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 [7]:
```
#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[7]:

	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 [9]:
```
#following instruction worth 5 points
#Visually inspect our dataframe using tail method
df.tail()
```

Out[9]:

	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 [11]:
```
#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 column in df.columns:
    print(f"'{column}'")
```

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

In [13]:
```
#display the number of rows and columns(worth 5 points)
rows, columns = df.shape
print(f"Number of rows: {rows}")
print(f"Number of columns: {columns}")
```

```
Number of rows: 1704
Number of columns: 6
```

In [15]:
```
#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
```

Exercise 2: Changing column datatype (worth 15 points)

In [17]:
```
##print a list of all column names and their data types of your dataframe df (worth
print(df.dtypes)
```

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

In [19]:
```
#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
```

Exercise 3 Pivoting (worth 30 points)

In [21]:
```
#The following steps (worth 1o points)
#import the required libraries
import pandas as pd


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

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

Out[21]:

	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 [23]:
```
# select two columns continent and lifeExp from gapminder dataframe and assign it t
df = gapminder[['continent', 'lifeExp']]
df.head()
```

Out[23]:

	continent	lifeExp
0	Asia	28.801
1	Asia	30.332
2	Asia	31.997
3	Asia	34.020
4	Asia	36.088

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

In [35]:
```
#type your line of code here
pivot_table = pd.pivot_table(df, values='lifeExp', columns='continent', aggfunc='me
pivot_table
```

Out[35]:

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

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 [38]:
```
#create a data frame df that includes only required columns as mentioned in the exe
df = gapminder[['country', 'lifeExp']]


#call the pivot_table method that will provide you with mean lifeExp for each count
pivot_table_countries = pd.pivot_table(df, values='lifeExp', index='country', aggfu
pivot_table_countries
```

Out[38]:

country	lifeExp
Afghanistan	37.478833
Albania	68.432917
Algeria	59.030167
Angola	37.883500
Argentina	69.060417
...	...
Vietnam	57.479500
West Bank and Gaza	60.328667
Yemen Rep.	46.780417
Zambia	45.996333
Zimbabwe	52.663167

142 rows × 1 columns

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

 title

In [44]:
```
# create a dataframe name it df1 that will include only the required columns for ex
df1 = gapminder[['year', 'continent', 'lifeExp']]
```

In [46]:
```
#call the pivot_table and specify the values for the different arguments to produce
#(worth 15 points)
pivot_table_min_lifeExp = pd.pivot_table(df1, values='lifeExp', index='year', colum
pivot_table_min_lifeExp
```

Out[46]:

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

Exercise 6: Filling missing data (worth 20 points)

In [52]:
```
#consider the following data frame

import numpy as np
import pandas as pd

# Create the DataFrame with missing values
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 the 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 [54]:
```
# Replace all NaN elements with 0s. (worth 5 points)
df_filled = df.fillna(0)

print(df_filled)
```

```
     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 [56]:
```
#propagate non-null values forward (worth 5 points)
df_forward_filled = df.fillna(method='ffill')

print(df_forward_filled)
```

```
     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 [58]:
```
#Only replace the first NaN element with 0 (worth 5 points)
df_first_nan_filled = df.fillna(0, limit=1)

print(df_first_nan_filled)
```

```
     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  NaN  3.0  NaN  4
```

In [60]:
```
#replace missing values with the column mean (worth 5 points)
df_mean_filled = df.apply(lambda x: x.fillna(x.mean()), axis=0)

print(df_mean_filled)
```

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

Exercise 7:Dropna (worth 15 points)

In [66]:
```
#following steps (worth 5 points)
#import required libraries
import numpy as np
import pandas as pd


#create a dataframe df
df = pd.DataFrame([[np.nan, 2, np.nan, 0], [3, 4, np.nan, 1], [np.nan, np.nan, np.n
                   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 [68]:
```
#remve all column where all value is 'NaN' exists (worth 5 points)
df_cleaned = df.dropna(axis=1, how='all')

print(df_cleaned)
```

```
     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 [70]:
```
#remve all column if there is non-'NaN' value is less than 2 (worth 5 points)
df_filtered = df.dropna(axis=1, thresh=2)

print(df_filtered)
```

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

Exericse 8:Replacing values (worth 5 points)

In [19]:
```
#consider the following
data = pd.Series([1., 100., 2.,100., -1000., 3.])
data
```

Out[19]:
```
0       1.0
1     100.0
2       2.0
3     100.0
4   -1000.0
5       3.0
dtype: float64
```

In [72]:
```
#replace all 100. by nan (worth 5 points)
import pandas as pd
import numpy as np

data = pd.Series([1., 100., 2., 100., -1000., 3.])

data_replaced = data.replace(100., np.nan)

print(data_replaced)
```

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

In [ ]: