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

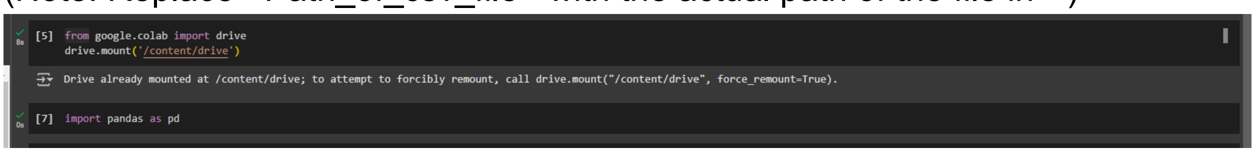
- Load data in Pandas.
- Description of the dataset.
- Drop columns that aren't useful.
- Drop rows with maximum missing values.
- Take care of missing data.
- Create dummy variables.
- Find out outliers (manually)
- standardization and normalization of columns

Steps:

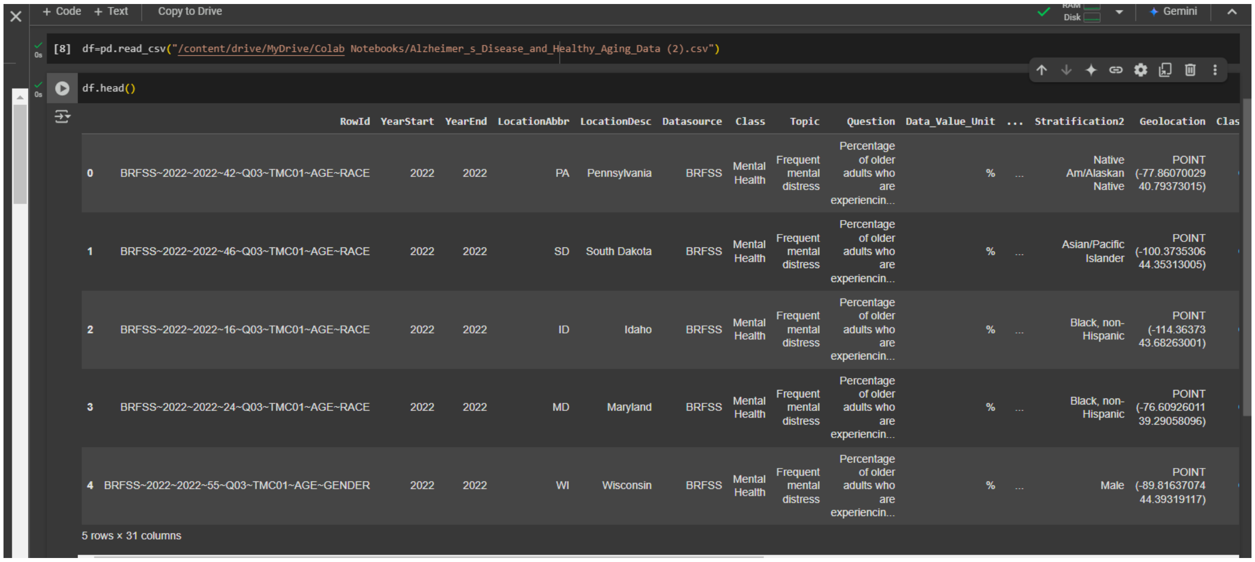
1) Load the file.

To load a file onto python for analysis, we need to make use of the pandas library. It gives us functionalities to read a CSV (Comma Separated Values) file and perform various functions on it.

Commands: **import pandas as pd** (Importing 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 “”)



To check whether the file is loaded or not, we will run the command **df.head().** This command gives the first 5 rows of the dataset as the output.



```
[8] df=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Alzheimer_s_Disease_and_Healthy_Aging_Data (2).csv")
df.head()
```

	RowId	YearStart	YearEnd	LocationAbbr	LocationDesc	Datasource	Class	Topic	Question	Data_Value_Unit	...	Stratification2	Geolocation	Clas
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)	
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)	
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)	
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)	
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)	

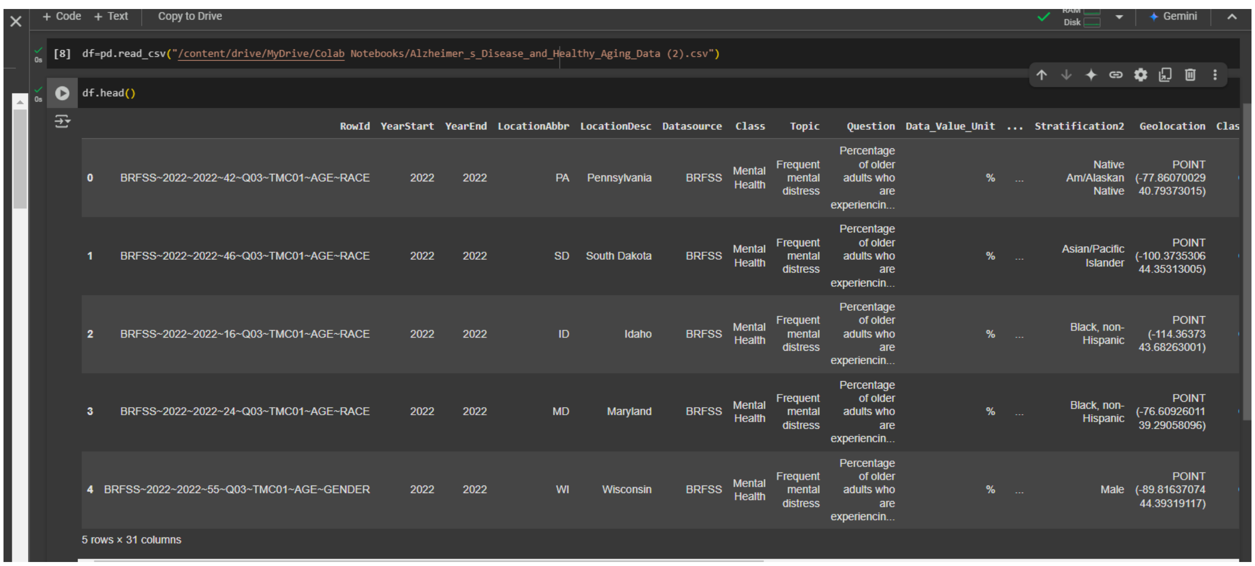
5 rows × 31 columns

2) Description of the dataset

The description of the dataset gives the user an idea on what are the features, what is the count of rows and columns, etc. To achieve this, we can use the following commands.

Command 1: df.head()

As mentioned before, **head** function give us the first 5 rows of the dataset. This allows for the user to get an overview on what values are being listed in the dataset.




```
[8] df=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Alzheimer_s_Disease_and_Healthy_Aging_Data (2).csv")
df.head()
```

	RowId	YearStart	YearEnd	LocationAbbr	LocationDesc	Datasource	Class	Topic	Question	Data_Value_Unit	...	Stratification2	Geolocation	Clas
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)	
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)	
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)	
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)	
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)	

5 rows × 31 columns

Command 2: df.info()

This command gives all the information about the features (columns) of the dataset and the data type of each of these columns. It also gives a summary of all the values in the dataset.

+ Code + Text | Copy to Drive

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 46299 entries, 0 to 46298
Data columns (total 31 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   RowId                       46299 non-null  object
 1   YearStart                   46299 non-null  int64
 2   YearEnd                     46299 non-null  int64
 3   LocationAbbr                46299 non-null  object
 4   LocationDesc                46299 non-null  object
 5   Datasource                  46299 non-null  object
 6   Class                       46299 non-null  object
 7   Topic                       46298 non-null  object
 8   Question                    46298 non-null  object
 9   Data_Value_Unit             46298 non-null  object
 10  DataValueTypeID             46298 non-null  object
 11  Data_Value_Type             46298 non-null  object
 12  Data_Value                  31337 non-null  float64
 13  Data_Value_Alt              31337 non-null  float64
 14  Data_Value_Footnote_Symbol  17102 non-null  object
 15  Data_Value_Footnote         17102 non-null  object
 16  Low_Confidence_Limit        31302 non-null  float64
 17  High_Confidence_Limit       31302 non-null  float64
 18  StratificationCategory1     46298 non-null  object
 19  Stratification1             46298 non-null  object
 20  StratificationCategory2     40126 non-null  object
 21  Stratification2             40126 non-null  object
 22  Geolocation                 41801 non-null  object
 23  ClassID                     46298 non-null  object
 24  TopicID                     46298 non-null  object
 25  QuestionID                  46298 non-null  object
 26  LocationID                  46298 non-null  float64
 27  StratificationCategoryID1   46298 non-null  object
 28  StratificationID1           46298 non-null  object
 29  StratificationCategoryID2   46298 non-null  object
 30  StratificationID2           46298 non-null  object
dtypes: float64(5), int64(2), object(24)
e: 11.0+ MB
```

Command 3: df.describe()

This command gives the details of all the values under all the features of the dataset. The command having no parameters gives information about count, max, min, standard deviation, top 25%ile, 50%ile, 75%ile and max value of the dataset.

```
df.describe()
```

	YearStart	YearEnd	Data_Value	Data_Value_Alt	Low_Confidence_Limit	High_Confidence_Limit	LocationID
count	46299.000000	46299.000000	31337.000000	31337.000000	31302.000000	31302.000000	46298.000000
mean	2021.592821	2021.913367	39.260829	39.260829	34.093272	44.694559	730.058447
std	0.819803	0.300529	24.203755	24.203755	23.378049	24.985639	2404.300067
min	2015.000000	2015.000000	0.000000	0.000000	0.000000	1.300000	1.000000
25%	2021.000000	2022.000000	19.100000	19.100000	14.825000	23.500000	18.000000
50%	2022.000000	2022.000000	34.400000	34.400000	27.900000	41.500000	33.000000
75%	2022.000000	2022.000000	58.900000	58.900000	51.000000	66.400000	49.000000
max	2022.000000	2022.000000	100.000000	100.000000	97.800000	100.000000	9004.000000

Open settings

0s completed at 10:28 PM

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.

```
df.describe(include="all")
```

	RowId	YearStart	YearEnd	LocationAbbr	LocationDesc	Datasource	Class	Topic	Question	Data_Value_Unit	...	Stratification2	Geoloca
count	46299	46299.000000	46299.000000	46299	46299	46299	46299	46298	46298	46298	...	40126	4
unique	6769	NaN	NaN	59	59	1	8	35	35	2	...	7	
top	BRFSS~2022~2022~42~Q03~TMC01~AGE~RACE	NaN	NaN	VA	Virginia	BRFSS	Overall Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Female	P (-78.4578 37.5426
freq	15	NaN	NaN	915	915	46299	12897	1941	1941	44315	...	6031	
mean	NaN	2021.592821	2021.913367	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	
std	NaN	0.819803	0.300529	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	
min	NaN	2015.000000	2015.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	
25%	NaN	2021.000000	2022.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	
50%	NaN	2022.000000	2022.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	
75%	NaN	2022.000000	2022.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	
max	NaN	2022.000000	2022.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	

11 rows × 31 columns

3) Drop columns that are not useful

In data science, it is important to drop the columns that would not help the user while working on the dataset as it would make it cleaner to work with.

Here, we will first check the columns that are present using **df.columns** command

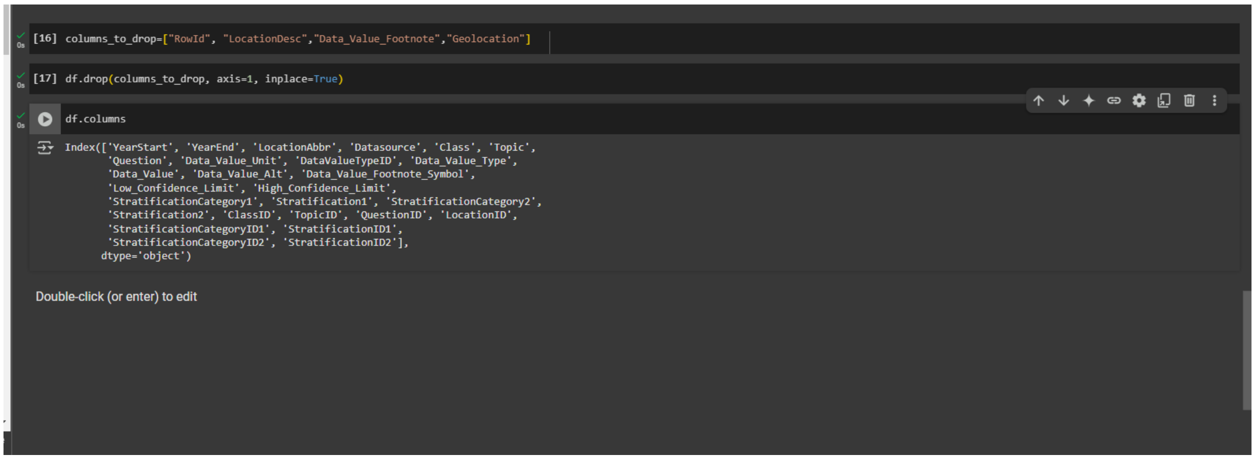
```
df.columns
```

```
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')
```

Now, we will list down the columns that are to be dropped and then pass it on to the command

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 columns.



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

4) Drop rows with maximum missing columns

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. To perform this, follow these steps:

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

```
dtype='object')
```

```
df.describe()
```

	YearStart	YearEnd	Data_Value	Data_Value_Alt	Low_Confidence_Limit	High_Confidence_Limit	LocationID
count	46299.000000	46299.000000	31337.000000	31337.000000	31302.000000	31302.000000	46298.000000
mean	2021.592821	2021.913367	39.260829	39.260829	34.093272	44.694559	730.058447
std	0.819803	0.300529	24.203755	24.203755	23.378049	24.985639	2404.300067
min	2015.000000	2015.000000	0.000000	0.000000	0.000000	1.300000	1.000000
25%	2021.000000	2022.000000	19.100000	19.100000	14.825000	23.500000	18.000000
50%	2022.000000	2022.000000	34.400000	34.400000	27.900000	41.500000	33.000000
75%	2022.000000	2022.000000	58.900000	58.900000	51.000000	66.400000	49.000000
max	2022.000000	2022.000000	100.000000	100.000000	97.800000	100.000000	9004.000000

Double-click (or enter) to edit

df["missing_count"] = df.isnull().sum(axis=1)
max_missing = df["missing_count"].max()
df = df[df["missing_count"] != max_missing]

The above set of commands do the following function:
i) Create a column called missing_count where the sum of all the cells having null values is stored.
ii) The maximum value from this missing_count column is considered for deletion
iii) Finally, we update the dataset by keeping the rows which have missing values less than the maximum value.

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 284036.

```
df.describe()
```

	YearStart	YearEnd	Data_Value	Data_Value_Alt	Low_Confidence_Limit	High_Confidence_Limit	LocationID	missing_count
count	284036.000000	284036.000000	192808.000000	192808.000000	192597.000000	192597.000000	284036.000000	284036.000000
mean	2018.595625	2018.657191	37.676757	37.676757	33.027824	42.595333	800.600941	1.545114
std	2.302762	2.359986	25.213484	25.213484	24.290016	26.156408	2511.992210	1.808961
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	34.000000	0.000000
75%	2021.000000	2021.000000	56.900000	56.900000	49.400000	64.600000	49.000000	4.000000
max	2022.000000	2022.000000	100.000000	100.000000	99.600000	100.000000	9004.000000	4.000000

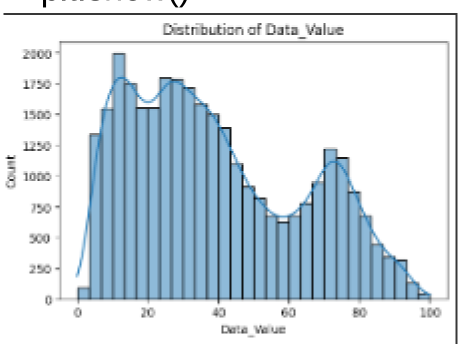
5) Take care of missing data

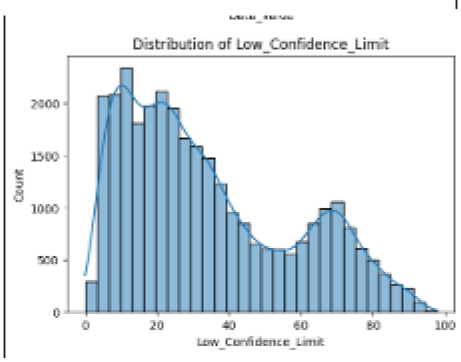
To take care of the missing data that has not been removed, one of the 2 methods can be used:

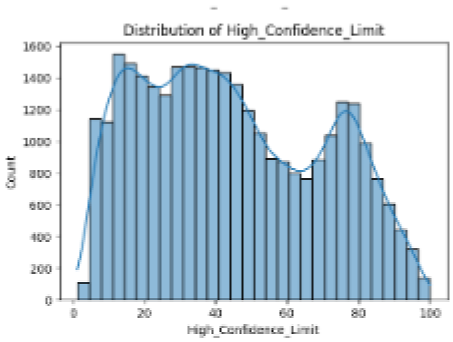
→ If the feature is of a numeric data type, we can use either mean, median or mode of the feature. If the data is normally distributed, use mean, if it is skewed, use median, and if many values are repeated, use mode.
→ If the feature contains different categories, there are 2 ways. Either fill it with 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
To check how to fill the missing data, follow the steps below.

i) **Check for skewness**

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

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)

For columns StratificationCategory2 and Stratification2, as sufficient data is not available, we would fill the missing values with a placeholder “Data Unavailable”

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

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

enings and Vaccines	Pap test within past 3 years	of older adult women with an intact...	%	PRCTG	Percentage	29.6	29.6	16.3	47.7	Age Group	Overall	Race/Ethnicity	Native Am/Alaskan Native
on/Physical vity/Obesity	Obesity	Percentage of older adults who are currently o...	%	PRCTG	Percentage	32.8	32.8	27.0	38.9	Age Group	50-64 years	Race/Ethnicity	Asian/Pacific Islander
on/Physical vity/Obesity	Obesity	Percentage of older adults who are currently o...	%	PRCTG	Percentage	39.1	39.1	35.4	42.9	Age Group	50-64 years	Gender	Male
on/Physical vity/Obesity	Obesity	Percentage of older adults who are currently o...	%	PRCTG	Percentage	42.4	42.4	34.2	51.1	Age Group	65 years or older	Race/Ethnicity	Hispanic
on/Physical vity/Obesity	Obesity	Percentage of older adults who are currently o...	%	PRCTG	Percentage	32.8	32.8	27.0	38.9	Age Group	65 years or older	Race/Ethnicity	Native Am/Alaskan Native
on/Physical vity/Obesity	Obesity	Percentage of older adults who are currently o...	%	PRCTG	Percentage	19.5	19.5	16.7	22.6	Age Group	Overall	Race/Ethnicity	Asian/Pacific Islander
on/Physical vity/Obesity	Obesity	Percentage of older adults who are currently o...	%	PRCTG	Percentage	48.7	48.7	41.4	56.0	Age Group	Overall	Race/Ethnicity	Black, non-Hispanic
enings and Vaccines	Colorectal cancer screening	Percentage of older adults who had either a ho...	%	PRCTG	Percentage	67.8	67.8	65.2	70.4	Age Group	50-64 years	Data Unavailable	Data Unavailable
	No leisure-	Percentage											

6) Create Dummy Variables

It is essential to create dummy variables to 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
categorial_columns = ["LocationAbbr", "Question", "StratificationCategory1", "Stratification1"]
df_dummies = pd.get_dummies(df, columns=categorial_columns, drop_first=True)

df_dummies.columns after dummy variables created

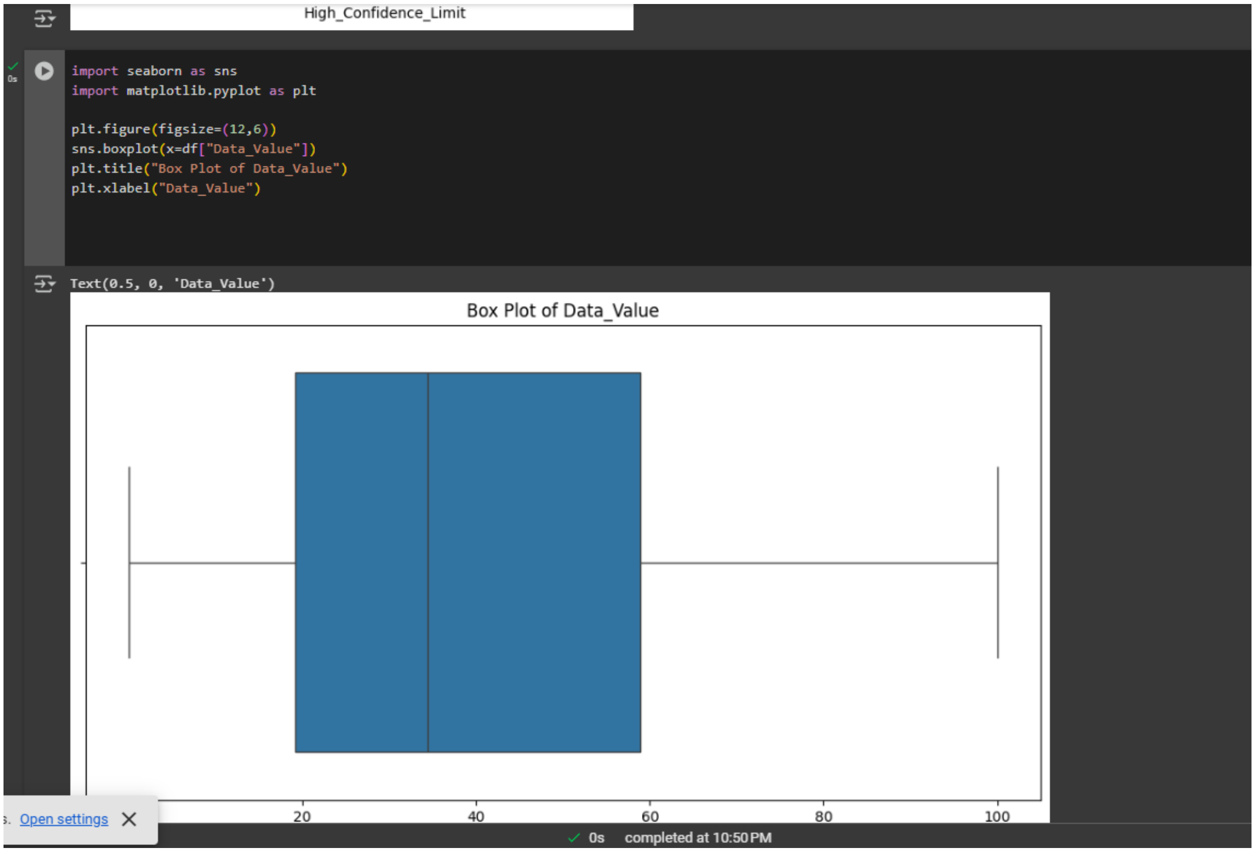
```
df_dummies.columns

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)
```

Find out outliers:

Method 1: Box-Plot
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 to find the number of outliers and what are the outliers.

```
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())
```

```python
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: 29810
    YearStart  YearEnd LocationAbbr Datasource                    Class  \
10       2022     2022           MD      BRFSS           Overall Health
11       2022     2022           NY      BRFSS           Overall Health
33       2022     2022           WA      BRFSS  Screenings and Vaccines
34       2022     2022           HI      BRFSS  Screenings and Vaccines
35       2022     2022           HI      BRFSS  Screenings and Vaccines

                        Topic  \
10  Oral health:  tooth retention
11  Oral health:  tooth retention
33  Mammogram within past 2 years
34  Mammogram within past 2 years
35  Mammogram within past 2 years

                                            Question Data_Value_Unit  \
10  Percentage of older adults who report having l...               %
11  Percentage of older adults who report having l...               %
33  Percentage of older adult women who have recei...               %
34  Percentage of older adult women who have recei...               %
35  Percentage of older adult women who have recei...               %

   DataValueTypeID Data_Value_Type  Data_Value  Data_Value_Alt  \
10           PRCTG      Percentage        71.5            71.5
11           PRCTG      Percentage        73.9            73.9
33           PRCTG      Percentage        73.2            73.2
34           PRCTG      Percentage        75.0            75.0
35           PRCTG      Percentage        84.4            84.4

    Low_Confidence_Limit  High_Confidence_Limit StratificationCategory1  \
10                  50.0                   86.3               Age Group
11                  60.4                   84.0               Age Group
33                  71.0                   75.3               Age Group
34                  67.9                   80.9               Age Group
35                  66.1                   93.7               Age Group

      Stratification1 StratificationCategory2          Stratification2
10  65 years or older          Race/Ethnicity  Asian/Pacific Islander
11  65 years or older          Race/Ethnicity  Asian/Pacific Islander
```

Standardization of columns:

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']])

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

	Data_Value	Standardized_Data_Value	Standardized Data Value Scalar
0	32.8	-0.158409	-0.158410
1	32.8	-0.158409	-0.158410
2	32.8	-0.158409	-0.158410
3	9.0	-1.297282	-1.297284
4	5.6	-1.459978	-1.459980

Normalization of columns:

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']])

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

	Data_Value	Data_Value_Normalized	Normalized Data Value Scalar
0	32.8	0.328	0.328
1	32.8	0.328	0.328
2	32.8	0.328	0.328
3	9.0	0.090	0.090
4	5.6	0.056	0.056