CSC 522 - Fall 2018

HOMEWORK 1

Homework Group 43

Name	**UnityID**
Aman Chauhan	achauha3
Khantil Choksi	khchoksi

Problem 1: Classifying attributes:

	Attribute Name	binary / discrete / continuous	nominal / ordinal / interval or ratio
(a)	Hair color (Black, Blonde, Red)	discrete	nominal
(b)	Level of agreement (yes, maybe, no)	discrete	ordinal (Yes - can be considered as maximum level of agreement; maybe - can be considered as neutral level of agreement and no - can be considered as minimum level of agreement)
(c)	Income earned in a week	continuous	ratio
(d)	Celsius temperature	continuous	interval
(e)	Genotype (Bb, bb, BB, bB)	discrete	nominal
(f)	ISBN numbers for books	discrete	nominal
(g)	Time in terms of AM or PM	discrete (like 6AM, 7AM, 2PM, hence there are only finite values in a day)	ordinal (You can order these times as 8AM comes before 11PM for the same day)
(h)	Waiting number for restaurant	discrete	ordinal
(i)	Years of work experience	continuous (1.5, 2.5)	ratio
(j)	Categorization of clothing (hat, shirt, pants, shoes)	discrete	nominal
(k)	Angles as measured in degrees between 0 and 360	continuous	ratio
(l)	Ratings of movies (G, PG, R)	discrete	ordinal (based on level of violence and profanity in language)

(m)	Coat check number	discrete	nominal (follows only property of distinctness e.g. if the number is "equal to" or "not equal to")

Problem 2: Term Frequency - Inverse Document Frequency

(A) Min and Max values for the functions

For equation (1), tf_{ij}' :

- Maximum value = **p * log m**
  If out of 'm' documents, word occured only in 1 document with p frequency then $df_i = 1$.
- Minimum value = **0**
  If a word occurs at least 1 time in each and every document then $df_i = m$. Therefore, log $\frac{m}{dfi} = 0$.

For equation (2), tf_{ij}''

- Maximum value = $p * log \frac{\sum_{k=1}^{m} d_k}{p}$
  If i^{th} word occurs only in j^{th} document and it is the only word occurring p times in j^{th} document then we will get maximum inverse document frequency.
- Minimum value = **0**
  If all the documents contains the i^{th} word. Therefore,

$$\sum_{k=1}^{m} d_k = \sum_{k=1}^{df_i} d_k \Rightarrow p * log\ 1 \Rightarrow 0$$

(B) Importance of TF-IDF

Both tf_{ij}' and tf_{ij}'' are used in natural language processing for classification of documents, information retrieval and as a tool to compare similarity between documents.

TF_{ij}' is used in scenarios where the length of all documents in the corpus are comparable. TF_{ij}'' is used in scenarios where the length of documents in the corpus vary a lot.

TF_{ij}' does not take into consideration the length of document in which that word occurs in the IDF portion. Thus it gives equal weightage to words which are unique in both small and long documents. TF_{ij}'' considers the length of the document in which that word is occuring. Thus, the IDF portion of this equation gives more weightage to terms that are unique in shorter documents.

Problem 3:

(A) Answer: Stratified Sampling
Reason: The sampling method depends on the overall population of the given data. Here the patients having albinism and normal skin is proportional of 5% to 95%.

- Random sampling involves the random selection of datapoint from the entire population; so each possible sample is equally likely to occur. If we choose to get a random sample of size 100 over the entire population then there is chance that the random sample turns out to be not well balanced across the patients having albinotic vs. normal skin and hence it is biased causing a significant error in estimation.
- In contrast, stratified sampling divides members of the population into homogeneous subgroups before sampling. A random sample is taken from each subgroup in direct proportion to the size of the stratum compared to the population. The sample subsets are then combined to create a final random sample.

- Here the population density varies greatly within a data region, stratified sampling will ensure that estimates can be made with equal accuracy in different parts of the region.

- So, after stratified sampling,
  number of albinotic patients sampled = $\frac{26150}{523000} * 1000 = 50$
  number of normal skin patients sampled = $\frac{496850}{523000} * 1000 = 950$

(B) The recorded systolic blood pressure (SBP) of 250 can be considered an **outlier** because this is a *true recorded value* which is significantly away from the critical value.
Reason 1 - If we consider a distribution of SBP, then the question suggests that an SBP of 180 would be critical, and thus can be considered as an upper extreme value of the distribution. The value of 250 is even beyond that upper extreme. Hence this is either noise or an error.
Reason 2 - This value was recorded by a person using an instrument, which means that the observation was supervised, ruling out the possibility of error generated by the machine.

Q4

Import the libraries

```
import random
import math
import sys
import os
```

Helper Functions

Function to print the matrix

```
def pprint(mat):
    pad = '{:>' + str(len(str(max([max(mat[i]) for i in range(len(mat))])))) + '}'
    print('[' + '\n '.join(['[' + ', '.join([pad.format(mat[i][j]) for j in range(len(mat[0]))]) + ']' for i in range(len(mat))]) + ']')
```

Function to create identity matrix of size 'dim'

```
def identity_matrix(dim):
    return [[1 if j==i else 0 for j in range(dim)] for i in range(dim)]
```

Function to set the column to a particular value

```
def column_manipulate(mat, col, val):
    for row in mat:
        row[col] = val
    return mat
```

Function to sum all values in the matrix

```
def matrix_sum(mat):
    cnt = 0
    for row in mat:
        for col in row:
            cnt += col
    return cnt
```

Transpose any given MxN matrix

```
def transpose(mat):
    result = [[None for j in range(len(mat))] for i in range(len(mat[0]))]
    for i in range(len(mat)):
        for j in range(len(mat[i])):
            result[j][i] = mat[i][j]
    return result
```

Calculate sum for a given row and sum of diagonal elements

```
def row_diagonal_sum(mat, row):
    rowcnt = 0
    dcnt = 0
    for i in range(len(mat)):
        for j in range(len(mat[i])):
            if i==j:
                dcnt += mat[i][j]
            if i==row:
                rowcnt += mat[i][j]
    return rowcnt, dcnt
```

Generate NxN Gaussian Matrix for a given mean and variance

```
def gaussian_matrix(dim, mean, variance, dtype):
    if dtype=="int":
        return [[math.floor(random.gauss(mean, math.sqrt(variance))) for j
in range(dim)] for i in range(dim)]
    else:
        return [[random.gauss(mean, math.sqrt(variance)) for j in
range(dim)] for i in range(dim)]
```

Function to multiply 2 matrices

```
def matrix_multiply(a, b):
    result = [[0 for j in range(len(b[0]))] for i in range(len(a))]
    for i in range(len(a)):
        for j in range(len(b[0])):
            for k in range(len(b)):
                result[i][j] += a[i][k] * b[k][j]
    return result
```

Function to multiply individual elements of given 2 matrices

```
def multiply(a, b):
    result = [[0 for j in range(len(a[0]))] for i in range(len(a))]
    for i in range(len(a)):
        for j in range(len(a[0])):
            result[i][j] = a[i][j] * b[i][j]
    return result
```

Function to add 2 matrices

```
def add(a, b):
    result = [[0 for j in range(len(a[0]))] for i in range(len(a))]
    for i in range(len(a)):
        for j in range(len(a[0])):
            result[i][j] = a[i][j] + b[i][j]
    return result
```

Function to shift all rows up by 1

```
def row_shift(mat):
    return mat[1:] + [mat[0]]
```

Function to find covariance between 2 vectors

```
def covariance(x, y):
    mean_x = sum(x)/len(x)
    mean_y = sum(y)/len(y)
    sumo = 0.0
    for i in range(len(x)):
        sumo += (x[i] - mean_x) * (y[i] - mean_y)
    sumo /= (len(x) - 1)
    return sumo
```

(A) Generate 5x5 identity matrix

```
A = identity_matrix(5)
pprint(A)
```

```
[[1, 0, 0, 0, 0]
 [0, 1, 0, 0, 0]
 [0, 0, 1, 0, 0]
 [0, 0, 0, 1, 0]
 [0, 0, 0, 0, 1]]
```

(B) Change all elements in the 2nd column of A to 3.

```
A = column_manipulate(A, 1, 3)
pprint(A)
```

```
[[1, 3, 0, 0, 0]
 [0, 3, 0, 0, 0]
 [0, 3, 1, 0, 0]
 [0, 3, 0, 1, 0]
 [0, 3, 0, 0, 1]]
```

(C) Sum of all elements in the matrix

```
print(matrix_sum(A))
```

```
19
```

(D) Transpose the matrix A

```
A = transpose(A)
pprint(A)
```

```
[[1, 3, 0, 0, 0]
 [0, 3, 0, 0, 0]
 [0, 3, 1, 0, 0]
 [0, 3, 0, 1, 0]
 [0, 3, 0, 0, 1]]
```

(E) Calculate sum of the 3rd row, and the diagonal in the matrix A.

```
rowcnt, dcnt = row_diagonal_sum(A, 2)
print('Sum of 3rd row - ' + str(rowcnt))
print('Sum of major diagonal - ' + str(dcnt))
```

```
Sum of 3rd row - 4
Sum of major diagonal - 7
```

(F) Generate a 5*5 matrix B following Gaussian Distribution with mean 5 and variance 3.

```
B = gaussian_matrix(5, 5, 3, "float")
pprint(B)
```

```
[[6.215699316621983, 5.3314324181394275, 3.160592195218215,
5.731340893558323, 3.8761641182554536]
 [2.726979912172165, 3.363145178336113, 5.168241217648324,
3.3272058142040364, 4.53812449216301]
 [5.8952555626790035, 5.008129339178955, 3.3278857168552056,
4.2163819866696235, 0.2846806253144418]
 [5.760377878211342, 5.911725654130385, 3.335467243819485,
5.183137167675729, 4.836748684684982]
 [4.616059063383704, 6.42657606799423, 4.242358931555765, 4.656873001603596,
2.9403643600492413]]
```

(G)

```
C1 = multiply(matrix_multiply([[1,0,0,0,0]]*5, row_shift(B)),
identity_matrix(5))
C2 = matrix_multiply(matrix_multiply([[1,0,0,0,0], [0,0,0,0,0]], B), C1)
C3 = matrix_multiply([[0,0,0,0,0], [0,0,1,1,-1]], B)
C = add(C2, C3)
pprint(C)
```

```
[[  16.9500871765304, 17.930381230690458,  16.33470285550438,
19.069350744232608,  17.59051532069851]
 [  7.03957437750664,   4.49327892531511, 2.4209940291189254,
4.742646152741756,  2.1810649499501 82]]
```

(H)

```
D1 = multiply([[2,3,4,5,6]]*5, identity_matrix(5))
D = matrix_multiply(C, D1)
pprint(D)
```

```
[[  33.9001743530608,  53.79114369207137,  65.33881142201751,
95.34675372116304, 105.54309192419106]
 [ 14.07914875501328,  13.47983677594533,  9.683976116475701,
23.71323076370878, 13.086389699701094]]
```

(I) Covariance Matrix

```
print('X\tY\tZ')
print('-\t-\t-')
print('{}\t{}\t{}'.format(2, 6, 1))
print('{}\t{}\t{}'.format(4, 5, 3))
print('{}\t{}\t{}'.format(6, 4, 5))
print('{}\t{}\t{}'.format(8, 3, 7))
X = [2, 4, 6, 8]
Y = [6, 5, 4, 3]
Z = [1, 3, 5, 7]
print('\nCovariance Matrix')
print('\tX\tY\tZ')
print('X\t{:0.2f}\t{:0.2f}\t{:0.2f}'.format(covariance(X, X), covariance(X, Y), covariance(X, Z)))
print('Y\t{:0.2f}\t{:0.2f}\t{:0.2f}'.format(covariance(Y, X), covariance(Y, Y), covariance(Y, Z)))
print('Z\t{:0.2f}\t{:0.2f}\t{:0.2f}'.format(covariance(Z, X), covariance(Z, Y), covariance(Z, Z)))
```

```
X   Y   Z
-   -   -
2   6   1
4   5   3
6   4   5
8   3   7

Covariance Matrix
    X   Y   Z
X   6.67    -3.33   6.67
Y   -3.33   1.67    -3.33
Z   6.67    -3.33   6.67
```

(J) Verify the equation

```
x = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
mean = sum(x)/len(x)
std = math.sqrt(sum([(y-mean)**2 for y in x])/len(x))
print('Mean - ' + str(mean))
print('Standard Deviation (sd) - ' + '{}'.format(std))
print('Mean of Squares - ' + '{}'.format(sum([y**2 for y in x])/len(x)))
print('Sum of square of mean (' + str(mean**2) + ') and square of standard deviation (' + str(std**2) + ') - ' + str(mean**2 + std**2))
```

```
Mean - 11.0
Standard Deviation (sd) - 5.744562646538029
Mean of Squares - 154.0
Sum of square of mean (121.0) and square of standard deviation (33.0) -
154.0
```

NOTE: This is considering the sample is the population itself. Hence we use the 'Uncorrected sample standard deviation'

Q5

Import the libraries

```
import scipy.spatial.distance as dist
import matplotlib.pyplot as plt
import numpy as np
import os
%matplotlib inline
```

Load the data

```
data = None
with open('Data' + os.sep + 'seeds.csv') as fp:
    data = [x.strip().split(',') for x in fp.readlines()]

headers = data[0]
class_field = len(headers)-1
data = [[int(x[i]) if i==class_field else float(x[i]) for i in
range(len(x))] for x in data[1:]]
data = np.asarray(data)

print('Attributes - ')
print('\t'.join([x[:6] for x in headers]))
for i in range(len(data[:10])):
    print('\t'.join(['{0:.3f}'.format(x) for x in data[i]]))
print('...')
print(str(len(data)-10) + ' more rows.')
```

```
Attributes -
area_A  perime  compac  kernel  kernel  asymme  kernel  Class
15.260  14.840  0.871   5.763   3.312   2.221   5.220   1.000
14.880  14.570  0.881   5.554   3.333   1.018   4.956   1.000
14.290  14.090  0.905   5.291   3.337   2.699   4.825   1.000
13.840  13.940  0.895   5.324   3.379   2.259   4.805   1.000
16.140  14.990  0.903   5.658   3.562   1.355   5.175   1.000
14.380  14.210  0.895   5.386   3.312   2.462   4.956   1.000
14.690  14.490  0.880   5.563   3.259   3.586   5.219   1.000
14.110  14.100  0.891   5.420   3.302   2.700   5.000   1.000
```

```
16.630  15.460  0.875   6.053   3.465   2.040   5.877   1.000
16.440  15.250  0.888   5.884   3.505   1.969   5.533   1.000
...
200 more rows.
```

```
types = {'[ORIGINAL]': None, '[NORMALIZED]': None, '[STANDARDIZED]': None}
type_names = ['[ORIGINAL]', '[NORMALIZED]', '[STANDARDIZED]']
dists = {'[EUCLIDEAN]': dist.euclidean,
         '[MAHALANOBIS]': dist.mahalanobis,
         '[CITY BLOCK]': dist.cityblock,
         '[MINKOWSKI (R=3)]': dist.minkowski,
         '[CHEBYSHEV]': dist.chebyshev,
         '[COSINE]': dist.cosine,
         '[CANBERRA]': dist.canberra}
dist_funcs = (dist.euclidean,
              dist.mahalanobis,
              dist.cityblock,
              dist.minkowski,
              dist.chebyshev,
              dist.cosine,
              dist.canberra)
dist_names = ('[EUCLIDEAN]',
              '[MAHALANOBIS]',
              '[CITY BLOCK]',
              '[MINKOWSKI (R=3)]',
              '[CHEBYSHEV]',
              '[COSINE]',
              '[CANBERRA]')
```

(A) Select the attributes and normalize/standardize

```
select_headers = [headers[0], headers[4]]
select = data[:,[0,4]]

print('Total Observations - ' + str(len(select)) + '\n')
print('Attributes - ')
print('\t'.join([x[:6] for x in select_headers]))
for i in range(len(select[:10])):
    print('\t'.join(['{0:.3f}'.format(x) for x in select[i]]))
print('...')
print(str(len(data)-10) + ' more rows.')
```

```
Total Observations - 210

Attributes -
area_A  kernel
15.260  3.312
14.880  3.333
14.290  3.337
13.840  3.379
16.140  3.562
14.380  3.312
14.690  3.259
14.110  3.302
16.630  3.465
16.440  3.505
...
200 more rows.
```

```
def printrange(val, headers, dtype):
    print(dtype + ' Range of values - ')
    amin = np.amin(val, axis=0)
    amax = np.amax(val, axis=0)
    print('\t' + '\t'.join(headers))
    print('min\t' + '\t'.join(['{0:.3f}'.format(x) for x in amin]))
    print('max\t' + '\t'.join(['{0:.3f}'.format(x) for x in amax]))
    print('range\t' + '\t'.join(['{0:.3f}'.format(x) for x in amax-amin]))
    print('\n')
    return amin, amax
```

```
amin, amax = printrange(select, select_headers, '[ORIGINAL]')
types['[ORIGINAL]'] = select

normal = np.copy(select)
normal = (normal - amin)/(amax - amin)
amin, amax = printrange(normal, select_headers, '[NORMALIZED]')
types['[NORMALIZED]'] = normal

amean = np.mean(select, axis=0)
astd = np.std(select, axis=0)
standard = np.copy(select)
standard = (standard - amean)/astd
amin, amax = printrange(standard, select_headers, '[STANDARDIZED]')
```

```
types['[STANDARDIZED]'] = standard

```

```
[ORIGINAL] Range of values -
    area_A  kernel_width
min 10.590  2.630
max 21.180  4.033
range   10.590  1.403

[NORMALIZED] Range of values -
    area_A  kernel_width
min 0.000   0.000
max 1.000   1.000
range   1.000   1.000

[STANDARDIZED] Range of values -
    area_A  kernel_width
min -1.467  -1.668
max 2.182   2.055
range   3.648   3.723
```

```
print('\t'.join(type_names))
print('\t'.join(['\t'.join([x[:6] for x in select_headers])]*3))
for i in range(len(select[:20])):
    t = []
    for y in type_names:
        t.append('\t'.join(['{0:.3f}'.format(x) for x in types[y][i]]))
    print('\t'.join(t))
print('...')
print(str(len(select)-20) + ' more rows.')
```

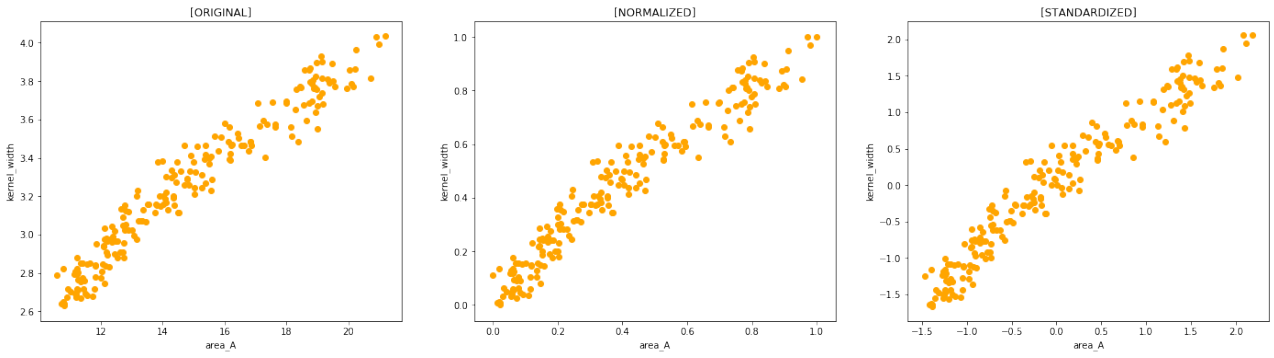
```
[ORIGINAL]  [NORMALIZED]    [STANDARDIZED]
area_A  kernel  area_A  kernel  area_A  kernel
15.260  3.312   0.441   0.486   0.142   0.142
14.880  3.333   0.405   0.501   0.011   0.197
14.290  3.337   0.349   0.504   -0.192  0.208
13.840  3.379   0.307   0.534   -0.347  0.320
16.140  3.562   0.524   0.664   0.445   0.805
14.380  3.312   0.358   0.486   -0.161  0.142
14.690  3.259   0.387   0.448   -0.054  0.001
14.110  3.302   0.332   0.479   -0.254  0.115
16.630  3.465   0.570   0.595   0.614   0.548
16.440  3.505   0.552   0.624   0.549   0.654
```

```
15.260  3.242   0.441   0.436   0.142   -0.044
14.030  3.201   0.325   0.407   -0.282  -0.153
13.890  3.199   0.312   0.406   -0.330  -0.158
13.780  3.156   0.301   0.375   -0.368  -0.272
13.740  3.114   0.297   0.345   -0.382  -0.384
14.590  3.333   0.378   0.501   -0.089  0.197
13.990  3.383   0.321   0.537   -0.295  0.330
15.690  3.514   0.482   0.630   0.290   0.678
14.700  3.466   0.388   0.596   -0.051  0.550
12.720  3.049   0.201   0.299   -0.733  -0.556
...
190 more rows.
```

(B) Distance Metrics

(i) Scatter Plots

```
fig, axes = plt.subplots(nrows=1, ncols=3)
fig.set_figheight(6)
fig.set_figwidth(24)

for i, data in enumerate(type_names):
    axes[i].set_title(data)
    axes[i].set_xlabel(select_headers[0])
    axes[i].set_ylabel(select_headers[1])
    axes[i].scatter(types[data][:,0], types[data][:,1], c='orange')

plt.show()
```



Analysis

1. All the plots show that there is high colinearity between the 2 attributes.
2. Normalization and Standardization do not change the relative positioning of the elements, but scale the values.

3. Normalization only scales the values so that the range of values in the dataset is 1 and the minimum and maximum are 1 and 0 respectively. This effectively reduces the spread of the dataset to 1 unit if we plot its histogram.
4. Standardization scales the data so that the mean of the dataset is 0 and has a unit standard deviation. This helps preserve the spread of the data, allowing us to compare the distribution of this data with the standard normal distribution.

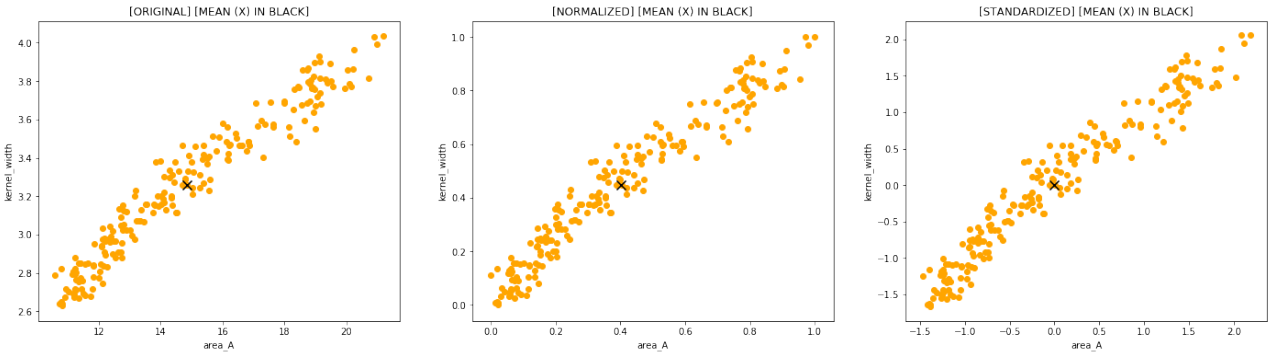
(ii) Mean values for all categories

```
def getmeanstd(val, headers, dtype):
    print(dtype + ' Mean and Standard Deviation - ')
    mean = np.mean(val, axis=0)
    std = np.std(val, axis=0)
    print('\t' + '\t'.join(headers))
    print('mean\t' + '\t'.join(['{0:.3f}'.format(x) for x in mean]))
    print('st.dev\t' + '\t'.join(['{0:.3f}'.format(x) for x in std]))
    print()
    return mean, std
```

```
mean = {x:None for x in types.keys()}
std = {x:None for x in types.keys()}

for data in types.keys():
    mean[data], std[data] = getmeanstd(types[data], select_headers, data)

fig, axes = plt.subplots(nrows=1, ncols=3)
fig.set_figheight(6)
fig.set_figwidth(24)

for i, data in enumerate(type_names):
    axes[i].set_title(data + ' [MEAN (X) IN BLACK]')
    axes[i].set_xlabel(select_headers[0])
    axes[i].set_ylabel(select_headers[1])
    axes[i].scatter(types[data][:,0], types[data][:,1], c='orange')
    axes[i].scatter(mean[data][0], mean[data][1], s=100, c='black',
marker='x')

plt.show()
```

```
[ORIGINAL] Mean and Standard Deviation -
    area_A  kernel_width
mean    14.848  3.259
st.dev  2.903   0.377
```

```
[NORMALIZED] Mean and Standard Deviation -
    area_A  kernel_width
mean    0.402   0.448
st.dev  0.274   0.269

[STANDARDIZED] Mean and Standard Deviation -
    area_A  kernel_width
mean    -0.000  -0.000
st.dev  1.000   1.000
```



(iii) Distance from mean for all categories over all distance metrics

```
def getdists(types, mean, dist_funcs, dist_names, type_names):
    t = []
    for i in range(len(dist_funcs)):
        ret = []
        for x in type_names:
            if dist_names[i]=='[MAHALANOBIS]':
                ret.append([dist_funcs[i](mean[x], y,
np.linalg.inv(np.cov(types[x].T))) for y in types[x]])
            elif dist_names[i]=='[MINKOWSKI (R=3)]':
                ret.append([dist_funcs[i](mean[x], y, 3) for y in
types[x]])
            else:
                ret.append([dist_funcs[i](mean[x], y) for y in types[x]])
        ret = np.asarray(ret)
        ret = np.transpose(ret)
        t.append(ret)
    return t
```

```
alldists = getdists(types, mean, dist_funcs, dist_names, type_names)
```

(iv) Top 10 nearest points for each distance metric

```
def gettop10s(alldists, type_names):
    t = []
    for x in alldists:
        d = {key:None for key in type_names}
        c = np.copy(x)
        c.sort(axis=0)
        c = c[9]
        for i in range(len(type_names)):
            d[type_names[i]] = np.where(x[:,i]<=c[i])[0]
        t.append(d)
    return t
```

```
top10s = gettop10s(alldists, type_names)
```

```
for i,x in enumerate(dist_names):
    print(x)
    for y in type_names:
        print('\t' + y + ' : ' + ', '.join([str(n) for n in top10s[i][y]]))
    print()
```

```
[EUCLIDEAN]
    [ORIGINAL] : 1, 6, 24, 34, 38, 47, 48, 49, 55, 57
    [NORMALIZED] : 6, 10, 24, 34, 38, 48, 49, 50, 55, 132
    [STANDARDIZED] : 6, 10, 24, 34, 38, 48, 49, 50, 55, 132

[MAHALANOBIS]
    [ORIGINAL] : 0, 6, 21, 24, 32, 33, 49, 53, 66, 67
    [NORMALIZED] : 0, 6, 21, 24, 32, 33, 49, 53, 66, 67
    [STANDARDIZED] : 0, 6, 21, 24, 32, 33, 49, 53, 66, 67

[CITY BLOCK]
    [ORIGINAL] : 1, 6, 24, 34, 38, 47, 48, 49, 55, 57
    [NORMALIZED] : 1, 6, 10, 24, 38, 48, 49, 50, 55, 132
    [STANDARDIZED] : 1, 6, 10, 24, 38, 48, 49, 50, 55, 132

[MINKOWSKI (R=3)]
    [ORIGINAL] : 1, 6, 24, 34, 38, 47, 48, 49, 55, 57
```

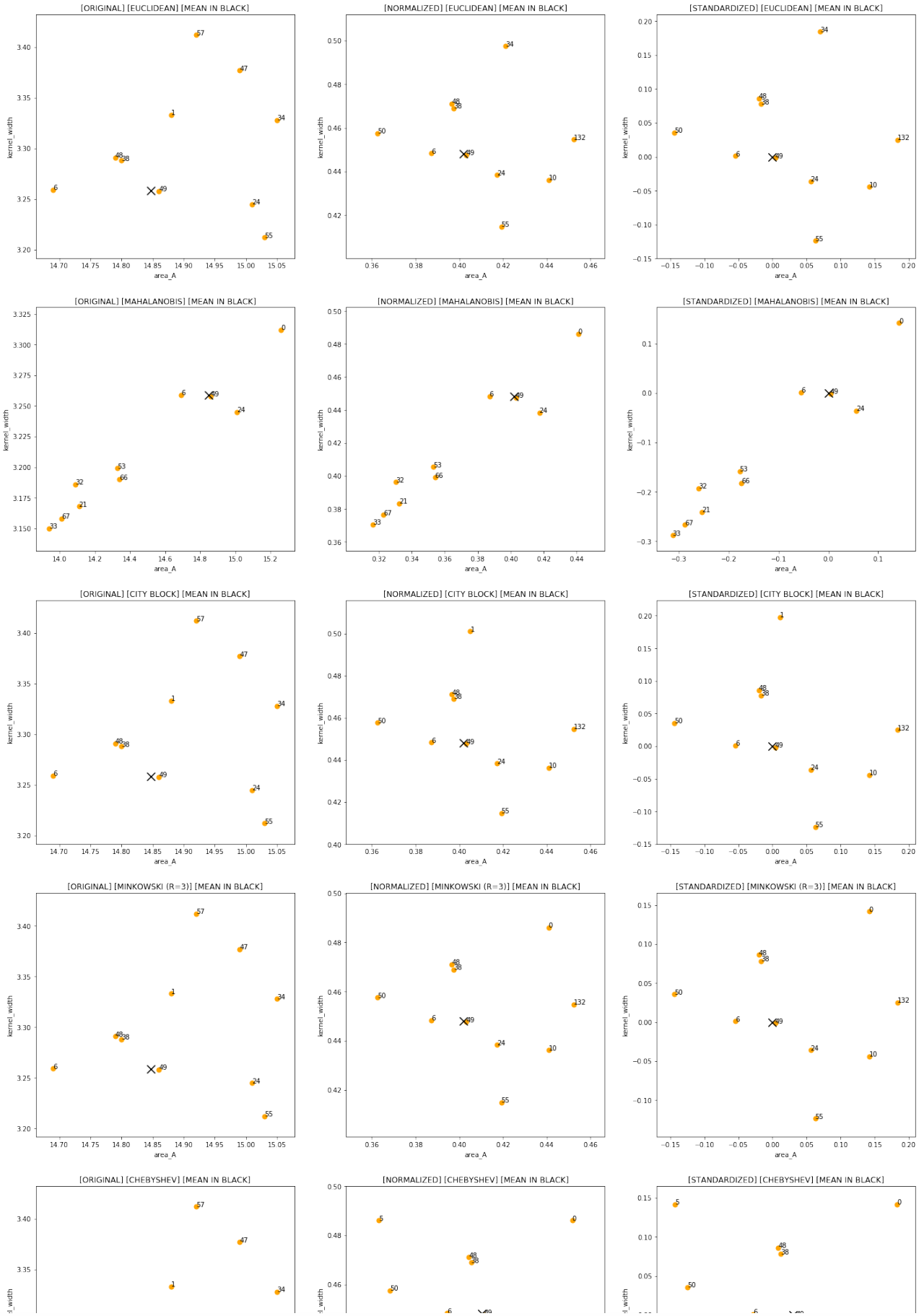
```
    [NORMALIZED] : 0, 6, 10, 24, 38, 48, 49, 50, 55, 132
    [STANDARDIZED] : 0, 6, 10, 24, 38, 48, 49, 50, 55, 132

[CHEBYSHEV]
    [ORIGINAL] : 1, 6, 24, 34, 38, 47, 48, 49, 55, 57
    [NORMALIZED] : 0, 5, 6, 10, 24, 38, 48, 49, 50, 55
    [STANDARDIZED] : 0, 5, 6, 10, 24, 38, 48, 49, 50, 55

[COSINE]
    [ORIGINAL] : 4, 20, 22, 31, 34, 43, 46, 49, 68, 134
    [NORMALIZED] : 0, 9, 36, 49, 66, 112, 130, 139, 140, 162
    [STANDARDIZED] : 14, 26, 66, 163, 164, 171, 182, 183, 191, 206

[CANBERRA]
    [ORIGINAL] : 1, 6, 10, 24, 34, 38, 48, 49, 50, 55
    [NORMALIZED] : 1, 6, 10, 24, 38, 48, 49, 50, 55, 132
    [STANDARDIZED] : 6, 10, 18, 24, 28, 38, 48, 49, 137, 207
```

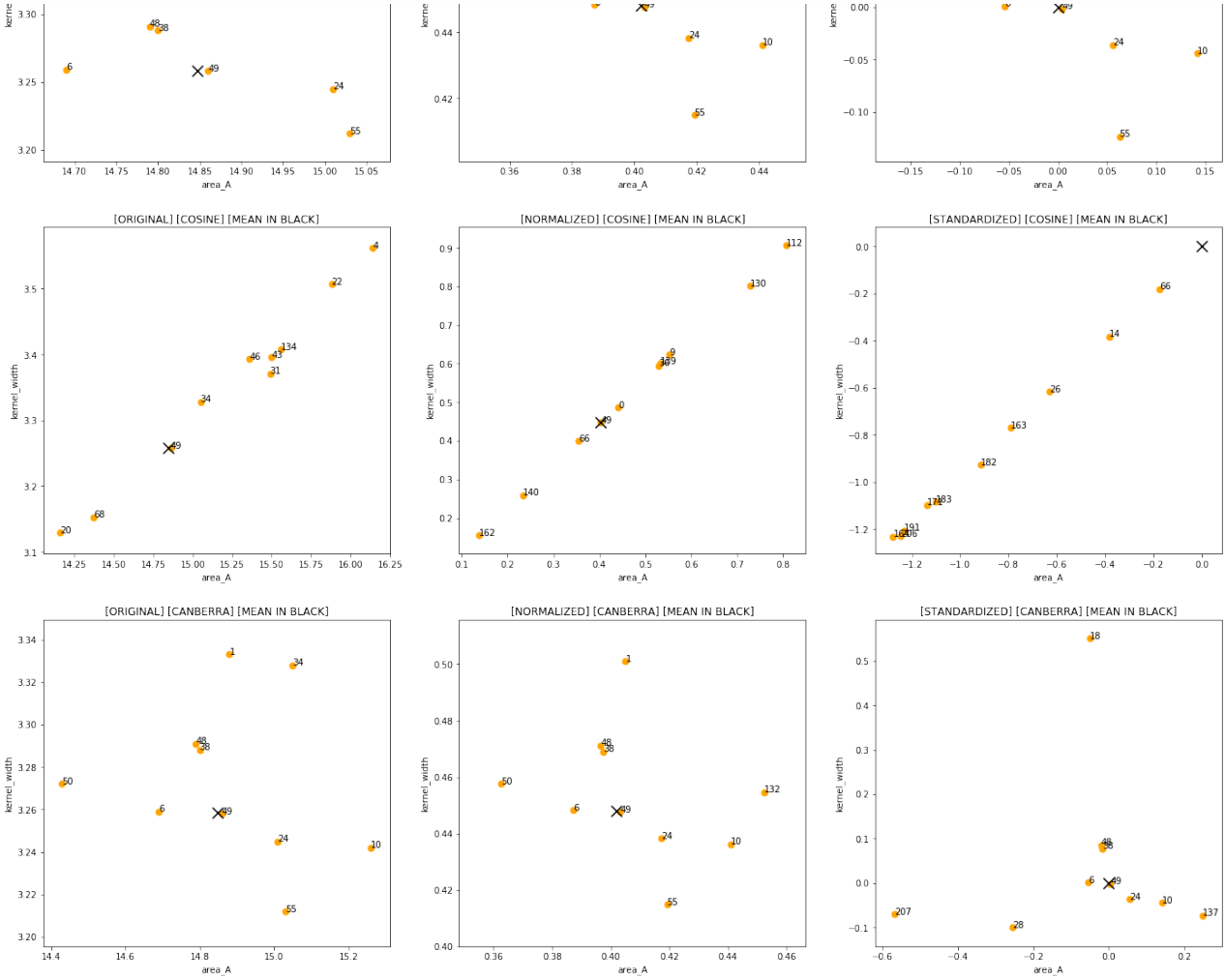
(v) Plot top 10 nearest points

```
def gplot(axes, i, j, title, dist, mean, headers, original, labels):
    axes[i,j].set_title(title)
    axes[i,j].scatter(dist[:,0], dist[:,1], s=50, c='orange')
    axes[i,j].scatter(mean[0], mean[1], s=150, c='black', marker='x')
    axes[i,j].set_xlabel(headers[0])
    axes[i,j].set_ylabel(headers[1])
    for k,label in enumerate(labels):
        axes[i,j].annotate(label, (dist[k,0], dist[k,1]))
```

```
fig, axes = plt.subplots(nrows=len(alldists), ncols=len(type_names))
fig.set_figheight(56)
fig.set_figwidth(24)

for i in range(len(alldists)):
    for j in range(len(type_names)):
        gplot(axes, i, j, type_names[j] + ' ' + dist_names[i] + ' [MEAN IN BLACK]', types[type_names[j]][top10s[i][type_names[j]]], mean[type_names[j]], select_headers, types[type_names[j]], top10s[i][type_names[j]])

plt.show()
```

Please note that we have attached this image separately in our submission folder as Q5B5.png





(vi) Verification if the nearest points are similar across all distance metrics

```
s = {}
for y in type_names:
    print(y)
    t = set()
    for i,x in enumerate(dist_names):
        if x not in ['[MAHALANOBIS]', '[COSINE]', '[CANBERRA]']:
            if len(t)==0:
                t = set(top10s[i][y])
            else:
                t = t.intersection(top10s[i][y])
        print('\t' + x + ' : ' + ', '.join([str(n) for n in top10s[i][y]]))
    s[y] = sorted(list(t))
    print()
```

```
[ORIGINAL]
    [EUCLIDEAN] : 1, 6, 24, 34, 38, 47, 48, 49, 55, 57
    [MAHALANOBIS] : 0, 6, 21, 24, 32, 33, 49, 53, 66, 67
    [CITY BLOCK] : 1, 6, 24, 34, 38, 47, 48, 49, 55, 57
    [MINKOWSKI (R=3)] : 1, 6, 24, 34, 38, 47, 48, 49, 55, 57
    [CHEBYSHEV] : 1, 6, 24, 34, 38, 47, 48, 49, 55, 57
    [COSINE] : 4, 20, 22, 31, 34, 43, 46, 49, 68, 134
    [CANBERRA] : 1, 6, 10, 24, 34, 38, 48, 49, 50, 55

[NORMALIZED]
    [EUCLIDEAN] : 6, 10, 24, 34, 38, 48, 49, 50, 55, 132
    [MAHALANOBIS] : 0, 6, 21, 24, 32, 33, 49, 53, 66, 67
    [CITY BLOCK] : 1, 6, 10, 24, 38, 48, 49, 50, 55, 132
    [MINKOWSKI (R=3)] : 0, 6, 10, 24, 38, 48, 49, 50, 55, 132
    [CHEBYSHEV] : 0, 5, 6, 10, 24, 38, 48, 49, 50, 55
    [COSINE] : 0, 9, 36, 49, 66, 112, 130, 139, 140, 162
    [CANBERRA] : 1, 6, 10, 24, 38, 48, 49, 50, 55, 132

[STANDARDIZED]
    [EUCLIDEAN] : 6, 10, 24, 34, 38, 48, 49, 50, 55, 132
    [MAHALANOBIS] : 0, 6, 21, 24, 32, 33, 49, 53, 66, 67
    [CITY BLOCK] : 1, 6, 10, 24, 38, 48, 49, 50, 55, 132
    [MINKOWSKI (R=3)] : 0, 6, 10, 24, 38, 48, 49, 50, 55, 132
    [CHEBYSHEV] : 0, 5, 6, 10, 24, 38, 48, 49, 50, 55
    [COSINE] : 14, 26, 66, 163, 164, 171, 182, 183, 191, 206
    [CANBERRA] : 6, 10, 18, 24, 28, 38, 48, 49, 137, 207
```

Analysis

1. The top 10 points are not same for all distance metrics. Apart from **MAHALANOBIS, COSINE and CANBERRA**, almost all other metrics share more than 70% of the points.

- For **original data**, apart from the above 3, all other distance metrics give the same points - 1, 6, 24, 34, 38, 47, 48, 49, 55, 57
- For **normalized data**, apart from the above 3, 80% of the points are same across all metrics - 6, 10, 24, 38, 48, 49, 50, 55
- For **standardized data**, apart from the above 3, 80% of the points are same across all metrics - 6, 10, 24, 38, 48, 49, 50, 55

2. **MAHALANOBIS** gives different points as it lets the data decide the coordinate system, ie - it transforms the coordinates along the 1st and 2nd PCA components(in case of 2 dimensional data) and then calculates the Euclidean distance between points in this tranformed space.
3. **COSINE** distance selects points that are radially close to each other from the center. In our case, one of those points is the mean, hence this metric tends to select points which are lying

on or near the line connecting the mean and origin. If the angle between 2 points from the center is near to 0, cosine reaches its maximum. Cosine Distance is 1-cos(theta), hence this distance is minimum when the cosine is maximum.

4. **CANBERRA** distance for standardized distance is just random selection of 10 points. Canberra distance from mean (0,0) is 1 for all the points. This distance metric tends to select the points whose abcissa or ordinate is the same as the other point (in our case the mean). Hence most of the points are near the imaginary vertical and horizontal line intersecting at the mean.

(vii) Results

We can clearly see from the above scenario that most of the distance metrics are heavily affected by the range of values along each dimension of the data. If the range of one of the dimensions is much greater than the other, it tends to dominate the distance value. This might lead to unequal contribution of each dimension in calculating distance between 2 points.

Some distance metrics like Mahalanobis, Canberra and Cosine overcome this issue and give different points. Cosine is very favorable for calculating similarity between vectors. Mahalanobis is completely independent of the Euclidean co-ordinate axis, and calculates the distance along completely different co-ordinate system dictated by the variance in the data.

Thus it is important to transform the data to avoid effect of dimension with large range of values.

Normalization scales all the dimensions in the range of [0,1] Thus all dimensions have equal weightage. But this also incorporates the outliers in the actual data, thus it is not useful in scenarios which are highly sensitive to outliers.

Standardization on the other hand preserves the spread of the distribution along each dimension and also centers the mean of each dimension at 0 with a standard deviation of 1. This allows us to make comparisions with the Standard Normal Distribution and also helps eliminate outliers, while also reducing the range of data.

Q6

Import Libraries

```
from scipy import stats

import matplotlib.pyplot as plt
import numpy as np
import os
%matplotlib inline
```

Load Data

```
data = None
with open('Data' + os.sep + 'hw1q6_data.csv') as fp:
    data = [x.strip().split(',') for x in fp.readlines()]

headers = data[0]
class_field = len(headers)-1
data = [[int(x[i]) if i==class_field else float(x[i]) for i in
range(len(x))] for x in data[1:]]

print('Total Observations - ' + str(len(data)) + '\n')
print('Attributes - ')
print('\t'.join([x[:6] for x in headers]))
for i in range(len(data[:10])):
    print('\t'.join(['{}'.format(x) for x in data[i]]))
print('...')
print(str(len(data)-10) + ' more rows.')
```

```
Total Observations - 768

Attributes -
Glucos  BloodP  SkinTh  BMI Diabet  Age Class
148.0   72.0    35.0    33.6    0.627   50.0    1
85.0    66.0    29.0    26.6    0.351   31.0    0
183.0   64.0    0.0 23.3    0.672   32.0    1
89.0    66.0    23.0    28.1    0.167   21.0    0
137.0   40.0    35.0    43.1    2.288   33.0    1
```

```
116.0   74.0    0.0 25.6    0.201   30.0    0
78.0    50.0    32.0    31.0    0.248   26.0    1
115.0   0.0 0.0 35.3    0.134   29.0    0
197.0   70.0    45.0    30.5    0.158   53.0    1
125.0   96.0    0.0 0.0 0.232   54.0    1
...
758 more rows.
```

Helper Functions

```
def countPatients(data):
    diabetic = 0
    nondiabetic = 0

    for i in data[:,6]:
        if i == 1:
            diabetic += 1
        else:
            nondiabetic += 1

    return (diabetic, nondiabetic)
```

(A) Number of diabetic and nondiabetic patients

```
data = np.asarray(data)
diabetic, nondiabetic = countPatients(data)
print(f'Number of diabetic patients in the dataset: {diabetic}')
print(f'Number of nondiabetic patients in the dataset: {nondiabetic}')
```

```
Number of diabetic patients in the dataset: 268
Number of nondiabetic patients in the dataset: 500
```

(B) Missing rate for each feature

```
def countMissingValue(data):
    missing_values = 0
    for i in data:
        if i == 0:
            missing_values += 1
    return '{:.2f}'.format(missing_values/len(data) * 100)

print('Missing Rates:')
for i in range(len(headers)-1):
    percent = countMissingValue(data[:,i])
    print(f'{headers[i]} : {percent}% ')
```

```
Missing Rates:
Glucose : 0.65%
BloodPressure : 4.56%
SkinThickness : 29.56%
BMI : 1.43%
DiabetesPedigreeFunction : 0.00%
Age : 0.00%
```

(C) Methods to handle missing data

Listwise Deletion

The most simple method for handling missing data is to simply remove/neglect the rows (objects) having at least one missing attribute and analyze the rest of the data.

Advantages:

- The remaining dataset is complete and accurate.
- Most frequently used method in many fields. Therefore it is also called the default option in many statistical procedures in statistical libraries.

Disadvantages:

- Reduced sample size caused by removing incomplete data
  If there is not a large sample or the assumption of missing completely at random is not satisfied, this is not the optimal strategy.
- It increases chance of having a biased dataset to be substantial if data is not most completely at random.

Mean Substitution

As the name suggests, the missing data value for the attribute of a row(object) is replaced by mean value of that attribute in place of that.

Advantages:

- Allows the whole data to be used for analysis in a given incomplete dataset.
- As mean seems a reasonable estimate for a randomly selected observation from given normal distribution.

  *Disadvantages**:

- For missing values that are not strictly random, especially in presence of a great inequality in number of missing values for different variables, the mean substitution method may lead to inconsistent bias.
- May underestimate the error.

Removing missing data, rows of patient

```
remaining_data = data[(data[:,0:6]!=0).all(1)]
print('Total Observations - ' + str(len(remaining_data)) + '\n')
print('Attributes - ')
print('\t'.join([x[:6] for x in headers]))
for i in range(len(remaining_data[:10])):
    print('\t'.join(['{}'.format(x) for x in remaining_data[i]]))
print('...')
print(str(len(remaining_data)-10) + ' more rows.')
```

```
Total Observations - 532

Attributes -
Glucos  BloodP  SkinTh  BMI Diabet  Age Class
148.0   72.0    35.0    33.6    0.627   50.0    1.0
85.0    66.0    29.0    26.6    0.351   31.0    0.0
89.0    66.0    23.0    28.1    0.167   21.0    0.0
137.0   40.0    35.0    43.1    2.288   33.0    1.0
78.0    50.0    32.0    31.0    0.248   26.0    1.0
197.0   70.0    45.0    30.5    0.158   53.0    1.0
189.0   60.0    23.0    30.1    0.398   59.0    1.0
166.0   72.0    19.0    25.8    0.587   51.0    1.0
118.0   84.0    47.0    45.8    0.551   31.0    1.0
103.0   30.0    38.0    43.3    0.183   33.0    0.0
...
522 more rows.
```

```
print('After removing missing data, missing Rates:')
for i in range(len(headers)-1):
    percent = countMissingValue(remaining_data[:,i])
    print(f'{headers[i]} : {percent}%')
```

```
After removing missing data, missing Rates:
Glucose : 0.00%
BloodPressure : 0.00%
SkinThickness : 0.00%
BMI : 0.00%
DiabetesPedigreeFunction : 0.00%
Age : 0.00%
```

(D) Number of diabetic and nondiabetic patients in remaining data

```
diabetic, nondiabetic = countPatients(remaining_data)
print(f'Number of diabetic patients in the dataset: {diabetic}')
print(f'Number of nondiabetic patients in the dataset: {nondiabetic}')
```

```
Number of diabetic patients in the dataset: 177
Number of nondiabetic patients in the dataset: 355
```

(E) Summary Statistics

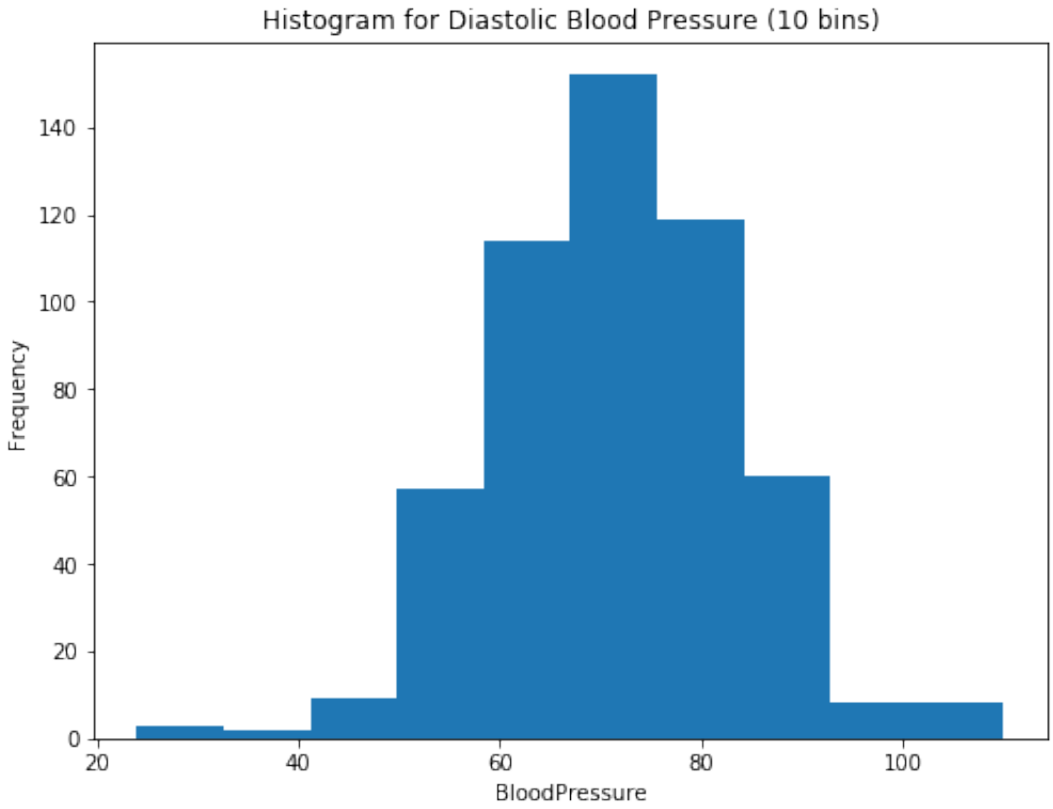
```
print('After removing missing data, Statistics:')
for i in range(len(headers)-1):
    mean = np.average(remaining_data[:,i])
    median = np.median(remaining_data[:,i])
    std_deviation = np.std(remaining_data[:,i])
    data_range = np.ptp(remaining_data[:,i])
    percentile_25 = np.percentile(remaining_data[:,i],25)
    percentile_50 = np.percentile(remaining_data[:,i],50)
    percentile_75 = np.percentile(remaining_data[:,i],75)
    print(f'\n{headers[i]} :-> Mean: {mean:.3f}, \n\t Median: {median:.3f}, \n\t Standard Deviation: {std_deviation:.3f}, \n\t Range: {data_range:.3f}, '
          f'\n\t 25th percentile: {percentile_25:.3f}, \n\t 50th percentile: {percentile_50:.3f}, \n\t 75th percentile: {percentile_75:.3f}')
```

```
After removing missing data, Statistics:

Glucose :-> Mean: 121.030,
     Median: 115.000,
     Standard Deviation: 30.970,
     Range: 143.000,
     25th percentile: 98.750,
     50th percentile: 115.000,
     75th percentile: 141.250

BloodPressure :-> Mean: 71.506,
     Median: 72.000,
     Standard Deviation: 12.299,
     Range: 86.000,
     25th percentile: 64.000,
     50th percentile: 72.000,
     75th percentile: 80.000

SkinThickness :-> Mean: 29.182,
     Median: 29.000,
     Standard Deviation: 10.514,
     Range: 92.000,
     25th percentile: 22.000,
     50th percentile: 29.000,
     75th percentile: 36.000

BMI :-> Mean: 32.890,
     Median: 32.800,
```

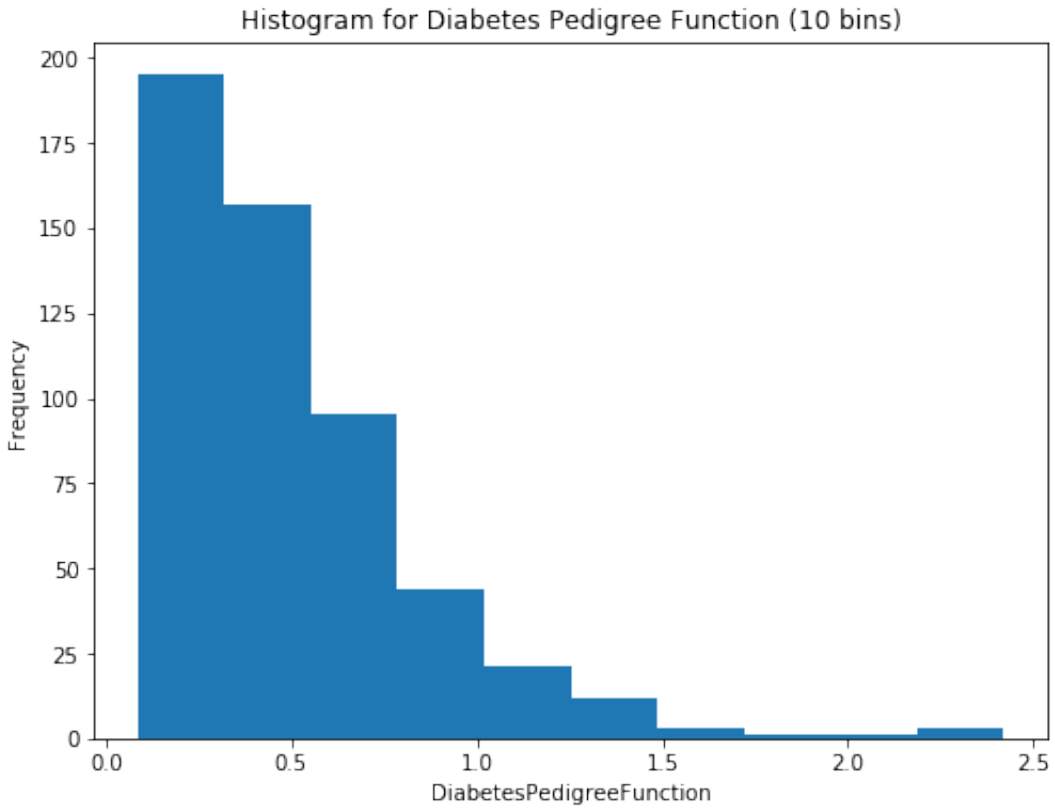
```
     Standard Deviation: 6.875,
     Range: 48.900,
     25th percentile: 27.875,
     50th percentile: 32.800,
     75th percentile: 36.900

DiabetesPedigreeFunction :-> Mean: 0.503,
     Median: 0.416,
     Standard Deviation: 0.344,
     Range: 2.335,
     25th percentile: 0.259,
     50th percentile: 0.416,
     75th percentile: 0.659

Age :-> Mean: 31.615,
     Median: 28.000,
     Standard Deviation: 10.751,
     Range: 60.000,
     25th percentile: 23.000,
     50th percentile: 28.000,
     75th percentile: 38.000
```

(F) Histograms for Blood Pressure and Diabetes Pedigree Function

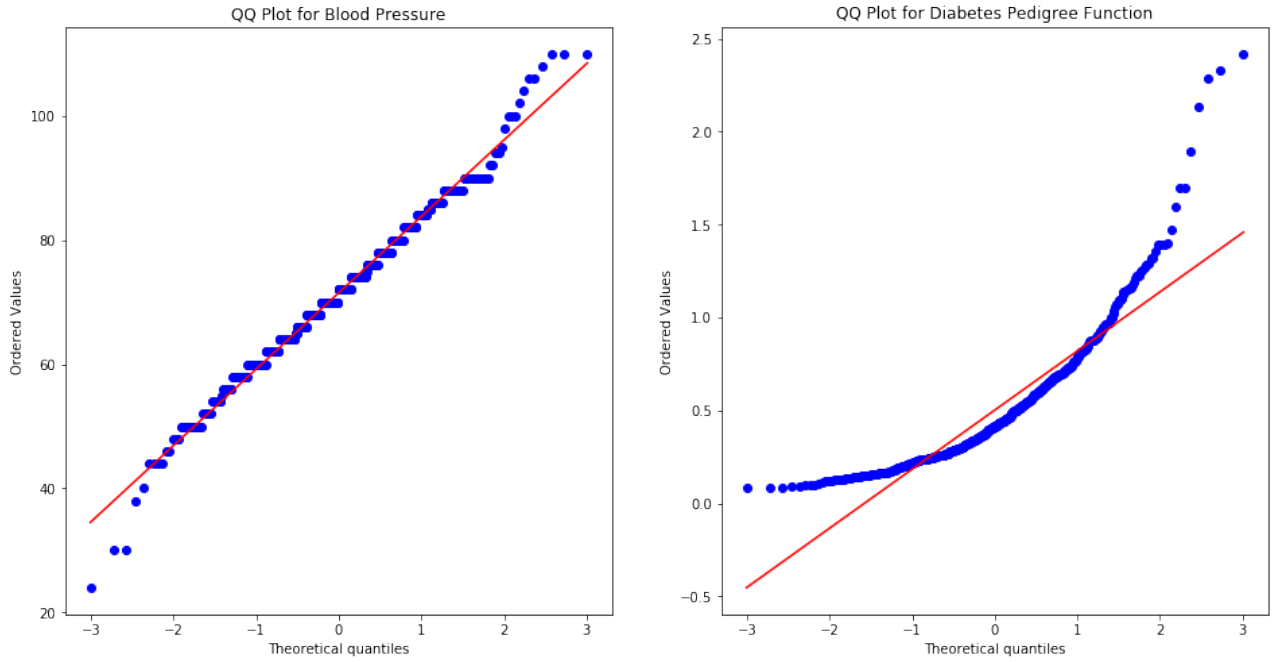
```
#Histogram for Blood Pressure
bins = np.histogram(remaining_data[:,1])
fig, axes = plt.subplots(nrows=1, ncols=1)
fig.set_figheight(6)
fig.set_figwidth(8)
axes.set_title('Histogram for Diastolic Blood Pressure (10 bins)')
axes.set_xlabel(headers[1])
axes.set_ylabel('Frequency')
axes.hist(remaining_data[:,1], bins[1])
plt.show()
```




```
#Histogram for DiabetesPedigreeFunction
bins = np.histogram(remaining_data[:,4])
fig, axes = plt.subplots(nrows=1, ncols=1)
fig.set_figheight(6)
fig.set_figwidth(8)
axes.set_title('Histogram for Diabetes Pedigree Function (10 bins)')
axes.set_xlabel(headers[4])
axes.set_ylabel('Frequency')
axes.hist(remaining_data[:,4], bins[1])
plt.show()
```




```
fig, axes = plt.subplots(nrows=1, ncols=2)
fig.set_figheight(8)
fig.set_figwidth(16)
stats.probplot(remaining_data[:,1], plot=axes[0])
stats.probplot(remaining_data[:,4], plot=axes[1])
axes[0].set_title('QQ Plot for Blood Pressure')
axes[1].set_title('QQ Plot for Diabetes Pedigree Function')
plt.show()
```



(G) QQ Plot Analysis

QQ Plot is often used to compare the distribution of data with the standard normal distibution. This helps us in predicting the shape of the distribution and figure out how to select outliers.

If we extend the zero point on x-axis to the red line, the y co-ordinate of that intersection will be the mean of our distribution. We compare the other points with a horizontal line parallel to x-axis that passes through that intersection. The red line then dictates how far the points should be from the mean if our distribution is similar to the standard normal distribution.

Long tails mean that the values at extremes of the distribution have a higher chance of occuring when compared with the standard normal distribution. Short tails mean that there are limited values with significant probability of occuring compared to the standard normal distribution.

- QQ Plot for Blood Pressure - We can see that on the left hand of 0 on the x-axis, the values are deviating from the red line and ***away*** from mean. This means that the distribution of Blood Pressure has a ***long left tail***. Similarly, the values on to the right of 0 are also deviating from the red line and ***away*** from the mean, which means that the Blood Pressure distribution has a ***long right tail***.
- QQ Plot for Diabetes Pedigree Function - We can see that on the left hand of 0 on the x-axis, the values are deviating largely from the red line and ***towards*** the mean. This means that the distribution of Diabetes Pedigree Function has a ***very short left tail***. On the contrary, the values to the right of 0 are deviating largely from the red line but ***away*** from the mean, which mean that Diabetes Pedigree Function distibution has a ***long right tail***.

These conclusions can be backed by looking at the histograms we plotted above. They show the same structure as suggested here.