Data Mining
Home work 12
Clustering, PCA, OLAP

Aqeel Labash
Lecturer: Jaak Vilo

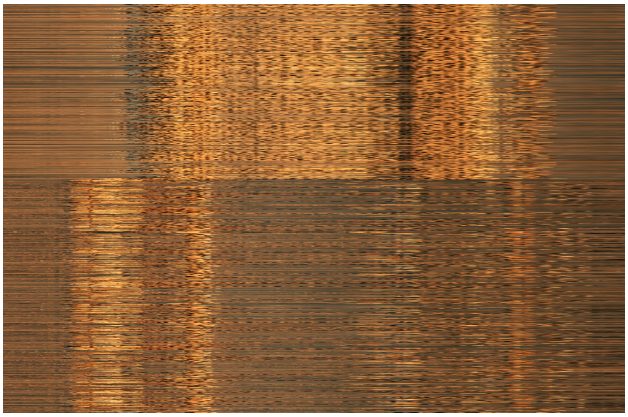
May 14, 2016

First Question

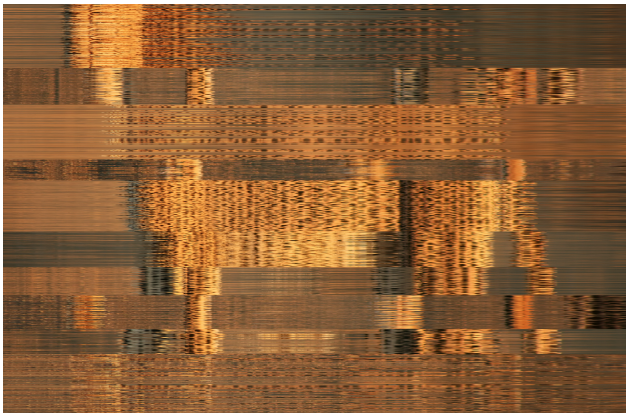
For this question I tried K-means. I used the following code :

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
with open ('DM2016_org.txt') as f:
    d = {}
    headers = f.readline().split(' ')
    values = map(lambda x:x.split(),f.readlines())
    for i in range(len(values[0])):
        d[i]=[]
        for v in values:
            d[i].append(v[i])
data = pd.DataFrame(d)
npdata = data[data.columns[1:]].as_matrix()
npdata = np.array(npdata,dtype=int)
random_state = 170
nc = 339
y_pred = KMeans(n_clusters=nc, random_state=random_state,precompute_distances=True).
    fit_predict(npdata)
output=[]
for i in range(nc):
    print i
    map(lambda x:output.append(x),map(lambda x:data[data.index[0]][x].values,np.where(y_pred==
    i))[0])
f = open('Kmeans_399.txt','w')
for element in output:
    f.write('\n{}'.format(element))
f.close()
```

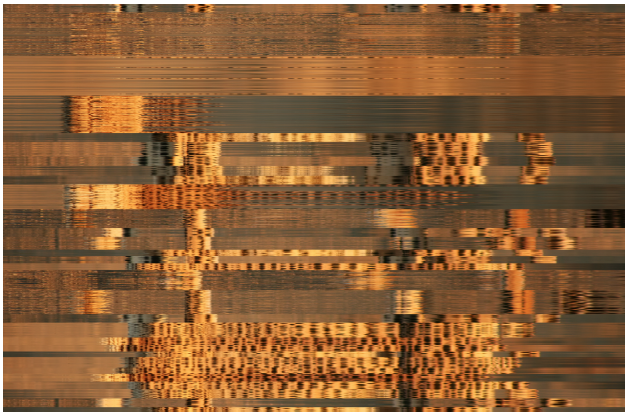
I tried 3 values as number of clusters 2,10,30,399. The results were as following:



(a) KMeans 2 clusters



(b) KMeans 10 clusters



(c) KMeans 30 clusters

(d) KMeans 30 clusters

Fig. 1: Shows Comparsion between different number of clusters using K-Means

Only getting a cluster values won't work because in that case we only know the cluster elements.Which won't let us know the order of the cluster. That information is crucial in our case.
After that I tried hierarchical clustering using Scipy library. The full size dendogram was as following :

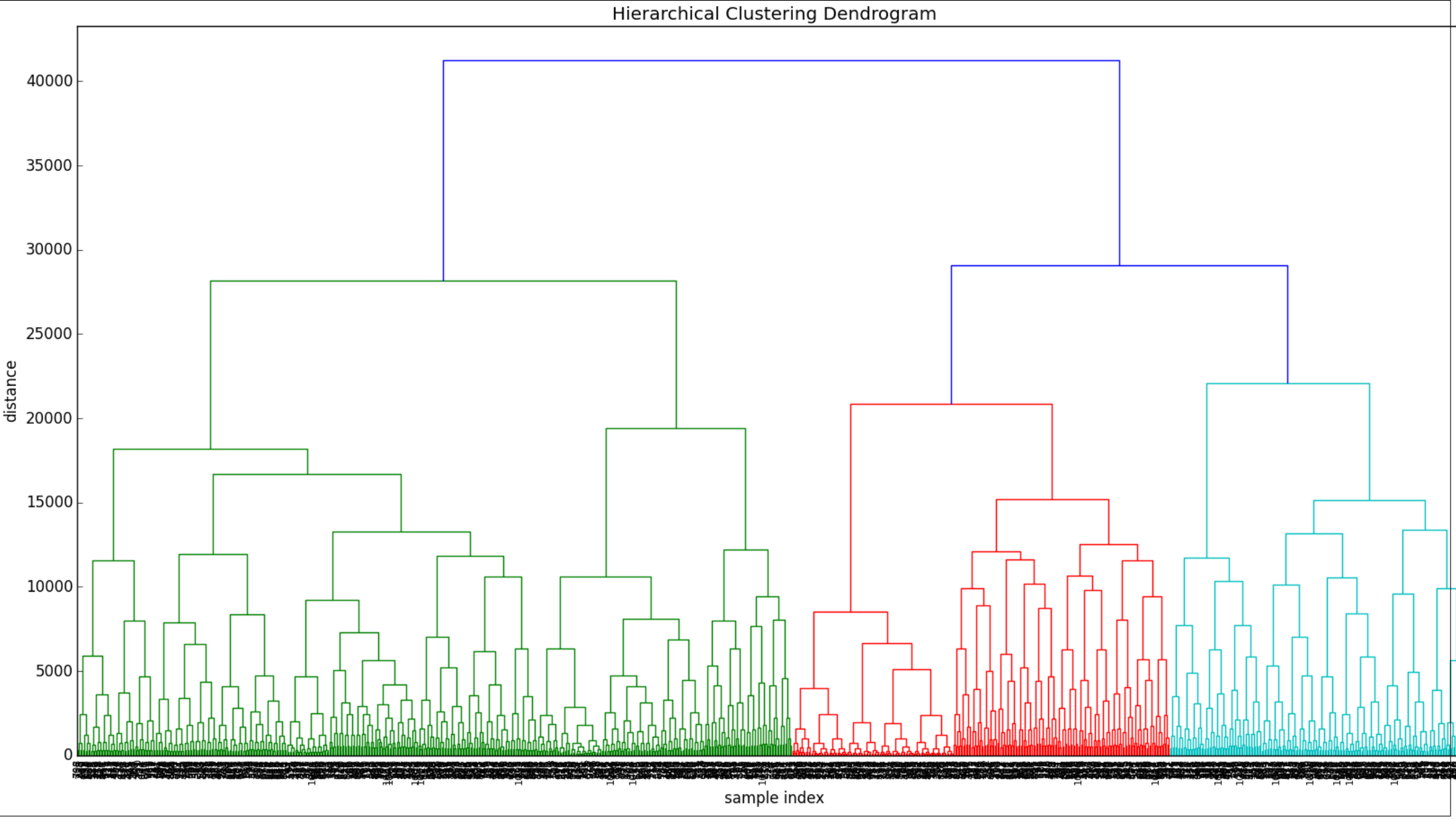


Fig. 2: Full dendogram

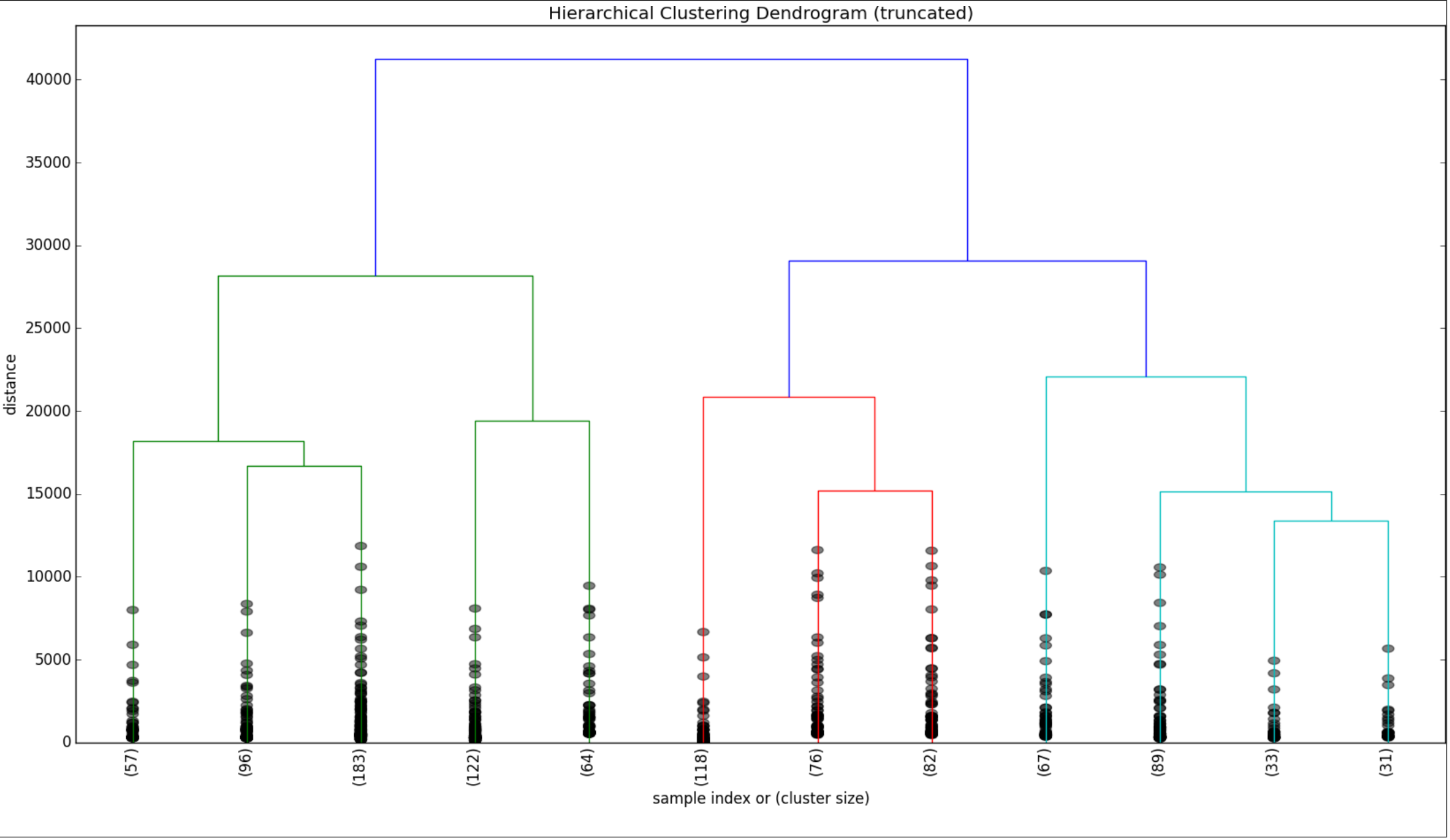


Fig. 3: Truncated Dendogram

Second Question

For this question I read the file in python and extract the data as matrix and then used sklearn for PCA and used matplotlib for 3d plotting. Here is the code :

```
import numpy as np
from sklearn.decomposition import PCA
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
with open ('DM2016_org.csv') as f:
    d = {}
    headers = f.readline().split(' ')
    values = map(lambda x:x.split(),f.readlines())
    for i in range(len(values)):
        d[headers[i]]=[]
        for v in values:
            d[headers[i]].append(v[i])
data = pd.DataFrame(d)
npdata = data[data.columns[1:]].as_matrix()
pca = PCA(n_components=3)
pca.fit(npdata)
output = pca.transform(npdata)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
n = 100
for x, y, z in output:
    ax.scatter(x, y, z)

ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

plt.show()
```

And here is the output figure:

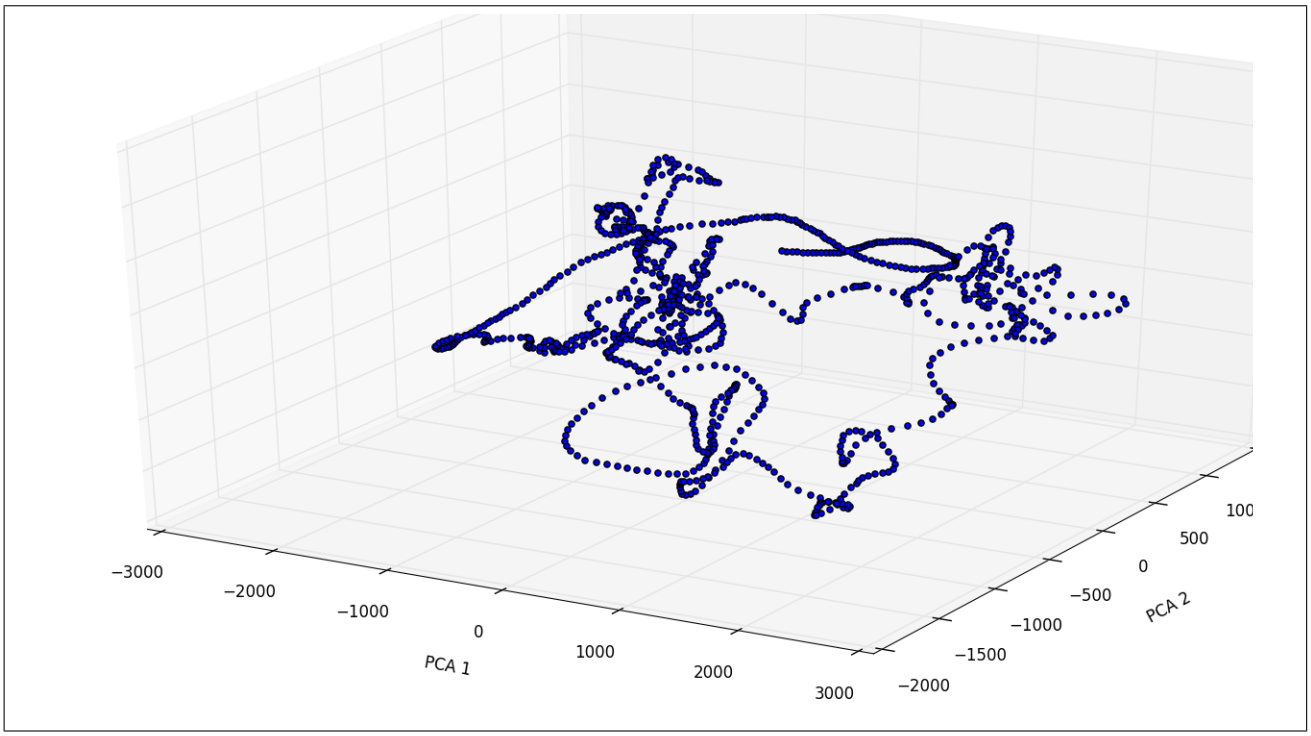


Fig. 4: X: PCA1 , Y:PCA2 , Z:PCA3

Third Question

For this question I used the following code to find pivot table on basis of Gender and education :

```
import pandas as pd
import numpy as np
```

```
%matplotlib inline
import matplotlib.pyplot as plt
data = pd.read_csv('extract_medium.csv',sep=';')
data.head(1)
table = pd.pivot_table(data,values='Earnings',index=['Sex', 'Education'],aggfunc=np.mean)
##OR###
temp1 = data.groupby(['Sex', 'Education']).Earnings.mean()
temp1
temp1 = temp1.values
test=temp1.reshape(2,17)
test.shape
plt.pcolor(test,cmap=plt.cm.Reds,vmin=np.min(test), vmax=np.max(test))
plt.yticks([0,1],range(3))
plt.xticks(range(17),range(17))
plt.title('Heat map of average Earnings Gender Vs Education')
plt.show()
plt.close()
```

And here is the pivot table :

Sex	Education	Average earning
Male	Not in universe (Under 3 years)	0.000000
	No schooling completed	3504.784689
	Nursery school to 4th grade	690.859232
	5th or 6th grade	5083.671988
	7th or 8th grade	4073.961606
	9th grade	9596.498516
	10th grade	11185.848485
	11th grade	11167.638889
	12th grade,no diploma	19404.356436
	High school graduate	24012.275826
	college,less than 1 year	28201.210614
	college 1+ years, no degree	28488.347335
	Associate degree	35081.001821
	Bachelor,s degree	53294.264282
	Master.s degree	70755.173611
	Professional degree	94245.204918
	Doctorate degree	61467.676768
Female	Not in universe (Under 3 years)	0.000000
	No schooling completed	947.790323
	Nursery school to 4th grade	387.342995
	5th grade or 6th grade	2771.055195
	7th or 8th grade	2256.137405
	9th grade	3326.657609
	10th grade	4281.338798
	11th grade	5003.164557
	12th grade,no diploma	9200.885781
	High school graduate	11515.832571
	college,less than 1 year	16305.045514
	college 1+ years, no degree	16949.317489
	Associate degree	21991.480263
	Bachelor,s degree	33193.713496
	Master.s degree	44412.231834
	Professional degree	46821.961290
	Doctorate degree	41476.065574

And here is the heatmap :

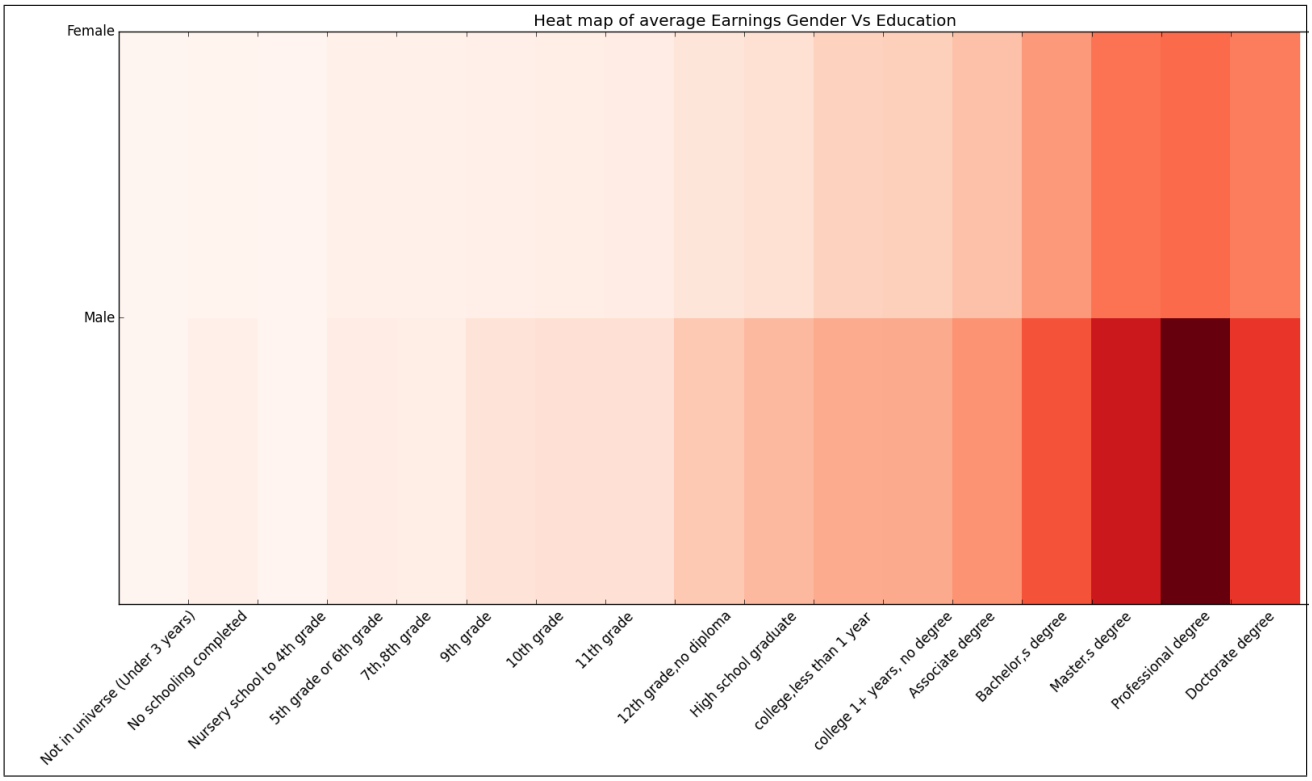


Fig. 5: Heatmap gender vs education in earnings

It's really cool. Have profissinoal degree or master is better than have a doctoral. I think that's because most doctoral degree work in education which is not the highest salaries.

Fourth Question

For this question I will show the relation between marital status , gender and earnings seemed interesting :) and here is the code :

```
table = pd.pivot_table(data,values='Earnings',index=['Sex', 'Marriage'],aggfunc=np.mean)
table = table.values
test=table.reshape(2,5)
fig = plt.figure()
ax = fig.add_subplot(111)
ax.pcolor(test,cmap=plt.cm.Reds,vmin=np.min(test), vmax=np.max(test))
ax.set_yticks([1,2])
ax.set_yticklabels(Genders)
ax.set_xticks(range(6))
ax.set_xticklabels(MarriageState)
for tick in ax.get_xticklabels():
    tick.set_rotation(45)
plt.gcf().subplots_adjust(bottom=0.20)
ax.set_title('Heat map of average Earnings Gender Vs Mariage')
plt.show()
plt.close()
```

The resulted heat map :

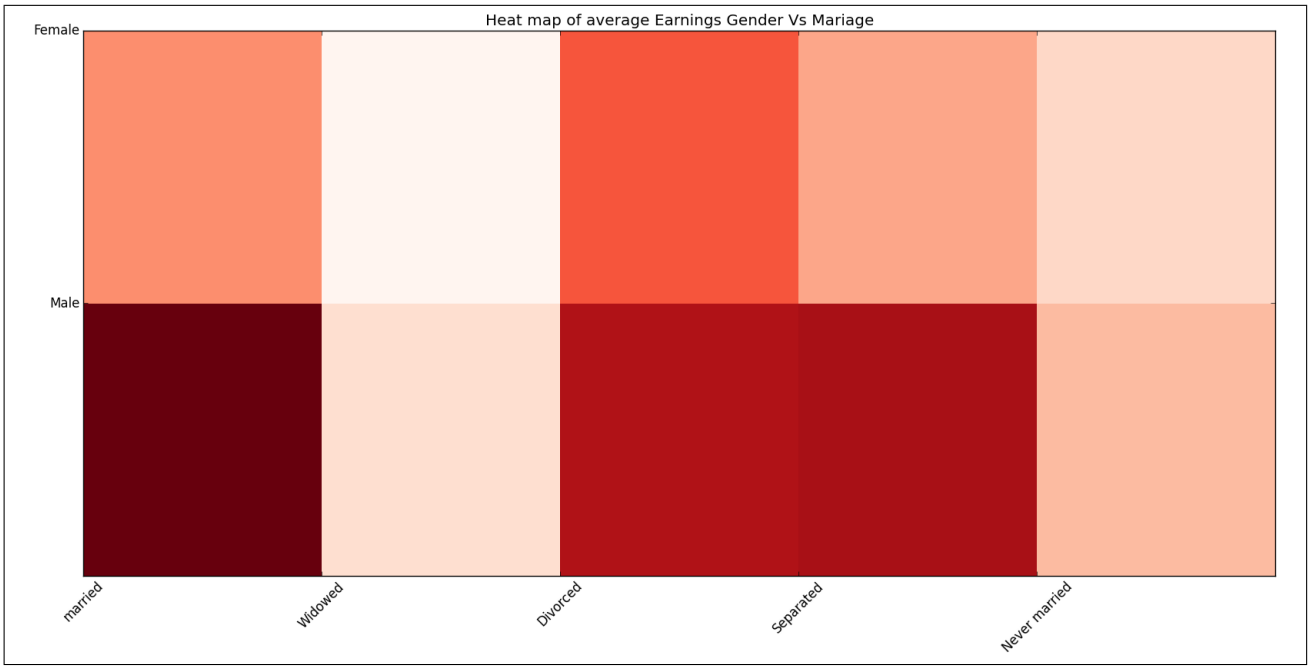


Fig. 6: Heatmap gender vs marriage status in earnings

depending on this data if you want to increase your earning just get married :D but don't kill your partner that will decrease your income dramatically.

Fifth Question

I did read most of the article but the operation were quite more understandable and clear on the wikipedia web page. For this task I'll aggregate (Sex,Marriage,Hours) as (X,Y,Z) please notice that Y we can convert it to days -¿ weeks depending on the value we have. The operations are :

1. **Slicing**: by this we can take one males or females.
2. **Dicing**: by this we can take group of hours (or group of days if we generalize this value).
3. **Drill-up and drill-down**:let's say Hours is genralized to days and we can generalize it to weeks and currently we are using days.At this point drill down will go to Hours for specific values and Drill-up will generalize to weeks.
4. **Pivoting**: is about changing axes like changing axes between sexes and hours.

To calculate the data cube I used the following code
Note: The following code depending on the previous code.

```
table = pd.pivot_table(data,values='Earnings',index=['Sex', 'Marriage','Hours'],aggfunc=np.
    mean)
```

The previous code will calculate the required data pivot but I could not find a good way to show interactive plot.

I hope I understood the question.

Sixth Question

For this question I used python as usual and used google TSP to increase the speed and performance. (even though it's not a straight forward solution). Here is the code :

```
__author__ = 'Modified by :aqeel'
import pandas as pd
import numpy as np
import RandomMatrix
import ShortestPathFinder
# Copyright 2010-2014 Google
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
```

```
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""Traveling Salesman Sample.
   This is a sample using the routing library python wrapper to solve a
   Traveling Salesman Problem.
   The description of the problem can be found here:
   http://en.wikipedia.org/wiki/Travelling_salesman_problem.
   The optimization engine uses local search to improve solutions, first
   solutions being generated using a cheapest addition heuristic.
   Optionally one can randomly forbid a set of random connections between nodes
   (forbidden arcs).
"""
import random
import gflags
from ortools.constraint_solver import pywrapcp
# function to Calculate the distance which can be replaced with any Type of Distance
# ex. Google API to Calculate the distance between two GPS Coordinations
def distance(p1, p2):
    dist = np.linalg.norm(p1-p2)
    return dist


class RandomMatrix(object):
    """Random matrix."""

    def __init__(self,points):
        self.lst = points
        """Initialize random matrix."""
        self.matrix = {}
        for from_node in xrange(len(self.lst)):
            self.matrix[from_node] = {}
            for to_node in xrange(len(self.lst)):
                if from_node == to_node:
                    self.matrix[from_node][to_node] = 0
                else:
                    self.matrix[from_node][to_node] = distance(self.lst[from_node], self.lst[
    to_node])
    def Distance(self, from_node, to_node):
        return self.matrix[from_node][to_node]
with open ('DM2016_org.csv') as f:
    d = {}
    headers = f.readline().split(' ')
    values = map(lambda x:x.split(),f.readlines())
    for i in range(len(values[0])):
        d[i]=[]
        for v in values:
            d[i].append(v[i])
data = pd.DataFrame(d)
npdata = data[data.columns[1:]].as_matrix()
npdata = np.array(npdata,dtype=int)
class ShortestPathFinder:
    def __init__(self, pointslist):
        reload(gflags)
        self.FLAGS = gflags.FLAGS
        self.points = pointslist

        gflags.DEFINE_integer('tsp_size', len(pointslist),
                              'Size of Traveling Salesman Problem instance.')
        gflags.DEFINE_boolean('tsp_use_random_matrix', True,
                              'Use random cost matrix.')
        gflags.DEFINE_integer('tsp_random_forbidden_connections', 0,
                              'Number of random forbidden connections.')
        gflags.DEFINE_integer('tsp_random_seed', 0, 'Random seed.')
        gflags.DEFINE_boolean('light_propagation', False, 'Use light propagation')

    def FindApproximateShortestWay(self):
        # Create routing model
        if self.FLAGS.tsp_size > 0:
```

```
            # Set a global parameter.
            param = pywrapcp.RoutingParameters()
            param.use_light_propagation = self.FLAGS.light_propagation
            pywrapcp.RoutingModel.SetGlobalParameters(param)

            # TSP of size FLAGS.tsp_size
            # Second argument = 1 to build a single tour (it's a TSP).
            # Nodes are indexed from 0 to FLAGS_tsp_size - 1, by default the start of
            # the route is node 0.
            routing = pywrapcp.RoutingModel(self.FLAGS.tsp_size, 1)

            parameters = pywrapcp.RoutingSearchParameters()
            # Setting first solution heuristic (cheapest addition).
            parameters.first_solution = 'PathCheapestArc'
            # Disabling Large Neighborhood Search, comment out to activate it.
            parameters.no_lns = True
            parameters.no_tsp = False

            # Setting the cost function.
            # Put a callback to the distance accessor here. The callback takes two
            # arguments (the from and to node inidices) and returns the distance between
            # these nodes.
            matrix = RandomMatrix(self.points)
            matrix_callback = matrix.Distance
            if self.FLAGS.tsp_use_random_matrix:
                routing.SetArcCostEvaluatorOfAllVehicles(matrix_callback)
            else:
                routing.SetArcCostEvaluatorOfAllVehicles(RandomMatrix.distance)
            # Forbid node connections (randomly).
            rand = random.Random()
            rand.seed(self.FLAGS.tsp_random_seed)
            forbidden_connections = 0
            while forbidden_connections < self.FLAGS.tsp_random_forbidden_connections:
                from_node = rand.randrange(self.FLAGS.tsp_size - 1)
                to_node = rand.randrange(self.FLAGS.tsp_size - 1) + 1
                if routing.NextVar(from_node).Contains(to_node):
                    print 'Forbidding connection ' + str(from_node) + ' -> ' + str(to_node)
                    routing.NextVar(from_node).RemoveValue(to_node)
                    forbidden_connections += 1

            # Solve, returns a solution if any.
            assignment = routing.SolveWithParameters(parameters, None)
            if assignment:
                # Solution cost.
                cost =  assignment.ObjectiveValue()
                # Inspect solution.
                # Only one route here; otherwise iterate from 0 to routing.vehicles() - 1
                route_number = 0
                node = routing.Start(route_number)
                result =[]
                while not routing.IsEnd(node):
                    result.append(self.points[int(node)])
                    node = assignment.Value(routing.NextVar(node))
                result.append(self.points[0])
                return cost,result
            else:
                print 'No solution found.'
        else:
            print 'Specify an instance greater than 0.'
resut = ShortestPathFinder(npdata)
totaldistance,plan = resut.FindApproximateShortestWay()
lst=[]
i=0
for p in plan:
    lst.append(np.argmax(np.sum(npdata==p,axis=1)))
    i+=1
    if i%100==0:
        print i
output = map(lambda x:data[data.index[0]][x],lst)
f = open('output.txt','w')
for element in output:
    f.write('\n{}'.format(element))
f.close()
```

The output Image was really close :

Fig. 7: Shows the scrumbled image after applying TSP

The problem in Figure 4 is deciding the first row. To decide which is the best cut I just searched for maximum distance between two neighbours and take the first one as the start.Here is the code that should be added the the previous one to fix the problem:

```
#Find the Best Cut
maxvalue=0
maxindex=-1
for i in range(len(plan)-1):
    dist = np.linalg.norm(plan[i]-plan[i+1])
    if (dist>maxvalue):
        maxvalue = dist
        maxindex=i

lst2=output[maxindex:] + output[:maxindex]
```

Fig. 8: Image after optimizing the start point

Just for fun I applied the same software in the second image and I got the following result:

Fig. 9: Image 2 after recovering using TSP with optimized start point

Seventh Question

For this question I used mysql.And I used the following steps to achieve the pivot table using sql

1. Created table in the database using the following code:

```
CREATE TABLE `census` (`State` varchar(50) DEFAULT NULL, `House_id` varchar(50) DEFAULT
    NULL, `Weight` varchar(50) DEFAULT NULL, `House_relation` varchar(50) DEFAULT NULL,
     `Sex` varchar(50) DEFAULT NULL, `Age` varchar(50) DEFAULT NULL, `Race` varchar(50)
    DEFAULT NULL, `Marriage` varchar(50) DEFAULT NULL, `Education` varchar(50) DEFAULT
    NULL, `Ancestry` varchar(50) DEFAULT NULL, `Language` varchar(50) DEFAULT NULL, `
    Employment_status` varchar(50) DEFAULT NULL, `Traveltime` varchar(50) DEFAULT NULL,
    `Industry` varchar(50) DEFAULT NULL, `Occupation` varchar(50) DEFAULT NULL, `Hours`
    varchar(50) DEFAULT NULL, `Weeks` varchar(50) DEFAULT NULL, `Salary` varchar(50)
    DEFAULT NULL, `Income` varchar(50) DEFAULT NULL, `Earnings` int(20) DEFAULT NULL)
```

2. After that I loaded the data into the table using the following code :

```
#Show the directory that mysql can accept files from
SHOW VARIABLES LIKE "secure_file_priv";
LOAD DATA INFILE '/var/lib/mysql-files/extract_large.csv'
INTO TABLE census
FIELDS TERMINATED BY ';'
```

3. At the end, I used the following code to generate the pivot table :

```
SELECT Sex,Education,avg(Earnings)
FROM census
group by Sex,Education
```

   Which clearly find the average earning for each set of (Sex,Education)

Here is the output table :

Sex	Education	Avg(Earnings)
1	00	0.0000
1	01	3436.9163
1	02	1018.5143
1	03	5719.2300
1	04	4585.9244
1	05	7892.1493
1	06	8702.8604
1	07	10341.6579
1	08	16704.6764
1	09	22399.7355
1	10	26831.3657
1	11	29900.6087
1	12	35249.8174
1	13	52814.0881
1	14	63902.4224
1	15	97160.4380
1	16	69790.5412
2	00	0.0000
2	01	1358.9230
2	02	386.2795
2	03	1996.0910
2	04	1656.8459
2	05	3048.7202
2	06	3878.9171
2	07	4711.3237
2	08	7794.4182
2	09	10865.3802
2	10	15102.8411
2	11	16310.1780
2	12	20708.8457
2	13	27804.7783
2	14	35058.9334
2	15	44529.5096
2	16	44325.0832

Note: All .py,.ipython,.tex,.pdf etc.. exist ongithub

References

[1] SciPy Hierarchical Clustering and Dendrogram Tutorial

E.O.F