CSCI 6364 - Machine Learning

Project 1 - Pima Indians Diabetes

Student: Shifeng Yuan

GWid: G32115270

Language: Python

Resource: Pima Indians Diabetes from Kaggle

In [91]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import seaborn as sns
```

1. Dataset Details

The dataset includes data from 768 women with 8 characteristics, in particular:

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)

Inspect the Dataset

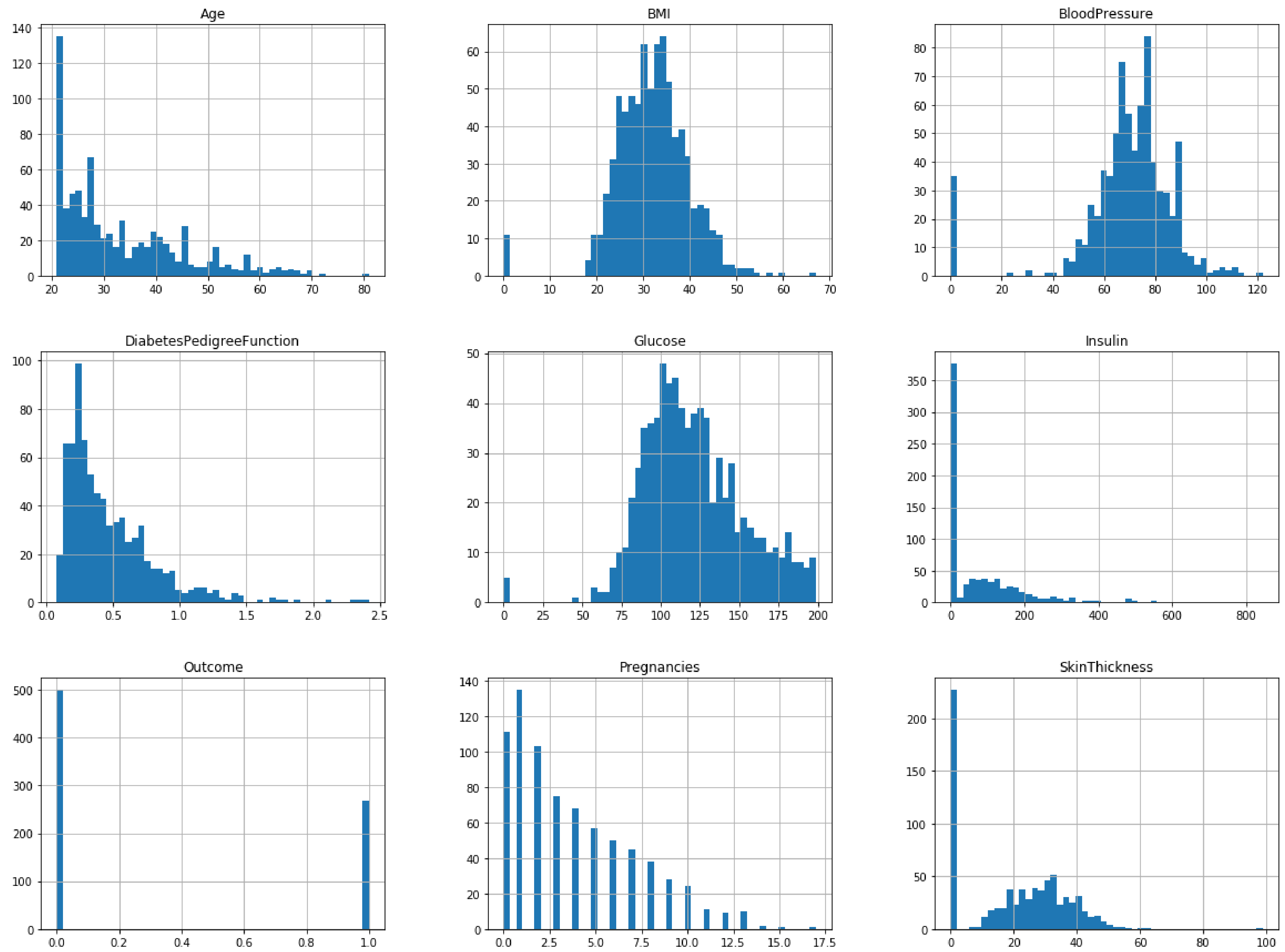
In [105]:
```
# Read the dataset and then print the head
dataset = pd.read_csv('diabetes.csv')
print( len(dataset) )
print( dataset.head() )
```

```
768
   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

   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
```

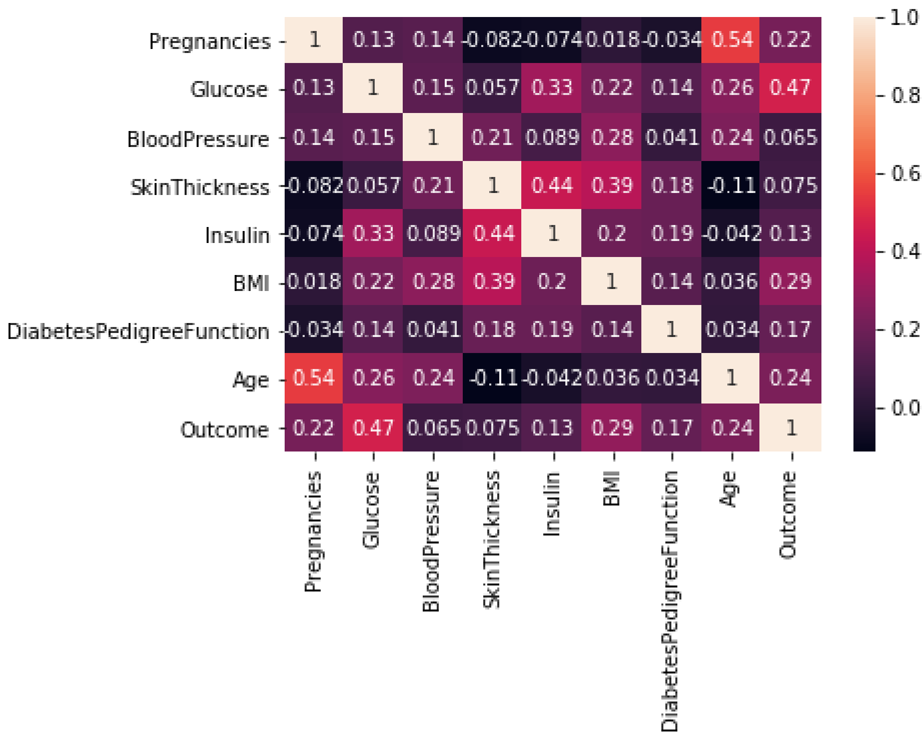
Dataset Visualization

In [93]:
```python
import matplotlib.pyplot as plt
dataset.hist(bins=50, figsize=(20, 15))
plt.show()
```



In [94]:
```python
column_x = dataset.columns[0:len(dataset.columns) - 1]
corr = dataset[dataset.columns].corr()
sns.heatmap(corr, annot = True)
```

Out[94]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1a4e4ac8>
```



Data Spliting

Usually, we divide our dataset into 2 to 3 parts. Here, I split the dataset into training data (80%) and testing data(20%)

In [95]:
```python
#split dataset
# x is the columns
X = dataset.iloc[:, 0:8]
# y is the last column which is the result
y = dataset.iloc[:, 8]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.2)
```

2. Algorithm Description

As we see in the dataset, there are some data of zeroes and null, and they will negatively influence the accuracy of our traning. In this case, I decide to replace them with the median value of the columns they locate.

In [96]:
```
# replace zeroes
zero_not_accepted = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
# For the columns that should not contain zeroes and null, I replace all the zeroes with NaN
# and then change all NaN to the median of that column
for column in zero_not_accepted:
    dataset[column] = dataset[column].replace(0, np.NaN)
    mean = int(dataset[column].mean(skipna=True))
    dataset[column] = dataset[column].replace(np.NaN, mean)
```

We should also standardize the dataset, here, I use the sklearn.preprocessing.StandardScaler() to standardize the dataset.This package use a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1. Below is the format:

Standardization:

$$z = \frac{x - \mu}{\sigma}$$

with mean:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} (x_i)$$

and standard deviation:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2}$$

In [97]:
```
# Standardization
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Selection of K

The selection of value K is important for KNN, usually, we make K the square root of the size of the test sample.

In [98]:
```
import math
math.sqrt(len(y_test))
```

Out[98]:
```
12.409673645990857
```

The package sklearn.neighbors.KNeighborsClassifier implementing the K-nearest Neighbors classification.

In general, the k is better to be an odd number, so we make it 11.
Using the sklearn KNeighborsClassifier package, define the metric method as euclidean. This dataset is not too big, so we can just use a brute force algorithm.

In [99]:
```
#define the model: Init knn
classifier = KNeighborsClassifier(n_neighbors = 11, algorithm = 'brute', p = 2, metric = 'euclidean')
```

Fit the model using X as training data and y as target values

In [100]:
```
#Fit model
classifier.fit(X_train, y_train)
```

Out[100]:
```
KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='euclidean',
           metric_params=None, n_jobs=1, n_neighbors=11, p=2,
           weights='uniform')
```

3. Algorithm Results

In [101]:
```
#predict the test result
y_pred = classifier.predict(X_test)
```

In [102]:
```
#Evaluate the model
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[92 15]
 [20 27]]
```

The table below shows the true positive, true negative, false positive and false negatives. In total 153 test samples, we get 92 true positives and 27 true negatives, which makes the accuracy score be 0.7727272727272727.

	TRUE	FALSE
Positive	92	15
Negative	20	74

In [103]:
```
print(accuracy_score(y_test, y_pred))
```

```
0.7727272727272727
```

4. Runtime

For d dimension, we need O(d) runtime to compute one distance between two data, so computing all the distance between one data to other data needs O(nd) runtime, then we need O(kn) runtime to find the K nearest neibors, so, in total, it takes O(dn+kn) runtime for the classifier to classify the data.

In [106]:
```
import time
start = time.time()
classifier = KNeighborsClassifier(n_neighbors = 11, p = 2, algorithm='brute', metric = 'euclidean')
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
end = time.time()
print(end-start)
```

```
0.031829118728637695
```

As is shown above, the "wall-clock" of the runtime is about 0.0318s