Homework 2

Before you start: Read Chapter 3 Data Visualization and Chapter 4 Dimension Reduction in the textbook.

Note: Please make sure your plots are complete and presentable with a title, proper axis names, labels and legends if applicable.

Please enter the code along with your comments in the **TODO** sections.

Please refer to the **Hint** section if you do not know where to start.

Alternative solutions are welcomed.

Part 1: Advanced Data Visualization

Problem 1

Dataset: Mismanaged waste

Introduction: Jambeck et al. quantified municipal and plastic waste streams from coastal populations in 2010 with projections to the year 2025. The authors define mismanaged and inadequately managed waste as follows: "mismanaged waste is material that is either littered or inadequately disposed. Inadequately disposed waste is not formally managed and includes disposal in dumps or open, uncontrolled landfills, where it is not fully contained. Mismanaged waste could eventually enter the ocean via inland waterways, wastewater outflows, and transport by wind or tides. "

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
!pip install --upgrade openpyxl
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: openpyxl in /usr/local/lib/python3.8/dist-packages (3.0.10)
Collecting openpyxl
  Downloading openpyxl-3.1.0-py2.py3-none-any.whl (250 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 250.0/250.0 KB 4.7 MB/s eta 0:00:00
Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.8/dist-packages (from openpyxl) (1.1.0)
Installing collected packages: openpyxl
  Attempting uninstall: openpyxl
    Found existing installation: openpyxl 3.0.10
    Uninstalling openpyxl-3.0.10:
      Successfully uninstalled openpyxl-3.0.10
Successfully installed openpyxl-3.1.0
```

```
#Load the dataset
from google.colab import files
file = files.upload()  #upload file into google colab session
```

```
Choose Files mismanage...bal-total.csv
• mismanaged-waste-global-total.csv(text/csv) - 4916 bytes, last modified: 2/8/2023 - 100% done
Saving mismanaged-waste-global-total.csv to mismanaged-waste-global-total.csv
```

```
df = pd.read_csv("mismanaged-waste-global-total.csv")
df.head()
```

	Entity	Code	Year	Mismanaged waste (% global total) (% of global total)
0	Albania	ALB	2010	0.0933
1	Algeria	DZA	2010	1.6347
2	Angola	AGO	2010	0.1964
3	Anguilla	AIA	2010	0.0002
4	Antigua and Barbuda	ATG	2010	0.0039

```
df["Year"].unique()
```

```
array([2010])
```

TODO1:

- Use a choropleth map to present the amount of mismanaged waste by country *(Highlight only the top 5 countries)*
- The label (hover) should include the country name and percentage of mismanaged waste
- Interpret your key findings from the map graph
- Considering the manufacturing volume of each country, is this graph misleading?

```
df.describe()
```

	Year	Mismanaged waste (% global total) (% of global total)
count	186.0	186.000000
mean	2010.0	0.537634
std	0.0	2.298779
min	2010.0	0.000000
25%	2010.0	0.006850
50%	2010.0	0.050200
75%	2010.0	0.208175
max	2010.0	27.696600

```
df.dtypes
```

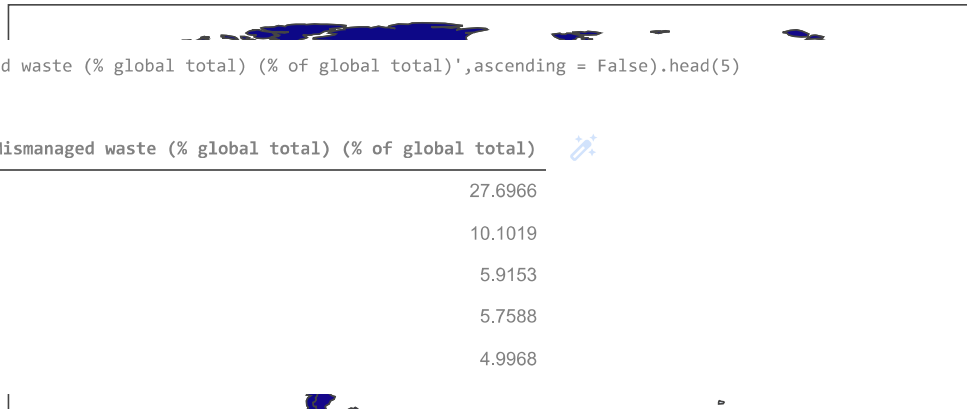
```
Entity                                                 object
Code                                                   object
Year                                                    int64
Mismanaged waste (% global total) (% of global total)    float64
dtype: object
```

```
df["Mismanaged waste (% global total) (% of global total)"].sum()
```

```
100.0
```

```
fig = px.choropleth(df, locations="Code",
                    color="Mismanaged waste (% global total) (% of global total)",
                    hover_name="Entity",
                    color_continuous_scale=px.colors.sequential.Plasma)
fig.update_layout(
    title_text='All countries in Mismanaged waste (% global total)'
)
fig.show()
```

All countries in Mismanaged waste (% global total)

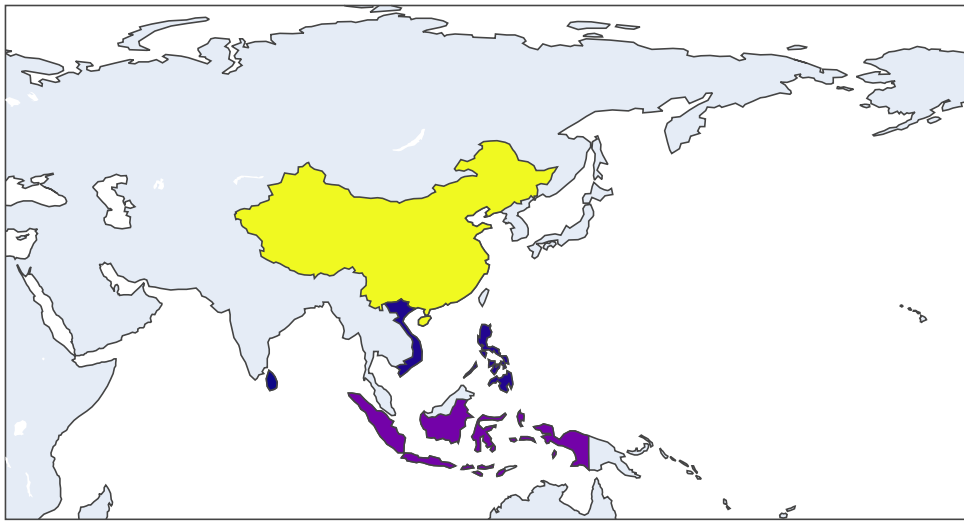


```
df_top = df.sort_values('Mismanaged waste (% global total) (% of global total)',ascending = False).head(5)
df_top
```

	Entity	Code	Year	Mismanaged waste (% global total) (% of global total)
28	China	CHN	2010	27.6966
80	Indonesia	IDN	2010	10.1019
134	Philippines	PHL	2010	5.9153
184	Vietnam	VNM	2010	5.7588
161	Sri Lanka	LKA	2010	4.9968

```
fig = px.choropleth(df_top, locations="Code",
                    color="Mismanaged waste (% global total) (% of global total)",
                    hover_name="Entity",
                    color_continuous_scale=px.colors.sequential.Plasma)
fig.update_layout(
    title_text='Top 5 countries in Mismanaged waste (% global total)'
)
fig.show()
```

Top 5 countries in Mismanaged waste (% global total)



- As we can see from the above China has the highest percentage of Mismanaged waste followed by Indonesia and so on can be observed from the graph
- But these results and visualizations dont take into accont the extent of size of a country, as the country with large area and population tend to produce more waste and hence it can reflect in the Mismanaged %.
- So its is not a good practice to only take the percentage of Mismanaged waste rather its better to take percentage density w.r.to Population or area of a country.

Hint:

- The variable "code" contains three letters ISO country codes.
- Use the built-in country code to create a choropleth map.

Problem 2

Dataset: Plastic disposal dataset

Information: Plastic disposal dataset methods shows how has global plastic waste disposal method changed over time. In the chart we see the share of global plastic waste that is discarded, recycled or incinerated from 1980 through to 2015.

```
#Load the dataset
from google.colab import files
file = files.upload()  #upload file into google colab session
```

```
Choose Files activity.xlsx
• activity.xlsx(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 10161 bytes, last modified: 2/8/2023 - 100% done
Saving activity.xlsx to activity.xlsx
```

```
df = pd.read_excel("activity.xlsx")
df.head()
```

	year	Value	Type
0	1960	88.1	Generation
1	1960	NaN	Composting*
2	1960	5.6	Recycling
3	1960	0.0	Combustion with energy recovery
4	1960	82.5	Landfilling and other disposal

```
df[df["year"] == 1960]
```

	year	Value	Type
0	1960	88.1	Generation
1	1960	NaN	Composting*
2	1960	5.6	Recycling
3	1960	0.0	Combustion with energy recovery
4	1960	82.5	Landfilling and other disposal

```
df.describe()
```

	year	Value
count	50.000000	47.000000
mean	1996.300000	89.961702
std	19.582635	83.080243
min	1960.000000	0.000000
25%	1980.000000	26.050000
50%	2002.500000	65.300000
75%	2015.000000	139.400000
max	2017.000000	267.800000

```
for col in df.columns:
  print(str(col)," ", df[col].unique())
```

```
year   [1960 1970 1980 1990 2000 2005 2010 2015 2016 2017]
Value   [ 88.1   nan   5.6   0.   82.5 121.1   8.    0.5 112.6 151.6  14.5   2.8
 134.3 208.3   4.2  29.   29.8 145.3 243.5  16.5  53.   33.7 140.3 253.7
  20.6  59.2  31.7 142.2 251.1  20.2  65.3  29.3 136.3 262.1  23.4  67.6
  33.5 137.6 266.8  25.1  68.6  33.9 139.2 267.8  27.   67.2  34.  139.6]
Type   ['Generation' 'Composting*' 'Recycling' 'Combustion with energy recovery'
 'Landfilling and other disposal']
```

```
df.isna().sum()
```

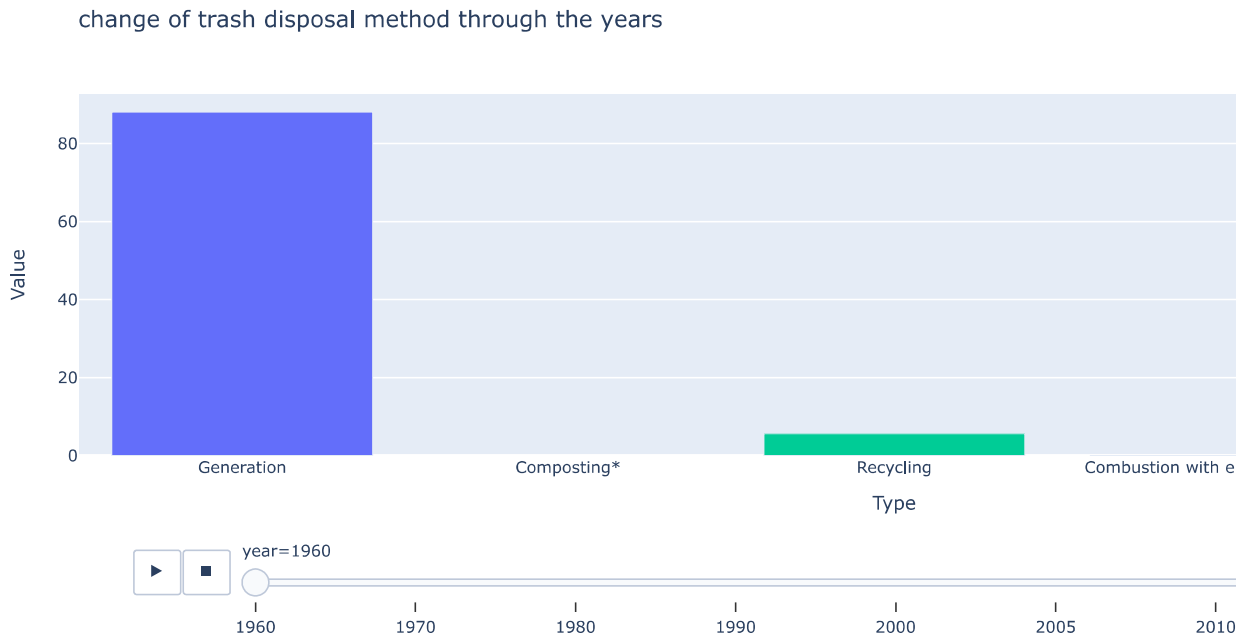
```
year     0
Value    3
```

```
Type     0
dtype: int64
```

TODO1:

- Use an animated bar chart to indicate the change of trash disposal method through the years
- Interpret your key findings from the graph

```
fig = px.bar(df, x="Type", y="Value", color="Type",
  animation_frame="year", animation_group="Type", title="change of trash disposal method through the years")
fig.show()
```



- From the graph we can see that initially at 1960 only methods of handling the waste are "Recycling" and "Landfilling and other disposal"
- After 1990 "Composting" is adopted and rate of usage of "Landfilling" method is decreased steadily
- But main thing we can observe from the graph is there is no plevalent and disposal method that can universally solve our disposal problem permanently

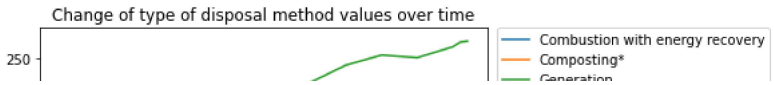
Hint: Animated Bar Charts with Plotly Express

TODO2:

- Suggest and show a better way to visualize the data (choose the most approporiate visualization for this use case)

```
data = df.pivot_table(values="Value", index="year", columns="Type")
data.plot()
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left', borderaxespad=0)
plt.xlabel('Year')
plt.ylabel('Value')
plt.title("Change of type of disposal method values over time")
```

```
Text(0.5, 1.0, 'Change of type of disposal method values over time')
```



Problem 3

Dataset: Global Fortune 500

Introduction: Fortune Global 500 List is a list of largest corporations worldwide which are measured by their total fiscal year revenues. Companies rankings sorted by total revenues for their respective fiscal years ended on or before March 31 of the relevant year.


```
#Upgrade the package "plotly" before you start to avoid future syntax error
#You only need to upgrade it once
!pip install plotly --upgrade
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: plotly in /usr/local/lib/python3.8/dist-packages (5.5.0)
Collecting plotly
  Downloading plotly-5.13.0-py2.py3-none-any.whl (15.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15.2/15.2 MB 68.6 MB/s eta 0:00:00
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.8/dist-packages (from plotly) (8.1.0)
Installing collected packages: plotly
  Attempting uninstall: plotly
    Found existing installation: plotly 5.5.0
    Uninstalling plotly-5.5.0:
      Successfully uninstalled plotly-5.5.0
Successfully installed plotly-5.13.0
WARNING: The following packages were previously imported in this runtime:
  [_plotly_utils,plotly]
You must restart the runtime in order to use newly installed versions.
```

RESTART RUNTIME

```
#Import packages
import pandas as pd
import numpy as np
import plotly.express as px

#Load the dataset
from google.colab import files
file = files.upload()  #upload file into google colab session
```

Choose Files Global Fortune 500.csv

- **Global Fortune 500.csv**(text/csv) - 37106 bytes, last modified: 2/8/2023 - 100% done

```
Saving Global Fortune 500.csv to Global Fortune 500.csv
```

```
df = pd.read_csv("Global Fortune 500.csv")
df.head()
```

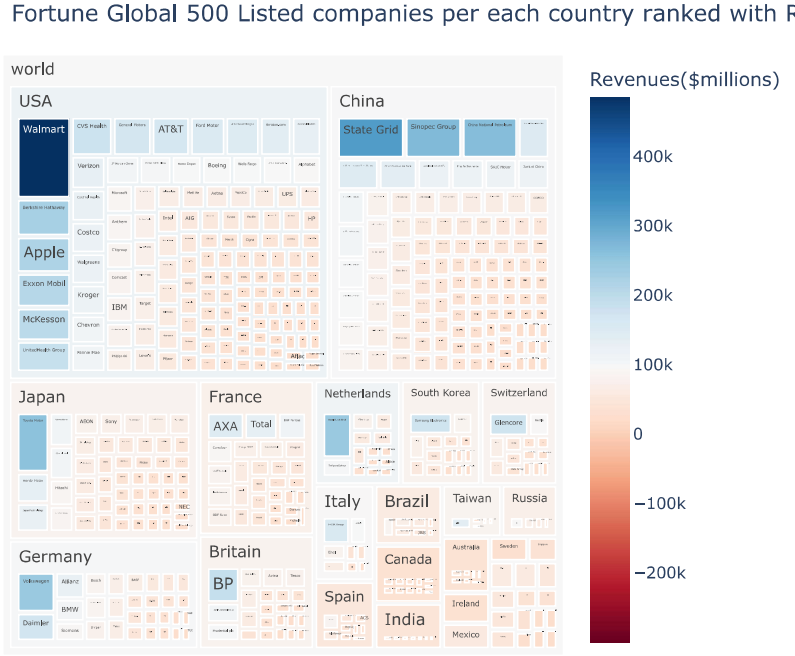
	Rank	Company Name	Country	Number of Employees	Previous Rank	Revenues($millions)	Revenue Change	Profits($millions)	Profit Change	Assets($millions
0	1	Walmart	USA	2,300,000	1	485873	0.80%	13643	-7.20%	19882
1	2	State Grid	China	926,067	2	315199	-4.40%	9571.3	-6.20%	48983
2	3	Sinopec Group	China	713,288	4	267518	-9.10%	1257.9	-65.00%	31072
3	4	China National	China	1.512.048	3	262573	-12.30%	1867.5	-73.70%	58561

TODO1:

- Build a treemap of the companies with "country" as the first hierarchy and "company" as the second hierarchy (Only show top 5 businesses for each country)
- The size of each block should indicate the corresponding company's revenue
- Interpret your key findings from the treemap

```
fig = px.treemap(df, path=[px.Constant("world"), 'Country', 'Company Name'], values='Revenues($millions)',
                 color='Revenues($millions)', hover_data=['Revenues($millions)'],
                 color_continuous_scale='RdBu',
```

```
                  color_continuous_midpoint=np.average(df['Revenues($millions)'], weights=df['Revenues($millions)']))
fig.update_layout(margin = dict(t=50, l=25, r=25, b=25),
                  title = "Fortune Global 500 Listed companies per each country ranked with Revenues"
                  )
fig.show()
```



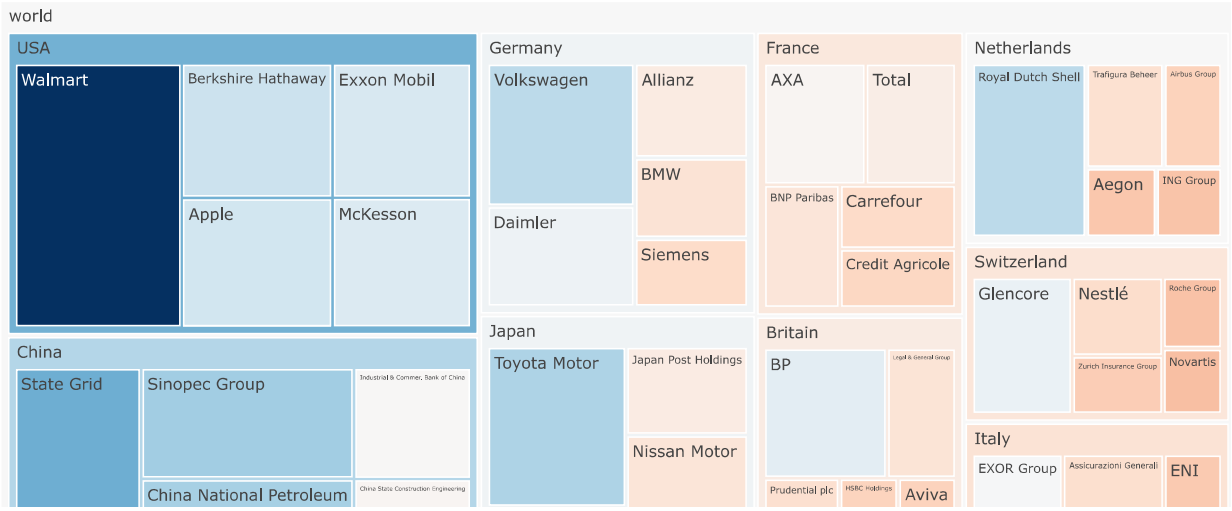

```
df_top = df.groupby(["Country"]).apply(lambda grp: grp.nlargest(5, 'Revenues($millions)'))
df_top.index = df_top.index.droplevel()

df_top
```

	Rank	Company Name	Country	Number of Employees	Previous Rank	Revenues($millions)	Revenue Change	Profits($millions)	Profit Change	Assets(
197	198	Wesfarmers	Australia	220,000	171	48003	-7.60%	296.1	-85.40%	
217	218	Woolworths	Australia	205,000	176	43925	-13.20%	-898.3	-150.30%	
332	333	Commonwealth Bank	Australia	45,129	269	32287	-14.30%	6712.9	-10.90%	
349	350	BHP Billiton	Australia	26,827	168	30912	-40.90%	-6385	-434.30%	
390	391	Westpac Banking	Australia	35,280	336	27704	-10.80%	5477	-12.70%	
...	...	...	...	...	...	...	...	...	...	
7	8	Berkshire Hathaway	USA	367,700	11	223604	6.10%	24074	-	
8	9	Apple	USA	116,000	9	215639	-7.70%	45687	-14.40%	
9	10	Exxon Mobil	USA	72,700	6	205004	-16.70%	7840	-51.50%	
10	11	McKesson	USA	64,500	12	198533	3.10%	5070	124.50%	

```
fig = px.treemap(df_top, path=[px.Constant("world"), 'Country', 'Company Name'], values='Revenues($millions)',
                  color='Revenues($millions)', hover_data=['Revenues($millions)'],
                  color_continuous_scale='RdBu',
                  color_continuous_midpoint=np.average(df_top['Revenues($millions)'], weights=df_top['Revenues($millions)']))
fig.update_layout(margin = dict(t=50, l=25, r=25, b=25),
                  title = "Fortune Global 500 Listed companies per each country ranked with Top 5 Revenues"
                  )
fig.show()
```

Fortune Global 500 Listed companies per each country ranked with Top 5 Revenues



- We can see that USA holds highest ranking with top 5 revenues then comes china and Germany so on...
- We can also view the companies and their respective revenues by hovering on them
  - For example we view details of Walmart like revenue
- We can also explore all companies from a particular by clicking on them
- These features can help to create a dynamic and interactive real-time Visualazation pipeline

Hint: Build a treemap with Plotly

Problem 4

Dataset: Air Quality

Introduction: The dataset contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. The device was located on the field in a significantly polluted area, at road level,within an Italian city. Data were recorded from March 2004 to February 2005 (one year)representing the longest freely available recordings of on field deployed air quality chemical sensor devices responses. Ground Truth hourly averaged concentrations for CO, Non Metanic Hydrocarbons, Benzene, Total Nitrogen Oxides (NOx) and Nitrogen Dioxide (NO2) and were provided by a co-located reference certified analyzer. Evidences of cross-sensitivities as well as both concept and sensor drifts are present as described in De Vito et al., Sens. And Act. B, Vol. 129,2,2008 (citation required) eventually affecting sensors concentration estimation capabilities. Missing values are tagged with -200 value.

```
#Import required libraries
import scipy.stats as stats
from sklearn import preprocessing
%matplotlib inline

#Load the dataset
from google.colab import files
file = files.upload()  #upload file into google colab session
```

```
Choose Files  Air Quality.xlsx
• Air Quality.xlsx(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 1183115 bytes, last modified: 2/8/2023 - 100% done
Saving Air Quality.xlsx to Air Quality.xlsx
```

```
df = pd.read_excel("Air Quality.xlsx")
df.head()
```

Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT0
2004-											

```
df.dtypes
```

```
Date             datetime64[ns]
Time                     object
CO(GT)                  float64
PT08.S1(CO)             float64
NMHC(GT)                  int64
C6H6(GT)                float64
PT08.S2(NMHC)           float64
NOx(GT)                 float64
PT08.S3(NOx)            float64
NO2(GT)                 float64
PT08.S4(NO2)            float64
PT08.S5(O3)             float64
T                       float64
RH                      float64
AH                      float64
dtype: object
```

TODO1:

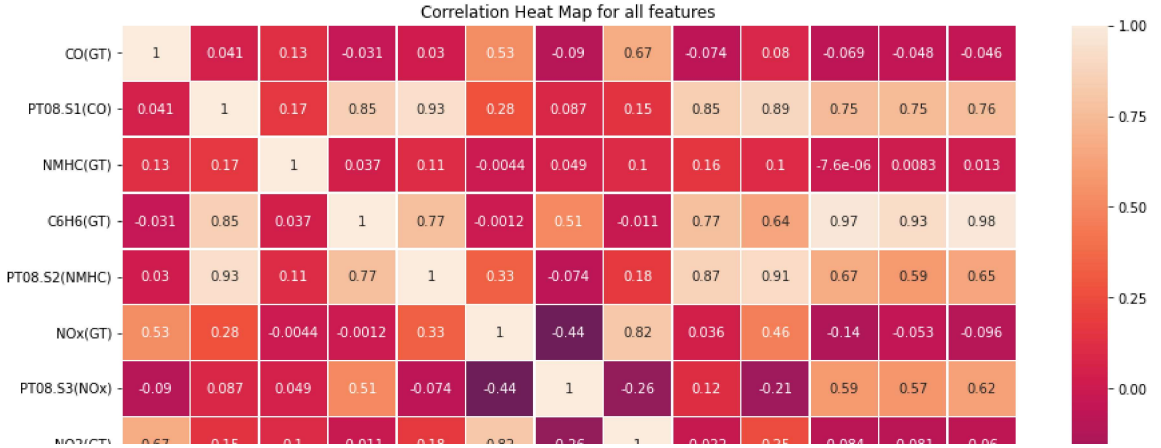
- Plot a correlation heatmap for the Air Quality dataset
- Interpret your key findings from the correlation heatmap (value of correlation should be displayed in the heatmap)

```
df.corr()
```

	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	P
CO(GT)	1.000000	0.041415	0.128351	-0.031377	0.029939	0.526450	-0.089981	0.671140	-0.073721	
PT08.S1(CO)	0.041415	1.000000	0.170009	0.852659	0.933101	0.278029	0.086931	0.154058	0.845133	
NMHC(GT)	0.128351	0.170009	1.000000	0.037329	0.110097	-0.004413	0.048832	0.103345	0.162689	
C6H6(GT)	-0.031377	0.852659	0.037329	1.000000	0.767401	-0.001163	0.512154	-0.010971	0.774649	
PT08.S2(NMHC)	0.029939	0.933101	0.110097	0.767401	1.000000	0.331331	-0.073748	0.176569	0.874761	
NOx(GT)	0.526450	0.278029	-0.004413	-0.001163	0.331331	1.000000	-0.436083	0.817138	0.035580	
PT08.S3(NOx)	-0.089981	0.086931	0.048832	0.512154	-0.073748	-0.436083	1.000000	-0.256217	0.122672	
NO2(GT)	0.671140	0.154058	0.103345	-0.010971	0.176569	0.817138	-0.256217	1.000000	-0.022092	
PT08.S4(NO2)	-0.073721	0.845133	0.162689	0.774649	0.874761	0.035580	0.122672	-0.022092	1.000000	
PT08.S5(O3)	0.080316	0.892436	0.101189	0.641306	0.909909	0.461916	-0.208935	0.253469	0.723670	
T	-0.068952	0.754806	-0.000008	0.971370	0.668984	-0.138457	0.588061	-0.084084	0.755053	
RH	-0.048231	0.745344	0.008288	0.925068	0.585775	-0.053008	0.573513	-0.081300	0.640685	
AH	-0.045892	0.764866	0.012500	0.984556	0.646535	-0.095841	0.621576	-0.060423	0.691889	

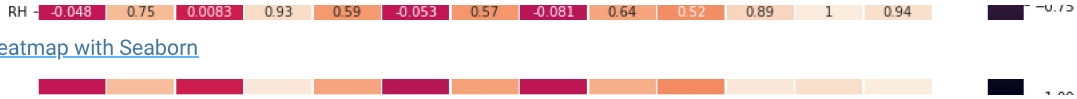
```
plt.figure(figsize = (15,10))
sns.heatmap(df.corr(), annot=True,linewidth=.5, vmin=-1, vmax=1)
plt.title("Correlation Heat Map for all features")
```

```
Text(0.5, 1.0, 'Correlation Heat Map for all features')
```



From the correlation heatmap we can visually intrepret the dependency of one variable on other and which can help us help to reduce the dimesion of data to move forward in creating a Machine learning models.

- For example The variable T and AH have a correlation near to 1 which implies that they are highly corelated and have same kind of effect in the further analysis, so logically we dont need to measure both the values which can save our resourses interms of time and money
- So we can logically and safely remove one of the variables with correlation values near to +1 and -1 depending on the logistics involved



Hint: Build a heatmap with Seaborn

Part 2: Dimension Reduction

Problem 5

please consider the **iris dataset**:

```
#Import the built-in dataset (Wine recognition) for this problem
import sklearn
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
print(sklearn.datasets.load_iris().DESCR)
```

```
**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
```

```
The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

```
#laod the dataset
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.head()
```

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
# laod the vector of target variable
y = iris.target
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

TODO1:

- Determine first two principal component scores for the data set with eigenvalues and eigenvectors on the RAW data (without standardization).
- Note that you are expected to perform matrix multipication, eigen value calculation **only** with the package Numpy, and sorted eigenvalues and eigenvectors in descending order (i.e. $\lambda_0 \geq \lambda_1 \geq \lambda 2 \geq \ldots \geq \lambda_n$ so do the eigenvectors).

```
def eigen(data):                                  # Creation and sortuing of eigen vectors
  eigenValues, eigenVectors = np.linalg.eig(data)
  idx = eigenValues.argsort()[::-1]
  eigenValues = eigenValues[idx]
  eigenVectors = eigenVectors[:,idx]
  print("eigenValues = ", eigenValues,"\n" ,"eigenVectors : \n", eigenVectors)
  return eigenValues, eigenVectors


X = X - X.mean()      ## Mean Adjustment of data set
X.cov()
```

	0	1	2	3
0	0.685694	-0.042434	1.274315	0.516271
1	-0.042434	0.189979	-0.329656	-0.121639
2	1.274315	-0.329656	3.116278	1.295609

```
X.head()
```

	0	1	2	3
0	-0.743333	0.442667	-2.358	-0.999333
1	-0.943333	-0.057333	-2.358	-0.999333
2	-1.143333	0.142667	-2.458	-0.999333
3	-1.243333	0.042667	-2.258	-0.999333
4	-0.843333	0.542667	-2.358	-0.999333

```
eigenValues, eigenVectors = eigen(X.cov())
```

```
eigenValues =  [4.22824171 0.24267075 0.0782095  0.02383509]
 eigenVectors :
 [[ 0.36138659 -0.65658877 -0.58202985  0.31548719]
 [-0.08452251 -0.73016143  0.59791083 -0.3197231 ]
 [ 0.85667061  0.17337266  0.07623608 -0.47983899]
 [ 0.3582892   0.07548102  0.54583143  0.75365743]]
```

```
variance_captured_each = (eigenValues)/sum(eigenValues)
variance_captured_each
```

```
array([0.92461872, 0.05306648, 0.01710261, 0.00521218])
```

- Here we can see the proportion of Variance captured by each component
- For example PC1 captures 92.46% of total variance
- Similarly PC2 captures 5.3% of total variance

```
variance_captured = np.cumsum(eigenValues)/sum(eigenValues)
variance_captured
```

```
array([0.92461872, 0.97768521, 0.99478782, 1.        ])
```

- Here we can see the Cumilative of Variance captured by components
- For example PC1 captures 92.46% of total variance
- As such PC1 & PC2 captures 97.76% of total variance and soon...

Depending on the Percentage of variance we require, we can form the transformed df

```
pca_df = pd.DataFrame(np.dot(np.array(X), eigenVectors), columns=["PC1","PC2","PC3","PC4"])
pca_df
```

	PC1	PC2	PC3	PC4
0	-2.684126	-0.319397	-0.027915	0.002262
1	-2.714142	0.177001	-0.210464	0.099027

```
pca_df.drop(["PC3", "PC4"], axis=1, inplace=True)  ### If we want only 2 PC's
pca_df
```

	PC1	PC2
0	-2.684126	-0.319397
1	-2.714142	0.177001
2	-2.888991	0.144949
3	-2.745343	0.318299
4	-2.728717	-0.326755
...	...	...
145	1.944110	-0.187532
146	1.527167	0.375317
147	1.764346	-0.078859
148	1.900942	-0.116628
149	1.390189	0.282661

150 rows × 2 columns

Hint:

1. Eigen value calculation with Numpy
2. Sorting a Numpy array

TODO2:

- Perform the data standardization on the data without using any buit-in functions.
- Determine the first two principal component scores with eigenvaues and eigenvectors on the standardized data by updataing your code form **TODO1**. (i.e. your code from **TODO** one should not be changed; you should only add a cell of code for standardization.)

```
X_std = X.copy()    # Looping over the columns to perform Standardization
for col in X_std.columns:
  X_std[col] = X_std[col]/X_std[col].std()
X_std
```

	0	1	2	3
0	-0.897674	1.015602	-1.335752	-1.311052
1	-1.139200	-0.131539	-1.335752	-1.311052
2	-1.380727	0.327318	-1.392399	-1.311052
3	-1.501490	0.097889	-1.279104	-1.311052
4	-1.018437	1.245030	-1.335752	-1.311052
...	...	...	...	...
145	1.034539	-0.131539	0.816859	1.443994
146	0.551486	-1.278680	0.703564	0.919223
147	0.793012	-0.131539	0.816859	1.050416
148	0.430722	0.786174	0.930154	1.443994
149	0.068433	-0.131539	0.760211	0.788031

150 rows × 4 columns

```
X_std = X.copy()   # We can also perform the standardization on whole dataset by this method
X_std = (X_std - X_std.mean())/X_std.std()
```

```
X_std
```

	0	1	2	3
0	-0.897674	1.015602	-1.335752	-1.311052
1	-1.139200	-0.131539	-1.335752	-1.311052
2	-1.380727	0.327318	-1.392399	-1.311052
3	-1.501490	0.097889	-1.279104	-1.311052
4	-1.018437	1.245030	-1.335752	-1.311052
...	...	...	...	...
145	1.034539	-0.131539	0.816859	1.443994
146	0.551486	-1.278680	0.703564	0.919223
147	0.793012	-0.131539	0.816859	1.050416
148	0.430722	0.786174	0.930154	1.443994
149	0.068433	-0.131539	0.760211	0.788031

150 rows × 4 columns

```
X.head()
```

	0	1	2	3
0	-0.743333	0.442667	-2.358	-0.999333
1	-0.943333	-0.057333	-2.358	-0.999333
2	-1.143333	0.142667	-2.458	-0.999333
3	-1.243333	0.042667	-2.258	-0.999333
4	-0.843333	0.542667	-2.358	-0.999333

```
eigenValues, eigenVectors = eigen(X_std.cov())
```

```
eigenValues =  [2.91849782 0.91403047 0.14675688 0.02071484]
 eigenVectors :
 [[ 0.52106591 -0.37741762 -0.71956635  0.26128628]
 [-0.26934744 -0.92329566  0.24438178 -0.12350962]
 [ 0.5804131  -0.02449161  0.14212637 -0.80144925]
 [ 0.56485654 -0.06694199  0.63427274  0.52359713]]
```

```
variance_captured_each = (eigenValues)/sum(eigenValues)
variance_captured_each
```

```
array([0.72962445, 0.22850762, 0.03668922, 0.00517871])
```

- Here we can see the proportion of Variance captured by each component
- For example PC1 captures 72.96% of total variance
- Similarly PC2 captures 22.85% of total variance

```
variance_captured = np.cumsum(eigenValues)/sum(eigenValues)
variance_captured
```

```
array([0.72962445, 0.95813207, 0.99482129, 1.        ])
```

- Here we can see the Cumilative of Variance captured by components
- For example PC1 captures 72.96% of total variance
- As such PC1 & PC2 captures 95.81% of total variance and soon...
- As such PC1 , PC2, PC3 captures 99.48% of total variance

Depending on the Percentage of variance we require, we can form the transformed df

```
pc_df = pd.DataFrame(np.dot(np.array(X_std), eigenVectors), columns=["PC1","PC2","PC3","PC4"])
pc_df
```

	PC1	PC2	PC3	PC4
0	-2.257141	-0.478424	-0.127280	0.024088
1	-2.074013	0.671883	-0.233826	0.102663
2	-2.356335	0.340766	0.044054	0.028282
3	-2.291707	0.595400	0.090985	-0.065735
4	-2.381863	-0.644676	0.015686	-0.035803
...	...	...	...	...
145	1.864258	-0.385674	0.255418	0.387957
146	1.559356	0.893693	-0.026283	0.219457
147	1.516091	-0.268171	0.179577	0.118773
148	1.368204	-1.007878	0.930279	0.026041
149	0.957448	0.024250	0.526485	-0.162534

150 rows × 4 columns

```
pc_df.drop(["PC3", "PC4"], axis=1, inplace=True)  ### If we want only 2 PC's
pc_df
```

	PC1	PC2
0	-2.257141	-0.478424
1	-2.074013	0.671883
2	-2.356335	0.340766
3	-2.291707	0.595400
4	-2.381863	-0.644676
...	...	...
145	1.864258	-0.385674
146	1.559356	0.893693
147	1.516091	-0.268171
148	1.368204	-1.007878
149	0.957448	0.024250

150 rows × 2 columns

TODO3:

- Use the built-in function called `PCA()` on the raw data to calculate the first two principal components.
- For each of the components, indentify the explained variance, proportion variance, and cummulative proportion of variance.
- Compare your result with **TODO1**, are they same? (You might expect that you will have the same result wilth **TODO1**. However, `PCA()` function automatically does `x-mean()` transformation. Therefore, do not worry that your result from this one is different than the result performed from **TODO1**.)

Hint:

- PCA Python

```
pca = PCA(n_components=2)
pca.fit(X)
explained_variance = pca.explained_variance_ratio_
cum_explained_variance = np.cumsum(explained_variance)
print("Explained captured variance by each component = ", explained_variance)
print("Explained cumulative captured variance by components = ", cum_explained_variance)

print("Explained captured variance with 2 components = ", sum(explained_variance))
```

```
Explained captured variance by each component =  [0.92461872 0.05306648]
Explained cumulative captured variance by components =  [0.92461872 0.97768521]
Explained captured variance with 2 components =  0.977685206318795
```

TODO4:

- Use the built-in function called `preprocessing.StandardScaler` and `PCA` to calculate the first two principal components.
- For each of the components, determine the explained variance, propotion variance, and cummulative proportion of variance.
- Compare your result with **TODO2**, are they same?

Hint: Data Standardization in Python

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X_std = scaler.transform(X)
#print(X_std)
pca1 = PCA(n_components=2)
data = pca1.fit_transform(X_std)
pca_df = pd.DataFrame(data, columns=["PC1","PC2"])

explained_variance = pca1.explained_variance_ratio_
cum_explained_variance = np.cumsum(explained_variance)
print("Explained captured variance by each component = ", explained_variance)
print("Explained cumulative captured variance by components = ", cum_explained_variance)

print("Explained captured variance with 2 components = ", sum(explained_variance))
```

```
Explained captured variance by each component =  [0.72962445 0.22850762]
Explained cumulative captured variance by components =  [0.72962445 0.95813207]
Explained captured variance with 2 components =  0.9581320720000165
```

TODO5:

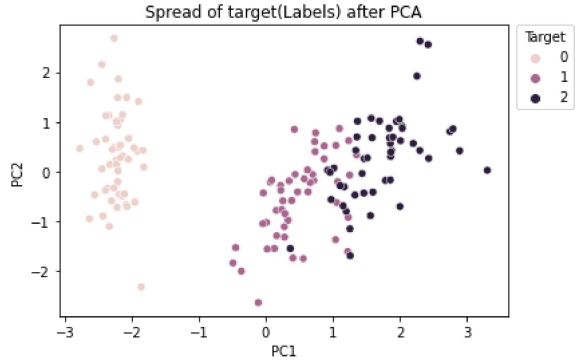
- Integrate from **TODO1** to **TODO4**, why is it important to standardize(normalize) variables before perfroming PCA
- Plot the records on 2D plane defined by the first two PCA components calcualted with standardized data and dffierentiate them using by **target** (i.e. y).
- Make some meaningful interpretation about the plot.

- As we can observe from above values before and after standardization the captured variance is being influenced drastically, can be atributed to the scales of the data of different variables
- With 2 components we had captured 97.7% before standardization but after the process we captured only 95.81% but we have eliminated the bias of scale

Double-click (or enter) to edit

```
sns.scatterplot(data=pca_df, x="PC1", y="PC2", hue=y)
plt.legend(title='Target', bbox_to_anchor=(1.02, 1), loc='upper left', borderaxespad=0)
plt.title("Spread of target(Labels) after PCA")
```

```
Text(0.5, 1.0, 'Spread of target(Labels) after PCA')
```



- As we can see from the plot the respective clusters of each target class clearly, that means the transformed data can sucesscfully classify the targets if we choose to apply any machine learning model and can expect high accuracy

Problem 6

Dataset: Life Expectancy

Introduction: The above dataset gives life expectancy related data for 37 countries in2014.

Consider only the following variables in your analysis: 'GDP', 'Income composition of resources', 'Schooling', and 'Total expenditure'.

```
#Import useful package
from sklearn.manifold import MDS

#Load the dataset
from google.colab import files
file = files.upload()  #upload file into google colab session
```

```
Choose Files  Life Expectancy.csv
• Life Expectancy.csv(text/csv) - 4703 bytes, last modified: 2/8/2023 - 100% done
Saving Life Expectancy.csv to Life Expectancy.csv
```

```
df = pd.read_csv("Life Expectancy.csv")
print(df.shape)
df.head()
```

```
(37, 22)
```

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	...	Polio	T expendi
0	Afghanistan	2014	Developing	59.9	271	64	0.01	73.523582	62.0	492	...	58	
1	Australia	2014	Developed	82.7	6	1	9.71	10769.363050	91.0	340	...	92	
2	Austria	2014	Developed	81.4	66	0	12.32	8350.193523	98.0	117	...	98	
3	Bangladesh	2014	Developing	71.4	132	98	0.01	10.446403	97.0	289	...	97	
4	Belgium	2014	Developed	89.0	76	0	12.60	7163.348923	98.0	70	...	99	

5 rows × 22 columns

```
df1 = df[['GDP', 'Income composition of resources', 'Schooling', 'Total expenditure']]
print(df1.shape)
df1.head()
```

```
(37, 4)
```

	GDP	Income composition of resources	Schooling	Total expenditure
0	612.696514	0.476	10.0	8.18
1	62214.691200	0.936	20.4	9.42
2	51322.639970	0.892	15.9	11.21
3	184.565430	0.570	10.0	2.82
4	47439.396840	0.890	16.3	1.59

```
df1.dtypes
```

```
GDP                              float64
Income composition of resources  float64
Schooling                        float64
Total expenditure                float64
dtype: object
```

TODO1:

- Standardize the numeric variables in the given data frame
- Run MDS() (Multi Dimensional Scaling) on the standardized data
  - **Hint:** n_components = 2

- Plot data points on a 2D plane defined by the first two components
- Use color to differentiate the statues of each country with legend
- Use text label to specify the country name for each point
- Comment your findings from the graph

Hint: MDS Python

```
df1=(df1-df1.mean())/df1.std()   # Standardization
df1.head()
```

	GDP	Income composition of resources	Schooling	Total expenditure
0	-0.634367	-1.546330	-1.142610	0.610704
1	2.151252	1.219384	2.111841	1.003404
2	1.658717	0.954838	0.703665	1.570285
3	-0.653727	-0.981163	-1.142610	-1.086771
4	1.483119	0.942813	0.828836	-1.476304

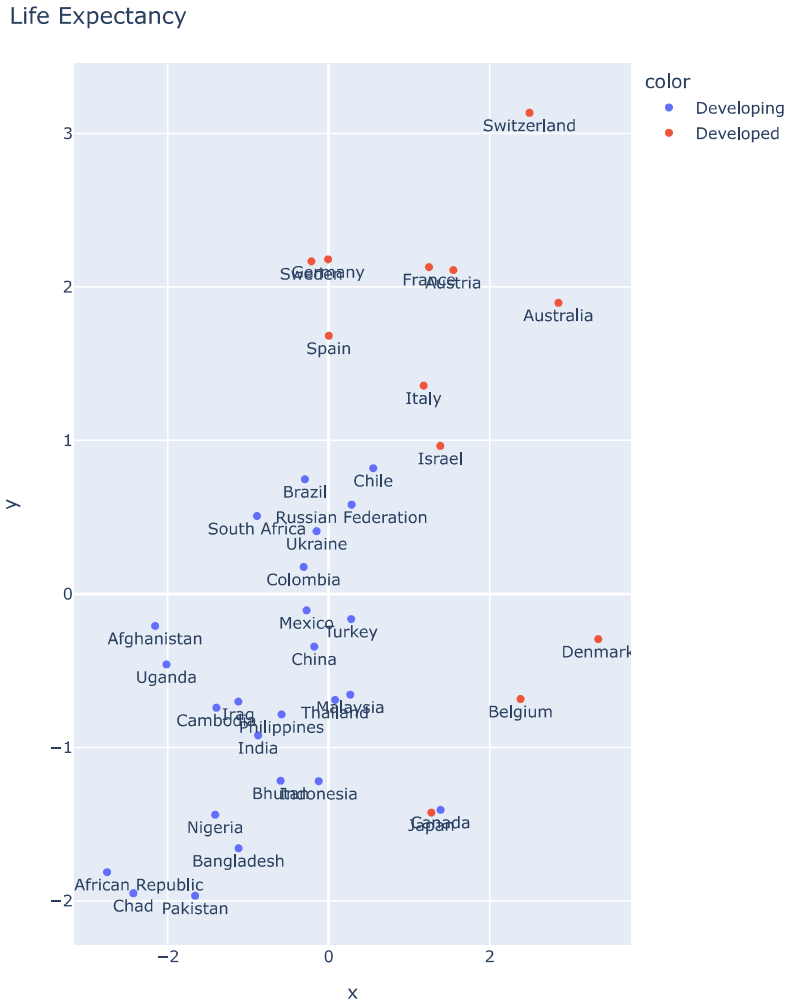
```
std = MDS(n_components=2)
df1_transformed = std.fit_transform(df1)
print(df1_transformed.shape)
df1_transformed
```

```
(37, 2)
array([[-2.15618555e+00, -2.08664704e-01],
       [ 2.85343682e+00,  1.89634011e+00],
       [ 1.54834769e+00,  2.10972421e+00],
       [-1.12018301e+00, -1.65737607e+00],
       [ 2.38292287e+00, -6.84773427e-01],
       [-5.97641692e-01, -1.21769432e+00],
       [-2.95366446e-01,  7.46781217e-01],
       [-1.39487930e+00, -7.41862560e-01],
       [ 1.38971593e+00, -1.40746971e+00],
       [-2.75265918e+00, -1.81273879e+00],
       [-2.42625496e+00, -1.95020591e+00],
       [ 5.53242718e-01,  8.18781146e-01],
       [-1.79685344e-01, -3.43269599e-01],
       [-3.11368052e-01,  1.75679999e-01],
       [ 3.34773054e+00, -2.94818466e-01],
       [ 1.24594576e+00,  2.12920046e+00],
       [-7.52309720e-03,  2.17941225e+00],
       [-8.77334237e-01, -9.21577099e-01],
       [-1.24853140e-01, -1.22019653e+00],
       [-1.12291475e+00, -7.01834331e-01],
       [ 1.38600871e+00,  9.64047156e-01],
       [ 1.17919314e+00,  1.35698087e+00],
       [ 1.27521460e+00, -1.42491777e+00],
       [ 2.67921079e-01, -6.56438605e-01],
       [-2.74825035e-01, -1.07279207e-01],
       [-1.40987518e+00, -1.43885602e+00],
       [-1.65917790e+00, -1.96640779e+00],
       [-5.85610164e-01, -7.85533230e-01],
       [ 2.84599575e-01,  5.81281576e-01],
       [-8.90568242e-01,  5.07123763e-01],
       [ 1.54219570e-03,  1.68196108e+00],
       [-2.15142536e-01,  2.16665591e+00],
       [ 2.49402277e+00,  3.13405256e+00],
       [ 7.89457277e-02, -6.90604775e-01],
       [ 2.78827283e-01, -1.63941552e-01],
       [-2.01497486e+00, -4.59740460e-01],
       [-1.50594740e-01,  4.08178624e-01]])
```

```
fig = px.scatter(df1_transformed, x=df1_transformed.T[0], y=df1_transformed.T[1],text=df["Country"], color=df["Status"])

fig.update_traces(textposition="bottom center")

fig.update_layout(
    height=800,
    title_text='Life Expectancy'
)
fig.show()
```



- As we can see from the plot the respective clusters of each Status class of country clearly, that means the transformed data can sucesscfully classify the targets if we choose to apply any machine learning model and can expect high accuracy.
- And the choosen 'GDP', 'Income composition of resources', 'Schooling', 'Total expenditure' are enough to capture and draw a boundary of classification between Developed and Developing Countries

Problem 7

Dataset: Game of thrones Books

Introduction: If you haven't heard of Game of Thrones, then you must be really good at hiding. Game of Thrones is the hugely popular television series by HBO based on the (also) hugely popular book series A Song of Ice and Fire by George R.R. Martin. You need to analyze the co-occurrence network of the characters in the Game of Thrones books. Here, two characters are considered to co-occur if their names appear in the vicinity of 15 words from one another in the books.

This dataset (5 files attached in zip file) constitutes a network and is given as a text file describing the edges between characters, with some attributes attached to each edge.

Loading the required libraries

```
!pip install pyvis
import pyvis
import networkx as nx
from pyvis.network import Network
!pip install decorator==5.0.9
!pip install --user networkx==2.3
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pyvis
  Downloading pyvis-0.3.1.tar.gz (748 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 748.9/748.9 KB 11.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: jinja2>=2.9.6 in /usr/local/lib/python3.8/dist-packages (from pyvis) (2.11.3)
Requirement already satisfied: networkx>=1.11 in /usr/local/lib/python3.8/dist-packages (from pyvis) (3.0)
Requirement already satisfied: ipython>=5.3.0 in /usr/local/lib/python3.8/dist-packages (from pyvis) (7.9.0)
Collecting jsonpickle>=1.4.1
  Downloading jsonpickle-3.0.1-py2.py3-none-any.whl (40 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40.5/40.5 KB 5.0 MB/s eta 0:00:00
Requirement already satisfied: decorator in /usr/local/lib/python3.8/dist-packages (from ipython>=5.3.0->pyvis) (4.4.2)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.8/dist-packages (from ipython>=5.3.0->pyvis) (57.
Requirement already satisfied: pexpect in /usr/local/lib/python3.8/dist-packages (from ipython>=5.3.0->pyvis) (4.8.0)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.8/dist-packages (from ipython>=5.3.0->pyvis) (5.7.1
Requirement already satisfied: backcall in /usr/local/lib/python3.8/dist-packages (from ipython>=5.3.0->pyvis) (0.2.0)
Requirement already satisfied: prompt-toolkit<2.1.0,>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from ipython>=5.3.0-
Requirement already satisfied: pickleshare in /usr/local/lib/python3.8/dist-packages (from ipython>=5.3.0->pyvis) (0.7.5)
Requirement already satisfied: pygments in /usr/local/lib/python3.8/dist-packages (from ipython>=5.3.0->pyvis) (2.6.1)
Collecting jedi>=0.10
  Downloading jedi-0.18.2-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 48.5 MB/s eta 0:00:00
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dist-packages (from jinja2>=2.9.6->pyvis) (2.0.
Requirement already satisfied: parso<0.9.0,>=0.8.0 in /usr/local/lib/python3.8/dist-packages (from jedi>=0.10->ipython>=5.3
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.8/dist-packages (from prompt-toolkit<2.1.0,>=2.0.0->ipy
Requirement already satisfied: wcwidth in /usr/local/lib/python3.8/dist-packages (from prompt-toolkit<2.1.0,>=2.0.0->ipytho
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.8/dist-packages (from pexpect->ipython>=5.3.0->pyv
Building wheels for collected packages: pyvis
  Building wheel for pyvis (setup.py) ... done
  Created wheel for pyvis: filename=pyvis-0.3.1-py3-none-any.whl size=755850 sha256=d067d1ba27b155304b4e612bfcfa5e729f022e5
  Stored in directory: /root/.cache/pip/wheels/a4/0c/61/8469ca276f96ab772c3acc7f47d71e9737cbdf6f446f017f48
Successfully built pyvis
Installing collected packages: jsonpickle, jedi, pyvis
Successfully installed jedi-0.18.2 jsonpickle-3.0.1 pyvis-0.3.1
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting decorator==5.0.9
  Downloading decorator-5.0.9-py3-none-any.whl (8.9 kB)
Installing collected packages: decorator
  Attempting uninstall: decorator
    Found existing installation: decorator 4.4.2
    Uninstalling decorator-4.4.2:
      Successfully uninstalled decorator-4.4.2
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour i
moviepy 0.2.3.5 requires decorator<5.0,>=4.0.2, but you have decorator 5.0.9 which is incompatible.
Successfully installed decorator-5.0.9
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting networkx==2.3
  Downloading networkx-2.3.zip (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 19.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.8/dist-packages (from networkx==2.3) (5.0.9)
Building wheels for collected packages: networkx
  Building wheel for networkx (setup.py) ... done
  Created wheel for networkx: filename=networkx-2.3-py2.py3-none-any.whl size=1556009 sha256=f1327cd3c87a0d80a6d0ea6de94254
  Stored in directory: /root/.cache/pip/wheels/ff/62/9e/0ed2d25fd4f5761e2d19568cda0c32716556dfa682e65ecf64
Successfully built networkx
Installing collected packages: networkx
Successfully installed networkx-2.3
WARNING: The following packages were previously imported in this runtime:
  [networkx]
```

TODO1:

- Load data for the all the books of Game of Thrones (Merge all files into one and create a dataframe)
- Select only the rows which has weight more than 10

```
from google.colab import files
uploaded = files.upload()
```

Choose Files 5 files

- **book1.csv**(text/csv) - 28344 bytes, last modified: 2/8/2023 - 100% done
- **book2.csv**(text/csv) - 31768 bytes, last modified: 2/8/2023 - 100% done
- **book3.csv**(text/csv) - 41174 bytes, last modified: 2/8/2023 - 100% done
- **book4.csv**(text/csv) - 28973 bytes, last modified: 2/8/2023 - 100% done
- **book5.csv**(text/csv) - 31515 bytes, last modified: 2/8/2023 - 100% done

```
Saving book1.csv to book1.csv
Saving book2.csv to book2.csv
Saving book3.csv to book3.csv
Saving book4.csv to book4.csv
Saving book5.csv to book5.csv
```

TODO2: Load the dataframe as networkx graph

Hint: Network analysis in python

```
df = pd.concat(map(pd.read_csv, ['book1.csv', 'book2.csv','book3.csv', 'book4.csv','book5.csv']))
df.head()
```

	Source	Target	Type	weight	book
0	Addam-Marbrand	Jaime-Lannister	Undirected	3	1.0
1	Addam-Marbrand	Tywin-Lannister	Undirected	6	1.0
2	Aegon-I-Targaryen	Daenerys-Targaryen	Undirected	5	1.0
3	Aegon-I-Targaryen	Eddard-Stark	Undirected	4	1.0
4	Aemon-Targaryen-(Maester-Aemon)	Alliser-Thorne	Undirected	4	1.0

```
df.shape
```

```
(3909, 5)
```

```
df.isna().sum()
```

```
Source    0
Target    0
Type      0
weight    0
book      1
dtype: int64
```

```
df = df[df["weight"]>10]
print(df.shape)
df.head()
```

```
(747, 5)
```

	Source	Target	Type	weight	book
8	Aemon-Targaryen-(Maester-Aemon)	Jeor-Mormont	Undirected	13	1.0
9	Aemon-Targaryen-(Maester-Aemon)	Jon-Snow	Undirected	34	1.0
16	Aerys-II-Targaryen	Robert-Baratheon	Undirected	12	1.0
17	Aggo	Daenerys-Targaryen	Undirected	11	1.0
30	Alliser-Thorne	Jon-Snow	Undirected	32	1.0

```
df["book"].unique()
```

```
array([1., 2., 3., 4., 5.])
```

```
G = nx.from_pandas_edgelist(df, source = 'Source', target = 'Target', edge_attr= 'weight' )
G["Aggo"]
```

```
AtlasView({'Daenerys-Targaryen': {'weight': 11}, 'Jhogo': {'weight': 12}})
```

```
print("No of unique characters:", len(G.nodes))
print("No of connections:", len(G.edges))
```

```
No of unique characters: 258
No of connections: 557
```

TODO3:

Create viz network Hint: Use Pyvis

```
g = Network(height = 800, width = 800, notebook = True)

g.toggle_hide_edges_on_drag(False)

g.from_nx(G)
```

```
display(g.show("ex.html"))
```

```
    Local cdn resources have problems on chrome/safari when used in jupyter-notebook.
```

✓ 0s completed at 8:28 PM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.