Experiment 2: Data Visualization and Data Cleaning

September 25, 2023

This experiment focuses on discussing concepts and implementing code snippets to demonstrate various techniques used for data cleaning as part of an Exploratory Data Analysis (EDA). EDA plays a crucial role in comprehending and examining datasets. Throughout the experiment, you will also need to solve a few exercises to demonstrate your comprehension and acquire the necessary skills.

---

1 Data Visualization

Data visualization is the process of transforming data into visual representations, such as charts, graphs, and maps. It is a powerful tool that can be used to communicate information clearly and concisely.

In artificial intelligence (AI), data visualization is used to:

- ***Explore and understand data***: Data visualization can be used to explore data and identify patterns and trends. This can be helpful for tasks such as data mining and machine learning.

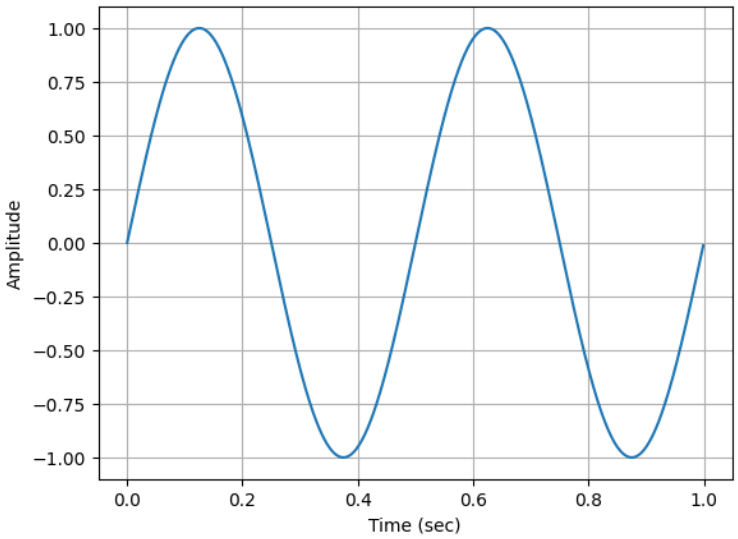
- ***Communicate insights***: Data visualization can be used to communicate insights from data to stakeholders. This can be helpful for tasks such as reporting and decision-making.

- ***Generate hypotheses***: Data visualization can be used to generate hypotheses about the data. This can be helpful for tasks such as research and problem-solving. As an example: A researcher can use a scatter plot to visualize the relationship between the number of hours studied and exam scores. If there is a positive correlation, the researcher can hypothesize that more hours studied leads to higher exam scores.

- ***Validate models***: Data visualization can be used to validate models created by AI algorithms. This can help to ensure that the models are accurate and reliable. For example, A retailer has developed a model to predict customer purchase behavior. They can use a heatmap to visualize the relationship between customer features and purchase behavior. The darker the color, the stronger the relationship.

##2.1.1 Using Matplotlib Matplotlib package is a data visualization library that is used to create professional figures and plots. To make a plot, you need first to import the **pyplot** sub-module and then use the **plot** method with proper arguments.

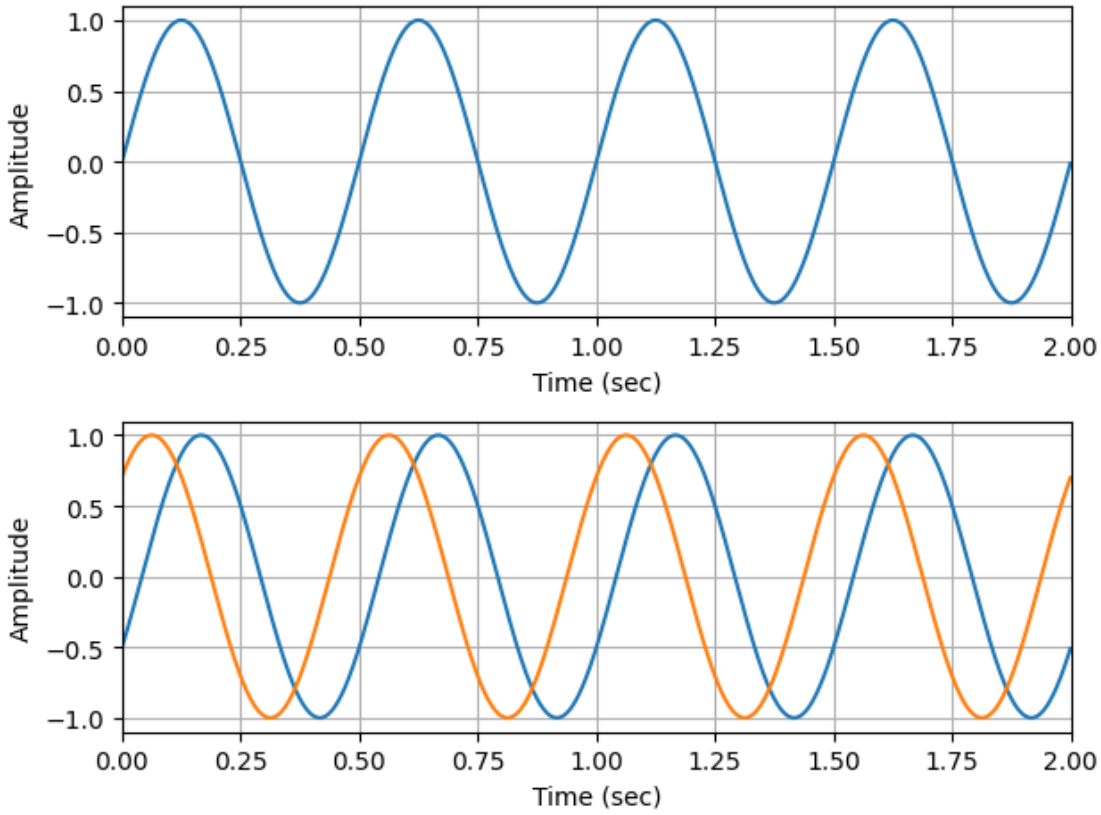
[ ]:
```
import matplotlib.pyplot as plt
import numpy as np

# generating a sinwwave signal
t = np.arange(0, 1, 0.001)
sig = np.sin(2 * np.pi * 2 * t)

plt.plot(t,sig)
plt.xlabel('Time (sec)')
plt.ylabel('Amplitude')
plt.grid(True)

plt.show()
```



The **Matplotlib** package can also be used to plot multiple axes in the same figure or two curves on the same axis.

[ ]:
```
# generating two sinwwave signals
t = np.arange(0, 2, 0.001)
sig1 = np.sin(2 * np.pi * 2 * t)
sig2 = np.sin(2 * np.pi * 2 * t - np.pi/6)
sig3 = np.sin(2 * np.pi * 2 * t + np.pi/4)

fig, axs = plt.subplots(2, 1)
axs[0].plot(t, sig1)
axs[0].set_xlim(0, 2)
axs[0].set_xlabel('Time (sec)')
axs[0].set_ylabel('Amplitude')
axs[0].grid(True)
```

```
axs[1].plot(t, sig2, t, sig3)
axs[1].set_xlim(0, 2)
axs[1].set_xlabel('Time (sec)')
axs[1].set_ylabel('Amplitude')
axs[1].grid(True)

fig.tight_layout()
plt.show()
```



Task 2.1: Refer to the matplot documentation at https://matplotlib.org/stable/gallery/color/named_colors.html to plot two curves in the same figure, add markers of specific size, add a legend, and add a figure title.

[ ]:
```
#write you code here
```

Task 2.2: Create two axes next to each other, and plot a sinwave in one axis and a cosine wave on the other. Add markers, legend, and a title for the figure.

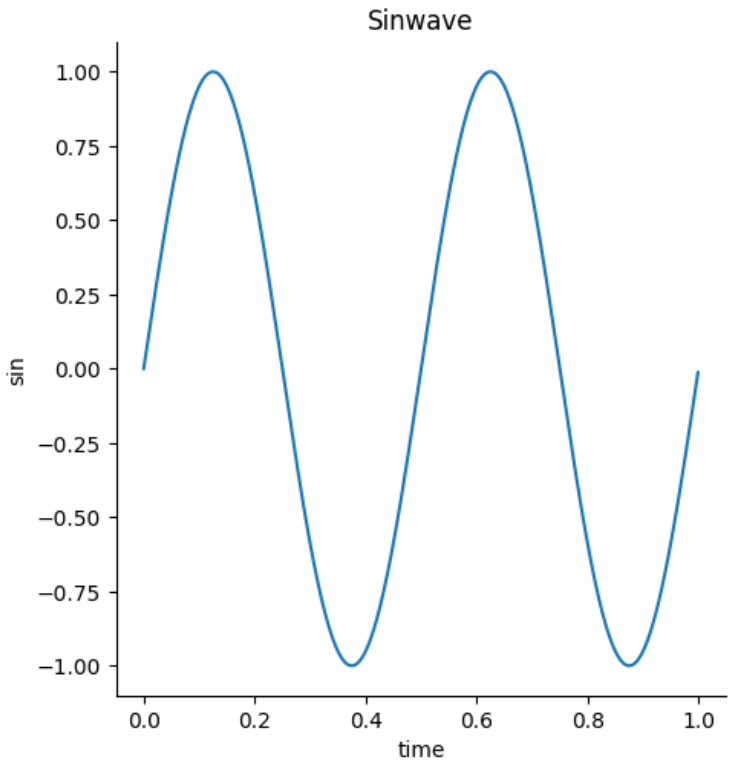
[ ]:
```
#write you code here
```

##2.1.2 Using Seaborn Seaborn is another data visualization library based on matplotlib. It pro-

vides a high-level interface for drawing informative graphics. To make a plot, you need first to import the **sns** sub-module and then use a specific method with proper arguments. For example you may use the **relplot** method to create relational plots (plots in which the relationship between two or more variables is visually represented).

[ ]:
```
import seaborn as sns
import pandas as pd

# generating a sinwwave signal
t = np.arange(0, 1, 0.001)
sin = np.sin(2 * np.pi * 2 * t)
cos = np.cos(2 * np.pi * 2 * t)

#Creating a dataframe from the time, sin, and cos curves
df = pd.DataFrame({'time':t, 'sin':sin, 'cos':cos})

# Create a visualization
sns.relplot(data=df,kind="line",x="time",y="sin").set(title='Sinwave')
```

[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7e859f6602b0>
```



You can also use the **relplot** function to create more advanced visualizations of data. For instance, let's take the "tips" dataset as an example. This dataset contains information about tips received

by a waiter over a few months in a restaurant. It has details like how much tip was given, the bill amount, whether the person paying the bill is male or female, if there were smokers in the group, the day of the week, the time of day, and the size of the group. To import the dataset use the following code

[ ]:
```
# Load an example dataset
tips = sns.load_dataset("tips")
tips.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   total_bill  244 non-null    float64
 1   tip         244 non-null    float64
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB
```

Execute the following code to view the first 10 rows in the dataset

[ ]:
```
tips.head()
```

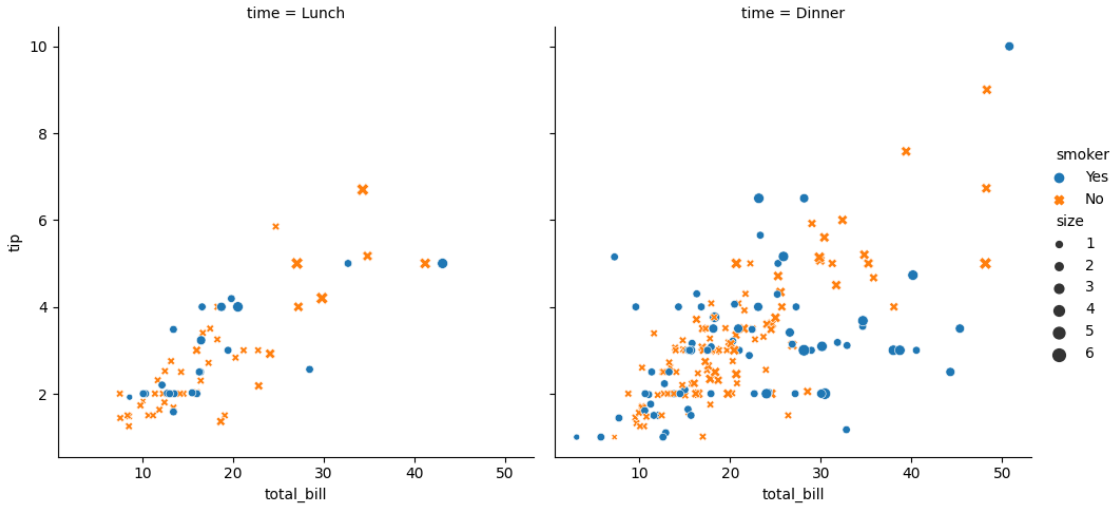
[ ]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Using the **relplot()** method helps us understand patterns in the dataset and how different factors might be connected.

[ ]:
```
# Create a visualization
sns.relplot(data=tips,
    x="total_bill", y="tip", col="time",
    hue="smoker", style="smoker", size="size",
)
```

[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7e859f379210>
```



From observing the visualization of the tips dataset, we can infer that as the total bill size grows, the tip value tends to increase proportionally. Additionally, it's apparent that both the total bill and tip value are higher when the group size is larger. **Can you observe any other patterns?**

Task 2.3: Load a dataset from the sns repository and then use the **relplot()** method to visualize and understand patterns in the dataset. You may list the datasets in the sns repository using the **sns.get_dataset_names()** method.

[ ]:
```
#write you code here
```

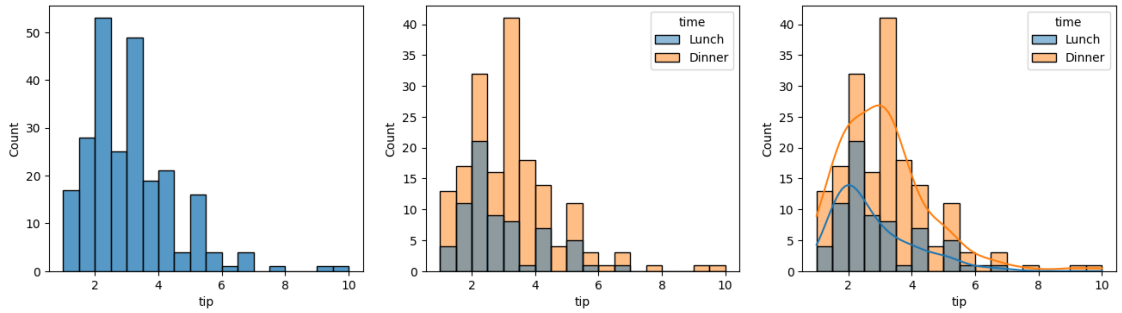
The **histplot()** is another method in the **sns** submodule that can be used to plot univariate or bivariate histograms to show distributions of datasets.

Note: A histogram is a graphical representation of data that shows the distribution of values within a dataset. It is a way to visualize how often different values or ranges of values occur in the data. In a histogram, the x-axis represents the possible values or ranges of values, and the y-axis represents the frequency or count of those values in the dataset. Each bar in the histogram represents a group or "bin" of values, and the height of the bar corresponds to the number of data points that fall into that group.

[ ]:
```
fig, axs = plt.subplots(figsize=(16, 4),ncols=3)

#Create histograms displaying the distribution of tip values
sns.histplot(data=tips, x="tip", ax=axs[0])

#Create histograms displaying the distribution of tip values based on the time␣
 ↪of day
sns.histplot(data=tips, x="tip", hue="time",ax=axs[1])

#Create histograms displaying the distribution of tip values based on the time␣
 ↪of day, and incorporate the actual distribution curve.
```

```
sns.histplot(data=tips, x="tip", hue="time",ax=axs[2], kde=True)
```

[ ]: <Axes: xlabel='tip', ylabel='Count'>



Task 2.4: Create a histogram plot for the dataset you loaded in task 2.3 and incorporate the actual distribution curve.

[ ]:
```
#write you code here
```

Task 2.5: Use the **scatterplot()** method within the **sns** submodule and the **subplots()** method within the **matplotlib** submodule to generate visual representations for the dataset you loaded in step 2.3.

[ ]:
```
#write you code here
```

##2.1.3 Using Pandas Data visualization using pandas is a common task in data analysis and manipulation. As explained in experiment #1, pandas provides an easy-to-use DataFrame structure that allows you to store, manipulate, and analyze data efficiently. When combined with data visualization libraries like Matplotlib or Seaborn, pandas can generate a wide range of visualizations to explore and communicate insights from your data. In this section, we will present few types of visualizations that can be created using pandas.

Let us create a sample DataFrame using the folowing code.

[ ]:
```
import pandas as pd
import numpy as np


# Create a sample DataFrame
data = {
    'Category': ['Fruits','Fruits','Fruits','Fruits','Fruits',␣
↪'Vegetables','Vegetables','Vegetables','Vegetables','Vegetables','Grains','Grains','Grains'],
    'Item': ['Apple', 'Banana', 'Orange', 'Mango', 'Grapes', 'Spinach',␣
↪'Tomato', 'Cucumber','Cauliflower','Eggplant','Rice', 'Wheat', 'Corn'],
    'Weight': [200, 150, 250, 150, 200, 120, 200, 120, 120, 250, 120, 300, 300],
    'Cost': [0.5, 0.3, 0.2, 2.5, 1.0, 1.5, 0.3, 0.3, 0.5, 1.2, 1.6, 0.8, 0.5],
    'Calories': [95, 23, 205, 335, 120, 33, 50, 250, 350, 300, 420, 200, 250]
```

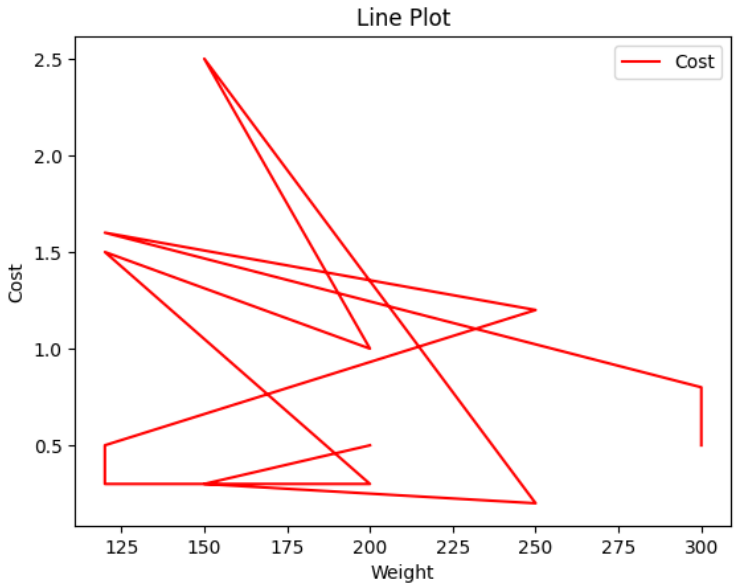
```
}


food = pd.DataFrame(data)
food.head()
```

[ ]:

	Category	Item	Weight	Cost	Calories
0	Fruits	Apple	200	0.5	95
1	Fruits	Banana	150	0.3	23
2	Fruits	Orange	250	0.2	205
3	Fruits	Mango	150	2.5	335
4	Fruits	Grapes	200	1.0	120

Line Plot: To create a line plot, you can use the **plot()** method on the DataFrame.

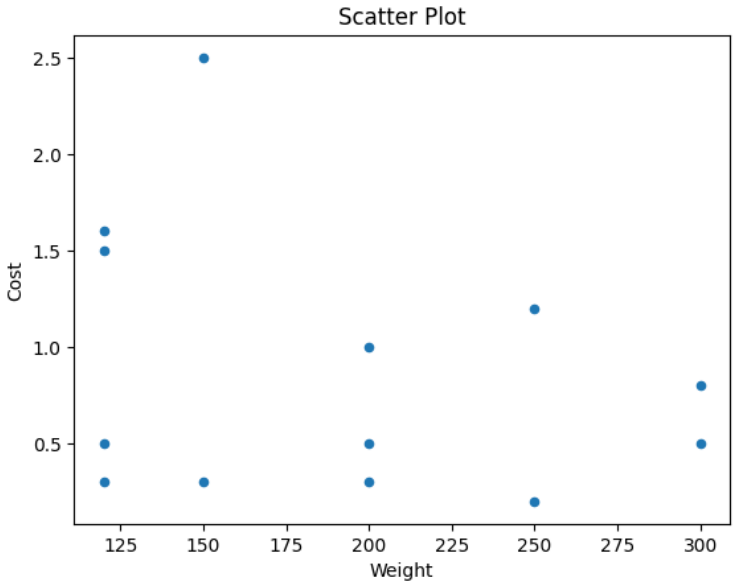
[ ]:
```
import matplotlib.pyplot as plt
food.plot(x='Weight', y='Cost', kind='line', color='red')
plt.xlabel('Weight')
plt.ylabel('Cost')
plt.title('Line Plot')
plt.show()
```



Scatter Plot: To create a scatter plot use the **plot()** method with **kind='scatter'**.

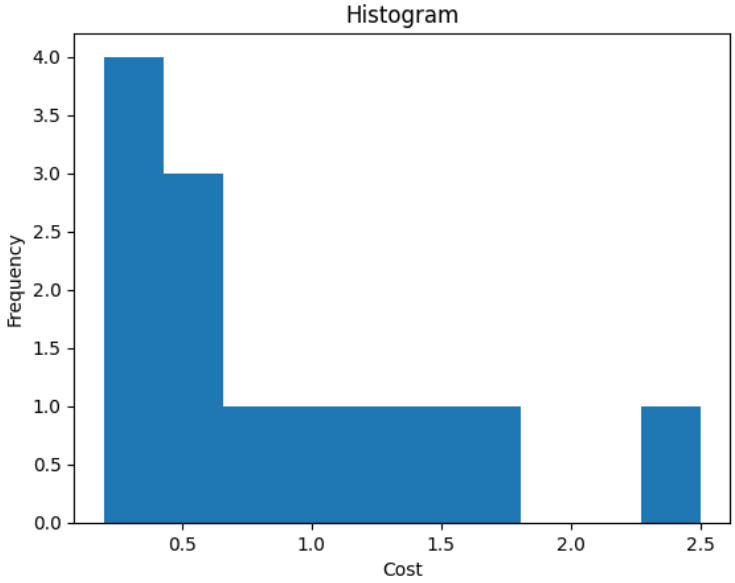
[ ]:
```
food.plot(x='Weight', y='Cost', kind='scatter')
plt.xlabel('Weight')
plt.ylabel('Cost')
plt.title('Scatter Plot')
```

```
plt.show()
```



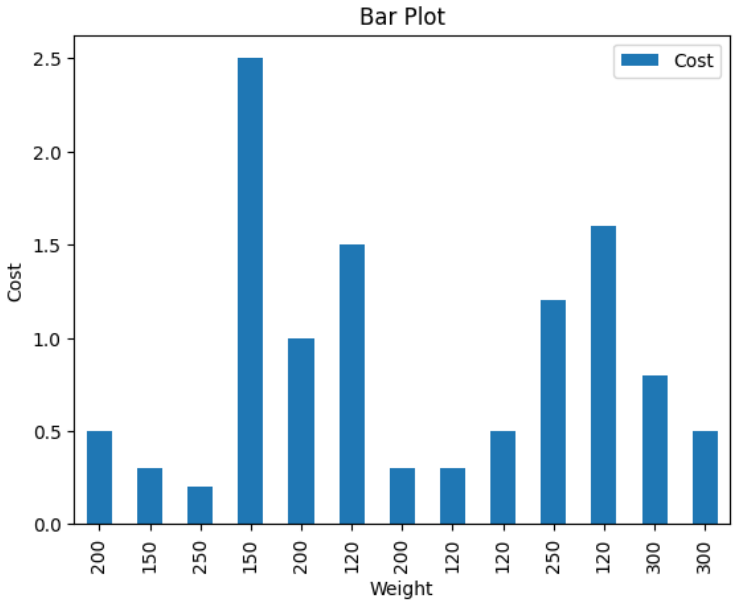
Histogram: To create a histogram use the **plot()** method with **kind='hist'**.

[ ]:
```
food['Cost'].plot(kind='hist', bins=10)
plt.xlabel('Cost')
plt.ylabel('Frequency')
plt.title('Histogram')
plt.show()
```



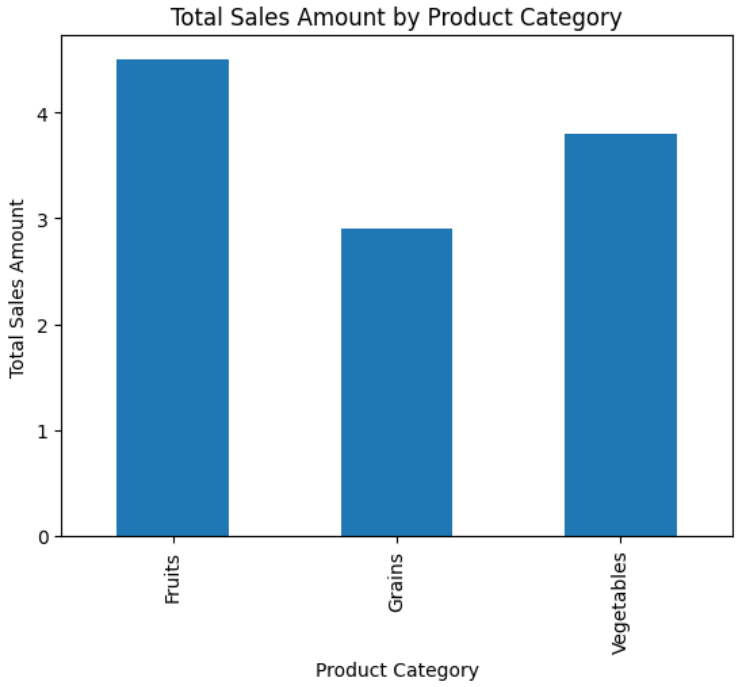
Bar Plot: To create a bar plot use the **plot()** method with **kind='bar'**.

```
[ ]: food.plot(x='Weight', y='Cost', kind='bar')
     plt.xlabel('Weight')
     plt.ylabel('Cost')
     plt.title('Bar Plot')
     plt.show()
```



You can also utilize the **groupby()** function along with the **plot()** function with the **kind='bar'** option to aggregate column values and generate insightful bar visualizations.

```
[ ]: # Grouping by 'Category' and summing 'Cost'
     grouped_data = food.groupby('Category')['Cost'].sum()

     # Creating a bar plot
     grouped_data.plot(kind='bar')
     plt.xlabel('Product Category')
     plt.ylabel('Total Sales Amount')
     plt.title('Total Sales Amount by Product Category')
     plt.show()
```



Task 2.7: Create a histogram for the food weights in the Food DataFrame defined in this section.

```
[ ]: #write you code here
```

Task 2.8: Generate informative bar charts illustrating the calorie distribution across food categories using the Food DataFrame introduced in this section.

```
[ ]: #write you code here
```

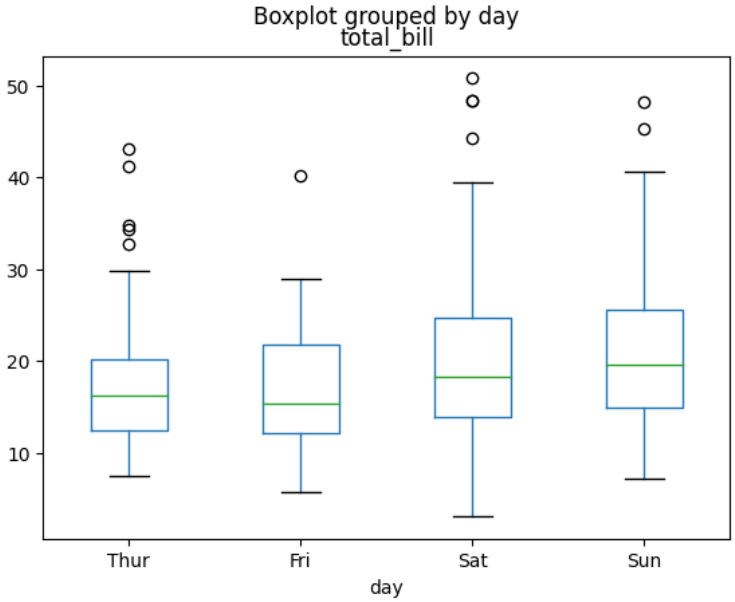
###2.1.4 Boxplot

A boxplot is a graphical representation that provides insights into the distribution and variability of data. It help us visualize the spread and central tendency of the data. A boxplot consists of several components: a box, whiskers, and individual data points. The rectangular "box" is drawn to represent the interquartile range (IQR), which spans from the first quartile (Q1) to the third quartile (Q3) of the data. The first quartile (Q1) represents the point where a quarter (25%) of the data values fall below when arranged in increasing order. On the other hand, the third quartile (Q3), marks the threshold beneath which three-quarters (75%) of the data values are situated when organized in increasing order. The whiskers extend from the box and show the range within which most of the data falls. The lower whisker typically extends to the smallest data point that is greater than or equal to Q1 - 1.5 * IQR. The upper whisker typically extends to the largest data point that is less than or equal to Q3 + 1.5 * IQR.

The boxplot can be created using the **boxplot()** method within the **panda** package (panada DataFrame).

```
[ ]: tips.boxplot(by ='day', column =['total_bill'], grid = False)
```

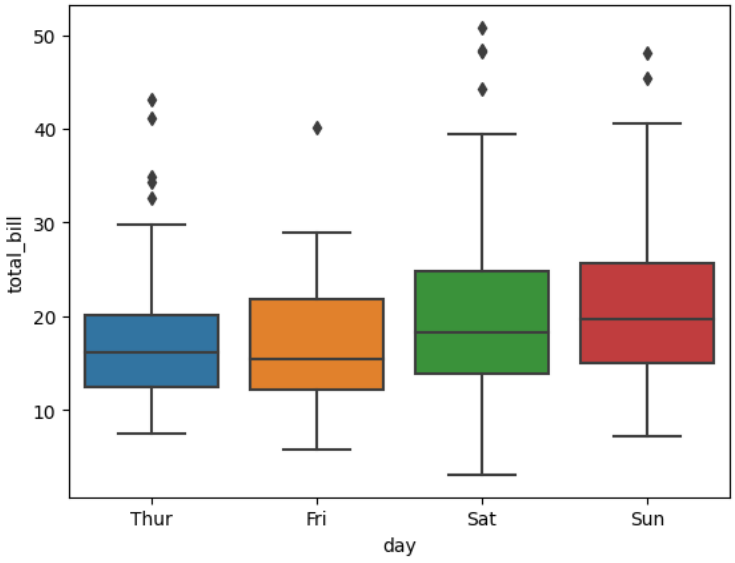
[ ]: <Axes: title={'center': 'total_bill'}, xlabel='day'>



The boxplot can also be generated using the **boxplot()** method within the **sns** submodule.

[ ]:
```
sns.boxplot(x = 'day', y = 'total_bill', data = tips)
```

[ ]: <Axes: xlabel='day', ylabel='total_bill'>



Task 2.9: Use the **boxplot()** method within the **sns** submodule to generate boxplot for the dataset

you loaded in step 2.3.

[ ]:
```
#write you code here
```

2 Descriptive Statistics

Descriptive statistics involves analyzing and summarizing data to gain insights into its central tendencies, variability, and overall distribution. It playes a crucial role in machine learning for many reasons including data understanding and exploration and data cleaning and preprocessing. In data understanding and exploration, descriptive statistics provide an initial overview of the dataset, helping you understand its distribution, central tendencies, and variability. This exploration phase is vital for identifying data patterns, anomalies, and potential issues that might impact the quality of your machine learning models. While in data cleaning and preprocessing, descriptive statistics help you identify missing values, outliers, and inconsistencies that need to be addressed. Cleaning and preprocessing ensure that your machine learning model receives accurate and reliable input data.

In this section, we will demonstrate descriptive statistics by working with the data stored in the **ENCS5141_Exp2_DescriptiveStatistics.csv** and **ENCS5141_Exp2_ShapeDistribution.csv** files. These files can be found in the GitHub repository located at https://github.com/mkjubran/ENCS5141Datasets. To clone the repository, you can execute the following code:

[ ]:
```
!rm -rf ./ENCS5141Datasets
!git clone https://github.com/mkjubran/ENCS5141Datasets.git
```

```
Cloning into 'ENCS5141Datasets'...
remote: Enumerating objects: 44, done.
remote: Counting objects: 100% (44/44), done.
remote: Compressing objects: 100% (38/38), done.
remote: Total 44 (delta 7), reused 38 (delta 4), pack-reused 0
Receiving objects: 100% (44/44), 2.94 MiB | 4.60 MiB/s, done.
Resolving deltas: 100% (7/7), done.
```

First we will read the ENCS5141_Exp2_DescriptiveStatistics.csv into a DataFrame use the **pd.read_csv()** method.

[ ]:
```
import pandas as pd
df = pd.read_csv("/content/ENCS5141Datasets/ENCS5141_Exp2_DescriptiveStatistics.
 ↪csv")
df.head()
```

[ ]:

	col_1	col_2	col_3	col_4	col_5	col_6
0	0.22	-4.21	-1.07	7.64	-0.88	-4.59
1	1.23	-5.61	2.08	6.54	1.21	1.38
2	1.41	-4.66	-1.49	4.46	-2.39	-0.54
3	2.20	-5.31	0.07	2.54	0.40	-2.11
4	-1.52	-4.34	0.79	4.06	-4.97	1.38

##2.2.1 Central Tendency The following measures are employed to assess the central tendency of a distribution of data:
Mean: The average value of the data.
Median: The middle value when the data is sorted.
Mode: The value that appears most frequently in the data.
Pandas provides methods like **mean()**, **median()**, and **mode()** to calculate these measures.

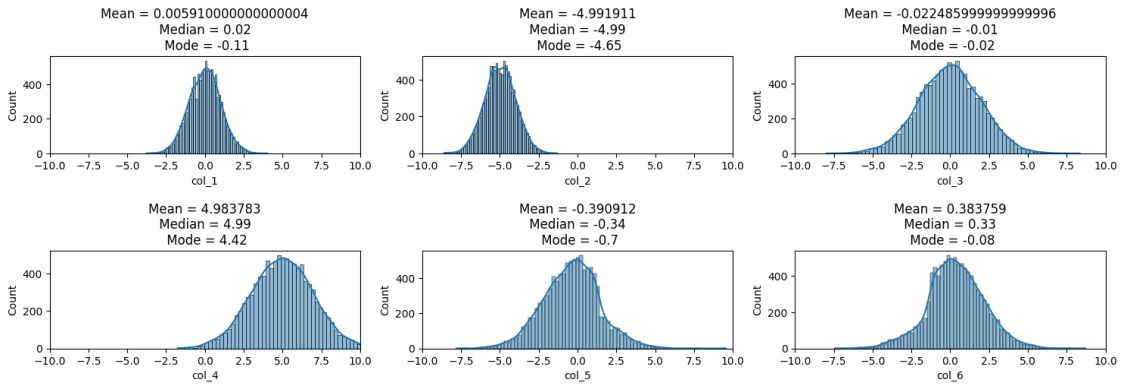
[ ]:
```
# Calculate the mean for each column in the DataFrame
Mean = df.mean()
print(f"Mean values: {Mean.values}\n")

# Calculate the median for each column in the DataFrame
Median = df.median()
print(f"Median values: {Median.values}\n")

# Calculate the mode for each column in the DataFrame
Mode = df.mode().values
print(f"Mode values: {Mode}\n")
```

```
Mean values: [ 0.00591  -4.991911 -0.022486  4.983783 -0.390912  0.383759]

Median values: [ 0.02 -4.99 -0.01  4.99 -0.34  0.33]

Mode values: [[-0.11 -4.65 -0.02  4.42 -0.7  -0.08]
 [ 0.12   nan   nan   nan   nan   nan]]
```

To gain a visual understanding of the central tendency measures of the loaded data, we can utilize the **sns.histplot()** method to display the column distributions of the dataset.

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
fig, axs = plt.subplots(figsize=(18, 5),ncols=3,nrows=2)

#Create histograms displaying the distribution of specific column values and␣
 ↪incorporate the actual distribution curve.
sns.histplot(data=df, x="col_1", ax=axs[0,0], kde=True);axs[0,0].set(xlim=(-10,␣
 ↪10))
axs[0,0].set_title(f"Mean = {Mean[0]}\n Median = {Median[0]} \n Mode =␣
 ↪{Mode[0,0]}")

sns.histplot(data=df, x="col_2", ax=axs[0,1], kde=True);axs[0,1].set(xlim=(-10,␣
 ↪10))
axs[0,1].set_title(f"Mean = {Mean[1]}\n Median = {Median[1]} \n Mode =␣
 ↪{Mode[0,1]}")

sns.histplot(data=df, x="col_3", ax=axs[0,2], kde=True);axs[0,2].set(xlim=(-10,␣
 ↪10))
```

```
axs[0,2].set_title(f"Mean = {Mean[2]}\n Median = {Median[2]} \n Mode =␣
↪{Mode[0,2]}")

sns.histplot(data=df, x="col_4", ax=axs[1,0], kde=True);axs[1,0].set(xlim=(-10,␣
↪10))
axs[1,0].set_title(f"Mean = {Mean[3]}\n Median = {Median[3]} \n Mode =␣
↪{Mode[0,3]}")

sns.histplot(data=df, x="col_5", ax=axs[1,1], kde=True);axs[1,1].set(xlim=(-10,␣
↪10))
axs[1,1].set_title(f"Mean = {Mean[4]}\n Median = {Median[4]} \n Mode =␣
↪{Mode[0,4]}")

sns.histplot(data=df, x="col_6", ax=axs[1,2], kde=True);axs[1,2].set(xlim=(-10,␣
↪10))
axs[1,2].set_title(f"Mean = {Mean[5]}\n Median = {Median[5]} \n Mode =␣
↪{Mode[0,5]}")

fig.subplots_adjust(hspace=1)
```



Note: The bell-like shapes above (especialy for col_1, col_2, col_3, and 'col_4') are for normal distibutions also known as a Gaussian distribution. In a normal distribution, the mean, median, and mode are all centered at the same value. This central value is the highest point on the symmetrical curve. It's important to note that the mean, median, and mode being the same in a normal distribution is a characteristic that distinguishes it from other types of distributions.

Task 2.10: Explain how the mean, median, and mode measure central tendency by observing the distribution curves depicted in the graphs above.

##2.2.2 Variation The subsequent metrics are utilized to evaluate the dispersion of data distribution:
Variance: A measure of how much the data points deviate from the mean.
Standard Deviation: The square root of the variance, indicating the spread of data.
Pandas provides functions such as **var()** and **std()** for computing the variance and standard devi-

ation of columns within the DataFrame.

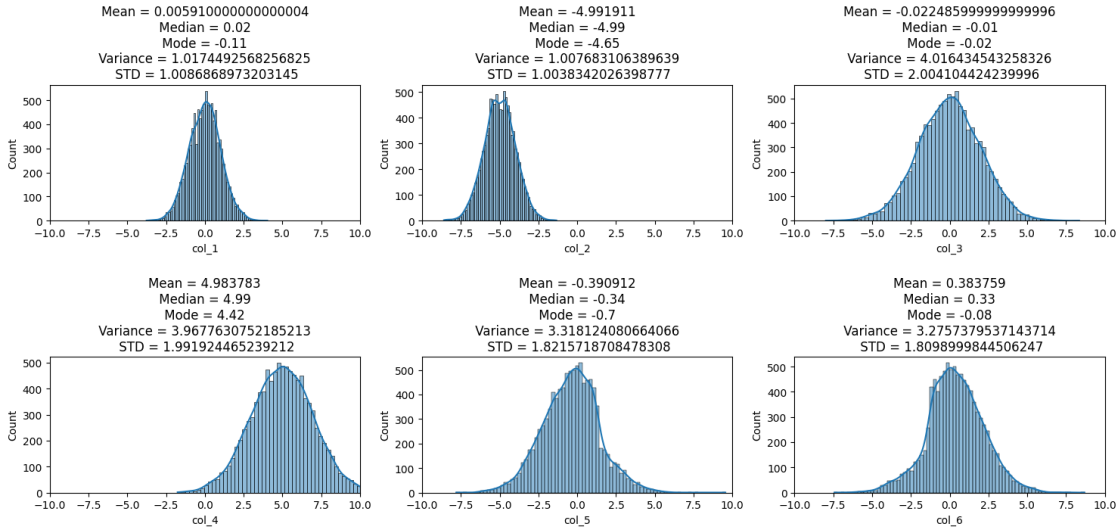
[ ]:
```
# Calculate the variance for each column in the DataFrame
Variance = df.var()
print(f"Variance values: {Variance.values}\n")

# Calculate the standard deviaton for each column in the DataFrame
STD = df.std()
print(f"Standard deviation values: {STD.values}\n")
```

```
Variance values: [1.01744926 1.00768311 4.01643454 3.96776308 3.31812408
3.27573795]

Standard deviation values: [1.0086869  1.0038342  2.00410442 1.99192447
1.82157187 1.80989998]
```

Again, let us use the use the **sns.histplot()** method to gain a visual understanding of the variation measures of the loaded data.

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
fig, axs = plt.subplots(figsize=(18, 7),ncols=3,nrows=2)

#Create histograms displaying the distribution of specific column values and␣
 ↪incorporate the actual distribution curve.
sns.histplot(data=df, x="col_1", ax=axs[0,0], kde=True);axs[0,0].set(xlim=(-10,␣
 ↪10))
axs[0,0].set_title(f"Mean = {Mean[0]}\n Median = {Median[0]} \n Mode =␣
 ↪{Mode[0,0]} \n Variance = {Variance[0]} \n STD = {STD[0]}")

sns.histplot(data=df, x="col_2", ax=axs[0,1], kde=True);axs[0,1].set(xlim=(-10,␣
 ↪10))
axs[0,1].set_title(f"Mean = {Mean[1]}\n Median = {Median[1]} \n Mode =␣
 ↪{Mode[0,1]} \n Variance = {Variance[1]} \n STD = {STD[1]}")

sns.histplot(data=df, x="col_3", ax=axs[0,2], kde=True);axs[0,2].set(xlim=(-10,␣
 ↪10))
axs[0,2].set_title(f"Mean = {Mean[2]}\n Median = {Median[2]} \n Mode =␣
 ↪{Mode[0,2]} \n Variance = {Variance[2]} \n STD = {STD[2]}")

sns.histplot(data=df, x="col_4", ax=axs[1,0], kde=True);axs[1,0].set(xlim=(-10,␣
 ↪10))
axs[1,0].set_title(f"Mean = {Mean[3]}\n Median = {Median[3]} \n Mode =␣
 ↪{Mode[0,3]} \n Variance = {Variance[3]} \n STD = {STD[3]}")

sns.histplot(data=df, x="col_5", ax=axs[1,1], kde=True);axs[1,1].set(xlim=(-10,␣
 ↪10))
```

```
axs[1,1].set_title(f"Mean = {Mean[4]}\n Median = {Median[4]} \n Mode =␣
↪{Mode[0,4]} \n Variance = {Variance[4]} \n STD = {STD[4]}")

sns.histplot(data=df, x="col_6", ax=axs[1,2], kde=True);axs[1,2].set(xlim=(-10,␣
↪10))
axs[1,2].set_title(f"Mean = {Mean[5]}\n Median = {Median[5]} \n Mode =␣
↪{Mode[0,5]} \n Variance = {Variance[5]} \n STD = {STD[5]}")

fig.subplots_adjust(hspace=1.0)
```



Task 2.11: Explain how the variance and standard deviation measure the variation by observing the distribution curves depicted in the graphs above.

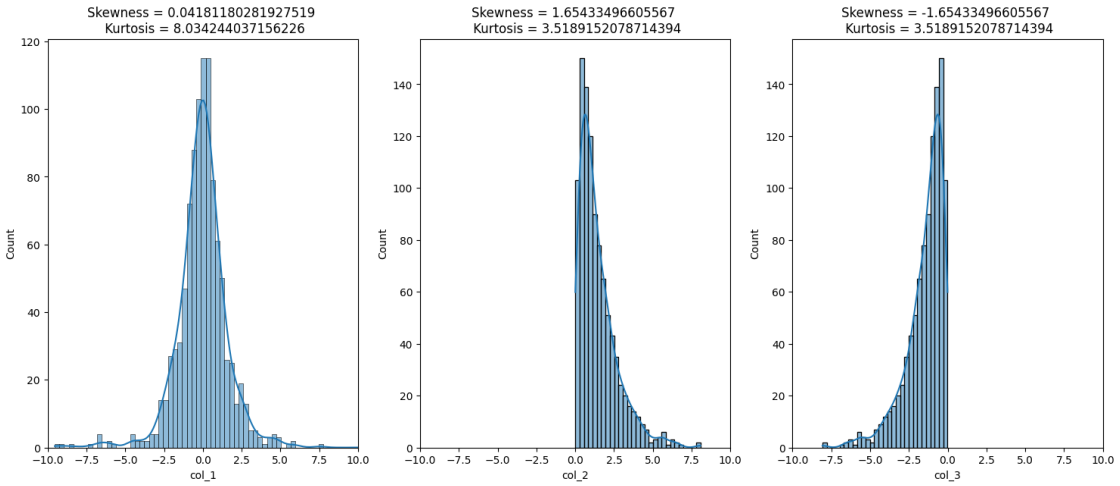
##2.2.3 Shape of Distribution Skewness and kurtosis are statistical measures that provide insights into the shape of a distribution:
Skewness: Measures the asymmetry of the data distribution.
Kurtosis: Measures the peakedness of the data distribution.
If you need to calculate skewness and kurtosis for multiple columns or across the entire DataFrame, you can use **df.skew()** and **df.kurtosis()** without specifying a column name. These functions return Series with the skewness or kurtosis values for each column. To grasp the connection between skewness and kurtosis and the form of the distribution, we'll employ the dataset contained within ENCS5141_Exp2_ShapeDistribution.csv.

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

df_shape = pd.read_csv("/content/ENCS5141Datasets/
↪ENCS5141_Exp2_ShapeDistribution.csv")
df_shape.head()
```

```
# Calculate the Skewness for each column in the DataFrame
Skewness = df_shape.skew()
print(f"Skewness values: {Skewness.values}\n")

# Calculate the Kurtosis for each column in the DataFrame
Kurtosis = df_shape.kurtosis()
print(f"Kurtosis values: {Kurtosis.values}\n")

fig, axs = plt.subplots(figsize=(18, 7),ncols=3)

#Create histograms displaying the distribution of specific column values and␣
 ↪incorporate the actual distribution curve.
sns.histplot(data=df_shape, x="col_1", ax=axs[0], kde=True);axs[0].
 ↪set(xlim=(-10, 10))
axs[0].set_title(f"Skewness = {Skewness[0]} \n Kurtosis = {Kurtosis[0]}")

sns.histplot(data=df_shape, x="col_2", ax=axs[1], kde=True);axs[1].
 ↪set(xlim=(-10, 10))
axs[1].set_title(f"Skewness = {Skewness[1]} \n Kurtosis = {Kurtosis[1]}")

sns.histplot(data=df_shape, x="col_3", ax=axs[2], kde=True);axs[2].
 ↪set(xlim=(-10, 10))
axs[2].set_title(f"Skewness = {Skewness[2]} \n Kurtosis = {Kurtosis[2]}")
```

```
Skewness values: [ 0.0418118   1.65433497 -1.65433497]

Kurtosis values: [8.03424404 3.51891521 3.51891521]
```

[ ]: Text(0.5, 1.0, 'Skewness = -1.65433496605567 \n Kurtosis = 3.5189152078714394')



Note: - If skewness is close to 0, the distribution is approximately symmetric. - If skewness is negative, the tail is longer on the left side (left-skewed). - If skewness is positive, the tail is longer on the right side (right-skewed). - If kurtosis is close to 3, the distribution has similar tails as a normal distribution. - If kurtosis is less than 3, the distribution has lighter tails and a flatter peak (a lighter-tailed distribution has a lower probability of producing values that are far from the mean compared to a distribution with heavier tails. This means that extreme values or outliers are less common in a dataset that follows a lighter-tailed distribution). - If kurtosis is greater than 3, the distribution has heavier tails and a sharp peak (a heavy-tailed distribution is one where the tail of the distribution decays more slowly than that of a normal distribution, implying that extreme values are more likely to occur.).

Task 2.11: Explain how the skewness and kurtosis are related to the shape of the distribution by observing the distribution curves depicted in the graphs above,

Task 2.12: Compute the skewness and kurtosis of the dataset in ENCS5141_Exp2_DescriptiveStatistics.csv, and create relevant visualizations to showcase the connection between the skewness, kurtosis, and the underlying distribution shape of the data.

[ ]:
```
#write you code here
```

##2.2.4 Quantiles **Percentiles**: Values below which a given percentage of data falls.
Interquartile Range (IQR): The range between the 25th and 75th percentiles.
Pandas offers the **quantile()** method for calculating percentiles. For instance, to obtain the 75th percentile, employ quantile(0.75), and for the 25th percentile, utilize quantile(0.25). Similarly, to compute the interquartile range (IQR), you can apply the formula quantile(0.75) - quantile(0.25). This is demonstrated in the code snippet below.

[ ]:
```
# Compute percentiles using Pandas quantile() function
percentile_25 = df_shape['col_1'].quantile(0.25)
percentile_75 = df_shape['col_1'].quantile(0.75)

print("25th Percentile:", percentile_25)
print("75th Percentile:", percentile_75)

# Compute interquartile range (IQR)
iqr = percentile_75 - percentile_25
print("Interquartile Range (IQR):", iqr)
```

```
25th Percentile: -0.7994611007787031
75th Percentile: 0.7335962474687701
Interquartile Range (IQR): 1.5330573482474732
```

3 Handling Missing Data

What is missing data?
Missing data is data that is not available for one or more observations in a dataset. It can occur for a variety of reasons, such as human error, equipment malfunction, or deliberate omission.

Why is handling missing data important?
Missing data can have a significant impact on the quality and accuracy of a dataset. If not handled

properly, it can lead to biased results and inaccurate conclusions.

How to handle missing data?
There are a number of methods for handling missing data. The best method to use will depend on the specific circumstances and the goals of the analysis. Some common methods include:

- ***Dropping/Deletion***: This method is used if the number of missing values in a row is relatively small, and you believe these rows do not carry essential information for your analysis. Use the **dropna()** method in pandas to remove rows with missing values from your dataset.

- ***Create a Separate Category***: If missing values represent a distinct category or carry specific meaning in your analysis, assign a special label or category to missing values. This makes them a distinct class in your analysis.

- ***Imputation***: The imputation of missing values is used if the proportion of missing values is significant, and you believe these data points are valuable for your analysis. You can impute missing values using various techniques:
- Mean, Median, or Mode Imputation: Replace missing values with the mean, median, or mode of the observed values in the respective column.
- Regression Imputation: Train a regression model to predict missing values based on other features in the dataset.
- K-Nearest Neighbors (KNN) Imputation: Impute missing values by averaging the values of the K-nearest neighbors in feature space.
- Advanced Imputation Techniques: Utilize more advanced methods like decision trees, random forests, or matrix factorization.
- ***Modeling***: When you have sufficient data with complete values and believe that missing values can be predicted accurately. Treat predicting missing values as a separate machine learning task. Train a model to predict missing values based on other features in your dataset. This can be a more complex method, but it can also be more accurate than imputation.
- ***Time-Series Interpolation***: This method can be applied when dealing with time-series data with missing values. Interpolate missing values based on the observed values in the time series. Methods like linear interpolation or cubic spline interpolation can be used.
- ***Use Domain Knowledge***: Employ your domain expertise to make informed choices regarding missing data. Utilize your specialized knowledge to decide the most suitable approach for managing missing values, taking into account the dataset's context.

What is the impact of missing data?
The impact of missing data on a dataset will depend on the following factors:
- ***The amount of missing data***: The more missing data there is, the greater the impact it will have.
- ***The type of missing data***: Missing data can be either missing completely at random (MCAR), missing at random (MAR), or missing not at random (MNAR). MCAR is the least harmful type of missing data, while MNAR is the most harmful.

- ***The goals of the analysis***: The goals of the analysis will also affect the impact of missing data. If the analysis is sensitive to missing data, then it is important to use a method that will minimize its impact.

Case on Missing Data: Clinical Trial on Drug Efficacy

Imagine a pharmaceutical company conducts a clinical trial to evaluate the efficacy of a new drug in treating a particular medical condition. The trial spans several months and involves regular check-ups and tests for enrolled patients.

Now, let's consider the potential for missing data:

- ***Patient Dropout***: During the trial, some patients may drop out for various reasons, such as experiencing side effects, personal decisions, or being lost to follow-up. As a result, their data is missing from later time points.
- ***Incomplete Records***: Not all patients may complete every scheduled test or check-up. Some may miss appointments, leading to missing test results or medical data.
- ***Measurement Errors***: Occasionally, technical issues or measurement errors can result in missing or unreliable data points for certain patients.

In this case, the analysis of the drug's efficacy could be sensitive to the missing data. If the missing data isn't handled appropriately, it may bias the results or lead to incorrect conclusions about the drug's effectiveness. For example:

- If patients who experienced severe side effects and dropped out are not properly accounted for, the analysis might underestimate the drug's potential risks.
- Missing test results could affect the assessment of treatment outcomes and the drug's impact on the condition being studied.

Handling Missing Data in Cardiovascular Disease Dataset

In this section, we will demonstrate how to clean data. We will use a modified version of the cardiovascular dataset from Kaggle (https://www.kaggle.com/code/sulianova/eda-cardiovascular-data/data). Cardiovascular disease is a group of diseases that affect the heart and blood vessels. It is the leading cause of death in the world. This dataset contains 70,000 records of patient data with 12 features. The target variable "cardio" is equal to 1 if the patient has cardiovascular disease and 0 if the patient is healthy. The target variable is the variable that we are trying to predict. In this case, the target variable is "cardio", which indicates whether the patient has cardiovascular disease.

Initially, we need to import the numpy, pandas, and matplotlib libraries that will be used during data cleaning.

[ ]:
```
import numpy as np
import pandas as pd
from matplotlib import rcParams
import matplotlib.pyplot as plt
```

Accessing the Dataset

The cardiovascular disease dataset is saved in a file named **ENCS5141_Exp2_Cardiovascular.csv**. This file is located within the GitHub reposi-

tory that was accessed in section 2.2. To read and display the dataset, you can execute the following code.

[ ]:
```
# read the dataset using pandas
df_cardio = pd.read_csv("/content/ENCS5141Datasets/ENCS5141_Exp2_Cardiovascular.
 ↪csv",sep=";")

# display the first 5 lines
df_cardio.head()
```

[ ]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	\
0	0.0	18393.0	male	168.0	62.0	110.0	80.0	1.0	1.0	
1	1.0	20228.0	female	156.0	85.0	140.0	90.0	3.0	1.0	
2	2.0	18857.0	female	165.0	64.0	130.0	70.0	3.0	1.0	
3	3.0	17623.0	male	169.0	82.0	150.0	100.0	1.0	1.0	
4	4.0	17474.0	female	156.0	56.0	100.0	60.0	1.0	1.0	

	smoke	alco	active	cardio
0	No	0.0	1.0	0.0
1	No	0.0	1.0	1.0
2	No	0.0	0.0	1.0
3	No	0.0	1.0	1.0
4	No	0.0	0.0	0.0

Displaying Data Information and Checking NAN

To display the content of the data and type of features use the info() method in pandas. This method provides a concise summary of the DataFrame's structure, including column names, data types, and the count of non-null values. It's valuable for getting a quick overview of the dataset but doesn't directly reveal missing values.

[ ]:
```
df_cardio.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   id           69997 non-null  float64
 1   age          69997 non-null  float64
 2   gender       69008 non-null  object
 3   height       68996 non-null  float64
 4   weight       69993 non-null  float64
 5   ap_hi        69992 non-null  float64
 6   ap_lo        69991 non-null  float64
 7   cholesterol  69398 non-null  float64
 8   gluc         69995 non-null  float64
 9   smoke        69003 non-null  object
 10  alco         69997 non-null  float64
```

```
 11  active       69997 non-null  float64
 12  cardio       69997 non-null  float64
dtypes: float64(11), object(2)
memory usage: 6.9+ MB
```

Here the DataFrame consists of 70000 rows with 12 variables (features). Ten features are numerical and two features are objects (gender, smoke). We notice that for some of the features the number of non-null values does not equal 70000 which means that some feature values in the data are missing. We can get the exact number of missing values for each feature using the isnull() method. The isnull(), when applied to a DataFrame, returns a DataFrame of the same shape with True in places where values are missing (NaN) and False where values are present. You can then use methods like sum() to count missing values in each column.

[ ]:
```
df_cardio.isnull().sum()
```

[ ]:
```
id               3
age              3
gender         992
height        1004
weight           7
ap_hi            8
ap_lo            9
cholesterol    602
gluc             5
smoke          997
alco             3
active           3
cardio           3
dtype: int64
```

You may also obtain the number and percentage of patients' records that has one or more missing values using the **isnull().any()** method in pandas. This method is used to check if there are any missing values (NaN or None) in each column of a DataFrame. It returns a Series that indicates whether any missing values are present in each column.

[ ]:
```
print(df_cardio.isnull().any(axis=1).sum())
print(100*df_cardio.isnull().any(axis=1).sum()/df_cardio.shape[0],'%')
```

```
3530
5.042857142857143 %
```

Here are the records with missing values.

[ ]:
```
df_cardio[df_cardio.isnull().any(axis=1)]
```

[ ]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	\
8	13.0	17668.0	female	NaN	71.0	110.0	NaN	1.0	
11	16.0	18815.0	male	173.0	NaN	120.0	80.0	1.0	
14	23.0	14532.0	male	181.0	95.0	130.0	90.0	1.0	
21	31.0	21413.0	female	157.0	69.0	NaN	80.0	1.0	

```
22        32.0  23046.0  female     NaN    90.0  145.0   85.0          2.0
...        ...      ...     ...     ...     ...    ...    ...          ...
69919  99871.0  17312.0  female   159.0    45.0  110.0   70.0          NaN
69928  99890.0  14420.0  female     NaN    55.0  140.0   90.0          1.0
69962  99949.0  21151.0  female   178.0    69.0  130.0   90.0          1.0
69974  99962.0  18226.0  female     NaN    75.0  120.0   80.0          1.0
69995  99993.0  19240.0    male   168.0    76.0  120.0   80.0          NaN

       gluc smoke  alco  active  cardio
8       1.0    No   0.0     1.0     0.0
11      1.0    No   0.0     1.0     0.0
14      1.0   NaN   1.0     1.0     0.0
21      1.0    No   0.0     1.0     0.0
22      2.0    No   0.0     1.0     1.0
...     ...   ...   ...     ...     ...
69919   2.0    No   0.0     1.0     0.0
69928   1.0    No   0.0     1.0     0.0
69962   1.0   NaN   0.0     1.0     1.0
69974   1.0    No   0.0     1.0     0.0
69995   1.0   Yes   0.0     1.0     0.0

[3530 rows x 13 columns]
```

###2.3.1 Handling Missing Data Through Dropping

To identify records in which all features and the target value are missing (empty records) use the **isnull().all()** method in pandas. This method is used to check if all values in a DataFrame or Series are missing (NaN or None).

[ ]:
```
print(f"Number of empty records = {df_cardio.isnull().all(axis=1).sum()}")
df_cardio[df_cardio.isnull().all(axis=1)]
```

```
Number of empty records = 3
```

[ ]:
```
        id  age gender  height  weight  ap_hi  ap_lo  cholesterol  gluc smoke  \
383    NaN  NaN    NaN     NaN     NaN    NaN    NaN          NaN   NaN   NaN
437    NaN  NaN    NaN     NaN     NaN    NaN    NaN          NaN   NaN   NaN
22739  NaN  NaN    NaN     NaN     NaN    NaN    NaN          NaN   NaN   NaN

       alco  active  cardio
383     NaN     NaN     NaN
437     NaN     NaN     NaN
22739   NaN     NaN     NaN
```

As the number of these records is very small compared to the size of the dataset, we will drop them. To drop these empty records use the **dropna(how='all')** in pandas. This method is used to remove rows from a DataFrame where all values in those rows are missing (NaN or None). It essentially drops rows that are completely empty. The **inplace=True** parameter means that the operation will modify the original DataFrame directly.

```
[ ]: df_cardio.dropna(how='all', inplace=True)
     print(df_cardio.isnull().sum())
```

```
id                0
age               0
gender          989
height         1001
weight            4
ap_hi             5
ap_lo             6
cholesterol     599
gluc              2
smoke           994
alco              0
active            0
cardio            0
dtype: int64
```

When we contrast the count of NaN features before and after the final step, it becomes apparent that the three empty records have been eliminated. Additionally, we observe that the count of missing values for the features 'weight,' 'ap_hi,' 'ap_lo,' and 'gluc' is quite small compared to the number of records. Therefore, the best decision is to remove these records from the dataset.

Task 2.13: Write a code for the removal of records containing missing values in the 'weight,' 'ap_hi,' 'ap_lo,' and 'gluc' features. Afterwards, validate that the DataFrame no longer contains any missing values in these specified features. You may use the 'dropna()' method with the 'subset' parameter for this purpose. *The dropna() method in pandas can be used with the subset parameter to drop rows that contain missing values in specific columns of a DataFrame.*

```
[ ]: #write you code here
```

```
[ ]: # This code is required to advance the experiment, irrespective of task 2.13.
     df_cardio.dropna(subset=['weight'], inplace=True)
     df_cardio.dropna(subset=['ap_hi'], inplace=True)
     df_cardio.dropna(subset=['ap_lo'], inplace=True)
     df_cardio.dropna(subset=['gluc'], inplace=True)
     print(df_cardio.isnull().sum())
```

```
id              0
age             0
gender        989
height        999
weight          0
ap_hi           0
ap_lo           0
cholesterol   599
gluc            0
smoke         994
alco            0
```

```
active          0
cardio          0
dtype: int64
```

3.1 2.3.2 Handling Missing Data Through Imputation

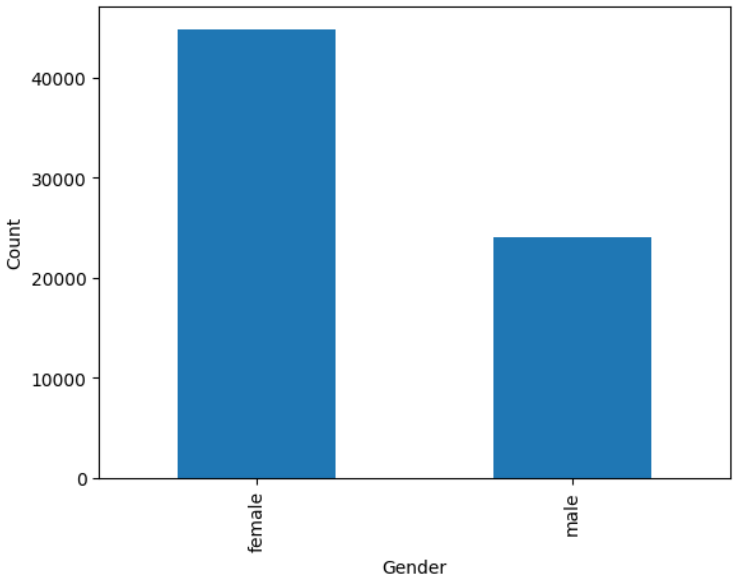
The 'gender' feature consists of 'male' and 'female' strings, but we have numerous missing values. One approach is to remove all records where the 'gender' feature is missing. However, this would entail discarding approximately 1.4% of the records, a decision that should be made by domain experts. In this section, we will opt for imputation as our method for addressing missing values in the 'gender' feature.

[ ]:
```
print(f"The number of records where gender is missing equals {df_cardio.
 ↪isnull()['gender'].sum()}")
print(f"The proportion of records where gender is missing equals {100*df_cardio.
 ↪isnull()['gender'].sum()/df_cardio.shape[0]}%")
```

```
The number of records where gender is missing equals 989
The proportion of records where gender is missing equals 1.4132407367714095%
```

One approach is to employ mode imputation, which is suitable for addressing missing categorical or nominal data. In this method, missing values are substituted with the mode, representing the most frequently occurring category or class within the variable. To provide a visual representation of the dataset's gender distribution, we will create a bar plot showing the proportions of 'male' and 'female'.

[ ]:
```
# Grouping by 'gender' and summing 'Cost'
df_cardio_gender = df_cardio.groupby('gender')['id'].count()

# Creating a bar plot
df_cardio_gender.plot(kind='bar')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()

print(df_cardio.groupby('gender')['id'].count() / df_cardio['gender'].shape[0])
```




```
gender
female    0.641388
male      0.344479
Name: id, dtype: float64
```

An alternative method involves attempting to identify a correlation between the 'gender' feature and the other features within the DataFrame. Subsequently, the missing values in the 'gender' feature can be determined based on the values of other attributes within the same record. Before calculating this correlation, we'll convert the non-numeric features into numeric ones. If you find the conversion process or the accompanying code below unclear, there's no need to worry, as we will explore this in detail in the upcoming experiment. Following the conversion, we will utilize the **corr()** method in pandas to assess the correlation."

[ ]:
```
from sklearn.preprocessing import LabelEncoder

# create a copy of the original DataFrame to mainting the original DataFrame
df_cardio_corr=df_cardio.copy()

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the categorical feature
df_cardio_corr['gender_encoded'] = label_encoder.
 ↪fit_transform(df_cardio_corr['gender'])
df_cardio_corr['smoke_encoded'] = label_encoder.
 ↪fit_transform(df_cardio_corr['smoke'])

# Drop categorical features
```

```
df_cardio_corr.drop(['gender','smoke'],axis=1,inplace=True)

df_cardio_corr.corr()
```

[ ]:

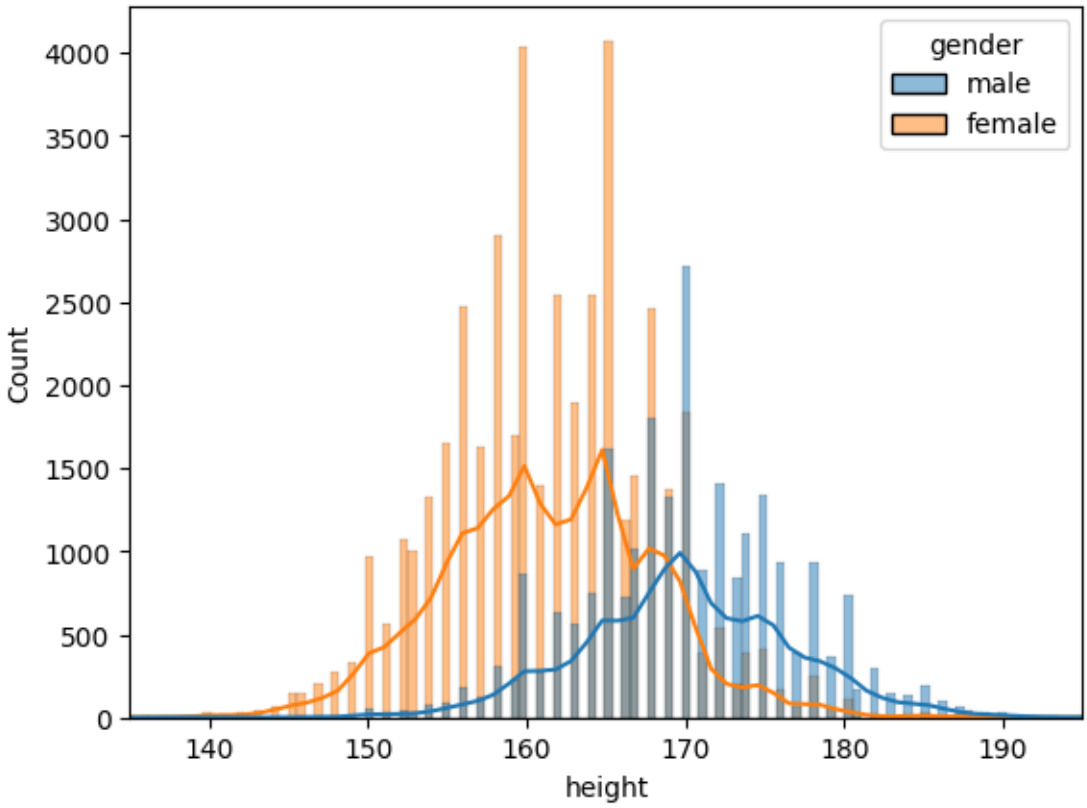
	id	age	height	weight	ap_hi	ap_lo	cholesterol	gluc	alco	active	cardio	gender_encoded	smoke_encoded
id	1.000000	0.003402	-0.002135	-0.001914	0.003349	-0.002557	0.006169	0.002481	0.001251	0.003885	0.003768	0.001744	0.000454
age	0.003402	1.000000	-0.081673	0.053721	0.020763	0.017644	0.153639	0.098723	-0.029652	-0.009914	0.238176	-0.019102	-0.038875
height	-0.002135	-0.081673	1.000000	0.289770	0.005543	0.005535	-0.050013	-0.018372	0.093818	-0.006532	-0.010975	0.457329	0.142792
weight	-0.001914	0.053721	0.289770	1.000000	0.030685	0.043703	0.141897	0.106767	0.067125	-0.016853	0.181575	0.142945	0.050784
ap_hi	0.003349	0.020763	0.005543	0.030685	1.000000	0.016084	0.023705	0.011826	0.001403	-0.000031	0.054461	0.005255	-0.001962
ap_lo	-0.002557	0.017644	0.005535	0.043703	0.016084	1.000000	0.023527	0.010808	0.010599	0.004792	0.065715	0.013032	-0.000209
cholesterol	0.006169	0.153639	-0.050013	0.141897	0.023705	0.023527	1.000000	0.451870	0.035570	0.010454	0.221130	-0.034861	0.003532
gluc	0.002481	0.098723	-0.018372	0.106767	0.011826	0.010808	0.451870	1.000000	0.011291	-0.006804	0.089344	-0.021616	-0.009287
alco	0.001251	-0.029652	0.093818	0.067125	0.001403	0.010599	0.035570	0.011291	1.000000	0.025481	-0.007406	0.155100	0.263388
active	0.003885	-0.009914	-0.006532	-0.016853	-0.000031	0.004792	0.010454	-0.006804	0.025481	1.000000	-0.035638	0.002539	0.020579
cardio	0.003768	0.238176	-0.010975	0.181575	0.054461	0.065715	0.221130	0.089344	-0.007406	-0.035638	1.000000	0.007021	-0.013119
gender_encoded	0.001744	-0.019102	0.457329	0.142945	0.005255	0.013032	-0.034861	-0.021616	0.155100	0.002539	0.007021	1.000000	0.238362
smoke_encoded	0.000454	-0.038875	0.142792	0.050784	-0.001962	-0.000209	0.003532	-0.009287	0.263388	0.020579	-0.013119	0.238362	1.000000

```
smoke_encoded          0.238362        1.000000
```

As can be observed, the 'gender' (encoded as 'gender_encoded') feature exhibits the highest correlation with the continuous 'height' feature and also has some correlation with the weight and alco features. This suggests that K-Nearest Neighbors (KNN) imputation could be used to estimate missing values for the gender feature. However, since K-Nearest Neighbors (KNN) technique have not been covered in this laboratory, we will not utilize this method. Let's attempt to examine how the 'height' feature is distributed for each gender.

[ ]:
```
#Create histograms displaying the distribution of height values based on the␣
↪gender, and incorporate the actual distribution curve.
sns.histplot(data=df_cardio, x="height", hue="gender", kde=True)
plt.xlim([135, 195])

print(df_cardio.groupby('gender')['height'].mean())

print(df_cardio.groupby('gender')['height'].std())
```

```
gender
female    161.359007
male      169.949754
Name: height, dtype: float64
gender
female    7.058470
male      7.238919
Name: height, dtype: float64
```



From the distribution, it's noticeable that male records tend to have higher heights, while female records tend to have lower heights. A possible threshold for the 'height' feature between males and females could be estimated as 167. Consequently, any missing gender value with a height greater than or equal to 167 will be assigned as male, while any height less than 167 will be assigned as female.

[ ]:
```
# Create a copy of the df_cardio DataFrame where the height feature is greater␣
 ↪or equal to 167
df_cardio_G167 = df_cardio.loc[df_cardio['height']>=167].copy()

# Create a copy of the df_cardio DataFrame where the height feature is less that␣
 ↪167
df_cardio_L167 = df_cardio.loc[df_cardio['height']<167].copy()

# Create a copy of the df_cardio DataFrame where the height feature is missing
df_cardio_NaN = df_cardio.loc[df_cardio['height'].isna()].copy()

# Fill the gender missing values in the df_cardio_G167 by male
df_cardio_G167['gender'].fillna(value='male', inplace=True)

# Fill the gender missing values in the df_cardio_L167 by female
```

```
df_cardio_L167['gender'].fillna(value='female', inplace=True)

# Reconstruct the df_cardio DataFrame by concatinaing the three sub Dataframes;␣
 ↪df_cardio_L167, df_cardio_G167, and df_cardio_NaN
df_cardio = pd.concat([df_cardio_L167, df_cardio_G167, df_cardio_NaN], axis=0)

print('The distribution of gender categories in the DataFrame following the␣
 ↪replacement of missing values.')
print(df_cardio.groupby('gender')['id'].count() / df_cardio['gender'].shape[0])

print('\n\nInformation about the DataFrame')
print(df_cardio.isnull().sum())
```

```
The distribution of gender categories in the DataFrame following the replacement
of missing values.
gender
female    0.649748
male      0.350095
Name: id, dtype: float64


Information about the DataFrame
id               0
age              0
gender          11
height         999
weight           0
ap_hi            0
ap_lo            0
cholesterol    599
gluc             0
smoke          994
alco             0
active           0
cardio           0
dtype: int64
```

Gender values remained unfilled because the corresponding height values were also missing. For these missing gender values, we will apply mode imputation such tha the missing values will be replaced with a 'female'.

[ ]:
```
#Replace gender missing values with 'female'
df_cardio['gender'].fillna(value='female', inplace=True)
```

To confirm the successful application of mode imputation and ensure that there are no remaining missing values in the 'gender' feature, execute the following code snippet

[ ]:
```
print('The distribution of gender categories in the DataFrame following the␣
 ↪replacement of missing values.')
print(df_cardio.groupby('gender')['id'].count() / df_cardio['gender'].shape[0])

print('\n\nInformation about the DataFrame')
print(df_cardio.isnull().sum())
```

```
The distribution of gender categories in the DataFrame following the replacement
of missing values.
gender
female    0.649905
male      0.350095
Name: id, dtype: float64


Information about the DataFrame
id               0
age              0
gender           0
height         999
weight           0
ap_hi            0
ap_lo            0
cholesterol    599
gluc             0
smoke          994
alco             0
active           0
cardio           0
dtype: int64
```

To handle the missing values of cholesterol feature, it's notable in the correlation matrix that the 'cholesterol' feature shows the strongest correlation with the 'gluc' feature. Both of these features are categorical, with 'cholesterol' taking values of 1.0, 2.0, and 3.0, and 'gluc' having values of 1.0, 2.0, and 3.0 as well. We will utilize the crosstab() method in pandas to investigate this correlation.

[ ]:
```
# Create a cross-tabulation
cross_tab = pd.crosstab(df_cardio['cholesterol'], df_cardio['gluc'])
cross_tab
```

[ ]:
```
gluc           1.0   2.0   3.0
cholesterol
1.0          48247  2216  1463
2.0           6663  2419   376
3.0           4042   513  3443
```

We can also obtain the marginal distributions of cholesterol for each value of the gluc feature as shown below

[ ]:
```
print(f"For gluc = 1.0, the distribution of cholesterol is {cross_tab.loc[:,1.0]/
 ↪cross_tab.loc[:,1.0].sum()*100}\n")
print(f"For gluc = 2.0, the distribution of cholesterol is {cross_tab.loc[:,2.0]/
 ↪cross_tab.loc[:,2.0].sum()*100}\n")
print(f"For gluc = 3.0, the distribution of cholesterol is {cross_tab.loc[:,3.0]/
 ↪cross_tab.loc[:,3.0].sum()*100}\n")
```

```
For gluc = 1.0, the distribution of cholesterol is cholesterol
1.0    81.841159
2.0    11.302416
3.0     6.856426
Name: 1.0, dtype: float64

For gluc = 2.0, the distribution of cholesterol is cholesterol
1.0    43.045843
2.0    46.989122
3.0     9.965035
Name: 2.0, dtype: float64

For gluc = 3.0, the distribution of cholesterol is cholesterol
1.0    27.697842
2.0     7.118516
3.0    65.183643
Name: 3.0, dtype: float64
```

[ ]:
```
# Create a copy of the df_cardio DataFrame where the gluc feature euqls 1.0
df_cardio_G1 = df_cardio.loc[df_cardio['gluc']==1.0].copy()

# Create a copy of the df_cardio DataFrame where the gluc feature euqls 2.0
df_cardio_G2 = df_cardio.loc[df_cardio['gluc']==2.0].copy()

# Create a copy of the df_cardio DataFrame where the gluc feature euqls 3.0
df_cardio_G3 = df_cardio.loc[df_cardio['gluc']==3.0].copy()

# Fill the cholesterol missing values in the df_cardio_G1 by 1.0
df_cardio_G1['cholesterol'].fillna(value=1.0, inplace=True)

# Fill the cholesterol missing values in the df_cardio_G2 by 2.0
df_cardio_G2['cholesterol'].fillna(value=2.0, inplace=True)

# Fill the cholesterol missing values in the df_cardio_G3 by 3.0
df_cardio_G3['cholesterol'].fillna(value=3.0, inplace=True)

# Reconstruct the df_cardio DataFrame by concatinaing the three sub Dataframes;␣
 ↪df_cardio_G1, df_cardio_G2, and df_cardio_G3
df_cardio = pd.concat([df_cardio_G1, df_cardio_G2, df_cardio_G3], axis=0)
```

```
print('The distribution of cholesterol categories in the DataFrame following the␣
 ↪replacement of missing values.')
print(df_cardio.groupby('cholesterol')['id'].count() / df_cardio['cholesterol'].
 ↪shape[0])

print('\n\nInformation about the DataFrame')
print(df_cardio.isnull().sum())
```

```
The distribution of cholesterol categories in the DataFrame following the
replacement of missing values.
cholesterol
1.0    0.749332
2.0    0.135708
3.0    0.114960
Name: id, dtype: float64


Information about the DataFrame
id               0
age              0
gender           0
height         999
weight           0
ap_hi            0
ap_lo            0
cholesterol      0
gluc             0
smoke          994
alco             0
active           0
cardio           0
dtype: int64
```

Task 2.14: Follow the same approach as demonstrated earlier to handle the missing values of the 'smoke' feature

[ ]:
```
#write you code here
```

Task 2.15: To handle the missing data in the height feature, follow these steps:

1- Examine the correlation between the height and gender features.
2- Create and analyze the height distribution for each gender.
3- Substituting missing height values with the median height of the corresponding gender.
4- Confirm that all missing height values have been appropriately handled.

[ ]:
```
#write you code here
```

```
[ ]: # This code is required to advance the experiment, irrespective of task 2.14 and␣
     ↪task 2.15.

     print('The distribution of smoke categories in the DataFrame.')
     print(df_cardio.groupby('smoke')['id'].count() / df_cardio['smoke'].shape[0])

     #Replace smoke missing values with 'female'
     df_cardio['smoke'].fillna(value='No', inplace=True)

     #Replace length missing values with median of the heigh values
     df_cardio['height'].fillna(value=df_cardio['height'].median(), inplace=True)


     print('\n\nInformation about the DataFrame')
     print(df_cardio.isnull().sum())
```

```
The distribution of smoke categories in the DataFrame.
smoke
No     0.898944
Yes    0.086852
Name: id, dtype: float64


Information about the DataFrame
id             0
age            0
gender         0
height         0
weight         0
ap_hi          0
ap_lo          0
cholesterol    0
gluc           0
smoke          0
alco           0
active         0
cardio         0
dtype: int64
```

#2.4. Removing Outliers

Removing outliers from a dataset is a critical data preprocessing step to improve the quality and reliability of your data analysis and modeling. Here's a general approach on how to remove outliers from a dataset:

1- **Identify the Outliers**: Visualize your data using box plots, histograms, or scatter plots to spot potential outliers. You can also use statistical methods like the z-score or the interquartile range (IQR) to detect outliers. Data points with z-scores beyond a certain threshold or lying outside the IQR are often considered outliers.

2- **Choose a Removal Strategy**: There are several strategies to handle outliers such as Deletion, Transformations, Capping or Winsorizing, Imputation, and Model-Based, and the choice depends on your specific use case and dataset.

3- **Apply the Chosen Strategy**: Implement the chosen outlier removal strategy to clean your dataset.

4- **Document and Justify**: Document all the changes you made to your dataset, including which outliers were removed and why. This documentation is crucial for transparency and reproducibility.

5- **Reevaluate**: After removing outliers, re-evaluate your data to ensure it aligns with your analysis goals and that the removal process did not introduce any unintended biases.

6- **Sensitivity Analysis**: Perform sensitivity analysis to assess how different outlier removal methods impact your results. This can help you choose the most appropriate method for your specific analysis.

Remember that the decision to remove outliers should be made carefully and with a deep understanding of your data and the problem you're trying to solve. Outliers may contain valuable information or indicate data quality issues, so it's essential to strike the right balance between data integrity and data cleaning.

Removing Outliers from Cardiovascular Disease Dataset

After resolving the missing data, our next step involves examining the cardiovascular diseases dataset for outliers. Let us have a close look at the statistical properties of the numaric features.

[ ]:
```
df_cardio.describe()
```

[ ]:

	id	age	height	weight	ap_hi	\
count	69981.000000	69981.000000	69981.000000	69981.000000	69981.000000	
mean	49979.962104	19468.939298	164.366542	74.206175	128.817822	
std	28847.603066	2467.338981	8.153997	14.395588	154.031978	
min	0.000000	10798.000000	55.000000	10.000000	-150.000000	
25%	25017.000000	17664.000000	159.000000	65.000000	120.000000	
50%	50009.000000	19703.000000	165.000000	72.000000	120.000000	
75%	74891.000000	21327.000000	170.000000	82.000000	140.000000	
max	99999.000000	23713.000000	250.000000	200.000000	16020.000000	

	ap_lo	cholesterol	gluc	alco	active	\
count	69981.000000	69981.000000	69981.000000	69981.000000	69981.000000	
mean	96.634929	1.365628	1.226419	0.053772	0.803675	
std	188.497832	0.679610	0.572238	0.225568	0.397220	
min	-70.000000	1.000000	1.000000	0.000000	0.000000	
25%	80.000000	1.000000	1.000000	0.000000	1.000000	
50%	80.000000	1.000000	1.000000	0.000000	1.000000	
75%	90.000000	2.000000	1.000000	0.000000	1.000000	
max	11000.000000	3.000000	3.000000	1.000000	1.000000	

	cardio
count	69981.000000

```
mean        0.499721
std         0.500003
min         0.000000
25%         0.000000
50%         0.000000
75%         1.000000
max         1.000000
```

Typically, the 'id' feature does not exhibit outliers. Therefore, our focus shifts to examining the 'age' feature. As indicated in the dataset description, age is provided in days. To facilitate a more intuitive interpretation, we'll convert both the minimum and maximum age records from days to years. At this stage, we aim to leverage domain expertise to identify any conspicuous outliers, such as erroneous age values

[ ]:
```
print(f"Minimum Age in Years ~ {(df_cardio['age'].min() / 365)}")
print(f"Maximum Age in Years ~ {(df_cardio['age'].max() / 365)}")
print(f"Mean Age in Years ~ {(df_cardio['age'].mean() / 365)}")
```

```
Minimum Age in Years ~ 29.583561643835615
Maximum Age in Years ~ 64.96712328767123
Mean Age in Years ~ 53.33955972002576
```

The minimum age in the datset is about 30 years, the maximum is about 65 years, and the average is about 53.33 years. Next, let us examine the 'height' feature, the minimum height is 55cm which is too short for the records of persons with a minimum age of 30. Similarly, the maximum height is 250cms which is too rare value for a person's height. So there must be an error in the height feature. Let us use the Interquartile Range and Boxplots to detect and identify outliers.

##2.4.1 Detecting Outliers by Using Interquartile Range and Boxplots

As explained earlier, the boxplot is a graphical representation of the distribution of a dataset that can help you detect outliers. It provides a visual summary of the data's central tendency, spread, and potential outliers. Here's how you can use a boxplot to detect outliers:

1- **Understand the Components of a Boxplot**: - A boxplot consists of several components: a box, whiskers, and individual data points. - The box represents the interquartile range (IQR), which contains the middle 50% of the data. - The line inside the box represents the median (50th percentile). - The whiskers extend from the box and show the range within which most of the data falls. - Individual data points beyond the whiskers are considered potential outliers.

2- **Create or Generate the Boxplot**: - Use data visualization software or programming languages like Python, R, Excel, or specialized statistical software to create the boxplot. - Input your dataset into the chosen tool and generate the boxplot for the variable of interest.

3- **Interpret the Boxplot**: - Examine the box: The height of the box represents the IQR. The taller the box, the larger the IQR. - Look at the whiskers: They extend from the box to the minimum and maximum values within a defined range. Outliers are usually identified based on this range. - The lower whisker typically extends to the smallest data point that is greater than or equal to Q1 - 1.5 * IQR (where Q1 is the first quartile). - The upper whisker typically extends to the largest data point that is less than or equal to Q3 + 1.5 * IQR (where Q3 is the third quartile). - Identify potential outliers: Individual data points that fall outside the whiskers can be potential outliers.

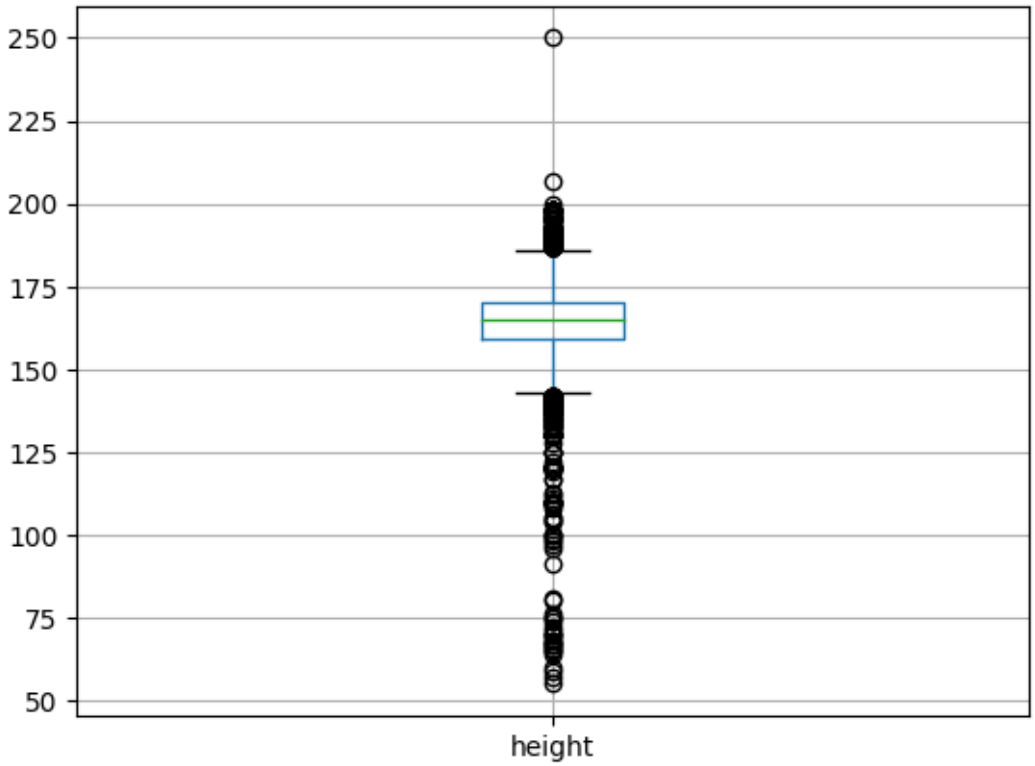
4- **Set a Threshold for Outliers**: - Decide on a threshold or rule to determine outliers. The most common method is the "1.5 * IQR rule": - Calculate the IQR (IQR = Q3 - Q1). - Calculate the lower bound: Q1 - 1.5 * IQR. - Calculate the upper bound: Q3 + 1.5 * IQR. - Any data point below the lower bound or above the upper bound can be considered a potential outlier.

5- **Identify and Label Outliers**: - On your boxplot, mark or label the potential outliers that fall beyond the defined threshold. - Some software packages automatically label outliers for you.

To generate a boxplot for the heigth feature, we will use the **boxplot()** method in pandas.

[ ]:
```
df_cardio.boxplot(column=['height'])
```

[ ]: <Axes: >



It can be observed from the plot that several height values fall beyond the whisker limits. We need to calculate the lower bound (Q1 - 1.5 * IQR) and upper bound (Q3 + 1.5 * IQR).

[ ]:
```
# Compute percentiles using Pandas quantile() function
percentile_25 = df_cardio['height'].quantile(0.25)
percentile_50 = df_cardio['height'].quantile(0.5)
percentile_75 = df_cardio['height'].quantile(0.75)

print("25th Percentile:", percentile_25)
print("50th Percentile:", percentile_50)
```

```
print("75th Percentile:", percentile_75)

# Compute interquartile range (IQR)
iqr = percentile_75 - percentile_25
print("Interquartile Range (IQR):", iqr)

LowerBound_Height = percentile_25 - 1.5*iqr
UpperBound_Height = percentile_75 + 1.5*iqr
print(f"Lower Bound = {LowerBound_Height}, and Upper Bound =␣
↪{UpperBound_Height}")
```

```
25th Percentile: 159.0
50th Percentile: 165.0
75th Percentile: 170.0
Interquartile Range (IQR): 11.0
Lower Bound = 142.5, and Upper Bound = 186.5
```

[ ]:
```
NumRecordsBefore=df_cardio.shape[0]
DroppedRecords=df_cardio[(df_cardio['height'] < LowerBound_Height) |␣
↪(df_cardio['height'] > UpperBound_Height)].shape[0]
print(f"Number of outliers based on the Interquartile Range and Boxplots is␣
↪{DroppedRecords} ({100*DroppedRecords/NumRecordsBefore}%)")
```

```
Number of outliers based on the Interquartile Range and Boxplots is 510
(0.7287692373644274%)
```

##2.4.2 Statistical Outlier Detection Using Z-Score

Z-Score is a statistical measure used to identify and potentially remove outliers from a dataset. By calculating the Z-Score for each data point, you can determine how far away each point is from the mean of the dataset in terms of standard deviations. Typically, data points with Z-Scores that exceed a certain threshold are considered outliers and can be removed. Here's how to use the Z-Score method to remove outliers:

1- **Calculate the Z-Score for Each Data Point**: For each data point in the dataset, calculate the Z-Score using the formula:

$Z_{Score} = (X - \mu)/\sigma$

Where:

- X is the data point you want to standardize.
- μ (mu) is the mean (average) of the dataset.
- σ (sigma) is the standard deviation of the dataset.

2- **Set a Threshold for Outliers**: Determine a Z-Score threshold beyond which data points are considered outliers. Commonly used thresholds are ± 2, ± 2.5, or ± 3 standard deviations from the mean. The choice of threshold depends on the desired level of sensitivity to outliers.

3- **Identify Outliers**: - Data points with Z-Scores that exceed the chosen threshold are considered outliers. - If the Z-Score is greater than the threshold (in absolute value), it's an outlier.

The choice of threshold of outliers should be done carefully, considering the specific characteristics of your data and the goals of your analysis.

let us examine the 'height' feature for any outliers again but using the z-score method.

[ ]:
```
mean=df_cardio['height'].mean()
std=df_cardio['height'].std()
print(f"For the height feature, the mean = {mean} and standard deviaton = {std}␣
↪")

LowerZScore_Height = mean - 2.5*std
UpperZScore_Height = mean + 2.5*std
print(f"Lower Z-Score = {LowerBound_Height}, and Upper Z-Score =␣
↪{UpperBound_Height}")
```

```
For the height feature, the mean = 164.36654234720854 and standard deviaton =
8.153996895089511
Lower Z-Score = 142.5, and Upper Z-Score = 186.5
```

In our case, we notice that the limits (thresholds) are the same when applying either the IQ-range detection method or the Z-Score detection method. Nonetheless, it's important to acknowledge that this consistency may not always hold true. Ultimately, the choice of the most suitable technique depends on your data's context and the specific objectives you aim to achieve.

##2.4.3 Handling Detected Outliers

There are several strategies to handle detected outliers, and the choice depends on your specific use case and dataset. Here are some common approaches:

- **Deletion**: Simply remove the outliers from your dataset. This can be done by excluding the rows containing outliers. However, this approach can lead to a loss of information.
- **Transformations**: Apply mathematical transformations to your data to make it less sensitive to outliers. Common transformations include taking the logarithm, square root, or cube root of the data.
- **Capping or Winsorizing**: Cap the extreme values by setting a threshold beyond which values are considered as outliers. You can replace these extreme values with the threshold value itself or with the nearest non-outlying data point.
- **Imputation**: Instead of removing outliers, you can replace them with more reasonable values. This could be the mean, median, or a value predicted from a regression model. Imputation is often preferred when you don't want to lose too much data.
- **Model-Based**: If you're working with predictive modeling, you can use robust models that are less sensitive to outliers, such as support vector machines, random forests, or robust regression techniques.

Remember that the interpretation of outliers and the choice of handling them should be done carefully, considering the context of your data and the specific goals of your analysis. Not all outliers are errors, and they can sometimes provide valuable insights into your dataset.

Handling Detected Outliers Through Deletion For the height feature, we will apply the IQ-range detection method. To determine the number of outliers, execute the following code snippet.

```
[ ]: NumRecordsBefore=df_cardio.shape[0]
df_cardio = df_cardio[(df_cardio['height'] >= LowerBound_Height) &␣
 ↪(df_cardio['height'] <= UpperBound_Height)]
NumRecordsAfter=df_cardio.shape[0]
DroppedRecords=NumRecordsBefore-NumRecordsAfter
print(f"Number of detected outliers is {DroppedRecords} ({100*DroppedRecords/
 ↪NumRecordsBefore}%)")
```

```
Number of detected outliers is 510 (0.7287692373644274%)
```

As the proportion of height outliers is small ($< 0.8\%$), we will exclude (delete) all records where the height falls below or exceeds the specified lower and upper thresholds, respectively.

```
[ ]: df_cardio = df_cardio[(df_cardio['height'] >= LowerBound_Height) &␣
 ↪(df_cardio['height'] <= UpperBound_Height)]
```

Task 2.16: Apply the same approach to detect and handle outliers in the 'weight,' 'ap_hi,' and 'ap_lo' features. Depending on the number of outliers detected for each feature, consider employing different methods to handle them. It's worth noting that the minimum recorded weight is 10 kg, which seems unrealistically low for individuals with a minimum age of 30. This observation raises concerns about the 'weight' feature. Additionally, similar concerns apply to the systolic pressure 'ap_hi' and the diastolic pressure 'ap_lo' features, especially considering that blood pressure values cannot be negative.

```
[ ]: #write you code here
```

Case Study 2.1: For this case study, your task is to apply the data cleaning techniques previously covered on a modified version of the Health Insurance Dataset obtained from Kaggle (https://www.kaggle.com/datasets/teertha/ushealthinsurancedataset). You can access the dataset in the **ENCS5141_Exp2_HealthInsurance.csv** file. This file is available in the GitHub repository hosted at https://github.com/mkjubran/ENCS5141Datasets.

```
[ ]: #write you code here
```