

Data Mining

Home work 12

Clustering, PCA, OLAP

Aqeel Labash
Lecturer: Jaak Vilo

May 14, 2016

First Question

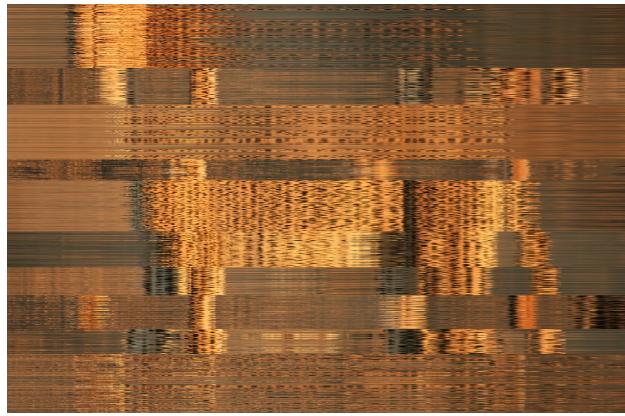
For this question I tried K-means. I used the following code :

```
1 import numpy as np
2 import pandas as pd
3 from sklearn.cluster import KMeans
4 with open ('DM2016.org.txt') as f:
5     d = {}
6     headers = f.readline().split(' ')
7     values = map(lambda x:x.split(),f.readlines())
8     for i in range(len(values[0])):
9         d[i] = []
10        for v in values:
11            d[i].append(v[i])
12 data = pd.DataFrame(d)
13 npdata = data[data.columns[1:]].as_matrix()
14 npdata = np.array(npdata,dtype=int)
15 random_state = 170
16 nc = 339
17 y_pred = KMeans(n_clusters=nc, random_state=random_state, precompute_distances=True).
18     fit_predict(npdata)
19 output = []
20 for i in range(nc):
21     print i
22     map(lambda x:output.append(x),map(lambda x:data[data.index[0]][x].values,np.where(y_pred ==
23 i))[0])
24 f = open('Kmeans_399.txt','w')
25 for element in output:
26     f.write('\n{}'.format(element))
27 f.close()
```

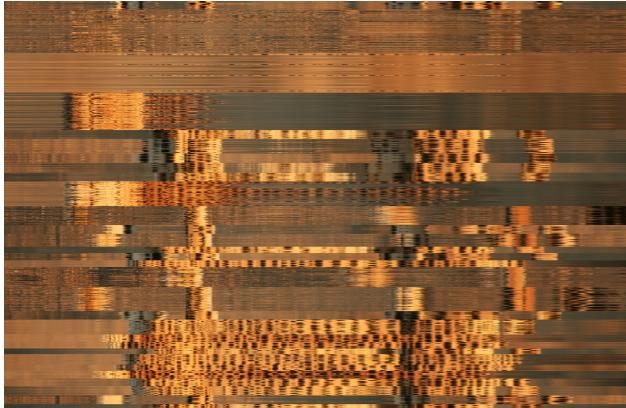
I tried 3 values as number of clusters 2,10,30,399. The results were as following:



(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 dendrogram was as following :

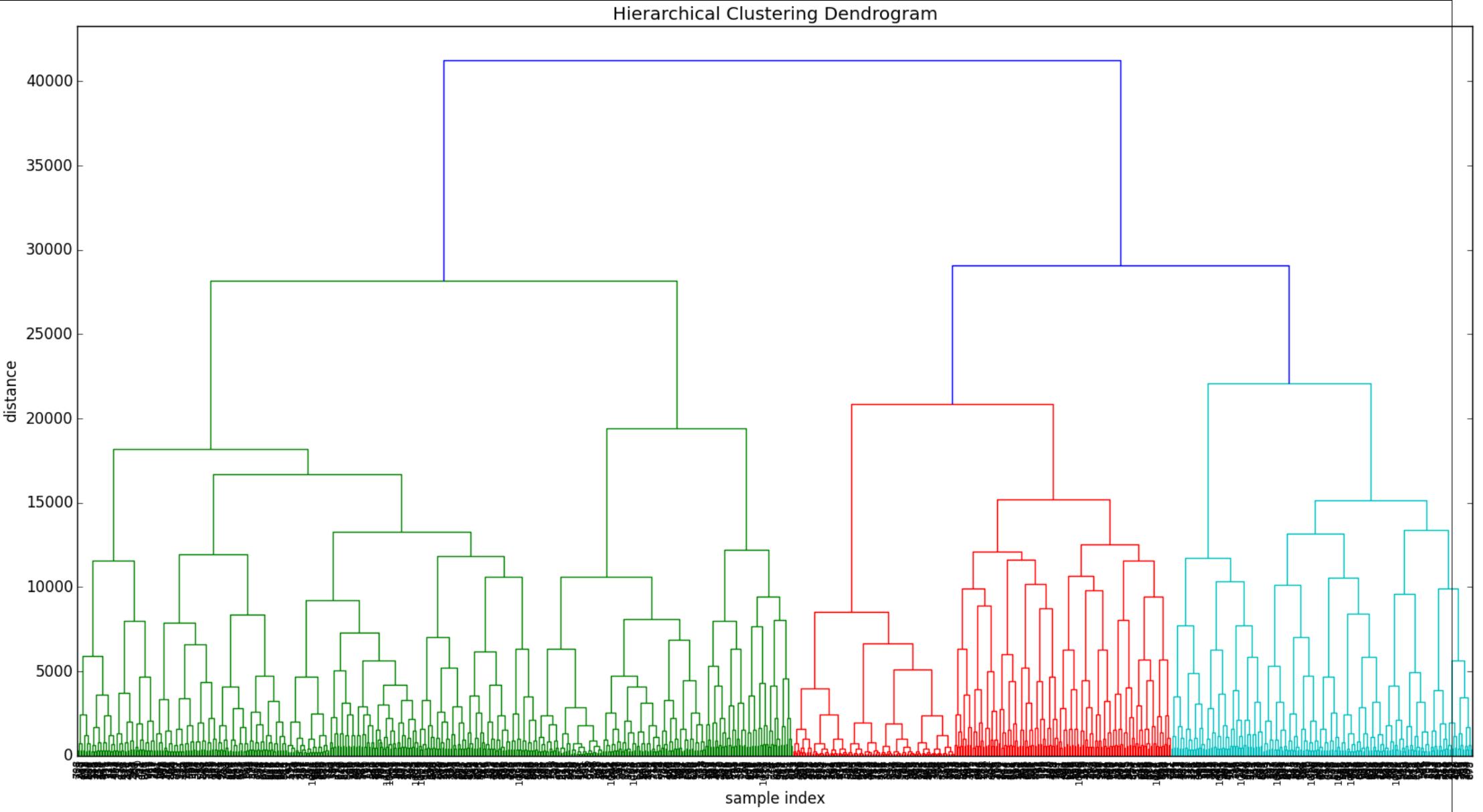


Fig. 2: Full dendrogram

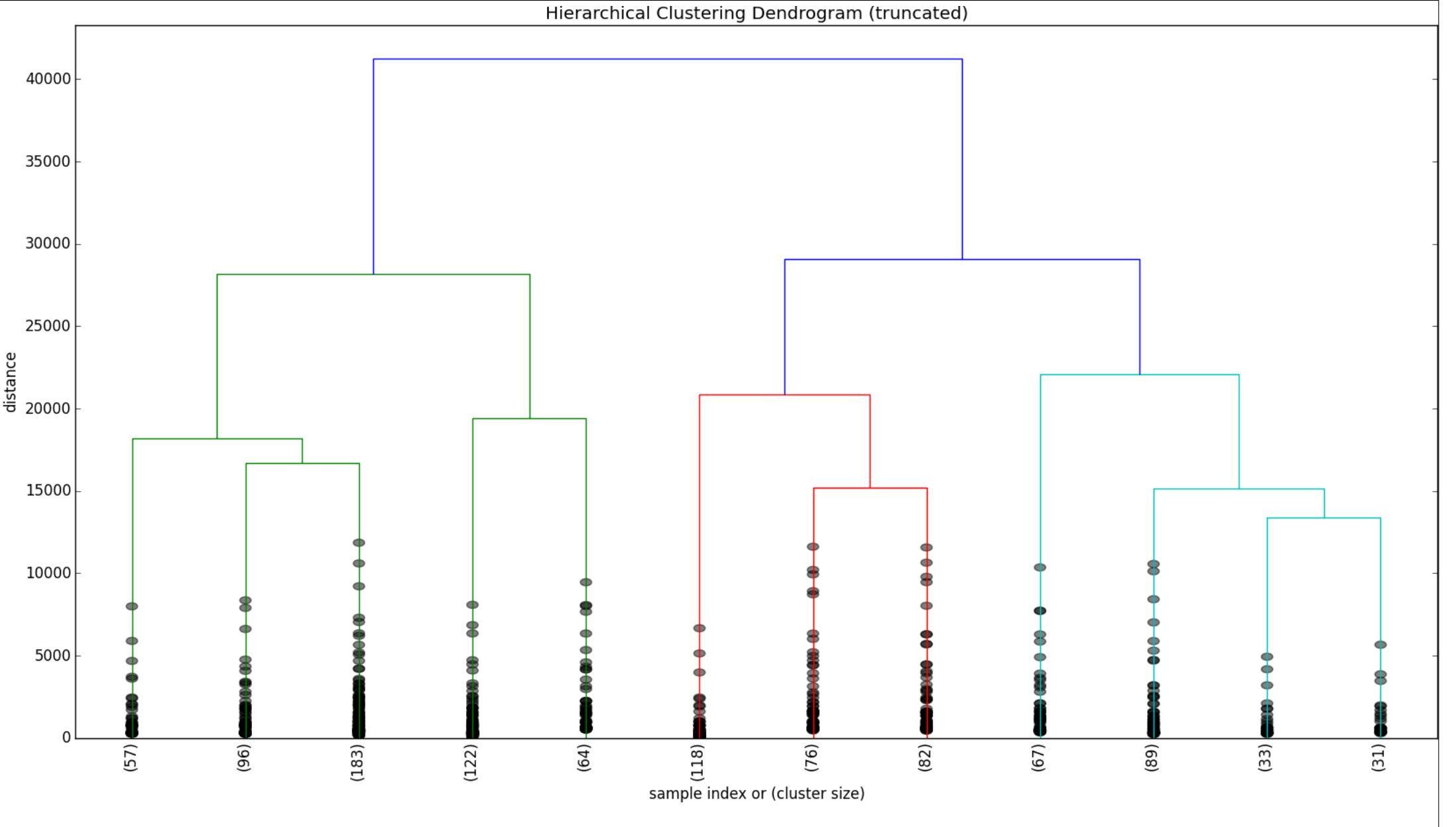


Fig. 3: Truncated Dendogram

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 :

```
1 import numpy as np
2 from sklearn.decomposition import PCA
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 from mpl_toolkits.mplot3d import Axes3D
6 with open ('DM2016_org.csv') as f:
7     d = {}
8     headers = f.readline().split(' ')
9     values = map(lambda x:x.split(),f.readlines())
10    for i in range(len(values)):
11        d[headers[i]] = []
12        for v in values:
13            d[headers[i]].append(v[i])
14 data = pd.DataFrame(d)
15 npdata = data[data.columns[1:]].as_matrix()
16 pca = PCA(n_components=3)
17 pca.fit(npdata)
18 output = pca.transform(npdata)
19 fig = plt.figure()
20 ax = fig.add_subplot(111, projection='3d')
21 n = 100
22 for x, y, z in output:
23     ax.scatter(x, y, z)
24
25 ax.set_xlabel('X Label')
26 ax.set_ylabel('Y Label')
27 ax.set_zlabel('Z Label')
28
29 plt.show()
```

And here is the output figure:

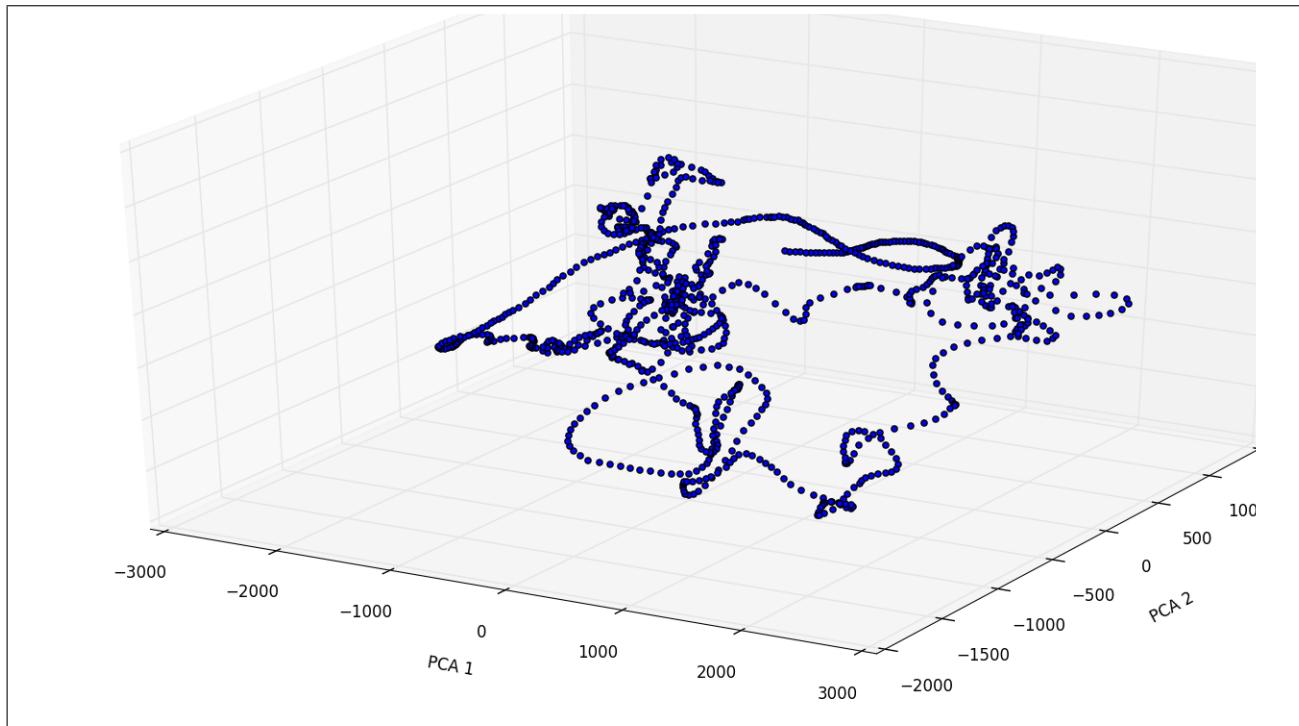


Fig. 4: X: PCA1 , Y:PCA2 , Z:PCA3

Third Question

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

```
1 import pandas as pd
2 import numpy as np
```

```

3 %matplotlib inline
4 import matplotlib.pyplot as plt
5 data = pd.read_csv('extract_medium.csv',sep=';')
6 data.head(1)
7 table = pd.pivot_table(data, values='Earnings',index=['Sex', 'Education'],aggfunc=np.mean)
8 ######
9 temp1 = data.groupby(['Sex', 'Education']).Earnings.mean()
10 temp1
11 temp1 = temp1.values
12 test=temp1.reshape(2,17)
13 test.shape
14 plt.pcolor(test,cmap=plt.cm.Reds,vmin=np.min(test), vmax=np.max(test))
15 plt.yticks([0,1],range(3))
16 plt.xticks(range(17),range(17))
17 plt.title('Heat map of average Earnings Gender Vs Education')
18 plt.show()
19 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 :

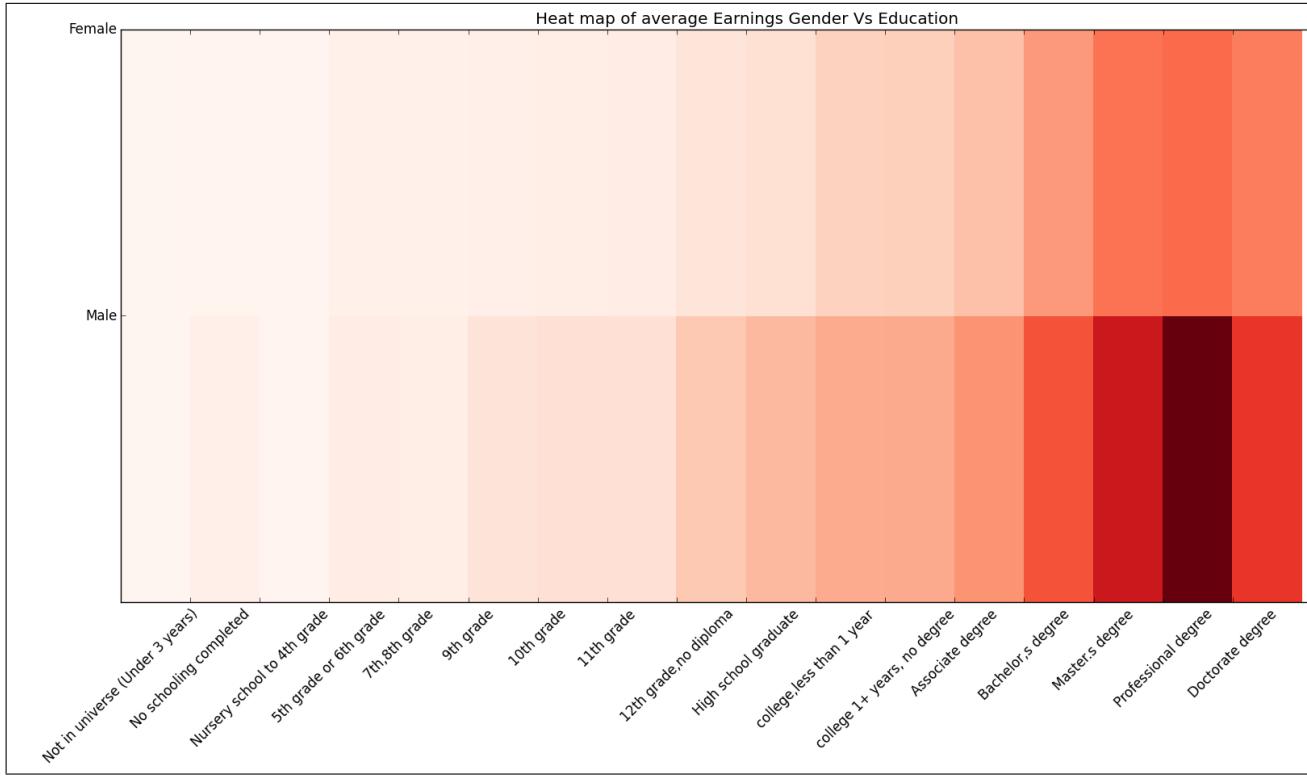


Fig. 5: Heatmap gender vs education in earnings

It's really cool. Have professional 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 :

```

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

The resulted heat map :

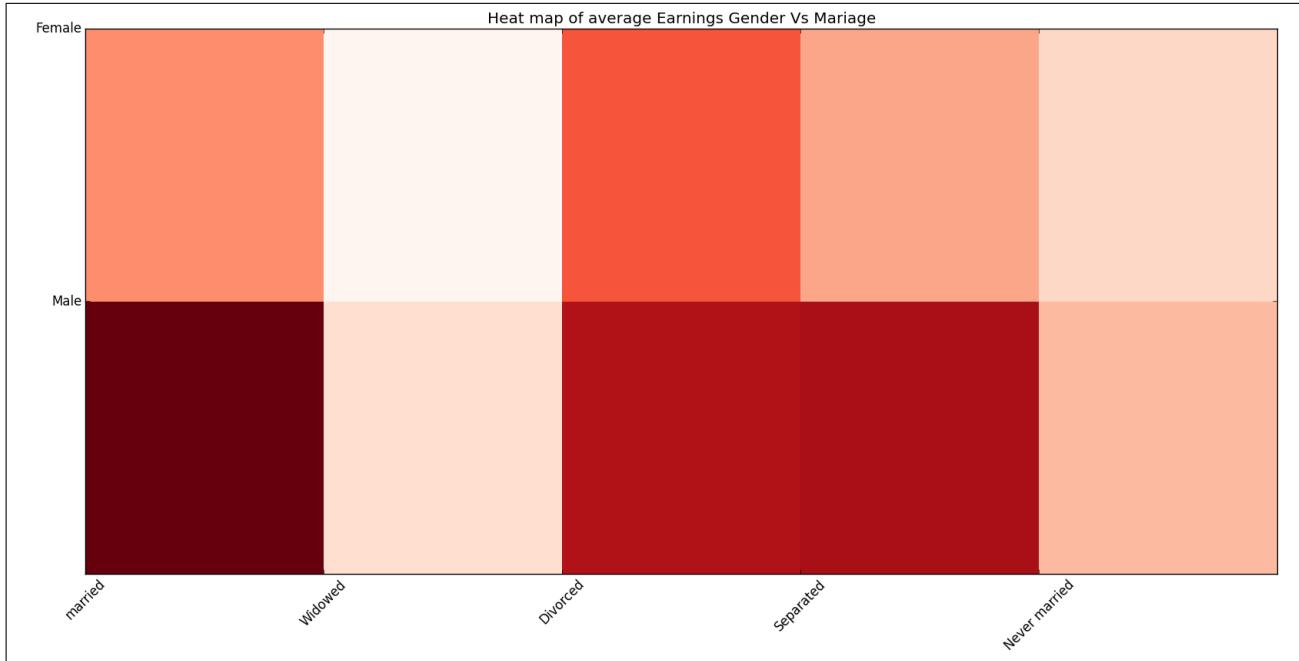


Fig. 6: Heatmap gender vs marriage status in earnings

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 generalized 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.

```
1 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 :

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

```

9 # You may obtain a copy of the License at
10 #
11 #      http://www.apache.org/licenses/LICENSE-2.0
12 #
13 # Unless required by applicable law or agreed to in writing, software
14 # distributed under the License is distributed on an "AS IS" BASIS,
15 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
16 # See the License for the specific language governing permissions and
17 # limitations under the License.
18
19 """ Traveling Salesman Sample.
20 This is a sample using the routing library python wrapper to solve a
21 Traveling Salesman Problem.
22 The description of the problem can be found here:
23 http://en.wikipedia.org/wiki/Travelling_salesman_problem.
24 The optimization engine uses local search to improve solutions, first
25 solutions being generated using a cheapest addition heuristic.
26 Optionally one can randomly forbid a set of random connections between nodes
27 (forbidden arcs).
28 """
29 import random
30 import gflags
31 from ortools.constraint_solver import pywrapcp
32 # function to Calculate the distance which can be replaced with any Type of Distance
33 # ex. Google API to Calculate the distance between two GPS Coordinations
34 def distance(p1, p2):
35     dist = np.linalg.norm(p1-p2)
36     return dist
37
38
39 class RandomMatrix(object):
40     """Random matrix."""
41
42     def __init__(self, points):
43         self.lst = points
44         """Initialize random matrix."""
45         self.matrix = {}
46         for from_node in xrange(len(self.lst)):
47             self.matrix[from_node] = {}
48             for to_node in xrange(len(self.lst)):
49                 if from_node == to_node:
50                     self.matrix[from_node][to_node] = 0
51                 else:
52                     self.matrix[from_node][to_node] = distance(self.lst[from_node], self.lst[to_node])
53     def Distance(self, from_node, to_node):
54         return self.matrix[from_node][to_node]
55 with open('DM2016_org.csv') as f:
56     d = {}
57     headers = f.readline().split(' ')
58     values = map(lambda x:x.split(),f.readlines())
59     for i in range(len(values[0])):
60         d[i]=[]
61         for v in values:
62             d[i].append(v[i])
63 data = pd.DataFrame(d)
64 npdata = data[data.columns[1:]].as_matrix()
65 npdata = np.array(npdata,dtype=int)
66 class ShortestPathFinder:
67     def __init__(self, pointslist):
68         reload(gflags)
69         self.FLAGS = gflags.FLAGS
70         self.points = pointslist
71
72         gflags.DEFINE_integer('tsp_size', len(pointslist),
73                               'Size of Traveling Salesman Problem instance.')
74         gflags.DEFINE_boolean('tsp_use_random_matrix', True,
75                               'Use random cost matrix.')
76         gflags.DEFINE_integer('tsp_random_forbidden_connections', 0,
77                               'Number of random forbidden connections.')
78         gflags.DEFINE_integer('tsp_random_seed', 0, 'Random seed.')
79         gflags.DEFINE_boolean('light_propagation', False, 'Use light propagation')
80
81     def FindApproximateShortestWay(self):
82         # Create routing model
83         if self.FLAGS.tsp_size > 0:

```

```

84 # Set a global parameter.
85 param = pywrapcp.RoutingParameters()
86 param.use_light_propagation = self.FLAGS.light_propagation
87 pywrapcp.RoutingModel.SetGlobalParameters(param)
88
89 # TSP of size FLAGS.tsp_size
90 # Second argument = 1 to build a single tour (it's a TSP).
91 # Nodes are indexed from 0 to FLAGS.tsp_size - 1, by default the start of
92 # the route is node 0.
93 routing = pywrapcp.RoutingModel(self.FLAGS.tsp_size, 1)
94
95 parameters = pywrapcp.RoutingSearchParameters()
96 # Setting first solution heuristic (cheapest addition).
97 parameters.first_solution = 'PathCheapestArc'
98 # Disabling Large Neighborhood Search, comment out to activate it.
99 parameters.no_lns = True
100 parameters.no_tsp = False
101
102 # Setting the cost function.
103 # Put a callback to the distance accessor here. The callback takes two
104 # arguments (the from and to node indices) and returns the distance between
105 # these nodes.
106 matrix = RandomMatrix(self.points)
107 matrix_callback = matrix.Distance
108 if self.FLAGS.tsp_use_random_matrix:
109     routing.SetArcCostEvaluatorOfAllVehicles(matrix_callback)
110 else:
111     routing.SetArcCostEvaluatorOfAllVehicles(RandomMatrix.distance)
112 # Forbid node connections (randomly).
113 rand = random.Random()
114 rand.seed(self.FLAGS.tsp_random_seed)
115 forbidden_connections = 0
116 while forbidden_connections < self.FLAGS.tsp_random_forbidden_connections:
117     from_node = rand.randrange(self.FLAGS.tsp_size - 1)
118     to_node = rand.randrange(self.FLAGS.tsp_size - 1) + 1
119     if routing.NextVar(from_node).Contains(to_node):
120         print 'Forbidding connection ' + str(from_node) + ' -> ' + str(to_node)
121         routing.NextVar(from_node).RemoveValue(to_node)
122         forbidden_connections += 1
123
124 # Solve, returns a solution if any.
125 assignment = routing.SolveWithParameters(parameters, None)
126 if assignment:
127     # Solution cost.
128     cost = assignment.ObjectiveValue()
129     # Inspect solution.
130     # Only one route here; otherwise iterate from 0 to routing.vehicles() - 1
131     route_number = 0
132     node = routing.Start(route_number)
133     result = []
134     while not routing.IsEnd(node):
135         result.append(self.points[int(node)])
136         node = assignment.Value(routing.NextVar(node))
137     result.append(self.points[0])
138     return cost, result
139 else:
140     print 'No solution found.'
141 else:
142     print 'Specify an instance greater than 0.'
143 resut = ShortestPathFinder(npdata)
144 totaldistance, plan = resut.FindApproximateShortestWay()
145 lst=[]
146 i=0
147 for p in plan:
148     lst.append(np.argmax(np.sum(npdata==p, axis=1)))
149     i+=1
150     if i%100==0:
151         print i
152 output = map(lambda x:data[data.index[0]][x],lst)
153 f = open('output.txt', 'w')
154 for element in output:
155     f.write('\n{}'.format(element))
156 f.close()

```

The output Image was really close :

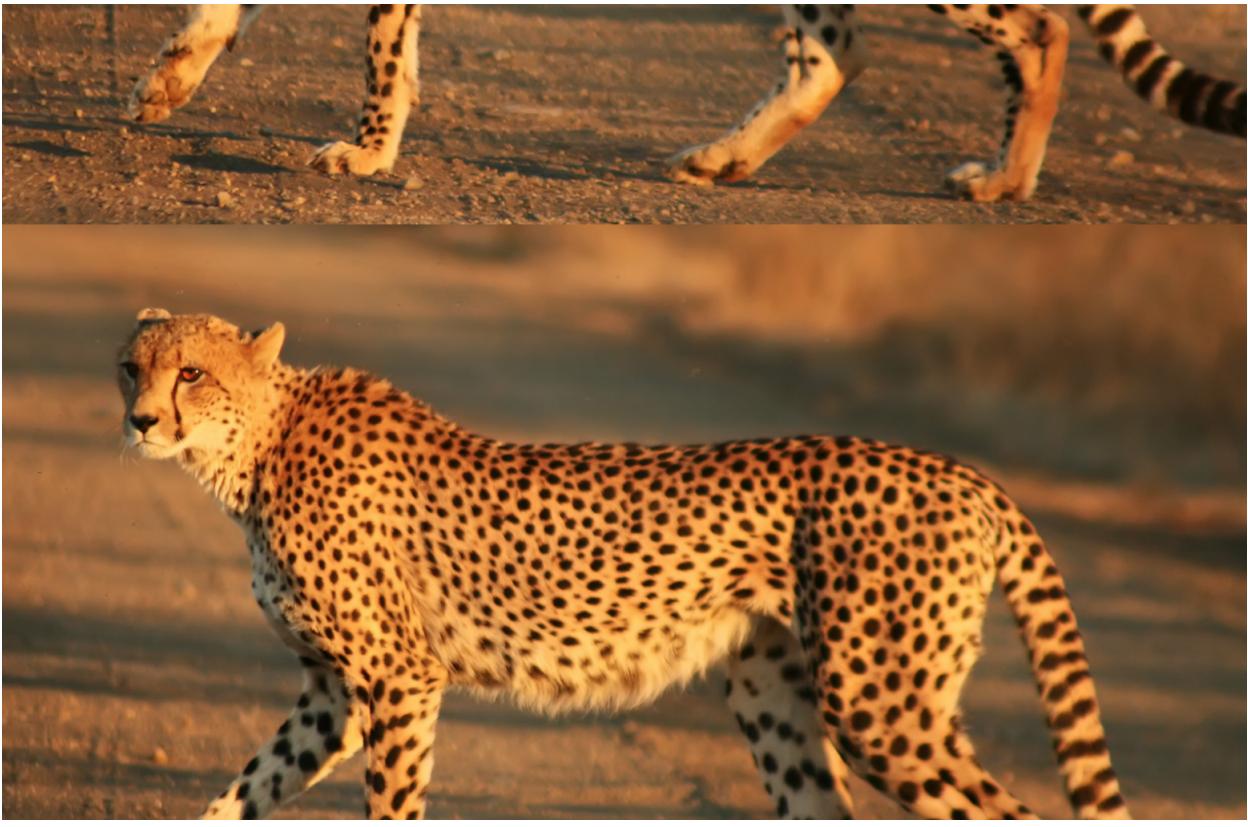


Fig. 7: Shows the scrambled image after applying TSP

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 to the previous one to fix the problem:

```
1 #Find the Best Cut
2 maxvalue=0
3 maxindex=-1
4 for i in range(len(plan)-1):
5     dist = np.linalg.norm(plan[i]-plan[i+1])
6     if (dist>maxvalue):
7         maxvalue = dist
8         maxindex=i
9
10 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:

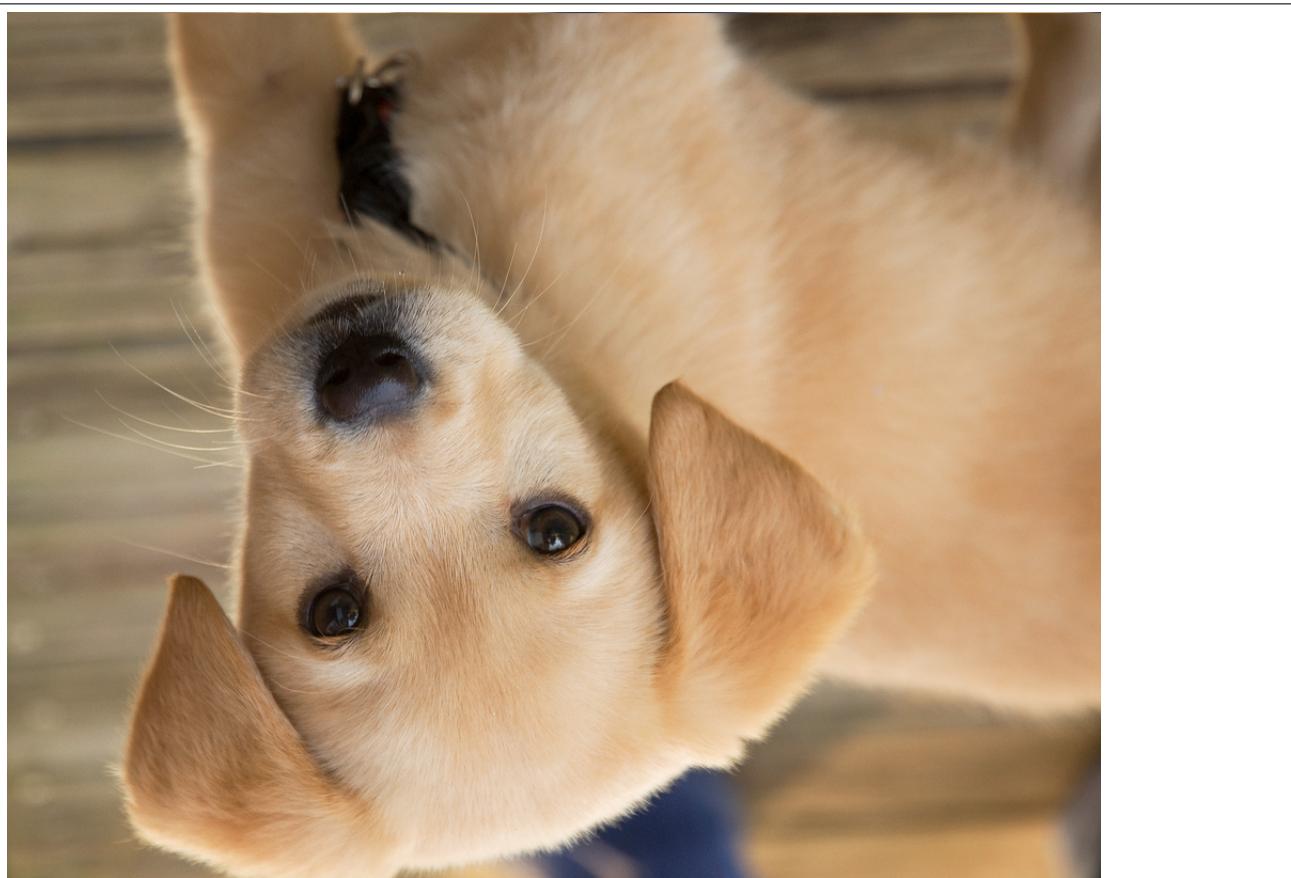


Fig. 9: Image 2 after recovering using TSP with optimized start point

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:

```
1 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, 'Travelttime' 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 :

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

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

```
1 SELECT Sex, Education, avg(Earnings)
2 FROM census
3 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

References

[1] SciPy Hierarchical Clustering and Dendrogram Tutorial

Note:All .py,.ipython,.tex,.pdf etc.. exist on github

E.O.F