# Problem Statement:

A survey of countries’ GDP, Energy Consumption and Population development from 1971 - 2014. Determining the relationship of countries’ GDP and Energy Consumption against Population. Finally, an attempt to build a forecast model using Prophet and scikit-learn. It is useful in terms of giving the reader an outlook of the future picture.

Original code are from: <https://github.com/smdltn/Springboard-project>

<https://github.com/smdltn/Springboard-project/blob/master/Data%20Story.ipynb>

<https://github.com/smdltn/Springboard-project/blob/master/Statistical%20Data%20Analysis.ipynb>

# Data importing and initial exploratory

The data were downloaded from a single source - The World Bank - to GDP.csv, population.csv and energy.csv respectively.

To create a better uniformed DataFrames from these source csv files, importing these files required some initial cleanup. The first four rows were not needed and to be omitted, country names will be used as index. Furthermore, three columns: Country Code, Indicator Name and Indicator Code were also not needed and will be dropped from the DataFrame. While importing csv files, any missing cell (represented by blank cells) will be replaced with NaN values.

|  |
| --- |
| #Importing data  population = pd.read\_csv('population.csv', skiprows=4, index\_col=0, na\_values = ' ')  energy = pd.read\_csv('energy.csv', skiprows=4, index\_col=0, na\_values=' ')  gdp = pd.read\_csv('gdp.csv', skiprows=3, index\_col=0, na\_values = ' ')  #Cleaning up unneeded columns  population = population.drop(['Country Code', 'Indicator Name', 'Indicator Code'], axis=1)  energy = energy.drop(['Country Code', 'Indicator Name', 'Indicator Code'], axis=1)  gdp = gdp.drop(['Country Code', 'Indicator Name', 'Indicator Code'], axis=1)  gdp = gdp.drop(['Unnamed: 64'], axis=1) |

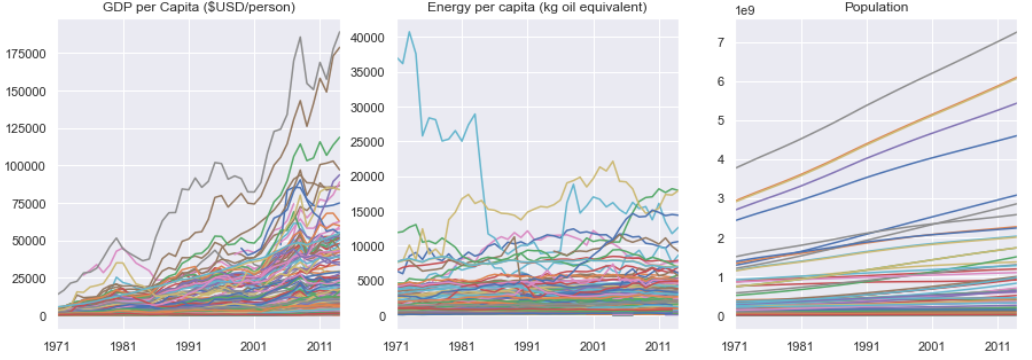
Each DataFrame has indices as country names and columns are the years that data were collected. GDP DataFrame however, has its last column named ‘Unnamed: 64’ that was also dropped for uniformity purposes.

Although each DataFrame has the years listed from 1960 - 2019,however, not all data are available consistently throughout that time frame. The data were available consistently from 1971 - 2014, thus we will use that time frame for any further analysis. Cleaning up the available data:

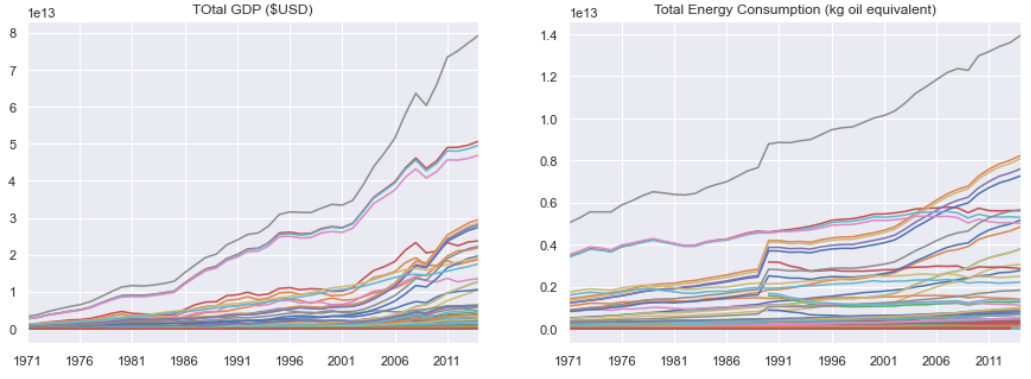
|  |
| --- |
| gdp = gdp.drop(gdp.loc[:, '1960':'1970'].columns, axis =1)  gdp = gdp.drop(gdp.loc[:, '2015':'2019'].columns, axis =1)  population = population.drop(population.loc[:, '1960':'1970'].columns, axis =1)  population = population.drop(population.loc[:, '2015':'2019'].columns, axis =1)  energy = energy.drop(energy.loc[:, '1960':'1970'].columns, axis =1)  energy = energy.drop(energy.loc[:, '2015':'2019'].columns, axis =1) |

## Initial exploratory analysis

To carry out the initial analysis, each DataFrame was plotted using a simple plot() method available for panda. Each graph will have the year on the x-axis and their respective values (population, energy consumption, GDP) on the y-axis. Each colored represents a country. Noted that GDP is converted to per Capita value to be consistent with Energy Consumption



With the exception of population plot (right-most), per Capita GDP and Energy Consumption do not represent a clear trend. We will look at the Total GDP and Total Energy Consumption values instead:



It is clearly better looking than the previous plots.

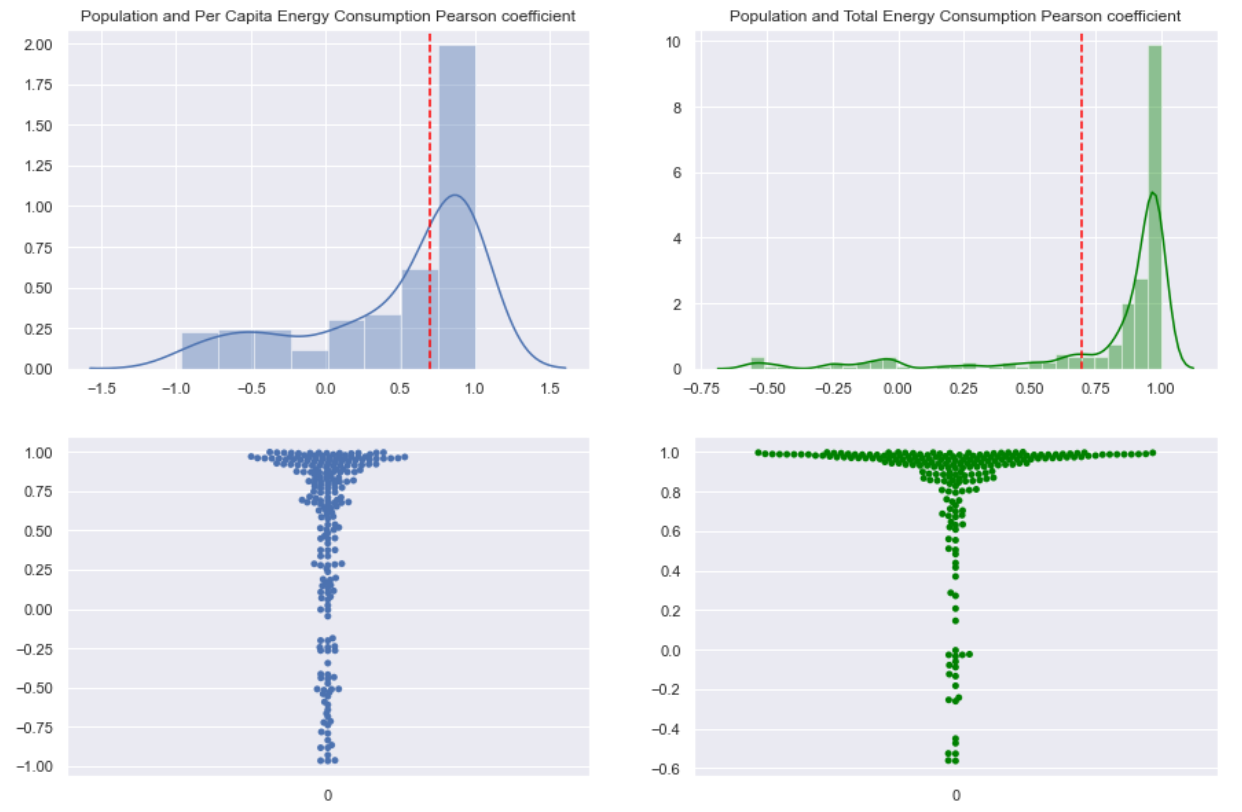
From these initial plots, we can conclude that most countries have a positive development (increase over time) from 1971 - 2014. Although, the overall positive trend does not always translate to the overall positive trend to each person's quality of life. The initial suspect is that population plays a large role here: as countries’ population increases, the total contribution (GDP and Energy Consumption) also increases.

We will further use Pearson Correlation Coefficient (Pearson CC) method to quantify this suspicion.

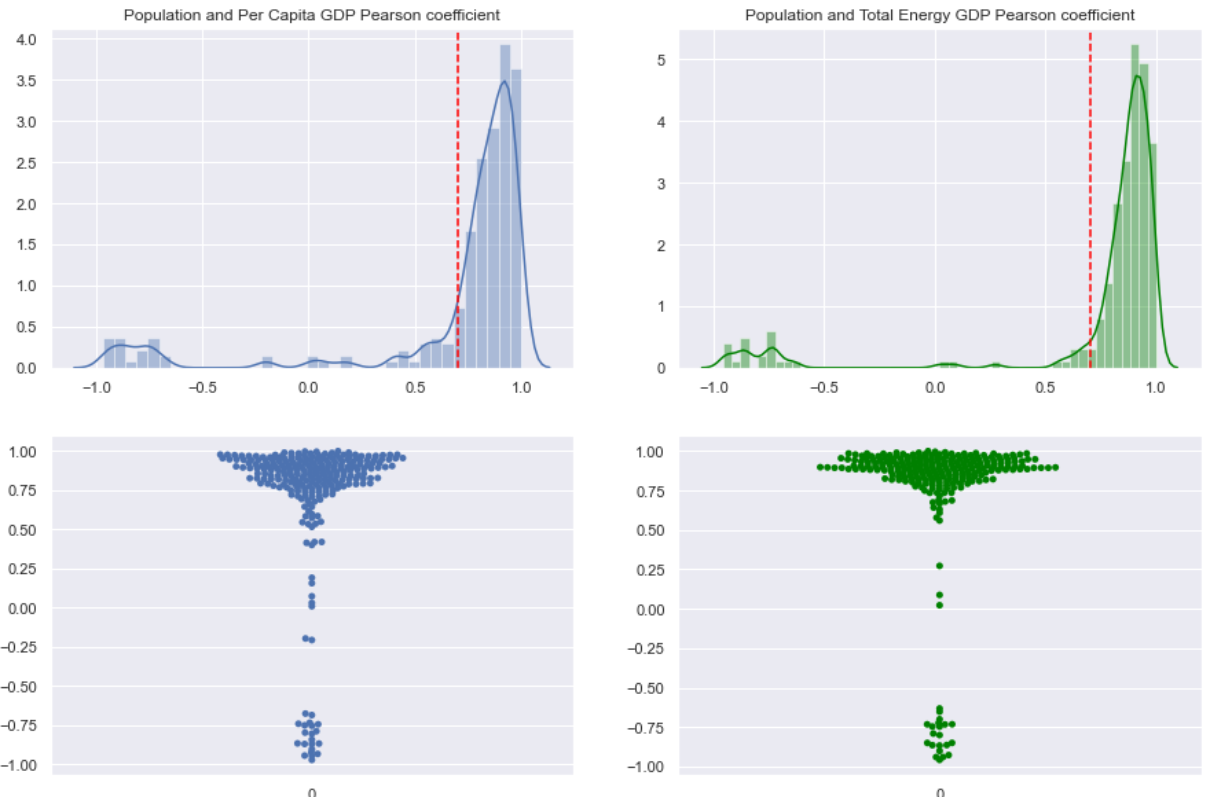
For each country, we will measure the correlation between Energy Consumption and GDP against Population. We will plot these Pearson CC values using histogram and swarmplot.

|  |
| --- |
| # Determine the Pearson CC for (GDP vs Population) and (Energy Consumption vs Population) Total and Per Capita values  #Per Capita values  pop\_eng = population.corrwith(energy, axis=1)  pop\_gdp = population.corrwith(gdp/population, axis=1)  #Total values  pop\_eng\_tot = population.corrwith(energy\*population, axis=1)  pop\_gdp\_tot = population.corrwith(gdp, axis=1) |

The Pearson CC between Energy Consumption and Population



The Pearson CC between GDP and Population



Just by the appearance of these plots, we can see their distribution are not even comparing the per Capita value (plots on the left) and Total values (plots on the right)

It is considered two variables have positive correlation if their Pearson CC values are larger or equal to 0.7. A further calculation of the percentage of these Pearson CC values >= 0.7 was conducted.

* Energy Consumption: it was 66% of countries have their Pearson CC values >= 0.7 if using Total values; while it was only 43% if per Capita values were used against population
* GDP: it was 85% vs. 79%

Conclusion: If we use GDP and Energy Consumption as the measurements for one’s quality of life then despite the overall positive trend of countries’ development, the positive trend does not always translate to the improvement of their citizen quality of life

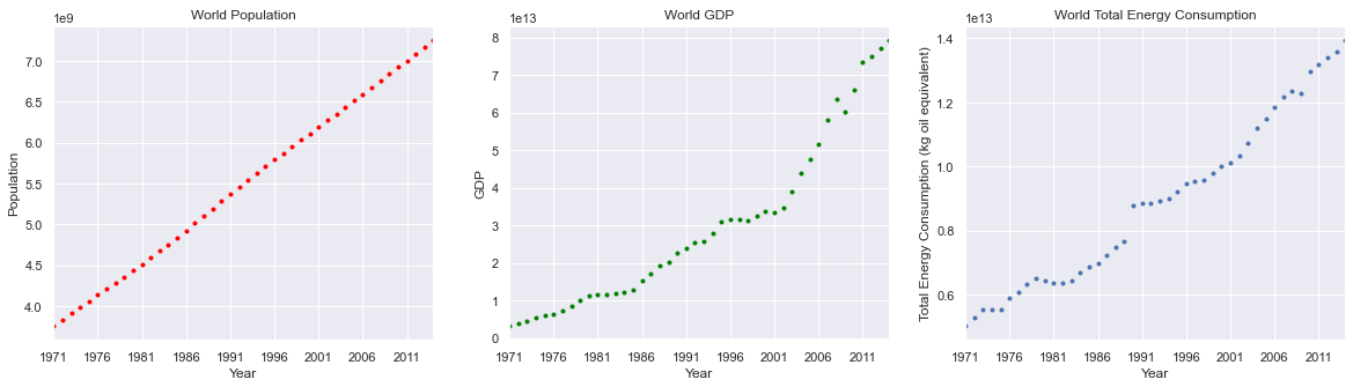
# 

# Building Forecasting Model using scikit-learn and Prophet

We will now use World data to build our forecasting model. Extracting World data from the DataFrames:

|  |
| --- |
| w\_pop = population.loc['World']  w\_gdp = gdp.loc['World']  w\_eng = energy.loc['World']\*w\_pop |

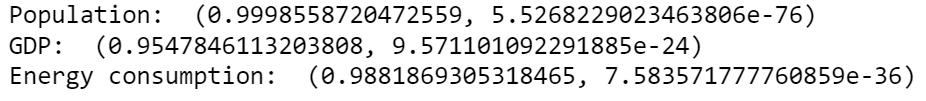
Plotting these data using plot() method of panda:



Comparing these three plots, it looks like building a forecasting model with population will yield the least error and least uncertainties.

We can also prove that intuition by calculating the Pearson CC values of each of these features against time:.

|  |
| --- |
| #Determining Pearson CCs and their associated p-values:  print('Population: ', stats.pearsonr(w\_pop.index.astype(int), w\_pop.values))  print('GDP: ', stats.pearsonr(w\_gdp.index.astype(int), w\_gdp.values))  print('Energy consumption: ', stats.pearsonr(w\_eng.index.astype(int), w\_eng.values)) |



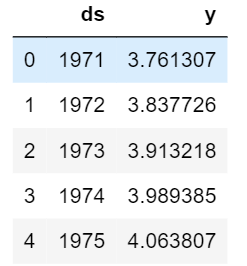
Population vs Time seems to have the highest correlation (highest Pearson CC). Its p-value is also smaller than alpha value of 0.05. Thus, we can reject the Null hypothesis and conclude that it is not only by chance that Population has that high of a Pearson CC with time. WIth this high level of correlation, we can have more confidence in our forecasting model.

## Using Prophet

We will create a new DataFrame with the requirements that it has two columns: date name as ‘ds’ and values column named as ‘y’. We will also use billion to drop the 1e9 from our calcaulation

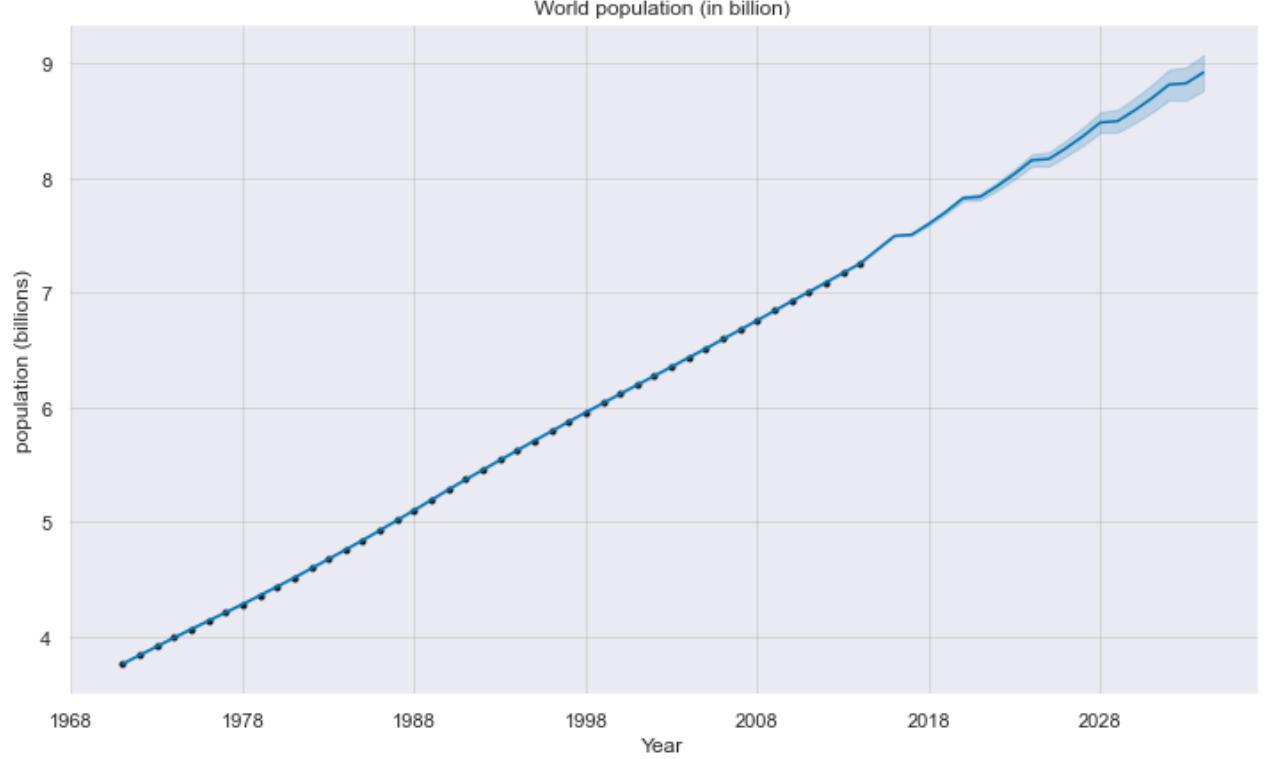
|  |
| --- |
| pop = w\_pop  pop = pop.reset\_index()  pop = pop.rename(columns={'index':'ds', 'World':'y'})  pop['y'] = pop['y']/1e9 |

Our new population dataframe will look like this:



Coding for using Prophet:

|  |
| --- |
| pop\_prophet = fbprophet.Prophet(changepoint\_prior\_scale=0.15)  pop\_prophet.fit(pop)  pop\_forecast = pop\_prophet.make\_future\_dataframe(periods=20, freq='Y')  pop\_forecast = pop\_prophet.predict(pop\_forecast) |



Plotting the forecast here. The future values were extended to 20 years beyond 2014.

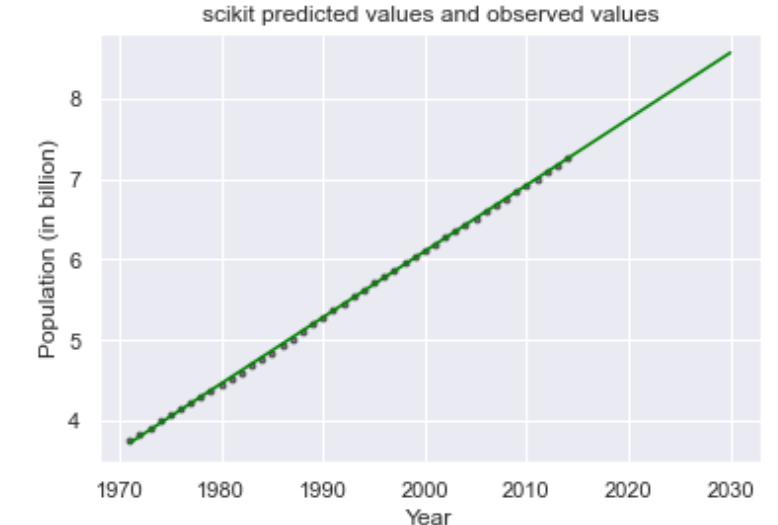
## Using scikit-learn

|  |
| --- |
| from sklearn.linear\_model import LinearRegression  # This creates a LinearRegression object  X = pop.drop('y', axis=1)  lm = LinearRegression() |

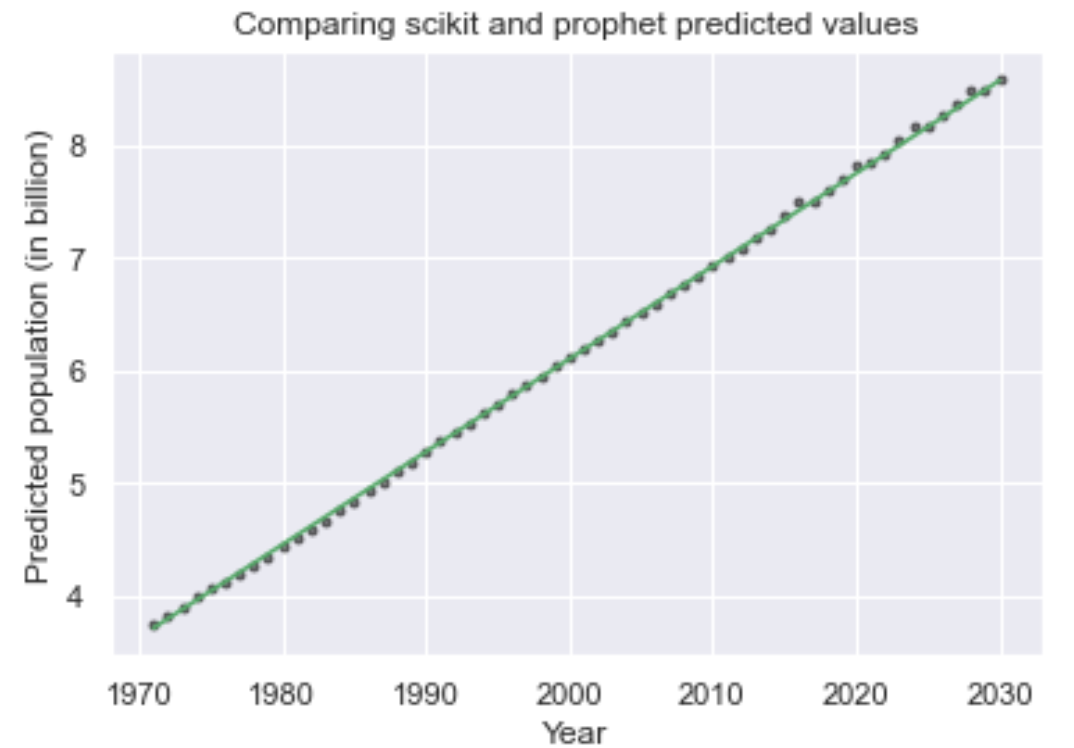
|  |
| --- |
| lm.fit(X, pop.y) |

This is a relatively simple use of scikit-learn. Here, we estimated a linear relationship with our available population data. Then we can use the slope and intercept to plug in future data as x-values are the future years (2015 - 2034). Or we can use the the predict() method available

|  |
| --- |
| sk\_predict = pd.DataFrame(lm.predict(np.arange(1971, 2031, 1).reshape(-1,1)))  sk\_predict = sk\_predict.set\_index(np.arange(1971, 2031, 1))  sk\_predict = sk\_predict.rename(columns={0:'sk\_predict'}) |

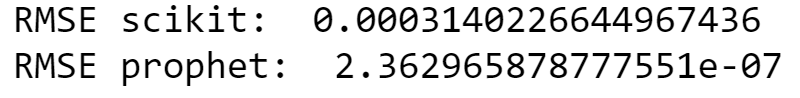


A visual comparison of the two forecast method: black dots represent predicted values using the Prophet while green line represents scikit-learn method.



They are indeed not too far from each other. We can calculate the Root Mean Squared Error of each method to compare their performance:

|  |
| --- |
| print('RMSE scikit: ', mean\_squared\_error(w\_pop/1e9, sk\_predict.loc['1971':'2014']))  print('RMSE prophet: ', mean\_squared\_error(w\_pop/1e9, fbprophet\_.loc['1971':'2014'])) |



Prophet seems to perform a lot better than scikit-learn in this situation. Although, it is not included here but predicted values are listed in the pop\_forecast DataFrame in ‘yhat’ column.