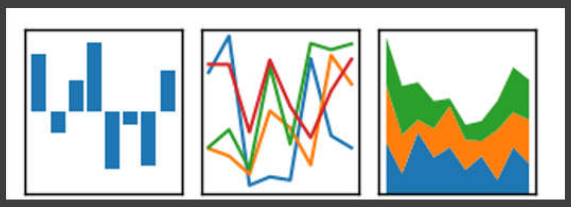
TASK #1. UNDERSTAND THE PROBLEM STATEMENT

PROJECT OVERVIEW

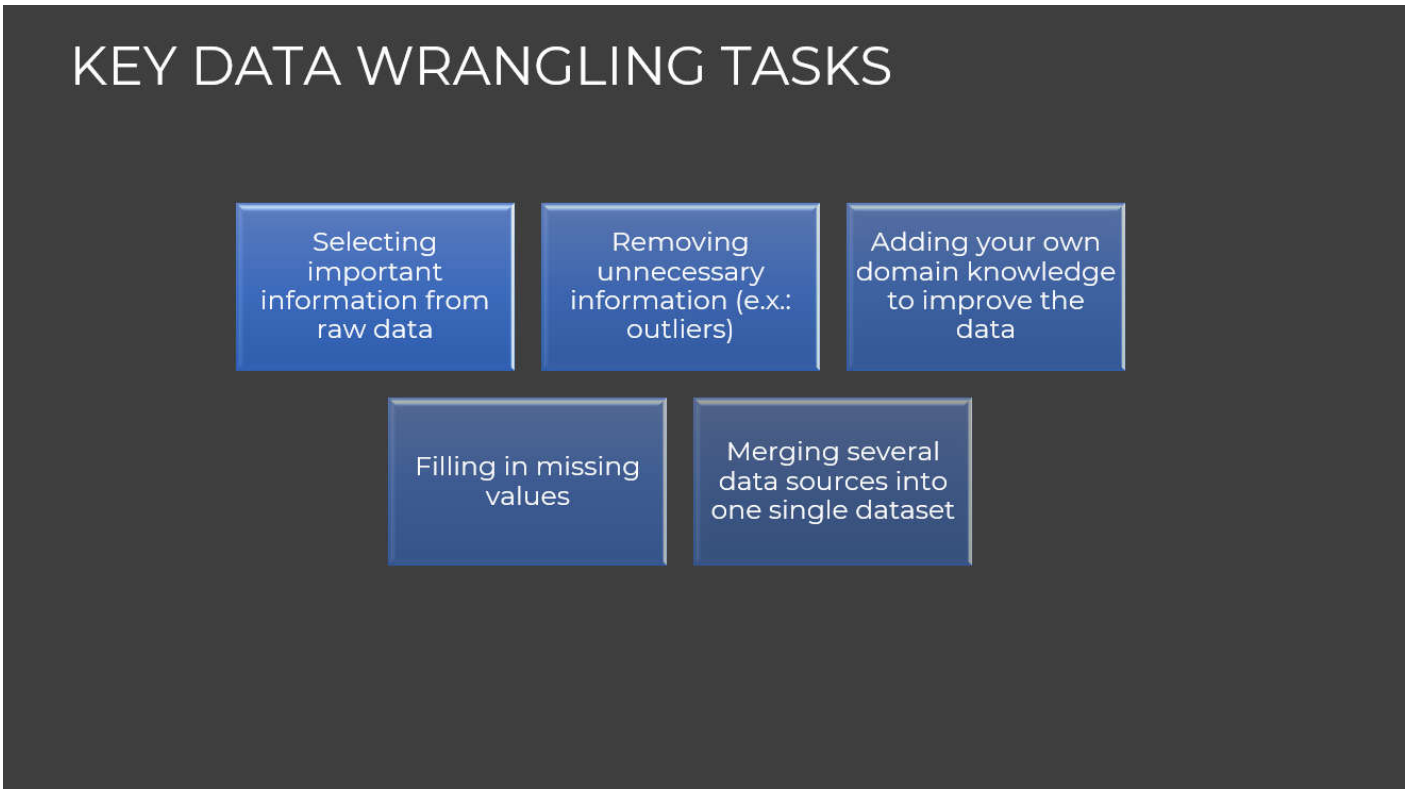
- In this project, we will analyze life expectancy data by performing data wrangling & exploratory data analysis (EDA).
- Pandas is a powerful open source data analysis tools in python.
- Exploratory Data Analysis (EDA) is a process of analyzing data to gain valuable insights such as statistical summary & visualizations.

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



"Data scientists spend ~80% of their time performing data wrangling & EDA"



TASK #2. IMPORT DATASET AND PERFORM BASIC STATISTICAL DATA ANALYSIS

In [2]:
```
# Import Pandas Library
import pandas as pd
```

In [2]:
```
# Force Pandas to display all rows and columns
pd.set_option('max_columns', None)
# pd.set_option('display.max_rows', None)
```

In [2]:
```
# Let's read a CSV file using Pandas as follows
df = pd.read_csv('Life_Expectancy_Data.csv')
```

In [4]:
```
# Let's obtain the datatype
type(df)
```

Out[4]:
```
pandas.core.frame.DataFrame
```

In [3]:
```
# you can view the first couple of rows using .head()
df.head(6)
```

Out[3]:

	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI
0	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	19.1
1	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492	18.6
2	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	18.1
3	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	17.6
4	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	17.2
5	2010	Developing	58.8	279.0	74	0.01	79.679367	66.0	1989	16.7

6 rows × 21 columns

In [6]:
```
# you can view the last couple of rows using .tail()
df.tail(4)
```

Out[6]:

	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	
2934	2003	Developing	44.5	715.0	26	4.06	0.0	7.0	998	
2935	2002	Developing	44.8	73.0	25	4.43	0.0	73.0	304	
2936	2001	Developing	45.3	686.0	25	1.72	0.0	76.0	529	
2937	2000	Developing	46.0	665.0	24	1.68	0.0	79.0	1483	

In [7]:
```
# Calculate the average values for employee_df dataframe
round(df.mean())
```

Out[7]:
```
Year                                2008.0
Life expectancy                       69.0
Adult Mortality                      165.0
infant deaths                         30.0
Alcohol                                5.0
percentage expenditure               738.0
Hepatitis B                           81.0
Measles                             2420.0
 BMI                                  38.0
under-five deaths                     42.0
Polio                                 83.0
Total expenditure                      6.0
Diphtheria                            82.0
 HIV/AIDS                              2.0
GDP                                7483.0
Population                     12753375.0
 thinness  1-19 years                  5.0
 thinness 5-9 years                    5.0
Income composition of resources        1.0
Schooling                             12.0
dtype: float64
```

In [8]:
```
# 21 features in total
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2938 entries, 0 to 2937
Data columns (total 21 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Year                             2938 non-null   int64  
 1   Status                           2938 non-null   object 
 2   Life expectancy                  2928 non-null   float64
 3   Adult Mortality                  2928 non-null   float64
 4   infant deaths                    2938 non-null   int64  
 5   Alcohol                          2744 non-null   float64
 6   percentage expenditure           2938 non-null   float64
 7   Hepatitis B                      2385 non-null   float64
 8   Measles                          2938 non-null   int64  
 9    BMI                             2904 non-null   float64
 10  under-five deaths                2938 non-null   int64  
 11  Polio                            2919 non-null   float64
 12  Total expenditure                2712 non-null   float64
 13  Diphtheria                       2919 non-null   float64
 14   HIV/AIDS                        2938 non-null   float64
 15  GDP                              2490 non-null   float64
 16  Population                       2286 non-null   float64
 17   thinness  1-19 years            2904 non-null   float64
 18   thinness 5-9 years             2904 non-null   float64
 19  Income composition of resources  2771 non-null   float64
 20  Schooling                        2775 non-null   float64
dtypes: float64(16), int64(4), object(1)
memory usage: 482.1+ KB
```

In [9]:
```
# Obtain a Statistical Summary about the data
df.describe()
```

Out[9]:

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis
count	2938.000000	2928.000000	2928.000000	2938.000000	2744.000000	2938.000000	2385.0000
mean	2007.518720	69.224932	164.796448	30.303948	4.602861	738.251295	80.9404
std	4.613841	9.523867	124.292079	117.926501	4.052413	1987.914858	25.0700
min	2000.000000	36.300000	1.000000	0.000000	0.010000	0.000000	1.0000
25%	2004.000000	63.100000	74.000000	0.000000	0.877500	4.685343	77.0000
50%	2008.000000	72.100000	144.000000	3.000000	3.755000	64.912906	92.0000
75%	2012.000000	75.700000	228.000000	22.000000	7.702500	441.534144	97.0000
max	2015.000000	89.000000	723.000000	1800.000000	17.870000	19479.911610	99.0000

PRACTICE OPPORTUNITY #1 [OPTIONAL]:

- **Calculate the mean, maximum and minimum GDP considered in this study using a different strategy**
- **What does GDP mean? what is the relationship between GDP and life expectancy? [External Research is Required]**

In [ ]:

TASK #3. DEALING WITH MISSING DATA

In [13]:
```
df
```

Out[13]:

	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles
0	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154
1	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492
2	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430
3	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787
4	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013
...	...	...	...	...	...	...	...	...	...
2933	2004	Developing	44.3	723.0	27	4.36	0.000000	68.0	31
2934	2003	Developing	44.5	715.0	26	4.06	0.000000	7.0	998
2935	2002	Developing	44.8	73.0	25	4.43	0.000000	73.0	304
2936	2001	Developing	45.3	686.0	25	1.72	0.000000	76.0	529
2937	2000	Developing	46.0	665.0	24	1.68	0.000000	79.0	1483

2938 rows × 21 columns

In [14]:
```
# Let's locate rows that have Null values
# For example, notice Row index 1716 has many missing values
df.isnull()
```

Out[14]:

	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BM
0	False	False	False	False	False	False	False	False	False	Fals
1	False	False	False	False	False	False	False	False	False	Fals
2	False	False	False	False	False	False	False	False	False	Fals
3	False	False	False	False	False	False	False	False	False	Fals
4	False	False	False	False	False	False	False	False	False	Fals
...	...	...	...	...	...	...	...	...	...	.
2933	False	False	False	False	False	False	False	False	False	Fals
2934	False	False	False	False	False	False	False	False	False	Fals
2935	False	False	False	False	False	False	False	False	False	Fals
2936	False	False	False	False	False	False	False	False	False	Fals
2937	False	False	False	False	False	False	False	False	False	Fals

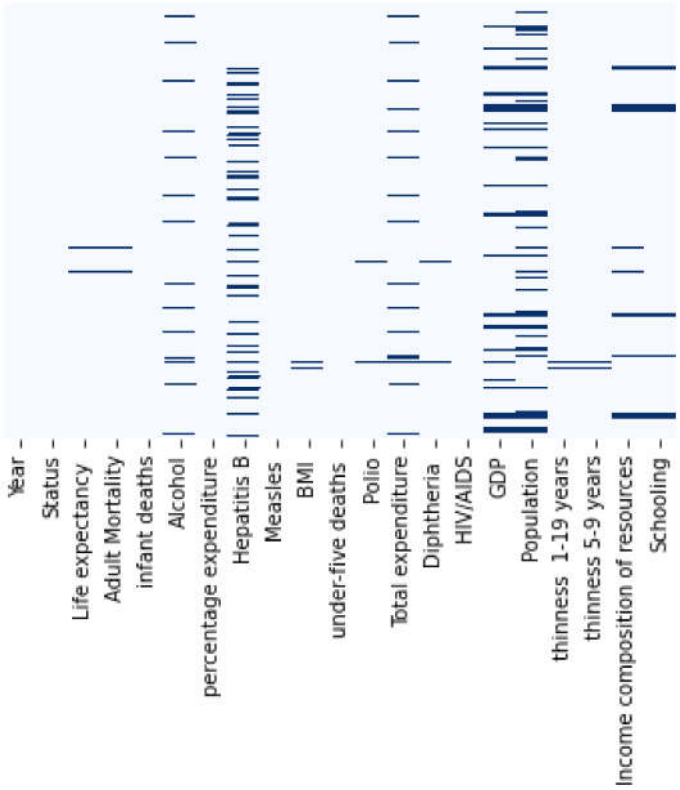
2938 rows × 21 columns

In [15]:
```
# Let's see the total number of missing elements per column
df.isnull().sum()
```

Out[15]:
```
Year                               0
Status                             0
Life expectancy                   10
Adult Mortality                   10
infant deaths                      0
Alcohol                          194
percentage expenditure             0
Hepatitis B                      553
Measles                            0
 BMI                              34
under-five deaths                  0
Polio                             19
Total expenditure                226
Diphtheria                        19
 HIV/AIDS                          0
GDP                              448
Population                       652
 thinness  1-19 years             34
 thinness 5-9 years               34
Income composition of resources  167
Schooling                        163
dtype: int64
```

In [16]:
```
# check if there are any Null values
import seaborn as sns
sns.heatmap(df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x22585c7a390>



In [17]:
```
# Drop any row that contains a Null value
# Note that the size of the dataframe has been reduced by from 2938 to 1649
df.dropna(how = 'any', inplace = True)
df
```

Out[17]:

	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles
0	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154
1	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492
2	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430
3	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787
4	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013
...	...	...	...	...	...	...	...	...	...
2933	2004	Developing	44.3	723.0	27	4.36	0.000000	68.0	31
2934	2003	Developing	44.5	715.0	26	4.06	0.000000	7.0	998
2935	2002	Developing	44.8	73.0	25	4.43	0.000000	73.0	304
2936	2001	Developing	45.3	686.0	25	1.72	0.000000	76.0	529
2937	2000	Developing	46.0	665.0	24	1.68	0.000000	79.0	1483

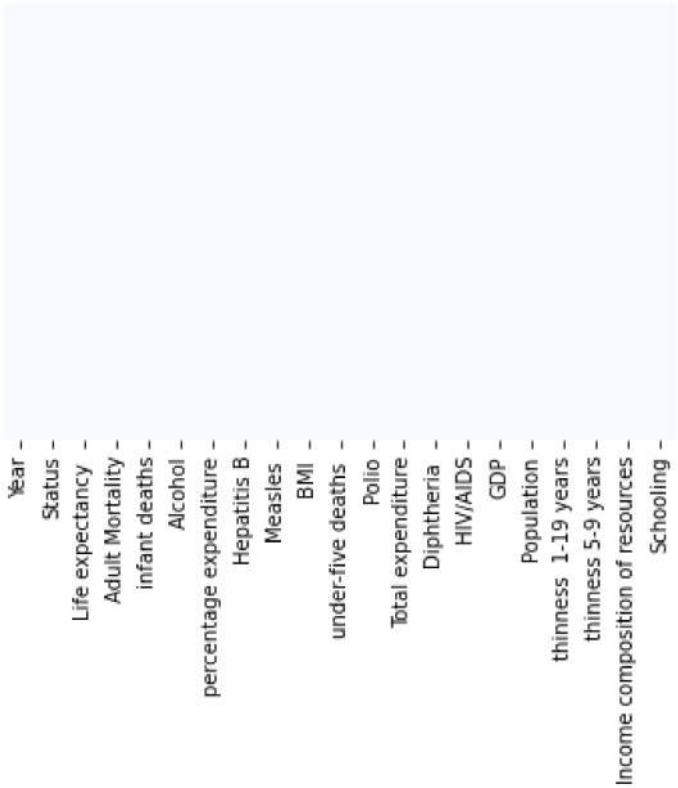
1649 rows × 21 columns

In [17]:
```
# Let's check if we still have any missing values
df.isnull().sum()
```

Out[17]:
```
Year                               0
Status                             0
Life expectancy                    0
Adult Mortality                    0
infant deaths                      0
Alcohol                            0
percentage expenditure             0
Hepatitis B                        0
Measles                            0
 BMI                               0
under-five deaths                  0
Polio                              0
Total expenditure                  0
Diphtheria                         0
 HIV/AIDS                          0
GDP                                0
Population                         0
 thinness  1-19 years              0
 thinness 5-9 years                0
Income composition of resources    0
Schooling                          0
dtype: int64
```

In [19]:
```
# check if there are any Null values
sns.heatmap(df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22587f5ba90>
```



In [20]:
```
# Let's explore threshold based method to deal with missing values
# Let's read the raw data again using Pandas as follows
df = pd.read_csv('Life_Expectancy_Data.csv')
df.head()
```

Out[20]:

	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI
0	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	19.1
1	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492	18.6
2	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	18.1
3	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	17.6
4	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	17.2

5 rows × 21 columns

In [11]:
```
# You can drop NaN rows based on a threshold
# Drop any row that has at least 3 NON-NaNs within it:
df.dropna(axis=0, thresh=18, inplace=True)
```

In [12]:
```
# Let's check if we still have any missing values
df.isnull().sum()
```

Out[12]:
```
Year                               0
Status                             0
Life expectancy                    3
Adult Mortality                    3
infant deaths                      0
Alcohol                          149
percentage expenditure             0
Hepatitis B                      488
Measles                            0
 BMI                              10
under-five deaths                  0
Polio                              7
Total expenditure                154
Diphtheria                         7
 HIV/AIDS                          0
GDP                              260
Population                       467
 thinness  1-19 years             10
 thinness 5-9 years               10
Income composition of resources    1
Schooling                          0
dtype: int64
```

In [19]:
```
df
```

Out[19]:

	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles
0	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154
1	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492
2	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430
3	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787
4	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013
...	...	...	...	...	...	...	...	...	...
2725	2004	Developing	44.3	723.0	27	4.36	0.000000	68.0	31
2726	2003	Developing	44.5	715.0	26	4.06	0.000000	7.0	998
2727	2002	Developing	44.8	73.0	25	4.43	0.000000	73.0	304
2728	2001	Developing	45.3	686.0	25	1.72	0.000000	76.0	529
2729	2000	Developing	46.0	665.0	24	1.68	0.000000	79.0	1483

2730 rows × 21 columns

In [18]:
```
#The drop=True parameter tells Pandas not to keep a backup copy of the origina
l index.
df.reset_index(drop=True, inplace=True)
df
```

Out[18]:

	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles
0	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154
1	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492
2	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430
3	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787
4	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013
...	...	...	...	...	...	...	...	...	...
2725	2004	Developing	44.3	723.0	27	4.36	0.000000	68.0	31
2726	2003	Developing	44.5	715.0	26	4.06	0.000000	7.0	998
2727	2002	Developing	44.8	73.0	25	4.43	0.000000	73.0	304
2728	2001	Developing	45.3	686.0	25	1.72	0.000000	76.0	529
2729	2000	Developing	46.0	665.0	24	1.68	0.000000	79.0	1483

2730 rows × 21 columns

In [5]:
```
# Let's explore an alternative (smarter) method to deal with missing values
# Let's read the raw data again using Pandas as follows
df = pd.read_csv('Life_Expectancy_Data.csv')
df.head()
```

Out[5]:

	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI
0	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	19.1
1	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492	18.6
2	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	18.1
3	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	17.6
4	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	17.2

5 rows × 21 columns

In [26]:
```
# Calculate the average monthly income
df['GDP'].mean()
```

Out[26]:
```
7483.158469138481
```

In [27]:
```
# You can use Fillna to fill a given column with a certain value
df['GDP'].fillna(df['GDP'].mean(), inplace = True)
```

In [28]:
```
# Let's check if we still have any missing values
df.isnull().sum()
```

Out[28]:
```
Year                               0
Status                             0
Life expectancy                   10
Adult Mortality                   10
infant deaths                      0
Alcohol                          194
percentage expenditure             0
Hepatitis B                      553
Measles                            0
 BMI                              34
under-five deaths                  0
Polio                             19
Total expenditure                226
Diphtheria                        19
 HIV/AIDS                          0
GDP                                0
Population                       652
 thinness  1-19 years             34
 thinness 5-9 years               34
Income composition of resources  167
Schooling                        163
dtype: int64
```

In [13]:
```
# bfill/ffill
df.fillna(method='ffill', inplace=True)
# Let's check if we still have any missing values
df.isnull().sum()
```

Out[13]:
```
Year                               0
Status                             0
Life expectancy                    0
Adult Mortality                    0
infant deaths                      0
Alcohol                            0
percentage expenditure             0
Hepatitis B                        0
Measles                            0
 BMI                               0
under-five deaths                  0
Polio                              0
Total expenditure                  0
Diphtheria                         0
 HIV/AIDS                          0
GDP                                0
Population                         0
 thinness  1-19 years              0
 thinness 5-9 years                0
Income composition of resources    0
Schooling                          0
dtype: int64
```

In [14]:
```
# Let's explore interolation method to deal with missing values
# Let's read the raw data again using Pandas as follows
df = pd.read_csv('Life_Expectancy_Data.csv')
```

In [16]:
```
#fill using interolation
df.interpolate(method ='linear', limit_direction ='forward', inplace=True)
# Let's check if we still have any missing values
df.isnull().sum()
```

Out[16]:
```
Year                               0
Status                             0
Life expectancy                    0
Adult Mortality                    0
infant deaths                      0
Alcohol                            0
percentage expenditure             0
Hepatitis B                        0
Measles                            0
 BMI                               0
under-five deaths                  0
Polio                              0
Total expenditure                  0
Diphtheria                         0
 HIV/AIDS                          0
GDP                                0
Population                         0
 thinness  1-19 years              0
 thinness 5-9 years                0
Income composition of resources    0
Schooling                          0
dtype: int64
```

methods : {'linear', 'time', 'index', 'values', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'barycentric', 'krogh', 'polynomial', 'spline', 'piecewise_polynomial', 'from_derivatives', 'pchip', 'akima'}

Linear interpolation is a method of estimating values between two known values in a series of data. In the context of filling missing values in a pandas DataFrame, linear interpolation estimates the missing values by computing a straight line between the two nearest known values. The method assumes that the change in the dependent variable (the missing value) is constant with respect to the independent variable (time or index). The missing value is then estimated as a weighted average of the two nearest known values, where the weight is proportional to the distance between the missing value and the known values.

In [8]:
```
#droping duplicates
df.drop_duplicates()
```

Out[8]:

	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles
0	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154
1	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492
2	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430
3	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787
4	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013
...	...	...	...	...	...	...	...	...	...
2933	2004	Developing	44.3	723.0	27	4.36	0.000000	68.0	31
2934	2003	Developing	44.5	715.0	26	4.06	0.000000	7.0	998
2935	2002	Developing	44.8	73.0	25	4.43	0.000000	73.0	304
2936	2001	Developing	45.3	686.0	25	1.72	0.000000	76.0	529
2937	2000	Developing	46.0	665.0	24	1.68	0.000000	79.0	1483

2938 rows × 21 columns

PRACTICE OPPORTUNITY #2 [OPTIONAL]:

- **Calculate the median total expenditure**
- **Use the calculated median values to fill out missing data in the total expenditure column**
- **Confirm that the process is successful**

In [ ]:
```

```

In [ ]:
```

```

TASK #4: DATA-TYPE CONVERSION AND ONE-HOT ENCODING

ONE-HOT ENCODING

- Can we simply replace colors with integer values?
- The machine learning model will assume that:

$$GREEN > YELLOW > RED$$

COLOR	ENCODED COLOR
RED	1
RED	1
YELLOW	2
GREEN	3
YELLOW	2

WRONG!

ONE-HOT ENCODING

- One hot encoding works by converting values such as “color” into columns with 1’s and 0’s in them.
- Since data science models deal with numbers, we perform one hot encoding to convert from categorical data into numerical.

COLOR	RED	YELLOW	GREEN
RED	1	0	0
RED	1	0	0
YELLOW	0	1	0
GREEN	0	0	1
YELLOW	0	1	0

Working with Data-types

In [62]:
```
# Let's load a file with datatype errors
columns=['education', 'age', 'capital-gain', 'race', 'capital-loss',
         'hours-per-week', 'gender', 'classification']
df =  pd.read_csv('census.csv',names=columns, header=None)
```

In [32]:
```
df.head()
```

Out[32]:

	education	age	capital-gain	race	capital-loss	hours-per-week	gender	classification
0	Bachelors	39	2174	White	0	40	Male	<=50K
1	Bachelors	50	?	White	0	13	Male	<=50K
2	HS-grad	38	?	White	0	40	Male	<=50K
3	11th	53	?	Black	0	40	Male	<=50K
4	Bachelors	28	0	Black	0	40	Female	<=50K

In [28]:
```
# lets look at the data types
df.dtypes
```

Out[28]:
```
education          object
age                 int64
capital-gain       object
race               object
capital-loss        int64
hours-per-week      int64
sex                object
classification     object
dtype: object
```

If your data types don't look the way you expected them, explicitly convert them to the desired type using the .to_datetime(), .to_numeric() etc.

In [63]:
```
df['capital-gain'] = pd.to_numeric(df['capital-gain'],errors='coerce')
```

In [30]:
```
# Let's look at the updated data type
df.dtypes
```

Out[30]:
```
education          object
age                 int64
capital-gain      float64
race               object
capital-loss        int64
hours-per-week      int64
sex                object
classification     object
dtype: object
```

Take note how to_numeric properly converts to decimal or integer depending on the data it finds. The errors='coerce' parameter instructs Pandas to enter a NaN at any field where the conversion fails.

errors{‘ignore’, ‘raise’, ‘coerce’}, default ‘raise’

If ‘raise’, then invalid parsing will raise an exception.

If ‘coerce’, then invalid parsing will be set as NaN.

If ‘ignore’, then invalid parsing will return the input.

Categorical nominal encoding

In [64]:
```
df.gender = df.gender.astype('category').cat.codes
```

In [34]:
```
df.head()
```

Out[34]:

	education	age	capital-gain	race	capital-loss	hours-per-week	gender	classification
0	Bachelors	39	2174	White	0	40	1	<=50K
1	Bachelors	50	?	White	0	13	1	<=50K
2	HS-grad	38	?	White	0	40	1	<=50K
3	11th	53	?	Black	0	40	1	<=50K
4	Bachelors	28	0	Black	0	40	0	<=50K

In [65]:
```
#Label Encoding
df.classification = df.classification.astype('category').cat.codes
df.head()
```

Out[65]:

	education	age	capital-gain	race	capital-loss	hours-per-week	gender	classification
0	Bachelors	39	2174.0	White	0	40	1	0
1	Bachelors	50	NaN	White	0	13	1	0
2	HS-grad	38	NaN	White	0	40	1	0
3	11th	53	NaN	Black	0	40	1	0
4	Bachelors	28	0.0	Black	0	40	0	0

Categorical Ordinal Encoding

In [66]:
```
from pandas.api.types import CategoricalDtype
categories=['Preschool', '1st-4th', '5th-6th', '7th-8th', '10th', '9th', '12th', '11th',
            'Some-college', 'HS-grad', 'Bachelors', 'Masters', 'Doctorate']
Dtype = CategoricalDtype(categories=categories, ordered=True)
```

In [67]:
```
df.education=df.education.astype(Dtype).cat.codes
```

In [68]:
```
df.head()
```

Out[68]:

	education	age	capital-gain	race	capital-loss	hours-per-week	gender	classification
0	10	39	2174.0	White	0	40	1	0
1	10	50	NaN	White	0	13	1	0
2	9	38	NaN	White	0	40	1	0
3	7	53	NaN	Black	0	40	1	0
4	10	28	0.0	Black	0	40	0	0

One Hot Encoding

In [47]:
```
df = pd.get_dummies(df,columns=['race'])
```

In [48]:
```
df.head()
```

Out[48]:

	education	age	capital-gain	capital-loss	hours-per-week	gender	classification	race_Amer-Indian-Eskimo	race_Asian-Pac-Islander	race
0	Bachelors	39	2174.0	0	40	1	0	0	0	
1	Bachelors	50	NaN	0	13	1	0	0	0	
2	HS-grad	38	NaN	0	40	1	0	0	0	
3	11th	53	NaN	0	40	1	0	0	0	
4	Bachelors	28	0.0	0	40	0	0	0	0	

TASK #5: Data Integration

In [3]:
```
uber1 = pd.read_csv('uber1.csv')
uber2 = pd.read_csv('uber2.csv')
uber3 = pd.read_csv('uber3.csv')
```