Data Reading

In [1]:
```
# importing necessary libraries
import pandas as pd
import numpy as np

clinical_dataset = pd.read_excel("C:/Users/James/OneDrive/Desktop/wetransfer_c
clinical_dataset
```

Out[1]:

	subject_id	age	sex	weight	height	trt_grp	RESPONSE
0	SUBJ_001	46.000	Female	84.660	1.59000	DRUG	N
1	SUBJ_001	46.000	Female	84.660	1.59000	DRUG	N
2	SUBJ_002	47.000	Female	71.210	1.64000	DRUG	Y
3	SUBJ_003	48.000	Female	69.850	1.73000	CONTROL	N
4	SUBJ_004	59.000	Female	62.940	1.50000	DRUG	Y
...	...	...	...	...	...	...	...
767	SUBJ_767	53.000	Male	88.670	1.72000	DRUG	Y
768	SUBJ_768	68.000	Female	80.290	1.63000	DRUG	Y
769	SUBJ_A69	7.200	Female	22.310	1.19300	DRUG	N
770	SUBJ_A70	8.310	Female	24.220	1.27440	CONTROL	N
771	SUBJ_A71	7.854	Male	23.176	1.26343	CONTROL	N

772 rows × 7 columns

In [2]:
```
protein_dataset = pd.read_excel("C:/Users/James/OneDrive/Desktop/wetransfer_cl
protein_dataset
```

Out[2]:

	participant_id	protein_concentration
0	SUBJ_001	148.0
1	SUBJ_002	85.0
2	SUBJ_003	183.0
3	SUBJ_004	89.0
4	SUBJ_005	137.0
...	...	...
763	SUBJ_764	101.0
764	SUBJ_765	122.0
765	SUBJ_766	121.0
766	SUBJ_767	126.0
767	SUBJ_768	93.0

768 rows × 2 columns

Data Cleaning

In [3]:
```
#checking information of the dataset
clinical_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 772 entries, 0 to 771
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   subject_id  772 non-null    object 
 1   age         772 non-null    float64
 2   sex         772 non-null    object 
 3   weight      761 non-null    float64
 4   height      772 non-null    float64
 5   trt_grp     772 non-null    object 
 6   RESPONSE    772 non-null    object 
dtypes: float64(3), object(4)
memory usage: 42.3+ KB
```

In [4]:
```
#checking information of the age column
clinical_dataset["age"].unique()
```

Out[4]:
```
array([46.   , 47.   , 48.   , 59.   , 63.   , 77.   , 57.   , 72.   ,
       73.   , 67.   , 53.   , 58.   , 55.   , 65.   , 54.   , 51.   ,
       61.   , 56.   , 64.   , 66.   , 69.   , 70.   , 49.   , 62.   ,
       68.   , 71.   , 60.   , 52.   , 78.   , 79.   , 43.   , 44.   ,
       76.   , 39.   , 74.   , 50.   , 45.   , 75.   , 37.   ,  7.2  ,
        8.31 ,  7.854])
```

In [5]:
```
#checking information of the age column
clinical_dataset["age"].describe()
```

Out[5]:
```
count    772.000000
mean      61.580782
std        7.866491
min        7.200000
25%       57.000000
50%       62.000000
75%       67.000000
max       79.000000
Name: age, dtype: float64
```

In [6]:
```
# Looking up a row in the age column
clinical_dataset[(clinical_dataset["age"] == 7.2)]
```

Out[6]:

	subject_id	age	sex	weight	height	trt_grp	RESPONSE
769	SUBJ_A69	7.2	Female	22.31	1.193	DRUG	N

In [7]:
```
# deleting a row in the age column
clinical_dataset = clinical_dataset.drop(769)
```

In [8]:
```
#checking information of the dataset
clinical_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 771 entries, 0 to 771
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   subject_id  771 non-null    object 
 1   age         771 non-null    float64
 2   sex         771 non-null    object 
 3   weight      760 non-null    float64
 4   height      771 non-null    float64
 5   trt_grp     771 non-null    object 
 6   RESPONSE    771 non-null    object 
dtypes: float64(3), object(4)
memory usage: 48.2+ KB
```

In [9]:
```
# looking up a row in the age column
clinical_dataset[(clinical_dataset["age"] == 8.31)]
```

Out[9]:

	subject_id	age	sex	weight	height	trt_grp	RESPONSE
770	SUBJ_A70	8.31	Female	24.22	1.2744	CONTROL	N

In [10]:
```
# deleting a row in the age column
clinical_dataset = clinical_dataset.drop(770)
```

In [11]:
```
clinical_dataset[(clinical_dataset["age"] == 7.854)]
```

Out[11]:

	subject_id	age	sex	weight	height	trt_grp	RESPONSE
771	SUBJ_A71	7.854	Male	23.176	1.26343	CONTROL	N

In [12]:
```
# deleting a row in the age column
clinical_dataset = clinical_dataset.drop(771)
```

In [13]:
```
#checking information of the age column
clinical_dataset["age"].unique()
```

Out[13]:
```
array([46., 47., 48., 59., 63., 77., 57., 72., 73., 67., 53., 58., 55.,
       65., 54., 51., 61., 56., 64., 66., 69., 70., 49., 62., 68., 71.,
       60., 52., 78., 79., 43., 44., 76., 39., 74., 50., 45., 75., 37.])
```

In [14]:
```
#changing the data type of the column
clinical_dataset["age"] = clinical_dataset["age"].astype(int)
```

In [15]:
```
#checking information of the age column
clinical_dataset["age"].unique()
```

Out[15]:
```
array([46, 47, 48, 59, 63, 77, 57, 72, 73, 67, 53, 58, 55, 65, 54, 51, 61,
       56, 64, 66, 69, 70, 49, 62, 68, 71, 60, 52, 78, 79, 43, 44, 76, 39,
       74, 50, 45, 75, 37])
```

In [16]:
```
#checking for null values
clinical_dataset.isna().sum()
```

Out[16]:
```
subject_id     0
age            0
sex            0
weight        11
height         0
trt_grp        0
RESPONSE       0
dtype: int64
```

In [17]:
```
#Displaying Nan Values
clinical_dataset[clinical_dataset["weight"].isna()]
```

Out[17]:

	subject_id	age	sex	weight	height	trt_grp	RESPONSE
10	SUBJ_010	73	Female	NaN	1.64	DRUG	Y
50	SUBJ_050	65	Male	NaN	1.71	DRUG	N
61	SUBJ_061	61	Male	NaN	1.79	CONTROL	N
82	SUBJ_082	65	Male	NaN	1.78	CONTROL	N
146	SUBJ_146	56	Female	NaN	1.57	CONTROL	N
372	SUBJ_372	67	Female	NaN	1.62	CONTROL	N
427	SUBJ_427	66	Male	NaN	1.83	CONTROL	N
495	SUBJ_495	60	Female	NaN	1.65	CONTROL	N
523	SUBJ_523	62	Female	NaN	1.60	DRUG	N
685	SUBJ_685	74	Male	NaN	1.81	CONTROL	N
707	SUBJ_707	72	Male	NaN	1.75	DRUG	Y

In [18]:
```
# Displaying the group of patients aged 73
clinical_dataset[(clinical_dataset["age"] == 73) & (clinical_dataset["sex"] ==
```

Out[18]:

	subject_id	age	sex	weight	height	trt_grp	RESPONSE
10	SUBJ_010	73	Female	NaN	1.64	DRUG	Y
20	SUBJ_020	73	Female	94.38	1.65	DRUG	Y
118	SUBJ_118	73	Female	86.98	1.61	CONTROL	N
258	SUBJ_258	73	Female	68.56	1.55	DRUG	Y
265	SUBJ_265	73	Female	80.99	1.59	DRUG	N

In [19]:
```
# Displaying the group of patients aged 73
clinical_dataset[(clinical_dataset["age"] == 73) & (clinical_dataset["sex"] ==
```

```
C:\Users\James\AppData\Local\Temp\ipykernel_9712\3400160153.py:2: FutureWarni
ng: The default value of numeric_only in DataFrame.mean is deprecated. In a f
uture version, it will default to False. In addition, specifying 'numeric_onl
y=None' is deprecated. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  clinical_dataset[(clinical_dataset["age"] == 73) & (clinical_dataset["sex"]
== "Female")].mean()
```

Out[19]:
```
age       73.0000
weight    82.7275
height     1.6080
dtype: float64
```

In [20]:
```
#Replacing the Nan values individually
clinical_dataset.at[10,"weight"] =  83.863
```

I used the mean of the groups of female patients aged 73 to replace the nan value and doing the same for the remainig nan values

In [21]:
```
# Displaying the group of patients aged 65
clinical_dataset[(clinical_dataset["age"] == 65) & (clinical_dataset["sex"] ==
```

```
C:\Users\James\AppData\Local\Temp\ipykernel_9712\3343773379.py:2: FutureWarni
ng: The default value of numeric_only in DataFrame.mean is deprecated. In a f
uture version, it will default to False. In addition, specifying 'numeric_onl
y=None' is deprecated. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  clinical_dataset[(clinical_dataset["age"] == 65) & (clinical_dataset["sex"]
== "Male")].mean()
```

Out[21]:
```
age        65.000000
weight     99.276471
height      1.744211
dtype: float64
```

In [22]:
```
#Replacing the Nan values individually
clinical_dataset.at[50,"weight"] = 99.276471
```

In [23]:
```
# Displaying the group of patients aged 61
clinical_dataset[(clinical_dataset["age"] == 61) & (clinical_dataset["sex"] ==
```

```
C:\Users\James\AppData\Local\Temp\ipykernel_9712\1356158029.py:2: FutureWarni
ng: The default value of numeric_only in DataFrame.mean is deprecated. In a f
uture version, it will default to False. In addition, specifying 'numeric_onl
y=None' is deprecated. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  clinical_dataset[(clinical_dataset["age"] == 61) & (clinical_dataset["sex"]
== "Male")].mean()
```

Out[23]:
```
age         61.00000
weight     103.72875
height       1.75200
dtype: float64
```

In [24]:
```
#Replacing the Nan values individually
clinical_dataset.at[61,"weight"] = 103.72875
```

In [25]:
```
# Displaying the group of patients aged 65
clinical_dataset[(clinical_dataset["age"] == 65) & (clinical_dataset["sex"] ==
```

```
C:\Users\James\AppData\Local\Temp\ipykernel_9712\3343773379.py:2: FutureWarni
ng: The default value of numeric_only in DataFrame.mean is deprecated. In a f
uture version, it will default to False. In addition, specifying 'numeric_onl
y=None' is deprecated. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  clinical_dataset[(clinical_dataset["age"] == 65) & (clinical_dataset["sex"]
== "Male")].mean()
```

Out[25]:
```
age       65.000000
weight    99.276471
height     1.744211
dtype: float64
```

In [26]:
```
#Replacing the Nan values individually
clinical_dataset.at[82,"weight"] = 99.276471
```

In [27]:
```
# Displaying the group of patients aged 56
clinical_dataset[(clinical_dataset["age"] == 56) & (clinical_dataset["sex"] ==
```

```
C:\Users\James\AppData\Local\Temp\ipykernel_9712\1152700976.py:2: FutureWarni
ng: The default value of numeric_only in DataFrame.mean is deprecated. In a f
uture version, it will default to False. In addition, specifying 'numeric_onl
y=None' is deprecated. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  clinical_dataset[(clinical_dataset["age"] == 56) & (clinical_dataset["sex"]
== "Female")].mean()
```

Out[27]:
```
age       56.000000
weight    88.405000
height     1.602174
dtype: float64
```

In [28]:
```
#Replacing the Nan values individually
clinical_dataset.at[146,"weight"] = 88.405000
```

In [29]:
```
# Displaying the group of patients aged 67
clinical_dataset[(clinical_dataset["age"] == 67) & (clinical_dataset["sex"] ==
```

```
C:\Users\James\AppData\Local\Temp\ipykernel_9712\3718369951.py:2: FutureWarni
ng: The default value of numeric_only in DataFrame.mean is deprecated. In a f
uture version, it will default to False. In addition, specifying 'numeric_onl
y=None' is deprecated. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  clinical_dataset[(clinical_dataset["age"] == 67) & (clinical_dataset["sex"]
== "Female")].mean()
```

Out[29]:
```
age       67.000000
weight    83.613333
height     1.612400
dtype: float64
```

In [30]:
```
#Replacing the Nan values individually
clinical_dataset.at[372,"weight"] = 83.613333
```

In [31]:
```
# Displaying the group of patients aged 66
clinical_dataset[(clinical_dataset["age"] == 66) & (clinical_dataset["sex"] ==
```

```
C:\Users\James\AppData\Local\Temp\ipykernel_9712\2347124795.py:2: FutureWarni
ng: The default value of numeric_only in DataFrame.mean is deprecated. In a f
uture version, it will default to False. In addition, specifying 'numeric_onl
y=None' is deprecated. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  clinical_dataset[(clinical_dataset["age"] == 66) & (clinical_dataset["sex"]
== "Male")].mean()
```

Out[31]:
```
age       66.000000
weight    99.812222
height     1.784211
dtype: float64
```

In [32]:
```
#Replacing the Nan values individually
clinical_dataset.at[427,"weight"] = 99.812222
```

In [33]:
```
# Displaying the group of patients aged 60
clinical_dataset[(clinical_dataset["age"] == 60) & (clinical_dataset["sex"] ==
```

```
C:\Users\James\AppData\Local\Temp\ipykernel_9712\3051124930.py:2: FutureWarni
ng: The default value of numeric_only in DataFrame.mean is deprecated. In a f
uture version, it will default to False. In addition, specifying 'numeric_onl
y=None' is deprecated. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  clinical_dataset[(clinical_dataset["age"] == 60) & (clinical_dataset["sex"]
== "Female")].mean()
```

Out[33]:
```
age       60.000000
weight    89.025909
height     1.620870
dtype: float64
```

In [34]:
```
#Replacing the Nan values individually
clinical_dataset.at[495,"weight"] = 89.025909
```

In [35]:
```
# Displaying the group of patients aged 62
clinical_dataset[(clinical_dataset["age"] == 62) & (clinical_dataset["sex"] ==
```

```
C:\Users\James\AppData\Local\Temp\ipykernel_9712\3223133000.py:2: FutureWarni
ng: The default value of numeric_only in DataFrame.mean is deprecated. In a f
uture version, it will default to False. In addition, specifying 'numeric_onl
y=None' is deprecated. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  clinical_dataset[(clinical_dataset["age"] == 62) & (clinical_dataset["sex"]
== "Female")].mean()
```

Out[35]:
```
age       62.00000
weight    82.33800
height     1.59125
dtype: float64
```

In [36]:
```
#Replacing the Nan values individually
clinical_dataset.at[523,"weight"] = 82.33800
```

In [37]:
```
# Displaying the group of patients aged 74
clinical_dataset[(clinical_dataset["age"] == 74) & (clinical_dataset["sex"] ==
```

```
C:\Users\James\AppData\Local\Temp\ipykernel_9712\4168779951.py:2: FutureWarni
ng: The default value of numeric_only in DataFrame.mean is deprecated. In a f
uture version, it will default to False. In addition, specifying 'numeric_onl
y=None' is deprecated. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  clinical_dataset[(clinical_dataset["age"] == 74) & (clinical_dataset["sex"]
== "Male")].mean()
```

Out[37]:
```
age       74.0000
weight    86.6275
height     1.7580
dtype: float64
```

In [38]:
```
#Replacing the Nan values individually
clinical_dataset.at[685,"weight"] = 86.6275
```

In [39]:
```
# Displaying the group of patients aged 72
clinical_dataset[(clinical_dataset["age"] == 72) & (clinical_dataset["sex"] ==
```

```
C:\Users\James\AppData\Local\Temp\ipykernel_9712\1537195054.py:2: FutureWarni
ng: The default value of numeric_only in DataFrame.mean is deprecated. In a f
uture version, it will default to False. In addition, specifying 'numeric_onl
y=None' is deprecated. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  clinical_dataset[(clinical_dataset["age"] == 72) & (clinical_dataset["sex"]
== "Male")].mean()
```

Out[39]:
```
age       72.000000
weight    96.942500
height     1.753333
dtype: float64
```

In [40]:
```
#Replacing the Nan values individually
clinical_dataset.at[707,"weight"] =  96.942500
```

In [41]:
```
#checking for duplicated entries
clinical_dataset.duplicated().sum()
```

Out[41]:
```
1
```

In [42]:
```
clinical_dataset = clinical_dataset.drop_duplicates()
```

In [43]:
```
#checking for duplicated entries
clinical_dataset.duplicated().sum()
```

Out[43]:
```
0
```

In [44]:
```
#checking for null values
clinical_dataset.isna().sum()
```

Out[44]:
```
subject_id    0
age           0
sex           0
weight        0
height        0
trt_grp       0
RESPONSE      0
dtype: int64
```

In [45]:
```
#Getting dataframe information
clinical_dataset.describe()
```

Out[45]:

	age	weight	height
count	768.000000	768.000000	768.000000
mean	61.811198	91.397538	1.679271
std	7.107447	21.995748	0.097888
min	37.000000	46.170000	1.420000
25%	57.000000	75.932500	1.600000
50%	62.000000	88.985000	1.680000
75%	67.000000	104.400000	1.760000
max	79.000000	182.500000	1.940000

In [46]:
```
#checking for null values
protein_dataset.isna().sum()
```

Out[46]:
```
participant_id           0
protein_concentration    5
dtype: int64
```

In [47]:
```
#Displaying Nan Values
protein_dataset[protein_dataset["protein_concentration"].isna()]
```

Out[47]:

	participant_id	protein_concentration
75	SUBJ_076	NaN
182	SUBJ_183	NaN
342	SUBJ_343	NaN
349	SUBJ_350	NaN
502	SUBJ_503	NaN

In [48]:
```
#Getting dataframe information
protein_dataset.describe()
```

Out[48]:

	protein_concentration
count	763.000000
mean	121.686763
std	30.535641
min	44.000000
25%	99.000000
50%	117.000000
75%	141.000000
max	199.000000

In [49]:
```
# replacing nan values with the mean
protein_dataset["protein_concentration"].fillna(protein_dataset["protein_conce
```

In [50]:
```
#checking for null values
protein_dataset.isna().sum()
```

Out[50]:
```
participant_id           0
protein_concentration    0
dtype: int64
```

Creating New Variables

In [51]:
```
# Renaming the participant_id to subject_id
protein_dataset.rename(columns={"participant_id": "subject_id"}, inplace=True)
```

In [52]:
```
# merging the clinical dataset with the protein dataset
clinical_dataset = pd.merge(clinical_dataset, protein_dataset, on="subject_id"
clinical_dataset
```

Out[52]:

	subject_id	age	sex	weight	height	trt_grp	RESPONSE	protein_concentration
0	SUBJ_001	46	Female	84.66	1.59	DRUG	N	148.0
1	SUBJ_002	47	Female	71.21	1.64	DRUG	Y	85.0
2	SUBJ_003	48	Female	69.85	1.73	CONTROL	N	183.0
3	SUBJ_004	59	Female	62.94	1.50	DRUG	Y	89.0
4	SUBJ_005	59	Female	113.91	1.63	CONTROL	N	137.0
...	...	...	...	...	...	...	...	...
763	SUBJ_764	61	Female	82.95	1.59	CONTROL	Y	101.0
764	SUBJ_765	65	Male	112.86	1.76	DRUG	N	122.0
765	SUBJ_766	60	Male	81.03	1.77	DRUG	N	121.0
766	SUBJ_767	53	Male	88.67	1.72	DRUG	Y	126.0
767	SUBJ_768	68	Female	80.29	1.63	DRUG	Y	93.0

768 rows × 8 columns

In [53]:
```
# Calculating and creating the BMI of patients
clinical_dataset["BMI_subject"] = clinical_dataset["weight"] / (clinical_datas
```

In [54]:
```
clinical_dataset
```

Out[54]:

	subject_id	age	sex	weight	height	trt_grp	RESPONSE	protein_concentration	BMI
0	SUBJ_001	46	Female	84.66	1.59	DRUG	N	148.0	3
1	SUBJ_002	47	Female	71.21	1.64	DRUG	Y	85.0	2
2	SUBJ_003	48	Female	69.85	1.73	CONTROL	N	183.0	2
3	SUBJ_004	59	Female	62.94	1.50	DRUG	Y	89.0	2
4	SUBJ_005	59	Female	113.91	1.63	CONTROL	N	137.0	4
...	...	...	...	...	...	...	...	...	
763	SUBJ_764	61	Female	82.95	1.59	CONTROL	Y	101.0	3
764	SUBJ_765	65	Male	112.86	1.76	DRUG	N	122.0	3
765	SUBJ_766	60	Male	81.03	1.77	DRUG	N	121.0	2
766	SUBJ_767	53	Male	88.67	1.72	DRUG	Y	126.0	2
767	SUBJ_768	68	Female	80.29	1.63	DRUG	Y	93.0	3

768 rows × 9 columns

Data Aggregation

```
In [55]: # Comparing mean age in two treatment groups
         grouped_data = clinical_dataset.groupby("trt_grp")["age"].mean()
```

```
In [56]: # displaying the data
         grouped_data
```

```
Out[56]: trt_grp
         CONTROL    61.862338
         DRUG       61.759791
         Name: age, dtype: float64
```

```
In [57]: # Comparing mean age in Responders vs non-responders groups
         grouped_data = clinical_dataset.groupby("RESPONSE")["age"].mean()
```

```
In [58]: # displaying the data
         grouped_data
```

```
Out[58]: RESPONSE
         N    61.748848
         Y    61.892216
         Name: age, dtype: float64
```

```
In [59]: # Comparing mean age for control vs drug group
         treatment_data = clinical_dataset[clinical_dataset["trt_grp"].isin(["CONTROL",

         grouped_data = treatment_data.groupby(["trt_grp", "RESPONSE"])["age"].mean()

         group1_non_responders = grouped_data.loc[("CONTROL", "N")]
         group1_responders = grouped_data.loc[("CONTROL", "Y")]
         group2_non_responders = grouped_data.loc[("DRUG", "N")]
         group2_responders = grouped_data.loc[("DRUG", "Y")]
```

```
In [60]: # displaying the mean age of the group
         grouped_data
```

```
Out[60]: trt_grp  RESPONSE
         CONTROL  N           62.042146
                  Y           61.483871
         DRUG     N           61.306358
                  Y           62.133333
         Name: age, dtype: float64
```

```
In [61]: # Comparing mean weight in Responders vs non-responders groups
         grouped_data = clinical_dataset.groupby("RESPONSE")["weight"].mean()
```

In [62]:
```
grouped_data
```

Out[62]:
```
RESPONSE
N    89.975700
Y    93.245076
Name: weight, dtype: float64
```

In [63]:
```
# Comparing mean weight in two treatment groups
grouped_data = clinical_dataset.groupby("trt_grp")["weight"].mean()
```

In [64]:
```
grouped_data
```

Out[64]:
```
trt_grp
CONTROL    91.956232
DRUG       90.835927
Name: weight, dtype: float64
```

In [65]:
```
# Comparing mean weight in Responders vs non-responders groups
grouped_data = clinical_dataset.groupby("RESPONSE")["protein_concentration"].m
```

In [66]:
```
grouped_data
```

Out[66]:
```
RESPONSE
N    134.884932
Y    104.537046
Name: protein_concentration, dtype: float64
```

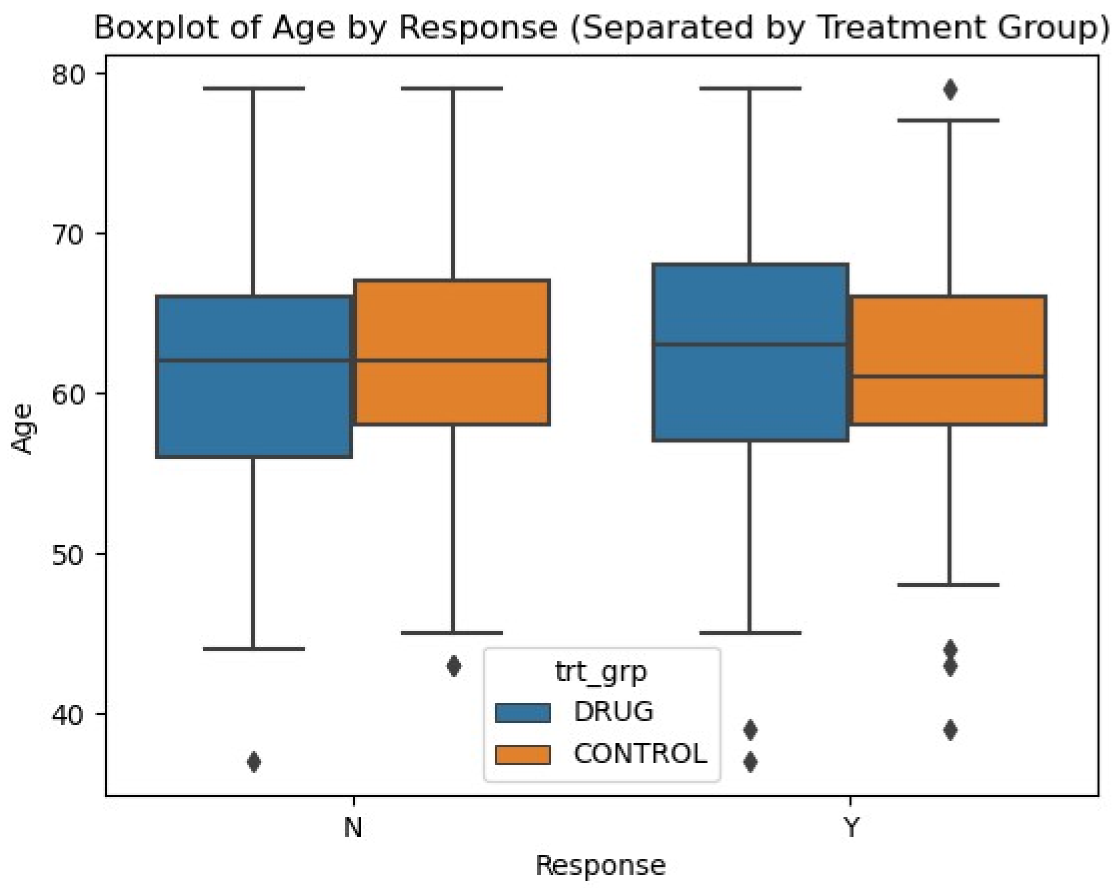
In [67]:
```
# Comparing mean protein concentration in two treatment groups
grouped_data = clinical_dataset.groupby("trt_grp")["protein_concentration"].me
```

In [68]:
```
grouped_data
```

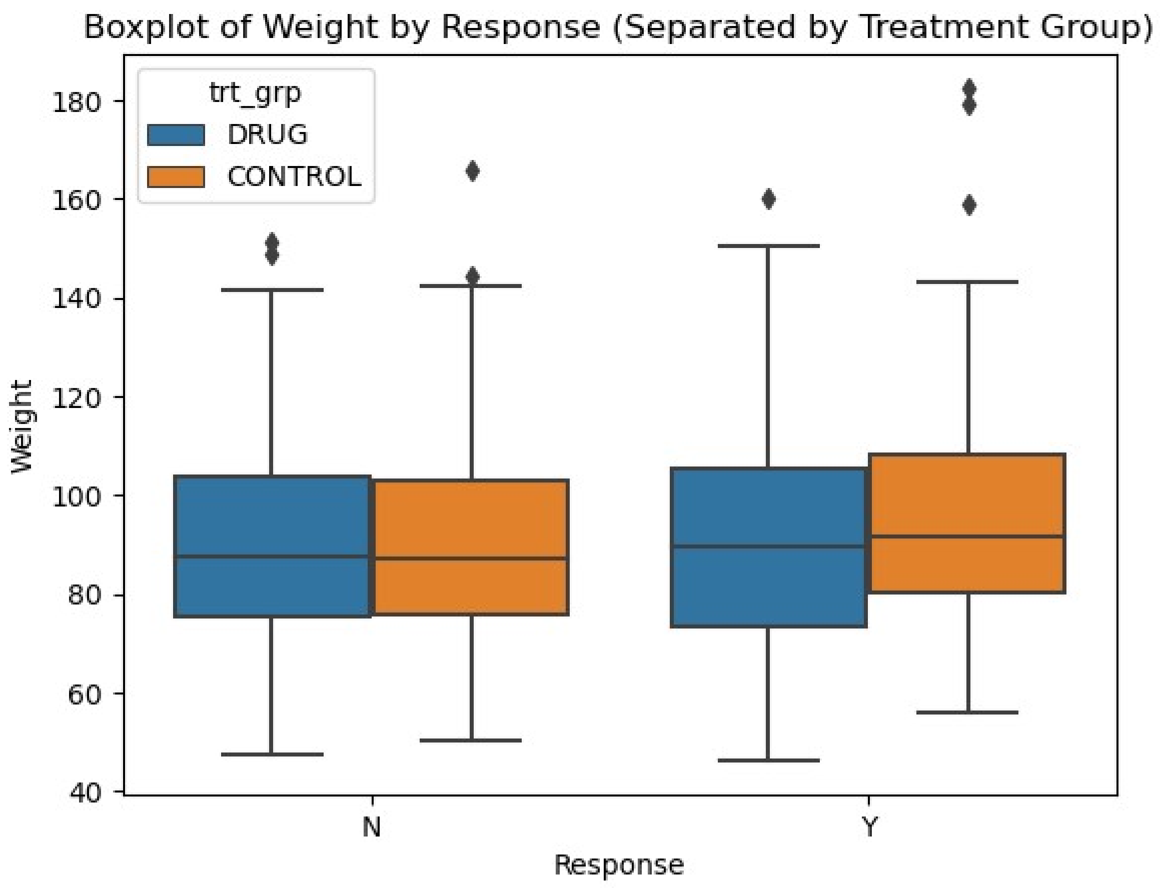
Out[68]:
```
trt_grp
CONTROL    121.297888
DRUG       122.077669
Name: protein_concentration, dtype: float64
```

Data Visualization

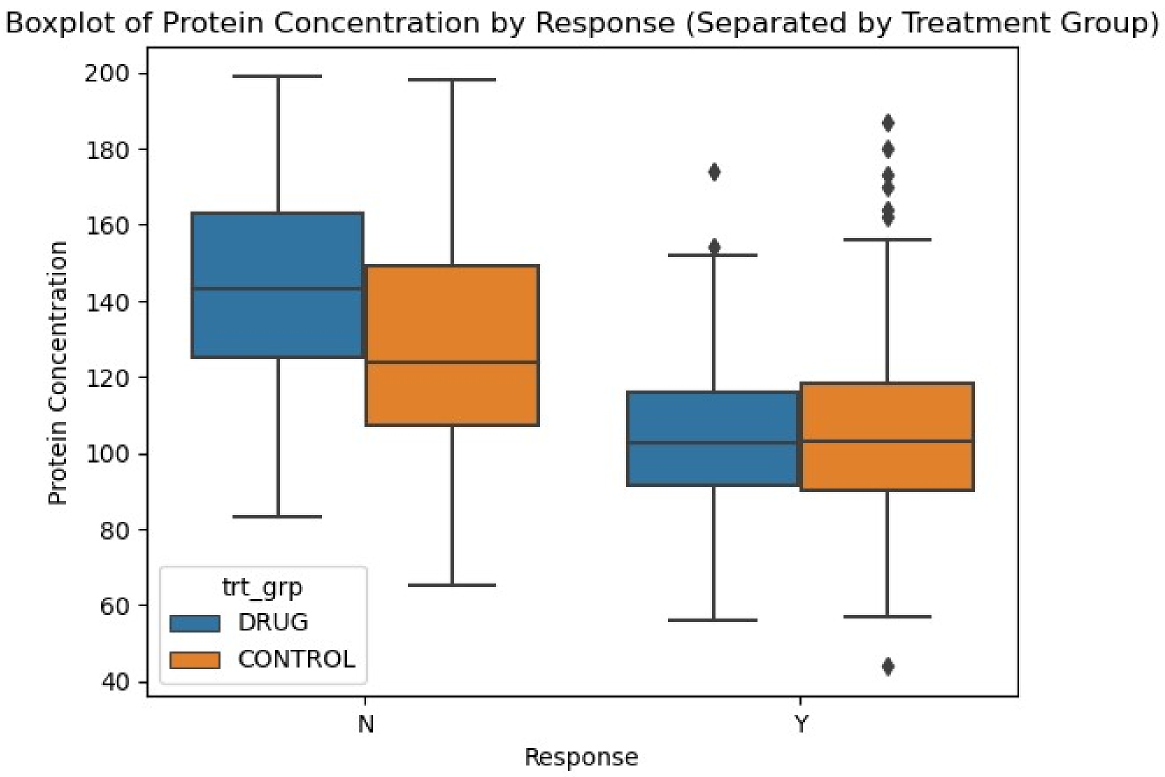
In [69]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Plotting Age by Response (Separated by Treatment Group)
sns.boxplot(x="RESPONSE", y="age", hue="trt_grp", data=clinical_dataset)
plt.title("Boxplot of Age by Response (Separated by Treatment Group)")
plt.xlabel("Response")
plt.ylabel("Age")
plt.show()
```



```
In [70]: # Plotting Weight by Response (Separated by Treatment Group)
         sns.boxplot(x="RESPONSE", y="weight", hue="trt_grp", data=clinical_dataset)
         plt.title("Boxplot of Weight by Response (Separated by Treatment Group)")
         plt.xlabel("Response")
         plt.ylabel("Weight")
         plt.show()
```

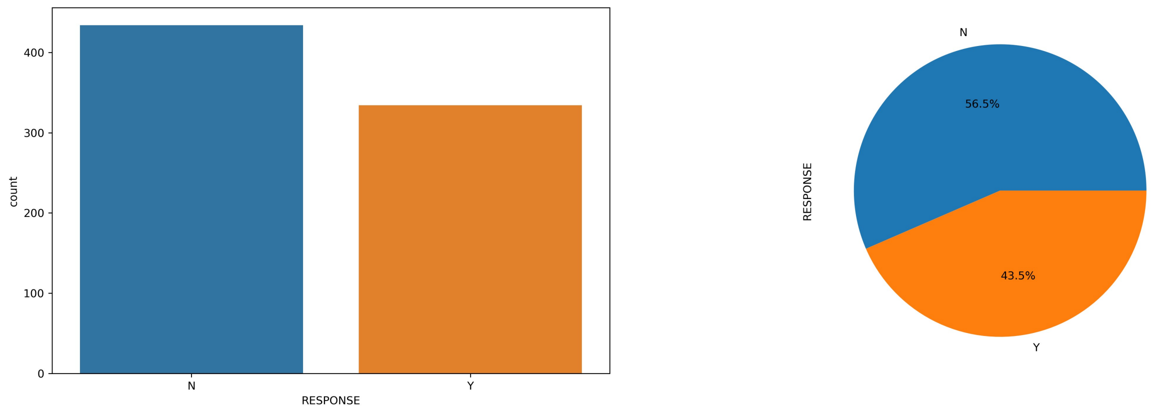


In [71]:
```
# Plotting  Protein Concentration by Response (Separated by Treatment Group)
sns.boxplot(x="RESPONSE", y="protein_concentration", hue="trt_grp", data=clini
plt.title("Boxplot of Protein Concentration by Response (Separated by Treatmen
plt.xlabel("Response")
plt.ylabel("Protein Concentration")
plt.show()
```



Data Modelling(using decision tree)

In [72]:
```
# seaborn plotting of the target variable (Response) of clinical dataset
fig, ax_position=plt.subplots(1,2,figsize=(20,6),dpi=270) # creates the framew
a = sns.countplot(x = 'RESPONSE', data = clinical_dataset, ax=ax_position[0])
a = clinical_dataset['RESPONSE'].value_counts().plot.pie(autopct="%1.1f%%", ax
```



In [73]:
```
from sklearn.model_selection import train_test_split # train_test_split is nee
from sklearn.preprocessing import MinMaxScaler # for normalisation or standard
from sklearn.preprocessing import LabelEncoder


# Convert target variable to numerical using label encoding
label_encoder = LabelEncoder()
clinical_dataset['RESPONSE'] = label_encoder.fit_transform(clinical_dataset['R
clinical_dataset['trt_grp'] = label_encoder.fit_transform(clinical_dataset['tr
clinical_dataset['sex'] = label_encoder.fit_transform(clinical_dataset['sex'])
clinical_dataset['subject_id'] = label_encoder.fit_transform(clinical_dataset[
```

In [74]:
```
clinical_dataset.head()
```

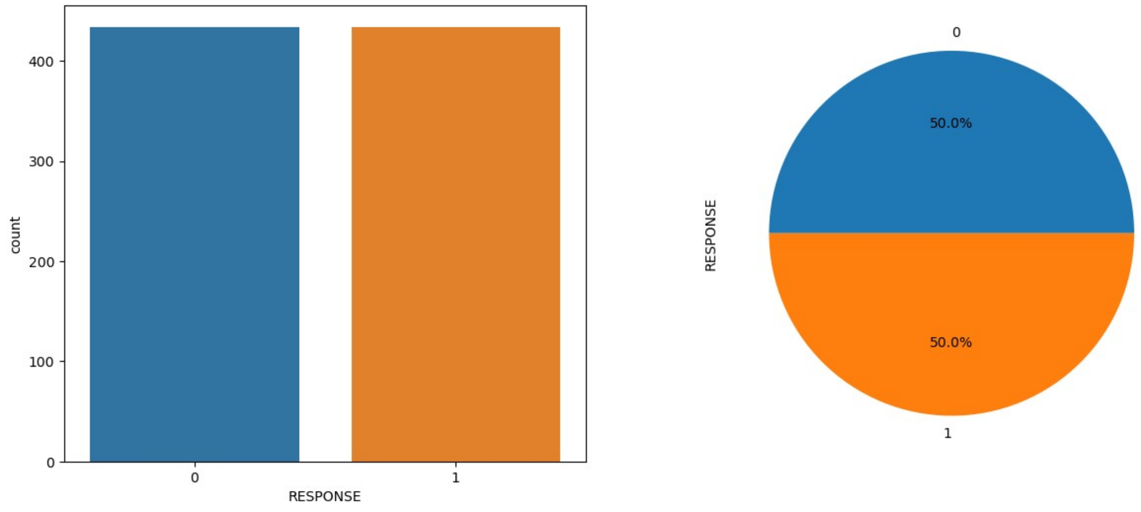
Out[74]:

	subject_id	age	sex	weight	height	trt_grp	RESPONSE	protein_concentration	BMI_subject
0	0	46	0	84.66	1.59	1	0	148.0	33.487599
1	1	47	0	71.21	1.64	1	1	85.0	26.476056
2	2	48	0	69.85	1.73	0	0	183.0	23.338568
3	3	59	0	62.94	1.50	1	1	89.0	27.973333
4	4	59	0	113.91	1.63	0	0	137.0	42.873273

In [75]:
```
# Split data into input and output variables
X = clinical_dataset.drop('RESPONSE', axis=1)
y = clinical_dataset['RESPONSE']
```

In [76]:
```
# import SMOTE
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42, k_neighbors = 2) # The object is created

# apply SMOTE to resample the dataset
X, y = sm.fit_resample(X, y)
```

In [77]:
```
# Plot of the dataset
bal_clinical_dataset = pd.concat([X, y], axis = 1) # creating a dataframe for
fig, ax=plt.subplots(1,2,figsize=(15,6)) # creating the axis shell for subplot
a = sns.countplot(x='RESPONSE',data=bal_clinical_dataset, ax=ax[0]) # assignin
a= bal_clinical_dataset['RESPONSE'].value_counts().plot.pie(autopct="%1.1f%%",
```



In [78]:
```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

#scaling our dataset
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [79]:
```
# importing our libraries and classifiers
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
```

In [80]:
```
# Fit decision tree classifier #training process
clf = DecisionTreeClassifier()

# Define cross-validation strategy
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Perform cross-validation
scores = cross_val_score(clf, X, y, cv=cv)
```

In [81]:
```
# Print mean and standard deviation of scores
print('Cross-validation scores: ', scores)
print('Mean score: ', scores.mean())
print('Standard deviation: ', scores.std())
```

```
Cross-validation scores:  [0.7183908  0.74137931 0.7183908  0.69364162 0.7167
6301]
Mean score:  0.7177131087635373
Standard deviation:  0.015108586829342682
```

In [82]:
```
clf.fit(X_train, y_train)
```

Out[82]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

In [83]:
```
# Make predictions on testing data
y_pred = clf.predict(X)

# Print classification report
print('Classification Report:')
print(classification_report(y, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.93      0.94       434
           1       0.93      0.94      0.94       434

    accuracy                           0.94       868
   macro avg       0.94      0.94      0.94       868
weighted avg       0.94      0.94      0.94       868
```

In [84]:
```
# Print confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(y, y_pred))
```

```
Confusion Matrix:
[[405  29]
 [ 24 410]]
```

the confusion matrix above shows the number of correctly classified enteries for each class from genre 0 to 1 which corresponds to the total number of Responses (Y OR N). from the first row starting form the left we see 403 enteries have been correctly classified for Response 0 which is "N" or non-responders and we see 31 entries wrongly classified. For next column we see 27 enteries wrong classified as Response 1 and 407 entries correctly classified as Responders or "Y" . the precision score of this model is in the 93% which shows promise with some tuning and more data to further test the model it can be used to correctly predict response.

overall the model performed well in classifying the dataset.