IBM - Machine Learning Professional Certificate

Unsupervised Machine Learning

Analysis of Cancer Mortality Ratios by Cancer Type Across Different Countries

JingZeng Xie

TABLE OF CONTENTS

---

1. INTRODUCTION

1.1 - Introduction

This project focuses on analyzing cancer prevalence by cancer type across different countries from 1990 to 2019. We will utilize unsupervised model with various clustering algorithms that are commonly used to segment information, and after evaluating and comparing these clustering algorithms, our goal is to

determine the clustering algorithm that is best suited for our project's unsupervised model.

A comprehensive exploration of the dataset will be conducted to assess its quality. To enhance the statistical significance and stability of our unsupervised model model, we will employ a range of standard data processing techniques to clean and optimize the dataset. Once the data has been refined, we will perform visual analyses to further evaluate its statistical properties.

After preparing the dataset for modeling, we will implement an unsupervised learning approach using several clustering algorithms, including **K-Means**, **Agglomerative Clustering**, **Bisecting K-Means**, **Gaussian Mixture Model**, and **Density-Based Spatial Clustering of Applications with Noise (DBSCAN)**. To evaluate the performance of these algorithms, we will use the davies bouldin, silhouette score and calinski harabasz metrics.

1.2 - Objective

- **Main Objective**: This analysis aims to clarify whether the focus will be on clustering or dimensionality reduction.
- **Dataset Overview**: The study utilizes a dataset that includes the cancer prevalence by cancer type across different countries from 1990 to 2019, summarizing its key attributes and relevance.
- **Data Exploration and Preparation**: A brief overview of the data exploration process is provided, detailing the actions taken for data cleaning and feature engineering to enhance the dataset's quality and applicability.
- **Model Training and Comparison**: We summarize the training of the unsupervised model using various clustering techniques, examining their differences in cohesion and separation.
- **Final Clustering Evaluation**: By employing the silhouette coefficient, we identify the clustering algorithm that best fits the unsupervised model, evaluating both cohesion and separation.
- **Key Findings and Insights**: A summary of the key findings and insights is presented, highlighting the main drivers of the final model and the valuable

insights derived from the dataset through the clustering analysis.

1.3 - Coding Environment

The following required modules are pre-installed in the Skills Network Labs environment. However if you run this notebook commands in a different Jupyter environment (e.g. Watson Studio or Ananconda) you will need to install these libraries by removing the `#` sign before `!mamba` in the code cell below.

In [ ]:
```
# All Libraries required for this lab are listed below. The libraries pre-ir
# !mamba install -qy pandas==1.3.4 numpy==1.21.4 seaborn==0.9.0 matplotlib==
# Note: If your environment doesn't support "!mamba install", use "!pip inst
```

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

from sklearn.preprocessing import StandardScaler, PowerTransformer, MinMaxSc
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer, KNNImputer
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski
from sklearn.cluster import KMeans, AgglomerativeClustering, BisectingKMeans
from sklearn.mixture import GaussianMixture

import scipy
import math
import random

import plotly.express as px
import json
```

---

2. DATA PROCESSING

2.1 - Data Collection

The dataset (Cancer and Deaths Dataset : 1990 to 2019 Globally) provides data on prevalence of people from various countries around the world collected between 1990 and 2019. It contains various types of cancer, which are essential for studying cancer treatment deficiencies in various countries and prevention of special types of cancer. With more than 30 years of statistics, this dataset is a perfect starting point for building machine learning models, analyzing cancer trends, and even developing projects related to cancer prevention.

Each row represents the prevalence of various cancers in a country in that year, including important characteristics of liver, kidney, larynx, breast, thyroid, bladder, uterine and other organ cancers. The data is based on age-standardized prevalence of different cancers in the population of both sexes.

The dataset can be used for:

- Building cancer prediction models
- Exploring cancer patterns
- Studying the prevalence relationship between various countries
- Improving forecast prevalence using machine learning models

In [ ]:
```
# Loading the dataset from local drive
data = pd.read_csv("/content/share-of-population-with-cancer-types.csv")

print(data)
```

```
         Country Code  ...  Gallbladder and biliary tract  Neoplasms
0     Afghanistan  AFG  ...                            0.0   0.476867
1     Afghanistan  AFG  ...                            0.0   0.476258
2     Afghanistan  AFG  ...                            0.0   0.475649
3     Afghanistan  AFG  ...                            0.0   0.475640
4     Afghanistan  AFG  ...                            0.0   0.480281
...           ...  ...  ...                            ...        ...
7699     Zimbabwe  ZWE  ...                            0.0   0.566873
7700     Zimbabwe  ZWE  ...                            0.0   0.567007
7701     Zimbabwe  ZWE  ...                            0.0   0.566521
7702     Zimbabwe  ZWE  ...                            0.0        NaN
7703     Zimbabwe  ZWE  ...                            0.0        NaN

[7704 rows x 25 columns]
```

2.2 - Data Description

The **Feature** Variables:

Features	Feature Type	Description	Data Type
Country	Metadata	Name of the country or region for which cancer prevalence data is reported	Object
Code	Metadata	Code assigned to the country or region for easy identification	Object
Year	Datetime	The year for which the cancer prevalence data is reported	Integer
Liver	Numeric	Prevalence - Liver cancer: Age-standardized prevalence of liver cancer in the population of both sexes	Float
Kidney	Numeric	Prevalence - Kidney cancer: Age-standardized prevalence of kidney cancer in the population of both sexes	Float
Larynx	Numeric	Prevalence - Laryngeal cancer: Age-standardized prevalence of laryngeal cancer in the population of both sexes	Float
Breast	Numeric	Prevalence - Breast cancer: Age-standardized prevalence of breast cancer in the population of both sexes	Float
Thyroid	Numeric	Prevalence - Thyroid cancer: Age-standardized prevalence of thyroid cancer in the population of both sexes	Float
Bladder	Numeric	Prevalence - Bladder cancer: Age-standardized prevalence of bladder cancer in the population of both sexes	Float
Uterine	Numeric	Prevalence - Uterine cancer: Age-standardized prevalence of uterine cancer in the population of both sexes	Float
Ovarian	Numeric	Prevalence - Ovarian cancer: Age-standardized prevalence of ovarian cancer in the population of both sexes	Float
Stomach	Numeric	Prevalence - Stomach cancer: Age-standardized prevalence of stomach cancer in the population of both sexes	Float
Prostate	Numeric	Prevalence - Prostate cancer: Age-standardized prevalence of prostate cancer in the population of both sexes	Float
Cervical	Numeric	Prevalence - Cervical cancer: Age-standardized prevalence of cervical cancer in the population of both sexes	Float
Testicular	Numeric	Prevalence - Testicular cancer: Age-standardized prevalence of testicular cancer in the population of both sexes	Float
Pancreatic	Numeric	Prevalence - Pancreatic cancer: Age-standardized prevalence of pancreatic cancer in the population of both sexes	Float

Features	Feature Type	Description	Data Type
Esophageal	Numeric	Prevalence - Esophageal cancer: Age-standardized prevalence of esophageal cancer in the population of both sexes	Float
Nasopharynx	Numeric	Prevalence - Nasopharyngeal cancer: Age-standardized prevalence of nasopharyngeal cancer in the population of both sexes	Float
Colon and Rectum	Numeric	Prevalence - Colon and rectum cancer: Age-standardized prevalence of colon and rectum cancer in the population of both sexes	Float
Non-Melanoma Skin	Numeric	Prevalence - Non-melanoma skin cancer: Age-standardized prevalence of non-melanoma skin cancer in the population of both sexes	Float
Lip and Oral Cavity	Numeric	Prevalence - Lip and oral cavity cancer: Age-standardized prevalence of lip and oral cavity cancer in the population of both sexes	Float
Brain and Nervous System	Numeric	Prevalence - Brain and nervous system cancer: Age-standardized prevalence of brain and nervous system cancer in the population of both sexes	Float
Tracheal, Bronchus, and Lung	Numeric	Prevalence - Tracheal, bronchus, and lung cancer: Age-standardized prevalence of tracheal, bronchus, and lung cancer in the population of both sexes	Float
Gallbladder and Biliary Tract	Numeric	Prevalence - Gallbladder and biliary tract cancer: Age-standardized prevalence of gallbladder and biliary tract cancer in the population of both sexes	Float

The **Target** Variables:

Target	Feature Type	Description	Data Type
Neoplasms	Numeric	Prevalence - Neoplasms: Age-standardized prevalence of neoplasms in the population of both sexes	Float

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7704 entries, 0 to 7703
Data columns (total 25 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Country                         7704 non-null   object 
 1   Code                            6146 non-null   object 
 2   Year                            7704 non-null   int64  
 3   Liver                           6752 non-null   float64
 4   Kidney                          6753 non-null   float64
 5   Larynx                          6752 non-null   float64
 6   Breast                          6752 non-null   float64
 7   Thyroid                         6753 non-null   float64
 8   Bladder                         6753 non-null   float64
 9   Uterine                         6752 non-null   float64
 10  Ovarian                         6752 non-null   float64
 11  Stomach                         6752 non-null   float64
 12  Prostate                        6752 non-null   float64
 13  Cervical                        6752 non-null   float64
 14  Testicular                      6752 non-null   float64
 15  Pancreatic                      6752 non-null   float64
 16  Esophageal                      6752 non-null   float64
 17  Nasopharynx                     6752 non-null   float64
 18  Colon and rectum                6752 non-null   float64
 19  Non-melanoma skin               6752 non-null   float64
 20  Lip and oral cavity             6752 non-null   float64
 21  Brain and nervous system        6468 non-null   float64
 22  Tracheal, bronchus, and lung    6752 non-null   float64
 23  Gallbladder and biliary tract   6752 non-null   float64
 24  Neoplasms                       6468 non-null   float64
dtypes: float64(22), int64(1), object(2)
memory usage: 1.5+ MB
```

In [ ]:
```
data_describe_metadata = data.describe(include='object')

data_describe_metadata.T
```

Out[ ]:

	count	unique	top	freq
Country	7704	259	Afghanistan	30
Code	6146	205	AFG	30

In [ ]:
```
data_describe_numeric = data.describe(include='number')

data_describe_numeric.T
```

Out[ ]:

	count	mean	std	min	25%	
Year	7704.0	2004.380062	8.594906	1990.000000	1997.000000	2004.00
Liver	6752.0	0.006216	0.009571	0.000000	0.000000	0.00
Kidney	6753.0	0.018140	0.019100	0.000000	0.000000	0.01
Larynx	6752.0	0.014499	0.010441	0.000000	0.010000	0.01
Breast	6752.0	0.235946	0.173355	0.030000	0.100000	0.18
Thyroid	6753.0	0.018359	0.013915	0.000000	0.010000	0.02
Bladder	6753.0	0.032748	0.033899	0.000000	0.010000	0.02
Uterine	6752.0	0.035501	0.030761	0.000000	0.010000	0.03
Ovarian	6752.0	0.015773	0.010625	0.000000	0.010000	0.01
Stomach	6752.0	0.021444	0.015027	0.000000	0.010000	0.02
Prostate	6752.0	0.138701	0.135701	0.010000	0.040000	0.08
Cervical	6752.0	0.053137	0.030654	0.000000	0.030000	0.05
Testicular	6752.0	0.013328	0.022839	0.000000	0.000000	0.00
Pancreatic	6752.0	0.003609	0.004937	0.000000	0.000000	0.00
Esophageal	6752.0	0.007798	0.007898	0.000000	0.000000	0.01
Nasopharynx	6752.0	0.003209	0.006312	0.000000	0.000000	0.00
Colon and rectum	6752.0	0.100933	0.095113	0.010000	0.030000	0.06
Non-melanoma skin	6752.0	0.017315	0.093134	0.000000	0.000000	0.00
Lip and oral cavity	6752.0	0.015296	0.014313	0.000000	0.010000	0.01
Brain and nervous system	6468.0	0.018314	0.022087	0.001409	0.005478	0.00
Tracheal, bronchus, and lung	6752.0	0.029138	0.023480	0.000000	0.010000	0.02
Gallbladder and biliary tract	6752.0	0.001096	0.003259	0.000000	0.000000	0.00
Neoplasms	6468.0	1.035583	0.836737	0.191388	0.467112	0.66

2.3 - Quality Assessment

2.3.1 - Normal Distribution

In Machine Learning, data satisfying **Normal Distribution** is beneficial for model building (**Especially regression based models**).

Models like **Linear Discriminant Analysis (LDA)**, **Gaussian Naive Bayes**, **Logistic Regression**, **Linear Regression**, etc., are explicitly calculated from the assumption that the distribution is a bivariate or multivariate normal.

In [ ]:
```
#-------------------------------------------------------------------------
# The summary of skewness and kurtosis
#-------------------------------------------------------------------------

# Get the data with type numeric
data_numeric = data.select_dtypes(include='number')
# Get the name of numeric column
data_numeric_column = list(data_numeric.columns)

# Get the skewness for numeric column
data_numeric_skew = data_numeric.skew()
# Get the kurtosis for numeric column
data_numeric_kurtosis = data_numeric.kurtosis()

data_normal_summary = pd.DataFrame( zip( data_numeric_column,
                        data_numeric_skew,
                        data_numeric_kurtosis ),
                      columns = [ "Column",
                              "Skewness",
                              "Kurtosis" ] )

data_normal_summary.sort_values(by="Skewness", ascending=False)
```

Out[ ]:

	Column	Skewness	Kurtosis
17	Non-melanoma skin	8.340226	79.977030
1	Liver	4.913082	44.635953
9	Stomach	3.767467	28.412286
18	Lip and oral cavity	3.144313	16.099572
21	Gallbladder and biliary tract	2.866245	7.515276
12	Testicular	2.678474	11.774667
19	Brain and nervous system	2.525787	7.338523
15	Nasopharynx	2.444365	7.499899
22	Neoplasms	2.026593	5.262395
3	Larynx	1.892118	7.706033
6	Bladder	1.808741	3.527994
11	Cervical	1.589112	6.551106
20	Tracheal, bronchus, and lung	1.366325	1.523893
10	Prostate	1.365805	1.073375
2	Kidney	1.314012	1.262953
16	Colon and rectum	1.192985	0.292989
5	Thyroid	1.083330	2.336385
4	Breast	1.083254	0.270889
7	Uterine	1.002731	0.278647
14	Esophageal	0.939927	0.963287
8	Ovarian	0.924266	0.932337
13	Pancreatic	0.741173	-1.046257
0	Year	0.004697	-1.196866

Based on the output above, we can conclude that most of the skewness in the numeric features are greater than 1, indicating that the dataset is **non normally distributed**.

The kurtosis values are predominantly positive, suggesting that the dataset is **leptokurtic**.

Overall, the dataset generally adheres to a non normal distribution, we can considerd to transformation the data.

2.3.2 - Missing Value

Missing Values contain in most of the real world datasets, i.e., feature entries with no data value stored. As many machine learning algorithms do not support missing values, detecting the missing values and properly handling them, can have a significant impact.

In [ ]:
```
# Quick check of missing variables
data.isnull().sum()
```

Out[ ]:

	0
Country	0
Code	1558
Year	0
Liver	952
Kidney	951
Larynx	952
Breast	952
Thyroid	951
Bladder	951
Uterine	952
Ovarian	952
Stomach	952
Prostate	952
Cervical	952
Testicular	952
Pancreatic	952
Esophageal	952
Nasopharynx	952
Colon and rectum	952
Non-melanoma skin	952
Lip and oral cavity	952
Brain and nervous system	1236
Tracheal, bronchus, and lung	952
Gallbladder and biliary tract	952
Neoplasms	1236

dtype: int64

In [ ]:
```
#-------------------------------------------------------------------------
# The summary of missing variables from whole columns
#-------------------------------------------------------------------------

# Get the name of columns
data_column = list(data.columns)
# Get the total rows
data_row_count = np.array([len(data)] * len(data_column))
```

```
# Count of missing variables
data_missing_count = data_row_count - np.array(data.count())
# Missing variables / Total rows
data_missing_rate = np.divide( data_missing_count, data_row_count, out=np.ze

data_missing_summary = pd.DataFrame( zip( data_column,
                       data_row_count,
                       data_missing_count,
                       data_missing_rate ),
                     columns = [ "Column",
                             "Rows",
                             "Missing Values",
                             "Missing Rate %" ] )

data_missing_summary.sort_values(by="Missing Values", ascending=False)
```

Out[ ]:

	Column	Rows	Missing Values	Missing Rate %
1	Code	7704	1558	20.223261
24	Neoplasms	7704	1236	16.043614
21	Brain and nervous system	7704	1236	16.043614
13	Cervical	7704	952	12.357217
23	Gallbladder and biliary tract	7704	952	12.357217
22	Tracheal, bronchus, and lung	7704	952	12.357217
20	Lip and oral cavity	7704	952	12.357217
19	Non-melanoma skin	7704	952	12.357217
18	Colon and rectum	7704	952	12.357217
17	Nasopharynx	7704	952	12.357217
16	Esophageal	7704	952	12.357217
15	Pancreatic	7704	952	12.357217
14	Testicular	7704	952	12.357217
12	Prostate	7704	952	12.357217
11	Stomach	7704	952	12.357217
10	Ovarian	7704	952	12.357217
9	Uterine	7704	952	12.357217
6	Breast	7704	952	12.357217
5	Larynx	7704	952	12.357217
3	Liver	7704	952	12.357217
8	Bladder	7704	951	12.344237
7	Thyroid	7704	951	12.344237
4	Kidney	7704	951	12.344237
2	Year	7704	0	0.000000
0	Country	7704	0	0.000000

2.3.3 - Invalid Value

Invalid Values (Badly Formatted Values) refer to inconsistent entries commonly found in datasets, such as variables with different units across data points or incorrect data types. For instance, numerical variables like percentages and fractions are sometimes mistakenly stored as strings. It is essential to detect and

correct these cases to ensure that machine learning algorithms can properly process and analyze the actual numerical values.

In [ ]:
```
data_describe_metadata.T
```

Out[ ]:

	count	unique	top	freq
Country	7704	259	Afghanistan	30
Code	6146	205	AFG	30

In [ ]:
```
data_describe_numeric.T
```

Out[ ]:

	count	mean	std	min	25%	
Year	7704.0	2004.380062	8.594906	1990.000000	1997.000000	2004.00
Liver	6752.0	0.006216	0.009571	0.000000	0.000000	0.00
Kidney	6753.0	0.018140	0.019100	0.000000	0.000000	0.01
Larynx	6752.0	0.014499	0.010441	0.000000	0.010000	0.01
Breast	6752.0	0.235946	0.173355	0.030000	0.100000	0.18
Thyroid	6753.0	0.018359	0.013915	0.000000	0.010000	0.02
Bladder	6753.0	0.032748	0.033899	0.000000	0.010000	0.02
Uterine	6752.0	0.035501	0.030761	0.000000	0.010000	0.03
Ovarian	6752.0	0.015773	0.010625	0.000000	0.010000	0.01
Stomach	6752.0	0.021444	0.015027	0.000000	0.010000	0.02
Prostate	6752.0	0.138701	0.135701	0.010000	0.040000	0.08
Cervical	6752.0	0.053137	0.030654	0.000000	0.030000	0.05
Testicular	6752.0	0.013328	0.022839	0.000000	0.000000	0.00
Pancreatic	6752.0	0.003609	0.004937	0.000000	0.000000	0.00
Esophageal	6752.0	0.007798	0.007898	0.000000	0.000000	0.01
Nasopharynx	6752.0	0.003209	0.006312	0.000000	0.000000	0.00
Colon and rectum	6752.0	0.100933	0.095113	0.010000	0.030000	0.06
Non-melanoma skin	6752.0	0.017315	0.093134	0.000000	0.000000	0.00
Lip and oral cavity	6752.0	0.015296	0.014313	0.000000	0.010000	0.01
Brain and nervous system	6468.0	0.018314	0.022087	0.001409	0.005478	0.00
Tracheal, bronchus, and lung	6752.0	0.029138	0.023480	0.000000	0.010000	0.02
Gallbladder and biliary tract	6752.0	0.001096	0.003259	0.000000	0.000000	0.00
Neoplasms	6468.0	1.035583	0.836737	0.191388	0.467112	0.66

Currently, the data types align with those described in the data documentation, and there are no invalid data types present in the dataset.

2.3.4 - Duplicate Value

Duplicate Values can appear in various forms, such as multiple entries of the same data point, repeated instances of entire columns, or duplication within an ID variable. While duplicates may be valid in some datasets, they often result from errors during data extraction or integration. Therefore, it is crucial to detect these duplicate values and determine whether they represent true duplicates or are a legitimate part of the dataset.

In [ ]:
```
# Quick check of duplicate row
data.duplicated().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
# Quick check of unique value
data_unique_count = data.nunique()

data_unique_count
```

Out[ ]:

	0
Country	259
Code	205
Year	30
Liver	11
Kidney	11
Larynx	9
Breast	83
Thyroid	10
Bladder	23
Uterine	17
Ovarian	7
Stomach	21
Prostate	67
Cervical	26
Testicular	15
Pancreatic	3
Esophageal	6
Nasopharynx	6
Colon and rectum	42
Non-melanoma skin	86
Lip and oral cavity	13
Brain and nervous system	6467
Tracheal, bronchus, and lung	15
Gallbladder and biliary tract	3
Neoplasms	6468

dtype: int64

In [ ]:
```
#-------------------------------------------------------------------------
# The summary of duplicate variables from whole columns
#-------------------------------------------------------------------------

# Unique variables / Total rows
data_duplicate_rate = ( np.ones(len(data_unique_count)) - np.divide( list(da

data_duplicate_summary = pd.DataFrame( zip ( data_column,
                         data_row_count,
```

```
                            data_unique_count,
                            data_duplicate_rate),
                        columns = [ "Column",
                                    "Rows",
                                    "Unique Values",
                                    "Duplicate Rate %" ] )

data_duplicate_summary.sort_values(by="Duplicate Rate %", ascending=False)
```

Out[ ]:

	Column	Rows	Unique Values	Duplicate Rate %
23	Gallbladder and biliary tract	7704	3	99.961059
15	Pancreatic	7704	3	99.961059
17	Nasopharynx	7704	6	99.922118
16	Esophageal	7704	6	99.922118
10	Ovarian	7704	7	99.909138
5	Larynx	7704	9	99.883178
7	Thyroid	7704	10	99.870197
3	Liver	7704	11	99.857217
4	Kidney	7704	11	99.857217
20	Lip and oral cavity	7704	13	99.831256
14	Testicular	7704	15	99.805296
22	Tracheal, bronchus, and lung	7704	15	99.805296
9	Uterine	7704	17	99.779335
11	Stomach	7704	21	99.727414
8	Bladder	7704	23	99.701454
13	Cervical	7704	26	99.662513
2	Year	7704	30	99.610592
18	Colon and rectum	7704	42	99.454829
12	Prostate	7704	67	99.130322
6	Breast	7704	83	98.922638
19	Non-melanoma skin	7704	86	98.883697
1	Code	7704	205	97.339045
0	Country	7704	259	96.638110
21	Brain and nervous system	7704	6467	16.056594
24	Neoplasms	7704	6468	16.043614

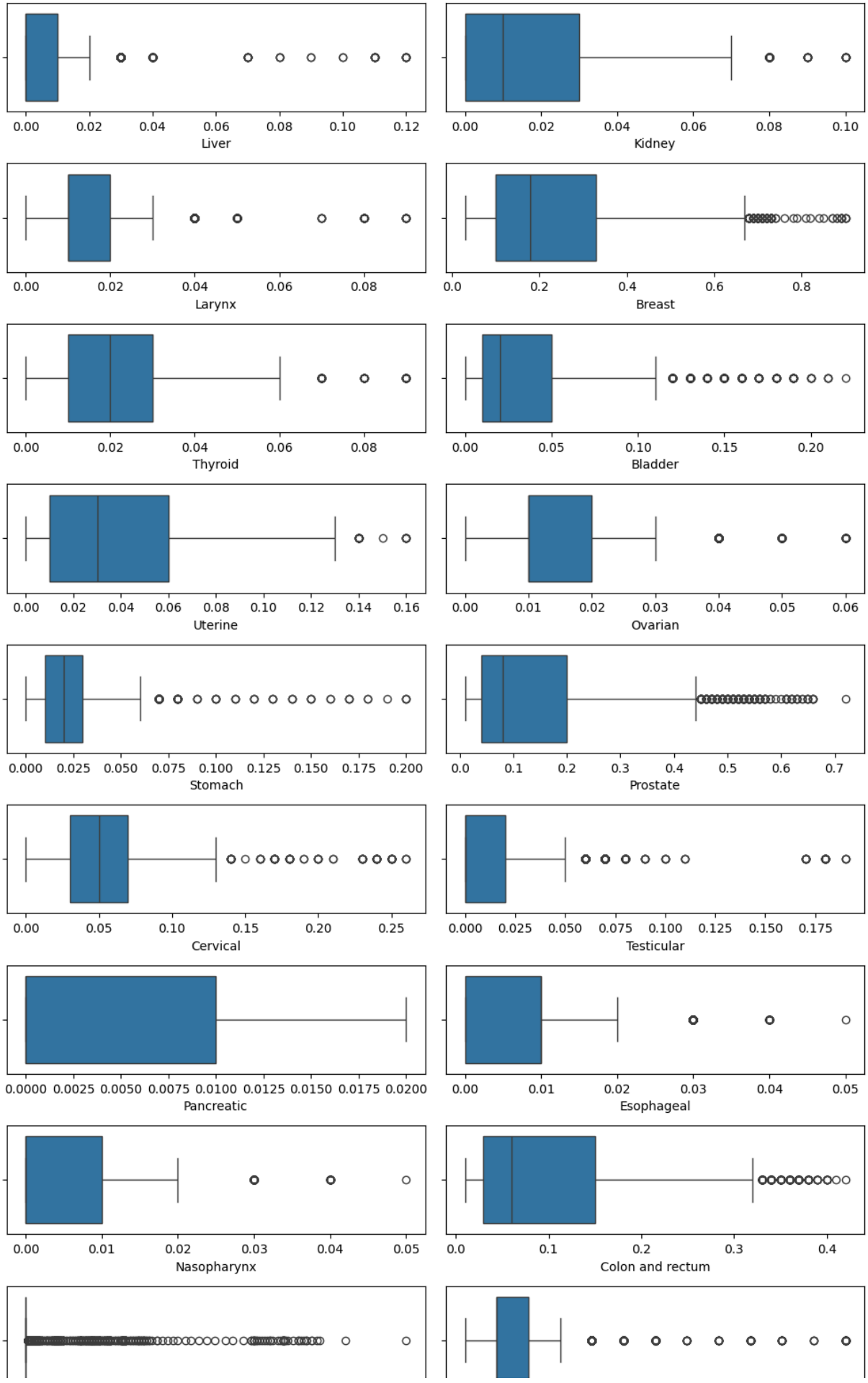
Upon reviewing the numeric features, we note that all features, except for the 'Brain and nervous system' and 'Neoplasms', have a duplicate rate close of

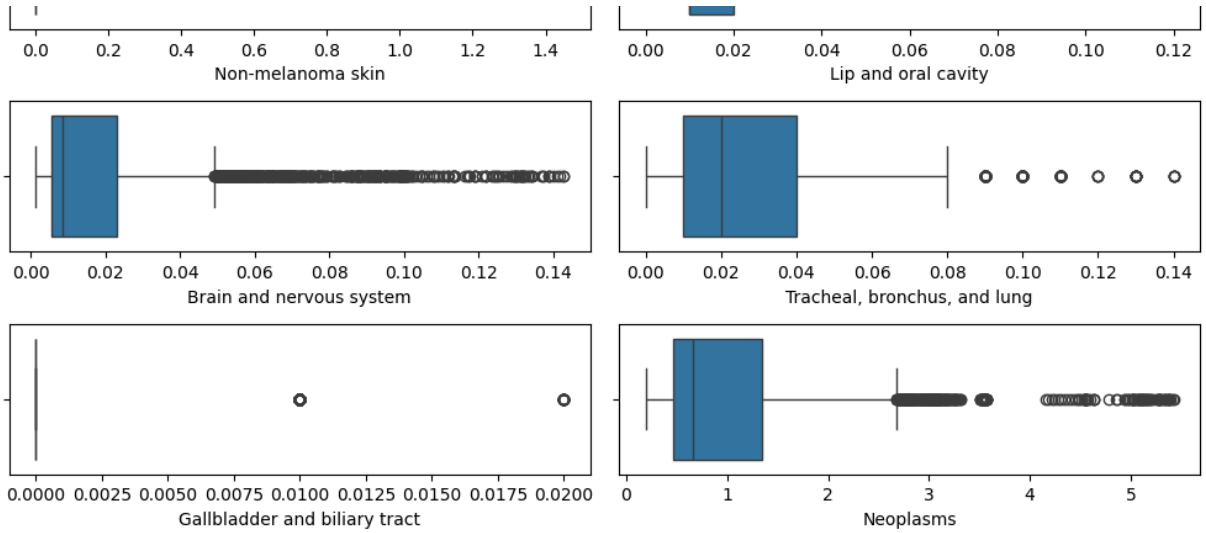
100%. This indicates that there are much duplicate values in the dataset we may need to handling.

2.3.5 - Outlier Value

Outliers (Anomalies) are data points that differ substantially from the rest of data, and they may arise due to the diversity of the dataset or because of errors/mistakes. As machine learning algorithms are sensitive to the range and distribution of attribute values, identifying the outliers and their nature is important for assessing the quality of the dataset.

In [ ]:
```
#-------------------------------------------------------------------------
# The boxplots of outlier
#-------------------------------------------------------------------------

data_numeric = data.select_dtypes(include='number').drop(columns=["Year"])

# Setting the size of subplots
_, ax = plt.subplots(nrows=11, ncols=2, figsize=(10, 20))
ax = ax.ravel()

# Display the boxplot
for index, column in enumerate(data_numeric.columns):
  sns.boxplot(data=data, ax=ax[index], x=column)

# Do not blocked any title or label
plt.tight_layout()
plt.show()
```


Liver
Kidney
Larynx
Breast
Thyroid
Bladder
Uterine
Ovarian
Stomach
Prostate
Cervical
Testicular
Pancreatic
Esophageal
Nasopharynx
Colon and rectum



In [ ]:

```
#-------------------------------------------------------------------------------
# The summary of outlier variables from numeric columns
#-------------------------------------------------------------------------------

data_numeric = data.select_dtypes(include='number').drop(columns=["Year"])

# Using IQR method to detect outlier variables except Year
data_describe_numeric_q1 = data_describe_numeric.drop(columns=["Year"]).T["2
data_describe_numeric_q3 = data_describe_numeric.drop(columns=["Year"]).T["7
data_describe_numeric_iqr = data_describe_numeric_q3 - data_describe_numeric

# Min and Max boundary to detect outlier
data_describe_numeric_min_iqr = data_describe_numeric_q1 - 1.5 * data_descri
data_describe_numeric_max_iqr = data_describe_numeric_q3 + 1.5 * data_descri

# Count the outliers
data_outlier_count = ( (data_numeric < data_describe_numeric_min_iqr) | (dat

# Get the name of numeric column
data_numeric_column = list(data_numeric.columns)

# Get the total rows
data_row_numeric_count = list( [ len(data_numeric) ] * len(data_numeric.colu

# outlier variables / Total rows of numeric column
data_outlier_rate = np.divide( list(data_outlier_count), data_row_numeric_co

data_outlier_summary = pd.DataFrame( zip( data_numeric_column,
                        data_row_numeric_count,
                        data_outlier_count,
                        data_outlier_rate ),
                    columns = [ "Column",
                            "Rows",
                            "Outlier Values",
                            "Outlier Rate %" ] )

data_outlier_summary.sort_values(by="Outlier Values", ascending=False)
```

Out[ ]:

	Column	Rows	Outlier Values	Outlier Rate %
16	Non-melanoma skin	7704	1576	20.456906
20	Gallbladder and biliary tract	7704	711	9.228972
18	Brain and nervous system	7704	533	6.918484
11	Testicular	7704	437	5.672378
17	Lip and oral cavity	7704	435	5.646417
7	Ovarian	7704	317	4.114746
21	Neoplasms	7704	303	3.933022
9	Prostate	7704	280	3.634476
2	Larynx	7704	243	3.154206
5	Bladder	7704	236	3.063344
0	Liver	7704	225	2.920561
15	Colon and rectum	7704	204	2.647975
19	Tracheal, bronchus, and lung	7704	202	2.622015
13	Esophageal	7704	194	2.518172
8	Stomach	7704	122	1.583593
3	Breast	7704	110	1.427830
14	Nasopharynx	7704	100	1.298027
1	Kidney	7704	90	1.168224
10	Cervical	7704	75	0.973520
4	Thyroid	7704	51	0.661994
6	Uterine	7704	20	0.259605
12	Pancreatic	7704	0	0.000000

The plots above indicate the presence of outliers in the most features, except Testicular features.

2.4 - Data Processing

2.4.1 - Data Cleaning

Data Cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When

combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct.

In **2.3 - Quality Assessment**, we examined the dataset and confirmed that there were no incorrect data formats. However, we identified missing values and outliers that need to be addressed.

The Year feature in the dataset is not essential for classifying cancer prevalence across different countries. Our primary goal is to investigate the internal relationships between the prevalence of various cancers and the living environments in different countries over the 30-year period from 1990 to 2019. As such, the Year feature is not needed.

We recommend discarding the Year feature and merging the Country and Code features to create unique identifiers for each country. The other feature data will be averaged based on these merged identifiers. This approach will enhance our clustering experiments while simplifying the model training process.

In [ ]:
```
# Discard the unimportant features
data.drop(columns=["Year"], inplace=True)

data_describe_numeric = data.describe(include='number')
```

In this list of unique values of Country, we found that 55 variables do not belong to countries but to regions. We do not intend to count regions, so we discard those regions.

In [ ]:
```
# Discard the region
non_countries = [ "African Region (WHO)",
           "Andean Latin America",
           "Australasia",
           "Caribbean",
           "Central Africa",
           "Central Asia",
           "Central Europe",
           "Central Europe, Eastern Europe, and Central Asia",
           "Central Latin America",
           "Central Sub-Saharan Africa",
           "England",
           "East Asia",
           "East Asia & Pacific (WB)",
           "Eastern Europe",
           "Eastern Mediterranean Region (WHO)",
           "Eastern Sub-Saharan Africa",
           "Europe & Central Asia (WB)",
           "European Region (WHO)",
           "G20",
           "High SDI",
           "High-income",
```

```
            "High-income Asia Pacific",
            "High-middle SDI",
            "Latin America & Caribbean (WB)",
            "Latin America and Caribbean",
            "Low SDI",
            "Low-middle SDI",
            "Middle East & North Africa (WB)",
            "Middle SDI",
            "North Africa and Middle East",
            "North America",
            "North America (WB)",
            "Northern Ireland",
            "OECD Countries",
            "Oceania",
            "Region of the Americas (WHO)",
            "South-East Asia Region (WHO)",
            "South Asia",
            "South Asia (WB)",
            "Southeast Asia",
            "Southeast Asia, East Asia, and Oceania",
            "Southern Latin America",
            "Southern Sub-Saharan Africa",
            "Sub-Saharan Africa",
            "Sub-Saharan Africa (WB)",
            "Scotland",
            "Tropical Latin America",
            "World",
            "World Bank High Income",
            "World Bank Low Income",
            "World Bank Lower Middle Income",
            "World Bank Upper Middle Income",
            "Western Sub-Saharan Africa",
            "Western Pacific Region (WHO)",
            "Western Europe",
            "Wales" ]

data.drop( data[ data["Country"].isin(non_countries) ].index, inplace=True )

data.reset_index(drop=True, inplace=True)
```

In [ ]:
```
# Merging the country to get the average outcome from 1990 to 2019.
data = data.groupby(["Country", "Code"]).mean().reset_index()
```

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204 entries, 0 to 203
Data columns (total 24 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Country                         204 non-null    object
 1   Code                            204 non-null    object
 2   Liver                           202 non-null    float64
 3   Kidney                          202 non-null    float64
 4   Larynx                          202 non-null    float64
 5   Breast                          202 non-null    float64
 6   Thyroid                         202 non-null    float64
 7   Bladder                         202 non-null    float64
 8   Uterine                         202 non-null    float64
 9   Ovarian                         202 non-null    float64
 10  Stomach                         202 non-null    float64
 11  Prostate                        202 non-null    float64
 12  Cervical                        202 non-null    float64
 13  Testicular                      202 non-null    float64
 14  Pancreatic                      202 non-null    float64
 15  Esophageal                      202 non-null    float64
 16  Nasopharynx                     202 non-null    float64
 17  Colon and rectum                202 non-null    float64
 18  Non-melanoma skin               202 non-null    float64
 19  Lip and oral cavity             202 non-null    float64
 20  Brain and nervous system        195 non-null    float64
 21  Tracheal, bronchus, and lung    202 non-null    float64
 22  Gallbladder and biliary tract   202 non-null    float64
 23  Neoplasms                       195 non-null    float64
dtypes: float64(22), object(2)
memory usage: 38.4+ KB
```

2.4.2 - Missing Value Handling

Missing Value Handling usually uses some techniques:

1. **Median or Mean**

- No matter use median or mean as imputation value, it has limitations. For example, imputing with the mean may not be appropriate if the data has extreme values, as it can be heavily influenced by outliers
- Similarly, imputing with the median may not be appropriate if the data is multimodal, as it may not represent the true central tendency of the data

2. **Iterative**

- An advanced imputation method that models each feature with missing values as a function of other features in a round-robin fashion. It uses a regression model to estimate missing values based on the observed values of other features. The imputation process is performed iteratively, with each

iteration refining the imputed values until convergence or a specified maximum number of iterations is reached
- Commonly used regression models: **Linear Regression**, **Bayesian Ridge** (regularized linear regression), **Decision Trees Regressor**, **Random Forest Regressor**, and **K-Neighbors Regressor**, etc.
- **K-Neighbors Regressor** is different from KNN imputation, which learns from samples with missing values by using a distance metric that accounts for missing values, rather than imputing them

3. **K-Nearest Neighbors (KNN)**

- KNN Imputer imputes missing values based on the nearest neighbors, which means it preserves the underlying relationships in the data. It takes into account the feature similarities between data points to estimate the missing values, making it more contextually relevant
- **Non-Parametric** method, which means it does not make assumptions about the data's distribution. It is suitable for both numeric and categorical data, making it versatile in handling various types of missing values

4. **Multiple Imputation by Chained Equations (MICE)**

- The procedure imputes missing data in a dataset through an iterative series of predictive models. In each iteration, each specified variable in the dataset is imputed using the other variables in the dataset. These iterations should be run until it appears that convergence has been met

Method - Median or Mean

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Median or Mean
#-------------------------------------------------------------------------
'''
# Median
data_missing_imputation = data_describe_numeric.T["50%"].T
# Average/Mean
data_missing_imputation = data_describe_numeric.T["mean"].T

data.fillna(value = data_missing_imputation)
'''
```

Method - Iterative

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Iterative
#-------------------------------------------------------------------------
'''
# Linear Regression
data_missing_iterative_estimator = LinearRegression()
# Bayesian Ridge
```

```
data_missing_iterative_estimator = BayesianRidge()
# Decision Trees Regressor
data_missing_iterative_estimator = DecisionTreeRegressor()
# Random Forest Regressor
data_missing_iterative_estimator = RandomForestRegressor()
# K-Neighbors Regressor
data_missing_iterative_estimator = KNeighborsRegressor()

# Initialization iterative imputation object
data_missing_iterative_imputation = IterativeImputer(estimator = data_missir

# Replace the result in the original dataset
data[data_numeric.columns] = pd.DataFrame(data_missing_iterative_imputation.
'''
```

Method - K-Nearest Neighbors (KNN)

In [ ]:
```
#-------------------------------------------------------------------------
# Method - K-Nearest Neighbors (KNN)
#-------------------------------------------------------------------------

# Initialization KNN imputation object
data_missing_knn_imputation = KNNImputer()

# Replace the result in the original dataset
data[data_numeric.columns] = pd.DataFrame(data_missing_knn_imputation.fit_tr
```

Summary after Missing Values Handling

In [ ]:
```
#-------------------------------------------------------------------------
# The summary of missing variables from whole columns
#-------------------------------------------------------------------------

# Get the name of columns
data_column = list(data.columns)
# Get the total rows
data_row_count = np.array([len(data)] * len(data_column))
# Count of missing variables
data_missing_count = data_row_count - np.array(data.count())
# Missing variables / Total rows
data_missing_rate = np.divide( data_missing_count, data_row_count, out=np.ze

data_missing_summary = pd.DataFrame( zip( data_column,
                       data_row_count,
                       data_missing_count,
                       data_missing_rate ),
                     columns = [ "Column",
                             "Rows",
                             "Missing Values",
                             "Missing Rate %" ] )

data_missing_summary.sort_values(by="Missing Values", ascending=False)
```

Out[ ]:

	Column	Rows	Missing Values	Missing Rate %
0	Country	204	0	0.0
1	Code	204	0	0.0
22	Gallbladder and biliary tract	204	0	0.0
21	Tracheal, bronchus, and lung	204	0	0.0
20	Brain and nervous system	204	0	0.0
19	Lip and oral cavity	204	0	0.0
18	Non-melanoma skin	204	0	0.0
17	Colon and rectum	204	0	0.0
16	Nasopharynx	204	0	0.0
15	Esophageal	204	0	0.0
14	Pancreatic	204	0	0.0
13	Testicular	204	0	0.0
12	Cervical	204	0	0.0
11	Prostate	204	0	0.0
10	Stomach	204	0	0.0
9	Ovarian	204	0	0.0
8	Uterine	204	0	0.0
7	Bladder	204	0	0.0
6	Thyroid	204	0	0.0
5	Breast	204	0	0.0
4	Larynx	204	0	0.0
3	Kidney	204	0	0.0
2	Liver	204	0	0.0
23	Neoplasms	204	0	0.0

2.4.3 - Duplicate Handling

In **2.3.4 - Duplicate Value**, we observed that the dataset contains significant duplicates across most features, with the exceptions of 'Brain and Nervous System' and 'Neoplasms.' While many features exhibit a high number of duplicate values, replacing them is not straightforward. These values are derived from population proportions to calculate cancer prevalence, and certain regions may have individual cases or a small population base, which further diminishes

their significance. Given that any transformation of the data could impact the final clustering results, we have decided not to handle these highly repeated features. Although the machine learning model will treat them as ordinal data, this approach aligns well with our clustering objectives.

In [ ]:
```
# Drop the duplicate rows
'''
data.drop_duplicates(inplace=True)
'''
```

Summary after Duplicate Handling

In [ ]:
```
#-------------------------------------------------------------------------
# The summary of duplicate variables from whole columns
#-------------------------------------------------------------------------
'''
# Quick check of unique value
data_unique_count = data.nunique()

# Unique variables / Total rows
data_duplicate_rate = ( np.ones(len(data_unique_count)) - np.divide( list(da

data_duplicate_summary = pd.DataFrame( zip ( data_column,
                         data_row_count,
                         data_unique_count,
                         data_duplicate_rate),
                       columns = [ "Column",
                               "Rows",
                               "Unique Values",
                               "Duplicate Rate %" ] )

data_duplicate_summary.sort_values(by="Duplicate Rate %", ascending=False)
'''
```

2.4.4 - Outlier Handling

Outlier Handling usually uses four different techniques:

1. **Deleting Observations**

- We delete outlier values if it is due to data entry error, data processing error or outlier observations are very small in numbers. We can also use trimming at both ends to remove outliers
- **BUT** deleting the observation is not a good idea when we have small dataset

2. **Transforming Values**

- Transforming variables can also eliminate outliers. These transformed values reduces the variation caused by extreme values

- If dataset has to many extreme values or skewed, **Log Transformation**, **Cube Root Normalization**, **Box-transformation**, **Yeo-Johnson Power Transformation**, etc., those techniques convert values in the dataset to smaller values
- **BUT** these technique not always give the best results. For example, **Log Transformation** requires that each transformed value not closing to zero; **Box-transformation** requires that each transformed value is positive, otherwise **Yeo-Johnson Power Transformation** needs to be used as an alternative

3. **Imputation**

- Like imputation of missing values, we can also impute outliers. We can use **Mean**, **Median**, **Zero** value in this methods. Since we imputing there is no loss of data
- Use missing value imputation methods, such as **Iterative Imputation** and **K-Nearest Neighbors (KNN) Imputation**

4. **Separately Treating**

- If there are significant number of outliers and dataset is small , we should treat them separately in the statistical model. One of the approach is to treat both groups as two different groups and build individual model for both groups and then combine the output
- **BUT** this technique is tedious when the dataset is large

In **2.3.5 - Outlier Value**, we examined the dataset and identified a lot of outlier values in the most features. Given the critical role that all data plays in clustering, we cannot afford to discard any outliers. However, to preserve the significance of extreme values while ensuring they remain meaningful within the analysis, we will employ transformation techniques. These methods will adjust the extreme values into a more reasonable range, maintaining their original importance in the dataset.

In [ ]:
```
data_numeric = data.select_dtypes(include='number')

# Using IQR method to detect outlier variables except Year
data_describe_numeric_q1 = data_describe_numeric.T["25%"] #data_numeric.quar
data_describe_numeric_q3 = data_describe_numeric.T["75%"] #data_numeric.quar
data_describe_numeric_iqr = data_describe_numeric_q3 - data_describe_numeric

# Min and Max boundary to detect outlier
data_describe_numeric_min_iqr = data_describe_numeric_q1 - 1.5 * data_descri
data_describe_numeric_max_iqr = data_describe_numeric_q3 + 1.5 * data_descri

# Count the outliers
data_outlier_count = ( (data_numeric < data_describe_numeric_min_iqr) | (dat
```

```
data_outlier_count.sort_values(ascending=False)
```

Out[ ]:

	0
Gallbladder and biliary tract	45
Non-melanoma skin	28
Brain and nervous system	26
Colon and rectum	19
Testicular	15
Lip and oral cavity	11
Ovarian	5
Pancreatic	0
Tracheal, bronchus, and lung	0
Nasopharynx	0
Esophageal	0
Liver	0
Kidney	0
Cervical	0
Prostate	0
Stomach	0
Uterine	0
Bladder	0
Thyroid	0
Breast	0
Larynx	0
Neoplasms	0

dtype: int64

Method - Deleting Observations

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Deleting observations
#-------------------------------------------------------------------------
'''
for _, column in enumerate(data_outlier_count.loc[data_outlier_count > 0].ir
  data.drop( data[ (data[column] < data_describe_numeric_min_iqr[column]) |
```

```
data.reset_index(drop=True, inplace=True)
'''
```

Method - Transforming Values

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Transforming values - Log Transformation
#-------------------------------------------------------------------------
'''
# If data value closing to 0, DO NOT use this method
for _, column in enumerate(data_outlier_count.loc[data_outlier_count > 0].ir
  data[column] = np.log(data[column])
'''
```

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Transforming values - Cube Root Normalization
#-------------------------------------------------------------------------

for _, column in enumerate(data_outlier_count.loc[data_outlier_count > 0].ir
  data[column] = (data[column]**(1/3))
```

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Transforming values - Box-Transformation or Yeo-Johnson Power Tra
#-------------------------------------------------------------------------
'''
# Boxcox requires all of the elements must be positive, otherwise use Yeo-Jo
for _, column in enumerate(data_outlier_count.loc[data_outlier_count > 0].ir
  if np.any(data[column] <= 0):
    data[column], _ = scipy.stats.yeojohnson(data[column], lmbda=None)
  else:
    data[column], _ = scipy.stats.boxcox(data[column], lmbda=None)
'''
```

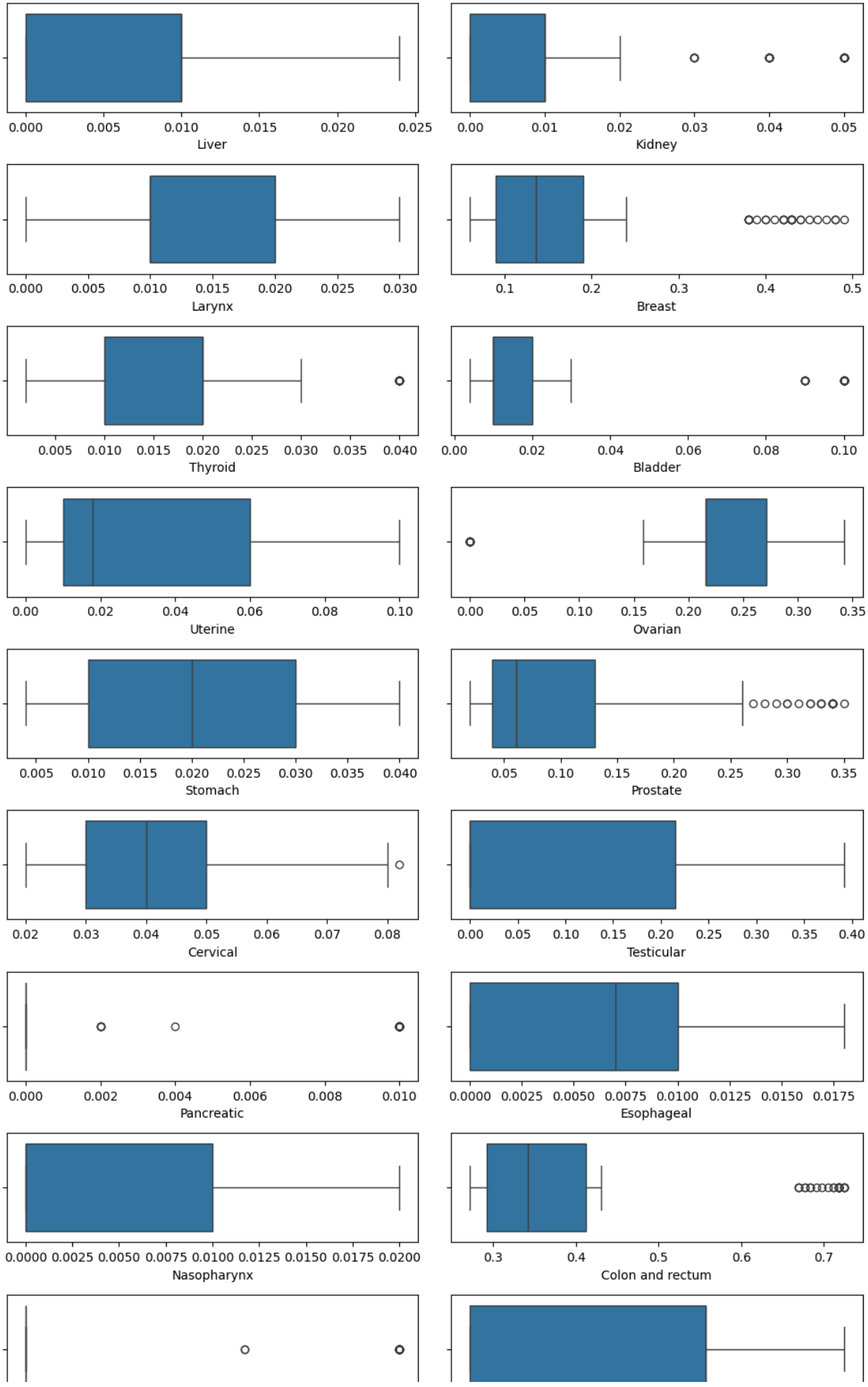
Method - Imputation

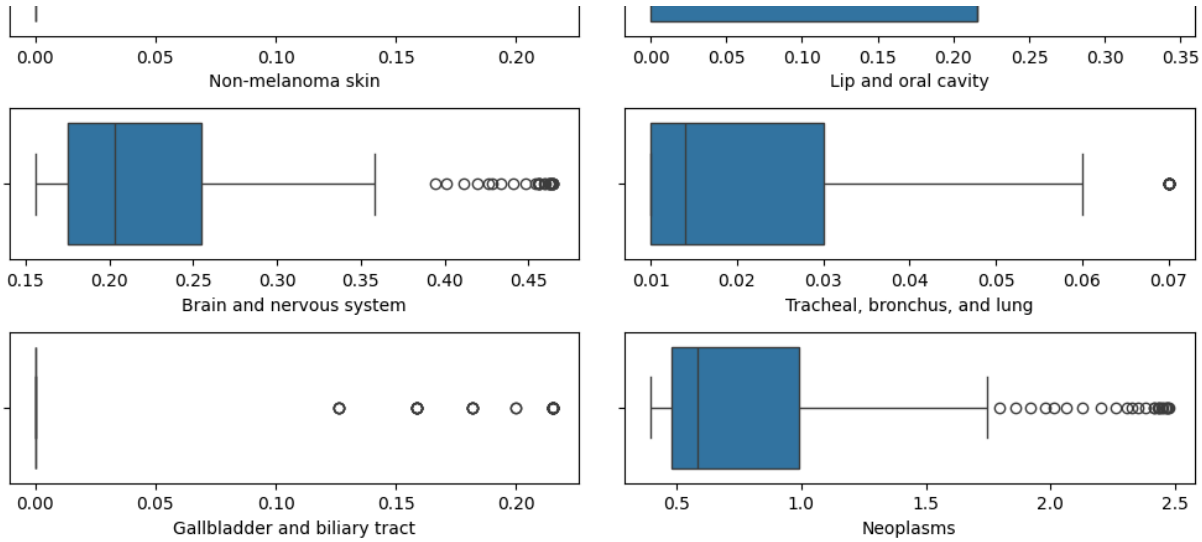
In [ ]:
```
#-------------------------------------------------------------------------
# Method - Imputation - Median, Mean, Zero
#-------------------------------------------------------------------------
'''
# Median
data_precipitation_outlier_imputation = data_describe_numeric["Precipitatior
# Average/Mean
data_precipitation_outlier_imputation = data_describe_numeric["Precipitatior
# Zero
data_precipitation_outlier_imputation = 0

data["Precipitation"] = np.where( (data["Precipitation"] < data_describe_num
'''
```

Visualize the Boxplot after Outlier Handling

In [ ]:
```
#-------------------------------------------------------------------------
# The boxplots
#-------------------------------------------------------------------------
```

```
# Setting the size of subplots
_, ax = plt.subplots(nrows=11, ncols=2, figsize=(10, 20))
ax = ax.ravel()

# Display the boxplot
for index, column in enumerate(data_numeric.columns):
  sns.boxplot(data=data, ax=ax[index], x=column)

# Do not blocked any title or label
plt.tight_layout()
plt.show()
```


0.000 0.005 0.010 0.015 0.020 0.025
Liver
0.00 0.01 0.02 0.03 0.04 0.05
Kidney
0.000 0.005 0.010 0.015 0.020 0.025 0.030
Larynx
0.1 0.2 0.3 0.4 0.5
Breast
0.005 0.010 0.015 0.020 0.025 0.030 0.035 0.040
Thyroid
0.00 0.02 0.04 0.06 0.08 0.10
Bladder
0.00 0.02 0.04 0.06 0.08 0.10
Uterine
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35
Ovarian
0.005 0.010 0.015 0.020 0.025 0.030 0.035 0.040
Stomach
0.05 0.10 0.15 0.20 0.25 0.30 0.35
Prostate
0.02 0.03 0.04 0.05 0.06 0.07 0.08
Cervical
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40
Testicular
0.000 0.002 0.004 0.006 0.008 0.010
Pancreatic
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175
Esophageal
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
Nasopharynx
0.3 0.4 0.5 0.6 0.7
Colon and rectum



2.4.5 - Centering and Scaling

Centering and Scaling ensures that the criterion for finding linear combinations of the predictors is based on how much variation they explain and therefore improves the numerical stability.

- **Standard Scaling** - Converts features to **standard normal** variables, and it centers and scales a variable to mean 0 and standard deviation 1
- **Min-Max Scaling** - Convert variables to continuous variables in the [0, 1] interval by mapping minimum values to 0 and maximum values to 1
- **Robust Scaling** - Similar to min-max scaling, but instead maps the **interquartile range** (Q3 - Q1) to [0, 1] interval, it means the variable itself takes values outside of the [0, 1] interval

In **2.2 - Data Decription**, we observed that most numeric features fall within the range [0, 1]. Additionally, since we have already addressed the extreme values, it is unlikely that any values will exceed this range significantly. Therefore, we can forgo the processes of centering and scaling at this stage.

In [ ]:
```
# Get the data with type numeric that AFTER cleaning and outlier handling
data_numeric = data.select_dtypes(include='number')
```

Method - Standard Scaling

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Standard Scaling
#-------------------------------------------------------------------------
'''
data[data_numeric.columns] = StandardScaler().fit_transform(data_numeric)
'''
```

Method - Min-Max Scaling

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Min-Max Scaling
#-------------------------------------------------------------------------
'''
data[data_numeric.columns] = MinMaxScaler().fit_transform(data_numeric)
'''
```

Method - Robust Scaling

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Robust Scaling
#-------------------------------------------------------------------------
'''
data[data_numeric.columns] = RobustScaler().fit_transform(data_numeric)
'''
```

Visualize the Boxplot after Centering and Scaling

In [ ]:
```
#-------------------------------------------------------------------------
# The boxplots
#-------------------------------------------------------------------------
'''
# Setting the size of subplots
_, ax = plt.subplots(nrows=3, ncols=2, figsize=(10, 5))
ax = ax.ravel()

# Display the boxplot
for index, column in enumerate(data_numeric.columns):
  sns.boxplot(data=data, ax=ax[index], x=column)

# Do not blocked any title or label
plt.tight_layout()
plt.show()
'''
```

2.4.6 - Data Transformation

Features and predicted data are often **Skewed** (distorted away from the center), it degrades the model's ability to describe typical cases as it has to deal with rare cases on extreme values (**especially regression based models**).

Data Transformation usually can solve the skewed data. To ensure that the machine learning model capabilities is not affected, skewed data has to be transformed to approximate to a normal distribution. The method used to transform the skewed data depends on the characteristics of the data.

- Popular data transformation techniques include **Log Transformation**, **Cube Root Normalization**, **Box-Transformation**, **Yeo-Johnson Power Transformation**, etc.
- **BUT** these technique not always give the best results. For example, **Log Transformation** requires that each transformed value not closing to zero; **Box-Transformation** requires that each transformed value is positive, otherwise **Yeo-Johnson Power Transformation** needs to be used as an alternative

It is worth noting that tree-based models are not affected by these issues, as they can effectively ignore correlation concerns. Consequently, tree-based models do not require data transformation, centering, or scaling.

Even after addressing outliers in **2.4.4 - Outlier Handling** and applying centering and scaling in **2.4.5 - Centering and Scaling**, the boxplot still shows these few features remains skewed.

Method - Log Transformation

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Log Transformation
#-------------------------------------------------------------------------
'''
# If data value closing to 0, DO NOT use this method
for _, column in enumerate(data_numeric.columns):
  data[column] = np.log(data[column])
'''
```

Method - Cube Root Transformation

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Cube Root Normalization
#-------------------------------------------------------------------------
'''
for _, column in enumerate(data_numeric.columns):
  data[column] = (data[column]**(1/3))
'''
```

Method - Box-Transformation or Yeo-Johnson Power Transformation

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Box-Transformation or Yeo-Johnson Power Transformation
#-------------------------------------------------------------------------

# Boxcox requires all of the elements must be positive, otherwise use Yeo-Jo
for _, column in enumerate(data_numeric.columns):
  if np.any(data[column] <= 0):
    data[column], _ = scipy.stats.yeojohnson(data[column], lmbda=None)
```

```
  else:
    data[column], _ = scipy.stats.boxcox(data[column], lmbda=None)
```

Summary after Transformation

In [ ]:

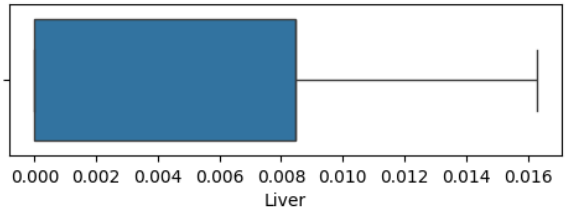
```
#-------------------------------------------------------------------------
# The summary of skewness and kurtosis
#-------------------------------------------------------------------------

# Get the data with type numeric
data_numeric = data.select_dtypes(include='number')

# Get the skewness for numeric column
data_numeric_skew = data_numeric.skew()
# Get the kurtosis for numeric column
data_numeric_kurtosis = data_numeric.kurtosis()

data_normal_summary = pd.DataFrame( zip( data_numeric_column,
                        data_numeric_skew,
                        data_numeric_kurtosis ),
                      columns = [ "Column",
                              "Skewness",
                              "Kurtosis" ] )

data_normal_summary.sort_values(by="Skewness", ascending=False)
```

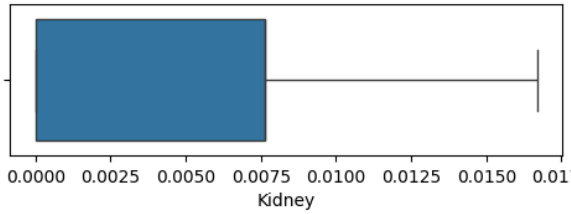
Out[ ]:

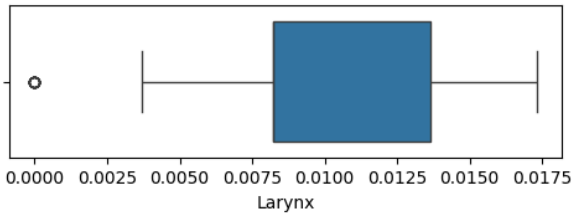
	Column	Skewness	Kurtosis
16	Non-melanoma skin	2.123955	2.536021
12	Pancreatic	1.704845	0.960410
20	Gallbladder and biliary tract	1.358734	-0.153871
14	Nasopharynx	0.695623	-1.397065
11	Testicular	0.681623	-1.480468
6	Uterine	0.350256	-1.327284
18	Brain and nervous system	0.311434	-1.278869
19	Tracheal, bronchus, and lung	0.288779	-1.631725
21	Neoplasms	0.278192	-1.154785
1	Kidney	0.248250	-0.642949
15	Colon and rectum	0.202569	-0.926156
5	Bladder	0.089328	1.921481
7	Ovarian	0.082608	1.107489
13	Esophageal	0.068715	-1.719113
0	Liver	0.066028	-0.951153
3	Breast	0.058736	-0.645161
9	Prostate	0.018982	-0.676255
2	Larynx	0.018952	0.171450
10	Cervical	-0.006397	-1.041333
4	Thyroid	-0.006627	-0.264848
8	Stomach	-0.045311	-1.115664
17	Lip and oral cavity	-0.139056	-0.779003

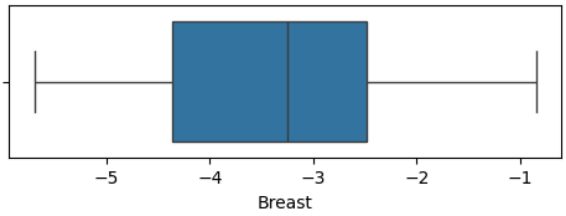
In [ ]:

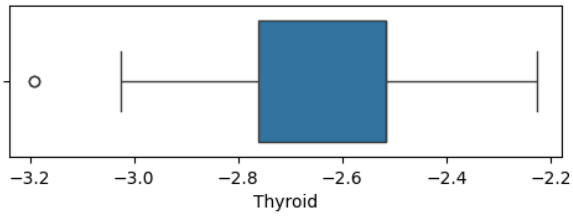
```
data_numeric = data.select_dtypes(include='number')

# Setting the size of subplots
_, ax = plt.subplots(nrows=11, ncols=2, figsize=(10, 20))
ax = ax.ravel()

# Display the boxplot
for index, column in enumerate(data_numeric.columns):
  sns.boxplot(data=data, ax=ax[index], x=column)

# Do not blocked any title or label
plt.tight_layout()
plt.show()
```

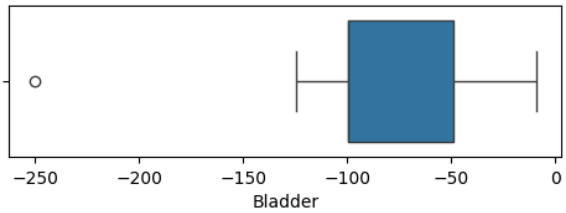

0.000 0.002 0.004 0.006 0.008 0.010 0.012 0.014 0.016
Liver

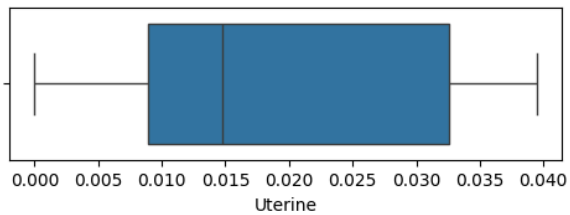

0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175
Kidney

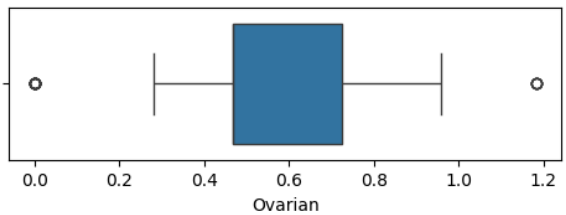

0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175
Larynx

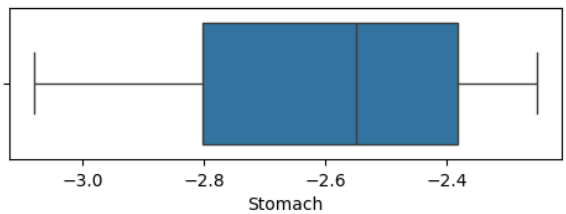

−5 −4 −3 −2 −1
Breast

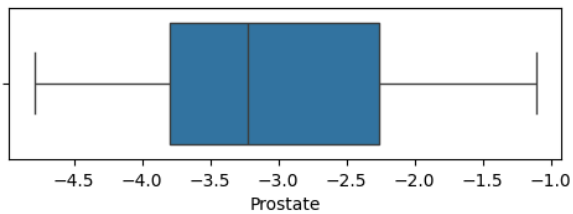

−3.2 −3.0 −2.8 −2.6 −2.4 −2.2
Thyroid

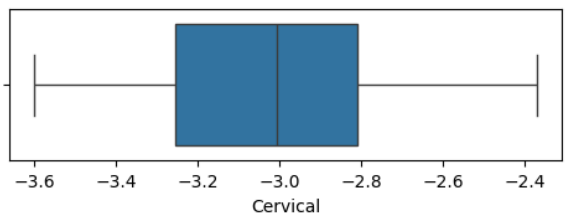

−250 −200 −150 −100 −50 0
Bladder

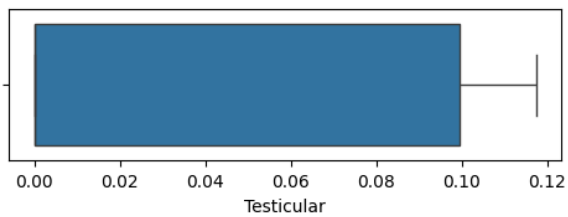

0.000 0.005 0.010 0.015 0.020 0.025 0.030 0.035 0.040
Uterine

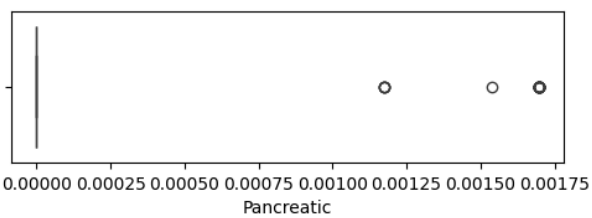

0.0 0.2 0.4 0.6 0.8 1.0 1.2
Ovarian

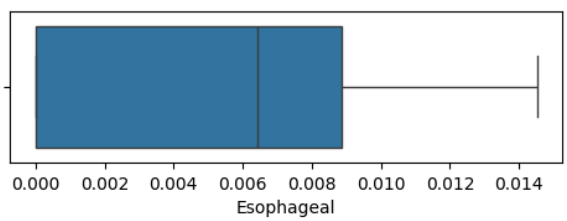

−3.0 −2.8 −2.6 −2.4
Stomach

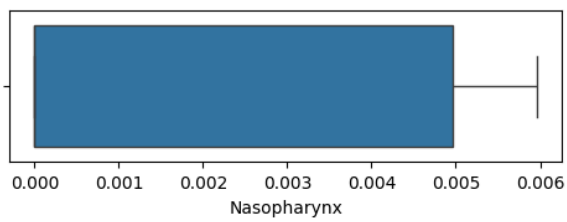

−4.5 −4.0 −3.5 −3.0 −2.5 −2.0 −1.5 −1.0
Prostate

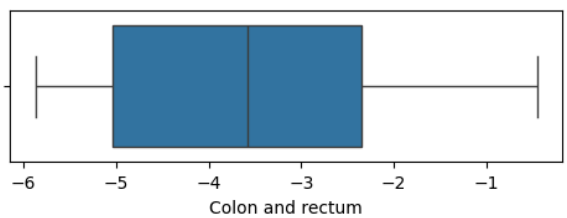

−3.6 −3.4 −3.2 −3.0 −2.8 −2.6 −2.4
Cervical

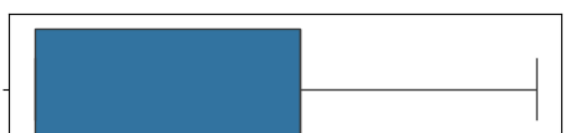

0.00 0.02 0.04 0.06 0.08 0.10 0.12
Testicular

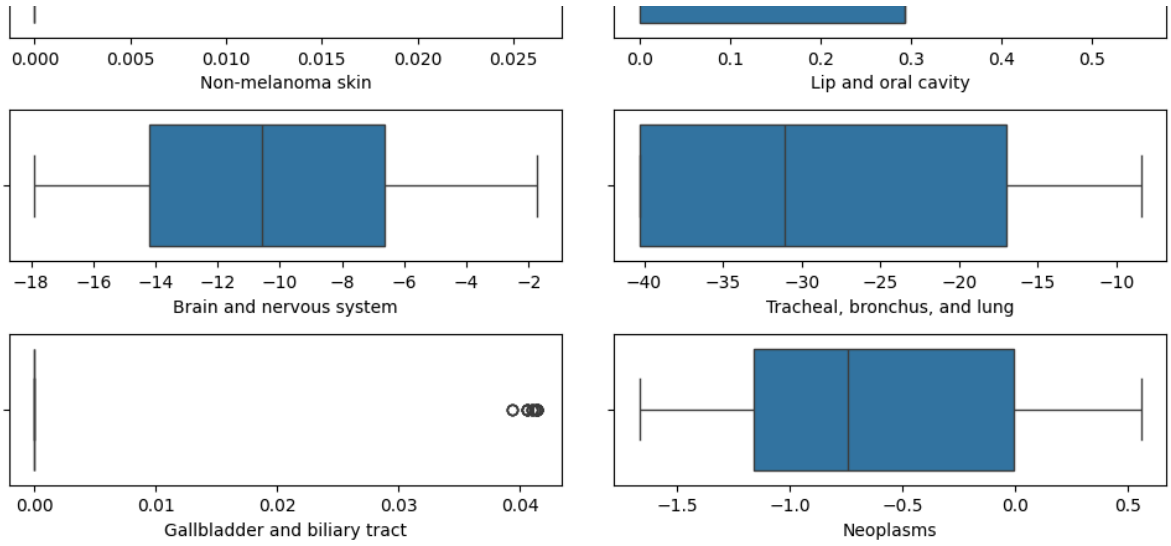

0.00000 0.00025 0.00050 0.00075 0.00100 0.00125 0.00150 0.00175
Pancreatic


0.000 0.002 0.004 0.006 0.008 0.010 0.012 0.014
Esophageal


0.000 0.001 0.002 0.003 0.004 0.005 0.006
Nasopharynx


−6 −5 −4 −3 −2 −1
Colon and rectum





2.4.7 - Correlation Coefficient

Understanding the correlations between variables in a model is essential for several reasons:

1. **Feature selection**

- Process of choosing which variables or features to use in the model. Highly correlated features provide redundant information, so feature selection aims to remove uninformative features to simplify models
- By analyzing correlations, we can identify redundant features and select a minimal set of important features that best represent the target variable. This prevents overfitting and improves a model's ability to generalize

2. **Reduce Bias**

- Correlation analysis is also important for ensuring model fairness and avoiding bias. When certain features are highly correlated with sensitive attributes like gender or ethnicity, it can inadvertently encode biases into machine learning models if not properly addressed
- If a model relies too heavily on these correlated features, it risks discriminating against or disadvantaging certain groups. By identifying correlations between input features and sensitive attributes, we can evaluate models for potential biases, monitor feature importance, and apply techniques like fair representation learning to mitigate bias

3. **Multicollinearity**

- Another important aspect of analyzing feature correlations is detecting multicollinearity. Multicollinearity occurs when two or more predictor variables in a model are highly linearly correlated with each other. It can negatively impact models by increasing variance and making it difficult to determine the significance and effect of individual predictors

- Variables with high multicollinearity provide redundant information, similar to how correlated features do. However, multicollinearity is more problematic because it inflates standard errors and undermines reliability of estimated coefficients. By examining correlation matrices and variance inflation factors, we can identify cases of multicollinearity between input features

4. **Interpretability and Debugging**

- Understanding correlations also aids in interpreting machine learning models. As models become increasingly complex with many interacting variables, it can be difficult to explain why a model makes certain predictions

- By analyzing the correlation between input features and output targets, we gain insights into which variables have the strongest impact on the model's decisions. Knowing feature correlations further assists in debugging models that perform poorly. It allows we to identify any features that may be overwhelming the model or causing unintended biases

Interpreting a Correlation Coefficient

- The value of the correlation coefficient always ranges between 1 and -1, and we treat it as a general indicator of the strength of the relationship between variables

- The sign of the coefficient reflects whether the variables change in the same or opposite directions: a positive value means the variables change together in the same direction, while a negative value means they change together in opposite directions

- The absolute value of a correlation coefficient tells the magnitude of the correlation: the greater the absolute value, the stronger the correlation

Correlation Coefficient	Strength of Linearity / Monotonically	Correlation Type
-0.75 to -1	Perfectly	Negative
-0.5 to -0.75	Strong	Negative
-0.25 to -0.5	Moderate	Negative
0 to -0.25	Weak	Negative

Correlation Coefficient	Strength of Linearity / Monotonically	Correlation Type
0	None	Zero
0 to 0.25	Weak	Positive
0.25 to 0.5	Moderate	Positive
0.5 to 0.75	Strong	Positive
0.75 to 1	Perfectly	Positive

Methods of Calculate the Correlation Coefficient

Usually we use two mainstream methods to calculate the correlation coefficient:

1. **Pearson's Correlation Coefficient**

- The Pearson's correlation coefficient describes the linear relationship between two quantitative variables
- The assumptions for use Pearson's correlation coefficient:
    1. Expect a linear relationship between the two variables
    2. Both variables are on an interval or ratio level of measurement
    3. Data from both variables follow normal distributions
    4. Data have no outliers
- **BUT** it's not a good measure of correlation if variables have a nonlinear relationship, or if data have outliers, skewed distributions, or come from categorical variables

2. **Spearman's Rank-Order Correlation**

- Spearman's rank correlation coefficient is the most common alternative to Pearson method. It uses the rankings of data from each variable (e.g., from lowest to highest) rather than the raw data itself
- Use Spearman method when data fail to meet the assumptions of Pearson method. This happens when at least one of variables is on an ordinal level of measurement or when the data from one or both variables do not follow normal distributions
- Spearman's correlation coefficient measures the monotonicity of relationships, and monotonic relationships are less restrictive than linear relationships
    - Positive monotonic: when one variable increases, the other also increases

- Negative monotonic: when one variable increases, the other decreases

In **2.4.6 - Data Transformation**, we deal with non normally distributed, but we still have couple of features that were non normally distributed. Therefore, some assumptions for Pearson's correlation coefficient are not met. Apply Spearman's rank-order correlation for our analysis.

Method - Pearson's Correlation Coefficient

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Pearson's Correlation Coefficient
#-------------------------------------------------------------------------
'''
# Computing the Pearson's correlation coefficient, BUT corr() won't have P-v
#data_numeric_correlation = data_numeric.corr(method='pearson')

# Computing the Pearson's correlation coefficient and P-value
data_numeric_correlation = np.ones(shape=(data_numeric.shape[1], data_numeri
data_numeric_correlation_p = np.zeros(shape=(data_numeric.shape[1], data_num
for index_1, column_1 in enumerate(data_numeric.columns):
  for index_2, column_2 in enumerate(data_numeric.columns):
    if index_1 != index_2:
      data_numeric_correlation[index_1, index_2], data_numeric_correlation_p
'''
```

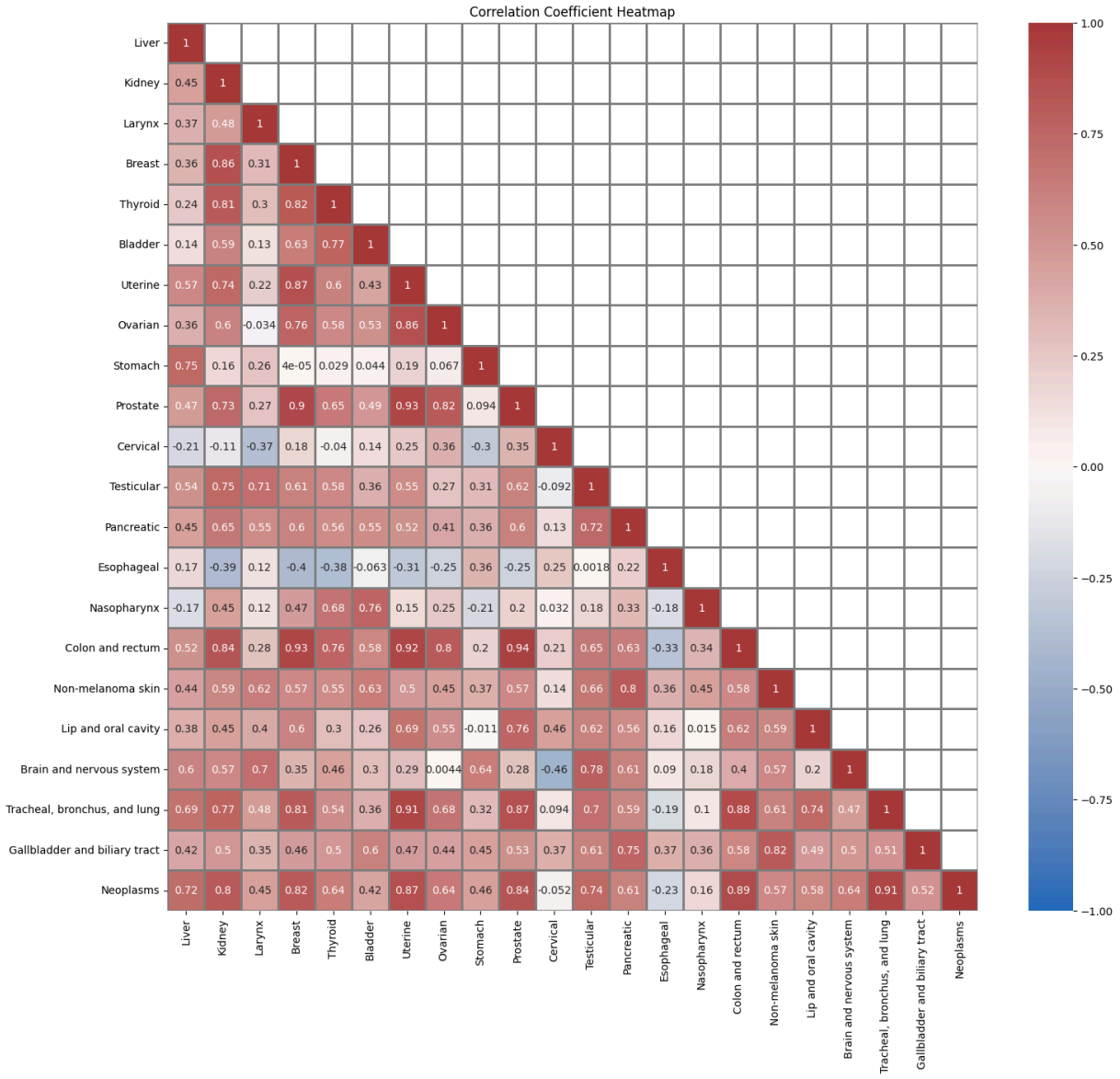
Method - Spearman's Correlation Coefficient

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Spearman's Rank-Order Correlation
#-------------------------------------------------------------------------

# Computing the Spearman's correlation coefficient, BUT corr() won't have P-
#data_numeric_correlation = data_numeric.corr(method='spearman')

# Computing the Spearman's correlation coefficient and P-value
data_numeric_correlation = np.ones(shape=(data_numeric.shape[1], data_numeri
data_numeric_correlation_p = np.zeros(shape=(data_numeric.shape[1], data_num
for index_1, column_1 in enumerate(data_numeric.columns):
  for index_2, column_2 in enumerate(data_numeric.columns):
    if index_1 != index_2:
      data_numeric_correlation[index_1, index_2], data_numeric_correlation_p
```

Visualize the Heatmap of Correlation Coefficient

In [ ]:
```
# Display the heatmap of correlation coefficient
_, ax = plt.subplots(figsize=(17, 15))
sns.heatmap( data_numeric_correlation,
       ax = ax,
       mask = np.triu(data_numeric_correlation, 1),
       xticklabels = data_numeric.columns,
       yticklabels = data_numeric.columns,
```

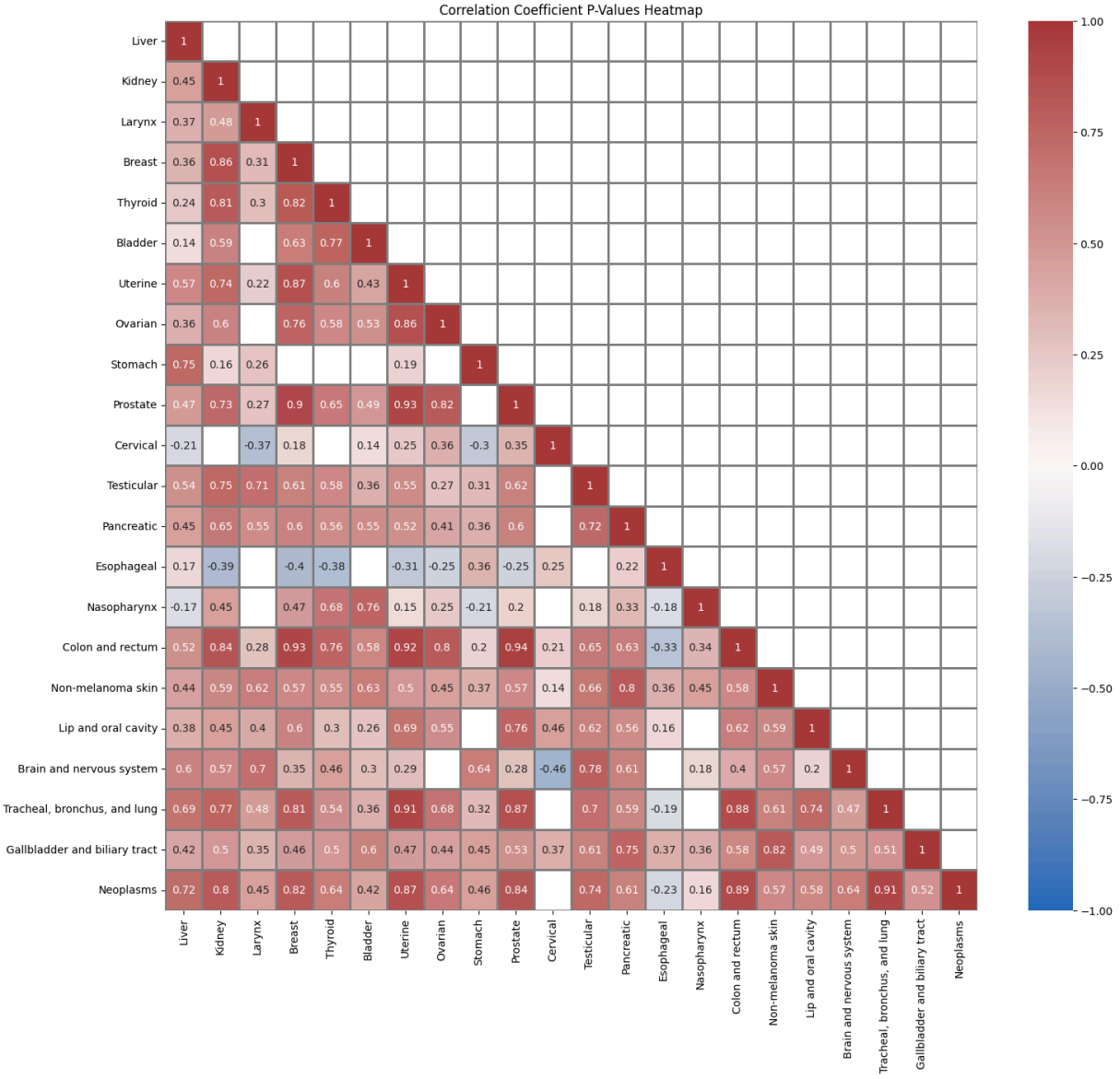
```
        annot = True,
        center = 0,
        vmin = -1,
        vmax = 1,
        cmap = 'vlag',
        linecolor = 'gray',
        linewidths = 1 )

plt.title("Correlation Coefficient Heatmap")
plt.show()
```



Visualize the Heatmap of Correlation Coefficient P-Value

In [ ]:
```
# Display the heatmap of correlation coefficient after P-values hypothesis t
_, ax = plt.subplots(figsize=(17, 15))
sns.heatmap( data_numeric_correlation,
        ax = ax,
        mask = np.invert(np.tril(data_numeric_correlation_p < 0.05)),
        xticklabels = data_numeric.columns,
        yticklabels = data_numeric.columns,
```

```
        annot = True,
        center = 0,
        vmin = -1,
        vmax = 1,
        cmap = 'vlag',
        linecolor = 'gray',
        linewidths = 1 )

plt.title('Correlation Coefficient P-Values Heatmap')
plt.show()
```



From the two plots above, we observed that all numerical features exhibit correlation with one another. Except for Cervical and Espohageal features has negative correlation to another features.

2.4.8 - Data Encoding

Data Encoding refers to the process of converting categorical or textual data into numerical format, so that it can be used as input for algorithms to process. The reason for encoding is that most machine learning algorithms work with numbers and not with text or categorical variables.

There are two types of categorical data:

- **Nominal Data** - The categories of data do not have an inherent order. This means that the categories cannot be ranked or ordered. For example: Occupational titles for doctor, lawyer, instructor, athlete, etc.
- **Ordinal Data** - The categories of data have an inherent order. This means that the categories can be ranked or ordered from highest to lowest or vice versa. For example: Grades start with A+, A, A-, B+, B, B-, etc.

Methods of Data Encoding

The choice of encoding method can have a significant impact on model performance, so it is important to choose an appropriate encoding technique based on the nature of the data and the specific requirements of the model.

1. **One-Hot Encoding**

- Binary column is created for each unique category in the variable. If a category is present in a sample, the corresponding column is set to 1, and all other columns are set to 0
- In the case of one-hot encoding, for N categories in a variable, it uses N binary variables
- For example, if a variable has three categories A, B and C, they can be represented as [1, 0, 0], [0, 1, 0] and [0, 0, 1], respectively

2. **Dummy Encoding**

- Dummy coding scheme is similar to one-hot encoding. This categorical data encoding method transforms the categorical variable into a set of binary variables 0/1
- The dummy encoding is a small improvement over one-hot-encoding. Dummy encoding uses N-1 features to represent N categories
- For example, if a variable has three categories A, B and C, they can be represented as [1, 0] and [0, 1], respectively

3. **Binary Encoding**

- Similar to one-hot encoding, but instead of creating a separate column for each category, the categories are represented as binary digits

- For example, if a variable has four categories A, B, C and D, they can be represented as 0001, 0010, 0100 and 1000, respectively

4. **Label Encoding**

- Each unique category is assigned a Unique Integer value
- But the assigned integers may be misinterpreted by the machine learning algorithm as having an ordered relationship when in fact they do not.
- For example, if a variable has four categories A, B, C and D, they can be represented as 0, 1, 2 and 3, respectively

4. **Ordinal Encoding**

- Ordinal encoding is used when the categories in a variable have an inherent ordering
- The categories are assigned a numerical value based on their order, such as 1, 2, 3, etc.
- For example, if a variable has categories Low, Medium and High, they can be assigned the values 1, 2, and 3, respectively

In sections **2.2 - Data Description** and **2.4.1 - Data Cleaning**, we identified two textual features: Country and Code. As there are no categorical features requiring further processing, we can proceed to the next step.

Method - One-Hot Encoding

In [ ]:
```
#-------------------------------------------------------------------------
# Method - One-Hot Encoding
#-------------------------------------------------------------------------
'''
data_encoding_object = pd.DataFrame(data["Location"], columns=["Location"])
data_encoding_object = pd.get_dummies(data_encoding_object["Location"], colu

# Drop the old features
data.drop(columns=["Location"], inplace=True)
# Combination to the original data
data = pd.concat( [data, data_encoding_object], axis=1 )
'''
```

Method - Dummy Encoding

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Dummy Encoding
#-------------------------------------------------------------------------
'''
data_encoding_object = pd.DataFrame(data["Location"], columns=["Location"])
data_encoding_object = pd.get_dummies(data_encoding_object["Location"], drop

# Drop the old features
data.drop(columns=["Location"], inplace=True)
```

```
# Combination to the original data
data = pd.concat( [data, data_encoding_object], axis=1 )
'''
```

Method - Binary Encoding

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Binary Encoding
#-------------------------------------------------------------------------
'''
data_encoding_object_rank = { city : index for index, city in enumerate(data

data["Location"] = data["Location"].map(data_encoding_object_rank).apply(lam
'''
```

Method - Label Encoding

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Label Encoding
#-------------------------------------------------------------------------
'''
data_encoding_object_rank = { city : index for index, city in enumerate(data

data["Location"] = data["Location"].map(data_encoding_object_rank)
'''
```

Method - Ordinal Encoding

In [ ]:
```
#-------------------------------------------------------------------------
# Method - Ordinal Encoding
#-------------------------------------------------------------------------
'''
EXAMPLE
# Identity the order of the categories
data_encoding_object_rank = { "low=": 0,
                 "medium": 1,
                 "high": 2 }

data["Location"] = data["Location"].map(data_encoding_object_rank)
'''
```

Summary after Data Encoding

In [ ]:
```
# Summary the data after encoding
'''
display(data)
data.info()
'''
```

2.5 - Exploratory Data Analysis

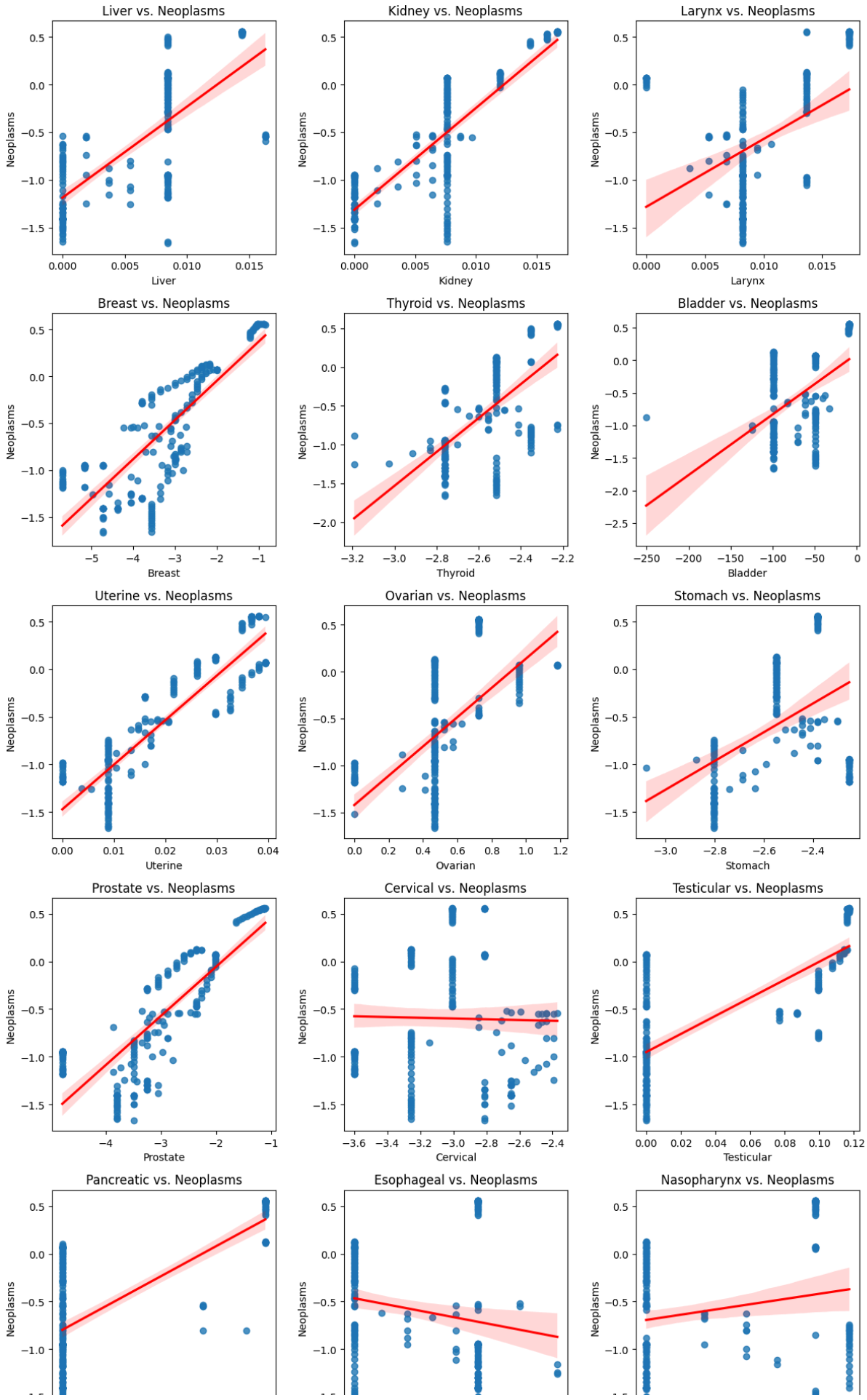
2.5.1 - Data Visualization Analysis

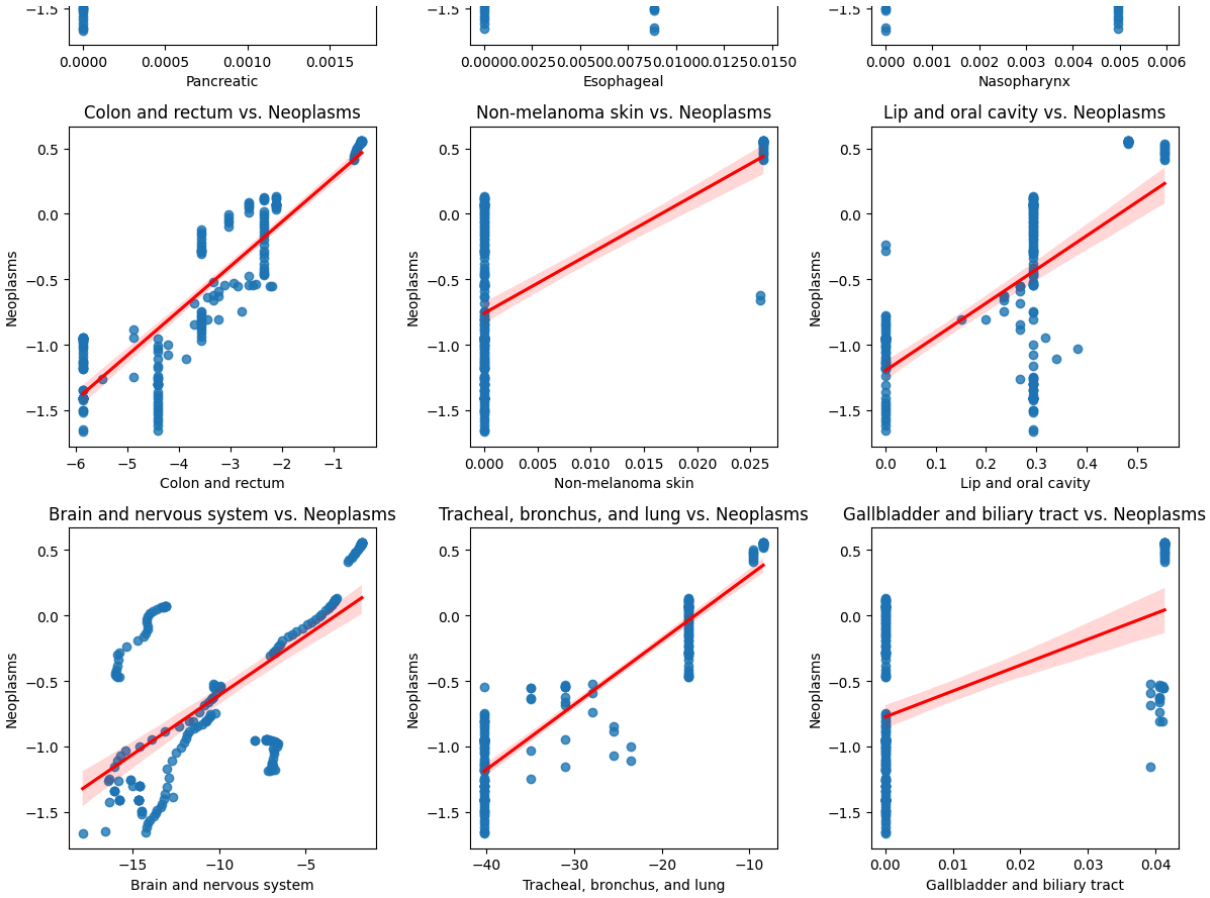
Data Visualization is an important component of Exploratory Data Analysis (EDA), because it helps us to understand the variables and relationships between them. These variables could be dependent or independent to each other.

Univariate Analysis	Bivariate Analysis	Multivariate Analysis
It only summarize single variable at a time	It only summarize two variables	It only summarize more than 2 variables
It does not deal with causes and relationships	It does deal with causes and relationships and analysis is done	It does not deal with causes and relationships and analysis is done
The main purpose is to describe	The main purpose is to explain	The main purpose is to study the relationship among them

For this section, we focus on the bivariate analysis to analyzing the relationship between the two variables are positive and negative, or show no clear pattern.

In [ ]:
```
#-------------------------------------------------------------------------
# The scatter plots conclude linear fitting line
#-------------------------------------------------------------------------

# Initialization the subplots
_, ax = plt.subplots(nrows=7, ncols=3, figsize=(12, 28))
ax = ax.ravel()

# Draw the scatter plots conclude linear fitting line
for index, column in enumerate(data_numeric.drop(columns=["Neoplasms"]).colu
  sns.regplot(data, ax=ax[index], x=column, y="Neoplasms", line_kws={"color"
  ax[index].set_xlabel(column)
  ax[index].set_ylabel("Neoplasms")
  ax[index].set_title(column + " vs. Neoplasms")

# Do not blocked any title or label
plt.tight_layout()
plt.show()
```


Liver vs. Neoplasms
Liver
Neoplasms
Kidney vs. Neoplasms
Kidney
Neoplasms
Larynx vs. Neoplasms
Larynx
Neoplasms
Breast vs. Neoplasms
Breast
Neoplasms
Thyroid vs. Neoplasms
Thyroid
Neoplasms
Bladder vs. Neoplasms
Bladder
Neoplasms
Uterine vs. Neoplasms
Uterine
Neoplasms
Ovarian vs. Neoplasms
Ovarian
Neoplasms
Stomach vs. Neoplasms
Stomach
Neoplasms
Prostate vs. Neoplasms
Prostate
Neoplasms
Cervical vs. Neoplasms
Cervical
Neoplasms
Testicular vs. Neoplasms
Testicular
Neoplasms
Pancreatic vs. Neoplasms
Neoplasms
Esophageal vs. Neoplasms
Neoplasms
Nasopharynx vs. Neoplasms
Neoplasms



From the above plots we have the following analysis.

- Increasing values for the following features are associated with the Neoplasms outcome:

    1. Breast
    2. Uterine
    3. Prostate
    4. Colon and rectum
    5. Brain and nervous system
- Decreasing values for the following features are associated with the Neoplasms outcome:

    - None
- The values for the following features are not associated with the Neoplasms outcome:

    1. Liver
    2. Kidney
    3. Larynx
    4. Thyroid
    5. Bladder
    6. Ovarian

7. Stomach
8. Cervical
9. Testicular
10. Pancreatic
11. Esophageal
12. Nasopharynx
13. Non-melanoma skin
14. Lip and oral cavity
15. Tracheal, bronchus, and lung
16. Gallbladder and biliary tract

2.5.2 - Hypothesis Testing

In terms of a P-value and a chosen significance level (alpha):

- If P-value <= alpha (usually 5%): significant result, reject null hypothesis (H0), dependent
- If P-value > alpha (usually 5%): not significant result, fail to reject null hypothesis (H0), independent

Hypothesis Testing

We seek to investigate whether there are significant differences in the prevalence of liver cancer and kidney cancer worldwide over the past 30 years.

To evaluate this, we formulated the following hypotheses:

- **Null (H0)**: There are no significant differences in the prevalence of liver cancer and kidney cancer across different regions worldwide.
- **Alternative (H1)**: There are significant differences in the prevalence of liver cancer and kidney cancer across different regions worldwide.

In [ ]:
```
# Get the data with type numeric
data_numeric = data.select_dtypes(include='number')
```

In [ ]:
```
# Split into two group by Rain Tomorrow 0 or 1
group_0 = data_numeric["Liver"]
group_1 = data_numeric["Kidney"]

# Calculate the T-test value and P-value by Welch"s t-test
data_t_test = scipy.stats.ttest_ind(group_0, group_1, equal_var=True)

print("T-test value: " + str(data_t_test[0]))
print("P-value: " + str(data_t_test[1]))
```

```
T-test value: -1.0771503729693153
P-value: 0.28205269389510274
```

The P-values is greater than 0.05 (5%), we do not reject the null hypothesis of the T-test and conclude that there is insignificant difference in the prevalence of liver cancer and kidney cancer across different countries over the past 30 years.

---

3. MODELING

Preparing the Clustering Data

In [ ]:
```
# Preparing the clustering data
data_clustering = data_numeric.drop(columns=["Neoplasms"], axis=1)
```

3.1 - Evaluation Metric

The following metrics are widely used in machine learning to evaluate clustering performance:

- **Davies Bouldin Score**
  - The score is defined as the average similarity measure of each cluster with its most similar cluster, where similarity is the ratio of within-cluster distances to between-cluster distances. Thus, clusters which are farther apart and less dispersed will result in a better score
  - The minimum score is 0, with lower values indicating better clustering
- **Silhouette Score**
  - A metric to evaluate the quality of clustering is referred to as silhouette analysis. Silhouette analysis can be applied to other clustering algorithms as well. Silhouette coefficient ranges between −1 and 1, where a higher silhouette coefficient refers to a model with more coherent clusters
  - Score close to +1 means the sample is far away from the neighboring clusters. A value of 0 means that the sample is on or very close to the decision boundary between two neighboring clusters. Finally, negative values indicate that the samples could have potentially been assigned to the wrong cluster
- **Calinski Harabasz Score**

- This is also known as the **Variance Ratio Criterion**, this measures the ratio of between-cluster variance and within-cluster variance. The higher the calinski-harabasz ratio, the more well-defined a cluster is

In [ ]:
```
# Setup the function to evaluate the clustering performance
def evaluation_clustering_model(clusters_num, acutal, predict):
  scores = pd.DataFrame( [ [ str(clusters_num),
                  davies_bouldin_score(acutal, predict),
                  silhouette_score(acutal, predict),
                  calinski_harabasz_score(acutal, predict) ] ],
                columns = [ "Clusters",
                        "Davies Bouldin Score",
                        "Silhouette Score",
                        "Calinski Harabasz Score" ] )
  return scores

clustering_model_scores = {}
```

3.2 - Centroid-Based Clustering

Centroid-Based Clustering is a type of clustering method that partitions or splits a data set into similar groups based on the distance between their centroids. Each cluster's centroid, or center, is either the mean or median of all the points in the cluster depending on the data.

One of the most commonly used centroid-based clustering techniques is:

- **K-Means**

3.2.1 - K-Means

K-Means is a hard clustering approach, meaning each data point is assigned to a separate cluster and no probability associated with cluster membership.

- K-means assumes that the center of each cluster defines the cluster using a distance measure, mostly commonly Euclidean distance, to the centroid
- To initialize the clustering, we provide a number of expected clusters, which represents the 'K' in K-means, and the algorithm attempts to find reasonable clusters across the data to match that number
- The optimal k clusters in a given dataset is identified by iteratively minimizing the total distance between each point and its assigned cluster centroid
- K-means works well when the clusters are of roughly equivalent size, and there are not significant outliers or changes in density across the data.

- K-means often performs poorly when the data is high dimensional or when clusters have significantly different sizes or densities.
- K-means is also especially sensitive to outliers since it tries to establish centroids based on the mean values of all values in the cluster and thus is susceptible to overfitting to include those outliers.

Hyperparameter:

- **N Clusters**: The number of clusters to form as well as the number of centroids to generate.
- **Init**: `k-means++` means selects initial cluster centroids using sampling based on an empirical probability distribution of the points' contribution to the overall inertia, and it speeds up convergence; `random` means choose **N Clusters** observations at random from data for the initial centroids.
- **N Init**: Number of times the k-means algorithm is run with different centroid seeds. The final results is the best output of n_init consecutive runs in terms of inertia. `auto` means number of runs depends on the value of **Init**. when `init='random'` then we have 10; when `init='k-means++'` then we have 1

In [ ]:
```
#-----------------------------------------------------------------------
# Centroid-Based Clustering - K-Means
#-----------------------------------------------------------------------

clustering_model_score = []

# We testing K from 2 to 10
for cluster in range(2, 11):
  model = KMeans(n_clusters=cluster, init='k-means++', n_init='auto')
  model.fit(data_clustering)
  clustering_model_score.append(evaluation_clustering_model(cluster, data_cl
```

Display the Performance of Different Clusters

In [ ]:
```
# Display the table of result
clustering_model_scores["K-Means"] = pd.concat(clustering_model_score)

clustering_model_scores["K-Means"].sort_values(by="Davies Bouldin Score", as
```

Out[ ]:

	Clusters	Davies Bouldin Score	Silhouette Score	Calinski Harabasz Score
0	9	0.357643	0.757626	1843.756030
0	8	0.375306	0.740643	1537.783295
0	7	0.405525	0.692428	1182.452710
0	3	0.421777	0.710483	472.113904
0	6	0.428827	0.724954	855.256414
0	10	0.433904	0.754507	1854.997254
0	5	0.491501	0.697244	415.767724
0	4	0.502071	0.700640	465.780362
0	2	0.588264	0.652830	420.710941

3.3 - Hierarchical Clustering

Hierarchical Clustering, sometimes called connectivity-based clustering, groups data points together based on the proximity and connectivity of their attributes.

This method determines clusters based on how close data points are to one another across all of the dimensions. The idea is that objects that are nearer are more closely related than those that are far from each other.

Unlike k-means, there is no need to pre-specify the number of clusters. Instead, the clustering algorithm creates a graph network of the clusters at each hierarchical level. This network is hierarchical, meaning that any given node in it only has one parent node but may have multiple child nodes.

Hierarchical clusters can be graphed with a dendrogram to help visually summarize and organize discovered clusters and the hierarchy that they may contain.

There are two approaches to performing hierarchical cluster analysis:

- **Agglomerative**
- **Divisive**

3.3.1 - Agglomerative

Agglomerative Clustering a bottom-up approach starts with individual data points and successively merges clusters by compute the proximity matrix of all the clusters at the current level of the hierarchy to create a tree-like structure. Once one level of clusters has been created where all the clusters have no or low inter-cluster similarity, the algorithm moves to the set of newly created clusters and repeats the process until there is one root node at the top of the hierarchical graph.

- In **Single-Linkage** clustering, the shortest distance between any pair of data points in two clusters is used as a similarity measure
- In **All-Pairs Linkage**, the average across all pairs of data points is used, whereas in sampled linkage, a sampling of the data points in the two clusters is used for calculating the average distance
- In **Centroid-Linkage**, the distance between the centroids is used

One challenge with agglomerative methods is that they can exhibit chaining, where larger clusters are naturally biased toward having closer distances to other points and so continue to get larger and larger and attract more data points into their cluster. Another disadvantage is that agglomerative methods may be much slower than divisive methods of constructing the hierarchy.

Hyperparameter:

- **N Clusters**: The number of clusters to form as well as the number of centroids to generate.
- **Linkage**: Which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion. `ward` minimizes the variance of the clusters being merged; `average` uses the average of the distances of each observation of the two sets. `complete` linkage uses the maximum distances between all observations of the two sets; `single` uses the minimum of the distances between all observations of the two sets.

In [ ]:
```
#------------------------------------------------------------------------------
# Hierarchical Clustering - Agglomerative
#------------------------------------------------------------------------------

clustering_model_score = []

# We testing clusters from 2 to 10
for cluster in range(2, 11):
  model = AgglomerativeClustering(n_clusters=cluster, linkage='ward')
  model.fit(data_clustering)
  clustering_model_score.append(evaluation_clustering_model(cluster, data_cl
```

Display the Performance of Different Clusters

In [ ]:
```
# Display the table of result
clustering_model_scores["Agglomerative"] = pd.concat(clustering_model_score)

clustering_model_scores["Agglomerative"].sort_values(by="Davies Bouldin Scor
```

Out[ ]:

	Clusters	Davies Bouldin Score	Silhouette Score	Calinski Harabasz Score
0	9	0.319015	0.782969	1896.549964
0	5	0.327571	0.716101	882.356161
0	10	0.333955	0.773743	2330.407012
0	4	0.342759	0.709330	586.607371
0	8	0.346269	0.768805	1651.386286
0	6	0.347952	0.709507	1092.128352
0	7	0.378543	0.753489	1492.111088
0	3	0.438564	0.698432	458.676186
0	2	0.600630	0.638025	392.408289

3.3.2 - Divisive

Divisive Hierarchical Clustering methods is a top-down approach successively partitions the data points into a tree-like structure. Divisive partitioning allows greater flexibility in terms of both the hierarchical structure of the tree and the level of balance in the different clusters. Divisive hierarchical clustering can be faster than agglomerative hierarchical clustering, especially when the data doesn't require constructing the tree all the way down to individual data points.

Bisecting K-Means is a kind of hierarchical clustering using a divisive approach. It build a random binary tree where each splitting (a node with two children) corresponds to splitting the points of dataset into 2 clusters using a flat-clustering method like K-Means. The clusters with the largest Sum of Squared Errors (SSE) are then partitioned further using a flat clustering method. The algorithm stops either when it reaches individual nodes or some minimum SSE.

Hyperparameter:

- **N Clusters**: The number of clusters to form as well as the number of centroids to generate
- **Init**: `k-means++` means selects initial cluster centers for k-mean clustering in a smart way to speed up convergence; `random` means choose **N Clusters** observations at random from data for the initial centroids

- **N Init**: Number of time the inner k-means algorithm will be run with different centroid seeds in each bisection

In [ ]:
```
#-----------------------------------------------------------------------
# Hierarchical Clustering - Divisive - Bisecting K-Means
#-----------------------------------------------------------------------

clustering_model_score = []

# We testing K from 2 to 10
for cluster in range(2, 11):
  model = BisectingKMeans(n_clusters=cluster, init='k-means++', n_init=1)
  model.fit(data_clustering)
  clustering_model_score.append(evaluation_clustering_model(cluster, data_cl
```

Display the Performance of Different Clusters

In [ ]:
```
# Display the table of result
clustering_model_scores["Divisive - Bisecting K-Means"] = pd.concat(clusteri

clustering_model_scores["Divisive - Bisecting K-Means"].sort_values(by="Davi
```

Out[ ]:

	Clusters	Davies Bouldin Score	Silhouette Score	Calinski Harabasz Score
0	9	0.327719	0.741308	1586.114776
0	6	0.352102	0.701951	1063.569787
0	5	0.362036	0.699513	872.457295
0	3	0.414133	0.701823	450.192239
0	8	0.417317	0.741552	1455.961632
0	10	0.491328	0.728800	1693.196015
0	7	0.521164	0.651643	1017.762416
0	4	0.528144	0.682457	445.173151
0	2	0.588264	0.652830	420.710941

3.4 - Distribution-Based Clustering

Distribution-Based Clustering, sometimes called probabilistic clustering, groups together data points based on their probability distribution. This approach assumes that there is a process generating normal distributions for each dimension of the data which create the cluster centers.

It's different from centroid-based clustering in that it doesn't use a distance metric like a Euclidean or Manhattan distance. Instead, distribution based

approaches look for a well-defined distribution which appears across each dimension.

The cluster means are the means of the Gaussian distribution across each dimension. Distribution based clustering is a model-based approach to clustering because it requires fitting a distribution multiple times across each dimension to find clusters, which means that it can be computationally expensive when working with large data sets.

One commonly used approach to distribution-based clustering is:

- **Gaussian Mixture**

3.4.1 - Gaussian Mixture

Gaussian Mixture Model is assumption that each cluster is defined by a normal distribution. Gaussian mixture operates on the principle that a complex, multi-modal distribution can be approximated by a combination of simpler Gaussian distributions, each representing a different cluster within the data. This is achieved through a process known as 'soft clustering', as opposed to 'hard clustering' methods like K-Means.

In soft clustering, instead of forcefully assigning a data point to a single cluster, gaussian mixture assigns probabilities that indicate the likelihood of that data point belonging to each of the Gaussian components. And the most powerful aspects of gaussian mixture is their capacity to compute the probability of each data point belonging to a particular cluster.

The essence of gaussian mixture lies in its ability to determine cluster characteristics such as mean, variance, and weight. Enables gaussian mixture to model the data with remarkable flexibility. By adjusting these parameters, a gaussian mixture can shape itself to fit a wide variety of data distributions, whether they are tightly clustered, widely dispersed, or overlapping with one another.

- The mean of each gaussian component give a central point, around which the data points are most densely clustered
- The variance provides insight into the spread or dispersion of the data points around this mean. A smaller variance indicates that the data points are closely clustered around the mean, while a larger variance suggests a more spread-out cluster
- The weights in a gaussian mixture are particularly significant. They represent the proportion of the dataset that belongs to each gaussian component.

Hyperparameter:

- **N Components**: The number of mixture components
- **Covariance Type**: String describing the type of covariance parameters to use. `full` means each component has its own general covariance matrix; `tied` means all components share the same general covariance matrix; `diag` means each component has its own diagonal covariance matrix; `spherical` means each component has its own single variance
- **Init Params**: The method used to initialize the weights, the means and the precisions. `kmeans` responsibilities are initialized using kmeans; `k-means++` use the k-means++ method to initialize; `random` responsibilities are initialized randomly; `random_from_data` initial means are randomly selected data points

In [ ]:
```
#-------------------------------------------------------------------------
# Distribution-Based Clustering - Gaussian Mixture
#-------------------------------------------------------------------------

clustering_model_score = []

# We testing K from 2 to 10
for cluster in range(2, 11):
  model = GaussianMixture(n_components=cluster, covariance_type='full', init
  model.fit(data_clustering)
  clustering_model_score.append(evaluation_clustering_model(cluster, data_cl
```

Display the Performance of Different Clusters

In [ ]:
```
# Display the table of result
clustering_model_scores["Gaussian Mixture"] = pd.concat(clustering_model_scc

clustering_model_scores["Gaussian Mixture"].sort_values(by="Davies Bouldin S
```

Out[ ]:

	Clusters	Davies Bouldin Score	Silhouette Score	Calinski Harabasz Score
0	9	0.317987	0.773603	1838.512348
0	5	0.331068	0.720288	967.514936
0	8	0.348381	0.768029	1658.830414
0	6	0.388041	0.708564	1162.827262
0	3	0.438564	0.698432	458.676186
0	10	0.459418	0.749034	1896.373459
0	7	0.465637	0.713569	1128.149042
0	4	0.502071	0.700640	465.780362
0	2	0.590217	0.650480	414.085265

3.5 - Density-Based Clustering

Density-Based Clustering works by detecting areas where points are concentrated and where they are separated by areas that are empty or sparse. Unlike centroid based approaches, like K-means, or distribution-based approaches, like Expectation Maximization, density-based clustering can detect clusters of an arbitrary shape. This can be extremely helpful when clusters aren't defined around a specific location or distribution.

Unlike other clustering algorithms, such as K-means and hierarchical clustering, a density-based algorithm can discover clusters of any shape, size, or density in your data. Density-based clustering also can distinguish between data points which are part of a cluster and those which should be labeled as noise. Density-based clustering is especially useful when working with datasets with noise or outliers or when we don't have prior knowledge about the number of clusters in the data.

One of clustering algorithms which takes a density-based approach to clustering:

- **Density-Based Spatial Clustering of Applications with Noise (DBSCAN)**

3.5.1 - DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) uses a density-based spatial clustering approach to create clusters with a density passed in by the user which centers around a spatial centroid. The area immediately around the centroid is referred to as a neighborhood and DBSCAN attempts to define neighborhoods of clusters that have the specified density. For each cluster, DBSCAN will define three types of data points:

- **Core Points**: A data point is a core point if the neighborhood around that data point contains at least as many points as the user specified minimum number of points.
- **Border Points**: A data point is a border point if the neighborhood around that data point contains less than the minimum number of data points but the neighborhood around that point contains a core point.
- **Outlier**: A data point is an outlier if it is neither a core point nor a border point. Essentially, this is the other class.

Hyperparameter:

- **Eps**: The maximum distance between two samples for one to be considered as in the neighborhood of the other. This is not a maximum bound on the distances of points within a cluster. This is the most important DBSCAN parameter to choose appropriately for your data set and distance function.
- **Min Samples**: The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself. If min_samples is set to a higher value, DBSCAN will find denser clusters, whereas if it is set to a lower value, the found clusters will be more sparse

In [ ]:
```
#-----------------------------------------------------------------------
# Density-Based Spatial Clustering of Applications with Noise (DBSCAN)
#-----------------------------------------------------------------------

clustering_model_score = []

# We testing K from 2 to 10
for cluster in range(2, 11):
  model = DBSCAN(eps=0.1, min_samples=cluster)
  model.fit(data_clustering)
  clustering_model_score.append(evaluation_clustering_model(cluster, data_cl
```

Display the Performance of Different Clusters

In [ ]:
```
# Display the table of result
clustering_model_scores["DBSCAN"] = pd.concat(clustering_model_score)

clustering_model_scores["DBSCAN"].sort_values(by="Davies Bouldin Score", asc
```

Out[ ]:

	Clusters	Davies Bouldin Score	Silhouette Score	Calinski Harabasz Score
0	9	0.422893	0.396578	42.460410
0	10	0.422893	0.396578	42.460410
0	5	0.941488	-0.133882	21.119246
0	8	0.954702	-0.276675	23.745818
0	7	0.995497	-0.238643	17.730825
0	3	0.997801	-0.130864	13.266700
0	4	1.000265	-0.126813	15.886881
0	6	1.002452	-0.215033	21.620486
0	2	1.011354	0.051079	9.356127

---

4. SUMMARY

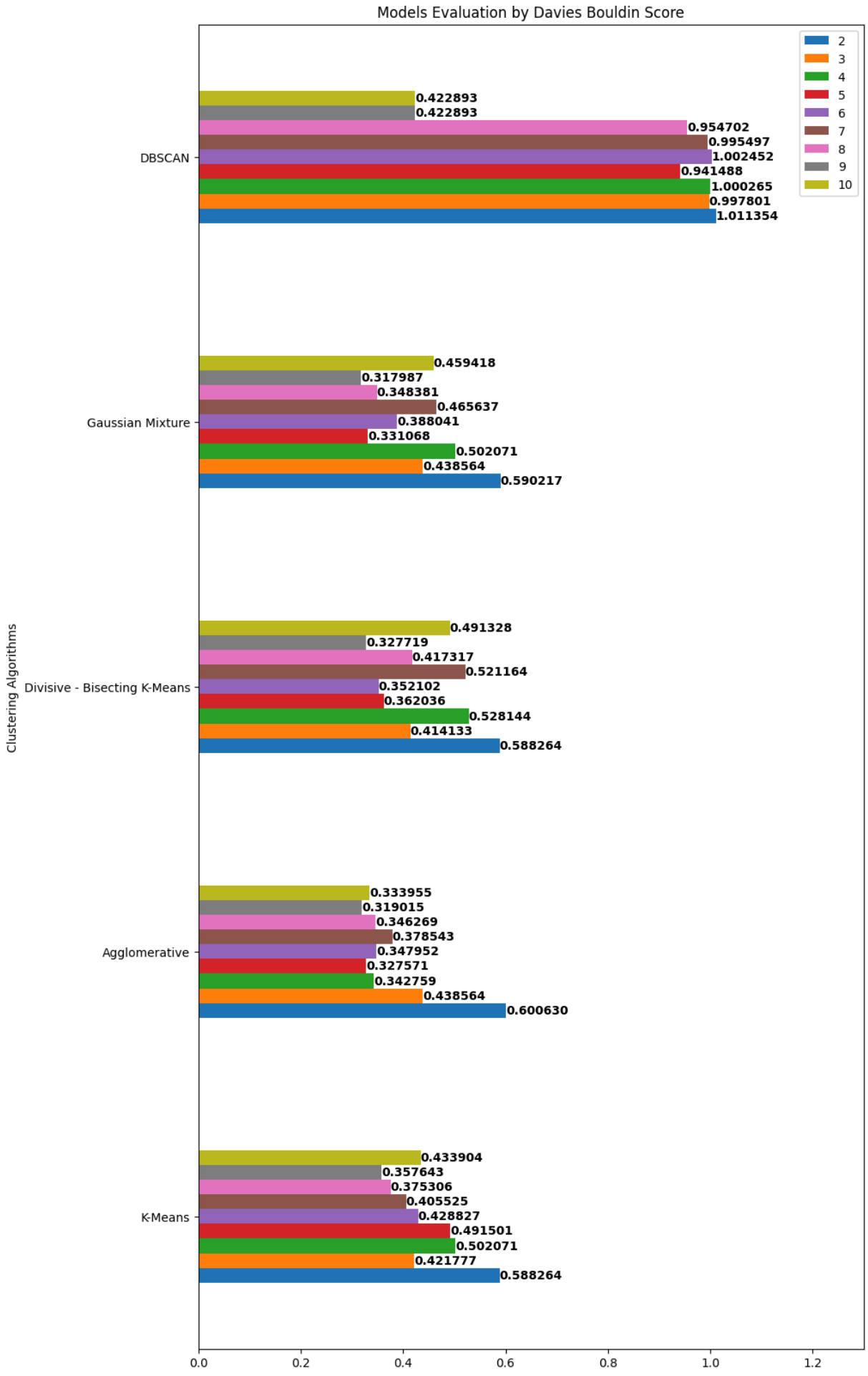
4.1 - Models Evaluation

We will now compare the prediction results of unsupervised model with various clustering algorithms on the data. The most suitable algorithm for the model will be selected based on metrics such as Davies Bouldin Score, Silhouette Score and Calinski Harabasz Score.

In [ ]:
```
# Split the metric values of train set and test set
model_scores = pd.concat( clustering_model_scores )
```

Clustering Values Evaluation Based on the Davies Bouldin Score

In [ ]:
```
model_scores_dbs_clusters = pd.DataFrame( { name : model_scores["Davies Boul
                        index = [name for name, _ in model_scores.index.uniqu


ax = model_scores_dbs_clusters.plot.barh( figsize = (10, 20),
                      xlabel = "Davies Bouldin Score",
                      ylabel = "Clustering Algorithms",
                      title = "Models Evaluation by Davies Bouldin Score",
                      xlim = [0.0, 1.3] )

for container in ax.containers:
  ax.bar_label(container, fmt='%.6f', fontweight='bold')

plt.show()
```

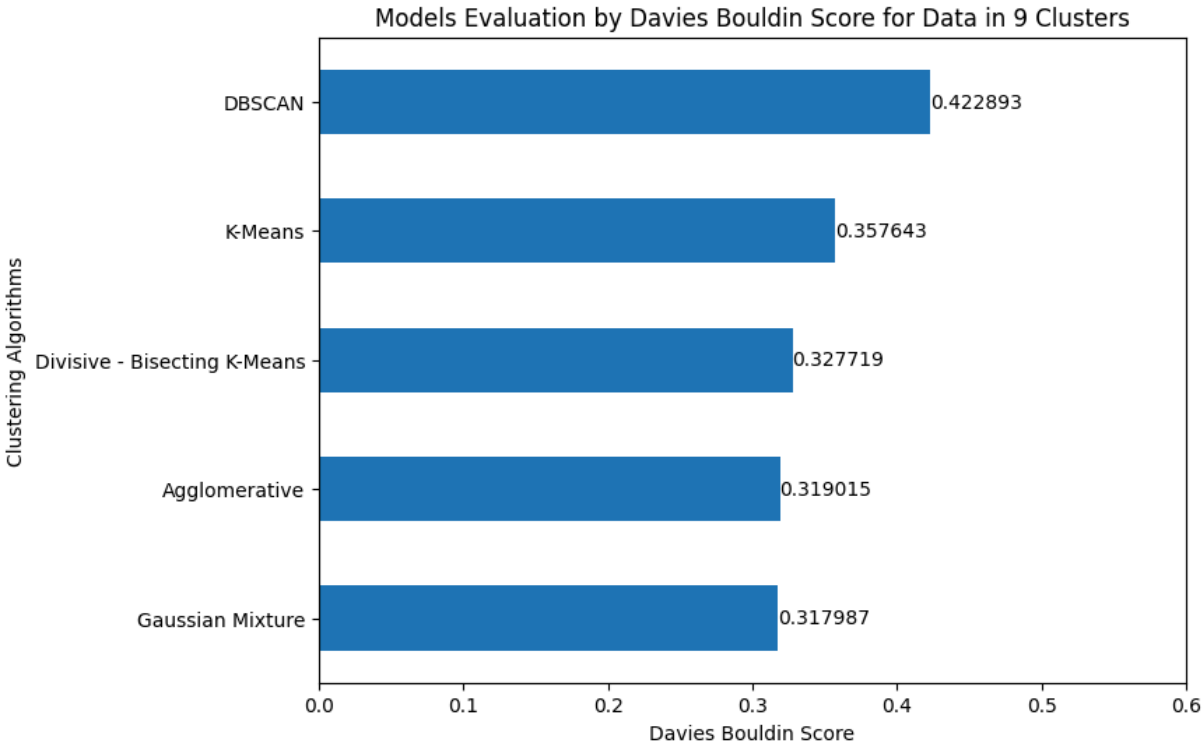

Models Evaluation by Davies Bouldin Score
Clustering Algorithms
DBSCAN
Gaussian Mixture
Divisive - Bisecting K-Means
Agglomerative
K-Means
2
3
4
5
6
7
8
9
10
0.422893
0.422893
0.954702
0.995497
1.002452
0.941488
1.000265
0.997801
1.011354
0.459418
0.317987
0.348381
0.465637
0.388041
0.331068
0.502071
0.438564
0.590217
0.491328
0.327719
0.417317
0.521164
0.352102
0.362036
0.528144
0.414133
0.588264
0.333955
0.319015
0.346269
0.378543
0.347952
0.327571
0.342759
0.438564
0.600630
0.433904
0.357643
0.375306
0.405525
0.428827
0.491501
0.502071
0.421777
0.588264
0.0
0.2
0.4
0.6
0.8
1.0
1.2

Davies Bouldin Score

Based on the findings from section **3. MODELING** and the accompanying plot, we determined that the optimal clustering value among the five algorithms tested is 9, as it consistently ranked first across all five algorithms.

Models Evaluation Based on the Davies Bouldin Score with Cluster of 9

In [ ]:
```
model_scores_dbs_clusters_9 = model_scores_dbs_clusters['9'].sort_values(asc

ax = model_scores_dbs_clusters_9.plot.barh( figsize = (8, 6),
                          xlabel = "Davies Bouldin Score",
                          ylabel = "Clustering Algorithms",
                          title = "Models Evaluation by Davies Bouldin Score f
                          xlim = [0.0, 0.6] )

for container in ax.containers:
  ax.bar_label(container, fmt='%.6f')

plt.show()
```

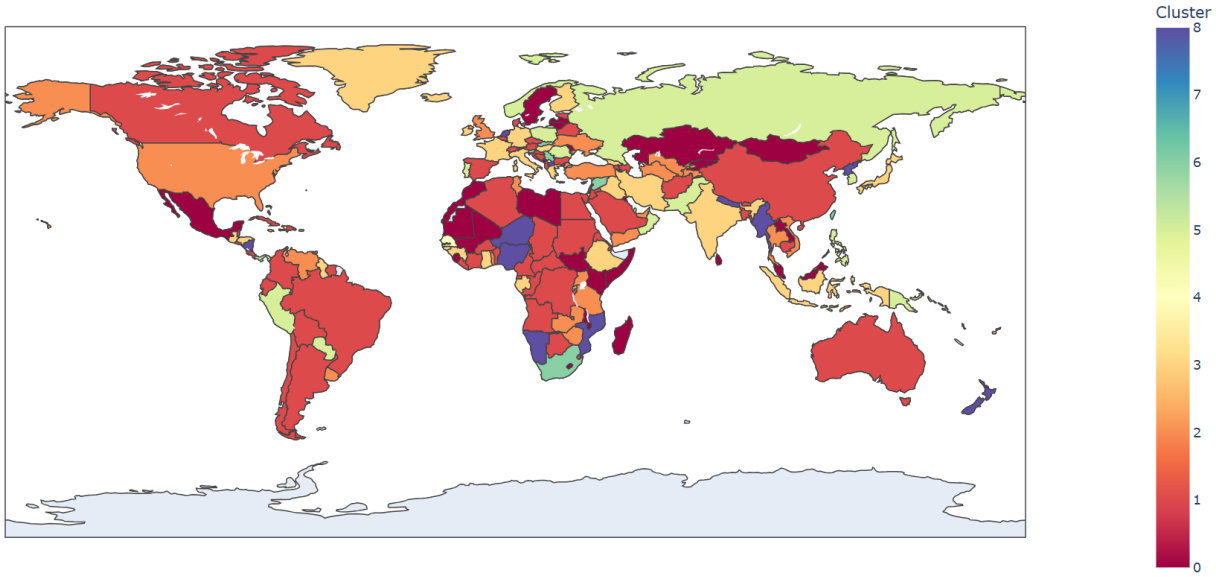


From the figure above, it is evident that the performance of agglomerative clustering and Gaussian mixture models is very similar, with the Gaussian mixture model performing slightly better. Therefore, we have selected the Gaussian mixture model with the optimal clustering value of 9 as our final unsupervised model.

Visualize the Clustering Performance of the Final Model

In [ ]:

```
# Get the model on the cluster we selected
model = GaussianMixture(n_components=9, covariance_type='full', init_params=
model.fit(data_clustering)

# Labelled data using the final clustering model
data["Cluster"] = model.predict(data_clustering)

# Loading world counties map
with open('/content/countries.geo.json') as json_file:
  file_contents = json_file.read()
world_counties_map = json.loads(file_contents)

# Draw the world counties map
fig = px.choropleth( data,
             geojson = world_counties_map,
             locations = "Code",
             color = "Cluster",
             color_continuous_scale = "spectral",
             range_color = (0, 8),
             hover_data = ["Country"] )

fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```



4.2 - Summary

Based on the analysis in the preceding sections, the most suitable model for our project is the **Gaussian Mixture** model, utilizing distribution-based clustering with 9 clusters.

Although the Gaussian mixture clustering algorithm, which calculates data probabilities within Gaussian components, outperformed the other four algorithms, its complexity presents a challenge. With 20 features, visually representing how the Gaussian model clusters the data becomes difficult, which is a significant drawback. In contrast, algorithms like K-Means, which offer easier comparability, could enhance post-clustering analysis.

In the future, we should incorporate Principal Components Analysis (PCA) to facilitate kernel-based clustering evaluations, which may yield improved performance.

---

5. REFERENCES

Sources	Article	Author
Book	Practitioner's Guide to Data Science	Hui Lin & Ming Li
Book	Machine Learning Guide for Oil and Gas Using Python	Hoss Belyadi & Alireza Haghighat
Publication	A Comparison of Document Clustering Techniques	Michael Steinbach
Kaggle	Cancer and Deaths Dataset : 1990 to 2019 Globally	Belayet Hossain
Kaggle	PowerTransformers In-Depth Understanding(Box -Cox & Yeo Johnson)	Abhishek Kukreja
IBM	What is unsupervised learning?	IBM Team
IBM	What is clustering?	IBM Team
Medium	How to Remove Outliers for Machine Learning?	Anuganti Suresh
Medium	Categorical Data Encoding Techniques	Krishnakanth Jarapala

Sources	Article	Author
Medium	Correlation in machine learning - All you need to know	Abdallah Ashraf
Medium	When to Use Mean, Median, and Mode for Imputing Missing Values	Chandrikasai
Medium	Handling Missing Data with KNN Imputer	Bhanupsingh
Medium	Multivariate Imputation by Chained-Equations (MICE)	Kunal
Medium	Hypothesis Testing with Python: T-Test, Z-Test, and P-Values	Praise James
Medium	Five Methods for Data Splitting in Machine Learning	Gen. David
Medium	Understanding Gaussian Mixture Models: A Comprehensive Guide	Juan Olamendy
GeeksforGeeks	Univariate, Bivariate and Multivariate data and its analysis	Aaradhana Thapliyal
Scribbr	Correlation Coefficient - Types, Formulas & Examples	Pritha Bhandari
TurinTech AI	Data Quality in Machine Learning: How to Evaluate and Improve?	Chrystalla Pavlou
Tableau	Guide To Data Cleaning: Definition, Benefits, Components, And How To Clean Your Data	Tableau Team
GitHub	world.geo.json	Johan Sundström
Plotly	Choropleth Maps in Python	Plotly Team