VISUAL ANALYTICS : BUILDING COMPUTER MODELS (LAB 4)

In [1]:

```
# Importing Libraries
import os

import numpy as np
import pandas as pd
pd.options.mode.chained_assignment = None
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

from scipy.stats import spearmanr, pearsonr
from scipy.spatial.distance import cdist

import statsmodels.api as sm

import geopandas as gp

np.random.seed(123)

# SETTING THE WORKING DIRECTORY : Change the path to where your data is
os.chdir('/Users/Samyer/Documents/Studies/City University of London/Modules/INM433 Visual
Analytics/Labs/Lab 4/Data/')
```

In [2]:

```
# Loading the data
wards=gp.read_file('london_wards_2011_wgs84.shp')
df = pd.read_csv('population_perc.csv')
```

In [3]:

```
# Joining dataframes (population with wards)
df_merged = df.merge(wards, left_on = 'id', right_on = 'CODE')
df.head()
```

Out[3]:

	id	Name	Borough	N of all usual residents	Area Hectares	Density (number of persons per hectare)	Mean Age	Median Age	Average distance to work (km)	age=0 to 4: Population % by age	...	distance to work=Less than 2km: Population % by distance travelled to work	distance to work=2km to less than 5km: Population % by distance travelled to work	P
0	E05000001	Aldersgate	NaN	1465.0	12.98	112.9	45.5	45.0	6.0	3.4	...	30.9	10.4	
1	E05000002	Aldgate	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
2	E05000003	Bassishaw	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
3	E05000004	Billingsgate	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
4	E05000005	Bishopsgate	NaN	222.0	81.53	2.7	34.8	31.0	4.0	0.9	...	36.5	8.6	

5 rows × 147 columns

In [4]:

```
# Dropping rows with null values in the Borough Column
df_merged = df_merged[df_merged.Borough.notnull()]
df_merged.describe()
```

Out[4]:

	N of all usual residents	Area Hectares	Density (number of persons per hectare)	Mean Age	Median Age	Average distance to work (km)	age=0 to 4: Population % by age	age=5 to 7: Population % by age	age=8 to 9: Population % by age	age=10 to 14: Population % by age	...
count	624.000000	624.000000	624.000000	624.000000	624.000000	624.000000	624.000000	624.000000	624.000000	624.000000	...
mean	13087.445513	251.482853	81.072596	35.809455	34.257212	11.316987	7.155449	3.651282	2.207051	5.557853	...
std	2430.502303	255.672602	47.597964	3.086301	4.019309	2.104608	1.471681	0.770669	0.500191	1.319420	...
min	5110.000000	35.360000	1.800000	29.000000	26.000000	6.400000	2.700000	1.300000	0.700000	1.400000	...
25%	11200.000000	119.900000	45.300000	33.575000	31.000000	9.900000	6.100000	3.200000	1.900000	4.700000	...
50%	12989.500000	184.565000	69.050000	35.400000	33.000000	11.400000	7.100000	3.600000	2.200000	5.750000	...
75%	14864.750000	284.425000	109.550000	37.900000	37.000000	12.800000	8.000000	4.100000	2.500000	6.500000	...
max	23084.000000	2903.520000	264.700000	44.100000	46.000000	18.100000	13.200000	6.300000	4.100000	9.400000	...

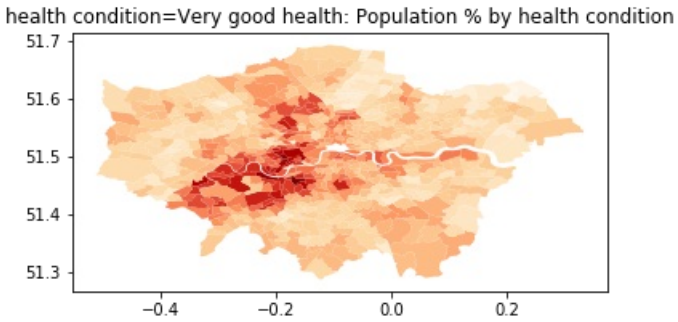
8 rows × 144 columns

In [5]:

```
# Transform df_merged into gepandas dataframe
df_merged = gp.GeoDataFrame(df_merged)
```

In [6]:

```
# Chloropeth Map
df_merged.crs = {'init' :'epsg:4326'}
attr="health condition=Very good health: Population % by health condition";
ax=df_merged.plot(column=attr,cmap='OrRd');
ax.set_title(attr);
plt.show();
```



⊙ Picking attributes to analyze.

In [7]:

```
# Health and Education attributes (extreme ones)
bad_health = 'health condition=Very bad health: Population % by health condition'
good_health = 'health condition=Very good health: Population % by health condition'
bad_edu = 'qualification (study)=No qualifications: Population % by qualification or study'
good_edu = 'qualification (study)=Level 4 qualifications and above: Population % by qualification
or study'
```

⦿ Explore visually the interrelations between the attributes referring to the qualification and health condition; choose an attribute pair for building a formal model (regression) describing their interrelationship.

In [8]:

```
# Let's create a sub-dataframe with  those 4 attributes (and some others for later use)
df_ex = df_merged[[good_health, bad_health, good_edu, bad_edu, 'Borough', 'Mean Age', 'Average dist
ance to work (km)',
                   'age=25 to 29: Population % by age','age=30 to 44: Population % by age',
                   'age=45 to 59: Population % by age','age=60 to 64: Population % by age', 'geomet
ry']]

# Let's rename the columns of our dataframe (their current names are too long, they will clutter o
ur plots if we keep them)
df_ex.columns = ['Good Health %', 'Bad Health %', 'Good Education %', 'Bad Education %', 'Borough'
,
                 'Mean Age', 'Avg Dist Work', 'Age 25_29 %', 'Age 30_44 %', 'Age 45_59 %', 'Age 60_
64 %', 'geometry']
df_ex.head()
```

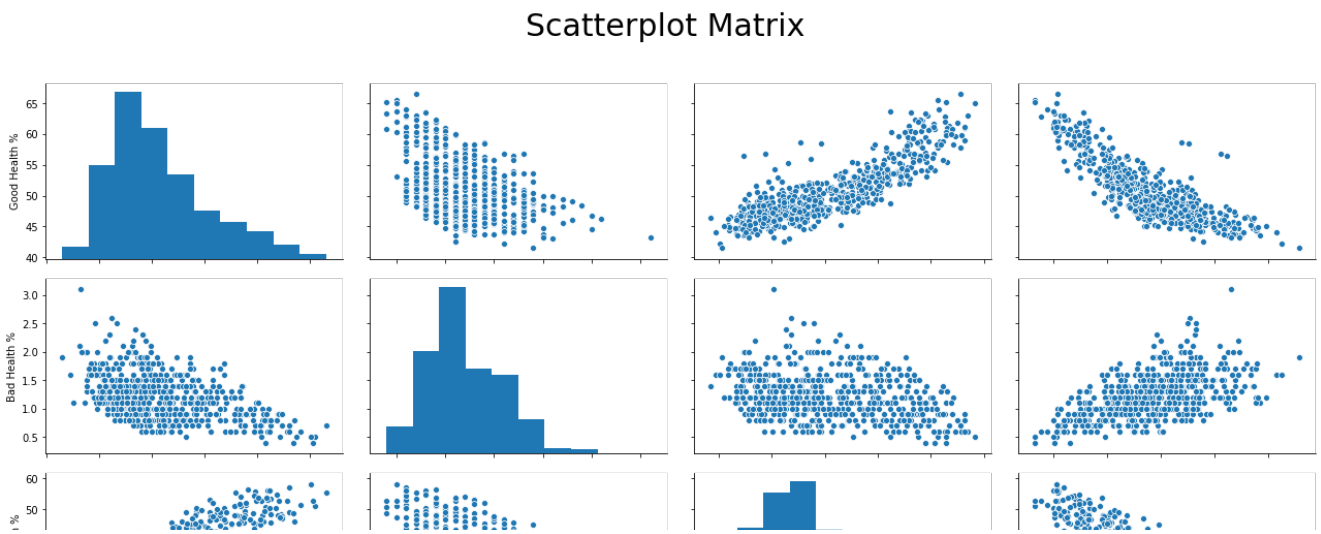
Out[8]:

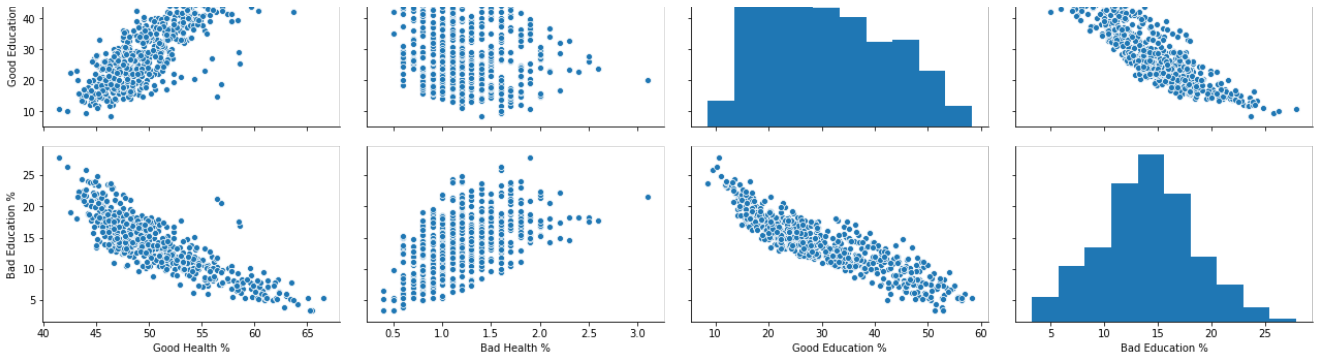
	Good Health %	Bad Health %	Good Education %	Bad Education %	Borough	Mean Age	Avg Dist Work	Age 25_29 %	Age 30_44 %	Age 45_59 %	Age 60_64 %	geometry
25	51.3	0.9	26.1	12.4	BarkingDagenham	29.4	14.1	14.7	27.6	11.6	2.5	(POLYGON ((0.0691258349339708 51.5388110598182...
26	46.6	1.8	12.3	23.0	BarkingDagenham	33.5	13.4	7.3	23.2	17.2	3.3	POLYGON ((0.1563506941179533 51.55101882115206...
27	46.9	1.5	15.4	20.9	BarkingDagenham	33.1	14.9	7.9	24.0	17.0	3.3	POLYGON ((0.1271285033616449 51.55560537620193...
28	43.8	1.5	14.7	22.0	BarkingDagenham	36.6	14.2	7.3	20.7	15.9	4.4	POLYGON ((0.1481797474653408 51.59895997155979...
29	46.1	1.4	14.6	23.6	BarkingDagenham	37.5	13.8	6.6	19.5	19.4	4.9	POLYGON ((0.1851212458724257 51.56478686939491...

In [9]:

```
# Let's display a scatterplot matrix
plt.figure(num=1)
ax = sns.pairplot(df_ex[['Good Health %', 'Bad Health %', 'Good Education %', 'Bad Education %']],
                  height = 3, aspect=1.5)
plt.suptitle('Scatterplot Matrix', size = 30)
plt.subplots_adjust(top=.9); # This line is to adjust the space between the title and the matrix o
f plots
```

```
<Figure size 432x288 with 0 Axes>
```





There is an increasing monotonic relationship between *Good Health %* and *Good Education %*. It seems to be linear as well, we'll see how a first degree polynomial fits. Let's pick these two as our attribute pair and try out different models !

⦿ Build model variants for different values of the parameter (order) and assess visually which variant gives a better fit.

Before actually plotting the regressions, let's assess the correlation between our variables (using Pearson's coefficient)

In [10]:

```
# Computing the correlation coefficient (we have to drop the rows with missing values before compu
ting the correlation)

print("Pearson's correlation coeff. between 'Good Health %' and 'Good Education %' :",
      pearsonr(df_ex[['Good Health %', 'Good Education %']].dropna()['Good Health %'],
               df_ex[['Good Health %', 'Good Education %']].dropna()['Good Education %']
              )[0]
     )
```

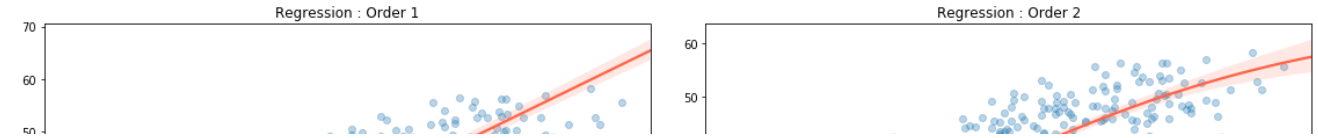
```
Pearson's correlation coeff. between 'Good Health %' and 'Good Education %' : 0.8494094368278114
```

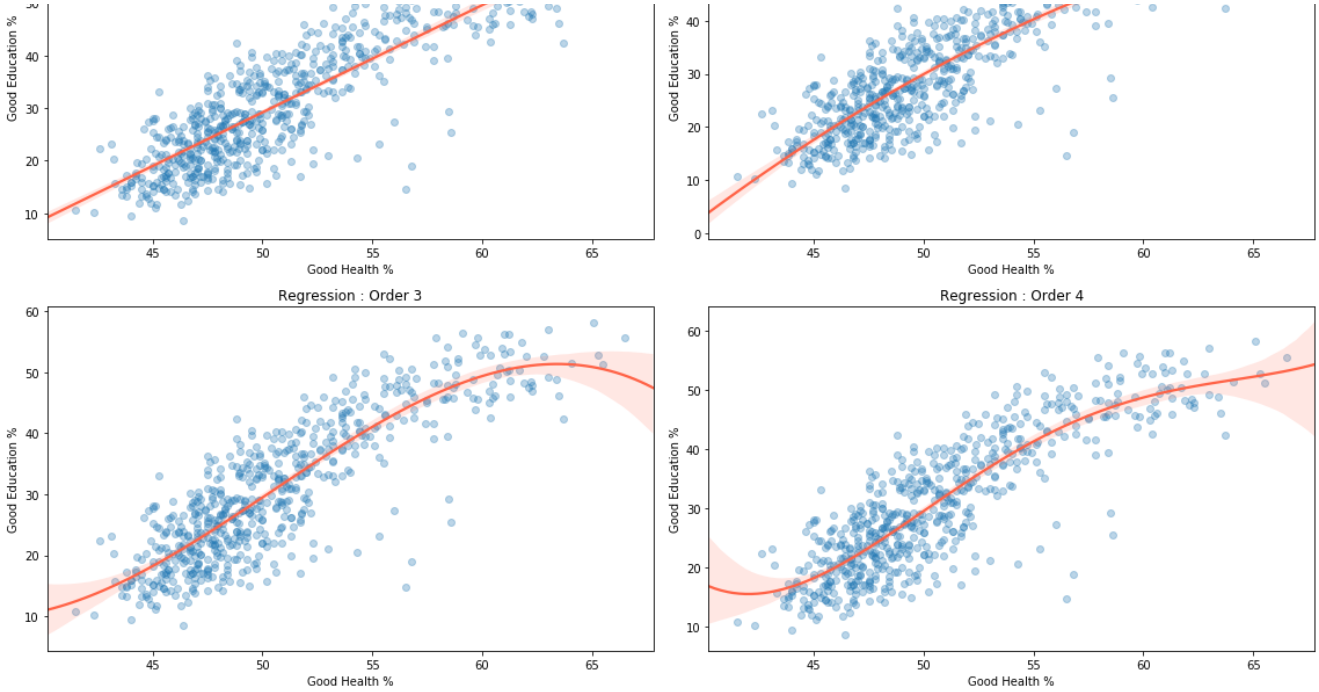
That's definitely a pretty high value. There is a strong linear relationship between both attributes. Now let's fit different order polynomials to our data and see what happens.

In [11]:

```
# Fitting first to fourth order polynomials
plt.figure(num=2, figsize = (16, 10))
plt.subplot(2,2,1)
ax1 = sns.regplot(df_ex['Good Health %'], df_ex['Good Education %'], order=1, scatter_kws={'alpha':
0.3}, line_kws={'color':'tomato'})
ax1.set(title = 'Regression : Order 1')
plt.subplot(2,2,2)
ax2 = sns.regplot(df_ex['Good Health %'], df_ex['Good Education %'], order=2, scatter_kws={'alpha':
0.3}, line_kws={'color':'tomato'})
ax2.set(title = 'Regression : Order 2')
plt.subplot(2,2,3)
ax3 = sns.regplot(df_ex['Good Health %'], df_ex['Good Education %'], order=3, scatter_kws={'alpha':
0.3}, line_kws={'color':'tomato'})
ax3.set(title = 'Regression : Order 3')
plt.subplot(2,2,4)
ax4 = sns.regplot(df_ex['Good Health %'], df_ex['Good Education %'], order=4, scatter_kws={'alpha':
0.3}, line_kws={'color':'tomato'})
ax4.set(title = 'Regression : Order 4')
plt.tight_layout();
```

```
/Users/Samyer/Documents/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tu
ple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



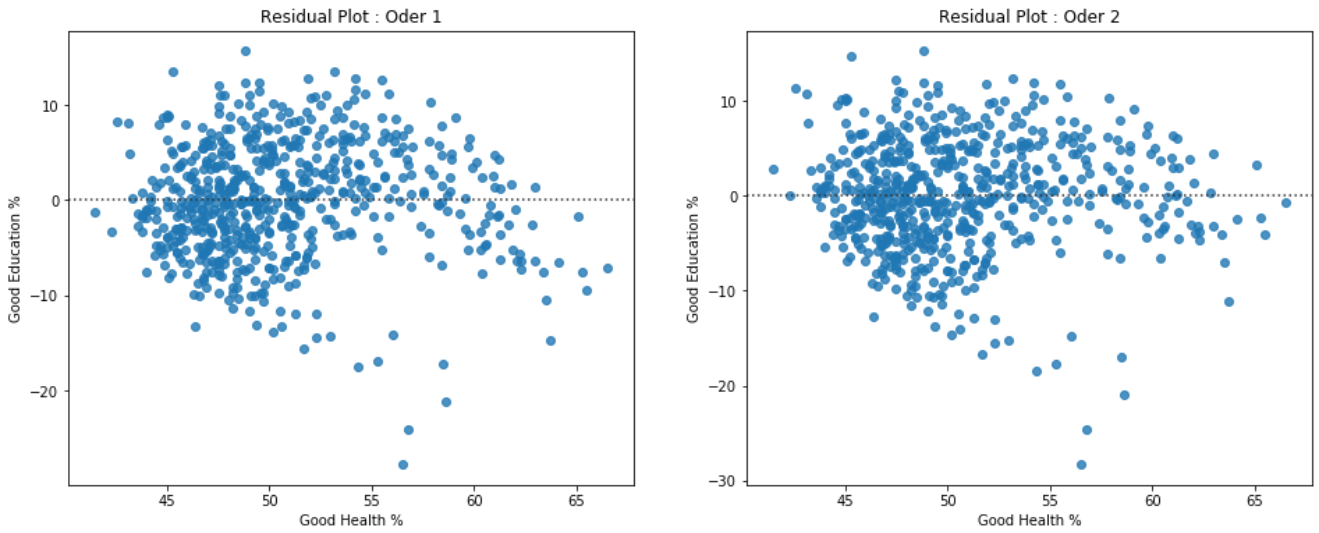


Visually speaking, order 2 polynomial seems to fit the data the best. However, its confidence around the higher values of the input variable is lower (bigger light red area), mainly because there aren't enough data points in that particular region. Let's check out the distribution of residuals for the first two orders.

⊙ Consider the distributions of the residuals for the model variants. Choose the variant with the best (i.e., the most random) distribution of the residuals. If two or more models are equivalent in this respect, choose the simpler one. How does your choice correspond to the previous visual assessment of the goodness of the model fit?

In [12]:

```
# Plotting residual plots
plt.figure(num= 3, figsize=(16, 6))
plt.subplot(1,2,1)
ax1 = sns.residplot(x = 'Good Health %', y = 'Good Education %', data = df_ex, order = 1)
ax1.set(title='Residual Plot : Oder 1')
plt.subplot(1,2,2)
ax2 = sns.residplot(x = 'Good Health %', y = 'Good Education %', data = df_ex, order = 2)
ax2.set(title='Residual Plot : Oder 2');
```



As we can see, the distribution of residuals looks somewhat similar for order 1 and order 2 polynomial regressions, although order 2 residuals seem more random, especially for higher values of x. We can say that both models are good fits for the data. So how do we choose which one to keep in this case ? Well one solution is to follow **Occam's Razor** which states : *"when presented with competing hypothetical answers to a problem, one should select the one that makes the fewest assumptions"*.

In machine learning, this translates to selecting the least complex model out of two or more equally performing ones (all other things being equal), which in this case means, the order 1 polynomial.

Visually speaking, there seems to be a curve in the scatterplot, which is why I picked order 2 at first. However, I'm opting for the simpler model for now. The 0.86 correlation coefficient is also a good evidence to pick order 1 since pearson's correlation measures *linear* relationships.

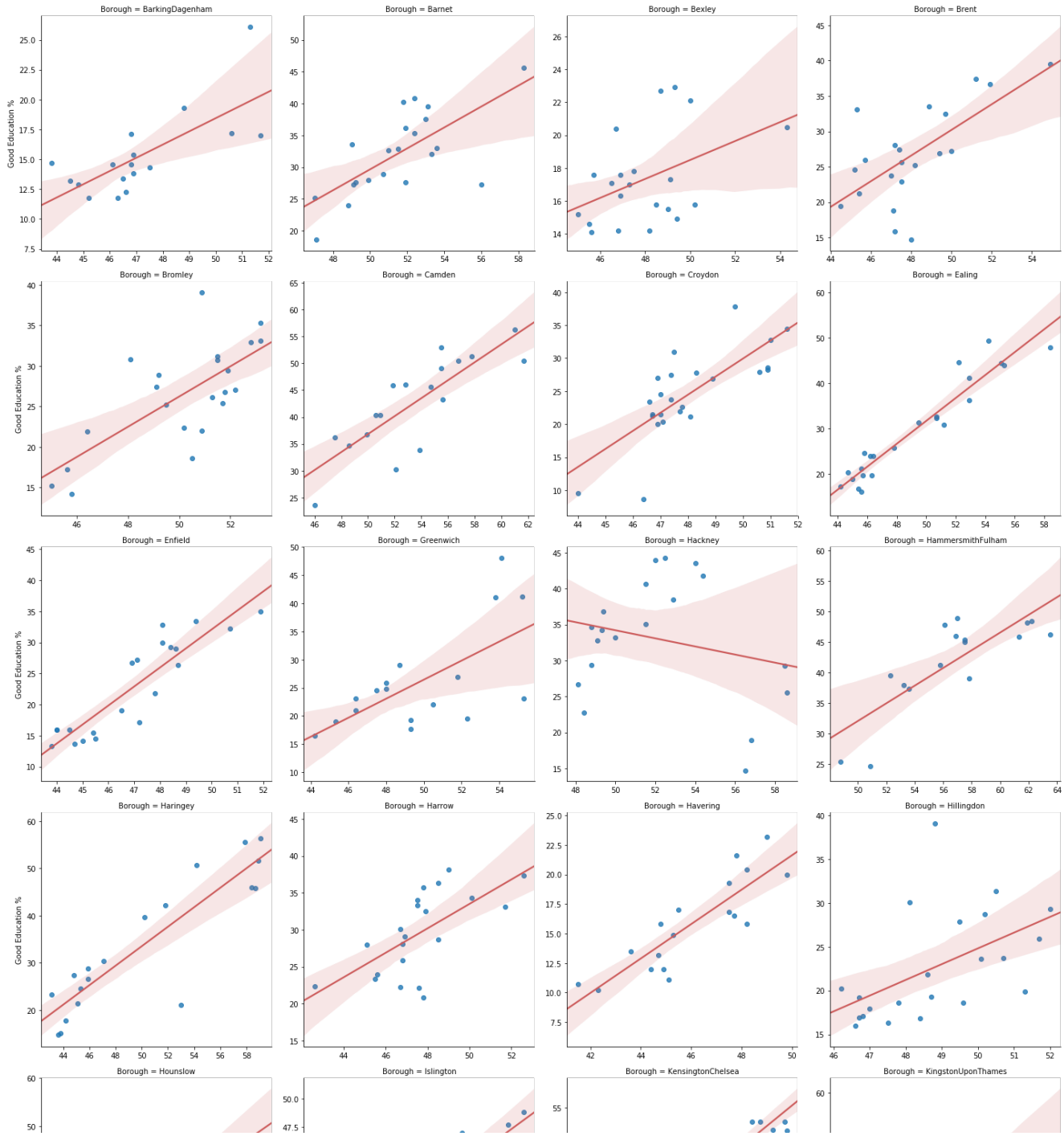
⦿ Check whether the interrelation varies over the territory. Build models for the data subsets based on the attribute 'Borough' and look for existence of significant differences.

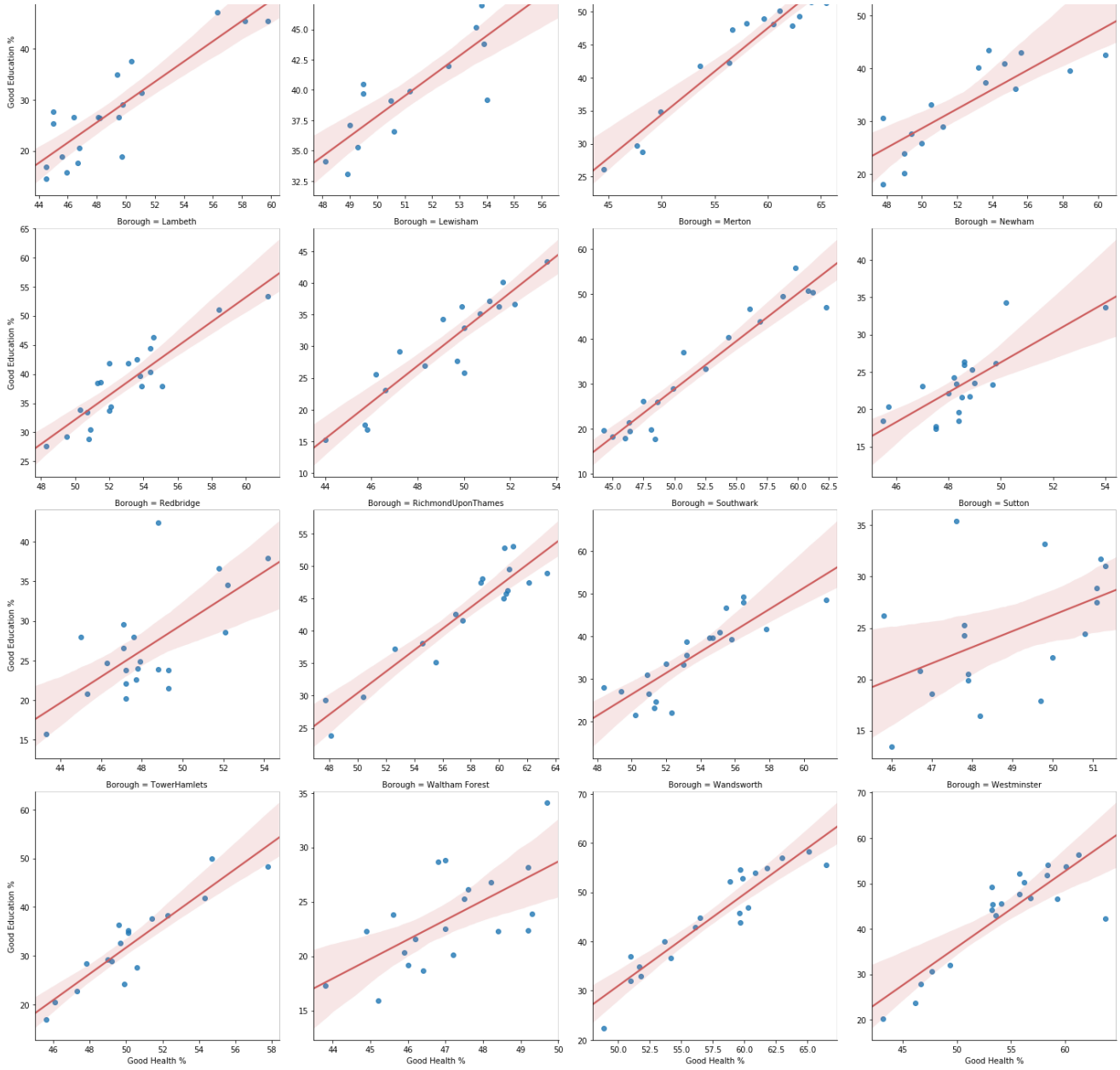
In [13]:

```
# Let's build a 1st order polynomial for each Borough
sns.lmplot(x = 'Good Health %', y = 'Good Education %', data = df_ex, order = 1,
           col = 'Borough', col_wrap = 4, sharex = False, sharey = False, line_kws={'color':'indianr
ed'})
# Setting sharex and sharey to False allows each plot to display its own range for the x and y axi
s. Otherwise the scale looks too small
```

Out[13]:

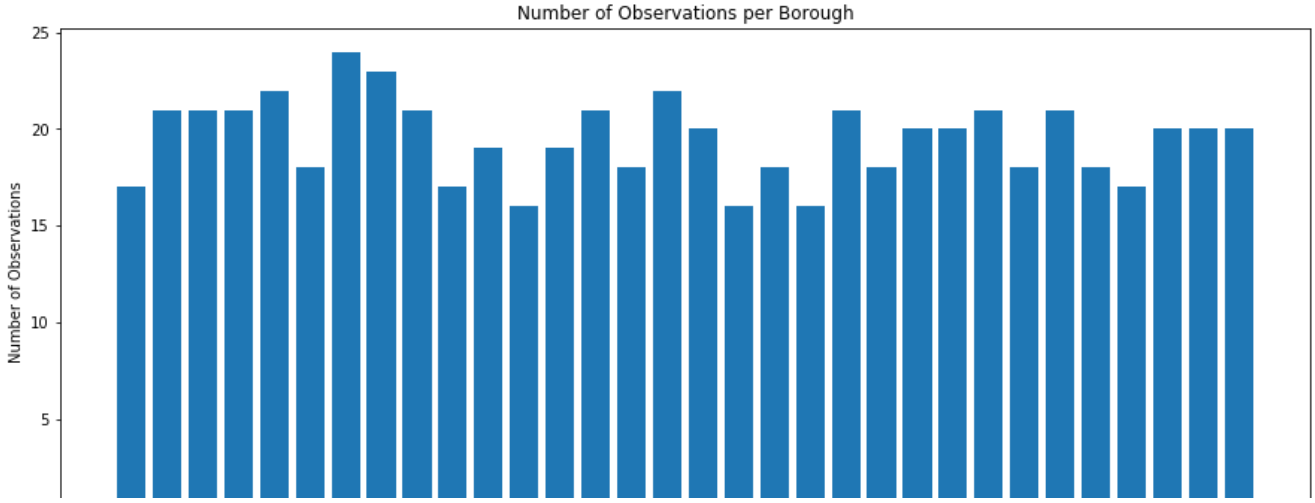
```
<seaborn.axisgrid.FacetGrid at 0x1c215d5a90>
```

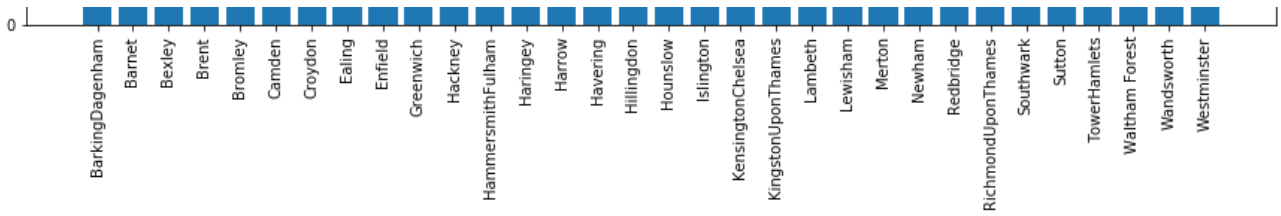




In [14]:

```
# Notice how some plots don't have many points ? Now let's show the number of Data Points per Borough
plt.figure(num=4, figsize=(15,6))
plt.bar(df_ex.groupby('Borough').size().index, df_ex.groupby('Borough').size().values)
plt.xticks(rotation=90)
plt.ylabel('Number of Observations')
plt.title('Number of Observations per Borough');
```





Some territories like Ealing, Enfield and Merton display linear relationships in accordance with a first order polynomial. However many others, such as Greenwich, Hackney and Hillingdon, show drastic differences. Can we use the variable Borough to build separate models for different territories ? Probably not, because the number of levels (categories or unique values) in this variable is too high, and more importantly, the levels do not have enough data points to build a proper model.

We need to find a better variable whose levels (or categories) show different relationships between our two attributes (good health and good education).

⊙ Check the possibilities to refine the model using some attributes as conditioning variables. Try attributes “Mean age” and “Average distance to work”. Which attribute has a better potential for refinement (i.e., the partial models differ more significantly)?

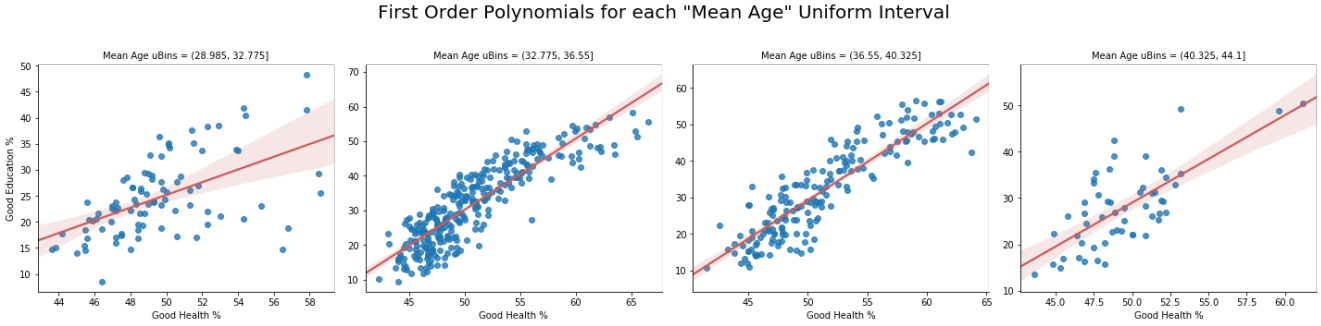
Basically we're trying to find a categorical attribute with a few levels (categories) for which each category represents a different relationship between our x (good health) and y (good education). Alright, but what if we want to condition over a continuous variable ? Well in this case we can transform the continous variable into a categorical one using a technique called *binning*, which basically means that we transform the range of that variable into intervals, and each data point (row), would be labeled by its interval number. This way, we move from a continous setting to a categorical one. Let's do that with the "Mean age" and "Average distance to work" continous variables.

In [15]:

```
# Binning the Mean Age attribute
df_ex['Mean Age uBins'] = pd.cut(df_ex['Mean Age'], bins = 4) # uBins for Uniform Bins : intervals
of the same size

# Visualizing 'Good Health %' and 'Good Education %' per bin of 'Mean Age uBins'
plt.figure(num=5)
ax = sns.lmplot(x = 'Good Health %', y = 'Good Education %', data = df_ex, order = 1,
                col = 'Mean Age uBins', col_wrap = 4, sharex = False, sharey = False, line_kws={'col
or':'indianred'})
plt.suptitle('First Order Polynomials for each "Mean Age" Uniform Interval', size = 20)
plt.subplots_adjust(top=.8);
```

```
/Users/Samyer/Documents/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tu
ple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

```
<Figure size 432x288 with 0 Axes>
```



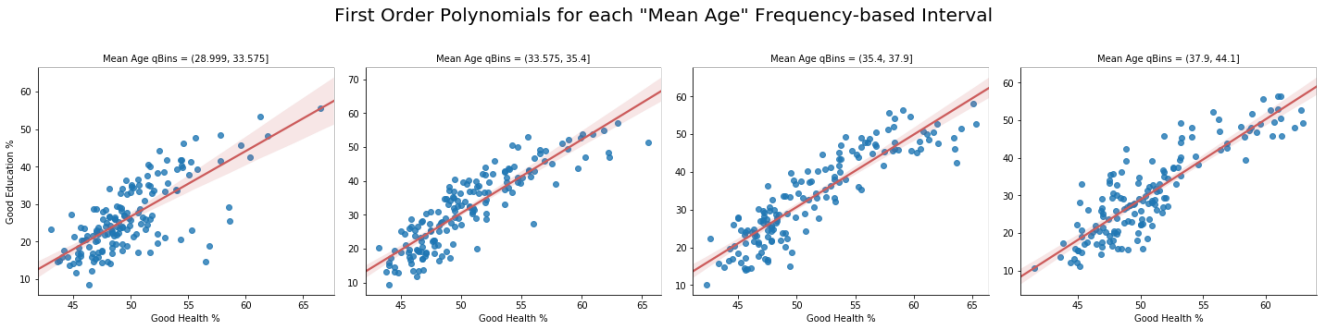
Note that the relationships seem to be similar across the 4 plots. However, the first and last plots don't have enough data points. In order to overcome this issue, we can bin the Mean Age variable differently : instead of splitting the range of the variable into equal length intervals, we can split it into intervals that have the same number of data points (so, not necessarily equal length). Let's try it !

In [16]:

```
# Binning the Mean Age attribute
df_ex['Mean Age qBins'] = pd.qcut(df_ex['Mean Age'], q = 4) # qBins for quantile-based Bins

# Visualizing 'Good Health %' and 'Good Education %' per bin of 'Mean Age qBins'
plt.figure(num=6)
ax = sns.lmplot(x = 'Good Health %', y = 'Good Education %', data = df_ex, order = 1,
                col = 'Mean Age qBins', col_wrap = 4, sharex = False, sharey = False, line_kws={'col
or':'indianred'})
plt.suptitle('First Order Polynomials for each "Mean Age" Frequency-based Interval', size = 20)
plt.subplots_adjust(top=.8);
```

```
/Users/Samyer/Documents/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tu
ple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

```
<Figure size 432x288 with 0 Axes>
```



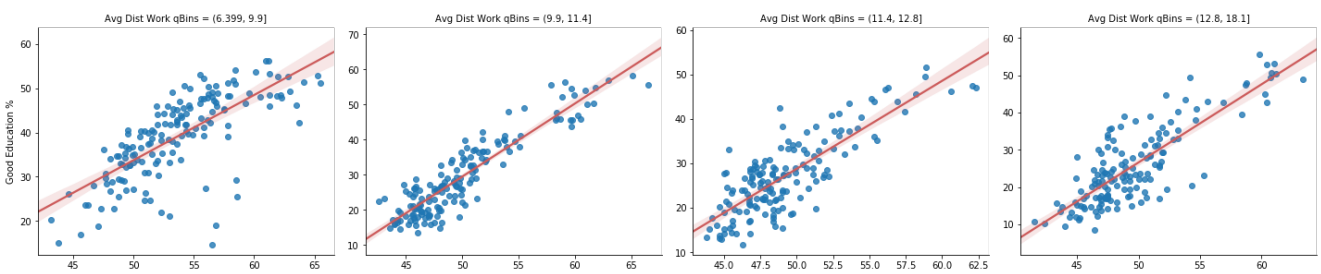
Now for each interval we have roughly the same number of points. However, there doesn't seem to be a difference between the relationships in each plot. They're all linear, although the slopes of the lines slightly vary. Let's try doing the same thing using another variable.

In [17]:

```
# Binning the Mean Age attribute
df_ex['Avg Dist Work qBins'] = pd.qcut(df_ex['Avg Dist Work'], q = 4) # qBins for quantiled-based B
ins

# Visualizing 'Good Health %' and 'Good Education %' per bin of 'Avg Dist Work qBins'
plt.figure(num=7)
ax = sns.lmplot(x = 'Good Health %', y = 'Good Education %', data = df_ex, order = 1,
                col = 'Avg Dist Work qBins', col_wrap = 4, sharex = False, sharey = False, line_kws=
{'color':'indianred'})
plt.suptitle('First Order Polynomials for each "Avg Dist Work" Frequency-based Interval', size = 2
0)
plt.subplots_adjust(top=.8);
```

```
/Users/Samyer/Documents/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tu
ple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

```
<Figure size 432x288 with 0 Axes>
```



Good Health % Good Health % Good Health % Good Health %

Here, we note that the first plot doesn't seem to be linear, maybe a second order polynomial would fit it best. However the other three seem to show a linear relationship, although the slopes and intercepts are not equal.

Let us try a different method to condition our model.

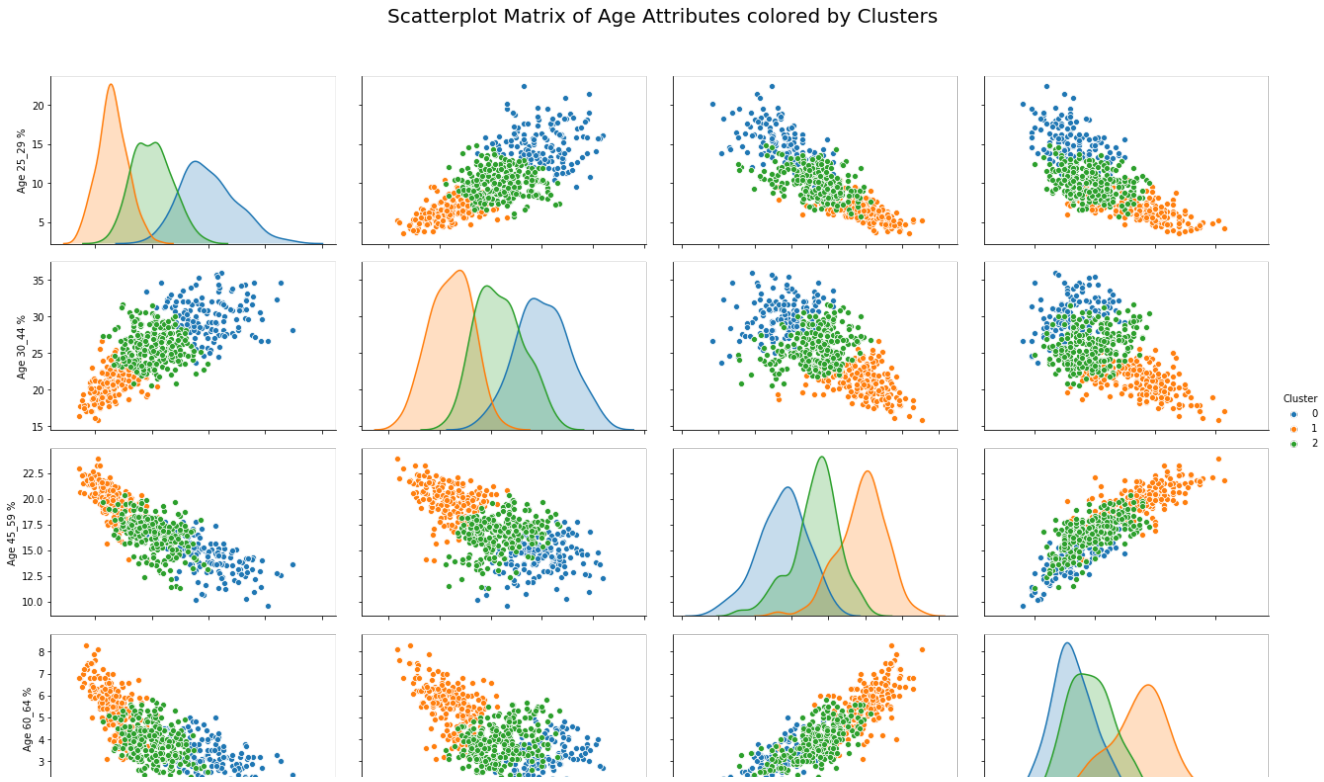
⦿ Check the possibility to refine the model by using clustering based on several attributes reflecting the age structure, in particular, proportions of the age groups from 25 to 64 years old, which seem the most relevant to the qualification level. Apply partition-based clustering and build a model for each cluster. Are there essential differences?

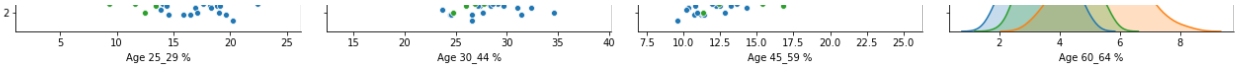
In [18]:

```
# We're going to use Kmeans to cluster our data, and model Good Education vs. Good Health for each
cluster
# First, let's define the variables we'll use in our K-means (the age groups)
km_features = ['Age 25_29 %', 'Age 30_44 %', 'Age 45_59 %', 'Age 60_64 %']

# Create the model
nb_clusters = 3 # you may change the number of clusters here to experiment
km = KMeans(n_clusters=nb_clusters, random_state=123)

# Fit the model to the data
km = km.fit(df_ex[km_features])

# Append a column with the cluster number to each data point (row)
df_ex['Cluster'] = km.predict(df_ex[km_features])

# Visualizing the variables used in the Clustering
plt.figure(num=8)
sns.pairplot(data=df_ex[km_features+['Cluster']], hue='Cluster', vars=km_features, aspect=1.5, heig
ht=3)
plt.suptitle('Scatterplot Matrix of Age Attributes colored by Clusters', size = 20)
plt.subplots_adjust(top=.9); # This line is to adjust the space between the title and the matrix o
f plots
```

```
/Users/Samyer/Documents/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tu
ple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

```
<Figure size 432x288 with 0 Axes>
```

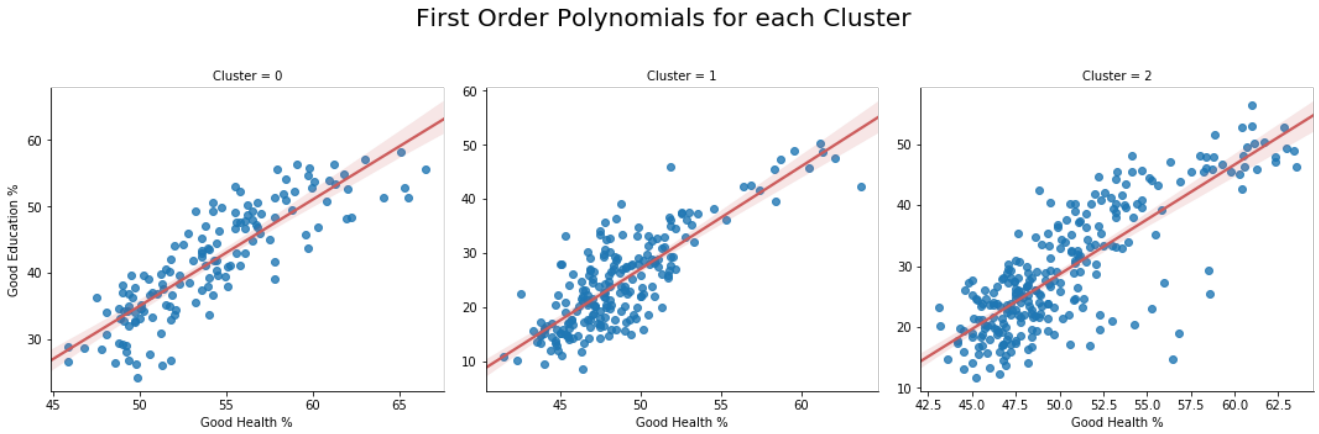




In [19]:

```
# Visualizing 'Good Health %' and 'Good Education %' per Cluster
plt.figure(num =9)
ax = sns.lmplot(x = 'Good Health %', y = 'Good Education %', data = df_ex, order = 1,
                col = 'Cluster', col_wrap = 3, sharex = False, sharey = False, line_kws={'color':'ir
dianred'})
plt.suptitle('First Order Polynomials for each Cluster', size = 20)
plt.subplots_adjust(top=.8);
```

```
<Figure size 432x288 with 0 Axes>
```



In [20]:

```
# Visualizing the residual plot of the model by a Clustering partition
g = sns.FacetGrid(df_ex, col="Cluster", sharex=False, sharey=False, aspect=1.5) # FacetGrid allows
you to display many plots based on the levels of a categorical variable !
g.map(sns.residplot, 'Good Health %', 'Good Education %'); # Fill the FacetGrid by residplots
```

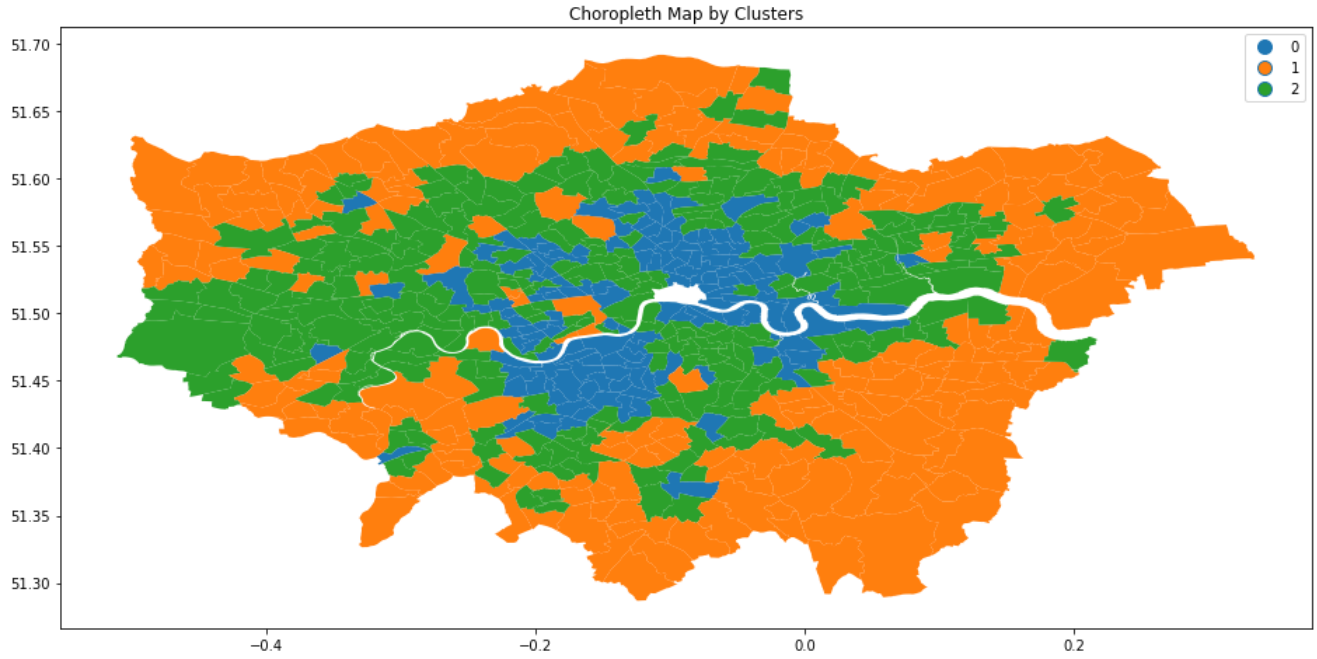
There are a few interesting things to take from this. First we need to describe our clusters in terms of the values of the clustering variables (age attributes used in k-means).

From the scatterplot matrix above, we notice that Cluster 0 (blue) includes mostly young working adults between 25 and 44 (look at the diagonal plots). Cluster 1 on the other hand, contains mostly older folks from 45 to 64. Lastly, Cluster 2 has a little bit of every age category except the oldest one above 60. Alright, now that we understand the distributions of our clusters, let's talk about the regression models.

From the regression and residual plots above, the best linear looking shape of the data is that of cluster 0. Naturally, it also has the most random looking residual plot with no apparent structure at all. This goes to say, that among the 3 clusters, cluster 0 is the one that characterizes the linear regression model of *Good Education %* on *Good Health %* the best. This shouldn't come as a surprise since for young adults (cluster 0), the confounding variable age doesn't impact Health or Education : the relationship shows that having a good health is the reflection of being an intellectual which itself is a reflection of having a good education. On the other hand, for old people, having poor health doesn't necessarily mean not having a good education, other variables are at play here, and illness due to age is one of them. This means that the linear relationship between Good Education and Good Health should be less apparent for cluster 1. And you can see it from the range of the x and y variables : the Good Health % is relatively low overall and has a restricted range (40-55%) while the Good Education % has a 'normal' longer range (10-50%).

In [21]:

```
# Transform the df_ex DataFrame into a GeoDataFrame
df_gp = gp.GeoDataFrame(df_ex)

# Create a color map (to keep the same colors as our clusters from earlier)
from matplotlib.colors import ListedColormap
cluster_cmap = ListedColormap(sns.color_palette().as_hex()[:nb_clusters])

# Visualize the Choropleth Map by Clusters
df_gp.crs = {'init':'epsg:4326'}
df_gp.plot(column='Cluster', categorical=True, cmap=cluster_cmap, legend=True, figsize=(16, 8))
plt.title('Choropleth Map by Clusters');
```



There is something worth noting here. With 3 clusters, we seem to see a grouping of cluster 0 and a bit of cluster 2 around central london. That is easily explained by the fact that central london is where most of the activity happens and Cluster 0 represents young working adults. Naturally, cluster 1 is spread along the borders, since old people, after retirement, prefer to migrate to the suburbs far from the noise and the action.

⦿ Based on the analysis done, choose a reasonable approach to modelling: to build a single model or several partial models; if several, how to divide the data; what order to use. Take into account model accuracy, complexity, and understandability.

Alright, everything we've done so far was leading up to this. How do we use everything we learned about our data in the modeling stage ?

Our goal is to build a final model based on a single model or many sub-models for each chosen partition of the data. In order to do so, we need to take a few things into account :

> - **Accuracy :** How do we measure the accuracy of our final model ? This is a regression problem, so we need a proper regression performance metric. Given the range of values taken by our output variable, RMSE (Root Mean Squared Error) seems a to be a good one. It is heavily impacted by outliers though, so we may want to keep that in mind.
> - **Complexity :** This refers to the number of parameters (degrees of freedom) in the model. It represents the ability of a model to capture complex dependencies. Given a choice between two equally good models, we'll always pick the simplest one.
> - **Understandability :** This term refers to the ability of the human analyst to understand how the output of the model will behave when the input values are slightly changed from a given position, or in other words, understand why the model outputs a certain prediction for some input sample. There will always be a trade-off between model complexity and understandability. In our case, linear regression is a class of interpretable models, but building several sub-models to combine, and having to keep track of all the partitions and their meanings can reduce understandability.

Since we have many other variables that we can use to model the relationship between *Good Health %* and *Good Education %*, I will

opt for a clustering-based approach to partition the data and build sub-models for each partition (if they show significant differences).

However, before doing that, let's build a simple model to use as a baseline (*i.e* a reference to compare it to the more sophisticated model). Alright, let's dive in !

In [22]:

```
# Splitting the data into a training set and a test set
df_train, df_test = train_test_split(df_merged, test_size = .25, random_state = 123) # Test set = 25% of all data
```

In [23]:

```
# Simple linear regression using statsmodels
lm_base = sm.OLS(df_train[good_edu], sm.add_constant(df_train[good_health])).fit()
pred_base = lm_base.predict(sm.add_constant(df_test[good_health]))

# Computing the value of Performance Metric (RMSE) for the Simple Linear Regression model
rmse_base = np.sqrt(mean_squared_error(df_test[good_edu], pred_base))
print('The RMSE of the base 1st Order model is :', rmse_base)
```

```
The RMSE of the base 1st Order model is : 5.807852604161108
```

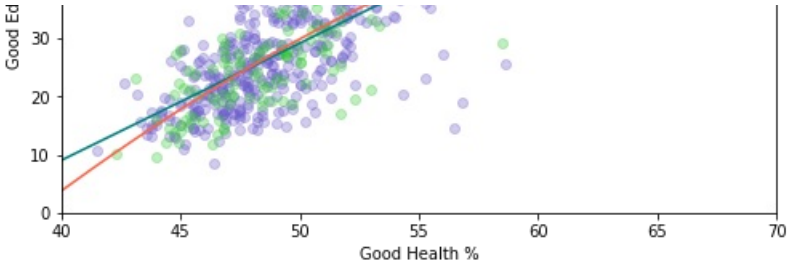
In [24]:

```
# Second order linear regression using statsmodels
lm_base2 = sm.OLS(df_train[good_edu], sm.add_constant(np.column_stack((df_train[good_health],
df_train[good_health]**2)))).fit() # sm.OLS doesn't support polynomial regression of higher
orders, so you have to manually create your higher order features
pred_base2 = lm_base2.predict(sm.add_constant(np.column_stack((df_test[good_health],
df_test[good_health]**2))))

# Computing the value of Performance Metric (RMSE) for the Second Order Polynomial model
rmse_base2 = np.sqrt(mean_squared_error(df_test[good_edu], pred_base2))
print('The RMSE of the base 2nd Order model is :', rmse_base2)
```

```
The RMSE of the base 2nd Order model is : 5.662960932063112
```

In [25]:

```
# Let's visualize the results
x = np.linspace(0, 100, 1000) # The x-axis vector to use for the line plots
plt.figure(num=10, figsize=(8,5))

# Scatter Plots (Train + Test data)
plt.scatter(df_merged.loc[df_train.index, good_health], df_merged.loc[df_train.index, good_edu],
            c='slateblue', alpha = .3, label='Train Data')
plt.scatter(df_merged.loc[df_test.index, good_health], df_merged.loc[df_test.index, good_edu],
            c='limegreen', alpha = .3, label='Test Data')

# Line Plots (1st Order + 2nd Order)
plt.plot(x, lm_base.predict(sm.add_constant(x)), c='teal', label='1st Order')
plt.plot(x, lm_base2.predict(sm.add_constant(np.column_stack((x, x**2)))), c='tomato', label='2nd Order')

plt.xlim(40,70)
plt.ylim(0, 75)
plt.xlabel('Good Health %')
plt.ylabel('Good Education %')
plt.title('Fitting First and Second Order Polynomials')
plt.legend();
```



Alright ! The simple linear regression had a RMSE of ~5.8 but the second order polynomial made it drop to ~5.7. That's not significant, so we will keep our First Order Polynomial as our base model. Now, let's see if we can create a more sophisticated model that can beat that.

The first thing to do now is to find cluster-based partitions (subsets of the data chosen using clustering) where the relationship between *Good Health %* and *Bad health %* is different. To do so, we first need to pick variables that we are going to use for our clustering.

The dataframe we have consists of 147 columns (crazy right ?). In order to choose the right columns for our clustering algorithm, we first need to understand what our columns mean and what type of information they provide. I checked the data in Excel, and I managed to group the columns we have into these categories :

- Location (geographical coordinates and surface area)
- Population count (raw number and density)
- Age (mean, median and percentages)
- Distance to work
- Sex
- Ethnic Group
- Religion
- Health Condition
- Social Status
- Economic Activity
- Occupation
- Qualification (Education level)

The way I'm going to proceed next to select the most useful features for my problem is by using a Wrapper Feature Selection method based on Linear Regression. Let's see what the suggested attributes are and perform the clustering on them !

In [27]:

```
# Feature Selection : Wrapper Method using Linear Regression
estimator = LinearRegression()
selector = RFE(estimator, 20, step=1) # Read the Sklearn Doc about RFE
selector = selector.fit(df_train.drop(columns=[good_edu, 'id', 'Name', 'Borough', 'CODE', 'OLDCODE'
, 'NAME', 'ALTNAME', 'geometry']), df_train[good_edu]) # Dropping the categorical variables
print('These are the 20 most relevant Features according to the wrapper method :\n', df_train.drop
(columns=[good_edu, 'id', 'Name', 'Borough', 'CODE', 'OLDCODE', 'NAME', 'ALTNAME', 'geometry']).col
umns[selector.support_].values)
```

```
These are the 20 most relevant Features according to the wrapper method :
 ['age=0 to 4: Population % by age' 'age=5 to 7: Population % by age'
 'age=8 to 9: Population % by age' 'age=10 to 14: Population % by age'
 'age=15: Population % by age' 'age=75 to 84: Population % by age'
 'age=85 to 89: Population % by age'
 'age=90 and over: Population % by age'
 'sex=male; economic activity=All usual residents aged 16 to 74: Population % by economic activity
and sex'
 'sex=male; economic activity=Active: Unemployed: Population % by economic activity and sex'
 'sex=male; economic activity=Inactive: Long-term sick or disabled: Population % by economic
activity and sex'
 'sex=female; economic activity=Active: In employment: Population % by economic activity and sex'
 'sex=female; economic activity=Inactive: Population % by economic activity and sex'
 'sex=female; economic activity=Unemployed: Age 16 to 24: Population % by economic activity and se
x'
 'qualification (study)=No qualifications: Population % by qualification or study'
 'qualification (study)=Level 1 qualifications: Population % by qualification or study'
 'qualification (study)=Level 2 qualifications: Population % by qualification or study'
 'qualification (study)=Apprenticeship: Population % by qualification or study'
 'qualification (study)=Level 3 qualifications: Population % by qualification or study'
 'qualification (study)=Other qualifications: Population % by qualification or study']
```

Right ! So Age is important, and so is Economic Activity. We're going to use variables from these two topics in our clustering. Qualification is obviously mentioned here as well because those variables are highly correlated with the target variable we want to predict : all of them sum up to 100.

First, let's list all the variables belonging to the Age and Economic Activity categories.

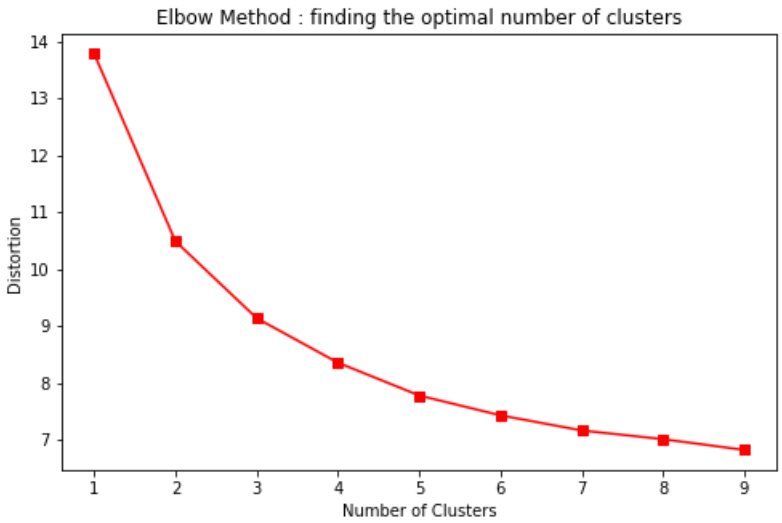
In [28]:

```
# Identifying the variables of each category of attributes
var_age = list(df_merged.columns[df_merged.columns.str.contains('Population % by age')])
var_econ = list(df_merged.columns[df_merged.columns.str.contains('Population % by economic
activity')])
var_X = 'health condition=Very good health: Population % by health condition'
var_y = 'qualification (study)=Level 4 qualifications and above: Population % by qualification or
study'
```

Now, I'd like to try out a method to finding out an optimal number of clusters for a given dataset. It's called the Elbow Method.

In [29]:

```
# K-means features
var_kmeans = var_age[8:12] + var_econ # We only keep ages between 25 and 64, hence the [8:12]

# Computing the distortion for every K (between 1 and 9) to find an optimal value of K
distortions = []
K = range(1,10)  # range of numbers of clusters K
for k in K:
    kmeanModel = KMeans(n_clusters=k).fit(df_merged[var_kmeans])
    distortions.append(sum(np.min(cdist(df_merged[var_kmeans], kmeanModel.cluster_centers_, 'euclid
ean'), axis=1)) / df_merged[var_kmeans].shape[0])
plt.figure(num=11, figsize=(8,5))
plt.xlabel('Number of Clusters')
plt.ylabel('Distortion')
plt.title('Elbow Method : finding the optimal number of clusters')
plt.plot(K, distortions, 'rs-');
```



The Elbow Method is a popular approach to estimating the optimal number of clusters. The principle is to compute Kmeans for different values of K (in this case, I went from 1 to 9) and for each K plot the associated distortion, which is nothing more than K-means' Cost Function (it represents the degree to which the intra-cluster distances are small and inter-cluster distances are big). The curve is always decreasing and reaches 0 when the number of clusters is equal to the number of data points (basically each point becomes a cluster). It's called the Elbow method because we're looking for the value of K after which the distortion levels off. In other words, the threshold at which we experience a jump in diminishing returns.

In this case, values from 3 to 5 are acceptable as optimal values. However, since I will be using these clusters to build separate models, I will stick to 3 clusters in order to keep my global model simpler.

In [30]:

```
# Create and Fit the model
```

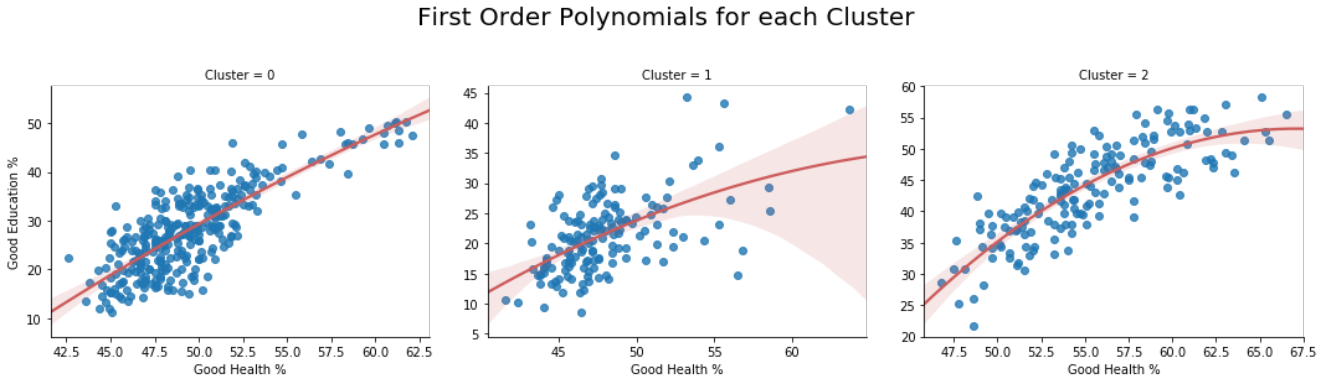
```
nb_clusters = 3
kmeans = KMeans(n_clusters=nb_clusters, random_state=123).fit(df_merged[var_kmeans])

# Append a column with the cluster number to each data point (row)
df_merged['Cluster'] = kmeans.predict(df_merged[var_kmeans])
```

In [31]:

```
# Visualizing 'Good Health %' and 'Good Education %' per Cluster
plt.figure(num=12)
ax = sns.lmplot(x = var_X, y = var_y, data = df_merged, order = 2,
                col = 'Cluster', col_wrap = 3, sharex = False, sharey = False, line_kws={'color':'ir
dianred'})
ax.set_axis_labels('Good Health %', 'Good Education %')
plt.suptitle('First Order Polynomials for each Cluster', size = 20)
plt.subplots_adjust(top=.8);
```

```
/Users/Samyer/Documents/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tu
ple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

```
<Figure size 432x288 with 0 Axes>
```



Alright, from the plot above we note that for the first cluster, the relationship seems linear. My first model is going to be linear for this cluster. Second cluster, the relationship isn't very clear, but I'm going for a first order linear regression here again to keep the global model simple. Last cluster is obviously showing a second order polynomial relationship. Alright let's build those 3 partial models.

In [32]:

```
# Splitting the data into a training set and a test set (Doing it again to include the new column
Cluster)
df_train, df_test = train_test_split(df_merged, test_size = .25, random_state = 123) # Test set = 2
5% of all data
```

In [33]:

```
# Building 3 different submodels : one for each cluster (on the same training data as our base mod
el)
lm_c0 = sm.OLS(df_train.loc[df_train.Cluster==0, good_edu], sm.add_constant(df_train.loc[df_train.C
luster==0, good_health])).fit()

lm_c1 = sm.OLS(df_train.loc[df_train.Cluster==1, good_edu], sm.add_constant(df_train.loc[df_train.C
luster==1, good_health])).fit()

lm_c2 = sm.OLS(df_train.loc[df_train.Cluster==2, good_edu],
               sm.add_constant(np.column_stack((df_train.loc[df_train.Cluster==2, good_health],
                                                df_train.loc[df_train.Cluster==2, good_health]**2))
.fit()

# Predicting the entire test set with each of the 3 sub-models
pred_c0 = np.array(lm_c0.predict(sm.add_constant(df_test[good_health])))
pred_c1 = np.array(lm_c1.predict(sm.add_constant(df_test[good_health])))
pred_c2 = np.array(lm_c2.predict(sm.add_constant(np.column_stack((df_test[good_health], df_test[go
od_health]**2)))))
```

```
# Combining the predictions : predict each test data point with the model corresponding to the clu
ster it belongs to
pred_g = np.array([pred_c0[index] if row.Cluster==0 else pred_c1[index] if row.Cluster==1 else pred
_c2[index]
                   for index,row in df_test.reset_index().iterrows()])
```

In [34]:

```
# Computing the value of Performance Metric (RMSE) for the sophisticated model (on the same test d
ata as the base model)
rmse_g = np.sqrt(mean_squared_error(df_test[good_edu], pred_g))
print('The RMSE of the sophisticated model is :', rmse_g)
```

```
The RMSE of the sophisticated model is : 4.471326864650877
```

Well, would you look at that ! We managed to reduce the RMSE by 1.3 (~22%) which is not bad at all. Obviously, the trade-off is that instead of having one first order model (our reference model), we now have two first order and one second order partial models combined, and you have to include the predictions of a clustering algorithm there as well. **Is it worth it ? You decide !**

“Life ain't about how hard you hit. It's about how hard you can get hit and keep moving forward.”
- Sylvester Stallone (Rocky Balboa)