In [ ]:

```
Pandas
Pandas is an open-source Python package that provides high-performance, easy-to-
When to use? Pandas is a perfect tool for data wrangling or munging. It is desig
Pandas take data in a CSV or TSV file or a SQL database and create a Python obje
What can you do with Pandas?
Indexing, manipulating, renaming, sorting, merging data frame
Update, Add, Delete columns from a data frame
Impute missing files, handle missing data or NANs
Plot data with histogram or box plot
This makes Pandas a foundation library in learning Python for Data Science.
```

In [ ]:

```
Ref:  https://pandas.pydata.org/pandas-docs/stable/index.html
```

In [2]:

```
os.chdir('/Users/tomisin/Dropbox/My Mac (Tomisins-MacBook-Pro.local)/Documents/Data
```

In [3]:

```
df = pd.read_csv("diabetes.csv")
```

In [4]:

```
import pandas as pd
import matplotlib.pyplot as plt
import statistics as st
import os
```

In [5]:

```
df.head()
```

Out[5]:

	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

Example of well behaved data

In [6]:

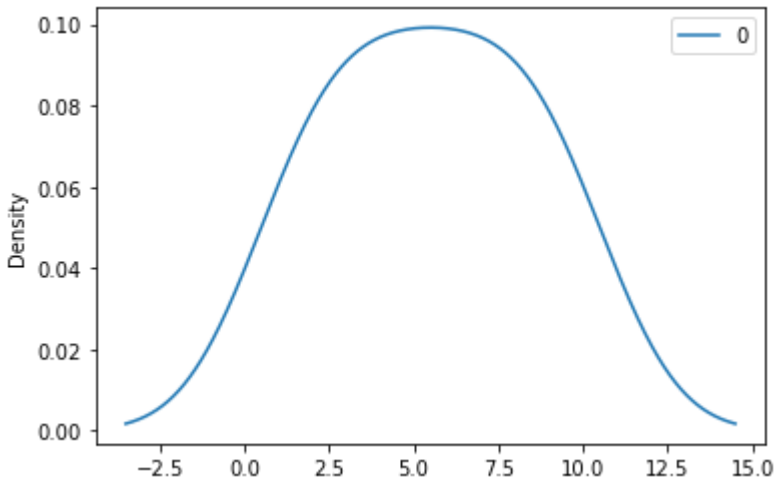
```
xdf = [1,2,3,4,5,6,7,8,9,10]

mean = 5.5
#mode =
median = [5,6]
```

In [7]:

```
dxdf = pd.DataFrame(xdf)

dxdf.plot(kind= 'density') # USE THIS TO CHECK THE DISTRIBUTION OF ANY DATAFRAME
```

Out[7]:

```
<AxesSubplot:ylabel='Density'>
```



Example of poorly behaved data

In [9]:

```
xdf2 = [1,2,3,4,5,5,3,4,5,6,5,6,1,7,8,9,10,20, 25]
#xdf2 = pd.DataFrame(xdf2)
mean = st.mean(xdf2)
print(mean)
median = st.median(xdf2)
print(median)
```

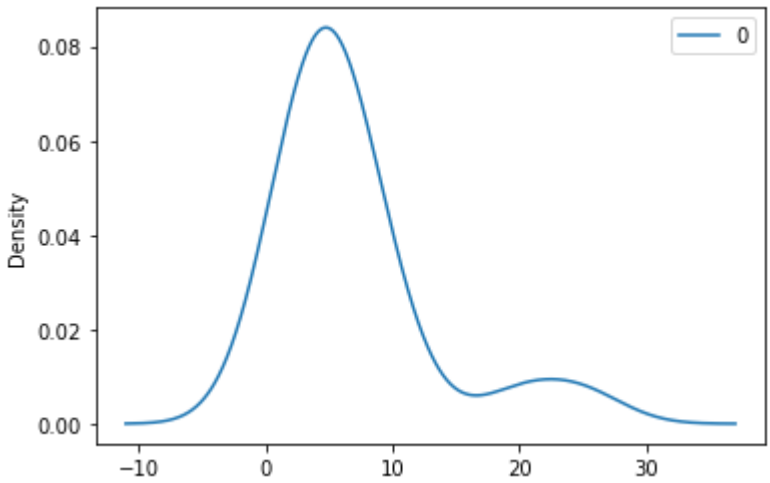
```
6.7894736842105265
5
```

In [10]:

```
dxdf2 = pd.DataFrame(xdf2)

dxdf2.plot(kind= 'density') # USE THIS TO CHECK THE DISTRIBUTION OF ANY DATAFRAM
```

Out[10]:

```
<AxesSubplot:ylabel='Density'>
```



In [ ]:

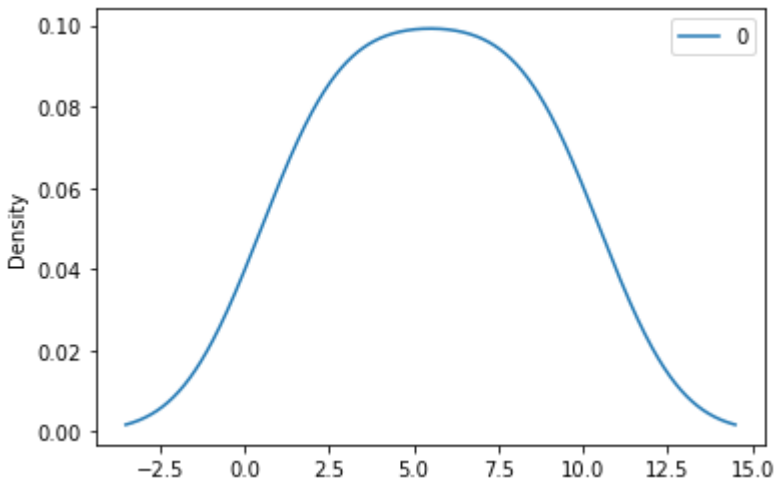
```

```

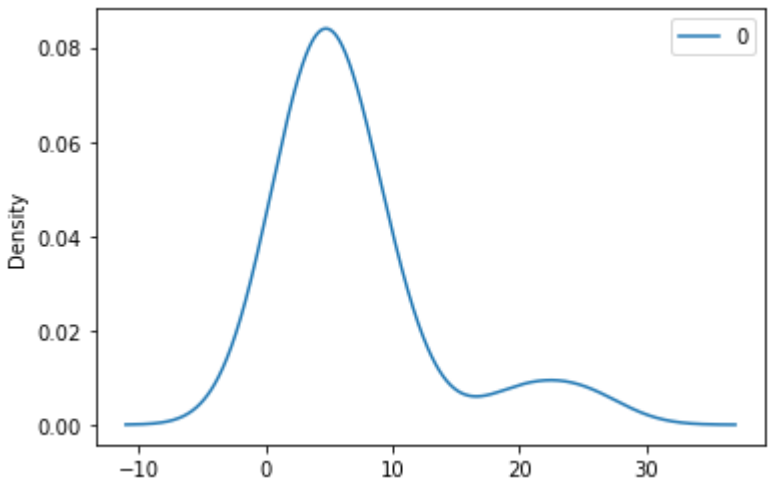
In [11]:

```
dxdf = pd.DataFrame(xdf)

dxdf.plot(kind= 'density') # USE THIS TO CHECK THE DISTRIBUTION OF ANY DATAFRAME


dxdf2 = pd.DataFrame(xdf2)

dxdf2.plot(kind= 'density') # USE THIS TO CHECK THE DISTRIBUTION OF ANY DATAFRAM
```

Out[11]:

```
<AxesSubplot:ylabel='Density'>
```

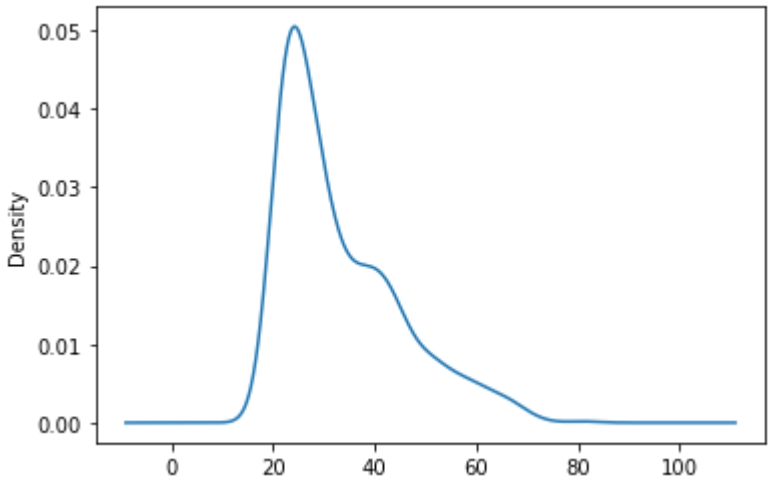




In [13]:

```
# plot a density chart of Age from the diabetes dataset


df.Age.plot(kind='density')

Age = df.Age.values
print(st.mean(Age))
print(st.median(Age))
print(max(Age))
print(min(Age))
```

```
33
29.0
81
21
```



In [ ]:

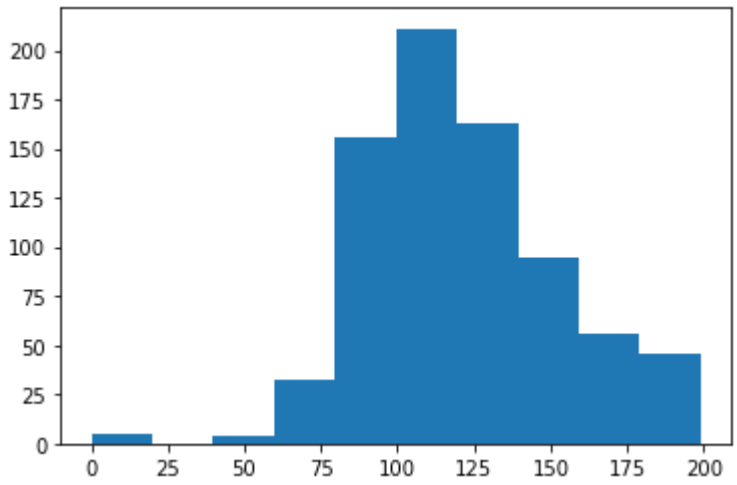
```
#non-normal distribution in data is referred to as non-gaussian
```

In [14]:

```
1 plt.hist(df.Glucose)
```

Out[14]:

```
(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>)
```



Machine Learning Algorithms

In [15]:

```
1 df
```

Out[15]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctic
0	6	148	72	35	0	33.6	0.6
1	1	85	66	29	0	26.6	0.3
2	8	183	64	0	0	23.3	0.6
3	1	89	66	23	94	28.1	0.1
4	0	137	40	35	168	43.1	2.2
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.1
764	2	122	70	27	0	36.8	0.3
765	5	121	72	23	112	26.2	0.2
766	1	126	60	0	0	30.1	0.3
767	1	93	70	31	0	30.4	0.3

768 rows × 9 columns

In [ ]:

```
1
```

In [17]:

```
1 df.describe()
```

Out[17]:

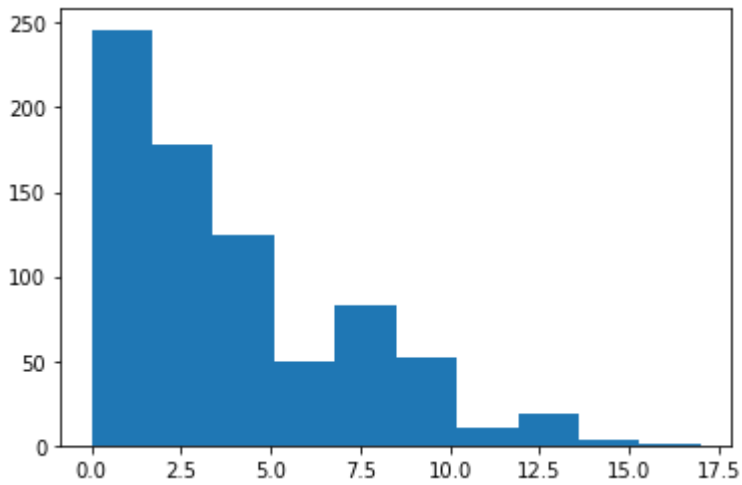
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	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 [18]:

```
1 plt.hist(df.Pregnancies)
2
```

Out[18]:

```
(array([246., 178., 125.,  50.,  83.,  52.,  11.,  19.,   3.,   1.]),
 array([ 0. ,  1.7,  3.4,  5.1,  6.8,  8.5, 10.2, 11.9, 13.6, 15.3, 1
7. ]),
 <BarContainer object of 10 artists>)
```

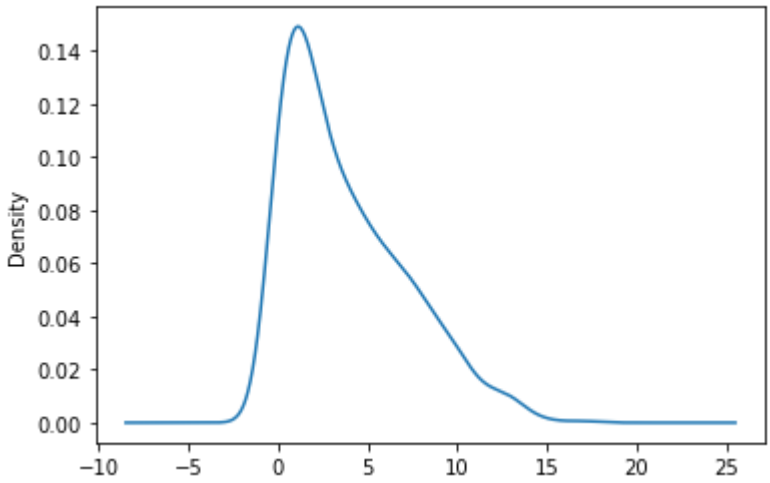


In [19]:

```
1 df.Pregnancies.plot(kind='density')
```

Out[19]:

```
<AxesSubplot:ylabel='Density'>
```



Using .iloc and .loc to separate Input and Ouput variables

In [20]:

```
1 X = df.iloc[:,0:8]
```

In [21]:

```
1 X
```

Out[21]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctic
0	6	148	72	35	0	33.6	0.6
1	1	85	66	29	0	26.6	0.3
2	8	183	64	0	0	23.3	0.6
3	1	89	66	23	94	28.1	0.1
4	0	137	40	35	168	43.1	2.2
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.1
764	2	122	70	27	0	36.8	0.3
765	5	121	72	23	112	26.2	0.2
766	1	126	60	0	0	30.1	0.3
767	1	93	70	31	0	30.4	0.3

768 rows × 8 columns

In [22]:

```
1 Y = df.iloc[:,8]
```

In [23]:

```
1 Y
```

Out[23]:

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

In [35]:

```
df.columns
```

Out[35]:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'In
sulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Rescale Data

In [24]:

```
# Rescale data (between 0 and 1)
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import MinMaxScaler


array = df.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX = scaler.fit_transform(X)
# summarize transformed data
#set_printoptions(precision=3)

#print(rescaledX[0:5,:])
print(rescaledX)
```

```
[[0.35294118 0.74371859 0.59016393 ... 0.50074516 0.23441503 0.4833333
3]
 [0.05882353 0.42713568 0.54098361 ... 0.39642325 0.11656704 0.1666666
7]
 [0.47058824 0.91959799 0.52459016 ... 0.34724292 0.25362938 0.1833333
3]
 ...
 [0.29411765 0.6080402  0.59016393 ... 0.390462   0.07130658 0.15
]
 [0.05882353 0.63316583 0.49180328 ... 0.4485842  0.11571307 0.4333333
3]
 [0.05882353 0.46733668 0.57377049 ... 0.45305514 0.10119556 0.0333333
3]]
```

In [25]:

```
rescaledXDF = pd.DataFrame(rescaledX, columns =['Pregnancies', 'Glucose', 'BloodPre
      'BMI', 'DiabetesPedigreeFunction', 'Age'])
```

In [26]:

```
1 rescaledXDF.head()
```

Out[26]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeF
0	0.352941	0.743719	0.590164	0.353535	0.000000	0.500745	0.
1	0.058824	0.427136	0.540984	0.292929	0.000000	0.396423	0.
2	0.470588	0.919598	0.524590	0.000000	0.000000	0.347243	0.
3	0.058824	0.447236	0.540984	0.232323	0.111111	0.418778	0.
4	0.000000	0.688442	0.327869	0.353535	0.198582	0.642325	0.

In [27]:

```
1 df.head()
```

Out[27]:

	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

In [28]:

```
1 rescaledXDF.describe()
```

Out[28]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	0.226180	0.607510	0.566438	0.207439	0.094326	0.476790	
std	0.198210	0.160666	0.158654	0.161134	0.136222	0.117499	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.058824	0.497487	0.508197	0.000000	0.000000	0.406855	
50%	0.176471	0.587940	0.590164	0.232323	0.036052	0.476900	
75%	0.352941	0.704774	0.655738	0.323232	0.150414	0.545455	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

In [ ]:

```
1
```

In [29]:

```
1 df.describe()
```

Out[29]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
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 [ ]:

```
1
```

Standardize Data

In [30]:

```
# Standardize data (0 mean, 1 stdev)
from sklearn.preprocessing import StandardScaler
from pandas import read_csv
from numpy import set_printoptions

#df = read_csv('c://datasets/diabetes.csv')

array = df.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = StandardScaler().fit(X)
reStandardX = scaler.transform(X)
# summarize transformed data
#set_printoptions(precision=3)
#print(rescaledX[0:5,:])

print(reStandardX)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
```

In [31]:

```
reStandardXDF = pd.DataFrame(reStandardX, columns =['Pregnancies', 'Glucose', 'B
       'BMI', 'DiabetesPedigreeFunction', 'Age'])
```

In [32]:

```
1 reStandardXDF
```

Out[32]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedig
0	0.639947	0.848324	0.149641	0.907270	-0.692891	0.204013	
1	-0.844885	-1.123396	-0.160546	0.530902	-0.692891	-0.684422	
2	1.233880	1.943724	-0.263941	-1.288212	-0.692891	-1.103255	
3	-0.844885	-0.998208	-0.160546	0.154533	0.123302	-0.494043	
4	-1.141852	0.504055	-1.504687	0.907270	0.765836	1.409746	
...	...	...	...	...	...	...	
763	1.827813	-0.622642	0.356432	1.722735	0.870031	0.115169	
764	-0.547919	0.034598	0.046245	0.405445	-0.692891	0.610154	
765	0.342981	0.003301	0.149641	0.154533	0.279594	-0.735190	
766	-0.844885	0.159787	-0.470732	-1.288212	-0.692891	-0.240205	
767	-0.844885	-0.873019	0.046245	0.656358	-0.692891	-0.202129	

768 rows × 8 columns

In [33]:

```
1 reStandardXDF.describe()
```

Out[33]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
count	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02
mean	2.544261e-17	3.614007e-18	-1.327244e-17	7.994184e-17	-3.556183e-17	2.295979e-16
std	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00
min	-1.141852e+00	-3.783654e+00	-3.572597e+00	-1.288212e+00	-6.928906e-01	-4.060474e+00
25%	-8.448851e-01	-6.852363e-01	-3.673367e-01	-1.288212e+00	-6.928906e-01	-5.955785e-01
50%	-2.509521e-01	-1.218877e-01	1.496408e-01	1.545332e-01	-4.280622e-01	9.419788e-04
75%	6.399473e-01	6.057709e-01	5.632228e-01	7.190857e-01	4.120079e-01	5.847705e-01
max	3.906578e+00	2.444478e+00	2.734528e+00	4.921866e+00	6.652839e+00	4.455807e+00

In [34]:

```
1 df.describe()
```

Out[34]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
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 [ ]:

```
1
```

In [ ]:

```
1
```

Normalize Data

In [35]:

```
# Normalize data (length of 1)
from sklearn.preprocessing import Normalizer
from pandas import read_csv
from numpy import set_printoptions

array = df.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = Normalizer().fit(X)
normalizedX = scaler.transform(X)
# summarize transformed data
#set_printoptions(precision=3)
#print(normalizedX[0:5,:])
print(normalizedX)
```

```
[[0.03355237 0.82762513 0.40262844 ... 0.18789327 0.00350622 0.2796030
8]
 [0.008424   0.71604034 0.55598426 ... 0.22407851 0.00295683 0.2611441
2]
 [0.04039768 0.92409698 0.32318146 ... 0.11765825 0.00339341 0.1615907
3]
 ...
 [0.02691539 0.65135243 0.38758161 ... 0.14103664 0.00131885 0.1614923
4]
 [0.00665306 0.83828547 0.39918356 ... 0.20025708 0.00232192 0.3126937
9]
 [0.00791454 0.73605211 0.55401772 ... 0.24060198 0.00249308 0.1820343
9]]
```

In [36]:

```
normalizedX = pd.DataFrame(normalizedX, columns =['Pregnancies', 'Glucose', 'Blo
       'BMI', 'DiabetesPedigreeFunction', 'Age'])

print(normalizedX)
```

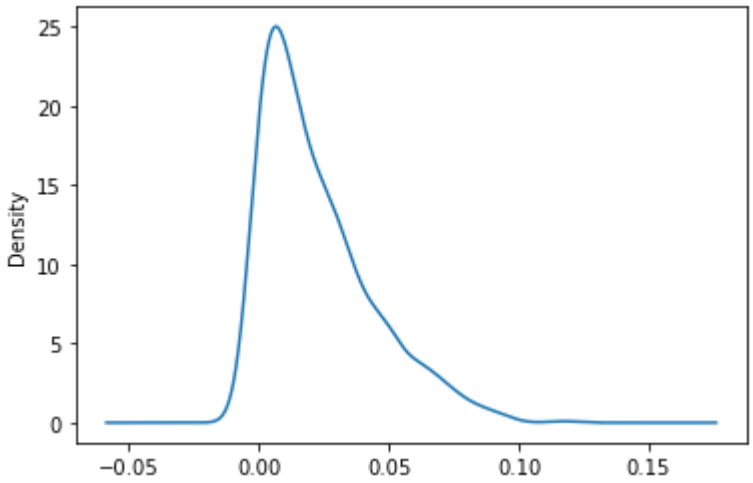
```
     Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin
BMI  \
0       0.033552  0.827625       0.402628       0.195722  0.000000  0.
187893
1       0.008424  0.716040       0.555984       0.244296  0.000000  0.
224079
2       0.040398  0.924097       0.323181       0.000000  0.000000  0.
117658
3       0.006612  0.588467       0.436392       0.152076  0.621527  0.
185797
4       0.000000  0.596386       0.174127       0.152361  0.731335  0.
187622
..           ...       ...            ...            ...       ...
...
763     0.042321  0.427443       0.321640       0.203141  0.761779  0.
139236
764     0.013304  0.811526       0.465629       0.179600  0.000000  0.
244788
765     0.026915  0.651352       0.387582       0.123811  0.602905  0.
141037
766     0.006653  0.838285       0.399184       0.000000  0.000000  0.
200257
767     0.007915  0.736052       0.554018       0.245351  0.000000  0.
240602

     DiabetesPedigreeFunction       Age
0                    0.003506  0.279603
1                    0.002957  0.261144
2                    0.003393  0.161591
3                    0.001104  0.138852
4                    0.009960  0.143655
..                        ...       ...
763                  0.000724  0.266623
764                  0.002262  0.179600
765                  0.001319  0.161492
766                  0.002322  0.312694
767                  0.002493  0.182034

[768 rows x 8 columns]
```

In [37]:

```
1 normalizedX.Pregnancies.plot(kind='density')
```

Out[37]:

<AxesSubplot:ylabel='Density'>

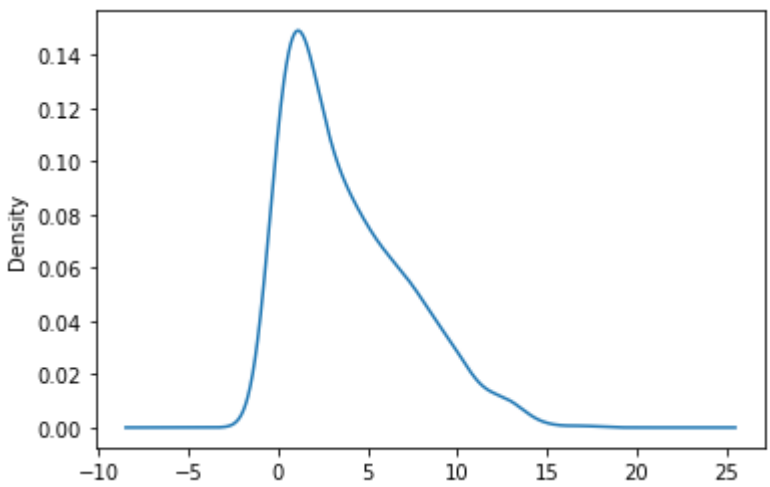


In [38]:

```
1 df.Pregnancies.plot(kind='density')
```

Out[38]:

<AxesSubplot:ylabel='Density'>

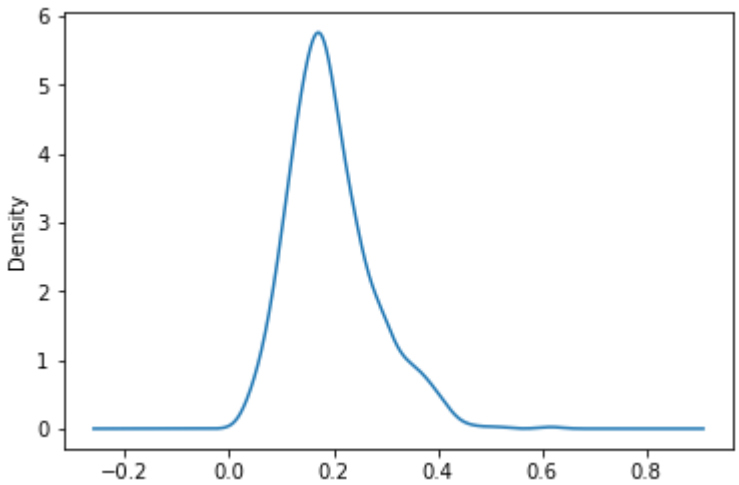


In [58]:

```
normalizedX.Age.plot(kind='density')
```

Out[58]:

<AxesSubplot:ylabel='Density'>

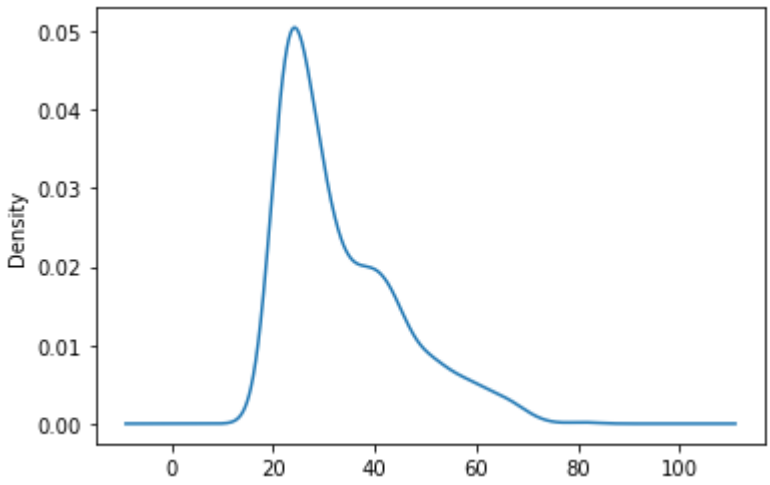


In [39]:

```
df.Age.plot(kind='density')
```

Out[39]:

<AxesSubplot:ylabel='Density'>



Feature Selection For Machine Learning

```
#### Data features and their interactions can affect model performance
Features that are irrelevant or partially relevant can negatively impact
model performance.

Feature selection is the selection of those features in your data that
contribute most to the prediction variable or output in which you are
interested.

Some techniques for feature selection:

```

```
1. Univariate Selection.
2. Recursive Feature Elimination.
3. Principle Component Analysis.
4. Feature Importance.

```

In [ ]:

```

```

In [ ]:

```

```