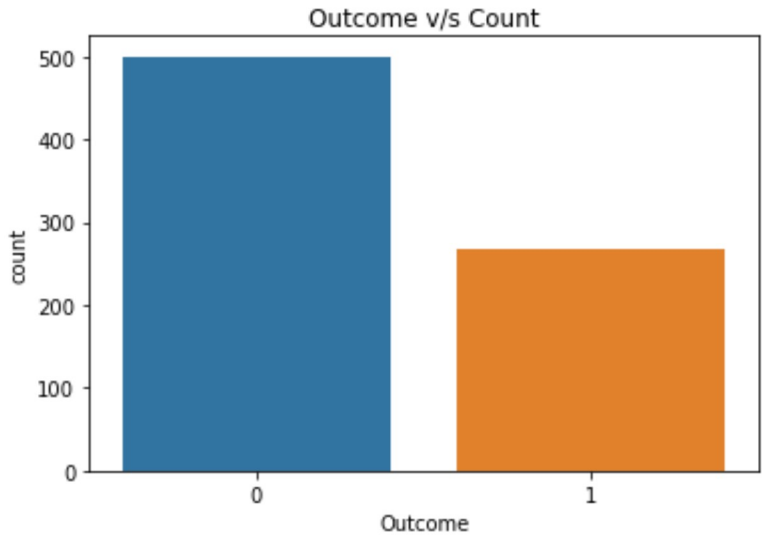
Out[50]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPec
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	
mean	3.298000	110.639880	70.774840	25.129480	101.260880	30.849600	
std	3.017185	24.702368	11.938616	9.168899	80.497603	6.501729	
min	0.000000	44.000000	24.000000	7.000000	15.000000	18.200000	
25%	1.000000	93.000000	63.500000	19.660000	68.790000	25.750000	
50%	2.000000	107.500000	70.000000	21.000000	68.790000	30.300000	
75%	5.000000	125.000000	78.000000	31.000000	105.000000	35.300000	
max	13.000000	197.000000	122.000000	60.000000	744.000000	57.300000	

In [51]:
```
# Check the balance of the data by plotting the count of outcomes by their value

plt.title('Outcome v/s Count')
sns.countplot(x='Outcome', data=df)
```

Out[51]:
```
<AxesSubplot:title={'center':'Outcome v/s Count'}, xlabel='Outcome', ylabel='count'
>
```



In [52]:
```
(df[df['Outcome']==0]).count()
# here we can see 500 individuals are non diabitic
```

Out[52]:
```
Pregnancies                 500
Glucose                     500
BloodPressure               500
SkinThickness               500
Insulin                     500
BMI                         500
DiabetesPedigreeFunction    500
Age                         500
Outcome                     500
dtype: int64
```

In [53]:
```
(df[df['Outcome']==1]).count()
# here we can see 268 individuals are diabitic
```

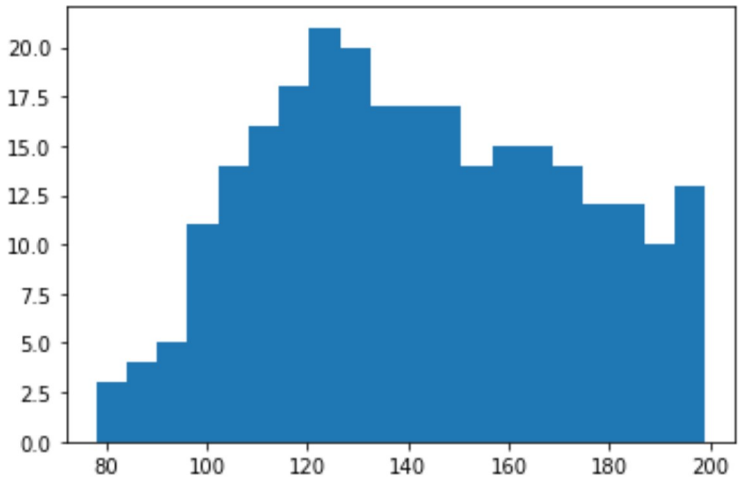
Out[53]:
```
Pregnancies                 268
Glucose                     268
BloodPressure               268
SkinThickness               268
Insulin                     268
BMI                         268
DiabetesPedigreeFunction    268
Age                         268
Outcome                     268
dtype: int64
```

In [54]:
```
# now we will check percentage of diabitic and non diabitic
print('Percentage of diabitic persons: ',(268/768)*100)
print('Percentage of non diabitic persons: ', (500/768)*100)
# Here we can see that data is divided in 65:35 ratio
```

```
Percentage of diabitic persons:  34.89583333333333
Percentage of non diabitic persons:  65.10416666666666
```

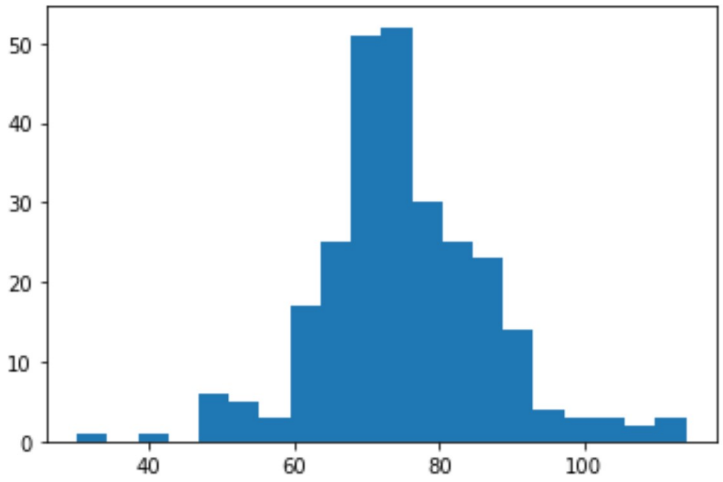
In [55]:
```
# now we will analyze variables for diabitic persons
plt.hist(df_diab['Glucose'], bins=20)
```

Out[55]:
```
(array([ 3.,  4.,  5., 11., 14., 16., 18., 21., 20., 17., 17., 17., 14.,
        15., 15., 14., 12., 12., 10., 13.]),
 array([ 78.  ,  84.05,  90.1 ,  96.15, 102.2 , 108.25, 114.3 , 120.35,
        126.4 , 132.45, 138.5 , 144.55, 150.6 , 156.65, 162.7 , 168.75,
        174.8 , 180.85, 186.9 , 192.95, 199.  ]),
 <BarContainer object of 20 artists>)
```



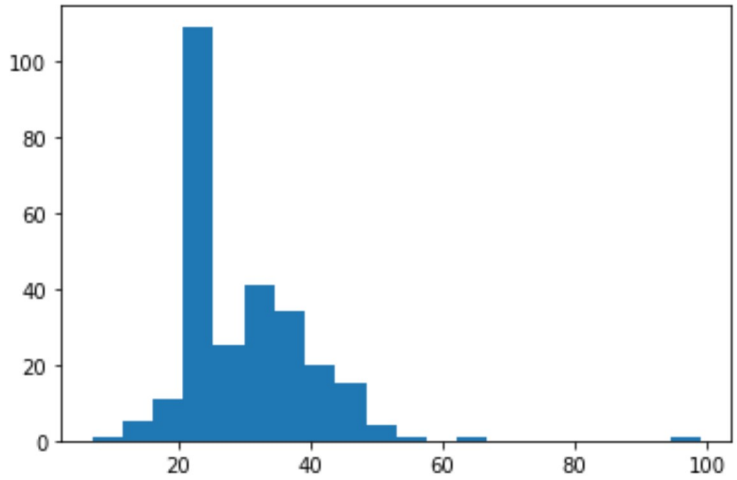
In [56]:
```
plt.hist(df_diab['BloodPressure'], bins=20)
```

Out[56]:
```
(array([ 1.,  0.,  1.,  0.,  6.,  5.,  3., 17., 25., 51., 52., 30., 25.,
        23., 14.,  4.,  3.,  3.,  2.,  3.]),
 array([ 30. ,  34.2,  38.4,  42.6,  46.8,  51. ,  55.2,  59.4,  63.6,
         67.8,  72. ,  76.2,  80.4,  84.6,  88.8,  93. ,  97.2, 101.4,
        105.6, 109.8, 114. ]),
 <BarContainer object of 20 artists>)
```



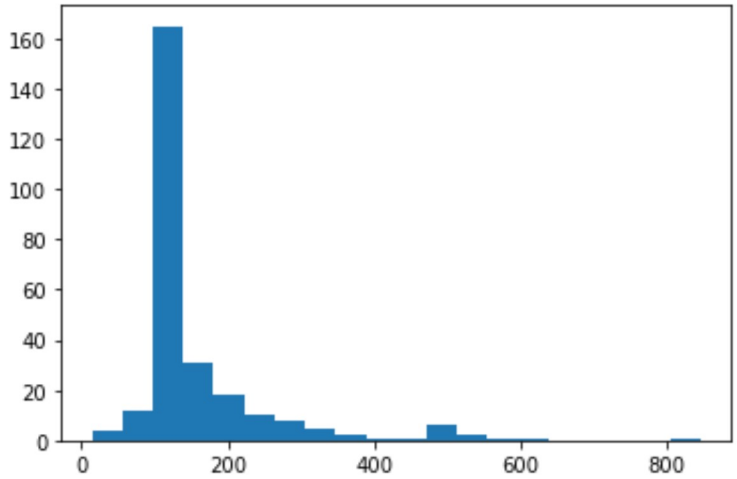
In [57]:
```
plt.hist(df_diab['SkinThickness'], bins=20)
```

Out[57]:
```
(array([  1.,   5.,  11., 109.,  25.,  41.,  34.,  20.,  15.,   4.,   1.,
          0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   1.]),
 array([ 7. , 11.6, 16.2, 20.8, 25.4, 30. , 34.6, 39.2, 43.8, 48.4, 53. ,
        57.6, 62.2, 66.8, 71.4, 76. , 80.6, 85.2, 89.8, 94.4, 99. ]),
 <BarContainer object of 20 artists>)
```



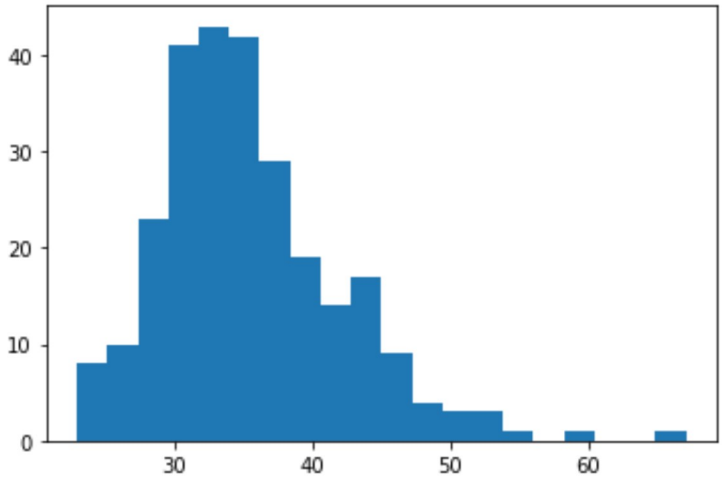
In [58]:
```
plt.hist(df_diab['Insulin'], bins=20)
```

Out[58]:
```
(array([  4.,  12., 165.,  31.,  18.,  10.,   8.,   5.,   2.,   1.,   1.,
          6.,   2.,   1.,   1.,   0.,   0.,   0.,   0.,   1.]),
 array([ 14. ,  55.6,  97.2, 138.8, 180.4, 222. , 263.6, 305.2, 346.8,
        388.4, 430. , 471.6, 513.2, 554.8, 596.4, 638. , 679.6, 721.2,
        762.8, 804.4, 846. ]),
 <BarContainer object of 20 artists>)
```



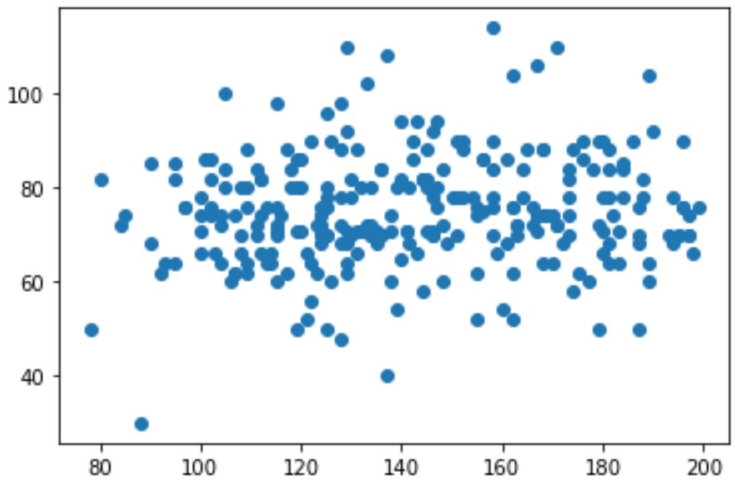
In [59]:
```
plt.hist(df_diab['BMI'], bins=20)
```

Out[59]:
```
(array([ 8., 10., 23., 41., 43., 42., 29., 19., 14., 17.,  9.,  4.,  3.,
         3.,  1.,  0.,  1.,  0.,  0.,  1.]),
 array([22.9 , 25.11, 27.32, 29.53, 31.74, 33.95, 36.16, 38.37, 40.58,
        42.79, 45.  , 47.21, 49.42, 51.63, 53.84, 56.05, 58.26, 60.47,
        62.68, 64.89, 67.1 ]),
 <BarContainer object of 20 artists>)
```



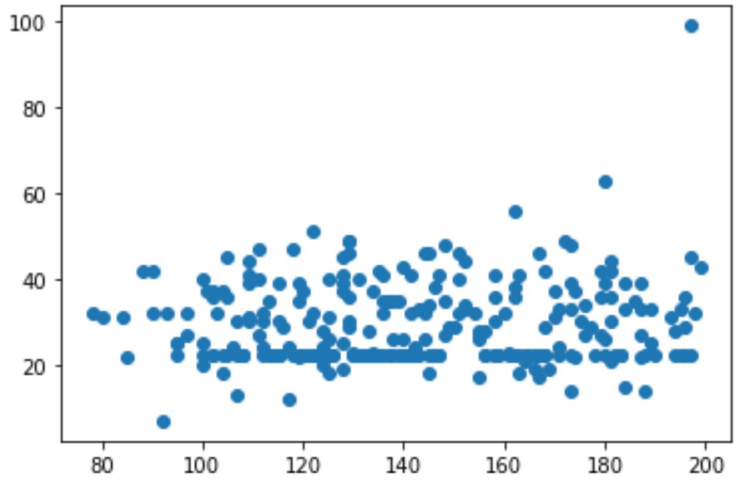
In [60]:
```
# Now we will create scatter chart of variables for diabitic cases
# Glucose       BloodPressure   SkinThickness   Insulin         BMI
x=df_diab['Glucose']
y=df_diab['BloodPressure']
plt.scatter(x,y)
plt.show
```

Out[60]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



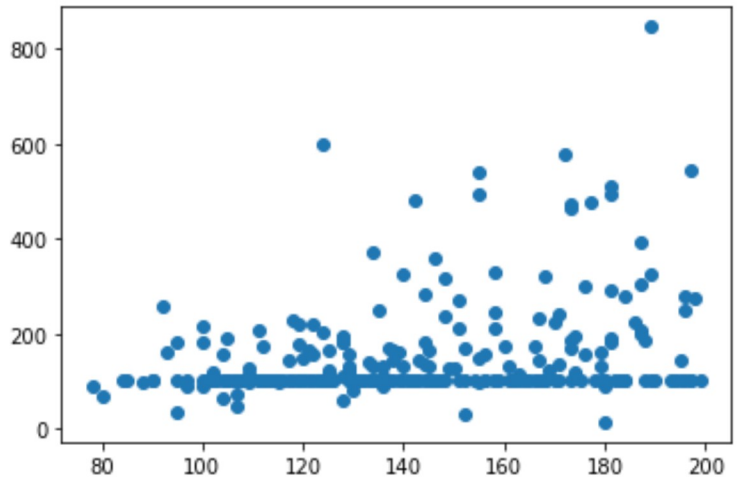
In [61]:
```
x=df_diab['Glucose']
y=df_diab['SkinThickness']
plt.scatter(x,y)
plt.show
```

Out[61]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



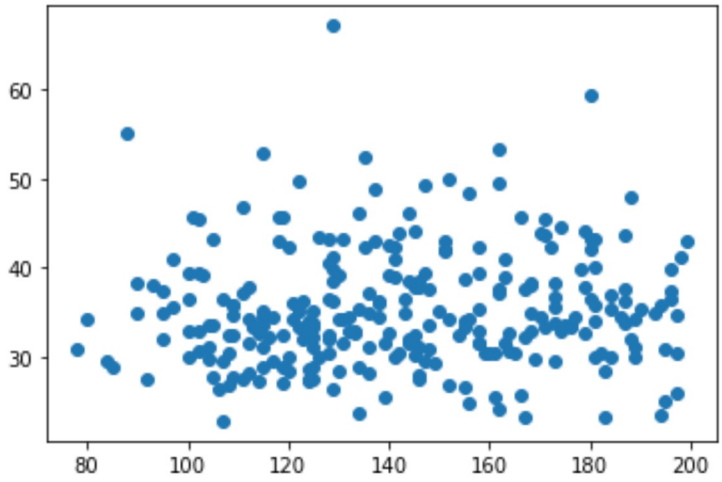
In [62]:
```
x=df_diab['Glucose']
y=df_diab['Insulin']
plt.scatter(x,y)
plt.show
```

Out[62]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



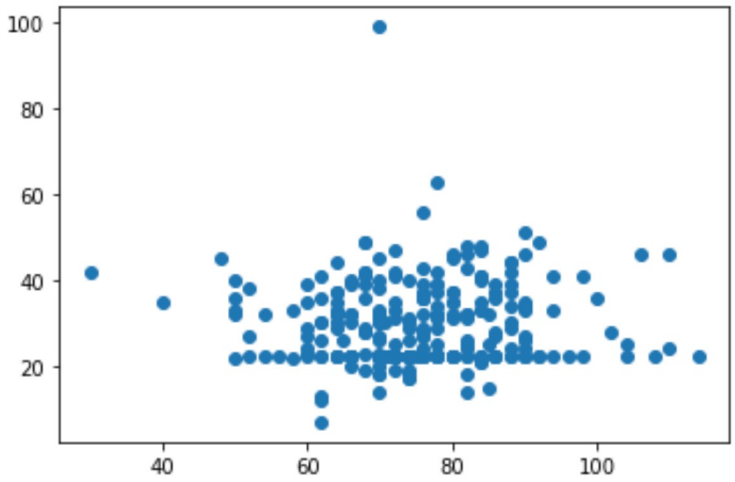
In [63]:
```
x=df_diab['Glucose']
y=df_diab['BMI']
plt.scatter(x,y)
plt.show
```

Out[63]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



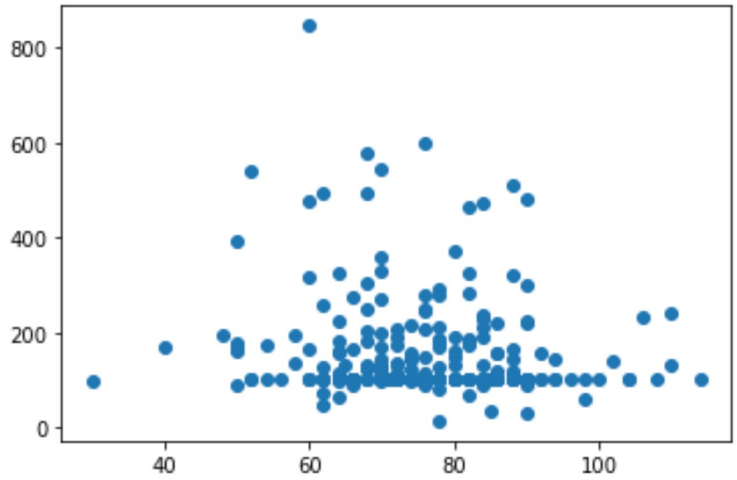
In [64]:
```
x=df_diab['BloodPressure']
y=df_diab['SkinThickness']
plt.scatter(x,y)
plt.show
```

Out[64]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



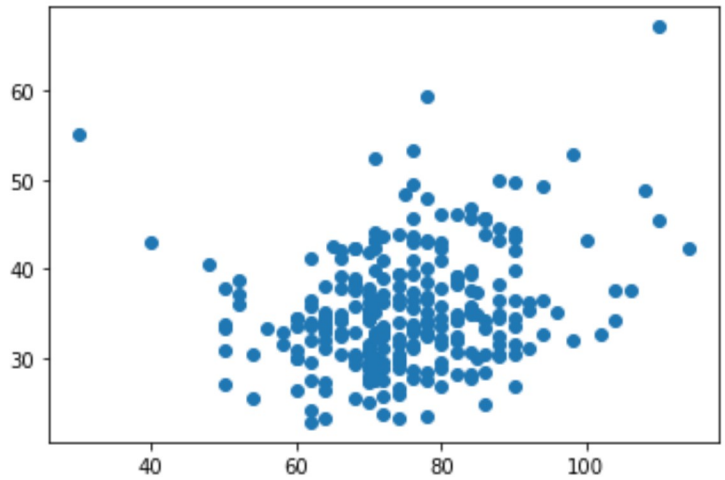
In [65]:
```
x=df_diab['BloodPressure']
y=df_diab['Insulin']
plt.scatter(x,y)
plt.show
```

Out[65]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



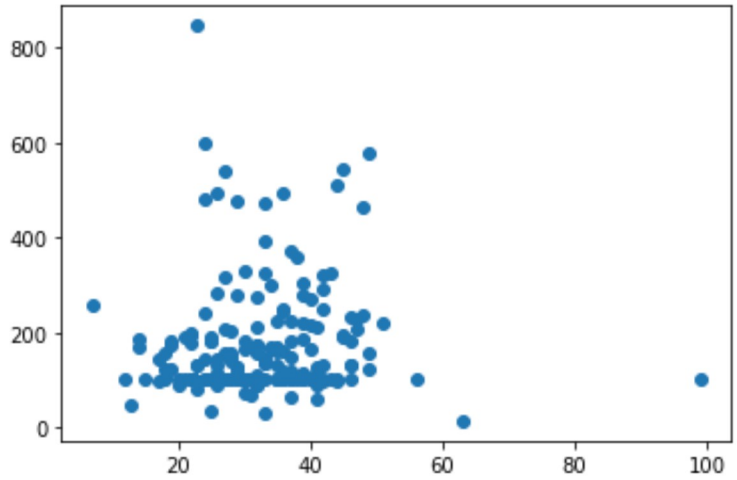
In [66]:
```
x=df_diab['BloodPressure']
y=df_diab['BMI']
plt.scatter(x,y)
plt.show
```

Out[66]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



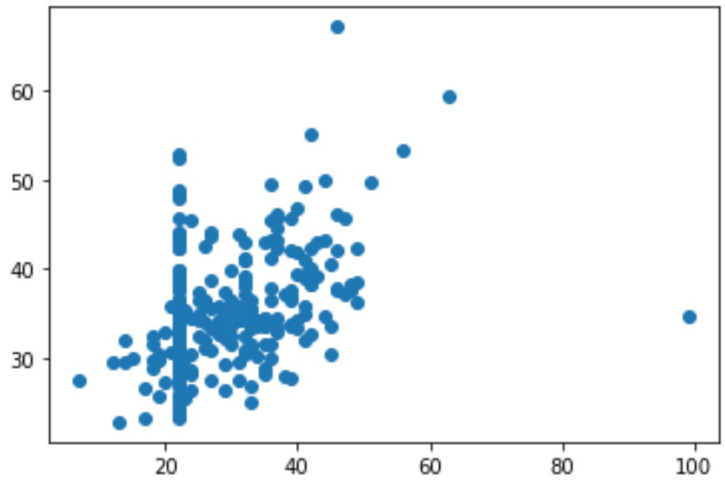
In [67]:
```
x=df_diab['SkinThickness']
y=df_diab['Insulin']
plt.scatter(x,y)
plt.show
```

Out[67]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



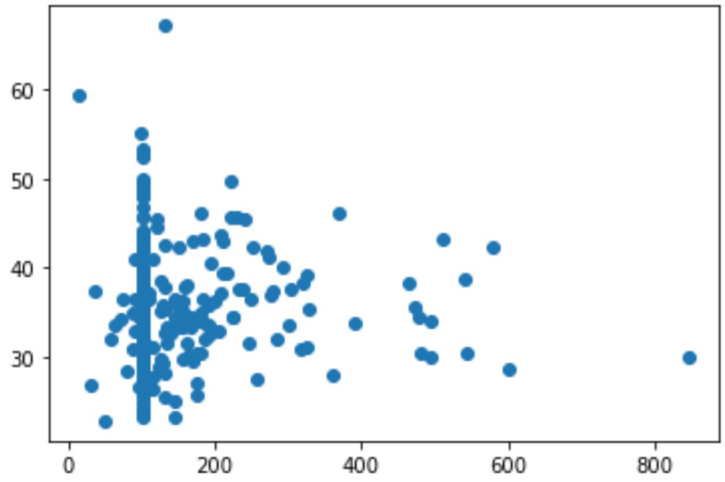
In [68]:
```
x=df_diab['SkinThickness']
y=df_diab['BMI']
plt.scatter(x,y)
plt.show
```

Out[68]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



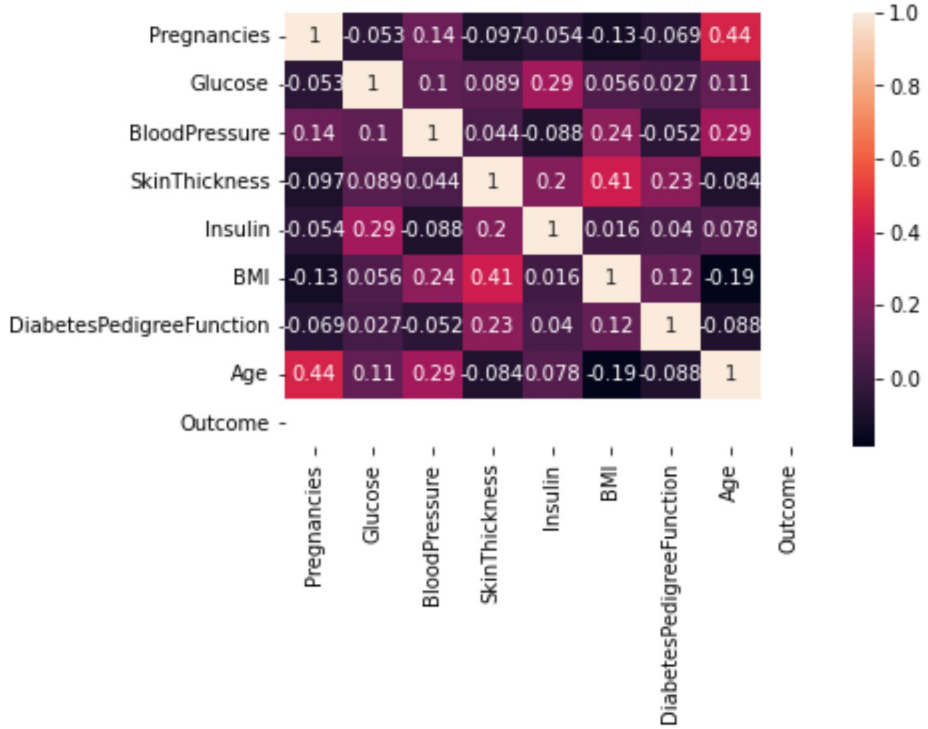
In [69]:
```
x=df_diab['Insulin']
y=df_diab['BMI']
plt.scatter(x,y)
plt.show
```

Out[69]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



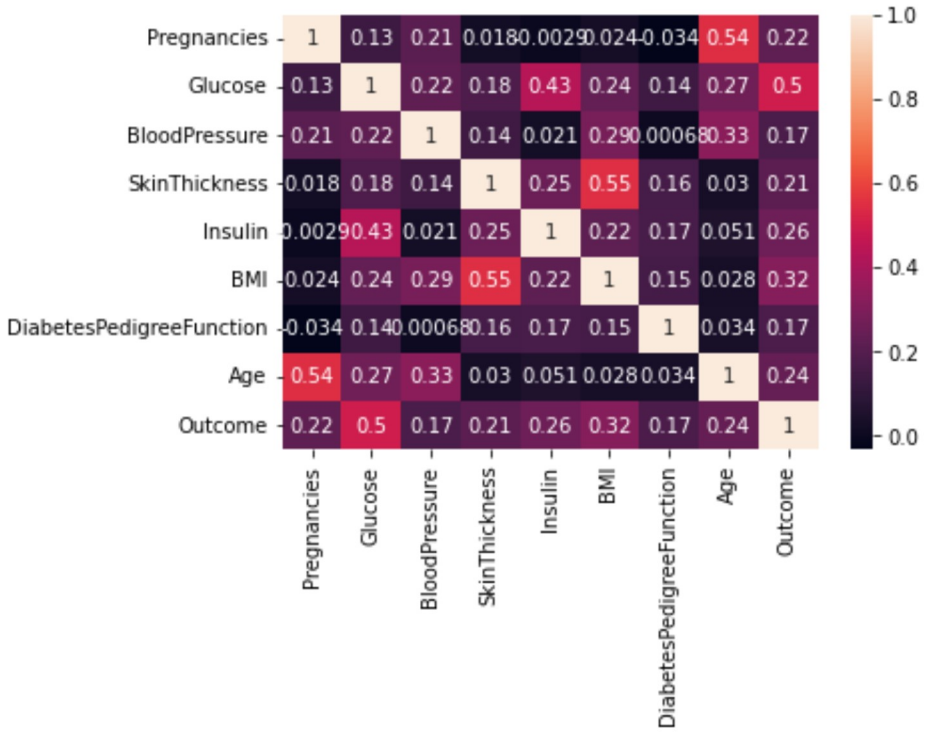
In [70]:
```
# Now we will create heat map for correlation analysis of diabitic persons
sns.heatmap(df_diab.corr(),annot=True)
# here we can see that max correlation is between skin thickness and BMI, Insulin a
```

Out[70]: <AxesSubplot:>



In [71]:
```
# heat map for correlation of non diabitic persons
sns.heatmap(df.corr(), annot=True)
# max correlation is between skin thickness and BMI, Insulin and Glucose
```

Out[71]: <AxesSubplot:>



In [72]:
```
# correlation matrix for whole data
df.corr()
```

Out[72]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BM
Pregnancies	1.000000	0.129898	0.209331	0.018033	-0.002902	0.024269
Glucose	0.129898	1.000000	0.223776	0.179743	0.431823	0.235823
BloodPressure	0.209331	0.223776	1.000000	0.138019	0.020934	0.285222
SkinThickness	0.018033	0.179743	0.138019	1.000000	0.250111	0.548612
Insulin	-0.002902	0.431823	0.020934	0.250111	1.000000	0.217975
BMI	0.024269	0.235823	0.285222	0.548612	0.217975	1.000000
DiabetesPedigreeFunction	-0.033523	0.138162	0.000683	0.156387	0.168236	0.152558
Age	0.544341	0.268613	0.326815	0.029935	0.051210	0.027931
Outcome	0.221898	0.495907	0.168399	0.213368	0.258681	0.315763

In [73]:
```
# correlation matrix for diabitic data
df_diab.corr()
```

Out[73]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BI
Pregnancies	1.000000	-0.053046	0.139959	-0.097205	-0.053676	-0.1311
Glucose	-0.053046	1.000000	0.101771	0.088721	0.289918	0.0564
BloodPressure	0.139959	0.101771	1.000000	0.043815	-0.087614	0.2351
SkinThickness	-0.097205	0.088721	0.043815	1.000000	0.204002	0.4149
Insulin	-0.053676	0.289918	-0.087614	0.204002	1.000000	0.0155
BMI	-0.131172	0.056412	0.235109	0.414945	0.015502	1.0000
DiabetesPedigreeFunction	-0.069195	0.027326	-0.052142	0.226557	0.039744	0.1181
Age	0.444987	0.113082	0.291603	-0.084205	0.078354	-0.1892
Outcome	NaN	NaN	NaN	NaN	NaN	Na

In [74]:
```
# This is a classification problem. Here we will use Logistic Regression.
# here we will break the data in features and label
features = df.iloc[:,[0,1,2,3,4,5,6,7]].values
label = df.iloc[:,8].values
```

In [75]:
```
# Now we'll do Train test split
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(features,
                                                 label,
                                                 test_size=0.3,
                                                 random_state =101)
```

In [76]:
```
#Now we'll Create our logistic regression model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train,y_train)
```

```
C:\Users\alwar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: C
onvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[76]:
```
LogisticRegression()
```

In [77]:
```
# now we'll check model score
print(model.score(X_train,y_train))
print(model.score(X_test,y_test))
```

```
0.7635009310986964
0.7705627705627706
```

In [78]:
```
# now we'll check confusion metrix

from sklearn.metrics import confusion_matrix
conf_mat= confusion_matrix(label,model.predict(features))
conf_mat
```

Out[78]:
```
array([[436,  64],
       [116, 152]], dtype=int64)
```

In [79]:
```
# classification matris

from sklearn.metrics import classification_report
print(classification_report(label,model.predict(features)))
```

```
              precision    recall  f1-score   support

           0       0.79      0.87      0.83       500
           1       0.70      0.57      0.63       268

    accuracy                           0.77       768
   macro avg       0.75      0.72      0.73       768
weighted avg       0.76      0.77      0.76       768
```

In [80]:
```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
model_dt = DecisionTreeClassifier(max_depth=5)
model_dt.fit(X_train,y_train)
```

Out[80]:
```
DecisionTreeClassifier(max_depth=5)
```

In [81]:
```
# checking model score
model_dt.score(X_train,y_train)
```

Out[81]:
```
0.9348230912476723
```

In [82]:
```
model_dt.score(X_test,y_test)
```

Out[82]:
```
0.8658008658008658
```

In [83]:
```
# Random Forest
from sklearn.ensemble import RandomForestClassifier
model_rf = RandomForestClassifier(n_estimators=11)
model_rf.fit(X_train,y_train)
```

Out[83]:
```
RandomForestClassifier(n_estimators=11)
```

In [84]:
```
# checking model score
model_rf.score(X_train,y_train)
```

Out[84]:
```
0.9962756052141527
```

In [85]:
```
model_rf.score(X_test,y_test)
```

Out[85]:
```
0.8787878787878788
```

In [86]:
```
# K nearest neighbour (KNN Algorithm)
from sklearn.neighbors import KNeighborsClassifier
model_knn = KNeighborsClassifier(n_neighbors=7,
                                 metric='minkowski',
                                 p = 2)
model_knn.fit(X_train,y_train)
```

Out[86]:
```
KNeighborsClassifier(n_neighbors=7)
```

In [87]:
```
# checking model score
model_knn.score(X_train,y_train)
```

Out[87]:
```
0.88268156424581
```

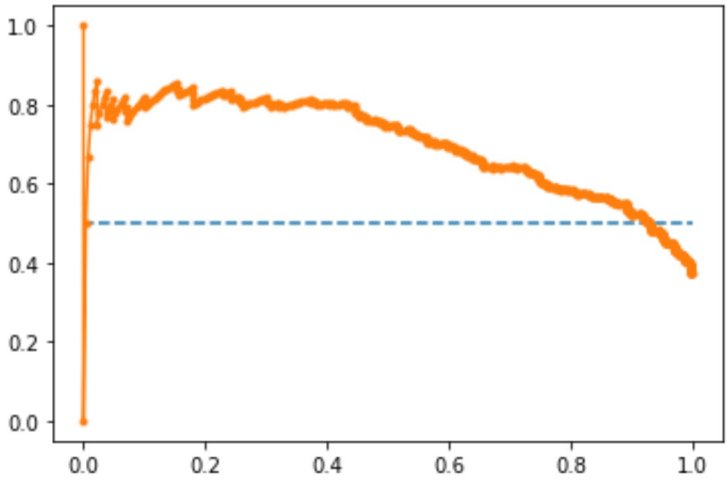
In [88]:
```
model_knn.score(X_test,y_test)
```

Out[88]:
```
0.8484848484848485
```

In [89]:
```
#Precision Recall Curve for Logistic Regression

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
# predict probabilities
prob_lr = model.predict_proba(features)
# keep probabilities for the positive outcome only
prob_lr = prob_lr[:, 1]
# predict class values
yhat = model.predict(features)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(label, prob_lr)
# calculate F1 score
f1 = f1_score(label, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(label, prob_lr)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
plt.plot(recall, precision, marker='.')
```

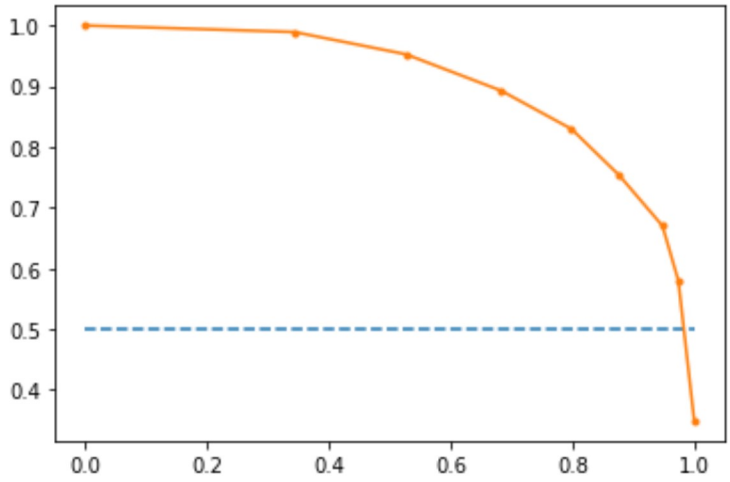
Out[89]:
```
f1=0.628 auc=0.701 ap=0.704
[<matplotlib.lines.Line2D at 0x2b009bf1af0>]
```



In [90]:
```
#Precision Recall Curve for KNN

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
# predict probabilities
prob_knn = model_knn.predict_proba(features)
# keep probabilities for the positive outcome only
prob_knn = prob_knn[:, 1]
# predict class values
yhat = model_knn.predict(features)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(label, prob_knn)
# calculate F1 score
f1 = f1_score(label, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(label, prob_knn)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
plt.plot(recall, precision, marker='.')
```

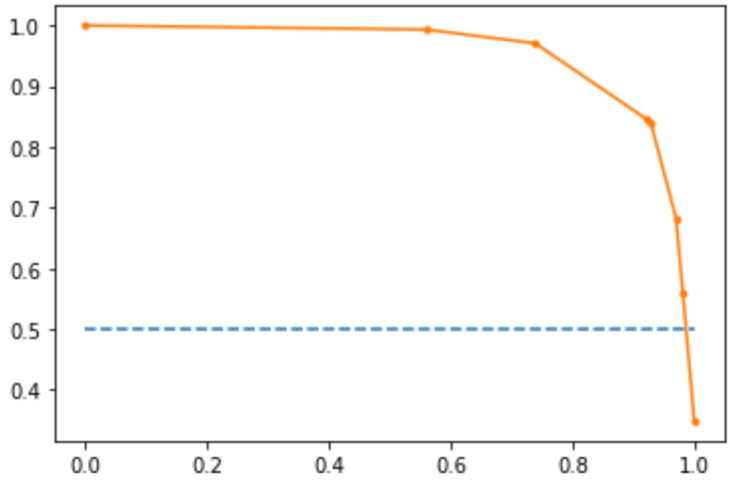
Out[90]:
```
f1=0.814 auc=0.904 ap=0.880
[<matplotlib.lines.Line2D at 0x2b009c57bb0>]
```



In [91]:
```
#Precision Recall Curve for Decission Tree Classifier

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
# predict probabilities
prob_dt = model_dt.predict_proba(features)
# keep probabilities for the positive outcome only
prob_dt = prob_dt[:, 1]
# predict class values
yhat = model_dt.predict(features)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(label, prob_dt)
# calculate F1 score
f1 = f1_score(label, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(label, prob_dt)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
plt.plot(recall, precision, marker='.')
```

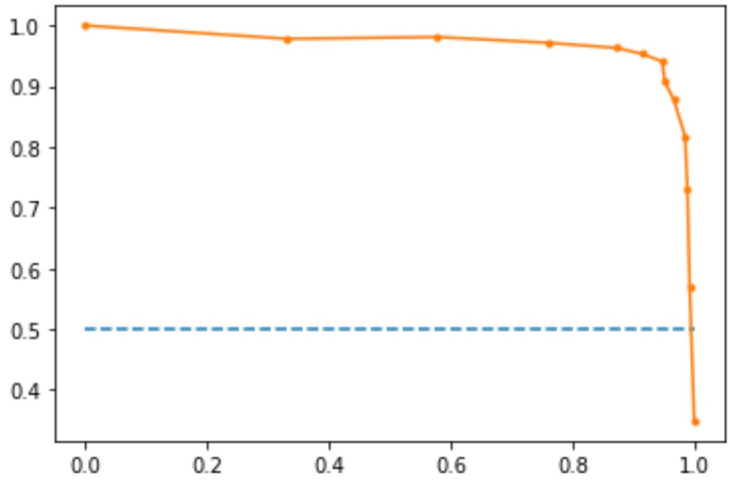
Out[91]:
```
f1=0.882 auc=0.953 ap=0.932
[<matplotlib.lines.Line2D at 0x2b009cc0790>]
```



In [92]:
```
#Precision Recall Curve for Random Forest

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
# predict probabilities
prob_rf = model_rf.predict_proba(features)
# keep probabilities for the positive outcome only
prob_rf = prob_rf[:, 1]
# predict class values
yhat = model_rf.predict(features)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(label, prob_rf)
# calculate F1 score
f1 = f1_score(label, yhat)
# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(label, prob_rf)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
plt.plot(recall, precision, marker='.')
```

Out[92]:
```
f1=0.944 auc=0.969 ap=0.962
[<matplotlib.lines.Line2D at 0x2b009d2d220>]
```



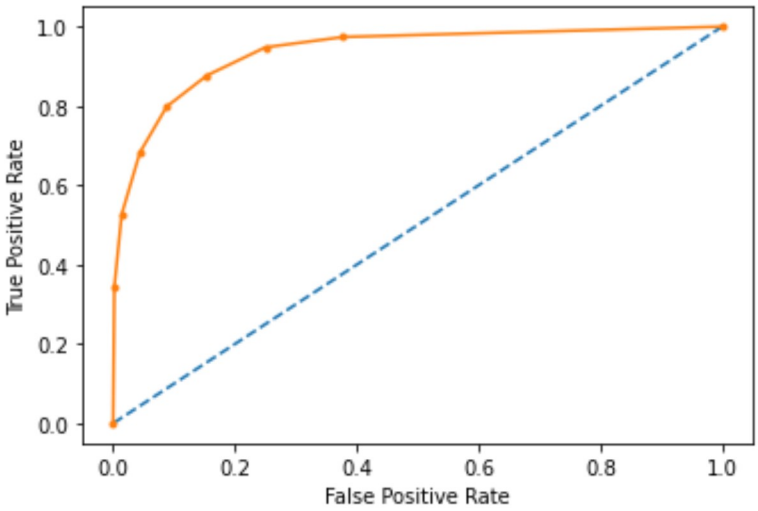
In [93]:
```
# Now we'll make ROC (Receiver Operating Characteristics) Curve for KNN model

from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# predict probabilities
prob_knn = model_knn.predict_proba(features)
# keep probabilities for the positive outcome only
prob_knn = prob_knn[:, 1]
# calculate AUC
auc = roc_auc_score(label, prob_knn)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(label, prob_knn)
print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tp
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
AUC: 0.936
True Positive Rate - [0.         0.34328358 0.5261194  0.68283582 0.79850746 0.8768
6567
 0.94776119 0.9738806  1.        ], False Positive Rate - [0.    0.002 0.014 0.044
0.088 0.154 0.25  0.378 1.   ] Thresholds - [2.         1.         0.85714286 0.714
28571 0.57142857 0.42857143
 0.28571429 0.14285714 0.        ]
```

Out[93]:
```
Text(0, 0.5, 'True Positive Rate')
```


True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0