Name:Bhushan Mukund Kor Academic Year:2024-2025

Division: D15C Roll No: 28

Aim: Introduction to Data science and Data preparation using Pandas steps.

1. Load data in Pandas.
2. Description of the dataset.
3. Drop columns that aren't useful.
4. Drop rows with maximum missing values.
5. Take care of missing data.
6. Create dummy variables.
7. Find out outliers (manually)
8. standardization and normalization of columns

Steps:

Step 1: Load the File

Pandas library is used to analyze data which provides powerful tools for handling data. It allows us to read **CSV (Comma Separated Values) files** efficiently and perform various data manipulation and analysis tasks.

Commands: import pandas as pd (To Import the pandas library onto Google Colab Notebook)

df = pd.read_csv(<Path_of_csv_file>) (Mounts and reads the file in Python and assigns it to variable df for ease of use further)
(Note: Replace <Path_of_csv_file> with the actual path of the file in “”)

```
[1] import pandas as pd
```

```
[2] df = pd.read_csv("/content/drive/MyDrive/Semester 6/AIDS/AIDS Lab/Alzheimer_s_Disease_and_Healthy_Aging_Data.csv")
```

Run command **df.info()** to check whether the file is loaded properly or not .This will print first 5 Rows with all Columns.

```
df.head()
```

	RowId	YearStart	YearEnd	LocationAbbr	LocationDesc	Datasource	Class	Topic	Question	Data_Value_Unit	...	Stratification2	Geolocation	ClassID	TopicII
0	BRFSS~2022~2022~42~Q03~TMC01~AGE~RACE	2022	2022	PA	Pennsylvania	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Native Am/Alaskan Native	POINT (-77.86070029 40.79373015)	C05	TMC0
1	BRFSS~2022~2022~46~Q03~TMC01~AGE~RACE	2022	2022	SD	South Dakota	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Asian/Pacific Islander	POINT (-100.3735306 44.35313005)	C05	TMC0
2	BRFSS~2022~2022~16~Q03~TMC01~AGE~RACE	2022	2022	ID	Idaho	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Black, non-Hispanic	POINT (-114.36373 43.68263001)	C05	TMC0
3	BRFSS~2022~2022~24~Q03~TMC01~AGE~RACE	2022	2022	MD	Maryland	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Black, non-Hispanic	POINT (-76.60926011 39.29058096)	C05	TMC0
4	BRFSS~2022~2022~55~Q03~TMC01~AGE~GENDER	2022	2022	WI	Wisconsin	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Male	POINT (-89.81637074 44.39319117)	C05	TMC0

5 rows × 31 columns

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Step 2: Description of the dataset

1)df.head() : This command is use to print first 5 Rows and all the columns.

```
df.head()
```

	RowId	YearStart	YearEnd	LocationAbbr	LocationDesc	Datasource	Class	Topic	Question	Data_Value_Unit	...	Stratification2	Geolocation	ClassID	TopicII
0	BRFSS~2022~2022~42~Q03~TMC01~AGE~RACE	2022	2022	PA	Pennsylvania	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Native Am/Alaskan Native	POINT (-77.86070029 40.79373015)	C05	TMC0'
1	BRFSS~2022~2022~46~Q03~TMC01~AGE~RACE	2022	2022	SD	South Dakota	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Asian/Pacific Islander	POINT (-100.3735306 44.35313005)	C05	TMC0'
2	BRFSS~2022~2022~16~Q03~TMC01~AGE~RACE	2022	2022	ID	Idaho	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Black, non-Hispanic	POINT (-114.36373 43.68263001)	C05	TMC0'
3	BRFSS~2022~2022~24~Q03~TMC01~AGE~RACE	2022	2022	MD	Maryland	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Black, non-Hispanic	POINT (-76.60926011 39.29058096)	C05	TMC0'
4	BRFSS~2022~2022~55~Q03~TMC01~AGE~GENDER	2022	2022	WI	Wisconsin	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Male	POINT (-89.81637074 44.39319117)	C05	TMC0'

5 rows × 31 columns

2)df.info() : This command is use to print all column names with count of non-null values are their data-type.

```
[4] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284142 entries, 0 to 284141
Data columns (total 31 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   RowId                       284142 non-null  object
 1   YearStart                   284142 non-null  int64
 2   YearEnd                     284142 non-null  int64
 3   LocationAbbr                284142 non-null  object
 4   LocationDesc                284142 non-null  object
 5   Datasource                  284142 non-null  object
 6   Class                       284142 non-null  object
 7   Topic                       284142 non-null  object
 8   Question                    284142 non-null  object
 9   Data_Value_Unit             284142 non-null  object
 10  DataValueTypeID             284142 non-null  object
 11  Data_Value_Type             284142 non-null  object
 12  Data_Value                  192808 non-null  float64
 13  Data_Value_Alt              192808 non-null  float64
 14  Data_Value_Footnote_Symbol  109976 non-null  object
 15  Data_Value_Footnote         109976 non-null  object
 16  Low_Confidence_Limit        192597 non-null  float64
 17  High_Confidence_Limit       192597 non-null  float64
 18  StratificationCategory1     284142 non-null  object
 19  Stratification1             284142 non-null  object
 20  StratificationCategory2     247269 non-null  object
 21  Stratification2             247269 non-null  object
 22  Geolocation                 253653 non-null  object
 23  ClassID                     284142 non-null  object
 24  TopicID                     284142 non-null  object
 25  QuestionID                  284142 non-null  object
 26  LocationID                  284142 non-null  int64
 27  StratificationCategoryID1   284142 non-null  object
 28  StratificationID1           284142 non-null  object
 29  StratificationCategoryID2   284142 non-null  object
 30  StratificationID2           284142 non-null  object
dtypes: float64(4), int64(3), object(24)
memory usage: 67.2+ MB
```

3)df.describe(): This command is use to print count,mean,std,min,25%,50%,75% and max of all columns which have data type int or float

```
[5] df.describe()
```

	YearStart	YearEnd	Data_Value	Data_Value_Alt	Low_Confidence_Limit	High_Confidence_Limit	LocationID
count	284142.000000	284142.000000	192808.000000	192808.000000	192597.000000	192597.000000	284142.000000
mean	2018.596065	2018.657735	37.676757	37.676757	33.027824	42.595333	800.322677
std	2.302815	2.360105	25.213484	25.213484	24.290016	26.156408	2511.564977
min	2015.000000	2015.000000	0.000000	0.000000	-0.700000	1.300000	1.000000
25%	2017.000000	2017.000000	15.900000	15.900000	12.600000	19.700000	19.000000
50%	2019.000000	2019.000000	32.800000	32.800000	27.000000	38.900000	34.000000
75%	2021.000000	2021.000000	56.900000	56.900000	49.400000	64.600000	49.000000
max	2022.000000	2022.000000	100.000000	100.000000	99.600000	100.000000	9004.000000

If the parameter of include=”all” is included { df.describe(include=“all”)}, this includes even the non numeric values and gives some more information on fields such as count of unique values, top value, etc.

```
[6] df.describe(include="all")
```

	RowId	YearStart	YearEnd	LocationAbbr	LocationDesc	Datasource	Class	Topic	Question	Data_Value_Unit	...	Stratification2	Geolocation	Cl
count	284142	284142.000000	284142.000000	284142	284142	284142	284142	284142	284142	284142	...	247269	253653	2
unique	36046	NaN	NaN	59	59	1	7	39	39	2	...	7	54	
top	BRFSS~2022~2022~42~Q03~TMC01~AGE~RACE	NaN	NaN	US	United States, DC & Territories	BRFSS	Overall Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	White, non-Hispanic	POINT (-120.1550313 44.56744942)	
freq	15	NaN	NaN	6132	6132	284142	96753	11092	11092	262048	...	36450	5916	
mean	NaN	2018.596065	2018.657735	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
std	NaN	2.302815	2.360105	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
min	NaN	2015.000000	2015.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
25%	NaN	2017.000000	2017.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
50%	NaN	2019.000000	2019.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
75%	NaN	2021.000000	2021.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
max	NaN	2022.000000	2022.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	

11 rows × 31 columns

Step3 : Drop columns that aren’t useful.

To make the dataset cleaner to work with it we drop the columns that aren’t useful.

df.columns command is used to list down all the columns.

Using this command we will list down the column names and we will pass it to the next command to drop the columns.

df.drop(<column_names>, axis=1, inplace=True)

Replace column names with either the list created previously, or with the column names itself.

The inplace attribute takes care that the dataset will stay updated for the rest of the analysis.

After running these commands, we run the **df.columns** command once again to check with the list of column names.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

```
[7] df.columns

Index(['RowId', 'YearStart', 'YearEnd', 'LocationAbbr', 'LocationDesc',
       'Datasource', 'Class', 'Topic', 'Question', 'Data_Value_Unit',
       'DataValueTypeID', 'Data_Value_Type', 'Data_Value', 'Data_Value_Alt',
       'Data_Value_Footnote_Symbol', 'Data_Value_Footnote',
       'Low_Confidence_Limit', 'High_Confidence_Limit',
       'StratificationCategory1', 'Stratification1', 'StratificationCategory2',
       'Stratification2', 'Geolocation', 'ClassID', 'TopicID', 'QuestionID',
       'LocationID', 'StratificationCategoryID1', 'StratificationID1',
       'StratificationCategoryID2', 'StratificationID2'],
      dtype='object')
```

```
columns_to_drop = ["RowId", "LocationDesc", "Data_Value_Footnote_Symbol", "Data_Value_Footnote", "Geolocation"]
df.drop(columns_to_drop, axis = 1, inplace = True)
```

```
[9] df.columns

Index(['YearStart', 'YearEnd', 'LocationAbbr', 'Datasource', 'Class', 'Topic',
       'Question', 'Data_Value_Unit', 'DataValueTypeID', 'Data_Value_Type',
       'Data_Value', 'Data_Value_Alt', 'Low_Confidence_Limit',
       'High_Confidence_Limit', 'StratificationCategory1', 'Stratification1',
       'StratificationCategory2', 'Stratification2', 'ClassID', 'TopicID',
       'QuestionID', 'LocationID', 'StratificationCategoryID1',
       'StratificationID1', 'StratificationCategoryID2', 'StratificationID2'],
      dtype='object')
```

As observed here, the columns of RowId, LocationDesc, Data_Value_Footnote_Symbol, Data_Value_Footnote and Geolocation have been dropped.

Step 4 : Drop rows with maximum missing values.

It is important to drop the rows with maximum missing values as they would hinder the performance of the analysis and can lead to inaccuracies in the dataset.

df.describe(): This command will give the count of rows present in the dataset.

```
df.describe()
```

	YearStart	YearEnd	Data_Value	Data_Value_Alt	Low_Confidence_Limit	High_Confidence_Limit	LocationID
count	284142.000000	284142.000000	192808.000000	192808.000000	192597.000000	192597.000000	284142.000000
mean	2018.596065	2018.657735	37.676757	37.676757	33.027824	42.595333	800.322677
std	2.302815	2.360105	25.213484	25.213484	24.290016	26.156408	2511.564977
min	2015.000000	2015.000000	0.000000	0.000000	-0.700000	1.300000	1.000000
25%	2017.000000	2017.000000	15.900000	15.900000	12.600000	19.700000	19.000000
50%	2019.000000	2019.000000	32.800000	32.800000	27.000000	38.900000	34.000000
75%	2021.000000	2021.000000	56.900000	56.900000	49.400000	64.600000	49.000000
max	2022.000000	2022.000000	100.000000	100.000000	99.600000	100.000000	9004.000000

To remove the max missing data rows we follow the below steps:

1) Create a column called missing_count where the sum of all the cells having null values is stored.

Command: df["missing_count"] = df.isnull().sum(axis=1)

```
[10] df["missing_count"] = df.isnull().sum(axis=1)
```

2) The maximum value from this missing_count column is considered for how many rows we have to delete by checking how much it will affect the data and how much it will help in cleaning the data.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Command: max_missing = df["missing_count"].max()

```
[13] max_missing = df["missing_count"].max()
```

```
[14] print(max_missing)
```

3) Finally, we update the dataset by keeping the rows which have missing values less than a particular value.
Here the maximum missing count is 6. So to clean up some of the data, we will remove the rows with 4 or more missing values.

Command: df = df[df["missing_count"] < 4]

```
[15] df = df[df["missing_count"] < 4]
```

After running these sets of commands, we run the command **df.describe()** once again. Using this, we can see that the number of rows dropped from 284142 to 192808. (~32.14%)

```
df.describe()
```

	YearStart	YearEnd	Data_Value	Data_Value_Alt	Low_Confidence_Limit	High_Confidence_Limit	LocationID	missing_count
count	192808.000000	192808.000000	192808.000000	192808.000000	192597.000000	192597.000000	192808.000000	192808.000000
mean	2018.595224	2018.655372	37.676757	37.676757	33.027824	42.595333	1145.457766	0.383573
std	2.301531	2.357772	25.213484	25.213484	24.290016	26.156408	2959.026567	0.787414
min	2015.000000	2015.000000	0.000000	0.000000	-0.700000	1.300000	1.000000	0.000000
25%	2017.000000	2017.000000	15.900000	15.900000	12.600000	19.700000	19.000000	0.000000
50%	2019.000000	2019.000000	32.800000	32.800000	27.000000	38.900000	35.000000	0.000000
75%	2021.000000	2021.000000	56.900000	56.900000	49.400000	64.600000	51.000000	0.000000
max	2022.000000	2022.000000	100.000000	100.000000	99.600000	100.000000	9004.000000	2.000000

Step 5 : Take care of missing data.
To take care of the missing data that has not been removed, one of the 2 methods can be used:
i) If the feature is of a **numeric data type**, we can use either **mean, median or mode** of the feature. If the data is **normally distributed**, use **mean**, if it is **skewed**, use **median**, and if many values are **repeated**, use **mode**.
ii) If the feature contains different categories, there are 2 ways. Either fill it with the mode of the column, or add a custom value such as “Data Unavailable”.

Here, we would be filling the missing data for columns of Data_Value, Low_Confidence_Limit and High_Confidence_Limit.

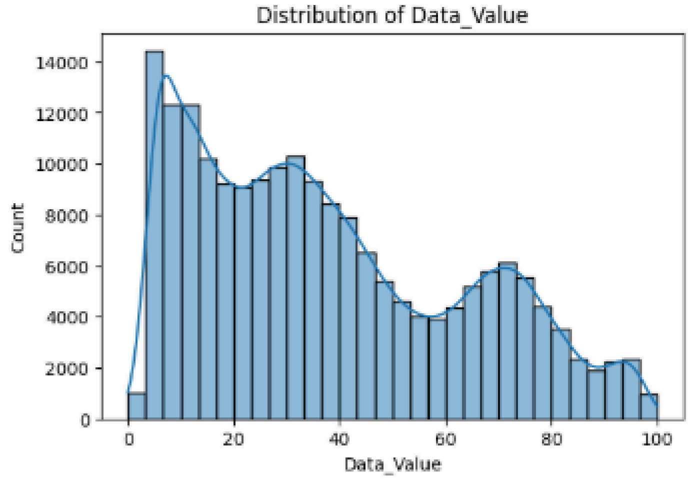
Follow the below steps to get how to fill the missing data:
i) **Check for skewness**
import seaborn as sns
import matplotlib.pyplot as plt

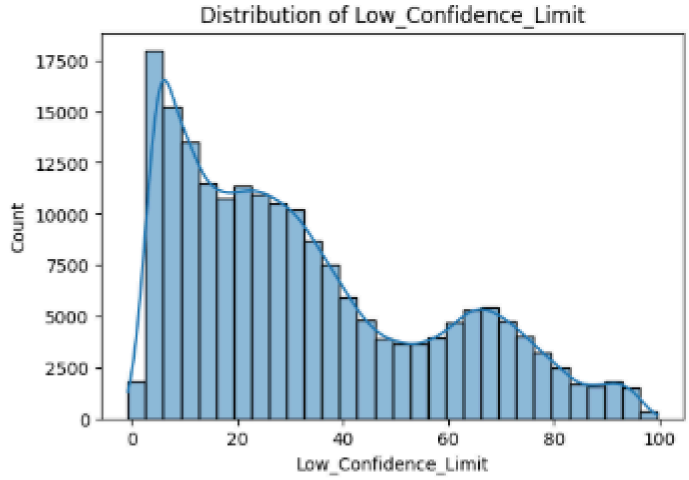
Name:Bhushan Mukund Kor **Academic Year:2024-2025**

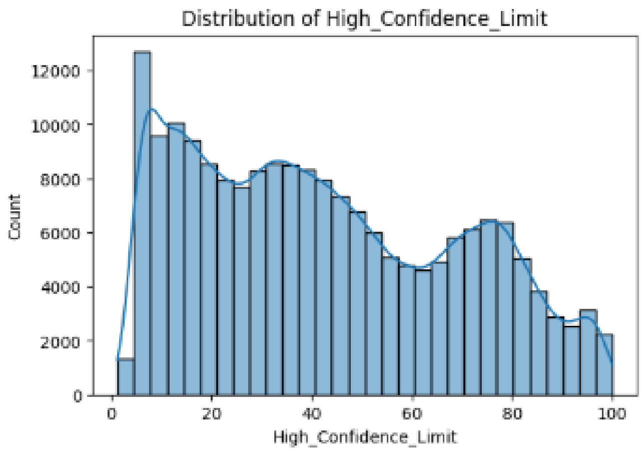
Division: D15C **Roll No: 28**

```
num_cols = ["Data_Value", "Low_Confidence_Limit", "High_Confidence_Limit"]

for col in num_cols:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(f"Distribution of {col}")
    plt.show()
```







As we can see here, there is a skewness to the left of the graph for each parameter, which means the data is not evenly distributed. Hence we use median.

Commands:
df[“Data_Value”].fillna(df[“Data_Value”].median(), inplace=True)
df[“Low_Confidence_Limit”].fillna(df[“Low_Confidence_Limit”].median(), inplace=True)
df[“High_Confidence_Limit”].fillna(df[“High_Confidence_Limit”].median(), inplace=True)

```
df["Data_Value"].fillna(df["Data_Value"].median(), inplace=True)

<ipython-input-21-49989ed8df82>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation in

  df["Data_Value"].fillna(df["Data_Value"].median(), inplace=True)
```

```
[22] df["Low_Confidence_Limit"].fillna(df["Low_Confidence_Limit"].median(), inplace=True)

<ipython-input-22-a155792c99bd>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation in

  df["Low_Confidence_Limit"].fillna(df["Low_Confidence_Limit"].median(), inplace=True)
```

```
[23] df["High_Confidence_Limit"].fillna(df["High_Confidence_Limit"].median(), inplace=True)

<ipython-input-23-e8d37b371347>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation in

  df["High_Confidence_Limit"].fillna(df["High_Confidence_Limit"].median(), inplace=True)
```

For columns **StratificationCategory2** and **Stratification2**, as sufficient data is not available, we would fill the missing values with a **placeholder “Data Unavailable”**
Commands:
df["StratificationCategory2"].fillna("Data Unavailable", inplace=True)
df["Stratification2"].fillna("Data Unavailable", inplace=True)

```
df["StratificationCategory2"].fillna("Data Unavailable", inplace=True)
df["Stratification2"].fillna("Data Unavailable", inplace=True)

<ipython-input-32-1f25356b9b45>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assi
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting v

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col]

  df["StratificationCategory2"].fillna("Data Unavailable", inplace=True)
<ipython-input-32-1f25356b9b45>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assi
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting v

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col]

  df["Stratification2"].fillna("Data Unavailable", inplace=True)
```

Now, we check the values by using the **df.head()** command.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

```
df.head(80)
```

	YearStart	YearEnd	LocationAbbr	Datasource	Class	Topic	Question	Data_Value_Unit	DataValueTypeID	Data_Value_Type	...	Stratification2	ClassID	TopicID
3	2022	2022	MD	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	PRCTG	Percentage	...	Black, non-Hispanic	C05	TMC01
4	2022	2022	WI	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	PRCTG	Percentage	...	Male	C05	TMC01
6	2022	2022	OK	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	PRCTG	Percentage	...	Native Am/Alaskan Native	C05	TMC01

Step 6 : Create dummy variables.

It is essential to create dummy variables for the columns that contain categorical data as most of the algorithms cannot understand the data directly. So they are classified as True and False or 0 and 1 which makes it easier.

To create the dummy variables, we will list the columns that fall under categorical columns and then **create another variable as pd_dummies** to get the output of this. Pandas library provides a inbuilt function called as get_dummies which takes the data from the columns and create all the required dummy variables

Command:
categorial_columns = ["LocationAbbr", "Question", "StratificationCategory1", "Stratification1"]
df_dummies = pd.get_dummies(df, columns=categorial_columns, drop_first=True)
To check output see the new columns added in the list of column.

```
[37] categorial_columns = ["LocationAbbr", "Question", "StratificationCategory1", "Stratification1"]
     df_dummies = pd.get_dummies(df, columns=categorial_columns, drop_first=True)
```

```
df_dummies.columns
```

```
Index(['YearStart', 'YearEnd', 'Datasource', 'Class', 'Topic',
       'Data_Value_Unit', 'DataValueTypeID', 'Data_Value_Type', 'Data_Value',
       'Data_Value_Alt',
       ...
       'Question_Percentage of older adults who reported that as a result of subjective cognitive decline or memory loss that they need assistance with day-to-day activities',
       'Question_Percentage of older adults who self-reported that their health is "fair" or "poor"',
       'Question_Percentage of older adults who self-reported that their health is "good", "very good", or "excellent"',
       'Question_Percentage of older adults with a lifetime diagnosis of depression',
       'Question_Percentage of older adults with subjective cognitive decline or memory loss who reported talking with a health care professional about it',
       'Question_Percentage of older adults without diabetes who reported a blood sugar or diabetes test within 3 years',
       'Question_Physically unhealthy days (mean number of days in past month)',
       'Question_Severe joint pain due to arthritis among older adults with doctor-diagnosed arthritis',
       'Stratification1_65 years or older', 'Stratification1_Overall'],
      dtype='object', length=112)
```

Step 7 : Find the outliers.

Outliers are those data values that vary vastly from the other dataset values. It is important to detect these values as they affect the analysis result.

There are 2 ways to find the outliers:

Method 1: Box-Plot
In this method, we use the column values to plot a box-plot graph. The values are usually in a box having lower and higher limits. If any outliers present, they come out of the box of the graph.

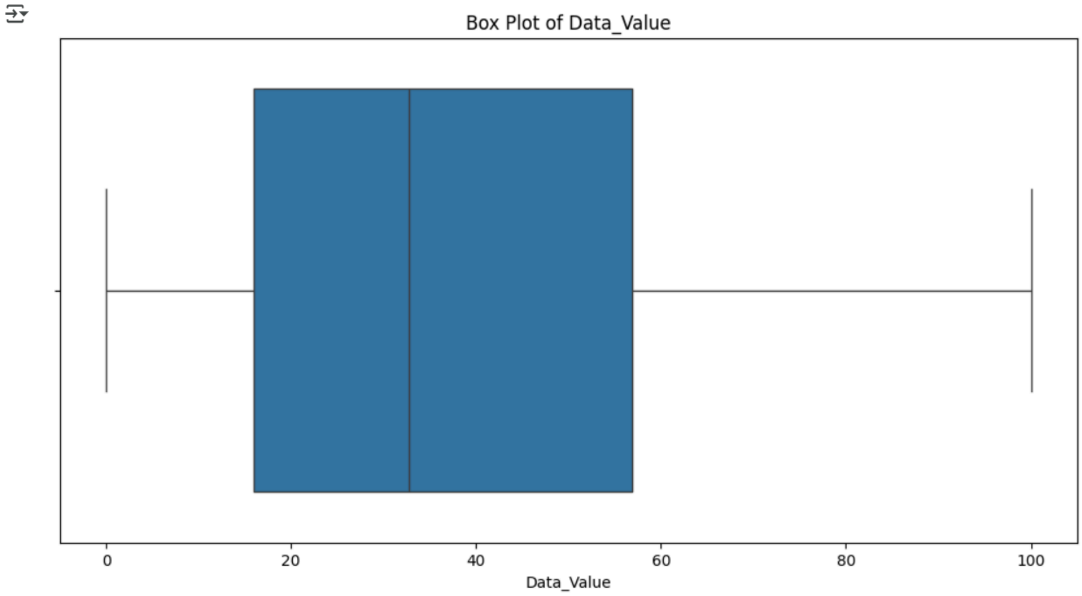
Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Command:

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))
sns.boxplot(x=df["Data_Value"])
plt.title("Box Plot of Data_Value")
plt.xlabel("Data_Value")
plt.show()
```



Method 2: Using IQR Value.

In this method, we find the IQR value fo the column; which is the difference between Q1 - 1.5 * IQR and Q3 + 1.5 * IQR. This is a standard that is followed, the factor 1.5 can be modified between 1 to 3 based on the requirement.

Command:

```
Q1 = df['Data_Value'].quantile(0.25)
Q3 = df['Data_Value'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['Data_Value'] < lower_bound) | (df['Data_Value'] > upper_bound)]
print("Number of Outliers in Data Value:", len(outliers))
print(outliers.head())
```

```
[46] Q1 = df['Data_Value'].quantile(0.25)
     Q3 = df['Data_Value'].quantile(0.75)
     IQR = Q3 - Q1
     lower_bound = Q1 - 1.5 * IQR
     upper_bound = Q3 + 1.5 * IQR
     outliers = df[(df['Data_Value'] < lower_bound) | (df['Data_Value'] > upper_bound)]
     print("Number of Outliers in Data Value:", len(outliers))
     print(outliers.head())

     Number of Outliers in Data Value: 0
     Empty DataFrame
     Columns: [YearStart, YearEnd, LocationAbbr, Datasource, Class, Topic, Question, Data_Value_Unit, DataValueTypeID, Data_Value_Type, Data_Value
     Index: []
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

From both the outputs, we get to know that there are some outliers present in the dataset. We can analyse the dataset manually to get the outliers, or use IQR score which gives us how many outliers are present based on our conditions

Step 8 : Standardization and Normalization of columns
We can standardize and normalize columns using 1 of 2 methods. Either by their formulae, or by the SKLearn Library.

Standardize Column:
Using formula:
mean_value = df["Data_Value"].mean()
std_value = df["Data_Value"].std()
df["Standardized_Data_Value"] = (df["Data_Value"] - mean_value) / std_value

Using Library:
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['Standardized Data Value Scalar'] = scaler.fit_transform(df[['Data_Value']])

```
[48] df["Standardized_Data_Value"] = (df["Data_Value"] - mean_value) / std_value
```

```
[49] from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
     df['Standardized Data Value Scalar'] = scaler.fit_transform(df[['Data_Value']])
```

```
df[["Data_Value", "Standardized_Data_Value", "Standardized Data Value Scalar"]].head()
```

	Data_Value	Standardized_Data_Value	Standardized Data Value Scalar
3	9.0	-1.137358	-1.137361
4	5.6	-1.272206	-1.272210
6	21.5	-0.641591	-0.641593
7	10.0	-1.097697	-1.097700
8	39.9	0.088177	0.088177

Normalize column:
Method 1: Formula
min_val = df['Data_Value'].min()
max_val = df['Data_Value'].max()

df['Data_Value_Normalized'] = (df['Data_Value'] - min_val) / (max_val - min_val)

Method 2: Scaler library
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['Normalized Data Value Scalar'] = scaler.fit_transform(df[['Data_Value']])

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

```
[51] min_val = df['Data_Value'].min()
     max_val = df['Data_Value'].max()

     df['Data_Value_Normalized'] = (df['Data_Value'] - min_val) / (max_val - min_val)
```

```
[52] from sklearn.preprocessing import MinMaxScaler
     scaler = MinMaxScaler()
     df['Normalized Data Value Scalar'] = scaler.fit_transform(df[['Data_Value']])
```

```
[53] df[['Data_Value', 'Data_Value_Normalized', 'Normalized Data Value Scalar']].head()
```

	Data_Value	Data_Value_Normalized	Normalized Data Value Scalar
3	9.0	0.090	0.090
4	5.6	0.056	0.056
6	21.5	0.215	0.215
7	10.0	0.100	0.100
8	39.9	0.399	0.399

Now to save the Changes in new file .

```
[54] df.to_csv("/content/drive/MyDrive/Semester 6/AIDS/AIDS Lab/Alzheimer_s_Disease_and_Healthy_Aging_Data_Updated.csv", index=False)
```

Conclusion:

We pre-processed the Alzheimer's disease and Healthy Aging dataset. To load the data, we used the pandas `read_csv()` function and checked the first five entries with the `head()` function.
For an overview of the data, we used methods like `head()`, `info()`, and `describe()` to get details about data types, mean, max, min, count, and other statistics.
Next, we dropped unnecessary columns from the dataset using the `drop()` function. These included columns like RowId, LocationDesc, Data_Value_Footnote_Symbol, Data_Value_Footnote, and Geolocation, as they wouldn't contribute much to the analysis.
To handle missing data, we identified and removed rows with the most missing values. This process reduced the dataset from 284,142 entries to 192,808 (~32.14%).
For the remaining missing values, we analyzed the data and used appropriate methods (mean, median, or mode) to fill them in.
Columns like Questions and LocationAbbr were causing issues during analysis, so we converted them into dummy variables (with 0s and 1s) to prevent errors.
To identify outliers, we created a boxplot, which helped us spot values outside the typical range. We then used the IQR method to further analyze and remove the outliers.
Finally, to avoid large values skewing the analysis, we normalized and standardized the data using min-max and standard deviation methods, bringing the values into a reasonable range for smoother analysis.`