Step 1. Loading Dataset

Load the `Prima Indians Onset of Diabates` dataset. It is a standard machine learning dataset available for free download from the UCI Machine Learning repository. It describes patient medical record data for Pima Indians and whether they had an onset of diabetes within five years. It is a binary classification problem (onset of diabetes as 1 or not as 0)

- NTP: Number of times pregnant.
- PGC: Plasma glucose concentration a 2 hours in an oral glucose tolerance test. 3. Diastolic blood pressure (mm Hg).
- TSFT: Triceps skin fold thickness (mm).
- 2hSI: 2-Hour serum insulin (mu U/ml).
- BMI: Body mass index.
- DPF: Diabetes pedigree function.
- Age: Age (years).
- OnDiab: Class, onset of diabetes within five years.

- Set the dataset columns names to `['NTP', 'PGC','DBP','TSFT','2hSI','BMI','DPF','Age','OnDiab']`.
- Print the size of the data set.
- Print the first 10 observations of your data set

In [5]:

```
import pandas as pd
dataset = pd.read_csv('diabetes.csv')
dataset.columns = ['NTP', 'PGC','DBP','TSFT','2hSI','BMI','DPF','Age','OnDiab']
dataset.shape
```

Out[5]:

```
(767, 9)
```

In [6]:

```
dataset.head(10)
```

Out[6]:

	NTP	PGC	DBP	TSFT	2hSI	BMI	DPF	Age	OnDiab
0	1	85	66	29	0	26.6	0.351	31	0
1	8	183	64	0	0	23.3	0.672	32	1
2	1	89	66	23	94	28.1	0.167	21	0
3	0	137	40	35	168	43.1	2.288	33	1
4	5	116	74	0	0	25.6	0.201	30	0
5	3	78	50	32	88	31.0	0.248	26	1
6	10	115	0	0	0	35.3	0.134	29	0
7	2	197	70	45	543	30.5	0.158	53	1
8	8	125	96	0	0	0.0	0.232	54	1
9	4	110	92	0	0	37.6	0.191	30	0

Step 2: Data type and description for each attribute

- Print the data type for each attribute using `dtypes` method of a pandas data frame
- Descriptive statistics can give you great insight into the properties of each attribute. Often you can create more summaries than you have time to review. The `describe()` function on the Pandas DataFrame lists 8 statistical properties of each attribute. They are: Count, Mean, Standard Deviation, Minimum Value, 25th Percentile, 50th Percentile (Median), 75th Percentile, Maximum Value.
- On classication problems you need to know how balanced the class values are. Highly imbalanced problems (a lot more observations for one class than another) are common and may need special handling in the data preparation stage of your project. You can quickly get an idea of the distribution of the `OnDiab` attribute in Pandas. Grup your data by `OnDiab` attribute and use the `size` method to count the number of different values of `class` attribute.

In [14]:

```
types=dataset.dtypes
print(types)
```

```
NTP          int64
PGC          int64
DBP          int64
TSFT         int64
2hSI         int64
BMI        float64
DPF        float64
Age          int64
OnDiab       int64
dtype: object
```

In [15]:

```
dataset.describe()
```

Out[15]:

	NTP	PGC	DBP	TSFT	2hSI	BMI	DPF	
count	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000	767.
mean	3.842243	120.859192	69.101695	20.517601	79.903520	31.990482	0.471674	33.
std	3.370877	31.978468	19.368155	15.954059	115.283105	7.889091	0.331497	11.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243500	24.
50%	3.000000	117.000000	72.000000	23.000000	32.000000	32.000000	0.371000	29.
75%	6.000000	140.000000	80.000000	32.000000	127.500000	36.600000	0.625000	41.
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.

In [16]:

```
dataset.groupby('OnDiab').size()
```

Out[16]:

```
OnDiab
0    500
1    267
dtype: int64
```

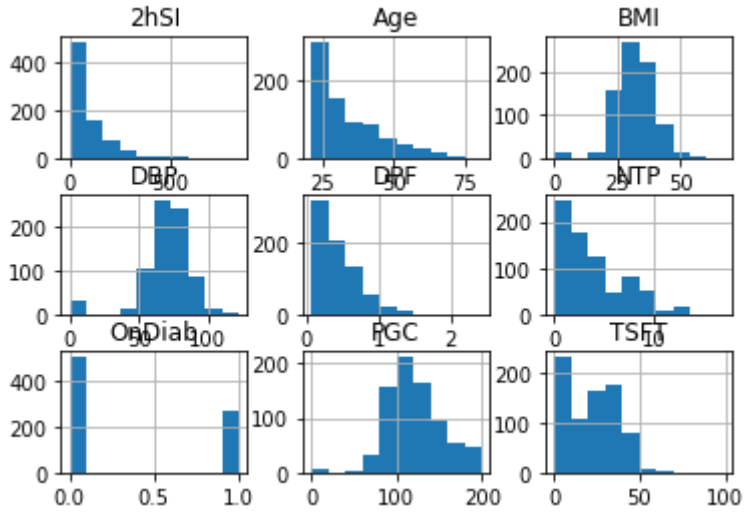
Step 3: Draw the histogram of your data attributes.

A fast way to get an idea of the distribution of each attribute is to look at histograms. Histograms group data into bins and provide you a count of the number of observations in each bin. From the shape of the bins you can quickly get a feeling for whether an attribute is Gaussian, skewed or even has an exponential distribution. It can also help you see possible outliers.

- Use `hist()` method of pandas data frame to plot the histograms.

In [18]:

```
dataset.hist();
```



Step 4: Running a Classifier

- Separate your dataset into feature set `X` and target variable `y`. Your target variable is `OnDiab`.
- Split your dataset into train and test datasets, keep the test dataset size as 0.25 using `test_size` parameter of `train_test_split`. Set the random seed top 7 using `random_state` parameter of `train_test_split`. Make a stratified split.
- Train `KNearestNeighbor` classifer on your train dataset and print the score on the the test dataset. Set number of neighboirs to 5.

In [23]:

```
X = dataset.drop("OnDiab", axis=1)
y = dataset.OnDiab
```

In [24]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,stratify=y, test_size=0.25
, random_state=7)
```

In [25]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5).fit(X_train, y_train)
print("Test: {:.3f}".format(knn.score(X_test, y_test)))
```

```
Test: 0.714
```

Step 5 : Grid Search

- Import `GridSearchCV` from `sklearn.model_selection`
- Split your data into train and test datasets
- For `neighbors=1 to 30` , compute `GridSearchCV` for train dataset with kfold=10.
- Print the best cross validation score
- Priont the best parameter
- Print the test score

In [27]:

```
from sklearn.model_selection import GridSearchCV
import numpy as np

param_grid = {'n_neighbors':  np.arange(1, 30, 2)}
grid = GridSearchCV(KNeighborsClassifier(), param_grid=param_grid, cv=10,return_tra
in_score=True)
grid.fit(X_train, y_train)
print("best mean cross-validation score: {:.3f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))

print("test-set score: {:.3f}".format(grid.score(X_test, y_test)))
```

```
best mean cross-validation score: 0.727
best parameters: {'n_neighbors': 29}
test-set score: 0.766
```

In [ ]:

```

```