Lab 1: Data Loading, Summary, and Visualization

1. Create Dataframe

In [1]:

```
import numpy
import pandas
myarray = numpy.array([[1,2,3],[4,5,6]])
rownames = ['a','b']
colnames=['f1','f2','f3']
mydataframe = pandas.DataFrame(myarray, index = rownames, columns=colnames)
print(mydataframe)
```

```
   f1  f2  f3
a   1   2   3
b   4   5   6
```

Change the type of data

In [2]:

```
import numpy
import pandas
myarray = numpy.array([['a','sandhya',9.6],[4,'shreya',6.5]])
rownames = ['r1','r2']
colnames=['f1','f2','f3']
mydataframe = pandas.DataFrame(myarray, index = rownames, columns=colnames)
print(mydataframe)
```

```
   f1       f2   f3
r1  a  sandhya  9.6
r2  4   shreya  6.5
```

2. Load csv file using pandas from a specific path or url

Copy dataset given in https://www.kaggle.com/uciml/pima-indians-diabetes-database to your local folder.

In [3]:

```
from pandas import read_csv
path='diabetes.csv'
data=read_csv(path)
print (data.shape) #to know size of the data
```

```
(768, 9)
```

In [4]:

```
from pandas import read_csv
url='https://gist.githubusercontent.com/manavpatadia
/a68d7a7923f32c556106dd396a5e33c8
/raw/56554151c5bee7c6ba2d028eafb93925ade32594/diabetes.csv'
data=read_csv(url)
colnames=
['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI',
'DiabetesPedigreeFuntion','Age','Outcome']
print (data)
```

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

     DiabetesPedigreeFunction  Age  Outcome  
0                       0.627   50        1  
1                       0.351   31        0  
2                       0.672   32        1  
3                       0.167   21        0  
4                       2.288   33        1  
..                        ...  ...      ...  
763                     0.171   63        0  
764                     0.340   27        0  
765                     0.245   30        0  
766                     0.349   47        1  
767                     0.315   23        0  

[768 rows x 9 columns]
```

3. To get statistical summary of the data

a. Get the data statistics

In [5]:

```
description = data.describe()
print(description)
```

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

              BMI  DiabetesPedigreeFunction         Age     Outcome  
count  768.000000                768.000000  768.000000  768.000000  
mean    31.992578                  0.471876   33.240885    0.348958  
std      7.884160                  0.331329   11.760232    0.476951  
min      0.000000                  0.078000   21.000000    0.000000  
25%     27.300000                  0.243750   24.000000    0.000000  
50%     32.000000                  0.372500   29.000000    0.000000  
75%     36.600000                  0.626250   41.000000    1.000000  
max     67.100000                  2.420000   81.000000    1.000000  
```

Here 25%, 50%, gives % of data that falls below a given corresponding value in each column.

b. Size of matrix

In [6]:

```
print(data.shape)
```

```
(768, 9)
```

c. Peek at data/ used to get the first n rows

In [7]:

```
print(data.head(4))
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
```

d. Group on the basis of a particular attribute

In [8]:

```
print(data.groupby('Outcome').size())
```

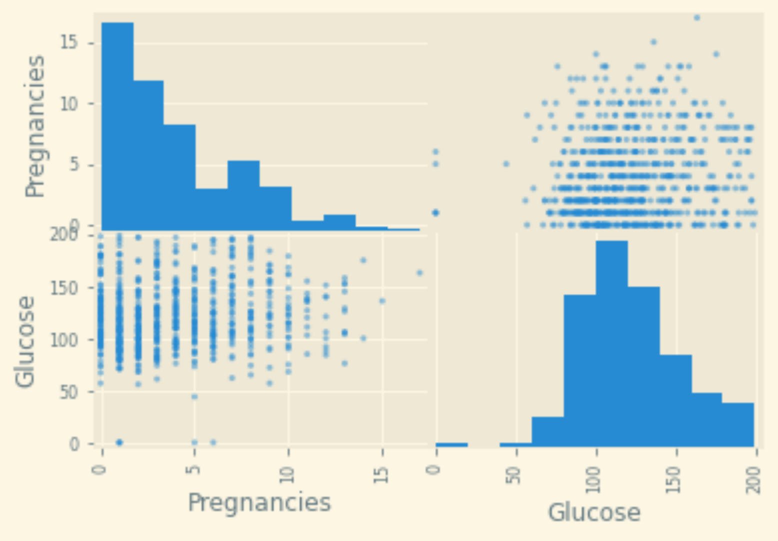
```
Outcome
0    500
1    268
dtype: int64
```

4. Data visualization

For plotting pairs of attributes as scattered plot, specify the attributes to be plotted explicitly

In [9]:

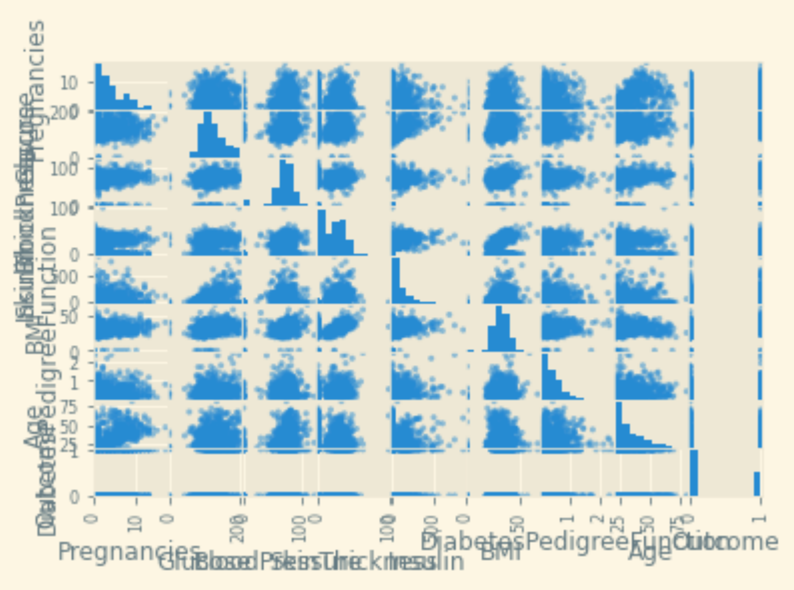
```
import matplotlib.pyplot as plt
plt.style.use("Solarize_Light2")
import pandas
from pandas.plotting import scatter_matrix
scatter_matrix(data[['Pregnancies','Glucose']])
plt.show()
```

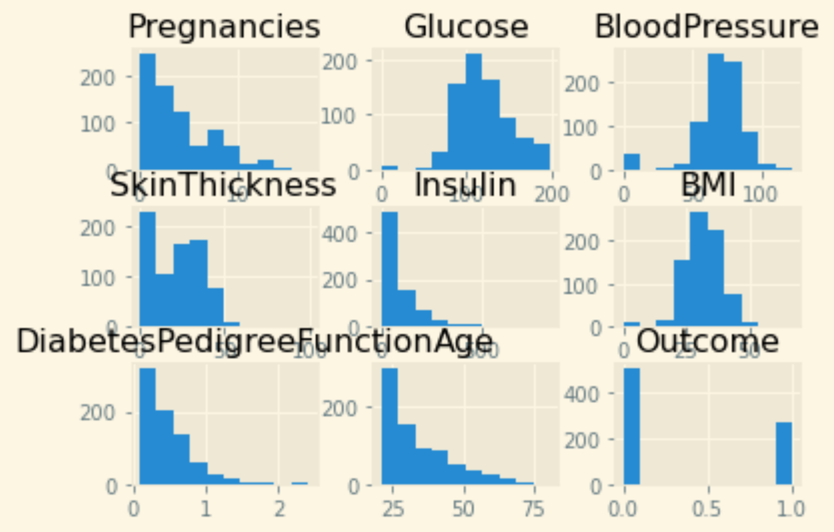


For plotting all pairs of attributes in data

In [10]:

```
import matplotlib.pyplot as plt
import pandas
from pandas.plotting import scatter_matrix
scatter_matrix(data) #scatter plot
plt.show()
data.hist() #histogram
plt.show()
```





5. Standardization of dataset

In [11]:

```
from sklearn.preprocessing import StandardScaler
import pandas
import numpy
arr=data.values #convert data frame to array
X=arr[:,0:8] #split columns
Y=arr[:,8]
scaler=StandardScaler().fit(X) #fit data for standardization
rescaledX=scaler.transform(X) #convert the data as per (x-μ)/σ
numpy.set_printoptions(precision=3)
print(rescaledX[0:2,:])
print(X[0:2,:])
```

```
[[ 0.64    0.848  0.15    0.907 -0.693  0.204  0.468  1.426]
 [-0.845 -1.123 -0.161  0.531 -0.693 -0.684 -0.365 -0.191]]
[[  6.    148.     72.     35.      0.     33.6     0.627  50.    ]
 [  1.     85.     66.     29.      0.     26.6     0.351  31.    ]]
```

6. Normalizing a column in pandas

Create the dataframe

In [12]:

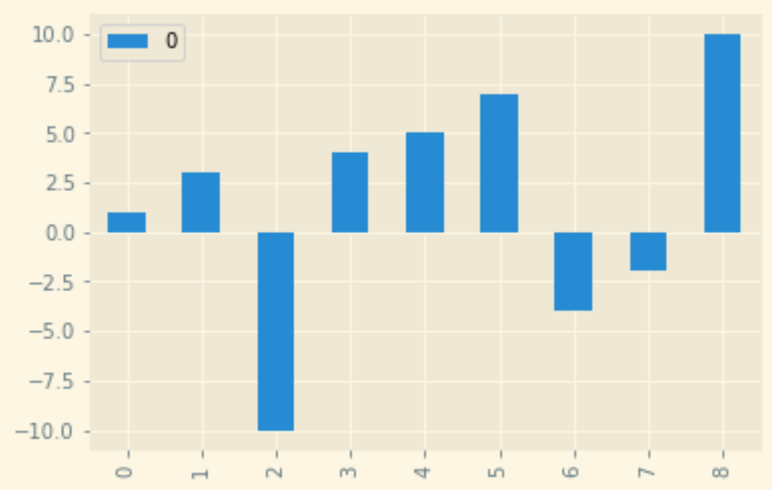
```
myarray=numpy.array([1,3,-10,4,5,7,-4,-2,10])
mydataframe = pandas.DataFrame(myarray)
print(mydataframe)
```

```
    0
0   1
1   3
2 -10
3   4
4   5
5   7
6  -4
7  -2
8  10
```

plot the data

In [13]:

```
mydataframe.plot(kind='bar')
plt.show()
```



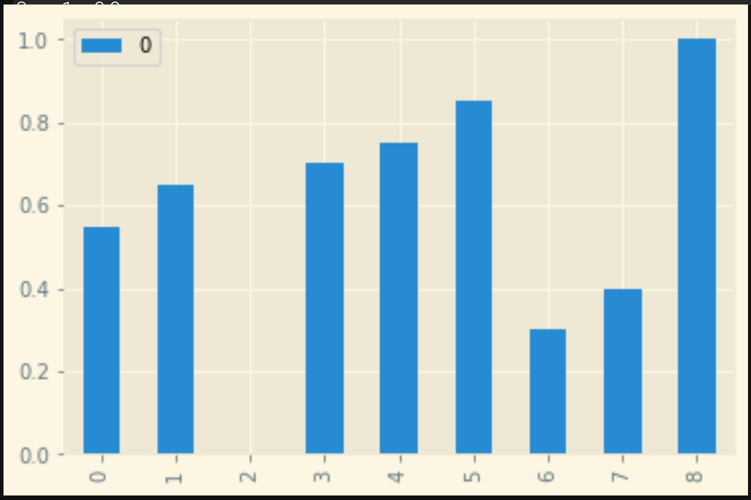
Plot normalized data

In [14]:

```
from sklearn import preprocessing
fl_x=mydataframe.values.astype(float)
#fl_x=mydataframe[['f1']].values.astype(float) #If specific feature name is to be converted
min_max_scaler=preprocessing.MinMaxScaler()
X_scaled=min_max_scaler.fit_transform(fl_x)
df_normalized=pandas.DataFrame(X_scaled)
print(df_normalized)
df_normalized.plot(kind='bar')
plt.show()
```

```
      0
0  0.55
1  0.65
```

```
2   0.00
3   0.70
4   0.75
5   0.85
6   0.30
7   0.40
```



In [ ]:

```
from sklearn import preprocessing
fl_x=X
#fl_x=mydataframe[['f1']].values.astype(float) #If specific feature name is to be converted
min_max_scaler=preprocessing.MinMaxScaler()
X_scaled=min_max_scaler.fit_transform(fl_x)
df_normalized=pandas.DataFrame(X_scaled)
print(df_normalized)
df_normalized.plot(kind='bar')
plt.show()
```

```
            0         1         2         3         4         5         6  \
0    0.352941  0.743719  0.590164  0.353535  0.000000  0.500745  0.234415
1    0.058824  0.427136  0.540984  0.292929  0.000000  0.396423  0.116567
2    0.470588  0.919598  0.524590  0.000000  0.000000  0.347243  0.253629
3    0.058824  0.447236  0.540984  0.232323  0.111111  0.418778  0.038002
4    0.000000  0.688442  0.327869  0.353535  0.198582  0.642325  0.943638
..        ...       ...       ...       ...       ...       ...       ...
763  0.588235  0.507538  0.622951  0.484848  0.212766  0.490313  0.039710
764  0.117647  0.613065  0.573770  0.272727  0.000000  0.548435  0.111870
765  0.294118  0.608040  0.590164  0.232323  0.132388  0.390462  0.071307
766  0.058824  0.633166  0.491803  0.000000  0.000000  0.448584  0.115713
767  0.058824  0.467337  0.573770  0.313131  0.000000  0.453055  0.101196

            7
0    0.483333
1    0.166667
2    0.183333
3    0.000000
4    0.200000
..        ...
763  0.700000
764  0.100000
765  0.150000
766  0.433333
767  0.033333

[768 rows x 8 columns]
```