# Section A

## Question 1

We are going to load and explore the dataset. First, we start by loading staff_dataset.csv into a pandas DataFrame to explore its structure and contents, and determine the total number of attributes (columns) present to understand the variety of data available.

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Import necessary library to ignore all futurewarning dialogs
import warnings
warnings.filterwarnings("ignore")
```

```
# Load the dataset and create a dataframe object "df"
df = pd.read_csv("staff_dataset.csv")
```

```
df
```

| | Age | BusinessTravel | MonthlyIncome | JobSatisfaction | Bonus | Department | DistanceFromHome | Education | EducationField | EmployeeCount | ... | JobRole | MaritalStatus | PerformanceRating | StockOptionLevel | TrainingTimesLastYear | WorkLifeBalance | YearsAtCompany | YearsSinceLastPromotion | OverTime | Attrition |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 41 | Travel_Rarely | 5993 | 4 | 17979 | Sales | 1 | 2 | Life Sciences | 1 | ... | Sales Executive | Single | 3 | 0 | 0 | 1 | 6 | 0 | Yes | Yes |
| 1 | 49 | Travel_Frequently | 5130 | 2 | 20620 | Research & Development | 8 | 1 | Life Sciences | 1 | ... | Research Scientist | Married | 4 | 1 | 3 | 3 | 10 | 1 | No | No |
| 2 | 37 | Travel_Rarely | 2090 | 3 | 6270 | Research & Development | 2 | 2 | Other | 1 | ... | Laboratory Technician | Single | 3 | 0 | 3 | 3 | 0 | 0 | Yes | Yes |
| 3 | 33 | Travel_Frequently | 2909 | 3 | 8727 | Research & Development | 3 | 4 | Life Sciences | 1 | ... | Research Scientist | Married | 3 | 0 | 3 | 3 | 8 | 3 | Yes | No |
| 4 | 27 | Travel_Rarely | 3468 | 2 | 10404 | Research & Development | 2 | 1 | Medical | 1 | ... | Laboratory Technician | Married | 3 | 1 | 3 | 3 | 2 | 2 | No | No |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1465 | 36 | Travel_Frequently | 2571 | 4 | 7713 | Research & Development | 23 | 2 | Medical | 1 | ... | Laboratory Technician | Married | 3 | 1 | 3 | 3 | 5 | 0 | No | No |
| 1466 | 39 | Travel_Rarely | 9991 | 1 | 29973 | Research & Development | 6 | 1 | Medical | 1 | ... | Healthcare Representative | Married | 3 | 1 | 5 | 3 | 7 | 1 | No | No |
| 1467 | 27 | Travel_Rarely | 6142 | 2 | 24568 | Research & Development | 4 | 3 | Life Sciences | 1 | ... | Manufacturing Director | Married | 4 | 1 | 0 | 3 | 6 | 0 | Yes | No |
| 1468 | 49 | Travel_Frequently | 5390 | 2 | 16170 | Sales | 2 | 3 | Medical | 1 | ... | Sales Executive | Married | 3 | 0 | 3 | 2 | 9 | 0 | No | No |
| 1469 | 34 | Travel_Rarely | 4404 | 3 | 13212 | Research & Development | 8 | 3 | Medical | 1 | ... | Laboratory Technician | Married | 3 | 0 | 3 | 4 | 4 | 1 | No | No |

1470 rows × 24 columns

```
# Determine the total number of attributes
print("Total number of attributes (columns):", len(df.columns))
```

```
Total number of attributes (columns): 24
```

According to the source code implementation, the total number of attributes is 24 in this dataset.

## Question 2

In this question, we are going to assess the dataset's dimensions by identifying both the number of rows and columns, providing insight into its overall size. Below source code is executed and produces the tuple output of the dataset's dimension.

```
# Get the information of the dataset's dimensions (rows, columns)
df.shape
```

```
(1470, 24)
```

According to the output, the staff_dataset.csv dataset contains 1470 rows with 24 columns (attributes).

**Question 3**

In this question, we are going to calculate the average values for key attributes: 'Age', 'Monthly Income', and 'Years at Company' with the attached codes, rounding these figures to two decimal places to ensure precision in the analysis.

```
ave_age = np.average(df["Age"]) # Average value for "Age" attribute
ave_monthly_income = np.average(df["MonthlyIncome"]) # Average value for "Monthly Income" attribute
ave_years_at_company = np.average(df["YearsAtCompany"]) # Average value for "Years at Company" attribute

# Rounding these average values to two decimal places to ensure precision in the analysis
print("Average age:", round(ave_age, 2))
print("Average Monthly Income:", round(ave_monthly_income, 2))
print("Average Years at Company:", round(ave_years_at_company, 2))
```

```
Average age: 36.92
Average Monthly Income: 6502.93
Average Years at Company: 7.01
```

According to the output, the average value of “Age” attribute is 36.92, 6502.93 for “MonthlyIncome”, and 7.01 for “YearsAtCompany”.

**Question 4**

In this question, we are going to explore the 'Monthly Income' attribute more deeply by finding both the minimum and maximum values, which will help in understanding the income range within the dataset with the below source codes.

```
print("Maximum Monthly Income:", max(df["MonthlyIncome"])) # Maximum value in the "MonthlyIncome" attribute
print("Minimum Monthly Income:", min(df["MonthlyIncome"])) # Minimun value in the "MonthlyIncome" attribute
```

```
Maximum Monthly Income: 19999
Minimum Monthly Income: 1009
```

According to the output, minimum monthly income among the attribute is 1009, while maximum monthly income is 19999.

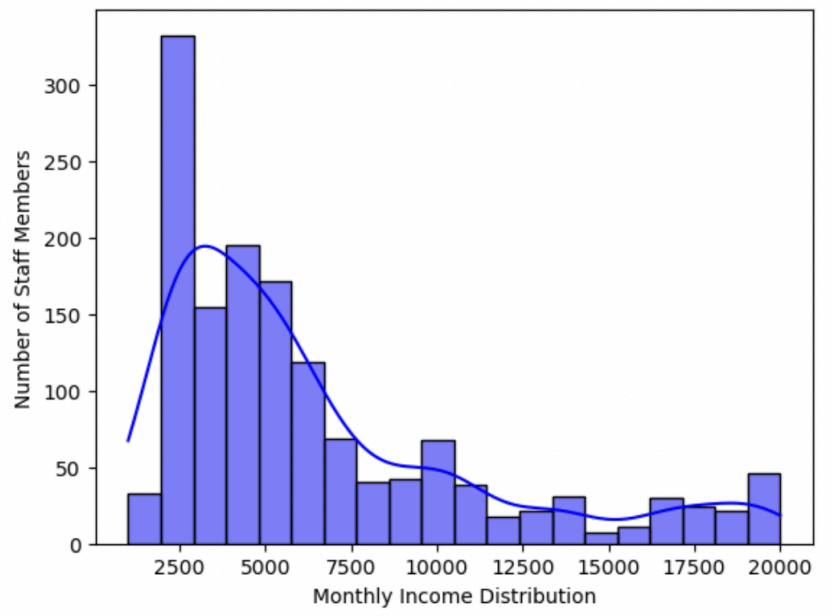
**Question 5**

In this question, we are going to visualize income distribution among the staff, create a histogram plotting 'Monthly Income' against the number of staff members, offering a clear view of how income is spread across the organization.

Below source codes are executed for the instruction.

```
# Create a histogram plotting "MonthlyIncome" against the number of staff members
sns.histplot(df["MonthlyIncome"], kde = True, color = "blue")
plt.xlabel("Monthly Income Distribution") # label x as "Monthly Income Distribution"
plt.ylabel("Number of Staff Members") # label y as "Number of Staff Members"
plt.show() # display the histogram
```

According to the result output, the right-skewed histogram plotting “MonthlyIncome” is generated, which illustrates the most frequent monthly income range is approximately the income range around 2500.



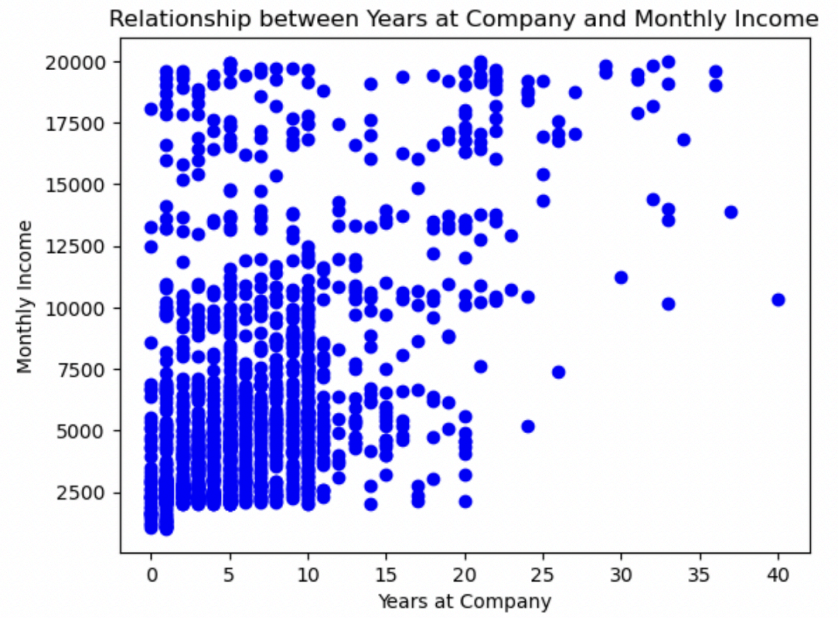

**Question 6**

In this question, we are going to create a scatter plot to examine the relationship between 'Years at Company' and 'Monthly Income', aiming to uncover any patterns or correlations between tenure and earnings.

Below source codes are executed for the instruction.

```
# Use scatter plot, "YearsAtCompany" as x, and "MonthlyIncome" as y
plt.scatter(df["YearsAtCompany"], df["MonthlyIncome"], color = "blue")
plt.xlabel("Years at Company") # label x as "Years at Company"
plt.ylabel("Monthly Income") # label y as "Monthly Income"
plt.title("Relationship between Years at Company and Monthly Income") # label the title of this scatter plot
plt.show() # display the scatter plot
```

According to the generated output, the below scatter plot is generated.




The scatter plot indicates a positive relationship between the "YearsAtCompany" and "MonthlyIncome", with a wide range of incomes at lower years of tenure and somewhat less variation at higher years. Most employees are concentrated between 0 to 10 years. It is explicitly illustrating and suggesting that as higher incomes do not significantly increase with tenure beyond a certain point. The pattern of income distribution is not rigorously linear, hinting at the influence of other factors beyond tenure that could influence monthly income levels.

**Question 7**

In this question, we are going to further analyze and discuss the dataset by calculating the correlation coefficient between 'Years at Company' and 'Monthly Income' to quantify their relationship, providing a statistical basis for understanding how closely these two variables are related.

Below source codes are executed for the instruction.

```
corr_coef = df[["MonthlyIncome", "YearsAtCompany"]].corr()
corr_coef
```

| | MonthlyIncome | YearsAtCompany |
|---|---|---|
| **MonthlyIncome** | 1.000000 | 0.514285 |
| **YearsAtCompany** | 0.514285 | 1.000000 |

According to the result output of correlations, the correlation coefficient of 0.514285 suggests a moderate positive relationship between the “YearsAtCompany” and “MonthlyIncome”.

## Question 8

- **(a)**

  We are going to discuss the range of monthly income at Company A in this question. Below codes are executed for the instruction.

```
# a. The range of monthly income at Company A
income_range = max(df["MonthlyIncome"]) - min(df["MonthlyIncome"])
print("The range of monthly income at Company A:", income_range)

max_income = max(df["MonthlyIncome"])
min_income = min(df["MonthlyIncome"])
print(f"Monthly Income ranges from {min_income} to {max_income}.")
```

```
The range of monthly income at Company A: 18990
Monthly Income ranges from 1009 to 19999.
```

  According to the output, The range of monthly income at Company A is 18990, from 1009 to 19999.

- **(b)**

  In this problem, we are going to discuss the most and least frequent monthly income values. Below codes are executed for the instruction.

```
# b. The most and least frequent monthly income values
income_freqs = df["MonthlyIncome"].value_counts()
print("The most frequent monthly income value:", income_freqs.idxmax())
print("The least frequent monthly income value:", income_freqs.idxmin())
```

```
The most frequent monthly income value: 2342
The least frequent monthly income value: 3423
```

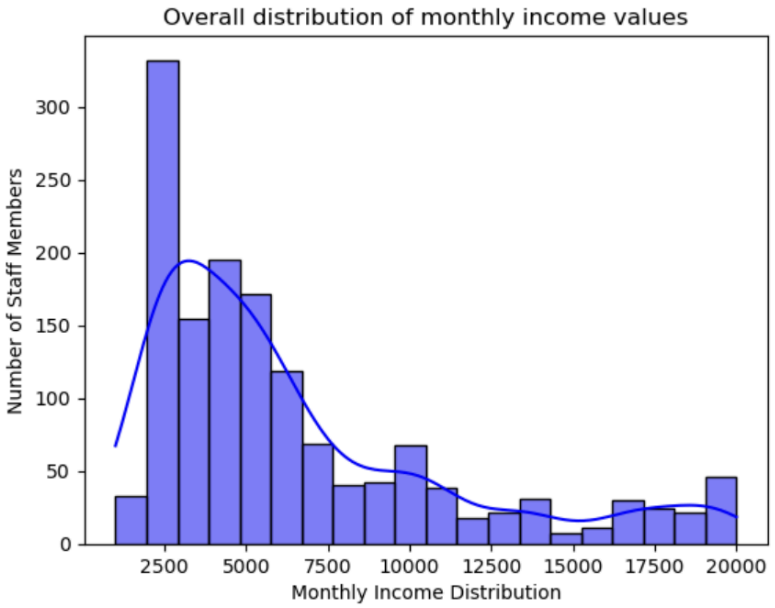
  According to the output, the most frequent monthly income value is 2342, while the least frequent monthly income value is 3423.

- **(c)**

  We are going to discuss the observations regarding the overall distribution of monthly income values. Below codes are executed for the instruction.

```
# c. the overall distribution of monthly income values
sns.histplot(df["MonthlyIncome"], kde = True, color = "blue")
plt.xlabel("Monthly Income Distribution")
plt.ylabel("Number of Staff Members")
plt.title("Overall distribution of monthly income values")
plt.show()
```

  The below histogram is generated, illustrates the overall distribution of monthly income values.




This histogram of staff members' monthly income illustrates that employees' earnings are highly concentrated between approximately 2,000 and 3,000, which represents the mode of this monthly income distribution. This distribution illustration is right-skewed, impling that only a few staff members have much higher salaries. This also indicates the existence of outliers or a few individuals with extraordinarily larger monthly incomes. However, monthly income values are widely distributed, showing considerable variation among staff members' earnings.

Due to the skewness, the mean of this income distribution is likely higher than the median, representing that the average income is pulled up by higher earners. The density curve overlaid on the histogram smooths out the distribution, confirming the skewness and variability in staff incomes. In general, this suggests a diverse income range within the organization with most employees earning at the lower end of the spectrum but with some exceptional higher earners.

- **(d)**

  In this problem, we are going to discuss whether there appears to be a linear relationship between an employee's years at the company and their monthly income, shedding light on career progression and financial growth within the organization. Below codes are executed for the instruction, generating the r-squared value of the linear regression of “YearsAtCompany” and “MonthlyIncome” to explain the relationship.

```
from scipy.stats import linregress
from sklearn.metrics import r2_score
```

```
slope, intercept, r_value, p_value, std_err = linregress(df["YearsAtCompany"], df["MonthlyIncome"])
print(f"Slope is: {slope}")
print(f"Intercept is: {intercept}", "\n")

r2 = round(r_value ** 2, 4)
print(f"R-squared value is: {r2}")
```

```
Slope is: 395.20456923368243
Intercept is: 3733.2731481323835

R-squared value is: 0.2645
```

Since the valid slope and intercept are provided, there is the linear relationship between an employee's years at the company and their monthly income. On the other hand, because the R-square value is 0.2645, we could say the relationship is extremely weak and not very trustworthy.

# Section B

## Question 1

### Import Libraries

We begin by importing necessary libraries including pandas, numpy, matplotlib and a couple of other necessary modules from scikit-learn library and so on. Below is the code executed for the importing.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix, accuracy_score, mean_squared_error, r2_score, mean_absolute_error
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LinearRegression
```

## Question 2

### Load Dataset

In this question, we are going to load the csv dataset "staff_dataset.csv" using pandas. Below source code was executed for the dataset loads.

```
df = pd.read_csv('/content/drive/MyDrive/DAML/Assignment/staff_dataset.csv')
df
```

Therefore, we have got the below output that represents the overall structure of the dataset.

**Output:**

| | Age | BusinessTravel | MonthlyIncome | JobSatisfaction | Bonus | Department | DistanceFromHome | Education | EducationField | EmployeeCount | ... | JobRole | MaritalStatus | PerformanceRating |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 41 | Travel_Rarely | 5993 | 4 | 17979 | Sales | 1 | 2 | Life Sciences | 1 | ... | Sales Executive | Single | 3 |
| 1 | 49 | Travel_Frequently | 5130 | 2 | 20520 | Research & Development | 8 | 1 | Life Sciences | 1 | ... | Research Scientist | Married | 4 |
| 2 | 37 | Travel_Rarely | 2090 | 3 | 6270 | Research & Development | 2 | 2 | Other | 1 | ... | Laboratory Technician | Single | 3 |
| 3 | 33 | Travel_Frequently | 2909 | 3 | 8727 | Research & Development | 3 | 4 | Life Sciences | 1 | ... | Research Scientist | Married | 3 |
| 4 | 27 | Travel_Rarely | 3468 | 2 | 10404 | Research & Development | 2 | 1 | Medical | 1 | ... | Laboratory Technician | Married | 3 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1465 | 36 | Travel_Frequently | 2571 | 4 | 7713 | Research & Development | 23 | 2 | Medical | 1 | ... | Laboratory Technician | Married | 3 |
| 1466 | 39 | Travel_Rarely | 9991 | 1 | 29973 | Research & Development | 6 | 1 | Medical | 1 | ... | Healthcare Representative | Married | 3 |
| 1467 | 27 | Travel_Rarely | 6142 | 2 | 24568 | Research & Development | 4 | 3 | Life Sciences | 1 | ... | Manufacturing Director | Married | 4 |
| 1468 | 49 | Travel_Frequently | 5390 | 2 | 16170 | Sales | 2 | 3 | Medical | 1 | ... | Sales Executive | Married | 3 |
| 1469 | 34 | Travel_Rarely | 4404 | 3 | 13212 | Research & Development | 8 | 3 | Medical | 1 | ... | Laboratory Technician | Married | 3 |

1470 rows × 24 columns

### Data Description

*Dataset Information*

```
#DATASET INFORMATION
print("\n DATASET INFORMATION")
df.info()
```

**Output:**

```
 DATASET INFORMATION
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 24 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   BusinessTravel            1470 non-null   object
 2   MonthlyIncome             1470 non-null   int64
 3   JobSatisfaction           1470 non-null   int64
 4   Bonus                     1470 non-null   int64
 5   Department                1470 non-null   object
 6   DistanceFromHome          1470 non-null   int64
 7   Education                 1470 non-null   int64
 8   EducationField            1470 non-null   object
 9   EmployeeCount             1470 non-null   int64
 10  EmployeeNumber            1470 non-null   int64
 11  EnvironmentSatisfaction   1470 non-null   int64
 12  Gender                    1470 non-null   object
 13  JobLevel                  1470 non-null   int64
 14  JobRole                   1470 non-null   object
 15  MaritalStatus             1470 non-null   object
 16  PerformanceRating         1470 non-null   int64
 17  StockOptionLevel          1470 non-null   int64
 18  TrainingTimesLastYear     1470 non-null   int64
 19  WorkLifeBalance           1470 non-null   int64
 20  YearsAtCompany            1470 non-null   int64
 21  YearsSinceLastPromotion   1470 non-null   int64
 22  OverTime                  1470 non-null   object
 23  Attrition                 1470 non-null   object
dtypes: int64(16), object(8)
```

Based on the output above, it is shown that all of the variables have two datatypes which are integer and object which suits the value type of the variables, thus it is not needed to change the datatypes.

*Statistical Summary*

```
[10] #STASTISTICAL SUMMARY
     print("\n STATISTICAL SUMMARY")
     df.describe()
```

**Output:**

STATISTICAL SUMMARY

| | Age | BusinessTravel | MonthlyIncome | JobSatisfaction | Bonus | DistanceFromHome | Education | EmployeeCount | EmployeeNumber | EnvironmentSatisfaction | JobLevel | PerformanceRating |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 1470.000000 | 1470.000000 | 1470.000000 | 1470.000000 | 1470.000000 | 1470.000000 | 1470.000000 | 1470.0 | 1470.000000 | 1470.000000 | 1470.000000 | 1470.000000 |
| mean | 36.923810 | 1.607483 | 6502.931293 | 2.728571 | 20479.501361 | 9.192517 | 2.912925 | 1.0 | 1024.865306 | 2.721769 | 2.063946 | 3.153741 |
| std | 9.135373 | 0.665455 | 4707.956783 | 1.102846 | 15056.272964 | 8.106864 | 1.024165 | 0.0 | 602.024335 | 1.093082 | 1.106940 | 0.360824 |
| min | 18.000000 | 0.000000 | 1009.000000 | 1.000000 | 3027.000000 | 1.000000 | 1.000000 | 1.0 | 1.000000 | 1.000000 | 1.000000 | 3.000000 |
| 25% | 30.000000 | 1.000000 | 2911.000000 | 2.000000 | 9333.750000 | 2.000000 | 2.000000 | 1.0 | 491.250000 | 2.000000 | 1.000000 | 3.000000 |
| 50% | 36.000000 | 2.000000 | 4919.000000 | 3.000000 | 15484.500000 | 7.000000 | 3.000000 | 1.0 | 1020.500000 | 3.000000 | 2.000000 | 3.000000 |
| 75% | 43.000000 | 2.000000 | 8379.000000 | 4.000000 | 26103.750000 | 14.000000 | 4.000000 | 1.0 | 1555.750000 | 4.000000 | 3.000000 | 3.000000 |
| max | 60.000000 | 2.000000 | 19999.000000 | 4.000000 | 79892.000000 | 29.000000 | 5.000000 | 1.0 | 2068.000000 | 4.000000 | 5.000000 | 4.000000 |

| TrainingTimesLastYear | WorkLifeBalance | YearsAtCompany | YearsSinceLastPromotion | Attrition |
|---|---|---|---|---|
| 1470.000000 | 1470.000000 | 1470.000000 | 1470.000000 | 1470.000000 |
| 2.799320 | 2.761224 | 7.008163 | 2.187755 | 0.161224 |
| 1.289271 | 0.706476 | 6.126525 | 3.222430 | 0.367863 |
| 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 |
| 2.000000 | 2.000000 | 3.000000 | 0.000000 | 0.000000 |
| 3.000000 | 3.000000 | 5.000000 | 1.000000 | 0.000000 |
| 3.000000 | 3.000000 | 9.000000 | 3.000000 | 0.000000 |
| 6.000000 | 4.000000 | 40.000000 | 15.000000 | 1.000000 |

Based on the output, we could observe the details of the dataset:

- count: the total number of non-empty rows in a feature.
- mean: the average or mean value of each feature.
- std: value of standard deviation of each feature.
- min: minimum value.
- max: maximum value.
- 25%, 50%, 75%: percentile/quartile of each feature.

Based on all of this information, it could be use in some cases, for example: if there are missing values in a specific feature, that missing value can be replaced with the mean value.

*Dataset Shape*

```
[11] #DATASET SHAPE
     print("\n DATASET SHAPE")
     df.shape
```

**Output:**

```
 DATASET SHAPE
(1470, 24)
```

Based on the output, it is shown that the dataset consists of 1470 observation units, and 9 variables.

**Data Cleaning**

*Dropping Duplicate Values*

```
[12] #DATA CLEANING
     #DROPPING DUPLICATED VALUES
     df=df.drop_duplicates()
     df
```

**Output:**

| | Age | BusinessTravel | MonthlyIncome | JobSatisfaction | Bonus | Department | DistanceFromHome | Education | EducationField | EmployeeCount | ... | JobRole | MaritalStatus | PerformanceRating |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 41 | Travel_Rarely | 5993 | 4 | 17979 | Sales | 1 | 2 | Life Sciences | 1 | ... | Sales Executive | Single | 3 |
| 1 | 49 | Travel_Frequently | 5130 | 2 | 20520 | Research & Development | 8 | 1 | Life Sciences | 1 | ... | Research Scientist | Married | 4 |
| 2 | 37 | Travel_Rarely | 2090 | 3 | 6270 | Research & Development | 2 | 2 | Other | 1 | ... | Laboratory Technician | Single | 3 |
| 3 | 33 | Travel_Frequently | 2909 | 3 | 8727 | Research & Development | 3 | 4 | Life Sciences | 1 | ... | Research Scientist | Married | 3 |
| 4 | 27 | Travel_Rarely | 3468 | 2 | 10404 | Research & Development | 2 | 1 | Medical | 1 | ... | Laboratory Technician | Married | 3 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1465 | 36 | Travel_Frequently | 2571 | 4 | 7713 | Research & Development | 23 | 2 | Medical | 1 | ... | Laboratory Technician | Married | 3 |
| 1466 | 39 | Travel_Rarely | 9991 | 1 | 29973 | Research & Development | 6 | 1 | Medical | 1 | ... | Healthcare Representative | Married | 3 |
| 1467 | 27 | Travel_Rarely | 6142 | 2 | 24568 | Research & Development | 4 | 3 | Life Sciences | 1 | ... | Manufacturing Director | Married | 4 |
| 1468 | 49 | Travel_Frequently | 5390 | 2 | 16170 | Sales | 2 | 3 | Medical | 1 | ... | Sales Executive | Married | 3 |
| 1469 | 34 | Travel_Rarely | 4404 | 3 | 13212 | Research & Development | 8 | 3 | Medical | 1 | ... | Laboratory Technician | Married | 3 |

1470 rows × 24 columns

Based on the output, as there is no duplicate values, there is no dropped values

*Checking Null Values*

```
#CHECKING FOR MISSING VALUES
df.isnull().sum()
```

**Output:**

```
Age                         0
BusinessTravel              0
MonthlyIncome               0
JobSatisfaction             0
Bonus                       0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EmployeeCount               0
EmployeeNumber              0
EnvironmentSatisfaction     0
Gender                      0
JobLevel                    0
JobRole                     0
MaritalStatus               0
PerformanceRating           0
StockOptionLevel            0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsSinceLastPromotion     0
OverTime                    0
Attrition                   0
dtype: int64
```

Based on the output, there is no null values

**Question 3**

**<u>Assign Independent and Dependent Variables</u>**

```
# INDEPENDENT VARIABLES
print("INDEPENDENT VARIABLES")
X = df[['Age','BusinessTravel', 'MonthlyIncome', 'JobSatisfaction']].astype(int)
print(X)

# DEPENDENT VARIABLES
print("\nDEPENDENT VARIABLES")
y = df[['Attrition']]
print (y)
```

**Output:**

```
INDEPENDENT VARIABLES
      Age  BusinessTravel  MonthlyIncome  JobSatisfaction
0      41               2           5993                4
1      49               1           5130                2
2      37               2           2090                3
3      33               1           2909                3
4      27               2           3468                2
...   ...             ...            ...              ...
1465   36               1           2571                4
1466   39               2           9991                1
1467   27               2           6142                2
1468   49               1           5390                2
1469   34               2           4404                3

[1470 rows x 4 columns]

DEPENDENT VARIABLES
      Attrition
0             1
1             0
2             1
3             0
4             0
...         ...
1465          0
1466          0
1467          0
1468          0
1469          0
```

The Independent and Dependent variables had been assigned to X and y

**Question 4**

**<u>Label Encoding</u>**

```
# LABEL ENCODING
encoder = preprocessing.LabelEncoder()

# Encode labels in column 'Business Travel' and 'Attrition'
df['BusinessTravel'] = encoder.fit_transform(df['BusinessTravel'])
df['Attrition'] = encoder.fit_transform(df['Attrition'])
df
```

**Output:**

| | Age | BusinessTravel | MonthlyIncome | JobSatisfaction | Bonus | Department | DistanceFromHome | Education | EducationField | EmployeeCount | ... | JobRole | MaritalStatus | PerformanceRating | S |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 41 | 2 | 5993 | 4 | 17979 | Sales | 1 | 2 | Life Sciences | 1 | ... | Sales Executive | Single | 3 | |
| 1 | 49 | 1 | 5130 | 2 | 20529 | Research & Development | 8 | 1 | Life Sciences | 1 | ... | Research Scientist | Married | 4 | |
| 2 | 37 | 2 | 2090 | 3 | 6279 | Research & Development | 2 | 2 | Other | 1 | ... | Laboratory Technician | Single | 3 | |
| 3 | 33 | 1 | 2909 | 3 | 8727 | Research & Development | 3 | 4 | Life Sciences | 1 | ... | Research Scientist | Married | 3 | |
| 4 | 27 | 2 | 3468 | 2 | 10404 | Research & Development | 2 | 1 | Medical | 1 | ... | Laboratory Technician | Married | 3 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 1465 | 36 | 1 | 2571 | 4 | 7713 | Research & Development | 23 | 2 | Medical | 1 | ... | Laboratory Technician | Married | 3 | |
| 1466 | 39 | 2 | 9991 | 1 | 29873 | Research & Development | 6 | 1 | Medical | 1 | ... | Healthcare Representative | Married | 3 | |
| 1467 | 27 | 2 | 6142 | 2 | 24568 | Research & Development | 4 | 3 | Life Sciences | 1 | ... | Manufacturing Director | Married | 4 | |
| 1468 | 49 | 1 | 5390 | 2 | 16170 | Sales | 2 | 3 | Medical | 1 | ... | Sales Executive | Married | 3 | |
| 1469 | 34 | 2 | 4404 | 3 | 13212 | Research & Development | 8 | 3 | Medical | 1 | ... | Laboratory Technician | Married | 3 | |

Based on the output, the value of *Business Travel* and *Attrition* features have changed into numerical form. For Example:

| *<u>Business Travel</u>* | *<u>Attrition</u>* |
|---|---|
| 0 – Non-Travel | 0 - No |
| 1 - Travel Frequently | 1 – Yes |
| 2 - Travel Rarely | |

**Question 5**

**<u>Split the dataset into training and test sets</u>**

```
[36] # Training Size 70% and Test Size 30%
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

**Output:**

```
X_train.shape, y_train.shape
((1029, 4), (1029, 1))

[45] X_test.shape, y_test.shape
     ((441, 4), (441, 1))
```

Training and Testing Sets have been split into 70% (Train Set) and 30% (Test Set).

**Question 6**

**<u>Data Normalization</u>**

```
# Data Normalization
scaler = StandardScaler()
X_Normalized_Test = scaler.fit_transform(X_test)
X_Normalized_Train = scaler.fit_transform(X_train)

X_Normalized_Test
X_Normalized_Train
```

**Output:**

```
array([[ 1.41369115,  0.59277912,  2.28479911,  1.1744894 ],
       [-0.09834647,  0.59277912, -0.31484463, -0.6497364 ],
       [-1.71838678,  0.59277912, -0.78686214, -0.6497364 ],
       ...,
       [-1.61038409, -0.92079337, -0.7505531 ,  1.1744894 ],
       [-0.85436528,  0.59277912, -0.50598877, -0.6497364 ],
       [ 1.41369115,  0.59277912,  2.68795465,  0.2623765 ]])
```

The independent variable has been normalized, the purpose of normalization is to scale the features into a similar range, where it will improve the performance and train the stability of the model.

**Question 7**

**<u>Define the Machine Learning Model (Naïve Bayes)</u>**

**Fit the train set into the model**

```
# ASSIGN THE MODEL
nb = GaussianNB()
nb.fit(X_Normalized_Train, y_train)
```

**Output:**

```
▾ GaussianNB
GaussianNB()
```

**Question 8**

**Model Evaluation using test set**

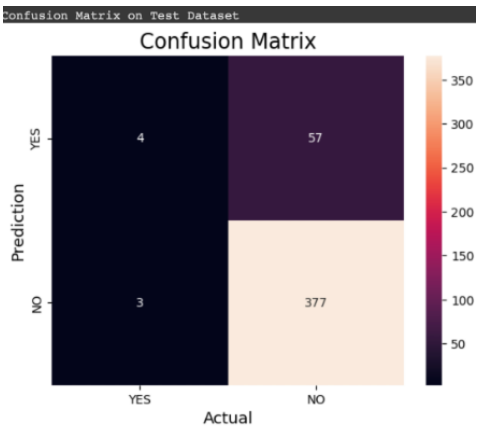
```python
# TEST SET PREDICTION
predict_test = nb.predict(X_Normalized_Test)

# TEST SET ACCURACY SCORE
accuracy_test = accuracy_score(y_test, predict_test)
print(f'Accuracy on Test Dataset:, {accuracy_test * 100:.2f}% \n')

# TEST SET CONFUSION MATRIX
matrix_test = confusion_matrix(y_test, predict_test, labels= [1,0])
print("Confusion Matrix on Test Dataset")
sns.heatmap(matrix_test,
            annot=True,
            fmt='g',
            xticklabels=['YES','NO'],
            yticklabels=['YES','NO'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```

**Output:**

Accuracy

```
Accuracy on Test Dataset:, 86.39%
```

Confusion Matrix




The accuracy of the model is 86.39%, moreover the confusion matrix above shows that there are 4 True Positive, 377 True Negative, 3 False Negative (type 1 error), and 57 False Positive (type 2 error). The higher the accuracy, true positive, and true negative, the better the prediction is.

**Question 9**

## Additional Analysis

### Naïve Bayes without Normalization

Fit the train set into the model

```
# MODEL TRAINING
nb.fit(X_train, y_train)
```

**Output:**

```
GaussianNB
GaussianNB()
```

Model Evaluation Using Test Set

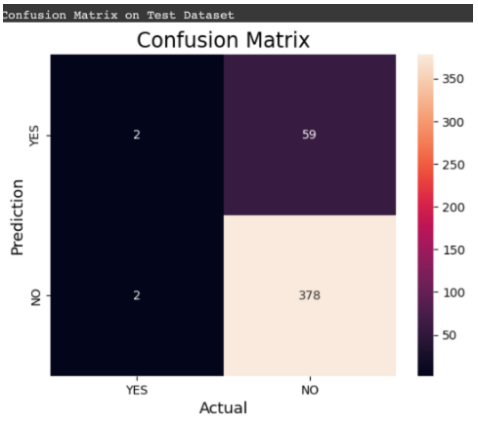
```
# MODEL EVALUATION WITH TEST SET
predict_test = nb.predict(X_test)

# TEST SET ACCURACY PREDICTION
accuracy_test = accuracy_score(y_test, predict_test)
print(f'Accuracy on Test Dataset:, {accuracy_test * 100:.2f}% \n')

# TEST SET CONFUSION MATRIX
matrix_test = confusion_matrix(y_test, predict_test, labels= [1,0])

print("Confusion Matrix on Test Dataset")
sns.heatmap(matrix_test,
            annot=True,
            fmt='g',
            xticklabels=['YES','NO'],
            yticklabels=['YES','NO'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```

**Output:**

Accuracy

```
Accuracy on Test Dataset:, 86.17%
```

Confusion Matrix




The accuracy of the model is 86.17%, moreover the confusion matrix above shows that there are 2 True Positive, 378 True Negative, 2 False Negative (type 1 error), and 59 False Positive

(type 2 error). The higher the accuracy, true positive, and true negative, the better the prediction is.

**Decision Tree Classifier**

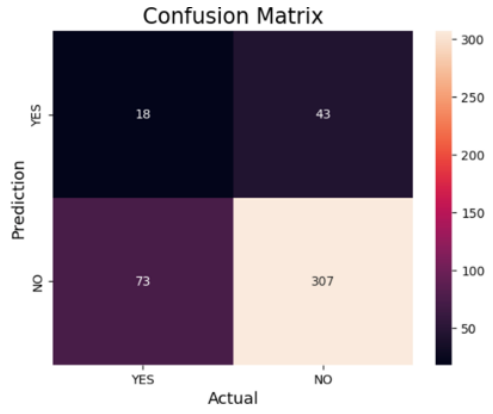
```
[25] print("DECISION TREE CLASS")

     # Creating a linear regression model
     model = DecisionTreeClassifier()

     # Training the model using the training data
     model.fit(X_Normalized_Train, y_train)

     # Making predictions on the testing data
     y_pred = model.predict(X_Normalized_Test)

     # TEST SET ACCURACY PREDICTION
     accuracy_test = accuracy_score(y_test, y_pred)
     print(f'Accuracy on Test Dataset:, {accuracy_test * 100:.2f}% \n')

     # TEST SET CONFUSION MATRIX
     matrix_test = confusion_matrix(y_test, y_pred, labels= [1,0])

     print("Confusion Matrix on Test Dataset")
     sns.heatmap(matrix_test,
                 annot=True,
                 fmt='g',
                 xticklabels=['YES','NO'],
                 yticklabels=['YES','NO'])
     plt.ylabel('Prediction',fontsize=13)
     plt.xlabel('Actual',fontsize=13)
     plt.title('Confusion Matrix',fontsize=17)
     plt.show()
```

With Normalization

Accuracy

```
DECISION TREE CLASS
Accuracy on Test Dataset:, 73.70%
```

Confusion Matrix

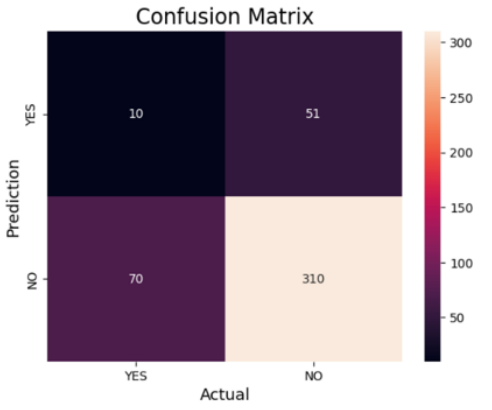



Without Normalization

Accuracy

```
DECISION TREE CLASS
Accuracy on Test Dataset:, 72.56%
```

Confusion Matrix




The accuracy of the model is 73.70%, moreover the confusion matrix above shows that there are 18 True Positive, 307 True Negative, 73 False Negative (type 1

error), and 43 False Positive (type 2 error). The higher the accuracy, true positive, and true negative, the better the prediction is.

The accuracy of the model is 72.56%, moreover the confusion matrix above shows that there are 10 True Positive, 310 True Negative, 70 False Negative (type 1 error), and 51 False Positive (type 2 error). The higher the accuracy, true positive, and true negative, the better the prediction is.

**Conclusion**

In conclusion, the process of assigned independent and dependent variables run well, as well as the data splitting and normalization. This is because based on our findings, the normalization process in the machine learning model resulted in improved in accuracy by 0.22%, which the total difference is not that much. Moreover, the confusion matrix shows positive results whereas the true positive and negative are higher than the false ones, and the accuracy of the model is 86.39%, which is considered as optimal accuracy results.

Apart from Naïve Bayes, an additional machine learning model (Decision Tree Classifier) was carried out as well. The results of the Decision Tree Classifier model have a slight improvement when the dataset was normalized, whereas the Accuracy of the normalized dataset improved by 1.14% when compared to the one without normalization.

Furthermore, the accuracy results of Naïve Bayes classifier is higher compared to decision tree classifier in both with normalization or without normalization applied, this is due to several factors such as *overfitting*, whereas decision tree tend to overfit when there are too much depth or unique data and also, *data limitation* might be another issue since Decision Tree classifier required more data to make decision effectively compared to Naïve Bayes, which it could still perfectly work even with small dataset.

## Section C

Before we introduce our solution works for this section c, we should complete importing necessary libraries for our works. Pandas, NumPy, Pyplot module in Matplotlib, etc. are required. Also, import warnings module to filter all warning console messages and ignore them, since they are not potentially affecting source code executions severely.

```
[ ]  # Import necessary libraries
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from scipy.stats import linregress

     import warnings
     warnings.filterwarnings("ignore")
```

The CSV dataset "student-mat.csv" should then be loaded, and a dataframe object should be created. There are two methods involved in this phase: the first is loading the dataset by mounting Google Drive, and the second is loading the dataset directly into the local environment.

The drive.mount() function must be used to mount Google Drive, and the drive module must be imported from the colab directory in the Google package in order to use Google Drive. Next, we need to set the file's absolute path in Google Drive to the CSV dataset.

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Load the dataset file and create into a dataframe object
df = pd.read_csv("/content/drive/MyDrive/DAML/Assignment/student-mat.csv")
```

Conversely, if we run our IPython notebook on local JupyterLab, etc., then we only need to create a dataframe object and read the file using the pd.read_csv() function. Only a relative path is needed; an absolute path is not required.

```
# Load the dataset file and create into a dataframe object
df = pd.read_csv("student-mat.csv")
```

The step is successfully executed if the below output is displayed by running “df”.

```
df
```

| | school | sex | age | address | famsize | Pstatus | Medu | Fedu | Mjob | Fjob | ... | famrel | freetime | goout | Dalc | Walc | health | absences | G1 | G2 | G3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | GP | F | 18 | U | GT3 | A | 4 | 4 | at_home | teacher | ... | 4 | 3 | 4 | 1 | 1 | 3 | 6 | 5 | 6 | 6 |
| 1 | GP | F | 17 | U | GT3 | T | 1 | 1 | at_home | other | ... | 5 | 3 | 3 | 1 | 1 | 3 | 4 | 5 | 5 | 6 |
| 2 | GP | F | 15 | U | LE3 | T | 1 | 1 | at_home | other | ... | 4 | 3 | 2 | 2 | 3 | 3 | 10 | 7 | 8 | 10 |
| 3 | GP | F | 15 | U | GT3 | T | 4 | 2 | health | services | ... | 3 | 2 | 2 | 1 | 1 | 5 | 2 | 15 | 14 | 15 |
| 4 | GP | F | 16 | U | GT3 | T | 3 | 3 | other | other | ... | 4 | 3 | 2 | 1 | 2 | 5 | 4 | 6 | 10 | 10 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 390 | MS | M | 20 | U | LE3 | A | 2 | 2 | services | services | ... | 5 | 5 | 4 | 4 | 5 | 4 | 11 | 9 | 9 | 9 |
| 391 | MS | M | 17 | U | LE3 | T | 3 | 1 | services | services | ... | 2 | 4 | 5 | 3 | 4 | 2 | 3 | 14 | 16 | 16 |
| 392 | MS | M | 21 | R | GT3 | T | 1 | 1 | other | other | ... | 5 | 5 | 3 | 3 | 3 | 3 | 3 | 10 | 8 | 7 |
| 393 | MS | M | 18 | R | LE3 | T | 3 | 2 | services | other | ... | 4 | 4 | 1 | 3 | 4 | 5 | 0 | 11 | 12 | 10 |
| 394 | MS | M | 19 | U | LE3 | T | 1 | 1 | other | at_home | ... | 3 | 2 | 3 | 3 | 3 | 5 | 5 | 8 | 9 | 9 |

395 rows × 33 columns

**Question 1**

We begin with displaying the dataset to view all columns, ensuring all attribute names are visible to interpret the scope of data available. Below source code is used to display all columns in the dataset.

```
# Print the column names
print(df.columns)
```

Below is the execution output. The result indicates names of those columns.

```
Index(['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
       'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime',
       'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery',
       'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc',
       'Walc', 'health', 'absences', 'G1', 'G2', 'G3'],
      dtype='object')
```

df.info() describes the more valuable information like count of values which are not N/A, and also the data type of each attribute.

```
# Print the name, number of non-null values, and dtype of each attribute
df.info()
```

The following output illustrates that information for each column in the df dataframe. According to the result, df contains 16 attributes as integer type and 17 attributes as object (equivalent with string) type.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 395 entries, 0 to 394
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   school      395 non-null    object
 1   sex         395 non-null    object
 2   age         395 non-null    int64
 3   address     395 non-null    object
 4   famsize     395 non-null    object
 5   Pstatus     395 non-null    object
 6   Medu        395 non-null    int64
 7   Fedu        395 non-null    int64
 8   Mjob        395 non-null    object
 9   Fjob        395 non-null    object
 10  reason      395 non-null    object
 11  guardian    395 non-null    object
 12  traveltime  395 non-null    int64
 13  studytime   395 non-null    int64
 14  failures    395 non-null    int64
 15  schoolsup   395 non-null    object
 16  famsup      395 non-null    object
 17  paid        395 non-null    object
 18  activities  395 non-null    object
 19  nursery     395 non-null    object
 20  higher      395 non-null    object
 21  internet    395 non-null    object
 22  romantic    395 non-null    object
 23  famrel      395 non-null    int64
 24  freetime    395 non-null    int64
 25  goout       395 non-null    int64
 26  Dalc        395 non-null    int64
 27  Walc        395 non-null    int64
 28  health      395 non-null    int64
 29  absences    395 non-null    int64
 30  G1          395 non-null    int64
 31  G2          395 non-null    int64
 32  G3          395 non-null    int64
dtypes: int64(16), object(17)
memory usage: 102.0+ KB
```

**Question 2**

In this phase, we explore the number of attributes contained in the dataset with the below source code.

```
# Determine the total number of attributes within the dataset
print(f"Total number of attributes within the dataset is {len(df.columns)}")
```

The result showed 33 columns are constructing the dataset.

```
Total number of attributes within the dataset is 33
```

**Question 3**

We explore the dataset's dimensions by identifying both the number of rows and columns, providing insight into the dataset's size.

```
# Assess the dataset's dimensions to identify both the number of rows and columns
print(f"The dataset has {df.shape[0]} rows with {df.shape[1]} columns")
```

395 rows with 33 columns are shaping the dataset.

```
The dataset has 395 rows with 33 columns
```

**Question 4**

In this question, we have to calculate the mean values for three attributes, "Dalc," "Walc," and "days of absences," with rounding these figures to two decimal places for precision. Below source code shows the average of those columns.

```
# Calculate the average values for "Dalc," "Walc," and "days of absences,"
# rounding these figures to two decimal places for precision
Dalc_avg = round(df["Dalc"].mean(), 2)
Walc_avg = round(df["Walc"].mean(), 2)
doa_avg = round(df["absences"].mean(), 2)

print(f"The average of Dalc is: {Dalc_avg}")
print(f"The average of Walc is: {Walc_avg}")
print(f"The average of days of absences is: {doa_avg}")
```

The average of a “Dalc” attribute is 1.48, 2.29 for a “Walc”, and 5.71 for a “days of absences”.

```
The average of Dalc is: 1.48
The average of Walc is: 2.29
The average of days of absences is: 5.71
```

**Question 5**

We would further explore the "days of absences" attribute by calculating both the minimum and maximum values of the column, to understand the range of student absence days.

```
# Finding both the minimum and maximum values of "days of absences"
doa_min = df["absences"].min()
doa_max = df["absences"].max()

print(f"The minimum value of days of absences is {doa_min}")
print(f"The maximum value of days of absences is {doa_max}")
```

According to the code execution, the minimum value of absence days is 0, while the maximum value of absence days is 75.

```
The minimum value of days of absences is 0
The maximum value of days of absences is 75
```

**Question 6**

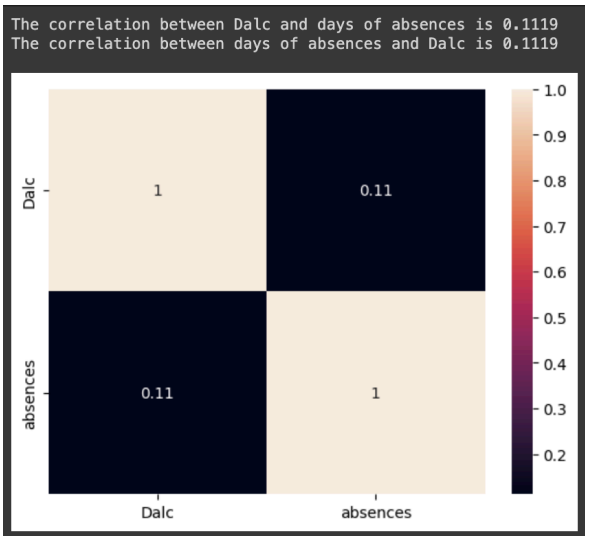
Now we are going to calculate the correlation between two attributes ("Dalc" and "absences") to quantify their relationship and visualize the correlation using a heatmap plot, to offer an obvious and concise graphical representation of the strength and direction of their relationship with visual context.

Below function calculates the Pearson correlation between two variables. We represent the correlation matrix as a heatmap which shows the degree and direction of the association between Dalc and absences. The colors used in the heatmap indicate the strength and direction of correlation. Warmer colors indicate a positive association, whereas colder colors suggest a negative correlation. The annotation for each heatmap cell displays the correlation coefficient value.

```
# Calculate the correlation between these two attributes to quantify their relationship
dtoa_corr = df["Dalc"].corr(df["absences"])
atod_corr = df["absences"].corr(df["Dalc"])

print(f"The correlation between Dalc and days of absences is {round(dtoa_corr, 4)}")
print(f"The correlation between days of absences and Dalc is {round(atod_corr, 4)}")
print()

# Visualize this correlation using a heatmap
sns.heatmap(df[["Dalc", "absences"]].corr(), annot = True)
plt.show()
```

The correlation between those two attributes is 0.1119, which implies the correlation between Dalc and absences is negligible due to its critically low relationship between each other.

```
The correlation between Dalc and days of absences is 0.1119
The correlation between days of absences and Dalc is 0.1119
```



**Question 7**

- (a)

```
# The range of days for absences observed in the dataset
doa_min = df["absences"].min()
doa_max = df["absences"].max()

print(f"The minimum value of days of absences is {doa_min}.")
print(f"The maximum value of days of absences is {doa_max}.")
print(f"Therefore, the days of absences ranges from {doa_min} to {doa_max}, gap is {doa_max - doa_min}.")
```

The range of absence days is represented below with the above source code execution.

```
The minimum value of days of absences is 0.
The maximum value of days of absences is 75.
Therefore, the days of absences ranges from 0 to 75, gap is 75.
```

The range of the days of absence observed in the dataset spans in between 0 to 75. This shows that the students have a varied range of absenteeism behavior. While some students are never absent, others have been absent for as many as 75 days which is quite high.

- (b)

```
# Identify the most and least frequent days for absences among students
doa_freqs = df["absences"].value_counts()

print(f"The most frequent days of absences is {doa_freqs.idxmax()} with frequency of {doa_freqs.max()}.")
print(f"The most frequent days of absences is {doa_freqs.idxmin()} with frequency of {doa_freqs.min()}.")
```

The most and least frequent days for absences among students to highlight common patterns or outliers, are represented below with the above source code execution.

```
The most frequent days of absences is 0 with frequency of 115.
The most frequent days of absences is 38 with frequency of 1.
```

According to the data, there are 115 cases where a student has zero absences recorded. It is important to point out that day 38 has the lowest frequency of absences, with only one recorded absence. This information provides useful insights into the patterns of student absenteeism by illustrating how absences are distributed throughout different days.

An interesting observation is that the day with the highest frequency of absences has a value of zero, which could indicate a default value in the dataset. In contrast, the least

common day with one absence could suggest an anomaly, indicating that further investigation might be necessary.

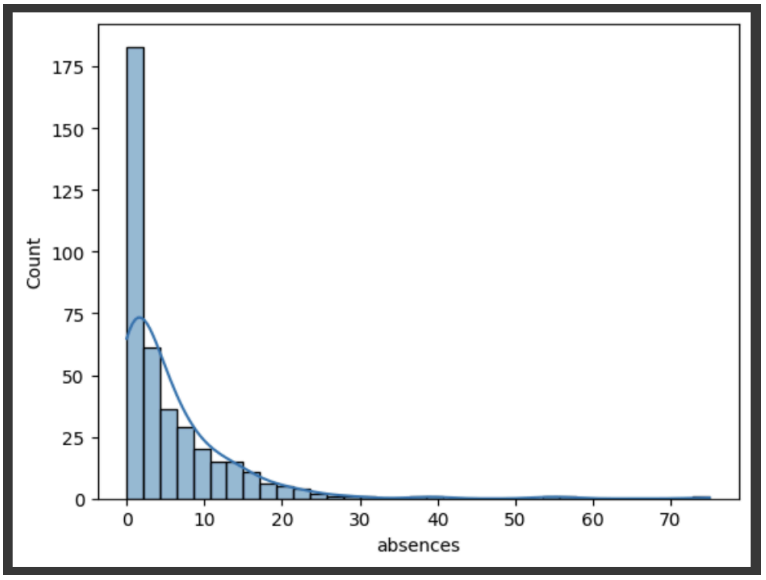
Overall, the data highlights the importance of tracking student attendance and identifying patterns of absenteeism. By understanding the trends in student absences, schools and educators can take proactive measures to address the issue and provide the necessary support to improve student attendance and academic performance.

- (c)

To make observations regarding the overall distribution of days for absences, noting any skewness, peaks, or unusual trends, we are going to generate the histogram of "absences" attribute with the source code below.

```
# Create a histogram to visualize the distribution of days for absences
sns.histplot(df["absences"], kde = True)
plt.show()
```

Below is the histogram, illustrates the distribution of "absences" attribute.




The analysis of the distribution of absences among students indicates that the distribution is right-skewed, with a higher frequency of days when students have fewer absences and a lower frequency of days when students have greater absences. Most students have a low absence rate and are clustered around the lower end of the spectrum, while a small percentage of students have a high absence rate, leading to a wider distribution of absence rates.

Notably, there was a significant peak in attendance on day 0, with a large number of students having perfect attendance. This peak suggests that students were particularly motivated to attend school on that day, perhaps due to a special event or a high-stakes test.

Occasional spikes in absences are observed in the distribution, indicating that there is a good amount of variation in students' absence rates. However, most students have a low absence rate, with only a few instances of high absences. Furthermore, there is a single absence on day 38, which stands out as an outlier in the distribution.

Overall, the analysis of the distribution of absences among students provides valuable insights into the trends and patterns of attendance in the school, which can be used to identify areas for improvement and develop strategies to support students who may be struggling with attendance.

- (d)

【Coding Works and Output for R-squared】

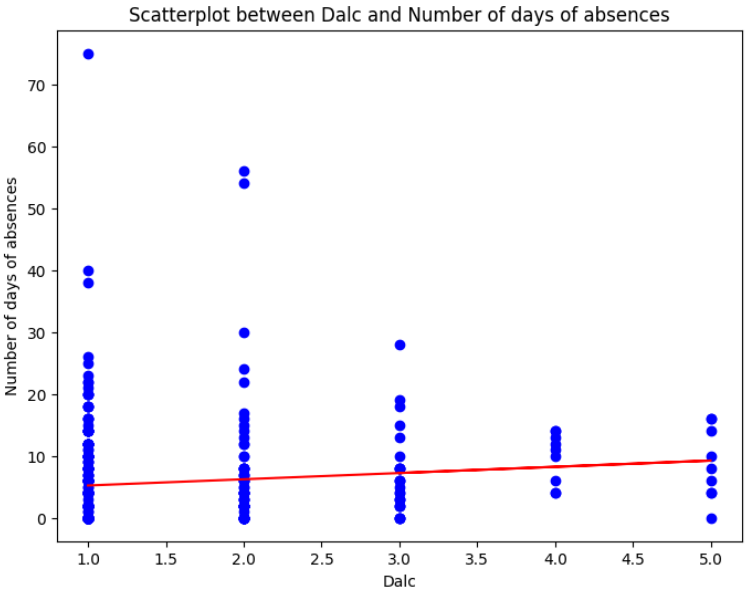
```
# Apply linear regression function from scipy into the dataset to see the relationship
slope, intercept, r_value, p_value, std_err = linregress(df["Dalc"], df["absences"])
pred = intercept + slope * df['Dalc']

# Calculate the R-squared value to observe the experiment performance
r2_score = round(r_value ** 2, 4)

print(f"R-squared evaluation score is: {r2_score}")
```

```
R-squared evaluation score is: 0.0125
```

【Coding Works and Output for the slope represents relationship between attributes】

```
# Illustration of plots to represent the relationship between "Dalc" and "absences"
plt.figure(figsize=(8, 6))
plt.scatter(df['Dalc'], df['absences'], color='blue')
plt.plot(df['Dalc'], pred, color='red')
plt.title('Scatterplot between Dalc and Number of days of absences')
plt.xlabel('Dalc')
plt.ylabel('Number of days of absences')
plt.show()
```

【Plots illustrating the linear relationship】




As per the provided graph, it is clear that there exists a correlation between 'Dalc' (daily alcohol consumption) and the number of days of 'absences.' The correlation coefficient between the two variables is 0.0125, indicating a weak correlation. This means that there is a tendency for higher alcohol consumption to be associated with increased absences, but the relationship between the two variables is not strong. It is essential to note that correlation does not imply causation. Therefore, it is necessary to investigate further to determine whether the correlation is statistically significant or not. Additionally, other factors such as personal health, family issues, or work-life balance could also impact the variables, and hence, we cannot solely attribute the observed correlation to alcohol consumption. Therefore, a more in-depth analysis is needed to establish the relationship between 'Dalc' and 'absences.'