Data cleaning and prepartion Exercise

Exercise 1: Inspecting your dataframe worth 35 points

In [5]:
```
#let us hide warnings from displaying in jupyter notebook (worth 5 points)
import warnings
warnings.filterwarnings('ignore')
```

In [33]:
```
print('I am Preston')
```

```
I am Preston
```

In [6]:
```
#following steps worth 10 points
# Import pandas
import pandas as pd


#Load your dataset data/gapminder-FiveYearData.csv into a dataframe
# Read the file into a DataFrame: df
df=pd.read_csv('data/gapminder-FiveYearData.csv')


#Visually inspect our dataframe using head method
df.head()
```

Out[6]:

	country	year	pop	continent	lifeExp	gdpPercap
0	Afghanistan	1952	8425333.0	Asia	28.801	779.445314
1	Afghanistan	1957	9240934.0	Asia	30.332	820.853030
2	Afghanistan	1962	10267083.0	Asia	31.997	853.100710
3	Afghanistan	1967	11537966.0	Asia	34.020	836.197138
4	Afghanistan	1972	13079460.0	Asia	36.088	739.981106

In [34]:
```
print('I am Preston')
```

```
I am Preston
```

In [7]:
```
#following instruction worth 5 points
#Visually inspect our dataframe using tail method

df.tail()
```

Out[7]:

	country	year	pop	continent	lifeExp	gdpPercap
1699	Zimbabwe	1987	9216418.0	Africa	62.351	706.157306
1700	Zimbabwe	1992	10704340.0	Africa	60.377	693.420786
1701	Zimbabwe	1997	11404948.0	Africa	46.809	792.449960
1702	Zimbabwe	2002	11926563.0	Africa	39.989	672.038623
1703	Zimbabwe	2007	12311143.0	Africa	43.487	469.709298

In [35]:
```
print('I am Preston')
```

```
I am Preston
```

In [8]:
```
#following instruction worth 5 points
#IF you have many columns in our dataframe
#We might find an extra space at the end or begining of the column name
#or bad characters
# iterating through the columns and display column names
for col in df.columns:
    print(col)
```

```
country
year
pop
continent
lifeExp
gdpPercap
```

In [36]:
```
print('I am Preston')
```

```
I am Preston
```

In [9]:
```
#display the number of rows and columns(worth 5 points)
df.shape
```

Out[9]:
```
(1704, 6)
```

In [37]:
```
print('I am Preston')
```

```
I am Preston
```

In [10]:
```
#following instruction worth 5 points
#We can use the a method to get additional information about our dataframe
#Inlcuding datatype of the different columns
#You can see how many missing values exist in each column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1704 entries, 0 to 1703
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   country    1704 non-null   object 
 1   year       1704 non-null   int64  
 2   pop        1704 non-null   float64
 3   continent  1704 non-null   object 
 4   lifeExp    1704 non-null   float64
 5   gdpPercap  1704 non-null   float64
dtypes: float64(3), int64(1), object(2)
memory usage: 80.0+ KB
```

In [38]:
```
print('I am Preston')
```

```
I am Preston
```

Exercise 2: Changing column datatype (worth 15 points)

In [11]:
```
##print a list of all column names and their data types of your dataframe df (wor
print(df.dtypes)
```

```
country       object
year           int64
pop          float64
continent     object
lifeExp      float64
gdpPercap    float64
dtype: object
```

In [39]:
```
print('I am Preston')
```

```
I am Preston
```

In [12]:
```
#change the datatype of country to category (worth 5 points)
df['country']=df['country'].astype('category')


#print the data types of each column after the change (worth 5 points)
print(df.dtypes)
```

```
country      category
year            int64
pop           float64
continent      object
lifeExp       float64
gdpPercap     float64
dtype: object
```

In [40]:
```
print('I am Preston')
```

```
I am Preston
```

Exercise 3 Pivoting (worth 30 points)

In [13]:
```
#The following steps (worth 1o points)
#import the required libraries
import numpy as np
import pandas as pd

#load the data set  data/gapminder-FiveYearData.csv into a dataframe named gapmir
gapminder=pd.read_csv('data/gapminder-FiveYearData.csv')

#display the head of the data fram
df.head()
```

Out[13]:

	country	year	pop	continent	lifeExp	gdpPercap
0	Afghanistan	1952	8425333.0	Asia	28.801	779.445314
1	Afghanistan	1957	9240934.0	Asia	30.332	820.853030
2	Afghanistan	1962	10267083.0	Asia	31.997	853.100710
3	Afghanistan	1967	11537966.0	Asia	34.020	836.197138
4	Afghanistan	1972	13079460.0	Asia	36.088	739.981106

In [41]:
```
print('I am Preston')
```

```
I am Preston
```

In [14]:
```
# select two columns continent and lifeExp from gapminder dataframe and assign it
pd.pivot_table(df,index=['continent','lifeExp'])
```

Out[14]:

continent	lifeExp	gdpPercap	pop	year
Africa	**23.599**	737.068595	7290203.0	1992.0
	30.000	485.230659	284320.0	1952.0
	30.015	3520.610273	4232095.0	1952.0
	30.331	879.787736	2143249.0	1952.0
	31.286	468.526038	6446316.0	1952.0
...	**...**	...	...	...
Oceania	**78.830**	26997.936570	18565243.0	1997.0
	79.110	23189.801350	3908037.0	2002.0
	80.204	25185.009110	4115771.0	2007.0
	80.370	30687.754730	19546792.0	2002.0
	81.235	34435.367440	20434176.0	2007.0

1661 rows × 3 columns

In [42]:
```
print('I am Preston')
```

```
I am Preston
```

- As a simple example, we can use Pandas pivot_table to convert the tall table to a wide table, computing the mean lifeExp across continents.
- To do that, we will use pd.pivot_table with the data frame as one of the arguments and specify which variable we would like use for columns and which variable we would like to summarize.
- One of the arguments of pivot_table, agg_func has mean as default.

In [15]:
```
# simple example with pivot_table
#write one line of code to display the following pivot table (worth 10 points)
```

continent	lifeExp
Africa	72.301
Africa	57.678
Europe	68.000
Europe	64.030
Asia	37.373

continent	Africa	Americas	Asia	Europe
lifeExp	48.86533	64.658737	60.064903	71.903686

In [16]:
```
#type your line of code here
pd.pivot_table(df,columns='continent',values='lifeExp',aggfunc='mean')
```

Out[16]:

continent	Africa	Americas	Asia	Europe	Oceania
lifeExp	48.86533	64.658737	60.064903	71.903686	74.326208

In [43]:
```
print('I am Preston')
```

```
I am Preston
```

Exercise 4: Pivoting (worth 20 points)

- Let us see another simple example of pivot_table.
- In the above example we used pivot_table to compute mean lifeExp for each continent.
- We can compute **mean lifeExp for each country.**

In [17]:
```
#create a data frame df that includes only required columns as mentioned in the e
pd.pivot_table(df,columns='country',values='lifeExp',aggfunc='mean')


#call the pivot_table method that will provide you with mean lifeExp for each cou
```

Out[17]:

country	Afghanistan	Albania	Algeria	Angola	Argentina	Australia	Austria	Bahrain	B
lifeExp	37.478833	68.432917	59.030167	37.8835	69.060417	74.662917	73.10325	65.605667	

1 rows × 142 columns

In [44]:
```
print('I am Preston')
```

```
I am Preston
```

Exercise 5:Pivoting (worth 25 points)

- As mentioned before, pivot_table uses mean function for aggregating or summarizing data by default.
- We can change the aggregating function, if needed.
- For example, we can use aggfunc='min' to compute "minimum" lifeExp instead of "mean" lifeExp for each year and continent values.
- The output should look something similar to the figure below

continent	Africa	Americas	Asia	Europe	Oceania
year					
1952	30.000	37.579	28.801	43.585	69.120
1957	31.570	40.696	30.332	48.079	70.260
1962	32.767	43.428	31.997	52.098	70.930
1967	34.113	45.032	34.020	54.336	71.100
1972	35.400	46.714	36.088	57.005	71.890
1977	36.788	49.923	31.220	59.507	72.220
1982	38.445	51.461	39.854	61.036	73.840
1987	39.906	53.636	40.822	63.108	74.320
1992	23.599	55.089	41.674	66.146	76.330
1997	36.087	56.671	41.763	68.835	77.550
2002	39.193	58.137	42.129	70.845	79.110
2007	39.613	60.916	43.828	71.777	80.204

In [18]:
```
# create a dataframe name it df1 that will include only the required columns for
df1=pd.pivot_table(df,index='year',columns='continent',values='lifeExp',aggfunc='

df1
```

Out[18]:

continent	Africa	Americas	Asia	Europe	Oceania
year					
1952	30.000	37.579	28.801	43.585	69.120
1957	31.570	40.696	30.332	48.079	70.260
1962	32.767	43.428	31.997	52.098	70.930
1967	34.113	45.032	34.020	54.336	71.100
1972	35.400	46.714	36.088	57.005	71.890
1977	36.788	49.923	31.220	59.507	72.220
1982	38.445	51.461	39.854	61.036	73.840
1987	39.906	53.636	40.822	63.108	74.320
1992	23.599	55.089	41.674	66.146	76.330
1997	36.087	56.671	41.763	68.835	77.550
2002	39.193	58.137	42.129	70.845	79.110
2007	39.613	60.916	43.828	71.777	80.204

In [45]:
```
print('I am Preston')
```

```
I am Preston
```

In [19]:
```
#call the pivot_table and specify the values for the different arguments to produ
#(worth 15 points)
```

Exercise 6: Filling missing data (worth 20 points)

In [31]:
```
#consider the following data frame

df = pd.DataFrame([[np.nan, 2, np.nan, 0],
                   [3, 4, np.nan, 1],
                  [np.nan, np.nan, np.nan, 5],
                   [np.nan, 3, np.nan, 4]],
                  columns=list('ABCD'))
#print dataframe
print(df)
```

```
     A    B   C  D
0  NaN  2.0 NaN  0
1  3.0  4.0 NaN  1
2  NaN  NaN NaN  5
3  NaN  3.0 NaN  4
```

In [46]:
```
print('I am Preston')
```

```
I am Preston
```

In [32]:
```
# Replace all NaN elements with 0s. (worth 5 points)
df.fillna(0)
```

Out[32]:

	A	B	C	D
0	0.0	2.0	0.0	0
1	3.0	4.0	0.0	1
2	0.0	0.0	0.0	5
3	0.0	3.0	0.0	4

In [47]:
```
print('I am Preston')
```

```
I am Preston
```

In [22]:
```
#propagate non-null values forward (worth 5 points)
df.fillna( method ='ffill')
```

Out[22]:

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	3.0	4.0	NaN	5
3	3.0	3.0	NaN	4

In [48]:
```
print('I am Preston')
```

```
I am Preston
```

In [30]:
```
#Only replace the first NaN element with 0 (worth 5 points)

df.fillna((0),limit=1)
```

Out[30]:

	A	B	C	D
0	0.0	2.0	0.0	0
1	3.0	4.0	NaN	1
2	NaN	0.0	NaN	5
3	3.0	4.0	NaN	1
4	3.0	4.0	0.0	1

In [49]:
```
print('I am Preston')
```

```
I am Preston
```

In [24]:
```
#replace missing values with the column mean (worth 5 points)
values={'A':'mean','B':'mean','C':'mean','D':'mean'}
df.fillna(value=values)
```

Out[24]:

	A	B	C	D
0	mean	2.0	mean	0
1	3.0	4.0	mean	1
2	mean	mean	mean	5
3	mean	3.0	mean	4

In [50]:
```
print('I am Preston')
```

```
I am Preston
```

Exercise 7:Dropna (worth 15 points)

In [51]:
```
#following steps (worth 5 points)
#import required libraries



#create a dataframe df
df = pd.DataFrame([[np.nan, 2, np.nan, 0], [3, 4, np.nan, 1], [np.nan, np.nan, np
                   columns=list('ABCD'))

#print dataframe
print(df)
```

```
     A    B    C  D
0  NaN  2.0  NaN  0
1  3.0  4.0  NaN  1
2  NaN  NaN  NaN  5
3  3.0  4.0  NaN  1
4  3.0  4.0  0.0  1
```

In [52]:
```
print('I am Preston')
```

```
I am Preston
```

In [26]:
```
#remve all column where all value is 'NaN' exists (worth 5 points)

df.dropna(axis=1, how='all')
```

Out[26]:

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	NaN	NaN	5
3	3.0	4.0	NaN	1
4	3.0	4.0	0.0	1

In [53]:
```
print('I am Preston')
```

```
I am Preston
```

In [27]:
```
#remve all column if there is non-'NaN' value is less than 2 (worth 5 points)

df.dropna(thresh=2)
```

Out[27]:

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
3	3.0	4.0	NaN	1
4	3.0	4.0	0.0	1

In [54]:
```
print('I am Preston')
```

```
I am Preston
```

Exericse 8:Replacing values (worth 5 points)

In [28]:
```
#consider the following
data = pd.Series([1., 100., 2.,100., -1000., 3.])
data
```

Out[28]:
```
0       1.0
1     100.0
2       2.0
3     100.0
4   -1000.0
5       3.0
dtype: float64
```

In [55]:
```
print('I am Preston')
```

```
I am Preston
```

In [29]:
```
#replace all 100. by nan (worth 5 points)

data.replace(100, np.nan)
```

Out[29]:
```
0       1.0
1       NaN
2       2.0
3       NaN
4   -1000.0
5       3.0
dtype: float64
```

In [56]:
```
print('I am Preston')
```

```
I am Preston
```

In [ ]: