CS2006 Python Practical 2

The first requirement for the practical is to refine the dataset.

In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patch
from ipywidgets import *
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D, get_test_data
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from matplotlib import cm
from operator import itemgetter

import pandas as pd
import numpy as np
import plotly
import plotly.graph_objs as go
import plotly.plotly as py

from ipywidgets import widgets
from IPython.display import display
from plotly.graph_objs import *
from plotly.widgets import GraphWidget
from plotly.offline import download_plotlyjs, init_notebook_mode, plot,
iplot
plotly.offline.init_notebook_mode(connected=True)

import cufflinks as cf
init_notebook_mode(connected=True)
```

So first to make sure the data is initially read in correctly, the columns which should be numbers (and not containing any strings) are selected and forced into numeric columns. This means that if any of the columns had invalid data, in the format of string for example, then that cell gets changed to 'NaN'.

In [6]:
```
low_memory = False
df=pd.read_csv("../data/census2011.csv")
#df=pd.read_csv("census2011.csv")
df[['Person ID','Family Composition', 'Population Base', 'Sex', 'Age',
    'Marital Status','Student', 'Country of Birth', 'Health', 'Ethnic Gr
oup',
    'Religion', 'Economic Activity', 'Occupation', 'Industry', 'Hours wo
rked per week', 'Approximated Social Grade']]= df[['Person ID','Family C
omposition', 'Population Base', 'Sex', 'Age',
    'Marital Status','Student', 'Country of Birth', 'Health', 'Ethnic Gr
oup',
    'Religion', 'Economic Activity', 'Occupation', 'Industry', 'Hours wo
rked per week', 'Approximated Social Grade']].apply(pd.to_numeric, error
s='coerce')
```

As shown below, the data types of the data set are numeric where expected.

In [7]:
```
df.dtypes
```

Out[7]:
```
Person ID                    int64
Region                      object
Residence Type              object
Family Composition           int64
Population Base              int64
Sex                          int64
Age                          int64
Marital Status               int64
Student                      int64
Country of Birth             int64
Health                       int64
Ethnic Group                 int64
Religion                     int64
Economic Activity            int64
Occupation                   int64
Industry                     int64
Hours worked per week        int64
Approximated Social Grade    int64
dtype: object
```

To actually refine the data, any rows containing a cell which is 'NaN' gets dropped, and any columns which are of type 'float' gets formatted.

In [8]:
```
refinedData = df.copy()
refinedData = refinedData.dropna()
pd.options.display.float_format = '{:,.0f}'.format
```

More refining includes dropping any rows which are duplicates of another.

In [9]:
```
refinedData = refinedData.drop_duplicates()
```

The final refining that takes place is to make sure all the values which are in the data set are valid values for it's column, meaning that all of the values are one of options specified in the file 'MicroDataTeachingVariables.pdf'.

In [10]:
```
#the different values each column can be
regions = ["E12000001","E12000002","E12000003","E12000004","E12000005","
E12000006","E12000007","E12000008","E12000009","W92000004"]
residenceType = ["C","H"]
famComp = [1,2,3,4,5,6,-9]
popBase = [1,2,3]
sex = [1,2]
age = [1,2,3,4,5,6,7,8]
maritalStatus = [1,2,3,4,5]
student = [1,2]
countryOfBirth = [1,2,-9]
health = [1,2,3,4,5,-9]
ethnicGroup = [1,2,3,4,5,-9]
religion = [1,2,3,4,5,6,7,8,9,-9]
econActivity = [1,2,3,4,5,6,7,8,9,-9]
occupation = [1,2,3,4,5,6,7,8,9,-9]
industry = [1,2,3,4,5,6,7,8,9,10,11,12,-9]
hoursWorkPerWeek = [1,2,3,4,-9]
approxSocialGrade = [1,2,3,4,-9]

#keeps track of all the rows which has invalid rows
rowsToRemove = []

#going through each column and checking if the values are valid
rCounter = 0
regs = refinedData['Region']
for reg in regs:
    flag = 1
    for region in regions:
                if region == reg:
                    flag = 0
    if flag == 1:
        rowsToRemove.append(rCounter)
    rCounter = rCounter + 1

rCounter = 0
res = refinedData['Residence Type']
for r in res:
    flag = 1
    for resType in residenceType:
                if resType == r:
                    flag = 0
    if flag == 1:
        rowsToRemove.append(rCounter)
    rCounter = rCounter + 1

rCounter = 0
family = refinedData['Family Composition']
for comp in family:
    flag = 1
    for f in famComp:
                if f == comp:
                    flag = 0
    if flag == 1:
        rowsToRemove.append(rCounter)
    rCounter = rCounter + 1

rCounter = 0
population = refinedData['Population Base']
for base in population:
    flag = 1
    for pop in popBase:
                if pop == base:
                    flag = 0
    if flag == 1:
        rowsToRemove.append(rCounter)
    rCounter = rCounter + 1
```

The second basic requirement is to perform descriptive analysis of the dataset.

The first descriptive analysis task is to determine the total number of records in the dataset.

In [11]:
```
len(refinedData)
```

Out[11]:
```
569741
```

The second descriptice analysis task is to determine the type of each variable in the dataset.

In [12]:
```
refinedData.dtypes
```

Out[12]:
```
Person ID                    int64
Region                      object
Residence Type              object
Family Composition           int64
Population Base              int64
Sex                          int64
Age                          int64
Marital Status               int64
Student                      int64
Country of Birth             int64
Health                       int64
Ethnic Group                 int64
Religion                     int64
Economic Activity            int64
Occupation                   int64
Industry                     int64
Hours worked per week        int64
Approximated Social Grade    int64
dtype: object
```

The third descriptive analysis task is for each variable, except "Person ID", find all values that it takes, and the number of occurances for each value.

In [13]:
```
region = refinedData.groupby('Region')
region.size()
```

Out[13]:
```
Region
E12000001    26349
E12000002    71436
E12000003    53471
E12000004    45782
E12000005    56875
E12000006    59411
E12000007    83582
E12000008    88084
E12000009    53774
W92000004    30977
dtype: int64
```

In [14]:
```
residenceType = refinedData.groupby('Residence Type')
residenceType.size()
```

Out[14]:
```
Residence Type
C     10654
H    559087
dtype: int64
```

In [15]:
```
familyComposition = refinedData.groupby('Family Composition')
familyComposition.size()
```

Out[15]:
```
Family Composition
-9     18851
 1     96690
 2    300962
 3     72641
 4      9848
 5     64519
 6      6230
dtype: int64
```

In [16]:
```
populationBase = refinedData.groupby('Population Base')
populationBase.size()
```

Out[16]:
```
Population Base
1    561040
2      6730
3      1971
dtype: int64
```

In [17]:
```
sex = refinedData.groupby('Sex')
sex.size()
```

Out[17]:
```
Sex
1    280569
2    289172
dtype: int64
```

In [18]:
```
age = refinedData.groupby('Age')
age.size()
```

Out[18]:
```
Age
1    106832
2     72785
3     75948
4     78641
5     77388
6     65666
7     48777
8     43704
dtype: int64
```

In [19]:
```
maritalStatus = refinedData.groupby('Marital Status')
maritalStatus.size()
```

Out[19]:
```
Marital Status
1    270999
2    214180
3     11951
4     40713
5     31898
dtype: int64
```

In [20]:
```
student = refinedData.groupby('Student')
student.size()
```

Out[20]:
```
Student
1    126537
2    443204
dtype: int64
```

In [21]:
```
countryOfBirth = refinedData.groupby('Country of Birth')
countryOfBirth.size()
```

Out[21]:
```
Country of Birth
-9      6804
 1    485645
 2     77292
dtype: int64
```

In [22]:
```
health = refinedData.groupby('Health')
health.size()
```

Out[22]:
```
Health
-9      6804
 1    264971
 2    191744
 3     74480
 4     24558
 5      7184
dtype: int64
```

In [23]:
```
ethnicGroup = refinedData.groupby('Ethnic Group')
ethnicGroup.size()
```

Out[23]:
```
Ethnic Group
-9      6804
 1    483477
 2     12209
 3     42712
 4     18786
 5      5753
dtype: int64
```

In [24]:
```
religion = refinedData.groupby('Religion')
religion.size()
```

Out[24]:
```
Religion
-9      6804
 1    141658
 2    333481
 3      2538
 4      8214
 5      2572
 6     27240
 7      4215
 8      2406
 9     40613
dtype: int64
```

In [25]:
```
economicActivity = refinedData.groupby('Economic Activity')
economicActivity.size()
```

Out[25]:
```
Economic Activity
-9    112618
 1    216025
 2     40632
 3     18109
 4     14117
 5     97480
 6     24756
 7     17945
 8     17991
 9     10068
dtype: int64
```

In [26]:
```
occupation = refinedData.groupby('Occupation')
occupation.size()
```

Out[26]:
```
Occupation
-9    149984
 1     39788
 2     64111
 3     44937
 4     53254
 5     48546
 6     37297
 7     38523
 8     34818
 9     58483
dtype: int64
```

In [27]:
```
industry = refinedData.groupby('Industry')
industry.size()
```

Out[27]:
```
Industry
-9     149984
 1       3957
 2      53433
 3      30708
 4      68878
 5      25736
 6      35240
 7      16776
 8      49960
 9      24908
 10     40560
 11     49345
 12     20256
dtype: int64
```

In [28]:
```
hoursWorkedPerWeek = refinedData.groupby('Hours worked per week')
hoursWorkedPerWeek.size()
```

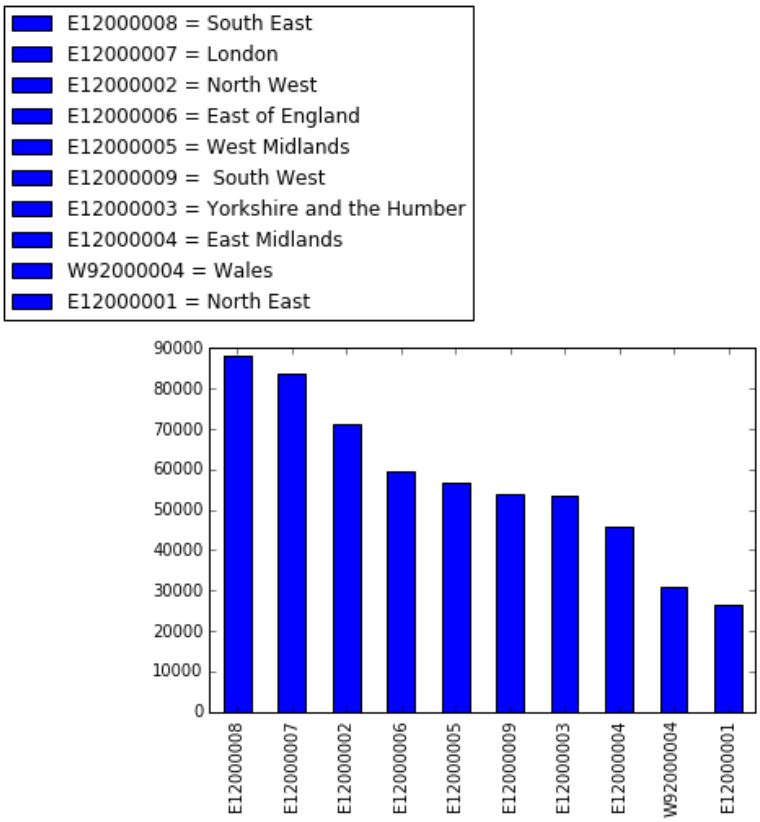
Out[28]:
```
Hours worked per week
-9    302321
 1     25776
 2     52133
 3    153938
 4     35573
dtype: int64
```

In [29]:
```
approximatedSocialGrade = refinedData.groupby('Approximated Social Grade
')
approximatedSocialGrade.size()
```

Out[29]:
```
Approximated Social Grade
-9    124103
 1     82320
 2    159642
 3     79936
 4    123740
dtype: int64
```

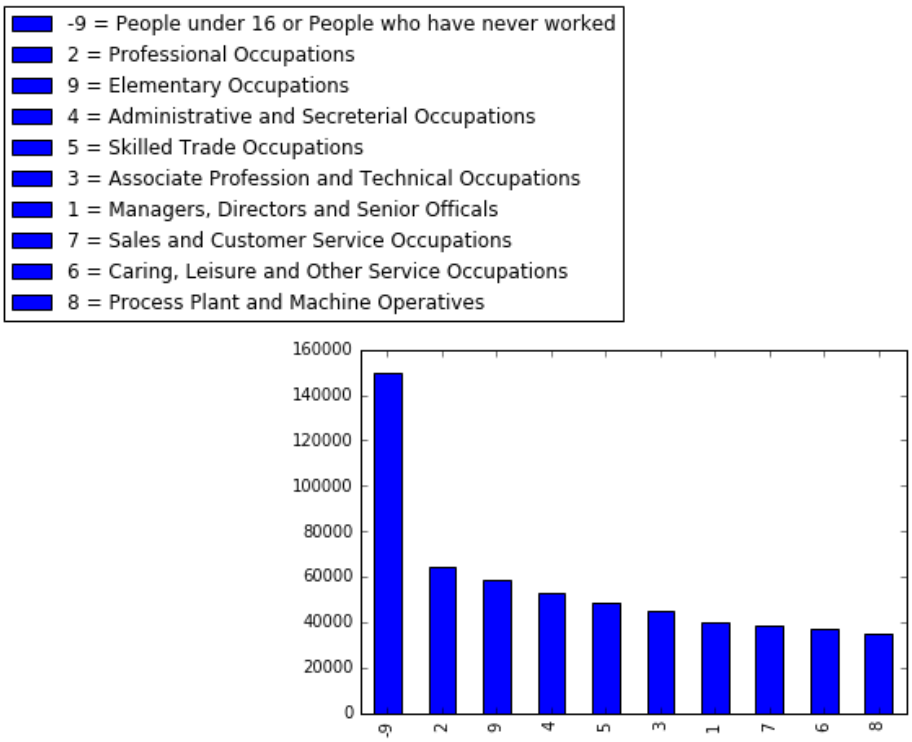
The third requirement was to build plots for the data specified in the practical specification. The first plot is a bar chart for the number of records for each region.

In [30]:
```
refinedData['Region'].value_counts().plot(kind="bar")

#the legend for the plot
region1_patch = patch.Patch(label='E12000008 = South East')
region2_patch = patch.Patch(label='E12000007 = London')
region3_patch = patch.Patch(label='E12000002 = North West')
region4_patch = patch.Patch(label='E12000006 = East of England')
region5_patch = patch.Patch(label='E12000005 = West Midlands')
region6_patch = patch.Patch(label='E12000009 =  South West')
region7_patch = patch.Patch(label='E12000003 = Yorkshire and the Humber'
)
region8_patch = patch.Patch(label='E12000004 = East Midlands')
region9_patch = patch.Patch(label='W92000004 = Wales')
region10_patch = patch.Patch(label='E12000001 = North East')

plt.legend(handles=[region1_patch,region2_patch,region3_patch,region4_pa
tch,region5_patch,region6_patch,region7_patch,region8_patch,region9_patc
h,region10_patch], loc = 'lower right', bbox_to_anchor=(0.5,1.05))
plt.show()
```



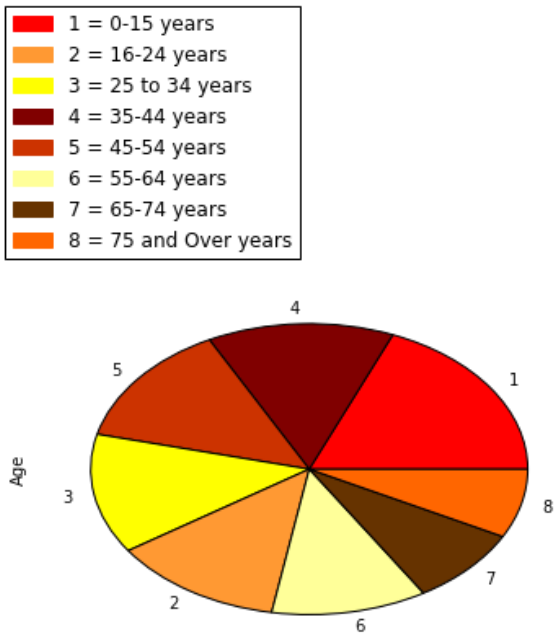
The second plot is a bar chart for the number of records for each occupation.

In [31]:
```
refinedData['Occupation'].value_counts().plot(kind="bar")

#the legend for the plot
occupation1_patch = patch.Patch(label='-9 = People under 16 or People wh
o have never worked')
occupation2_patch = patch.Patch(label='2 = Professional Occupations')
occupation3_patch = patch.Patch(label='9 = Elementary Occupations')
occupation4_patch = patch.Patch(label='4 = Administrative and Secreteria
l Occupations')
occupation5_patch = patch.Patch(label='5 = Skilled Trade Occupations')
occupation6_patch = patch.Patch(label='3 = Associate Profession and Tech
nical Occupations')
occupation7_patch = patch.Patch(label='1 = Managers, Directors and Senio
r Officals')
occupation8_patch = patch.Patch(label='7 = Sales and Customer Service Oc
cupations')
occupation9_patch = patch.Patch(label='6 = Caring, Leisure and Other Ser
vice Occupations')
occupation10_patch = patch.Patch(label='8 = Process Plant and Machine Op
eratives')

plt.legend(handles=[occupation1_patch,occupation2_patch,occupation3_patc
h,occupation4_patch,occupation5_patch,occupation6_patch,occupation7_patc
h,occupation8_patch,occupation9_patch,occupation10_patch], loc = 'lower
right', bbox_to_anchor=(0.5,1.05))
plt.show()
```



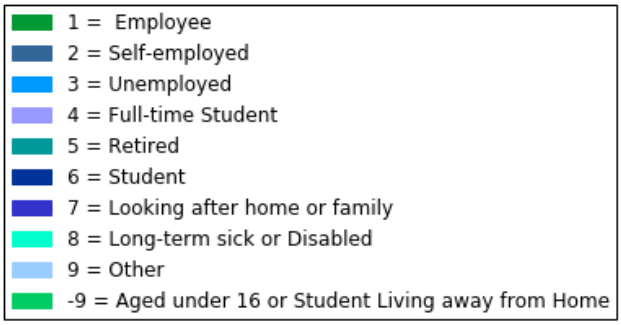
The third plot is a pie chart for the distribution of the sample by age.

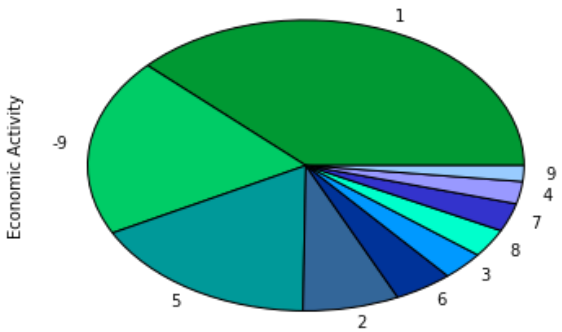
In [32]:
```
colors = ['#ff0000', '#800000', '#cc3300', '#ffff00', '#ff9933', '#ffff9
9', '#663300', '#ff6600']

refinedData['Age'].value_counts().plot(kind="pie", colors = colors)

#the legend for the plot
five_patch = patch.Patch(color='#cc3300', label='5 = 45-54 years')
four_patch = patch.Patch(color='#800000', label='4 = 35-44 years')
two_patch = patch.Patch(color='#ff9933', label='2 = 16-24 years')
six_patch = patch.Patch(color='#ffff99', label='6 = 55-64 years')
seven_patch = patch.Patch(color='#663300', label='7 = 65-74 years')
one_patch = patch.Patch(color='#ff0000', label='1 = 0-15 years')
eight_patch = patch.Patch(color='#ff6600', label='8 = 75 and Over years'
)
three_patch = patch.Patch(color='#ffff00', label='3 = 25 to 34 years')

plt.legend(handles=[one_patch, two_patch, three_patch, four_patch, five_
patch, six_patch, seven_patch, eight_patch], loc = 'lower right', bbox_t
o_anchor=(0.5,1.05))
plt.show()
```



The forth plot is a pie chart for the distribution of the sample by the economic activity.

In [33]:
```
colors = ['#009933', '#00cc66', '#009999', '#336699', '#003399', '#0099f
f', '#00ffcc', '#3333cc', '#9999ff', '#99ccff']
refinedData['Economic Activity'].value_counts().plot(kind="pie", colors
= colors)

#the legend for the plot
five_patch = patch.Patch(color='#009999', label='5 = Retired')
minusnine_patch = patch.Patch(color='#00cc66', label='-9 = Aged under 16
or Student Living away from Home')
six_patch = patch.Patch(color='#003399', label='6 = Student')
three_patch = patch.Patch(color='#0099ff', label='3 = Unemployed')
eight_patch = patch.Patch(color='#00ffcc', label='8 = Long-term sick or
Disabled')
one_patch = patch.Patch(color='#009933', label='1 =  Employee')
seven_patch = patch.Patch(color='#3333cc', label='7 = Looking after home
or family')
four_patch = patch.Patch(color='#9999ff', label='4 = Full-time Student')
nine_patch = patch.Patch(color='#99ccff', label='9 = Other')
two_patch = patch.Patch(color='#336699', label = '2 = Self-employed')

plt.legend(handles=[one_patch, two_patch, three_patch, four_patch, five_
patch, six_patch, seven_patch, eight_patch, nine_patch, minusnine_patch]
, loc = 'lower right', bbox_to_anchor=(0.5,1.05))
plt.show()
```





The first easy additional requirement, is to produce two tables, using 'groupby' objects. The first table to be produced is the number of record by region and industry.

In [34]:
```
byRegionAndIndustry = refinedData[['Region', 'Industry']].copy()
byRegionAndIndustry = byRegionAndIndustry.groupby(['Region', 'Industry']
).size()
#this puts the data into a nicely formatted table and adds the Label Cou
nt
byRegionAndIndustry.reset_index(name='Count')
```

Out[34]:

	Region	Industry	Count
0	E12000001	-9	6854
1	E12000001	1	132
2	E12000001	2	2851
3	E12000001	3	1574
4	E12000001	4	3087
5	E12000001	5	1300
6	E12000001	6	1438
7	E12000001	7	524
8	E12000001	8	1883
9	E12000001	9	1498
10	E12000001	10	1836
11	E12000001	11	2524
12	E12000001	12	848
13	E12000002	-9	18755
14	E12000002	1	357
15	E12000002	2	7726
16	E12000002	3	3778
17	E12000002	4	9016
18	E12000002	5	3355
19	E12000002	6	3981
20	E12000002	7	1597
21	E12000002	8	5822
22	E12000002	9	3096
23	E12000002	10	4890
24	E12000002	11	6764
25	E12000002	12	2299
26	E12000003	-9	14089
27	E12000003	1	362
28	E12000003	2	5956
29	E12000003	3	3028
...	...	...	...
100	E12000008	9	4145
101	E12000008	10	6593
102	E12000008	11	7344
103	E12000008	12	3316
104	E12000009	-9	12401
105	E12000009	1	697
106	E12000009	2	5012
107	E12000009	3	3033

The second table to produce is the number of records by occupation and social grade.

In [35]:
```
byOccupationAndSocialGrade = refinedData[['Occupation', 'Approximated So
cial Grade']].copy()
byOccupationAndSocialGrade = byOccupationAndSocialGrade.groupby(['Occupa
tion', 'Approximated Social Grade']).size()
#this puts the data into a nicely formatted table and adds the Label Cou
nt
byOccupationAndSocialGrade.reset_index(name='Count')
```

Out[35]:

	Occupation	Approximated Social Grade	Count
0	-9	-9	116915
1	-9	1	1051
2	-9	2	17787
3	-9	3	2062
4	-9	4	12169
5	1	-9	492
6	1	1	19190
7	1	2	18555
8	1	3	584
9	1	4	967
10	2	-9	884
11	2	1	48104
12	2	2	13223
13	2	3	891
14	2	4	1009
15	3	-9	819
16	3	1	7050
17	3	2	35435
18	3	3	647
19	3	4	986
20	4	-9	727
21	4	1	3000
22	4	2	44922
23	4	3	2353
24	4	4	2252
25	5	-9	678
26	5	1	585
27	5	2	2464
28	5	3	37190
29	5	4	7629
30	6	-9	478
31	6	1	1061
32	6	2	6343
33	6	3	15555
34	6	4	13860
35	7	-9	1031
36	7	1	964
37	7	2	12184
38	7	3	2997

The second easy additional requirement is to learn how to use pandas to perform various queries. The first being to find the number of economically active people by region. As shown by the table below, the number of economically active poeple per region are specified.

In [36]:
```
data = refinedData[(refinedData['Economic Activity'] < 5) & (refinedData
['Economic Activity']>-1)]

byRegionAndEconomicActivity = data[['Region', 'Economic Activity']].copy
()
byRegionAndEconomicActivity = byRegionAndEconomicActivity.groupby(['Regi
on']).size()
#this puts the data into a nicely formatted table and adds the Label Cou
nt
byRegionAndEconomicActivity.reset_index(name='Count')
```

Out[36]:

	Region	Count
0	E12000001	12897
1	E12000002	35204
2	E12000003	26843
3	E12000004	23106
4	E12000005	27930
5	E12000006	30568
6	E12000007	44454
7	E12000008	45551
8	E12000009	27453
9	W92000004	14877

The second to find the number of economically active people by age., As shown by the table below the number of economically active people are shown sorted by age.

In [37]:
```
data = refinedData[(refinedData['Economic Activity'] < 5) & (refinedData
['Economic Activity']>-1)]

byRegionAndEconomicActivity = data[['Age', 'Economic Activity']].copy()
byRegionAndEconomicActivity = byRegionAndEconomicActivity.groupby(['Age'
]).size()
#this puts the data into a nicely formatted table and adds the Label Cou
nt
byRegionAndEconomicActivity.reset_index(name='Count')
```

Out[37]:

	Age	Count
0	2	41663
1	3	64326
2	4	67050
3	5	65736
4	6	40584
5	7	8022
6	8	1502

The thrid to find wherther or not there are any discrepancies between the student status given by the question "Student" and answers on the question "Economic activity". As shown below the number of discrepancies for poeple saying differing answers to the questions is counted.

In [38]:
```
#getting the separate data for the answers to the relevant questions
data = refinedData[((refinedData['Student'] == 2)) & (
                    ((refinedData['Economic Activity'] == 4 ) |
                    (refinedData['Economic Activity'] == 6 )))]
print("The number of people who said 'no' to being a student, but answer
ed that they were one of the student options \nin the economically activ
e question:")
print("\t",len(data))


data = refinedData[((refinedData['Student'] == 1) & (
                    ((refinedData['Economic Activity'] != 4 ) &
                    (refinedData['Economic Activity'] != 6 ) &(refinedDat
a['Economic Activity'] != -9))))]
print("The number of people who said 'yes' to being a student, but answe
red that they weren't one of the student options \nin the economically a
ctive question:")
print("\t",len(data))
```

```
The number of people who said 'no' to being a student, but answered that
they were one of the student options
in the economically active question:
         918
The number of people who said 'yes' to being a student, but answered that
they weren't one of the student options
in the economically active question:
         0
```

Finally, the forth is to find the number of working hours per week for students. As shown below the hours worked by students is counted and printed out in a table.

In [39]:
```
data = refinedData[(refinedData['Economic Activity'] == 4) | (refinedDat
a['Economic Activity'] == 6)]

byRegionAndEconomicActivity = data[['Hours worked per week', 'Economic A
ctivity']].copy()
byRegionAndEconomicActivity = byRegionAndEconomicActivity.groupby(['Hour
s worked per week']).size()
#this puts the data into a nicely formatted table and adds the Label Cou
nt
byRegionAndEconomicActivity.reset_index(name='Count')
```

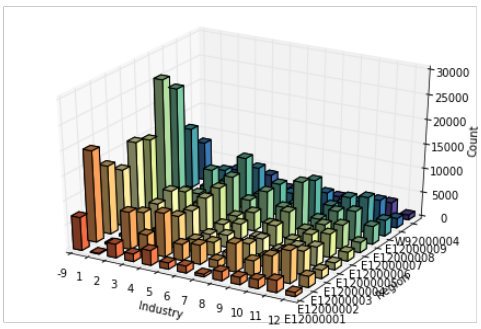
Out[39]:

	Hours worked per week	Count
0	-9	28110
1	1	6465
2	2	2334
3	3	1683
4	4	281

The first medium additional requirement is to create 3D plots for the data from the tables made in the first east additional requirement. The first being the 3D plot for the number of records by region and industry. Instead of changing the values on the plot to be the values of the variables, a legend was used to keep things clear as the plot axis can quickly become hard to see and understand.

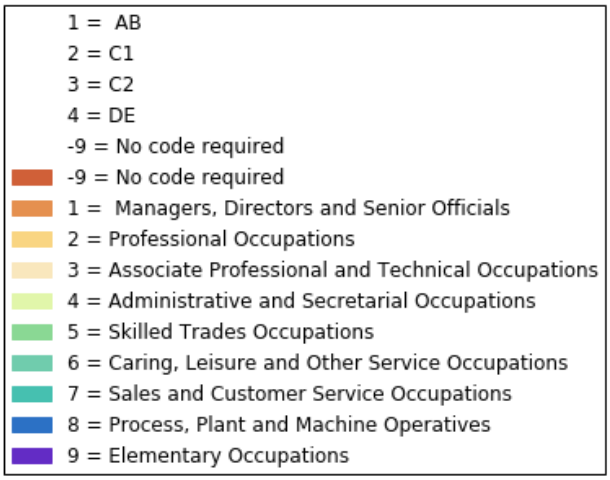
In [40]:
```
#arrays which will be used as the labels for the plot
regionarray = ["E12000001","E12000002","E12000003","E12000004","E1200000
5","E12000006","E12000007","E12000008","E12000009","W92000004"]
industryarray = [-9,1,2,3,4,5,6,7,8,9,10,11,12]

fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')
ax = Axes3D(fig)
ax.set_xlabel("Industry")
ax.set_ylabel("Region")
ax.set_zlabel("Count")

#getting teh specified data
byRegionAndIndustry = refinedData[['Region', 'Industry']].copy()
byRegionAndIndustry = byRegionAndIndustry.groupby(['Region', 'Industry']
)

#turning the count into the z axis
z = byRegionAndIndustry.size().tolist()
#getting all combinations of (x,y) for region and industry
axes = byRegionAndIndustry.groups.keys()
#sorting them so they are in the correct order for the corresponding cou
nts
axes = sorted(axes, key=itemgetter(1))
axes = sorted(axes, key=itemgetter(0))
#setting the number of rows and columns to be the range for all possible
values
x = list(range(0, len(regionarray)))
y = list(range(0, len(industryarray)))
ax.set_yticks(x)
ax.set_xticks(y)
#getting the separate values out of the tuples to be plotted
X, Y = np.meshgrid(y,x)
zs = np.array(z)
Z = zs.reshape(Y.shape)

values = np.linspace(0.2,1.,X.ravel().shape[0])
colours = plt.cm.Spectral(values)
#actually plotting the bar plot
ax.bar3d(X.ravel(), Y.ravel(), Z.ravel()*0, dx=0.5, dy=0.5, dz=Z.ravel()
, color=colours)
ax.set_xticklabels(np.array(industryarray))
ax.set_yticklabels(np.array(regionarray))

#the legend for the plot
five_patch = patch.Patch(color='white', label='5 = Accommodation and foo
d service activities')
minusnine_patch = patch.Patch(color='white', label='-9 = No code require
d')
six_patch = patch.Patch(color='white', label='6 = Transport and storage;
Information and communication')
three_patch = patch.Patch(color='white', label='3 = Construction')
eight_patch = patch.Patch(color='white', label='8 = Real estate activiti
es, Professional, scientific and technical activities, Administrative an
d support service activities')
one_patch = patch.Patch(color='white', label='1 =  Agriculture, forestry
and fishing')
seven_patch = patch.Patch(color='white', label='7 = Financial and insura
nce activities, Intermediation')
four_patch = patch.Patch(color='white', label='4 = Wholesale and retail
trade, Repair of motor vehicles and motorcycles')
nine_patch = patch.Patch(color='white', label='9 = Public administration
and defence, compulsory social security')
two_patch = patch.Patch(color='white', label = '2 = Mining and quarrying
, Manufacturing,  Electricity, gas, steam and air conditioning system, W
ater supply')
ten_patch = patch.Patch(color='white', label='10 = Education')
eleven_patch = patch.Patch(color='white', label='11 = Human health and s
```

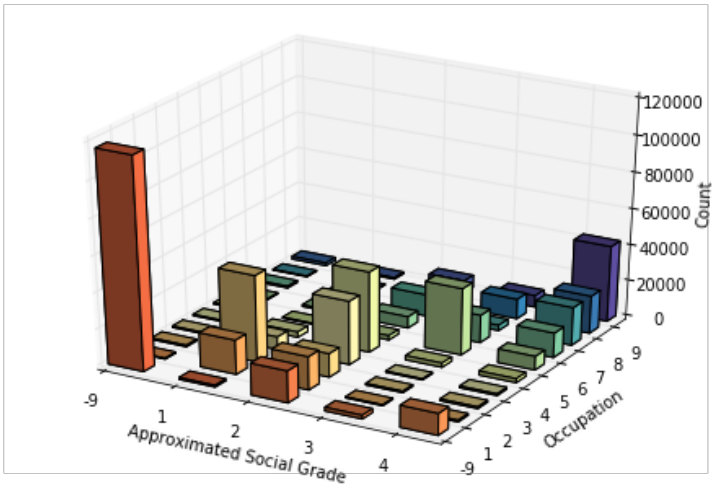
1 = Agriculture, forestry and fishing
2 = Mining and quarrying, Manufacturing, Electricity, gas, steam and air conditioning system, Water supply
3 = Construction
4 = Wholesale and retail trade, Repair of motor vehicles and motorcycles
5 = Accommodation and food service activities
6 = Transport and storage; Information and communication
7 = Financial and insurance activities, Intermediation
8 = Real estate activities, Professional, scientific and technical activities, Administrative and support service activities
9 = Public administration and defence, compulsory social security
10 = Education
11 = Human health and social work activities
12 = Other community, social and personal service activities, Private households employing domestic staff, Extra-territorial organisations and bodies
-9 = No code required
E12000001 = North East
E12000002 = North West
E12000003 = Yorkshire and the Humber
E12000004 = East Midlands
E12000005 = West Midlands
E12000006 = East of England
E12000007 = London
E12000008 = South East
E12000009 = South West
W92000004 = Wales



The second being the 3D plot for the number of records by occupation and social grade. Instead of changing the values on the plot to be the values of the variables, a legend was used to keep things clear as the plot axis can quickly become hard to see and understand.

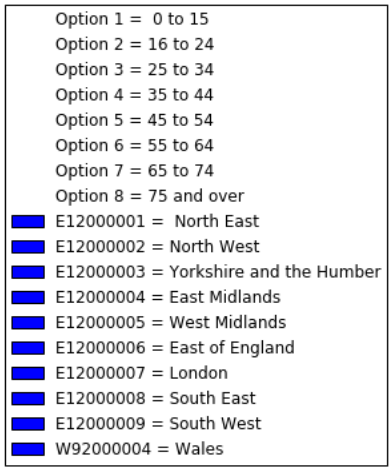
In [41]:
```
#the values which will be used as labels
occupationarray = [-9,1,2,3,4,5,6,7,8,9]
socialarray = [-9,1,2,3,4]

fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')
ax = Axes3D(fig)
ax.set_xlabel("Approximated Social Grade")
ax.set_ylabel("Occupation")
ax.set_zlabel("Count")
#separating out the relevant data
byOccupationAndSocialGrade = refinedData[['Occupation', 'Approximated So
cial Grade']].copy()
byOccupationAndSocialGrade = byOccupationAndSocialGrade.groupby(['Occupa
tion', 'Approximated Social Grade'])
#setting the counts to the z axis
z = byOccupationAndSocialGrade.size().tolist()
axes = byOccupationAndSocialGrade.groups.keys()
#sorting the (x,y) coordinates
axes = sorted(axes, key=itemgetter(1))
axes = sorted(axes, key=itemgetter(0))
#setting the number of rows and columns to be the range for all possible
values
x = list(range(0, len(occupationarray)))
y = list(range(0, len(socialarray)))
ax.set_yticks(x)
ax.set_xticks(y)
#turns x,y into 2D array
X, Y = np.meshgrid(y,x)
zs = np.array(z)
Z = zs.reshape(X.shape)
#setting the colors for the blocks, 'x coords', to be different for each
one
values = np.linspace(0.2,1.,X.ravel().shape[0])
colours = plt.cm.Spectral(values)
#ravel conversts it back into 1D array
ax.bar3d(X.ravel(), Y.ravel(), Z.ravel()*0, dx=0.5, dy=0.5, dz=Z.ravel()
, color=colours)
ax.set_xticklabels(np.array(socialarray))
ax.set_yticklabels(np.array(occupationarray))

#the legend for the plot
minusnine_patch = patch.Patch(color='white', label='-9 = No code require
d')
three_patch = patch.Patch(color='white', label='3 = C2')
one_patch = patch.Patch(color='white', label='1 =  AB')
four_patch = patch.Patch(color='white', label='4 = DE')
two_patch = patch.Patch(color='white', label = '2 = C1')

fivex_patch = patch.Patch(color='#8AD894', label='5 = Skilled Trades Occ
upations')
sixx_patch = patch.Patch(color='#70CCAD', label='6 = Caring, Leisure and
Other Service Occupations')
threex_patch = patch.Patch(color='#F9E7BD', label='3 = Associate Profess
ional and Technical Occupations')
eightx_patch = patch.Patch(color='#2C71C5', label='8 = Process, Plant an
d Machine Operatives')
onex_patch = patch.Patch(color='#E5904F', label='1 =  Managers, Director
s and Senior Officials')
sevenx_patch = patch.Patch(color='#46C0B0', label='7 = Sales and Custome
r Service Occupations')
fourx_patch = patch.Patch(color='#E1F6AA', label='4 = Administrative and
Secretarial Occupations')
ninex_patch = patch.Patch(color='#632CC5', label='9 = Elementary Occupat
ions')
twox_patch = patch.Patch(color='#F9D582', label = '2 = Professional Occu
pations')
tenx_patch = patch.Patch(color='#D06038', label='-9 = No code required')
```

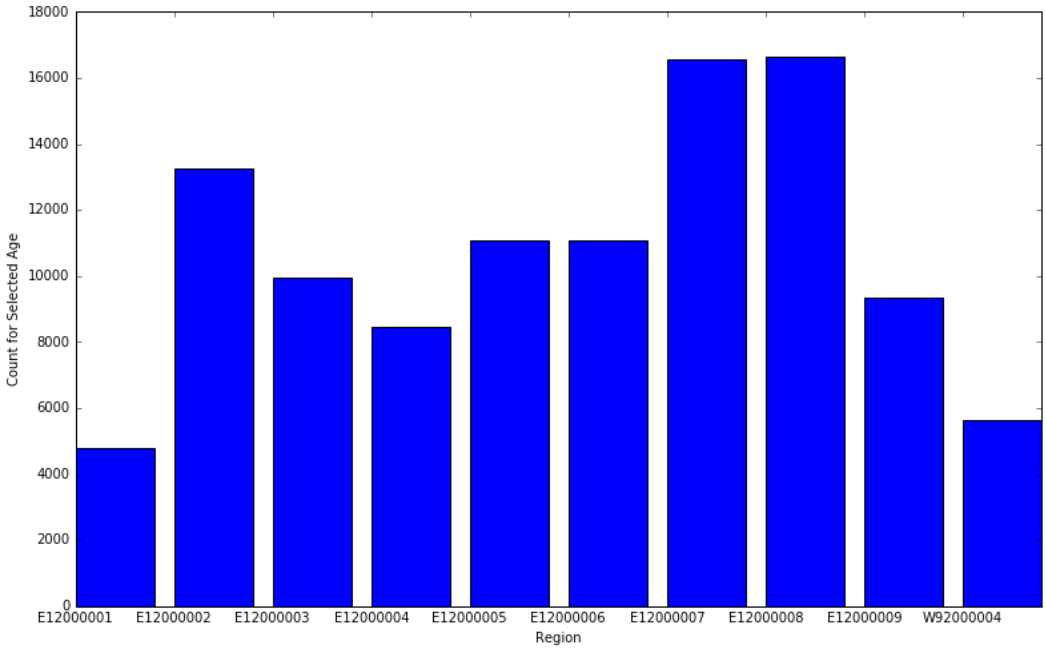




The second medium additional requirement is to use 'ipywidgets' to control plot properties. The first plot is plotting Region by the occurances of Age.

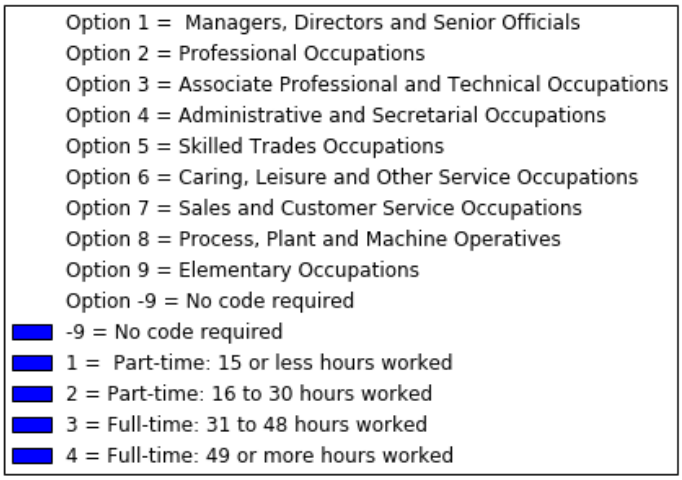
In [42]:
```
#the function to allow nultiple plots to be made
def update(Age = range(1,9)):
    fig = plt.figure(figsize=(13,8))
    ax = fig.add_subplot(1, 1, 1)
    #getting the data where the data is equal to the value selected by t
he user
    data = refinedData[(((refinedData['Age'] == Age )))]
    data = data.groupby(['Region', 'Age'])
    count = data.size().tolist()
    ax.set_xticks(list(range(1, len(regionarray)+1)))
    ax.set_xticklabels(np.array(regionarray))
    ax.set_xlabel("Region")
    ax.set_ylabel("Count for Selected Age")
    #sets the x axis to the correct number of 'bar's
    ax.bar([1,2,3,4,5,6,7,8,9,10], count)
    #the legend for the plot
    fivex_patch = patch.Patch(label='E12000005 = West Midlands')
    sixx_patch = patch.Patch(label='E12000006 = East of England')
    threex_patch = patch.Patch(label='E12000003 = Yorkshire and the Humb
er')
    eightx_patch = patch.Patch(label='E12000008 = South East')
    onex_patch = patch.Patch(label='E12000001 =  North East')
    sevenx_patch = patch.Patch(label='E12000007 = London')
    fourx_patch = patch.Patch(label='E12000004 = East Midlands')
    ninex_patch = patch.Patch(label='E12000009 = South West')
    twox_patch = patch.Patch(label = 'E12000002 = North West')
    tenx_patch = patch.Patch(label='W92000004 = Wales')

    five_patch = patch.Patch(color = 'white', label='Option 5 = 45 to 54
')
    six_patch = patch.Patch(color = 'white', label='Option 6 = 55 to 64'
)
    three_patch = patch.Patch(color = 'white', label='Option 3 = 25 to 3
4')
    eight_patch = patch.Patch(color = 'white', label='Option 8 = 75 and
over')
    one_patch = patch.Patch(color = 'white', label='Option 1 =  0 to 15'
)
    seven_patch = patch.Patch(color = 'white', label='Option 7 = 65 to 7
4')
    four_patch = patch.Patch(color = 'white', label='Option 4 = 35 to 44
')
    two_patch = patch.Patch(color = 'white', label = 'Option 2 = 16 to 2
4')

    plt.legend(handles=[one_patch, two_patch, three_patch, four_patch, f
ive_patch, six_patch, seven_patch, eight_patch, onex_patch, twox_patch,
threex_patch, fourx_patch, fivex_patch, sixx_patch, sevenx_patch, eightx
_patch, ninex_patch, tenx_patch], bbox_to_anchor=(0.5,2))

    fig.canvas.draw()

interact(update);
```

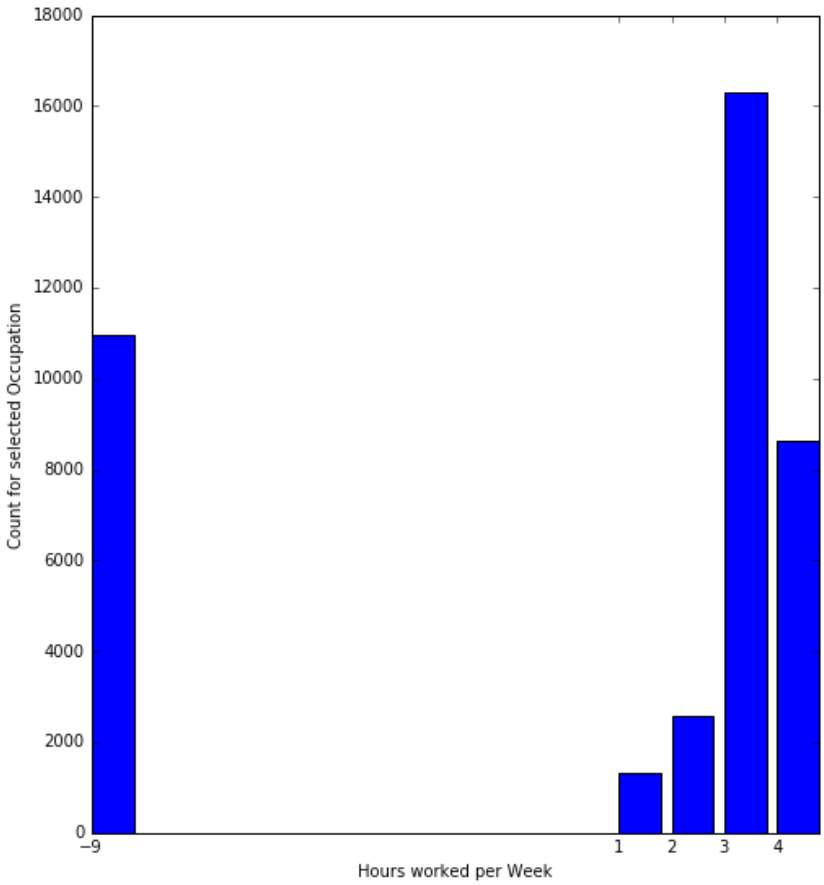




The next plot is Hours worked per week by Occupation.

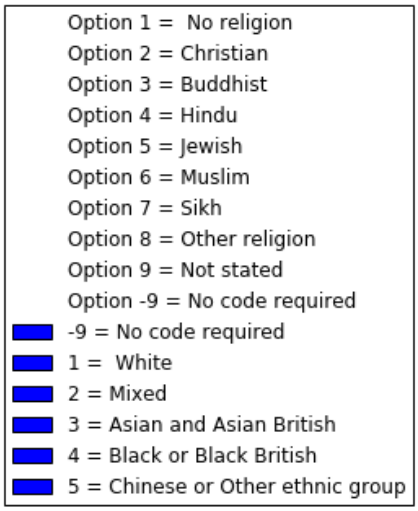
In [43]:
```
def update(Occupation = [1,2,3,4,5,6,7,8,9,-9]):
    fig = plt.figure(figsize=(8,9))
    ax = fig.add_subplot(1, 1, 1)
    #getting the relvant data where it is equal to the value the user sp
ecified
    data = refinedData[(((refinedData['Occupation'] == Occupation)))]
    data = data.groupby(['Occupation', 'Hours worked per week'])
    count = data.size().tolist()
    ax.set_xticks(list(range(1, len(hoursWorkPerWeek)+1))+[-9])
    ax.set_xlabel("Hours worked per Week")
    ax.set_ylabel("Count for selected Occupation")
     #the legend for the plot
    minusnine_patch = patch.Patch( label='-9 = No code required')
    three_patch = patch.Patch(label='3 = Full-time: 31 to 48 hours worke
d')
    one_patch = patch.Patch(label='1 =  Part-time: 15 or less hours work
ed')
    four_patch = patch.Patch( label='4 = Full-time: 49 or more hours wor
ked')
    two_patch = patch.Patch(label = '2 = Part-time: 16 to 30 hours worke
d')

    fivex_patch = patch.Patch(color='white', label='Option 5 = Skilled T
rades Occupations')
    sixx_patch = patch.Patch(color='white', label='Option 6 = Caring, Le
isure and Other Service Occupations')
    threex_patch = patch.Patch(color='white', label='Option 3 = Associat
e Professional and Technical Occupations')
    eightx_patch = patch.Patch(color='white', label='Option 8 = Process,
Plant and Machine Operatives')
    onex_patch = patch.Patch(color='white', label='Option 1 =  Managers,
Directors and Senior Officials')
    sevenx_patch = patch.Patch(color='white', label='Option 7 = Sales an
d Customer Service Occupations')
    fourx_patch = patch.Patch(color='white', label='Option 4 = Administr
ative and Secretarial Occupations')
    ninex_patch = patch.Patch(color='white', label='Option 9 = Elementar
y Occupations')
    twox_patch = patch.Patch(color='white', label = 'Option 2 = Professi
onal Occupations')
    tenx_patch = patch.Patch(color='white', label='Option -9 = No code r
equired')

    plt.legend(handles=[onex_patch,twox_patch, threex_patch,fourx_patch,
fivex_patch,sixx_patch,sevenx_patch,eightx_patch,ninex_patch, tenx_patch
, minusnine_patch, one_patch, two_patch, three_patch, four_patch], bbox_
to_anchor=(1,1.8))

    #making sure there is the right number of 'bar's in the plot as ther
e is in the data
    if len(count) == 1:
        ax.bar([-9,1,2,3,4], count+[0,0,0,0])
    else:
        ax.bar([-9,1,2,3,4], count)
    fig.canvas.draw()

interact(update);
```

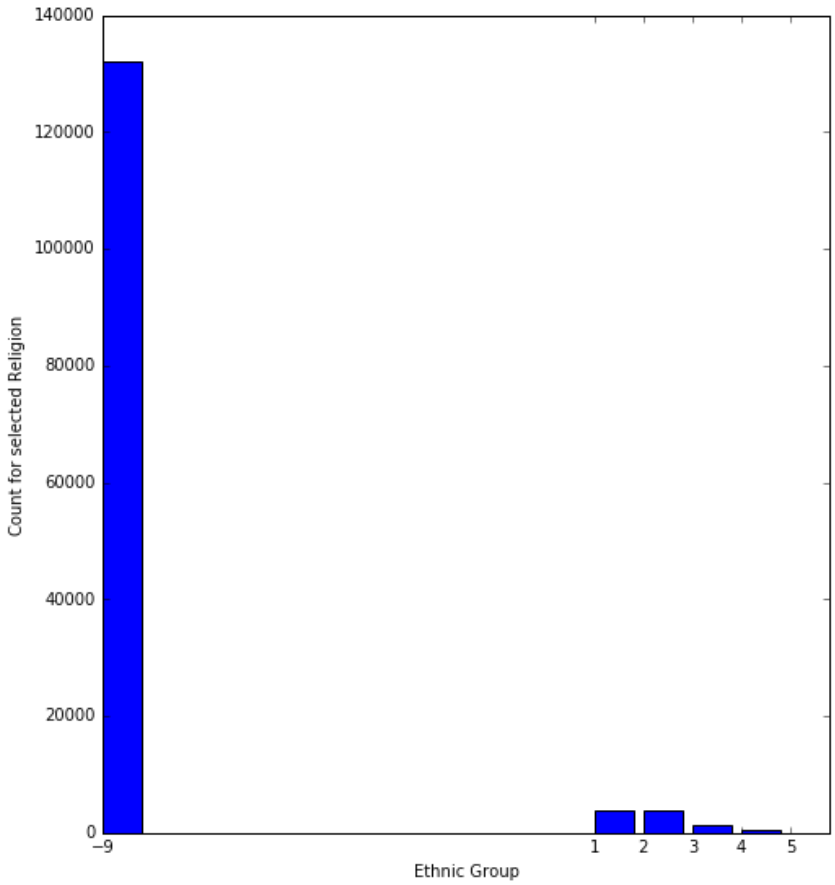




The next plot is Ethnic group by Religion.

In [44]:
```
 def update(Religion = [1,2,3,4,5,6,7,8,9,-9]):
    fig = plt.figure(figsize=(8,9))
    ax = fig.add_subplot(1, 1, 1)
    #getting the relvant data where it is equal to the value the user sp
ecified
    data = refinedData[(((refinedData['Religion'] == Religion)))]
    data = data.groupby(['Ethnic Group', 'Religion'])
    count = data.size().tolist()
    ax.set_xticks(list(range(1, len(ethnicGroup)+1))+[-9])
    ax.set_xlabel("Ethnic Group")
    ax.set_ylabel("Count for selected Religion")
    #making sure there is the right number of 'bar's in the plot as ther
e is in the data
    #the legend for the plot
    minusnine_patch = patch.Patch( label='-9 = No code required')
    three_patch = patch.Patch(label='3 = Asian and Asian British')
    one_patch = patch.Patch(label='1 =  White')
    four_patch = patch.Patch( label='4 = Black or Black British')
    two_patch = patch.Patch(label = '2 = Mixed')
    five_patch = patch.Patch(label = '5 = Chinese or Other ethnic group'
)

    fivex_patch = patch.Patch(color='white', label='Option 5 = Jewish')
    sixx_patch = patch.Patch(color='white', label='Option 6 = Muslim')
    threex_patch = patch.Patch(color='white', label='Option 3 = Buddhist
')
    eightx_patch = patch.Patch(color='white', label='Option 8 = Other re
ligion')
    onex_patch = patch.Patch(color='white', label='Option 1 =  No religi
on')
    sevenx_patch = patch.Patch(color='white', label='Option 7 = Sikh')
    fourx_patch = patch.Patch(color='white', label='Option 4 = Hindu')
    ninex_patch = patch.Patch(color='white', label='Option 9 = Not state
d')
    twox_patch = patch.Patch(color='white', label = 'Option 2 = Christia
n')
    tenx_patch = patch.Patch(color='white', label='Option -9 = No code r
equired')
    plt.legend(handles=[onex_patch,twox_patch, threex_patch,fourx_patch,
fivex_patch,sixx_patch,sevenx_patch,eightx_patch,ninex_patch, tenx_patch
, minusnine_patch, one_patch, two_patch, three_patch, four_patch, five_p
atch], bbox_to_anchor=(0.5,1.8))
    if len(count) == 1:
        ax.bar([-9,1,2,3,4,5], count+[0,0,0,0,0])
    else:
        ax.bar([-9,1,2,3,4,5], count+[0])
    fig.canvas.draw()

interact(update);
```





The final plot is Health by Marital Status.

In [45]:
```
def update(Status = widgets.IntSlider(min=1, max=5, step=1, value=1)):
    fig = plt.figure(figsize=(9,15))
    ax = fig.add_subplot(1, 1, 1)
    #getting the relvant data where it is equal to the value the user sp
ecified
    data = refinedData[(((refinedData['Marital Status'] == Status)))]
    data = data.groupby(['Marital Status', 'Health'])
    count = data.size().tolist()
    ax.set_xticks(list(range(1, len(health)+1))+[-9])
    ax.set_xlabel("Health")
    ax.set_ylabel("Count for selected Marital Status")
    #making sure there is the right number of 'bar's in the plot as ther
e is in the data
    #the legend for the plot
    minusnine_patch = patch.Patch( label='-9 = No code required')
    three_patch = patch.Patch(label='3 = Fair health')
    one_patch = patch.Patch(label='1 =  Very good health')
    four_patch = patch.Patch( label='4 = Bad health')
    two_patch = patch.Patch(label = '2 = Good health')
    five_patch = patch.Patch(label = '5 = Very bad health')
    threey_patch = patch.Patch(color = 'white', label='Option 3 = Separa
ted but still legally married or separated but still legally in a same-s
ex civil partnership')
    oney_patch = patch.Patch(color = 'white', label='Option 1 =  Single
(never married or never registered a same-sex civil partnership)')
    foury_patch = patch.Patch( color = 'white', label='Option 4 = Divorc
ed or formerly in a same-sex civil partnership which is now legally diss
olved')
    twoy_patch = patch.Patch(color = 'white', label = 'Option 2 = Marrie
d or in a registered same-sex civil partnership')
    fivey_patch = patch.Patch(color = 'white', label = 'Option 5 = Widow
ed or surviving partner from a same-sex civil partnership')

    plt.legend(handles=[minusnine_patch, one_patch, two_patch, three_pat
ch, four_patch, five_patch, oney_patch, twoy_patch, threey_patch, foury_
patch, fivey_patch], bbox_to_anchor=(1,1.3))
    if len(count) == 1:
        ax.bar([-9,1,2,3,4,5], count+[0,0,0,0,0])
    else:
        ax.bar([-9,1,2,3,4,5], count)
    fig.canvas.draw()

interact(update);
```

-9 = No code required
1 = Very good health
2 = Good health
3 = Fair health
4 = Bad health
5 = Very bad health
Option 1 = Single (never married or never registered a same-sex civil partnership)
Option 2 = Married or in a registered same-sex civil partnership
Option 3 = Separated but still legally married or separated but still legally in a same-sex civil partnership
Option 4 = Divorced or formerly in a same-sex civil partnership which is now legally dissolved
Option 5 = Widowed or surviving partner from a same-sex civil partnership