FinalProject-Backup_20190424

April 24, 2019

Data Science Lifecycle - Bringing it all together
I590 - Applied Data Science
Created by Vivek Vijayaraghavan
Date: 04/24/2019
Objective: Final Project for Applied Data Science (Gap Minder Dataset)
Table of Contents

1 Overview

Data analysis is not something new that was uncovered as part of the newly define field fo data science. In fact, data analysis has been an integral part of the evolution of humans (or species in general). Maybe because of this intrinsic characteristic, it has been very difficult to explain on how would you go about performing data analysis.

In this project, I will attempt to walk through on how one can go about performing data analysis. The framework we will use to elaborate on the concept of data analysis is described in detail in The Art of Data Science and is set within the context of "Epicycles of Data Analysis".

The following sections will walk us through in understanding this framework all within the context of a single dataset and the various nuances within each of the analysis steps. Note that the "steps" can be varied and is not a sequential list, but more of a guideline as the sequence of "steps" can vary within the framework, based on the question we are trying a address as well as the type and availability of data etc. We will explore the dataset and work with Supervised and Unsupervised learning models that can help provide deeper insights and interpretations of what the data means in realtion to the question and hypothesis we started with.

Finally, while we address certain analysis topics in thsi project, it is by no means expected to be detailed, comprehensive and exhaustive elaboration of the various data science concepts. Rather, this report is a walk-through on how one can iteratively elaborate and understand the general approach to data analysis.

1.1 Pre-Requisites

This section is primarily focussed on any or all pre-requisites that are required for this project. Sometimes, we might import a specific library as part of the code where it is needed, I will be following the general principle of grouping "all things needed" for the successful execution of the project in this section. - The core programming language used here is python 3.7 - We will also use Jupyter Notebook to create and share this project so that it can contain live code and visualizations. - The notebook iteself including code, data and any other artifacts required as part of the report will be stored in GitHub. The link will be shared as part of the submission of the project. - As part of the execution of the project, we will need and import a variety of libraries and packages such as scikit-learn, ggplot2,pandas, numpy etc. These imports will listed below, so we can get an idea of what packages we are using within the project.

```
In [1]: # Import Libraries
        import pandas as pd
        import numpy as np
        import matplotlib
        import matplotlib.cm as cm
        import seaborn as sns
        import statsmodels.api as sm
        import statsmodels.formula.api as smf
        import statsmodels.stats.multicomp as multi
        import pylab
        from scipy import stats
        from scipy.spatial.distance import cdist

        from sklearn import datasets
        from sklearn import linear_model
        from sklearn.linear_model import LinearRegression
        from sklearn.metrics import mean_squared_error
        from sklearn.model_selection import train_test_split
        from sklearn import preprocessing
        from sklearn.cluster import KMeans
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.ensemble import ExtraTreesClassifier
        from sklearn.preprocessing import MinMaxScaler

        from matplotlib.pyplot import figure, show, rc
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D
        plt.style.use('ggplot')
        import matplotlib.mlab as mlab

        # Set plotting options.
        %pylab inline
        pylab.rcParams['figure.figsize'] = (10, 8)

        sns.set_style('whitegrid')
        sns.set_context('talk')

        # Eliminate false positive SettingWithCopyWarning
        pd.options.mode.chained_assignment = None
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
/anaconda3/lib/python3.7/site-packages/IPython/core/magics/pylab.py:160: UserWarning: pylab im
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```

1.2 References

- The Art of Data Science - By Roger D. Peng & Elizabeth Matsui
- Gap Minder Data Analysis
- Datacamp Supervised Learning
- Regression Model Diagnostics
- Regression Diagnostics Assumptions
- K-Means Clustering - Datacamp

1.3 Epicycles of Analysis

As part of the framework described in The Art of Data Science, the epicycles of data anaysis has 5 core activities. They are: - Stating & Refining the Question: In this activity, we will understand the type of question and hypothesis we would like to start with. We will continue to refine the question, based on reserach and available datasets and make it sharp (focussed), so that we will be able to either confirm or reject the hypothesis we started with. - Exploring the data: We will explore the data and examing the structure and the various components of the dataset, its distributions of the various variables and the relationships with each other. - Building formal statitical models: We build statistical models to understand the real world. We will use supervised and unsupervised learning models to help in classifying as well as predicting some of the implications of the social impact based on the data at hand. We will understand the model performance and refine the model to improve on its performance. - Interpreting the results: As part of tis activity, we will confirm or reject the hypothesis we started with and the question we asked. Possible options are we will need more data for performing further anlsysis, the results are conclusive and whether aligned or not aligned with our initial hypothesis. As with all statistical models, the aligmnent and acceptance (or rejction) of the hypothesis will have a probabilistic factor (measure of certainity or uncertainity). - Communicting the results: We will finally communicate the results. For this project, the communication of the results is part of the Jupyter Notebook report that provides a mechanism to highlight live code and its associated interpretations as we walk from the initial question to final interpretation.

1.4 Project Dataset

For the final project there are two datasets that we can work with. - Bike Vendors, which provides sales information of the various bike vendors - Gap Minder, which provides data associated to income, life expectancy by country over a time period.

Which of these datasets we will use depends on the questions we are interested in framing as well as what topics interest us to perform further analysis.

I will use the gapminder dataset as part of elaborating on the data analysis. I find myslef more and more curious of the data available to understand the social aspects and implications. While there is much more data available in the digital world, we will narrow our focus only to the variables provided as part of the initial dataset of this project.

While we will continue to explore in detail in the subsequent sections, I have summarized the gapminder dataset, its associated metadata and context of the data. Gapminder dataset that is used in this project provides data for each country in the world over a period of time. The data associated are population, life expectatncy, income and region to which a given country is associated. the definitions of each of the variables are: - Country: A specific country in the world. - Year: Year during which the data was captured / summarized. - Life: The average number of years a newborn child will live if the current mortality patterns were to remain the same. - Population: Total population count as provided by Maddison & UN sources. - Income: Gross Domestic Product (GDP) per person adjusted for differences in purchasing power (in International Dollars) - Region: Region of the world that the country belongs to.

2 Stating & Refining the Question

2.1 Question Characteristics

One of the first steps inolved while stating a question (or hypotheis) is to first understand the type of question we would be interested in. For this project we will narrow down to the following types of questions: - Decsriptive: Here we seek to summarize a characteristic of the data. Possible questions would include - How is income distributed across the regions of the world? - What is the life expectancy rate by country or region over time? - Exploratory: In this case, we would analyze the data to see if there are patterns, relationships or trends between the variables. Possible questions would include - Does higher life expectancy increase the income levels for a given country? - Does higher life expectatncy mean higher population counts? - Inferential: Based on exploring data, we might form certain hypothesis. A restatement of such hypotheis would include - There is a positive linear (or non-linear) relationship between life expectancy to the total population in a given country - There is a positive linear (or non-linear) relationship between life expectancy to the income levels in a given country. - Predictive: For these type of questions, we are seeking to understand (or predict) what would the future trend look like, such as - Given the current trend based on data available, can we predict which country will have the most population in 2020 (or 2025). - Causal: Here, we are not just looking for relationships between the variables, but seek to understand if one (or more variables) actually cause an impact to another set of variables. - On average, a increased life expectancy of 5% will cause an increase of population in a country by 10%. In this scenario, we are very specific that independent of income level increase or decrease, changes to life expectancy will affect the population count variable. - Mechanistic: Here we seek to actually understand how the variables affect one another. - How does the rate of change life expectancy affect the income levels in a given country?

2.2 Stating the Question

The question that I am most interested in is - "What is the relationship between Income Level and Life Expectancy features?". My hypothesis would be that there is a strong positive relationship between Income and Life across the countries and regions. (In this case, this means our null-hypothesis is that there is no significatnt relationship between Income and Life variables).

2.3 Decribing the data

In this section we will import, load and understand the gapminder dataset provided. We will also understand the characteristics of the data such as missing values, types of variables, how tidy is the data. We will continue to research external sources for any additional frame of reference, pre-process the dataset making it ready for further analysis.

As we understand the data and its characteritics better, we will continue to refine the question and understand if there is a hypothesis we can propose and validate.

2.3.1 Import the Gapminder Dataset

```
In [2]: # Import the gapminder data as provided for the project.
        # Read the CSV file into a DataFrame: gm_df
        gm_df = pd.read_csv('../data/gapminder.csv')

        # Make the column names (Variable Names) consistent
        gm_df.columns = ["Country", "Year", "Life", "Population", "Income", "Region"]

        # The top 5 rows in the dataset..
        # gm_df.head()
```

```
In [3]: # The bottom 5 rows in the dataset..
        gm_df.tail()
```

```
Out[3]:       Country  Year   Life Population  Income                 Region
        41279   Åland  1993  81.80      24950     NaN  Europe & Central Asia
        41280   Åland  1994  80.63      25066     NaN  Europe & Central Asia
        41281   Åland  1995  79.88      25183     NaN  Europe & Central Asia
        41282   Åland  1996  80.00      25301     NaN  Europe & Central Asia
        41283   Åland  1997  80.10      25419     NaN  Europe & Central Asia
```

Just based on the top and bottom 5 rows, we can already see that there might be missing data and different variable data types. We can also see that region is a categorical variable, while Life, Population and Income are Quantitative valriables. This knowledge will come in handy later as we perform different analysis on the data set. We will now go on to better understand the dataset we have loaded.

2.3.2 Describe the gapminder dataset

The info() and shape function provides additional information relating to the dataset we just loaded.

```
In [4]: # Understand the dataset. How many rows and columns were loaded?
        gm_df.shape
```

```
Out[4]: (41284, 6)
```

```
In [5]: # Understand the dataset. What are the data types and structures of the dtaaset?
        gm_df.info()

        # We could fix Year to be categorical variable and type string object...
        # This will come in handy especially when we perform regression and cluster analysis
        gm_df['Year'] = gm_df['Year'].astype(str)
        # Understand the dataset. What are the data types and structures of the dtaaset?
        print('')
        print('Gapminder Dataframe datatypes after converting Year to String object...')
        print('')
        gm_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41284 entries, 0 to 41283
Data columns (total 6 columns):
Country       41284 non-null object
Year          41284 non-null int64
Life          41284 non-null float64
Population    15467 non-null object
Income        38943 non-null float64
Region        41284 non-null object
dtypes: float64(2), int64(1), object(3)
memory usage: 1.9+ MB

Gapminder Dataframe datatypes after converting Year to String object...

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41284 entries, 0 to 41283
Data columns (total 6 columns):
Country       41284 non-null object
Year          41284 non-null object
Life          41284 non-null float64
Population    15467 non-null object
Income        38943 non-null float64
Region        41284 non-null object
dtypes: float64(2), object(4)
memory usage: 1.9+ MB
```

We can see that the data structure of the loaded data is a pandas dataframe and it has 41284 observations and 6 variables. Manually checking with the raw data, we can confirm that all of the data was imported successfully.

In addition to the obervations and variables, we can also see that various data type of the variables. - The Country and Region variables of the string object datatype. In addition these variables are considered categorical variables (Nominal). - Year is also be considered as a categorical variable (Ordinal). - Life, Popoulation and Income are Numerical variables (Continuous). - Based on the question we had stated, Life would be considered as a target variable and all Population and Income would be considered features (inputs to the model selection below). Other variables such as Country, Year and Regions would be considrед as labels for categorizing data.

Based on the above characteristics, we can see that Supervised Learning models of type Classification is not be best suited for this dataset. Classification typically requires target variables to be Categorical in nature. Hence, we will focus on the Regression Model for Supervised Learning and Clustering for Unsupervised Learning models.

2.3.3 Pre-Processing messy data

Before we proceed forward with more exploration of the data, it is best to pre-process the data for any missing or inconsistent values. - The dataset itself seems to be tidy - Each variable represents data for that variable and does not represent composite data. - Each observation is contained in each row and is not spread across multiple rows. - The variables are clearly defined. - The Population variable is also loaded as a string object. The reason for this is because we have blanks in the propulated data. - We will first address the missing values as there is a lot of them (25817 observations) - In addition, there are some formatting issues with the data. Some of the data contains "," as a seperator while others do not. We need to remove all "," from the population data for it to be considered numeric. - Convert the datatype to int64 - The Income variable also has some missing values, but not as much as the Population variable.

2.3.4 Fixing Data - Population

Missing Data There are four possible ways to fix the Population variable missing data. - Drop the observations that have missing (or NaN) data. Since there is a large number of observations with population data missing, but other data available, we will lose close to two-thirds of the data. This is not acceptable. - We can default (impute) the data to a particular value. W could default to 0. However, this will not be correct as it would represent that the population of a country in that year vanished, which is not correct. We could default to the previous closest year hat has a value. This will be more correct as atelast we are closer to the data around the time. However, we can do better. - We could replace the missing data with min or max or average across the dataset or even across a decade (date range). This will possibly be closer to the expected raw data. Tis would mean lesser bias from the dataset. - The last option is to reasearch if there is any authoritative source that can provide us with the missing data. We have to be extremely careful with this option as the context of the data collected might be different from the original intent. The data even then might not be a complete overlap and could still require fixes.

In our case, we will consider the population of missing data to be the same as the closest non missing data of the previous year.

Format Data We will need to strip all "," that is used as a seperator so that the Population variable can be converted to numeric data type.

```
In [6]: # Convert Population variable missing values to NaN.
        gm_df['Population'] = gm_df['Population'].replace('', np.nan)

        # Convert NaN to the previous observation value...
        gm_df['Population'].fillna(method='ffill', inplace=True)

        # Strip all "," used as a seperator in the Population variable...
        gm_df['Population'] = gm_df['Population'].str.replace(',', '')
        # gm_df['Population'] = gm_df.Population.astype(str).str.rstrip(',')

        # Now convert the Population data type to numeric...
        gm_df['Population'] = pd.to_numeric(gm_df['Population'])
        # gm_df['Population'] = gm_df.Population.astype(float)

        gm_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41284 entries, 0 to 41283
Data columns (total 6 columns):
Country       41284 non-null object
Year          41284 non-null object
Life          41284 non-null float64
Population    41284 non-null int64
Income        38943 non-null float64
Region        41284 non-null object
dtypes: float64(2), int64(1), object(3)
memory usage: 1.9+ MB
```

2.3.5 Fixing Data - Income

Missing Data We could follow the same approach for income like what we did for Population variable. However, in this case, it might not be as appropriate as the previous values of income does not exist. We could reserach if there was supplemental data available for the missing income data and try and reduce the number of missing income data prior to dropping the missing income data.

However, it should be noted that even after bringing in the reference data, there was not much data that was updated. This is because the Country names are different and does not match exactly. We should be careful of such inconsistencies. We will continue by dropping the NaN variables when needed.

It seems to be best to just drop the na records. However, the exposure is that we lose close to 6% of the dataset.

```
In [7]: # Total number of NaN values in Income variable
        print('Number of observations with missing Income data {}'
              .format(pd.isna(gm_df['Income']).sum()))
        #gm_df[(gm_df['Income'].isnull()) & (gm_df['Region'] == 'America' )]
        print('% of data with missing income values {}%'
              .format(round((pd.isna(gm_df['Income']).sum()/gm_df.Income.count())*100,2)))

        #df_left[(df_left['Income'].isnull()) & (df_left['Country'] == 'French Guiana' )]
```

```
Number of observations with missing Income data 2341
% of data with missing income values 6.01%
```

```
In [8]: # Get income data from gapminder.org
        gm_inc = pd.read_csv('../data/income_per_person_gdppercapita_ppp_inflation_adjusted.cs
        # gm_inc.head()
```

```
In [9]: # Convert wide table into long (Tidy data)
        gm_inc_norm = pd.melt(gm_inc, id_vars=["country"],
                              var_name="Year", value_name="Value")

        # gm_inc_norm.head()
        gm_inc_norm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 46513 entries, 0 to 46512
Data columns (total 3 columns):
country    46513 non-null object
Year       46513 non-null object
Value      46513 non-null int64
dtypes: int64(1), object(2)
memory usage: 1.1+ MB
```

```
In [10]: # Perform a left join between gm_df and gm_inc_norm dataframes
         # Overlay NaN in the gm_df with the reference data downloaded
         gm_df_upd = (pd.merge(gm_df, gm_inc_norm, left_on=['Country','Year'],
                               right_on = ['country','Year'], how='left'))

         gm_df_upd ['Income']= np.where(gm_df_upd ['Income'].isnull(), gm_df_upd ['Value'], gm

         # Print the dataframe info for the merged dataframe
         print(gm_df_upd.info())

         print('')
         print('Number of observations with missing Income data {}'
               .format(pd.isna(gm_df_upd['Income']).sum()))
         print('')
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41284 entries, 0 to 41283
Data columns (total 8 columns):
Country       41284 non-null object
Year          41284 non-null object
Life          41284 non-null float64
Population    41284 non-null int64
Income        38963 non-null float64
Region        41284 non-null object
country       37506 non-null object
Value         37506 non-null float64
dtypes: float64(3), int64(1), object(4)
memory usage: 2.8+ MB
None

Number of observations with missing Income data 2321
```

It can be seen from the above that we were only able to update 20 records where income had missing values. The primary reason for this is the country names do not align between our initial dtaaset and teh reference dataset downloaded. So, we will continue with dropping the missing values in the income variable. We will also cleanup the dataframe by removing all other variables we introduced as part of the join.

```
In [11]: # Drop the variables country and Value from gm_df_upd dataframe
         gm_df_upd = gm_df_upd.drop(['country', 'Value'], axis=1)

         # Also drop na records from gm_df dataset
         gm_df_upd = gm_df_upd.dropna()

         # Print and understand the charcteristics of the gm_df_upd dataframe
         print(gm_df_upd.info())
         #print('')
         #print(gm_df_upd.head())
         #print('')
         print('')
         print('Number of observations with missing Income data {}'
               .format(pd.isna(gm_df_upd['Income']).sum()))
         print('')
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38963 entries, 0 to 41273
Data columns (total 6 columns):
Country       38963 non-null object
Year          38963 non-null object
Life          38963 non-null float64
Population    38963 non-null int64
Income        38963 non-null float64
Region        38963 non-null object
dtypes: float64(2), int64(1), object(3)
memory usage: 2.1+ MB
None

Number of observations with missing Income data 0
```

2.3.6 Other characteristics of the dataset

Dispersity of data Now, let us turn our attention to other characteristics of the gapminder dataset. We will start by looking at the summary statistics of the dataset. W will ignore the Year variable for our immediate consideration. The reason for this is because the variations in the data (min / max) is not that much.

The population and Income variables have a very wide range. - Population ranges from 1548 to 1,376,049,000. This is a wide range and we need to consider this when we apply regression or clusterning models on this dataset for bias and over/under fitting. - Similarly, the income also has a wide range from 142 to 182,688 amd need to factored in for bias. - As we perform more Exploratory Data Analysis in the subsequent section, we will consider normalizing the data as we consider model performance metrics.

```
In [12]: # Describe function that provides summary statistics of numerical data
         gm_df_upd.describe()
```

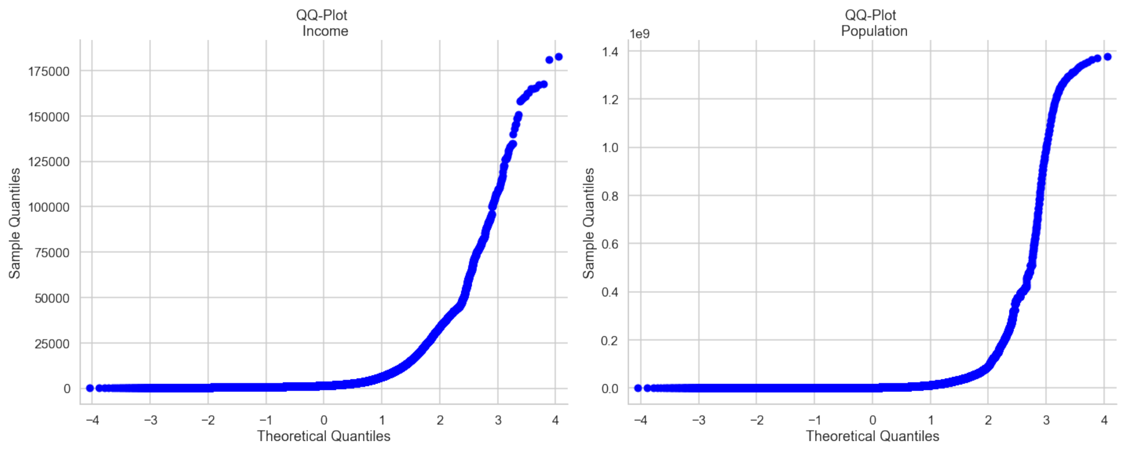
```
Out[12]:              Life    Population          Income
         count  38963.000000  3.896300e+04   38963.000000
         mean      42.935663  1.328037e+07    4569.059313
         std       16.141942  6.409550e+07   10094.242828
         min        1.000000  2.128000e+03     142.000000
         25%       31.100000  3.572800e+05     883.000000
         50%       35.400000  1.821769e+06    1449.000000
         75%       55.566330  6.057815e+06    3480.500000
         max       84.100000  1.376049e+09  182668.000000
```

Centering & Scaling Centering and scaling is also another very important pre-processing step to be performed for supervised learning. The scale of the features will adversely affect the model performance as the prediction can be biased based on scaled parameters. We know that both the Population and Income variables had a wide range of data and can bias the model performance.

Let us first understand the data distribution of our dataset by using a QQ-Plot. QQ-Plot is a good indicator to understand if the data is normally distributed. Based on the plot below, we know that the data is not normally distributed as the dots will all align towards a stright line across the diagonal of the plot, should it be a normal distribution.

```
In [13]: # Let us plot a QQ-Plot for Income & Population Variables..
         plt.tight_layout()
         #fig, ax =plt.subplots(1,2)
         fig, ax = plt.subplots(ncols=2, figsize=(20,8))
         ax[0].set_title('QQ-Plot \n Income')
         ax[1].set_title('QQ-Plot \n Population')

         fig = sm.qqplot(gm_df_upd.Income, ax=ax[0])
         fig = sm.qqplot(gm_df_upd.Population, ax=ax[1])

         fig.tight_layout()
         sns.despine()
```

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



While there are many ways to normalize the data, one common way is called Standardization. This method is subtracting the mean and dividing by the variance. This will make sure that all features are centralized around mean=0 with a variance=1.

```
In [14]: # Generate the feature and target variables
         gm_df_upd_y = gm_df_upd['Life'].values

         gm_df_upd_X = gm_df_upd.drop(['Life','Country','Year','Region'], axis=1).values
         #gm_df_upd_X = wwq.values
         #gm_df_upd_X.shape,gm_df_upd_y.shape
         #gm_df_upd.shape
         #gm_df_upd_X_scaled.shape
```

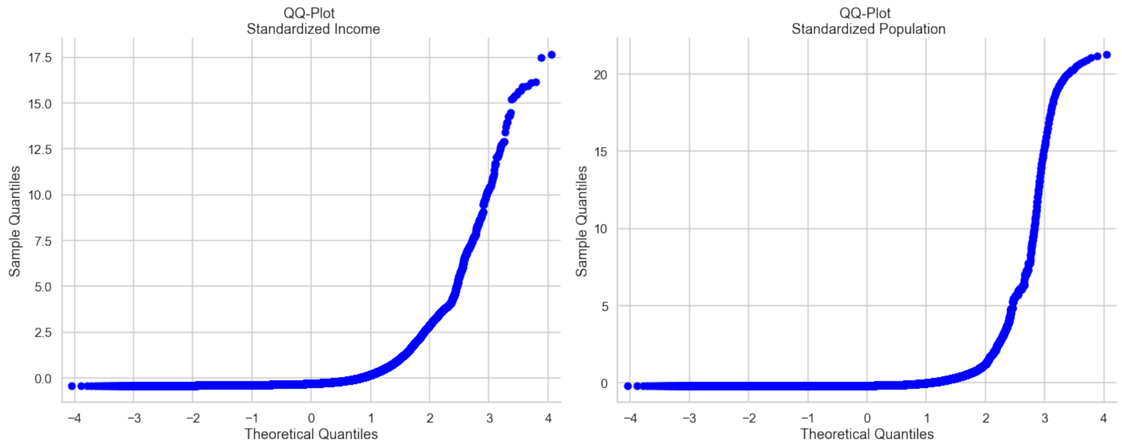
```
In [15]: # This section highlights the standardization of data
         # Import scale
         from sklearn.preprocessing import scale

         # Scale the features: X_scaled
         gm_df_upd_X_scaled = scale(gm_df_upd_X)

         # Print the mean and standard deviation of the unscaled features
         print("Mean of Unscaled Features: {}".format(np.mean(gm_df_upd_X)))
         print("Standard Deviation of Unscaled Features: {}".format(np.std(gm_df_upd_X)))

         # Print the mean and standard deviation of the scaled features
         print("Mean of Scaled Features: {}".format(np.mean(gm_df_upd_X_scaled)))
         print("Standard Deviation of Scaled Features: {}".format(np.std(gm_df_upd_X_scaled)))
```

```
Mean of Unscaled Features: 6642469.43138362
Standard Deviation of Unscaled Features: 45805304.47108211
Mean of Scaled Features: -9.482899740657858e-18
Standard Deviation of Scaled Features: 1.0
```

```
In [16]: gm_df_std = pd.DataFrame(gm_df_upd_X_scaled)

         gm_df_std.columns = ['std_Population', 'std_Income']

         # Let us plot a QQ-Plot for Income & Population Variables..
         plt.tight_layout()
         #fig, ax =plt.subplots(1,2)
         fig, ax = plt.subplots(ncols=2, figsize=(20,8))
         ax[0].set_title('QQ-Plot \n Standardized Income')
         ax[1].set_title('QQ-Plot \n Standardized Population')

         fig = sm.qqplot(gm_df_std.std_Income, ax=ax[0])
         fig = sm.qqplot(gm_df_std.std_Population, ax=ax[1])
```

```
fig.tight_layout()
sns.despine()
```

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



As can be seen from the above plots, Standardization rescales the data, but does not change the distribution of data. The data distribution of our data is still a non-linear distribution. We will now perform a data transformation (as oposed to scaling) and observe the effect of log transformation of the data.

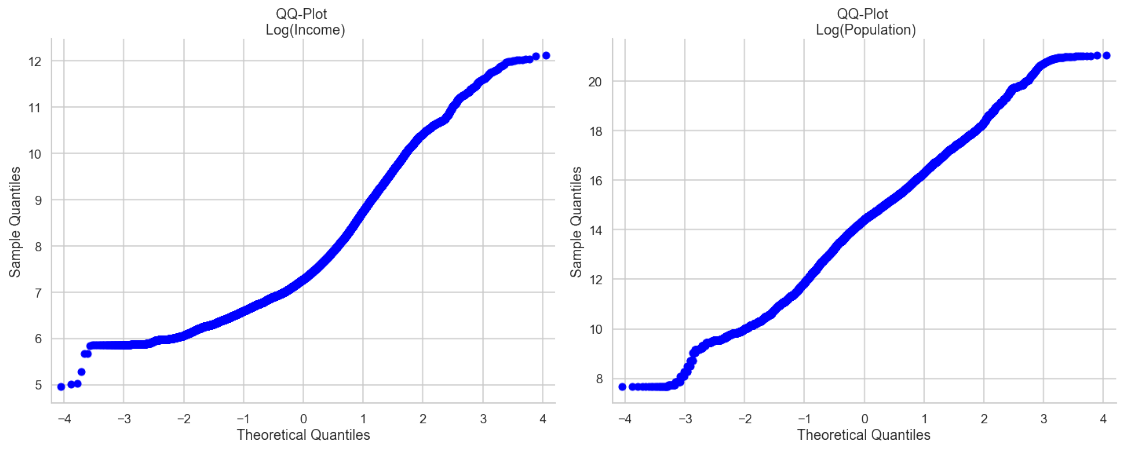
Another method is to log transform the data. In scenarios where there is a wide range of data values, log transform is also know to help normalize the data. The primary goal is to validate if we can align the data to normal distribution and validate if we can get better correlation between the feature and target variables.

```
In [17]: #Create dataframe that will hold the log values of Population & Income
         gm_df_log = gm_df_upd.copy(deep=True)

         gm_df_log['log_Population'] = np.log(gm_df_log['Population'])
         gm_df_log['log_Income'] = np.log(gm_df_log['Income'])

         #Cleanup the dataframes of other variables
         gm_df_log = gm_df_log.drop(['Income','Population'], axis=1)
         #gm_df_upd = gm_df_upd.drop(['log_Income','log_Population'], axis=1)
```

```
In [18]: # Let us plot a QQ-Plot for Log Transformed Income & Population Variables..
         plt.tight_layout()
         #fig, ax =plt.subplots(1,2)
         fig, ax = plt.subplots(ncols=2, figsize=(20,8))
         ax[0].set_title('QQ-Plot \n Log(Income)')
         ax[1].set_title('QQ-Plot \n Log(Population)')
```

```
        fig = sm.qqplot(gm_df_log.log_Income, ax=ax[0])
        fig = sm.qqplot(gm_df_log.log_Population, ax=ax[1])

        fig.tight_layout()
        sns.despine()
```

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



As can be seen from the above plots, the log transformed Income and Population variables are closer to the normal distribution than that of Non Transformed data. Note that while the log transformed data is closer to normal, it is not a perfect normal distribution as is expected for such scenarios. We will continue to carry both the Non Transformed and Log Transformed data as we further analyze and select models in the subsequent sections.

2.4 Refining the Question

Let us now, refine the above questions listed above based on 5 key characteristics: - Interest: Of the questions above, understanding the relationship between life expectancy, population and income is definitely of interest. More interesting is also to understand if there is a backward relationship between these variables. Is the income level higher because of higher life expectancy or is the life expectancy higher because of higher income levels (so we can afford better healthcare), or both? - Unanswered: While there is lot of reaserach done in this area, for the sake of this project, we will assume that the above questions have not been answered. - Plausible: It is reasonable to conclude that the questions relating to causal and mechnistic types are not plausible within the timeframe of this project, we can do much better than the descriptive type. Henece, we will narrow down our focus to the exploratory, inferential and predictive types of questions. - Answerable: Based on the data provided, we can certainly answer exploratory and inferential types of questions. While we can get a rudimentary version of prediction, we all know that there are much more variables that influence and affect the social characteristics within a country / region over time. Hence, we will narrow down our question to not the future years, but to be able to classify or cluster based on the relationship between the variables. - Specificity: Between the Exploratory and Inferential type of questions, we will now now need to get a little bit more specific on our question. We will continue to narrow the question down to the following:

Refined Question / Hypothesis What is the impact of income levels to Life Expectancy across the regions of the world?

The hypothesis is that people live longer when the income levels are higher independent of the country or region they are from. (Positive relation between Life and Income variables). In this case, this means our null-hypothesis is that there is no significatnt relationship between Income and Life variables.

3 Exploratory Data Analysis

Let us do more analysis on the data. Before we do that, here is a summary of what we know so far. - The gapminder dataset does not follow a normal distribution with non transformed data. - Scaling and centring (Standardization) helped scale the data but did not change the distribution. - Log Transforming the data helped bring the distribution closer to normal distribution. - Based on our question / hypothesis, Life is our Target Variable (Dependent), while Income and Population were our Features (Independent variable). - We do have Country, Year and Region as our labels.

Now let us plot a basic scatter plot, between Life and Income, where each dot represents a Country and the size of the dot represents the population of that country. Note that these values are the average over a time period between 1800 - 2015.

3.1 Plots & Visualization

We will start by creating a dataframe that summarizes the Life, Income and Population data by Country and Region. We will use this as the base dataset as we analyze the relationship between Life, Income and Population variables by country and/or region.

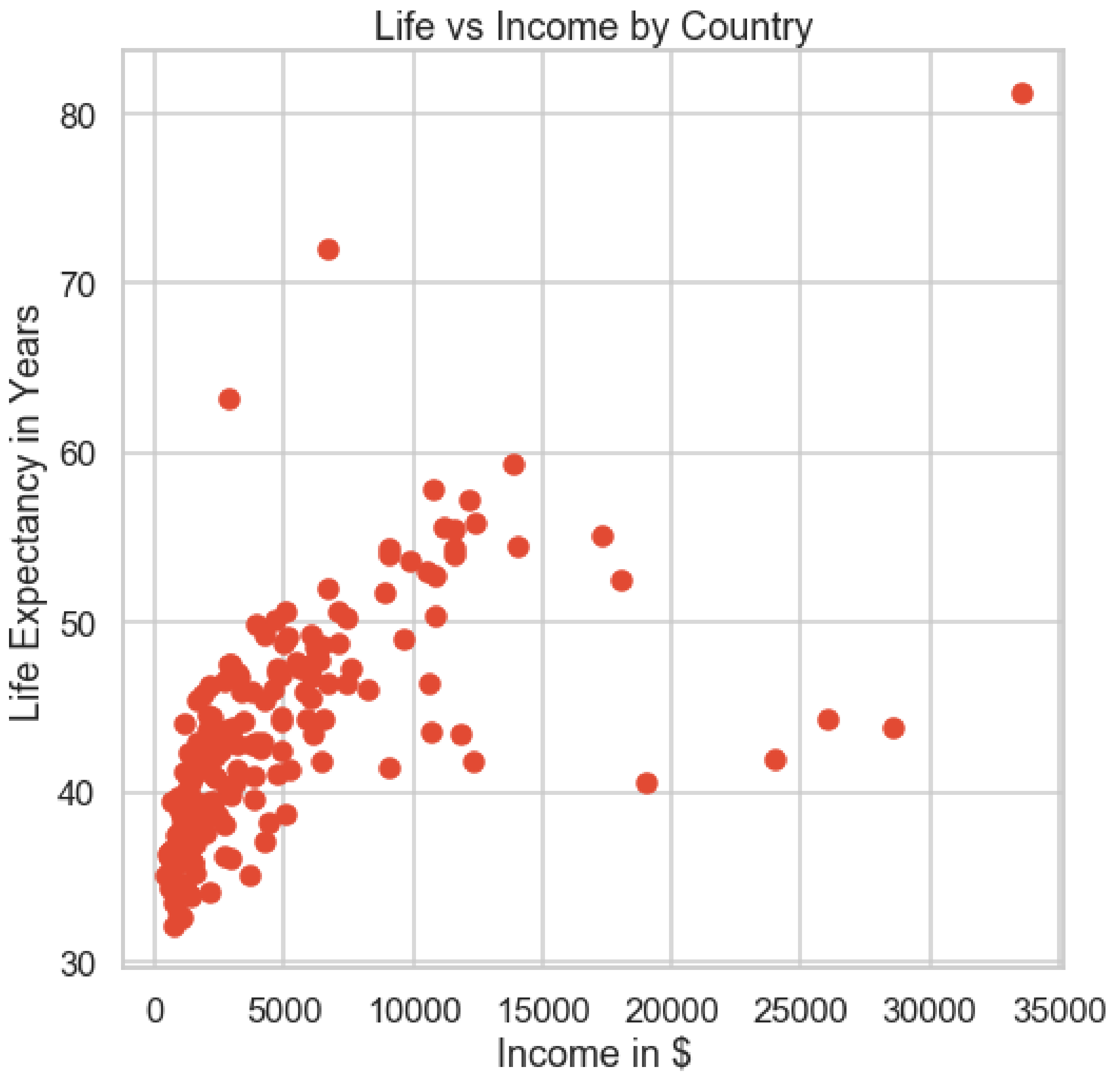
```
In [19]: # Creating a dataframe that summarizes by Country and Region...
         gm_df_country = (gm_df_upd.groupby(['Country', 'Region'], as_index=False).mean()
                          [['Country', 'Region', 'Life', 'Population', 'Income']])

         gm_df_country_log = (gm_df_log.groupby(['Country', 'Region'], as_index=False).mean()
                          [['Country', 'Region', 'Life', 'log_Population', 'log_Income']])
         #gm_df_country.head()
```

```
In [20]: # Creating feature and target arrays...
         # We will use Life as target array and Population & Income as feature array...
         gm_df_country_X = gm_df_country.drop('Life', axis=1).values
         gm_df_country_y = gm_df_country['Life'].values
         gm_df_country_cols = gm_df_country.drop(['Country','Region','Life'], axis=1).columns
         #gm_df_country_cols = gm_df_country.drop('Life', axis=1).columns
         gm_df_country_X.shape, gm_df_country_y.shape

         # Reshaping to get both the arrays aligned...
         gm_df_country_X_Income = gm_df_country_X[:,3]
         gm_df_country_y = gm_df_country_y.reshape(-1,1)
         gm_df_country_X_Income = gm_df_country_X_Income.reshape(-1,1)
         #gm_df_country_X_Income.shape, gm_df_country_y.shape
```

3.1.1 Life vs Income by Country

Let us visualize a basic scatter plot between Life and Income variables. Each dot in the plot represents a country. Based on the plot below, we can observe the following: - There are lot more countries whose average income is lower. - The general pattern that can be seen is that as income rises, so does the life expectancy. - There are certain countries who have higher income, but lower life expectancy. Likewise, there are a few countries whose income levels are lower but with much higher life expectancy. This leads me to believe that there would be other factors that influence life expectancy than just income.

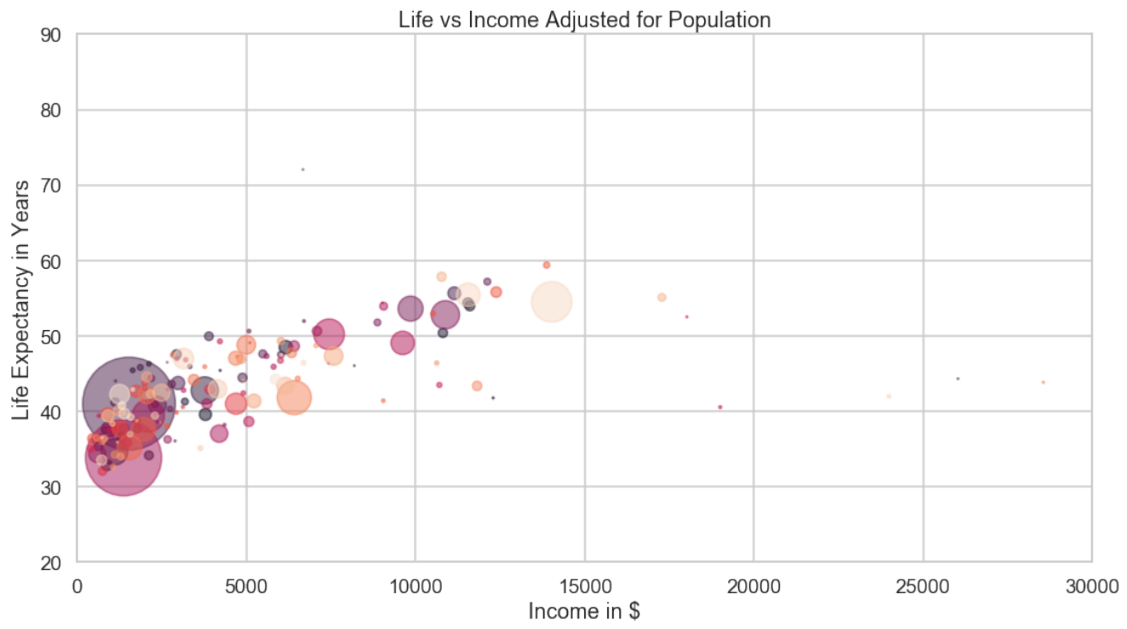
```
In [21]: # Plotting Life as a function of Income..
         # We can see more Income lead to longer values..
         plt.figure(figsize=(8,8))
         plt.scatter(gm_df_country_X_Income, gm_df_country_y)
         plt.ylabel('Life Expectancy in Years')
         plt.xlabel('Income in $')
         plt.title('Life vs Income by Country')
         plt.show()
```



3.1.2 Life vs Income by Country (Adjusted for Population)

While the basic scatter plot gives a general idea, let us overlay the population of each country in the plot as well. In this plot, each country is represented by a different color and the size of the dot represents relative average population of that country. Note that the scale of the population is 1/10000 of its actual average, so we can plot them in the canvas. Some additional observations are: - Population does not stand out as a major influencer in the relation between Life and Income. While in general, I would have expected to see higher income for larger population, it is clear from the plot that it is not the case. - It is also not clear that the life expectancy does not increase proprtionately (not linear) to the population size.

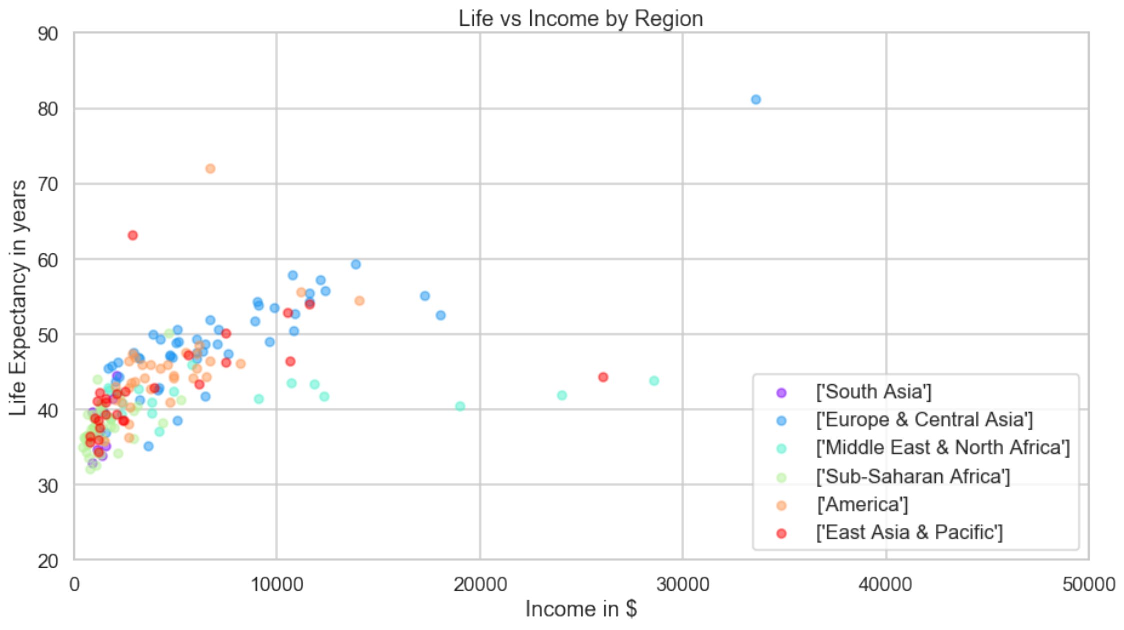
```
In [22]: plt.figure(figsize=(15,8))
         #N = gm_df_country.Region.count()
         x = gm_df_country_X_Income
         y = gm_df_country_y
         colors = pd.factorize(gm_df_country['Country'])[0]
         colors = colors.reshape(-1,1)
         area = np.round(gm_df_country.Population.values/100000,0)  # 0 to 15 point radii

         plt.scatter(x, y, s=area, c=colors, alpha=0.5) # Change color and transparency
         #plt.scatter(x, y, s=area, alpha=0.5) # Change color and transparency

         plt.ylabel('Life Expectancy in Years')
         plt.xlabel('Income in $')
         plt.title('Life vs Income Adjusted for Population')
         plt.xlim([0, 30000])
         plt.ylim([20, 90])

         plt.show()
```



3.1.3 Life vs Income by Region

In the final scatter plot, let us group (color by Regions) to observe if there is any inference we can draw on the relationship between Life and Income variables by regions. - We can see that certain regions (Europe, America) have higher incomes and Life Expectancy - Certain other regions (Sub Saharan Africa) have low income and life expectancy - South Asia region is interesting because while the income levels are not as high, some countries in this region have higher life expectancy. This indicates that there are other variables that influence life expectancy. - Middle East region has higher levels of income, but lower life expectancy. Other reasons like war might be influencing the life expectancy.

```
In [23]: plt.figure(figsize=(15,8))

         #colors = colors =10*["g","r","c","b","k",]

         regions = ({'South Asia':0,'Europe & Central Asia':1,'Middle East & North Africa':2,
                     'Sub-Saharan Africa':3, 'America':4, 'East Asia & Pacific':5})


         colors = cm.rainbow(np.linspace(0, 1, len(regions)))

         for i in range(0, len(gm_df_country['Region'].unique())):
             regions_df = gm_df_country[gm_df_country['Region'].map(regions) == i]
             plt.scatter(
                 regions_df['Income'],
                 regions_df['Life'],
                 color=colors[i],
                 s=50,
                 alpha=0.5,
                 label=regions_df['Region'].unique()
             )

         plt.xlabel('Income in $')
         plt.ylabel('Life Expectancy in years')
         plt.title('Life vs Income by Region')
         plt.legend(loc='lower right')
         plt.xlim([0, 50000])
         plt.ylim([20, 90])

         plt.show()
```

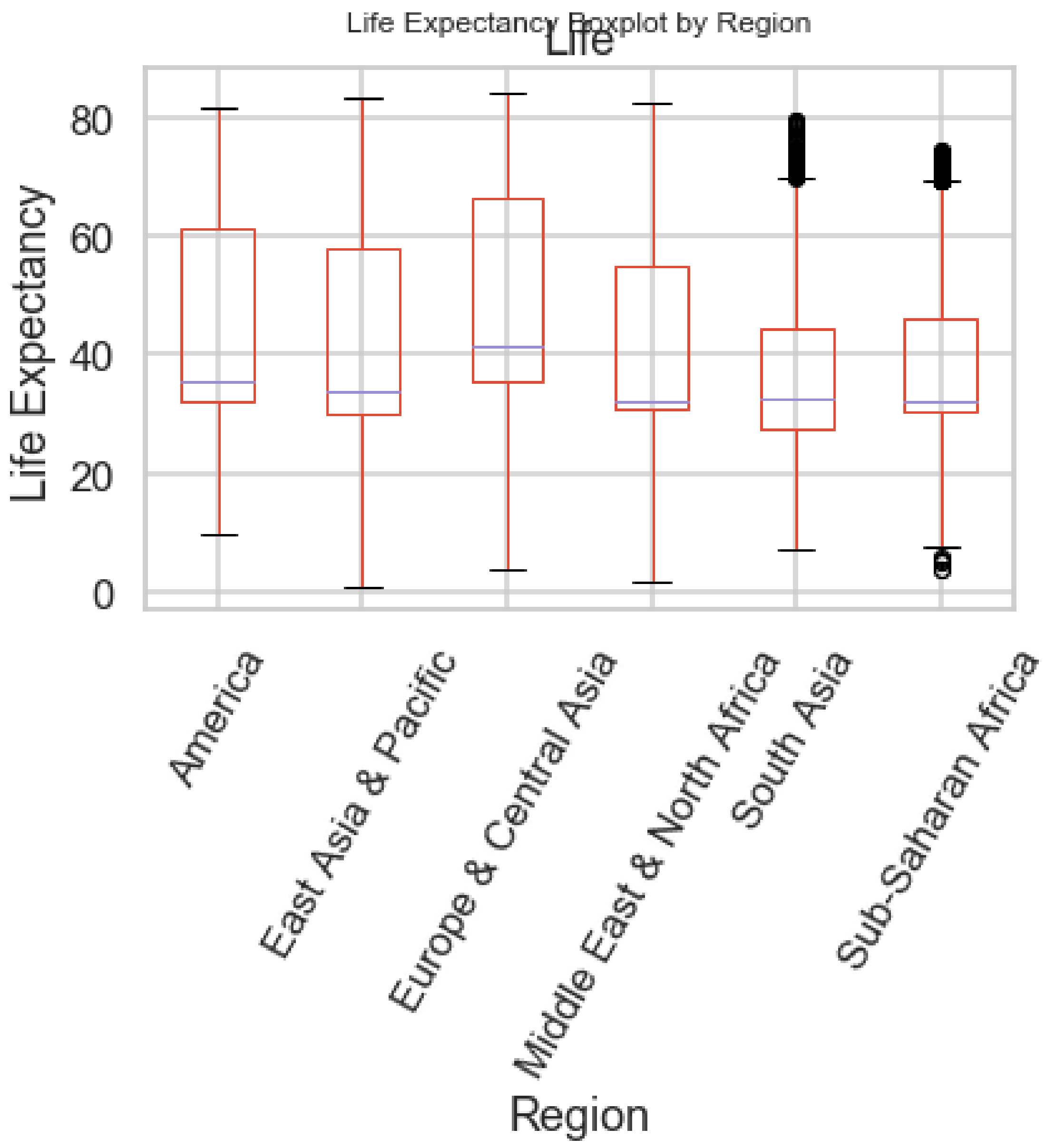


3.1.4 Boxplot of Life by Regions

A box plot of Life vs Income by regions also reiterate the same observations. We can see that there are outlier for South Asia and Sub-Saharan Africa regions, indicating that while the life expectancy in these regions are low, certain countries within the region fall outside this norm. Note that the box plot does not provide any indication of relation between Life and Income.

```
In [24]: # Let us dwell a little bit on the Regions..
         # Let us remove the Country variable out of the mix..
         gm_df_regions = gm_df_upd.drop(['Country'], axis=1)


         # Create a boxplot of life expectancy per region
```

```
gm_df_regions.boxplot('Life', 'Region', rot=60)

# Show the plot
plt.xlabel('Region')
plt.ylabel('Life Expectancy')
plt.suptitle('Life Expectancy Boxplot by Region')
plt.show()
```

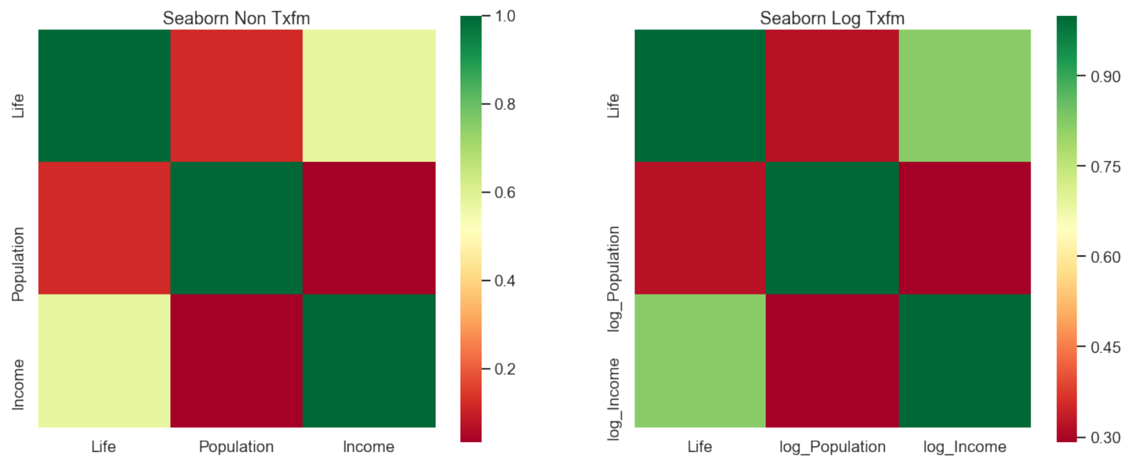


3.1.5 Seaborn Heatmap

Seaborn heatmap is another way to understand the relationship between the variables in the dataset. Plotted below are two seaborn heatmaps. The left is a heatmap of non transformed data of gap minder while the right is log transformed. It can be observed: - There is NO negative correlation between the variables in discussion (Life, Income and Population). - The correlation is strengthened by log transformed data than non transformed data. - The correlation between Life / Income is much stronger than the Life / Population correlation.

```
In [25]: # Now plot a heatmap showing the correlation between the different
         # features of the Gapminder dataset
         # Cells that are in green show positive correlation, while cells that are in red show
         # negative correlation.
         #plt.tight_layout()
         #fig, ax = plt.subplots(1,2)
         fig, ax = plt.subplots(ncols=2, figsize=(20,8))
         ax[0].set_title('Seaborn Non Txfm')
         ax[1].set_title('Seaborn Log Txfm')

         sns.heatmap(gm_df_upd.corr(), square=True, cmap='RdYlGn', ax=ax[0])
         sns.heatmap(gm_df_log.corr(), square=True, cmap='RdYlGn', ax=ax[1])
```

```
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1903d6d8>
```



The raw correlation values is used in the seaborn heatmap. These values are printed below and is no different than the ones visualized above.

```
In [26]: print('The correlation values for non transformed data is...')
         print('')
         print(gm_df_upd.corr())
         print('')
         print('')
         print('The correlation values for log transformed data is...')
```

```
        print('')
        print(gm_df_log.corr())
```

```
The correlation values for non transformed data is...

                Life  Population    Income
Life        1.000000    0.118539  0.576439
Population  0.118539    1.000000  0.034345
Income      0.576439    0.034345  1.000000


The correlation values for log transformed data is...

                    Life  log_Population  log_Income
Life            1.000000        0.319109    0.818373
log_Population  0.319109        1.000000    0.291940
log_Income      0.818373        0.291940    1.000000
```

3.2 Supervised Learning

In this section we will continue to analyze the gapminder data and build a supervised learning model based on the dataset as well understand the model performance. Since the Target and Feature variables are all quantitative, we will perform Regression analysis instread of Classification analysis.

Linear regression is where try and fit a line to the model data. The formula for line is:

```
y = ax + b, where

y = target variable = Life
x = single feature = Income, and
a (slope),b (constant) are the parameters of the model
```

For our analysis, going forward, we will use the log transformed data for our analysis.

3.2.1 Linear Regression Model

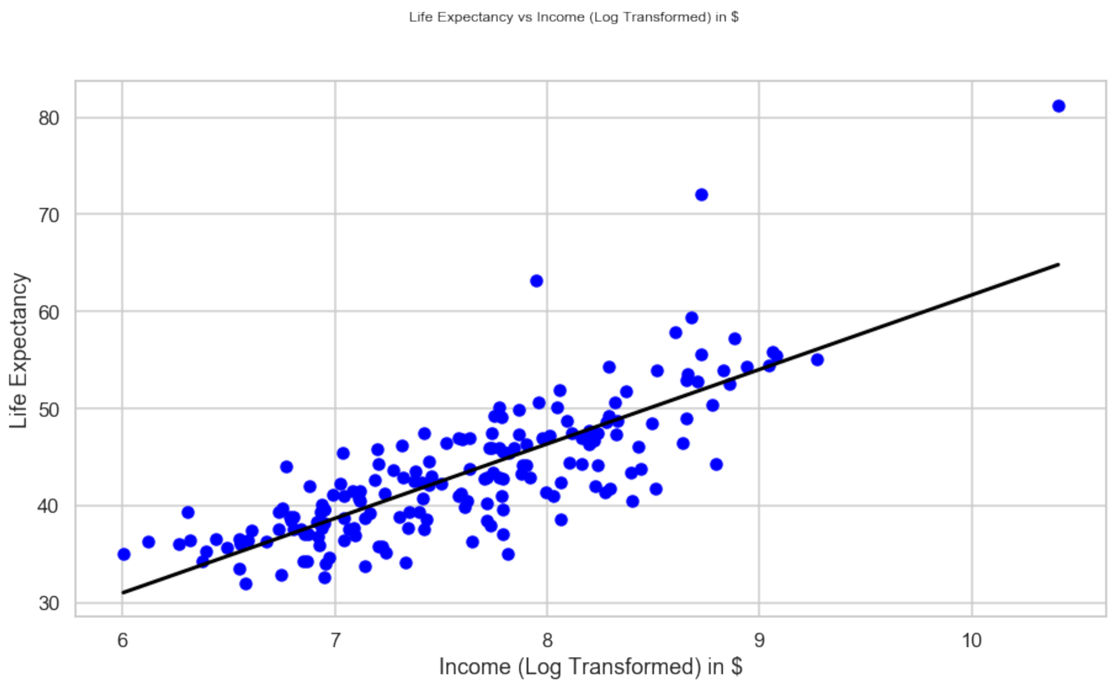
Our goal with linear regression is to minimise the vertical distance between all the data points and regression line. One of the common method is to define an error function for any given line and then use a and b to choose a line that minimizes the error function. Error functions are also called cost or loss function.

To minimize the error function, we need to draw a line that is as close to the datapoints as they can be. One way to achieve that is to minimize the vertical distance between the fit and the data. For each datapoint, this distance is called the residual. We can try to minimze the sum of the residuals, but then we could have the effect offset between a large positive residual and a large negative residual. For this reason, we will want to minimize the sum of the square distance and this becomes the lost function. This function is also called ordinary least square (ols) and it minimizes the sum of squares of the residuals. scilearn function does OLS under the hood when we call the linear regression function.

As a first step, we will create feature and target varaiables from the log transformed data and fit into a linear regression model.

```
In [27]: # Creating feature and target arrays...
         # We will use Life as target array and Population & Income as feature array...
         gm_df_country_log_X = gm_df_country_log.drop(['Country','Region','Life'], axis=1).valu
         gm_df_country_log_y = gm_df_country_log['Life'].values
         gm_df_country_log_cols = gm_df_country_log.drop(['Country','Region','Life'], axis=1).c
         gm_df_country_log_X.shape, gm_df_country_log_y.shape

         # Reshaping to get both the arrays aligned...
         gm_df_country_log_X_Income = gm_df_country_log_X[:,1]
         gm_df_country_log_y = gm_df_country_log_y.reshape(-1,1)
         gm_df_country_log_X_Income = gm_df_country_log_X_Income.reshape(-1,1)
         #gm_df_country_log_X_Income.shape, gm_df_country_log_y.shape
```

```
In [28]: # Fitting a linear regression model..

         gm_df_reg = linear_model.LinearRegression()
         gm_df_reg.fit(gm_df_country_log_X_Income, gm_df_country_log_y)

         # Let us check out the model's prediction
         gm_prediction_space = np.linspace(min(gm_df_country_log_X_Income), max(gm_df_country_l

         # Let us plot the prediction..
         plt.figure(figsize=(15,8))
         plt.scatter(gm_df_country_log_X_Income, gm_df_country_log_y, color='blue')
         plt.plot(gm_prediction_space, gm_df_reg.predict(gm_prediction_space), color='black', l

         plt.xlabel('Income (Log Transformed) in $')
         plt.ylabel('Life Expectancy')
         plt.suptitle('Life Expectancy vs Income (Log Transformed) in $')
         plt.show()
```



3.2.2 Regression Score & Root Mean Square Metrics

There are coupl eof important metrics that we calculate to measure the performance of the model. For linear regression, it would be the co-efficient of determination (how well the regressionline approximates the real data points) and Root Mean Squared Error (how far are the points from the regression line).

The score the we calculate next tells us the model performance and it is 64.1%. This value is not that great as we are trying to get a model performance close to 80% in real-life. By inclusing other features (in our case Population), we increased the performance slightly (67.63%). This is expected as we saw the correlation between Life and Population was not that strong. One way to improve model performance would be to bring other features that might be related to the Life expectancy.

```
In [29]: # Compute predictions over the prediction space: y_pred
         gm_y_pred = gm_df_reg.predict(gm_prediction_space)

         # Print R^2
         print('Score is {}'.format(gm_df_reg.score(gm_df_country_log_X_Income, gm_df_country_l
```

```
Score is 0.6407046374157207
```

```
In [30]: # Create training and test sets
         gm_X_train, gm_X_test, gm_y_train, gm_y_test = train_test_split(gm_df_country_log_X, g

         # Create the regressor: reg_all
```

```
        gm_reg_all = LinearRegression()

        # Fit the regressor to the training data
        gm_reg_all.fit(gm_X_train, gm_y_train)

        # Predict on the test data: y_pred
        gm_y_pred = gm_reg_all.predict(gm_X_test)

        # Compute and print R^2 and RMSE
        print("R^2: {}".format(gm_reg_all.score(gm_X_test, gm_y_test)))
        gm_rmse = np.sqrt(mean_squared_error(gm_y_test, gm_y_pred))
        print("Root Mean Squared Error: {}".format(gm_rmse))
```

```
R^2: 0.6763156301442532
Root Mean Squared Error: 3.83035683357173
```

3.2.3 5-fold Cross Validation

Model performance is dependent on the way the data is split between train and test. The dataset in the test might have some differences or peculiarity that was not found in the training dataset. This goes to indicate that the loss function is not representative of the model's ability to generalize. To combat this exception, we use a method called corss validation.

Cross Validation is the process where we hold out a fold or subset of data for test and then fit the model using the rest of the dataset as training data. We then predict and compute the metric from this dataset. However, we continue to hold out a fold or subset of different data as test while we fit the model with the rest of the data. We continue this approach until there is no more dataset available to fold. The more the number of folds the more the computational requirements.

It can be seen from our execution below that there are some peculiarities within the dataset. Cross Validation seems to have helped, but reduced the overall model performance even further (61.9%). At this time, we can state that this model performs poorly against the given dataset.

```
In [31]: # Now doing the same for Gap Minder dataset...
         # Import the necessary modules
         from sklearn.linear_model import LinearRegression
         from sklearn.model_selection import cross_val_score

         # Create a linear regression object: reg
         gm_reg = LinearRegression()

         # Compute 5-fold cross-validation scores: cv_scores
         gm_cv_scores = cross_val_score(gm_reg, gm_df_country_log_X, gm_df_country_log_y, cv=5

         # Print the 5-fold cross-validation scores
         print(gm_cv_scores)

         print("Average 5-Fold CV Score: {}".format(np.mean(gm_cv_scores)))
```

```
[0.67766615 0.68693916 0.44360188 0.63773023 0.65178825]
Average 5-Fold CV Score: 0.6195451350750328
```

3.2.4 Linear Regression Summary Values

In this section, we will cover some critical metrics that can be produced as output from the linear regression summary. Linear regression, also called Ordinary Least-Squares (OLS) Regression, is probably the most commonly used technique in Statistical Learning. We will focus on certain critical values in the below summary table below. Note that we first create the summary values for log transformed Income and compare it to the log transformed Population variable as they relate to Life.

The left part of the first table provides basic information about the model fit: - Dep. Variable, Which variable is the response in the model. In our case this is the Life Expectancy. - Model & Method, What model is used for fit and How the parameters of the model was calculated. It is obvious that we used Least Squares model and OLS are the method. - In addition, the observations, degrees of freedom and others are also captured.

The right part of the first table shows the goodness of fit. - R-Squared, The coefficient of determination. A statistical measure of how well the regression line approximates the real data points. The higher this number, the better. - Adjusted R-Squared accounts for additional complexity such as degrees of freedom and observations. This number is generally lower than the R-Squared value. - F-Statistic, A measure how significant the fit is. The mean squared error of the model divided by the mean squared error of the residuals.

The second table reports for each of the coefficients. - coef, The estimated value of the coefficient - std err, The basic standard error of the estimate of the coefficient. - t-statistic, The t-statistic value. This is a measure of how statistically significant the coefficient is. - P > |t|, P-value that the null-hypothesis that the coefficient = 0 is true. If it is less than the confidence level, often 0.05, it indicates that there is a statistically significant relationship between the term and the response.

In the next two cells, we calculate the OLS summary for: - Log transformed Income feature and - Log transformed Population feature.

It can be seen that the Income feature is much more significant than the Population feature. The overall performance of the model is 0.641 (64.1%) for Income, but only 0.004 (0.4%) for the Population variable. To help better this performance, we will need additional features that correlates with Life expectancy (such as child mortality, fertility rate etc). The p-value indicates if the null-hypotheis is true. In our case, the null-hypothesis was that the Income variable is not related to the Life. P-Values close to 0, indicate we reject the null hypothesis and confirm that there is a relation between Income and Life variables. In the case of Population, we can see that the P-Value is high and so we fail to reject the null-hypothesis that the population is not related to the Life expectancy.

```
In [32]: from statsmodels.formula.api  import ols
         #you need a Pandas dataframe df with columns labeled Y, X, & X2
         est_Income = ols(formula = 'Life ~ log_Income', data = gm_df_country_log).fit()
         est_Income.summary()

Out[32]: <class 'statsmodels.iolib.summary.Summary'>
         """
                                     OLS Regression Results
```

```
==============================================================================
Dep. Variable:                   Life   R-squared:                       0.641
Model:                            OLS   Adj. R-squared:                  0.639
Method:                 Least Squares   F-statistic:                     322.8
Date:                Wed, 24 Apr 2019   Prob (F-statistic):           4.33e-42
Time:                        16:38:40   Log-Likelihood:                -522.50
No. Observations:                 183   AIC:                             1049.
Df Residuals:                     181   BIC:                             1055.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    -14.9955      3.261     -4.599      0.000     -21.429      -8.562
log_Income     7.6700      0.427     17.966      0.000       6.828       8.512
==============================================================================
Omnibus:                       46.018   Durbin-Watson:                   2.110
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              155.157
Skew:                           0.954   Prob(JB):                     2.03e-34
Kurtosis:                       7.088   Cond. No.                         81.0
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
"""
```

```
In [33]: from statsmodels.formula.api  import ols
         #you need a Pandas dataframe df with columns labeled Y, X, & X2
         est_Population = ols(formula = 'Life ~ log_Population', data = gm_df_country_log).fit
         est_Population.summary()
```

```
Out[33]: <class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Life   R-squared:                       0.004
Model:                            OLS   Adj. R-squared:                 -0.002
Method:                 Least Squares   F-statistic:                    0.6895
Date:                Wed, 24 Apr 2019   Prob (F-statistic):              0.407
Time:                        16:38:40   Log-Likelihood:                -615.81
No. Observations:                 183   AIC:                             1236.
Df Residuals:                     181   BIC:                             1242.
Df Model:                           1
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercept         46.3877      3.736     12.416      0.000      39.015      53.760
```

```
log_Population    -0.2166      0.261     -0.830      0.407      -0.731       0.298
==============================================================================
Omnibus:                       67.237   Durbin-Watson:                   1.911
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              213.466
Skew:                           1.484   Prob(JB):                     4.43e-47
Kurtosis:                       7.381   Cond. No.                         103.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
"""
```

3.2.5 Linear Regression Diagnostics

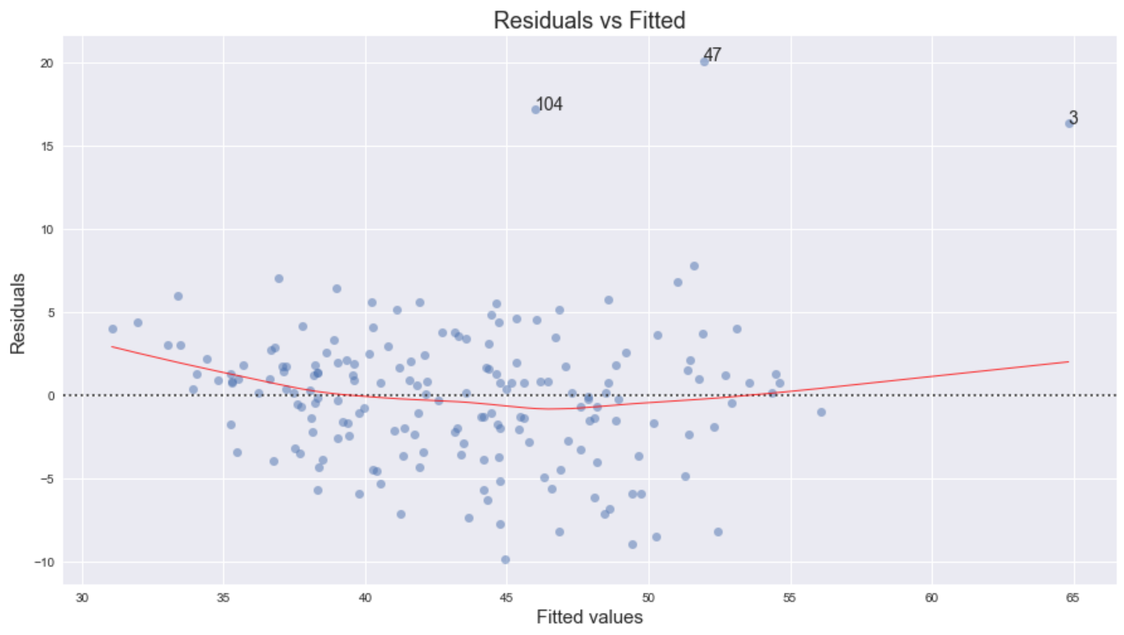
Residuals vs Fitted Plot This plot shows if residuals have non-linear patterns. There could be a nonlinear relationship between predictor variables and an outcome variable and the pattern could show up in this plot if the model doesn't capture the nonlinear relationship. If you find equally spread residuals around a horizontal line without distinct patterns, that is a good indication you don't have non-linear relationships.

The plot indicates that there is non-linear relationship between the predictor (Income) variable and outcome (Life) variable. The assumption of linearity is violated, as we can see that the residuals are not equally spread around the horizontal line. The pattern seems to be slightly curved as traced by the red line.

```
In [34]: %matplotlib inline

         import numpy as np
         import pandas as pd
         import seaborn as sns
         import matplotlib.pyplot as plt
         import statsmodels.formula.api as smf

         from statsmodels.graphics.gofplots import ProbPlot

         plt.style.use('seaborn') # pretty matplotlib plots

         plt.rc('font', size=14)
         plt.rc('figure', titlesize=18)
         plt.rc('axes', labelsize=15)
         plt.rc('axes', titlesize=18)

In [35]: # fitted values (need a constant term for intercept)
         model_fitted_y = est_Income.fittedvalues

         # model residuals
         model_residuals = est_Income.resid

         # normalized residuals
```

```
model_norm_residuals = est_Income.get_influence().resid_studentized_internal

# absolute squared normalized residuals
model_norm_residuals_abs_sqrt = np.sqrt(np.abs(model_norm_residuals))

# absolute residuals
model_abs_resid = np.abs(model_residuals)

# leverage, from statsmodels internals
model_leverage = est_Income.get_influence().hat_matrix_diag

# cook's distance, from statsmodels internals
model_cooks = est_Income.get_influence().cooks_distance[0]
```

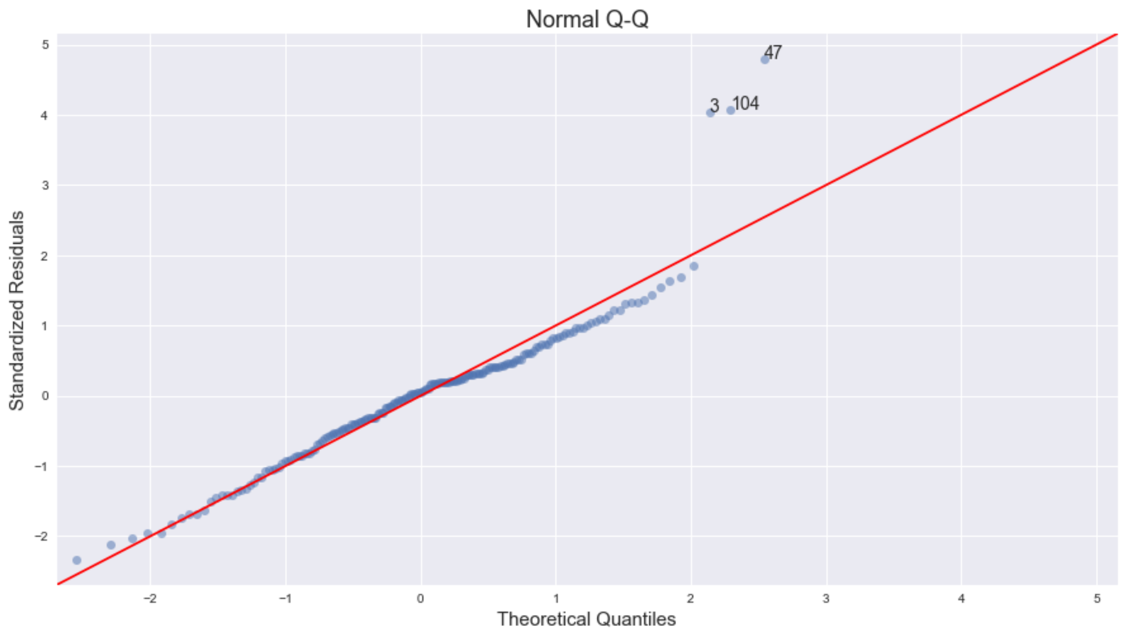
```
In [36]: plot_lm_1 = plt.figure(1)
         plot_lm_1.set_figheight(8)
         plot_lm_1.set_figwidth(15)

         plot_lm_1.axes[0] = sns.residplot(model_fitted_y, 'Life', data=gm_df_country_log,
                                   lowess=True,
                                   scatter_kws={'alpha': 0.5},
                                   line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})

         plot_lm_1.axes[0].set_title('Residuals vs Fitted')
         plot_lm_1.axes[0].set_xlabel('Fitted values')
         plot_lm_1.axes[0].set_ylabel('Residuals')

         # annotations
         abs_resid = model_abs_resid.sort_values(ascending=False)
         abs_resid_top_3 = abs_resid[:3]

         for i in abs_resid_top_3.index:
             plot_lm_1.axes[0].annotate(i,
                                        xy=(model_fitted_y[i],
                                            model_residuals[i]));
```



Normal Q-Q Plot This plot shows if residuals are normally distributed. Do residuals follow a straight line well or do they deviate severely? It's good if residuals are lined well on the straight dashed line.

The Q-Q Plot shows slight deviations from the straight line as drawn by the red line. This indicates that while the log transformation helped, the residuals are not completely normally distributed. We can expect this deviation to also affect the model performance.

```
In [37]: QQ = ProbPlot(model_norm_residuals)
         plot_lm_2 = QQ.qqplot(line='45', alpha=0.5, color='#4C72B0', lw=1)

         plot_lm_2.set_figheight(8)
         plot_lm_2.set_figwidth(15)

         plot_lm_2.axes[0].set_title('Normal Q-Q')
         plot_lm_2.axes[0].set_xlabel('Theoretical Quantiles')
         plot_lm_2.axes[0].set_ylabel('Standardized Residuals');

         # annotations
         abs_norm_resid = np.flip(np.argsort(np.abs(model_norm_residuals)), 0)
         abs_norm_resid_top_3 = abs_norm_resid[:3]

         for r, i in enumerate(abs_norm_resid_top_3):
             plot_lm_2.axes[0].annotate(i,
                                        xy=(np.flip(QQ.theoretical_quantiles, 0)[r],
                                            model_norm_residuals[i]));
```

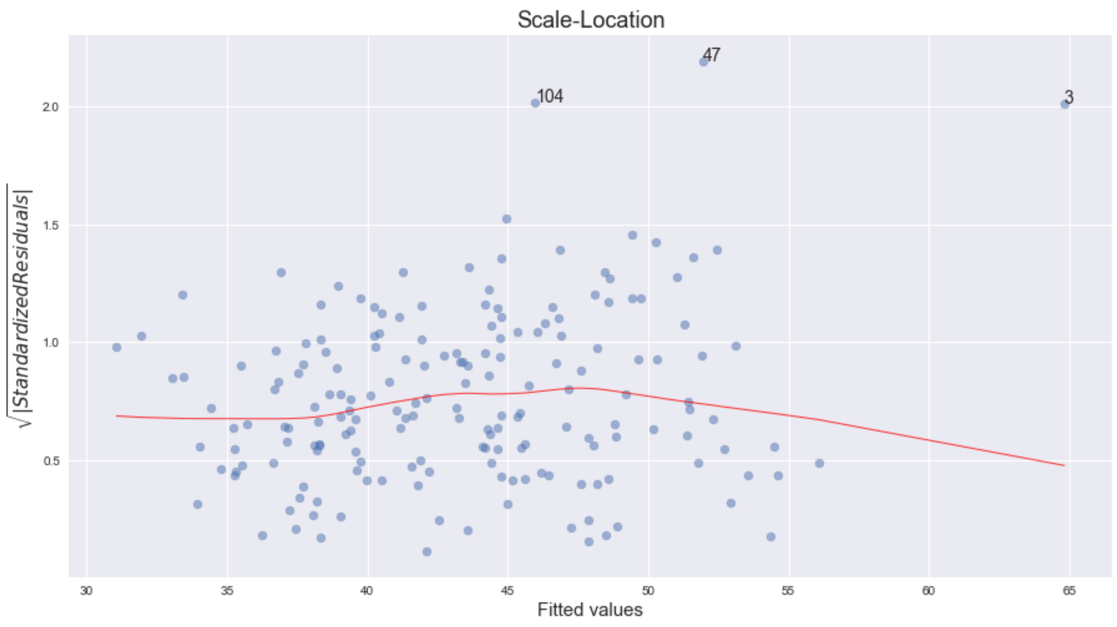


Scale - Location This plot shows if residuals are spread equally along the ranges of predictors. This is how you can check the assumption of equal variance (homoscedasticity). It's good if you see a horizontal line with equally (randomly) spread points.

In the plot below, it can be noticed that as we get to teh right, the residuals begin to spread wider around 55. In fact, after X-axis value of 50, we can very clearly see that there are residuals below the line, but not very many above the red line. The assumption of equal variance (homoscedasticity) is violated in our case.

```
In [38]: plot_lm_3 = plt.figure(3)
         plot_lm_3.set_figheight(8)
         plot_lm_3.set_figwidth(15)

         plt.scatter(model_fitted_y, model_norm_residuals_abs_sqrt, alpha=0.5)
         sns.regplot(model_fitted_y, model_norm_residuals_abs_sqrt,
                     scatter=False,
                     ci=False,
                     lowess=True,
                     line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})

         plot_lm_3.axes[0].set_title('Scale-Location')
         plot_lm_3.axes[0].set_xlabel('Fitted values')
         plot_lm_3.axes[0].set_ylabel('$\sqrt{|Standardized Residuals|}$');

         # annotations
         abs_sq_norm_resid = np.flip(np.argsort(model_norm_residuals_abs_sqrt), 0)
         abs_sq_norm_resid_top_3 = abs_sq_norm_resid[:3]
```

```
for i in abs_norm_resid_top_3:
    plot_lm_3.axes[0].annotate(i,
                               xy=(model_fitted_y[i],
                                   model_norm_residuals_abs_sqrt[i]));
```

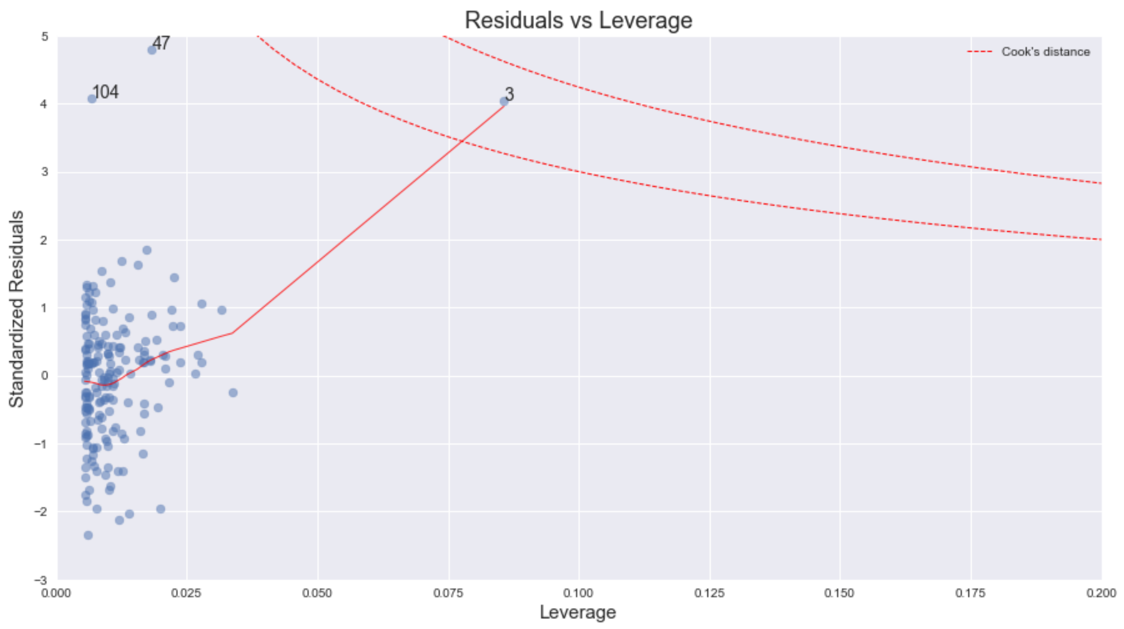


Residuals vs Leverage This plot helps us to find influential cases (i.e., subjects) if any. This plot indicates if there are any observations or cases that are influential and can help with improving the model performance by eliminating these observations. Another way to put it is that they don't get along with the trend in the majority of the cases.

We watch out for outlying values at the upper right corner or at the lower right corner. Those spots are the places where cases can be influential against a regression line. Look for cases outside of a dashed line, Cook's distance. When cases are outside of the Cook's distance (meaning they have high Cook's distance scores), the cases are influential to the regression results. The regression results will be altered if we exclude those cases.

In the plot below, we can see that observation case 3 is the most influential. However, even this observation falls within the cook's distance range and hence will not affect the model performance drastically. This indicates that eliminating this observation might give very little improvement, but not significant enough to try to recompute the regression line.

```
In [39]: plot_lm_4 = plt.figure(4)
         plot_lm_4.set_figheight(8)
         plot_lm_4.set_figwidth(15)

         plt.scatter(model_leverage, model_norm_residuals, alpha=0.5)
         sns.regplot(model_leverage, model_norm_residuals,
                     scatter=False,
                     ci=False,
                     lowess=True,
                     line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})

         plot_lm_4.axes[0].set_xlim(0, 0.20)
         plot_lm_4.axes[0].set_ylim(-3, 5)
         plot_lm_4.axes[0].set_title('Residuals vs Leverage')
         plot_lm_4.axes[0].set_xlabel('Leverage')
         plot_lm_4.axes[0].set_ylabel('Standardized Residuals')

         # annotations
         leverage_top_3 = np.flip(np.argsort(model_cooks), 0)[:3]

         for i in leverage_top_3:
             plot_lm_4.axes[0].annotate(i,
                                        xy=(model_leverage[i],
                                            model_norm_residuals[i]))

         # shenanigans for cook's distance contours
         def graph(formula, x_range, label=None):
             x = x_range
             y = formula(x)
             plt.plot(x, y, label=label, lw=1, ls='--', color='red')

         p = len(est_Income.params) # number of model parameters

         graph(lambda x: np.sqrt((0.5 * p * (1 - x)) / x),
               np.linspace(0.001, 0.200, 50),
               'Cook\'s distance') # 0.5 line
         graph(lambda x: np.sqrt((1 * p * (1 - x)) / x),
               np.linspace(0.001, 0.200, 50)) # 1 line
         plt.legend(loc='upper right');
```



3.2.6 Summary of Supervised Learning

As part of supervised learning models, we chose Regression analysis. The primary reason for this is the fact that the target and predictor features were quantitative. As we pre-processed data, it was observed that the Income and Population predictor variables did not follow normal distribution. As part of centeirng and scaling, we tried standardization, but that did not help much. We observed that log transforming the predictor variables helped in getting closer to normal distribution. We continued our further analysis on log transformed predictor variables.

We chose to use a linear regression model to fit and predict from our dataset. One of the first steps was to draw the scatter plot between Life and log transformed Income variables to observe the randomness spread of the data points. It can be seen that the datapoints below the line was more spread out than the ones above.

We continued to perform more linear regression diagnostics: - Resduals vs Fitted plot was used to understand if there were any non-linear patterns. In our case, we sure did have non-linear patterns. Th eplot did not indicate anything as drastic as a parabola (quadratic), but it can surely be see that the residuals did not follow a linear pattern as well. - Normal Q-Q Plot was used to validate the normality of the residual distribution. It was clearly observed that not all of the data points followed a straight line, indicating that we might have poor performance assuming normal distribution. - The spread-location plot was used to validate the assumption of equal variance (homoscedasticity). It was clear from the plot that the data invilidated our assumption of equal variance. This will also cause a decrease in our model performance. - Residuals vs Leverage plot did not indicate any observations (or cases) to unduly influence the regression line. Hence, we did not drop any observations to re-calculate the regression line.

As we fit and trained the model for prediction, we could see that the Regression Summary values indicated that log transformed Income variable was more correlated (and influential) than the log transformed Population predictor variable. This was confirmed by observing the P-value metric of the summary values. It did confirm that the test is significant and we reject the null hypothesis that there is no relationship between Income and Life variables. Hence, we confirm that there is a positive linear relationship between log transformed Income and Life variables.

The Income predictor was able to provide a 64.1% predictor score, which is considered poor performance. By including the Population variable as well, we were able to improve the predictor score to 67%. Cross Validation did not help in improving the performance any further.

Since, many of the diagnostic assumptions were not strongly validated, we can continue the analysis with other non-linear regression models. In addition, bringing in additional predictor variables might help improve the model performance as it is clear that Income by itself is not the only predictor of Life Expectancy.

3.3 Unsupervised Learning

Unsupervised machine learning algorithms infer patterns from a dataset without reference to known, or labeled, outcomes. Unlike supervised machine learning, unsupervised machine learning methods cannot be directly applied to a regression or a classification problem because you have no idea what the values for the output data might be, making it impossible for you to train the algorithm the way you normally would. Unsupervised learning can instead be used for discovering the underlying structure of the data.

Note that there are many ways to perform unsupervised learning such as clustering, association mining, support vector machines etc. We will focus on clustring technique in this report.

3.4 K-Means Clustering

Clustering allows you to automatically split the dataset into groups according to similarity. Often, however, cluster analysis overestimates the similarity between groups and doesn't treat data points as individuals. Clustering also works very well for lower dimensions (single predictor features to target variables) and get complex very quickly for higher dimensions. K-Means clustering is a method where we pre-define the number of clusters that we want the machine learning algorithm to partition the dataset.

In unsupervised learning techniques the diagnostic assumptions of regression models is not relevant and the primary goal is to learn from the data to group similar observations. However, centering and scaling is an important consideration as the clustering is based on the distance (eucledian or manhattan) between the datapoints. Hence, we will standardize the dataset we have and use the standardized data to perform clustering.

3.4.1 Validate Pre-Processed Data

For supervised learning, we will continue to work with the following three datasets: - gm_df_country, the dataset of non transformed Income & Population variables grouped by Country - gm_df_country_log, the dataset of log transformed Income & Population variables grouped by Country - gm_df_country_std, the dataset of Scaled/Standardized Income & Population variables grouped by Country

Since the pre-processing was already completed as part of the requirement for Regression model, we will not redo any of the pre-processing steps.

3.4.2 Center & Scaling

In the code below, we will create the standardized (scaled) values for Population and Income. We will now have an opportunity to understand the model performance for non transformed, log transformed and scaled datasets in the subsequent sections.

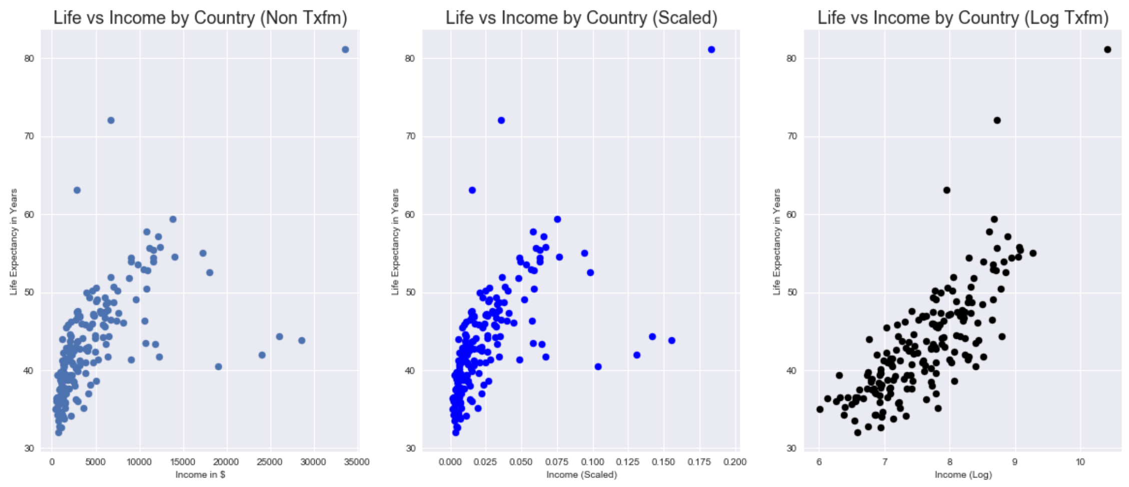
```
In [40]: # Turn off Data Conversion Warnings...
         from sklearn.exceptions import DataConversionWarning
         warnings.filterwarnings(action='ignore', category=DataConversionWarning)

         # Create scaled / standardized vaues for Population & Income
         scaler = MinMaxScaler()

         gm_df_stndrd = gm_df_upd.copy(deep=True)

         gm_df_stndrd[['Population', 'Income']] = scaler.fit_transform(gm_df_stndrd[['Populati

         #Rename Population and Income columns to reflect the correct labels...
         gm_df_stndrd.columns = ['Country','Year','Life','std_Population', 'std_Income','Regior

         # Group by Year..so we have average values by Country and region between 1800 and 201
         gm_df_country_std = (gm_df_stndrd.groupby(['Country', 'Region'], as_index=False).mean
                              [['Country', 'Region', 'Life', 'std_Population', 'std_Income']])

         # get some details of the dataframe..
         gm_df_country_std.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 183 entries, 0 to 182
Data columns (total 5 columns):
Country           183 non-null object
Region            183 non-null object
Life              183 non-null float64
std_Population    183 non-null float64
std_Income        183 non-null float64
dtypes: float64(3), object(2)
memory usage: 8.6+ KB
```

Now that we have the standardized data (scaled), let us do a scatter plot of the non transformed, log transformed and standardized datasets. It can be seen that the scatter pattern of non transformed and standardized datasets are exactly the same (except for the scale of the x-axis). For log transformed dataset, the pattern itself has changed, indicating a change to the distribution of data.

```
In [41]: #Let us do a scatter plots of the three datasets...
         # All of the datasets will be plotted between Life and Income variables...

         fig, ax = plt.subplots(ncols=3, figsize=(20,8))
         gm_df_country.plot.scatter(x='Income', y='Life', s=50, ax=ax[0])
         gm_df_country_std.plot.scatter(x='std_Income', y='Life', c='blue', s=50, ax=ax[1])
         gm_df_country_log.plot.scatter(x='log_Income', y='Life', c='black', s=50, ax=ax[2])
```

```
ax[0].set_ylabel('Life Expectancy in Years', fontsize=10)
ax[0].set_xlabel('Income in $', fontsize=10)
ax[0].set_title('Life vs Income by Country (Non Txfm)')

ax[1].set_ylabel('Life Expectancy in Years', fontsize=10)
ax[1].set_xlabel('Income (Scaled)', fontsize=10)
ax[1].set_title('Life vs Income by Country (Scaled)')

ax[2].set_ylabel('Life Expectancy in Years', fontsize=10)
ax[2].set_xlabel('Income (Log)', fontsize=10)
ax[2].set_title('Life vs Income by Country (Log Txfm)')

plt.show()
```



3.4.3 Predictor & Target Variables

The next step would be to idenitfy the predictor and target variables. Based on our question, we already know that the Life variable is our target and Income is the predictor variable. We will now setup these variables for each of the three datasets (Non Txfm, Scale & Log Txfm).

```
In [42]: # Make results reproducible
         np.random.seed(1234567890)

         gm_df_clust = gm_df_country
         #variables = ['Life','Income', 'Population']
         variables = ['Life','Income']
         gm_df_clust_log = gm_df_country_log
         #variables_log = ['Life','log_Income', 'log_Population']
         variables_log = ['Life','log_Income']
         gm_df_clust_std = gm_df_country_std
```

```
#variables_std = ['Life','std_Income', 'std_Population']
variables_std = ['Life','std_Income']


# convert to numeric format
for variable in variables:
    gm_df_clust[variable] = pd.to_numeric(gm_df_clust[variable], errors='coerce')

for variable in variables_log:
    gm_df_clust_log[variable] = pd.to_numeric(gm_df_clust_log[variable], errors='coer

for variable in variables_std:
    gm_df_clust_std[variable] = pd.to_numeric(gm_df_clust_std[variable], errors='coer


# listwise deletion of missing values
subset = gm_df_clust[variables].dropna()
subset_log = gm_df_clust_log[variables_log].dropna()
subset_std = gm_df_clust_std[variables_std].dropna()

# Print the rows and columns of the data frame
print('Size of study data')
print(subset.shape, subset_log.shape, subset_std.shape)
print("\n")

# Remove the first variable from the list since the target is derived from it
variables.pop(0)
variables_log.pop(0)
variables_std.pop(0)

# Center and scale data
#for variable in variables:
#    subset[variable]=preprocessing.scale(subset[variable].astype('float64'))

# Generate features and target variables
features = subset[variables]
targets = subset[['Life']]

features_log = subset_log[variables_log]
targets_log = subset_log[['Life']]

features_std = subset_std[variables_std]
targets_std = subset_std[['Life']]
```

```
Size of study data
(183, 2) (183, 2) (183, 2)
```

3.4.4 Train & Test datasets

For all unsupervised learning methods, we will need to split the dataset into training and test data. This is required as the machine learning algorithm solely depends on the dataset and we will need the test data for the ability to predict, measure and evaluate model performance.

```
In [43]: """
         " ==================  Split Data into Training and Test Sets  ==================
         """
         # Split into training and testing sets
         # We will split 70 - 30 between training and test data
         training_data, test_data, training_target, test_target  = train_test_split(features, 
         training_data_log, test_data_log, training_target_log, test_target_log  = (
             train_test_split(features_log, targets_log, test_size=.3))
         training_data_std, test_data_std, training_target_std, test_target_std  = (
             train_test_split(features_std, targets_std, test_size=.3))
         print('Size of training data')
         print(training_data.shape, training_data_log.shape, training_data_std.shape)
```

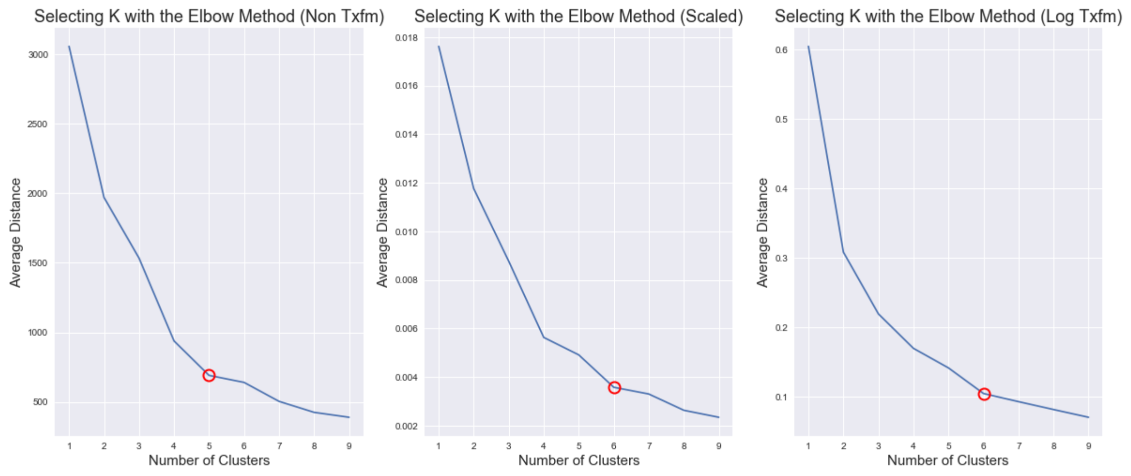
```
Size of training data
(128, 1) (128, 1) (128, 1)
```

3.4.5 Choosing K (Cluster Groups)

We will use Elbow method to choose the optimal value of k (The number of clusters). Note that the value of K can change based on non transformed, scaled and log transformed datasets.

```
In [44]: """
         " =====================  Determine the Number of Clusters  ====================
         """
         # Identify number of clusters using the elbow method
         clusters=range(1,10)
         meandist=[]
         meandist_log=[]
         meandist_std=[]

         for k in clusters:
             model=KMeans(n_clusters=k)
             model_log=KMeans(n_clusters=k)
             model_std=KMeans(n_clusters=k)

             model.fit(training_data)
             model_log.fit(training_data_log)
             model_std.fit(training_data_std)

             clusassign=model.predict(training_data)
```

```
    clusassign_log=model_log.predict(training_data_log)
    clusassign_std=model_std.predict(training_data_std)

    meandist.append(sum(np.min(cdist(training_data, model.cluster_centers_, 'euclidean
    meandist_log.append(sum(np.min(cdist(training_data_log, model_log.cluster_centers_
    meandist_std.append(sum(np.min(cdist(training_data_std, model_std.cluster_centers_


k = 5
k_log = 6
k_std = 6

# Visualize the elbow

fig, ax = plt.subplots(ncols=3, figsize=(20,8))

ax[0].plot(clusters, meandist)
ax[0].plot(clusters[(k-1)], meandist[(k-1)], marker='o', markersize=12,
    markeredgewidth=2, markeredgecolor='r', markerfacecolor='None')
ax[0].grid(True)
ax[0].set_xlabel('Number of Clusters')
ax[0].set_ylabel('Average Distance')
ax[0].set_title('Selecting K with the Elbow Method (Non Txfm)')

ax[1].plot(clusters, meandist_std)
ax[1].plot(clusters[(k_std-1)], meandist_std[(k_std-1)], marker='o', markersize=12,
    markeredgewidth=2, markeredgecolor='r', markerfacecolor='None')
ax[1].grid(True)
ax[1].set_xlabel('Number of Clusters')
ax[1].set_ylabel('Average Distance')
ax[1].set_title('Selecting K with the Elbow Method (Scaled)')

ax[2].plot(clusters, meandist_log)
ax[2].plot(clusters[(k_log-1)], meandist_log[(k_log-1)], marker='o', markersize=12,
    markeredgewidth=2, markeredgecolor='r', markerfacecolor='None')
ax[2].grid(True)
ax[2].set_xlabel('Number of Clusters')
ax[2].set_ylabel('Average Distance')
ax[2].set_title('Selecting K with the Elbow Method (Log Txfm)')

plt.show()
```

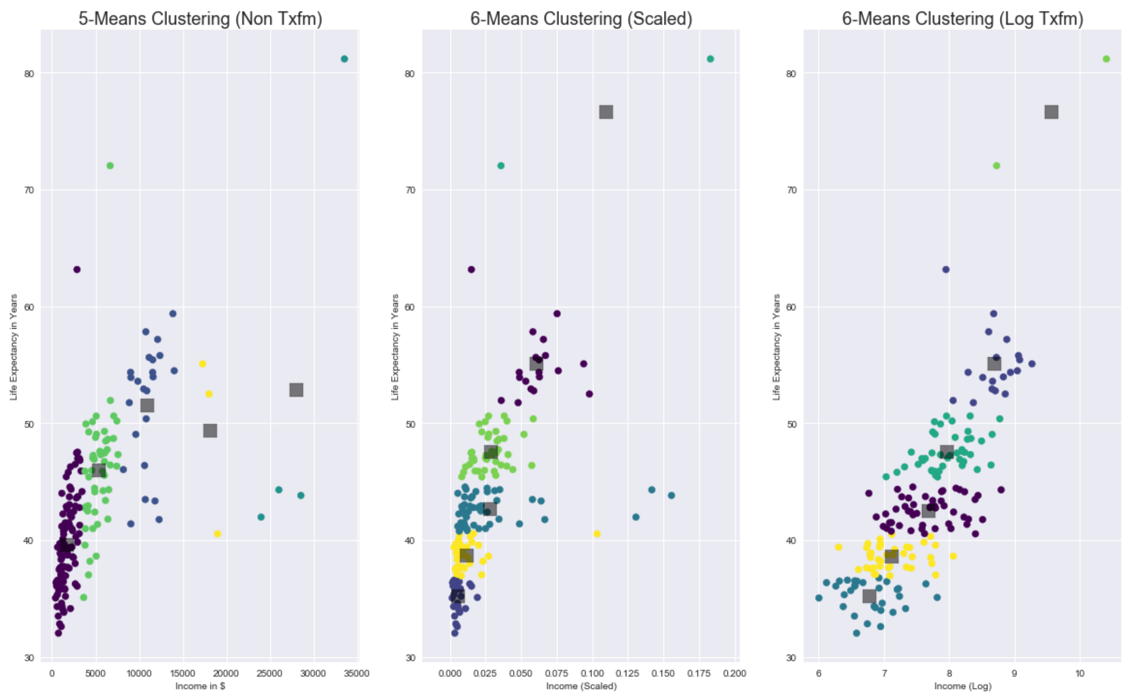


3.4.6 Cluster Visualization & Analysis

We will now visualize the scatter plots color coded by the cluster groupings. Since the cluster and optimizations are different for the three datasets, we will visualize all of them side by side. It is interesting to observe the influence of bias in the first plot (Non Txfm), while both the Scaled and Log Transformtion clustered the datapoints very similarly, even though the centriods for each of these plots are different.

```
In [45]: # Make results reproducible
         np.random.seed(1234567890)

         # Generate features and target variables
         f1 = gm_df_country['Income'].values
         f2 = gm_df_country['Life'].values
         X = np.matrix(list(zip(f1,f2)))

         f1_std = gm_df_country_std['std_Income'].values
         f2_std = gm_df_country_std['Life'].values
         X_std = np.matrix(list(zip(f1_std,f2_std)))

         f1_log = gm_df_country_log['log_Income'].values
         f2_log = gm_df_country_log['Life'].values
         X_log = np.matrix(list(zip(f1_log,f2_log)))

         # Convert to numpy array
         X = np.array(X)
         X_std = np.array(X_std)
         X_log = np.array(X_log)


         #Perform KMeans Algorithm with the optimal clusters identified using elbow method
```

```
model1=KMeans(n_clusters=k)
model1_std=KMeans(n_clusters=k_std)
model1_log=KMeans(n_clusters=k_log)

model1.fit(X)
model1_std.fit(X_std)
model1_log.fit(X_log)

clusassign=model1.predict(X)
clusassign_std=model1_std.predict(X_std)
clusassign_log=model1_log.predict(X_log)

# Convert the clusassign variable to numpy array
clusassign = np.array(clusassign)
clusassign_log = np.array(clusassign_log)
clusassign_std = np.array(clusassign_std)

# Identify the centroids for each of the plot
centers = model1.cluster_centers_
centers_std = model1_std.cluster_centers_
centers_log = model1_log.cluster_centers_

# Plot the scatter plots..
fig, ax = plt.subplots(ncols=3, figsize=(20,12))

ax[0].scatter(X[:, 0], X[:, 1], c=clusassign, s=50, cmap='viridis')
ax[1].scatter(X_std[:, 0], X_std[:, 1], c=clusassign_std, s=50, cmap='viridis')
ax[2].scatter(X_log[:, 0], X_log[:, 1], c=clusassign_log, s=50, cmap='viridis')

ax[0].set_ylabel('Life Expectancy in Years', fontsize=10)
ax[0].set_xlabel('Income in $', fontsize=10)
ax[0].set_title('5-Means Clustering (Non Txfm)')
ax[0].scatter(centers[:, 0], centers[:, 1], c='black', marker='s', s=200, alpha=0.5);

ax[1].set_ylabel('Life Expectancy in Years', fontsize=10)
ax[1].set_xlabel('Income (Scaled)', fontsize=10)
ax[1].set_title('6-Means Clustering (Scaled)')
ax[1].scatter(centers_std[:, 0], centers_std[:, 1], c='black', marker='s', s=200, alpl

ax[2].set_ylabel('Life Expectancy in Years', fontsize=10)
ax[2].set_xlabel('Income (Log)', fontsize=10)
ax[2].set_title('6-Means Clustering (Log Txfm)')
ax[2].scatter(centers_log[:, 0], centers_log[:, 1], c='black', marker='s', s=200, alpl

plt.show()
```



3.4.7 Model Metrics & Performance

As canbe seen from the plots and dataset, it is clear that Life target variable might not be the best variable for K-Means clustering model. The reason for this is the datapoints are too close to each other to establish any clear clustering grouping. The Ordinary Least Scores (Summary value) indicates the clustering model scores (R-Squared Values) that can be used as an indicator for the performance of the model. - The R-Squared value for Non transformed dataset is 0.534. This is achieved when k = 8 clusters. As can be seen from the Elbow plot, K = 5 can be considered where the elbow turn occurs. - R-Squared value for Scaled dataset is 0.647 at 8 cluster groupings. This is a slight improvement in the model performance, though the elbow plot seem to indicate 6 cluster groupings to be optimal. - R-Squared value for log transformed data is 0.658 at 8 cluster configurations. This dataset seems to offer the best model performance of the three datasets. This is close to 10% improvement from non transformed data. The elbow plot seem to indicate that optimal k value is 6.

```
In [46]: """
         " ====================  Examine Differences Between Clusters  =================
         """
         print('')
         print('Summary Values for Non Txfm dataset...')
         print('')
         training_target['cluster'] = model.labels_
         income_model = smf.ols(formula='Life ~ C(cluster)', data=training_target).fit()
         print (income_model.summary())
```

```
        print('')
        print('Summary Values for Scaled dataset...')
        print('')
        training_target_std['cluster'] = model_std.labels_
        income_model_std = smf.ols(formula='Life ~ C(cluster)', data=training_target_std).fit
        print (income_model_std.summary())

        print('')
        print('Summary Values for Log Transformed dataset...')
        print('')
        training_target_log['cluster'] = model_log.labels_
        income_model_log = smf.ols(formula='Life ~ C(cluster)', data=training_target_log).fit
        print (income_model_log.summary())
```

```
Summary Values for Non Txfm dataset...

                            OLS Regression Results
==============================================================================
Dep. Variable:                   Life   R-squared:                       0.534
Model:                            OLS   Adj. R-squared:                  0.503
Method:                 Least Squares   F-statistic:                     17.04
Date:                Wed, 24 Apr 2019   Prob (F-statistic):           1.15e-16
Time:                        16:38:44   Log-Likelihood:                -374.44
No. Observations:                 128   AIC:                             766.9
Df Residuals:                     119   BIC:                             792.5
Df Model:                           8
Covariance Type:            nonrobust
=====================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
Intercept          37.7742      0.731     51.708      0.000      36.328      39.221
C(cluster)[T.1]    13.4839      1.722      7.831      0.000      10.074      16.893
C(cluster)[T.2]     6.1909      1.219      5.080      0.000       3.778       8.604
C(cluster)[T.3]     6.0593      4.734      1.280      0.203      -3.315      15.434
C(cluster)[T.4]    11.1200      1.323      8.408      0.000       8.501      13.739
C(cluster)[T.5]     5.4334      1.159      4.687      0.000       3.138       7.729
C(cluster)[T.6]    14.7405      4.734      3.113      0.002       5.366      24.115
C(cluster)[T.7]     4.2062      4.734      0.888      0.376      -5.168      13.581
C(cluster)[T.8]    14.0310      1.913      7.335      0.000      10.243      17.819
==============================================================================
Omnibus:                       54.672   Durbin-Watson:                   2.159
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              280.469
Skew:                           1.368   Prob(JB):                     1.25e-61
Kurtosis:                       9.716   Cond. No.                         12.4
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Summary Values for Scaled dataset...

                            OLS Regression Results
==============================================================================
Dep. Variable:                   Life   R-squared:                       0.647
Model:                            OLS   Adj. R-squared:                  0.623
Method:                 Least Squares   F-statistic:                     27.22
Date:                Wed, 24 Apr 2019   Prob (F-statistic):           1.41e-23
Time:                        16:38:44   Log-Likelihood:                -370.98
No. Observations:                 128   AIC:                             760.0
Df Residuals:                     119   BIC:                             785.6
Df Model:                           8
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
Intercept          43.6434      0.860     50.723      0.000      41.940      45.347
C(cluster)[T.1]     8.5599      1.677      5.103      0.000       5.239      11.881
C(cluster)[T.2]    -1.6631      4.634     -0.359      0.720     -10.838       7.512
C(cluster)[T.3]     5.2916      1.427      3.709      0.000       2.466       8.117
C(cluster)[T.4]    37.5392      4.634      8.102      0.000      28.364      46.714
C(cluster)[T.5]    -5.8904      1.116     -5.277      0.000      -8.101      -3.680
C(cluster)[T.6]    10.1620      3.332      3.049      0.003       3.563      16.761
C(cluster)[T.7]     6.1437      1.825      3.366      0.001       2.530       9.758
C(cluster)[T.8]     0.0246      1.314      0.019      0.985      -2.578       2.627
==============================================================================
Omnibus:                       62.003   Durbin-Watson:                   2.086
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              335.349
Skew:                           1.578   Prob(JB):                     1.51e-73
Kurtosis:                      10.274   Cond. No.                         12.8
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Summary Values for Log Transformed dataset...

                            OLS Regression Results
==============================================================================
Dep. Variable:                   Life   R-squared:                       0.658
Model:                            OLS   Adj. R-squared:                  0.635
Method:                 Least Squares   F-statistic:                     28.56
Date:                Wed, 24 Apr 2019   Prob (F-statistic):           2.29e-24
Time:                        16:38:44   Log-Likelihood:                -353.63
No. Observations:                 128   AIC:                             725.3
```

```
Df Residuals:                     119   BIC:                             750.9
Df Model:                           8
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercept         39.7035      1.027     38.676      0.000      37.671      41.736
C(cluster)[T.1]    6.3155      1.578      4.002      0.000       3.190       9.441
C(cluster)[T.2]   14.7939      1.540      9.607      0.000      11.745      17.843
C(cluster)[T.3]   -1.9705      1.344     -1.466      0.145      -4.632       0.691
C(cluster)[T.4]   -3.0170      2.053     -1.469      0.144      -7.082       1.048
C(cluster)[T.5]    7.8111      1.373      5.688      0.000       5.092      10.530
C(cluster)[T.6]    1.4197      1.477      0.961      0.339      -1.506       4.345
C(cluster)[T.7]    3.5167      1.344      2.616      0.010       0.855       6.178
C(cluster)[T.8]   -4.1228      1.623     -2.540      0.012      -7.337      -0.909
==============================================================================
Omnibus:                       16.714   Durbin-Watson:                   1.987
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               37.599
Skew:                           0.483   Prob(JB):                     6.85e-09
Kurtosis:                       5.474   Cond. No.                         9.86
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

3.4.8 Summary of Unsupervised Learning

Based on what we have seen so far for unsupervised learning, we can see that gap minder target variable Life is not a great candidate for clustering. Having said that, log transformed data seem to be offering better model performance with an optimal cluster value of 5 or 7. From the scatter plot is can be clearly see that there is a positive linear relationship between Life and Income variables.

4 Conclusion

We have completed walking through a complete data analysis cycle for the gap minder dataset. We started with stating a question and then follow-through with the epicycle analysis in trying to refine the question even further. The refined question (and hypothesis) was to better understand the relationship between Life Expectancy (Target) and Income (predictor) variables.

Our intuition suggested that both Income and Population will have a positive relationship with Life.

While performing data wrangling and pre-processing, we observed that there were missing data, data with inconsistent format and needed different approaches to impute the data. Finally, QQ plot indicated that the dataset would perform better with log transformation. The correlation co-efficient indicated that there is definitely a positive relationship between Life and log transformed income variables. It was surprising that the population varaiable did not have much correlation with the Life variable.

We performed linear regression on the log transformed data. Th eprimary reason for regression model over classification model was due to the fact that the target and featire variables were quantitative in nature. The linear regression diagnostics clearly indicated violations in the assumptions of linearity, normality and homoscedasticity. We did not find a influential residual outlier to warrant dropping those outliers. The linear regression model performance was 64.1% that could be improved to 67.7% with 5-fold cross validation. This performance is not considred great or satisfactory. Other ways to improve the model performance is to explore non-linear regression models and/or look at adding more features that can influence the Life target variable. For example, healthcare variables such proximity to hospitals, # of hospitals, # of physicians etc can help improve the model performance when considred together.

We also performed unsupervised learning of k-means clustering to understand the groupings of data. We compared and contrasted non trasnformed, scaled and log transformed datasets with each other. It was clear that log transformed dataset performed the best (66%) in being able to cluster datapoints, as opposed to scaled dataset (64%) or non transformed datasets (54%). We visualized the scatter plots of the datapoints after grouping. We also used the Elbow method to identify possible optimal k values for the number of clusters.

Based on the p-values, we can see that the test is significant and we can reject the null hypotheisis (there is no relation between Life and Income variables). We can clearly see that there is indeed strong positive relation between income and life expectancy variables. However, we will need more data and/or additional variables to better predict life expectancy in the future. It would be interesting to understand the relation between income and life expectancy with a region and/or a country, but that is better left for the next project.

```
In [ ]:
```

```
In [ ]:
```