Technological Institute of the Philippines	**Quezon City - Computer Engineering**
Course Code:	CPE 019
Code Title:	Emerging Technologies 2 in CpE
2nd Semester	AY 2023-2024

ACTIVITY NO.	**Prelim Exam**
Name	Gamundoy, Jon Aviv Cloydd
Name	Pineda, Rachel Joy
Section	CPE32S3
Date Performed:	3/02/2024
Date Submitted:	3/06/2024
Instructor:	Engr. Roman M. Richard

Data Pre-processing

Training Dataset

Student-mat.csv

```
# Import necessary libraries
import pandas as pd # For dataframes and operations
import numpy as np # For dealing with null values
import matplotlib.pyplot as plt # For plotting
import seaborn as sns # For plotting
from scipy import stats # For computing statistics

data = pd.read_csv('student-mat.csv') # Read raw csv file

print(data.info()) # Check attributes and/or missing values

data.head() # Check contents
```

```
<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
None
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4
3	GP	F	15	U	GT3	T	4	2	health	services	...	3
4	GP	F	16	U	GT3	T	3	3	other	other	...	4

5 rows × 33 columns

Data Cleaning

```
# Replace yes/no data with '0' and '1'
data["schoolsup"] = data["schoolsup"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)

data["famsup"] = data["famsup"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)

data["paid"] = data["paid"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)

data["activities"] = data["activities"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)

data["nursery"] = data["nursery"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)

data["higher"] = data["higher"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)

data["internet"] = data["internet"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)

data["romantic"] = data["romantic"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)


# Verify changes
  # Isolate columns for verification
data.loc[:, ['schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', '
```

	schoolsup	famsup	paid	activities	nursery	higher	internet	romantic
0	1	0	0	0	1	1	0	0
1	0	1	0	0	0	1	1	0
2	1	0	1	0	1	1	1	0
3	0	1	1	1	1	1	1	1
4	0	1	1	0	1	1	0	0

```
# Replace School with binary values (GP = 0 | MS = 1)
data["school"] = data["school"].apply(lambda toLabel: 0 if toLabel == 'GP' else 1)

# Replace Gender with binary values (F = 0 | M = 1)
data["sex"] = data["sex"].apply(lambda toLabel: 0 if toLabel == 'F' else 1)

# Replace Address with binary values (U = 0 | R = 1)
data["address"] = data["address"].apply(lambda toLabel: 0 if toLabel == 'U' else 1)

# Replace Family Size with binary values (LE3 = 0 | GT3 = 1)
data["famsize"] = data["famsize"].apply(lambda toLabel: 0 if toLabel == 'LE3' else 1)

# Replace Parent Cohabitation Status with binary values (T = 0 | A = 1)
data["Pstatus"] = data["Pstatus"].apply(lambda toLabel: 0 if toLabel == 'T' else 1)

# Replace Parent's Occupation with numerical values (teacher = 0 | health = 1 | services = 2
  # Identify the word to be replaced and values to replace it with
replacements = {'teacher': '0', 'health': '1', 'services': '2', 'at_home': '3', 'other': '4'

  # Use the .map() function to replace the words with numerical values
data["Mjob"] = data["Mjob"].map(replacements).fillna(data["Mjob"])

  # Repeat with Father
data["Fjob"] = data["Fjob"].map(replacements).fillna(data["Fjob"])


# Verify changes
  # Isolate columns for verification
data.loc[:, ['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob']].head(20)
```

	school	sex	address	famsize	Pstatus	Mjob	Fjob
0	0	0	0	1	1	3	0
1	0	0	0	1	0	3	4
2	0	0	0	0	0	3	4
3	0	0	0	1	0	1	2
4	0	0	0	1	0	4	4
5	0	1	0	0	0	2	4
6	0	1	0	0	0	4	4
7	0	0	0	1	1	4	0
8	0	1	0	0	1	2	4
9	0	1	0	1	0	4	4
10	0	0	0	1	0	0	1
11	0	0	0	1	0	2	4
12	0	1	0	0	0	1	2
13	0	1	0	1	0	0	4
14	0	1	0	1	1	4	4
15	0	0	0	1	0	1	4
16	0	0	0	1	0	2	2
17	0	0	0	1	0	4	4
18	0	1	0	1	0	2	2
19	0	1	0	0	0	1	4

Data Verification

```
# Data Attribute Verification
data.info()
```

```
<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    int64
 1   sex         395 non-null    int64
 2   age         395 non-null    int64
 3   address     395 non-null    int64
```

```
 4   famsize     395 non-null    int64
 5   Pstatus     395 non-null    int64
 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    int64
 16  famsup      395 non-null    int64
 17  paid        395 non-null    int64
 18  activities  395 non-null    int64
 19  nursery     395 non-null    int64
 20  higher      395 non-null    int64
 21  internet    395 non-null    int64
 22  romantic    395 non-null    int64
 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(29), object(4)
memory usage: 102.0+ KB
```

```
# Data Content Verification
data.head(50)
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	fre
0	0	0	18	0	1	1	4	4	3	0	...	4	
1	0	0	17	0	1	0	1	1	3	4	...	5	
2	0	0	15	0	0	0	1	1	3	4	...	4	
3	0	0	15	0	1	0	4	2	1	2	...	3	
4	0	0	16	0	1	0	3	3	4	4	...	4	
5	0	1	16	0	0	0	4	3	2	4	...	5	
6	0	1	16	0	0	0	2	2	4	4	...	4	
7	0	0	17	0	1	1	4	4	4	0	...	4	
8	0	1	15	0	0	1	3	2	2	4	...	4	
9	0	1	15	0	1	0	3	4	4	4	...	5	
10	0	0	15	0	1	0	4	4	0	1	...	3	
11	0	0	15	0	1	0	2	1	2	4	...	5	
12	0	1	15	0	0	0	4	4	1	2	...	4	
13	0	1	15	0	1	0	4	3	0	4	...	5	
14	0	1	15	0	1	1	2	2	4	4	...	4	
15	0	0	16	0	1	0	4	4	1	4	...	4	
16	0	0	16	0	1	0	4	4	2	2	...	3	
17	0	0	16	0	1	0	3	3	4	4	...	5	
18	0	1	17	0	1	0	3	2	2	2	...	5	
19	0	1	16	0	0	0	4	3	1	4	...	3	
20	0	1	15	0	1	0	4	3	0	4	...	4	
21	0	1	15	0	1	0	4	4	1	1	...	5	
22	0	1	16	0	0	0	4	2	0	4	...	4	
23	0	1	16	0	0	0	2	2	4	4	...	5	
24	0	0	15	1	1	0	2	4	2	1	...	4	
25	0	0	16	0	1	0	2	2	2	2	...	1	
26	0	1	15	0	1	0	2	2	4	4	...	4	
27	0	1	15	0	1	0	4	2	1	2	...	2	
28	0	1	16	0	0	1	3	4	2	4	...	5	
29	0	1	16	0	1	0	4	4	0	0	...	4	

30	0	1	15	0	1	0	4	4	1	2	...	5
31	0	1	15	0	1	0	4	4	2	2	...	4
32	0	1	15	1	1	0	4	3	0	3	...	4
33	0	1	15	0	0	0	3	3	4	4	...	5
34	0	1	16	0	1	0	3	2	4	4	...	5
35	0	0	15	0	1	0	2	3	4	4	...	3
36	0	1	15	0	0	0	4	3	0	2	...	5
37	0	1	16	1	1	1	4	4	4	0	...	2
38	0	0	15	1	1	0	3	4	2	1	...	4
39	0	0	15	1	1	0	2	2	3	4	...	4
40	0	0	16	0	0	0	2	2	4	4	...	3
41	0	1	15	0	0	0	4	4	0	4	...	5
42	0	1	15	0	1	0	4	4	2	0	...	4
43	0	1	15	0	1	0	2	2	2	2	...	5
44	0	0	16	0	0	0	2	2	4	3	...	4
45	0	0	15	0	0	1	4	3	4	4	...	5
46	0	0	16	0	0	1	3	3	4	2	...	2
47	0	1	16	0	1	0	4	3	1	2	...	4
48	0	1	15	0	1	0	4	2	0	4	...	4
49	0	0	15	0	1	0	4	4	2	0	...	4

50 rows × 33 columns

Testing Dataset

Student-test.csv

```
test = pd.read_csv('student-test.csv') # Read raw csv file

print(test.info()) # Check attributes and/or missing values

test.head() # Check contents
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 649 entries, 0 to 648
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   school      649 non-null    object
 1   sex         649 non-null    object
 2   age         649 non-null    int64 
 3   address     649 non-null    object
 4   famsize     649 non-null    object
 5   Pstatus     649 non-null    object
 6   Medu        649 non-null    int64 
 7   Fedu        649 non-null    int64 
 8   Mjob        649 non-null    object
 9   Fjob        649 non-null    object
 10  reason      649 non-null    object
 11  guardian    649 non-null    object
 12  traveltime  649 non-null    int64 
 13  studytime   649 non-null    int64 
 14  failures    649 non-null    int64 
 15  schoolsup   649 non-null    object
 16  famsup      649 non-null    object
 17  paid        649 non-null    object
 18  activities  649 non-null    object
 19  nursery     649 non-null    object
 20  higher      649 non-null    object
 21  internet    649 non-null    object
 22  romantic    649 non-null    object
 23  famrel      649 non-null    int64 
 24  freetime    649 non-null    int64 
 25  goout       649 non-null    int64 
 26  Dalc        649 non-null    int64 
 27  Walc        649 non-null    int64 
 28  health      649 non-null    int64 
 29  absences    649 non-null    int64 
 30  G1          649 non-null    int64 
 31  G2          649 non-null    int64 
 32  G3          649 non-null    int64 
dtypes: int64(16), object(17)
memory usage: 167.4+ KB
None
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4
3	GP	F	15	U	GT3	T	4	2	health	services	...	3
4	GP	F	16	U	GT3	T	3	3	other	other	...	4

5 rows × 33 columns

Data Cleaning

```
# Replace yes/no data with '0' and '1'
test["schoolsup"] = test["schoolsup"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)

test["famsup"] = test["famsup"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)

test["paid"] = test["paid"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)

test["activities"] = test["activities"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)

test["nursery"] = test["nursery"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)

test["higher"] = test["higher"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)

test["internet"] = test["internet"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)

test["romantic"] = test["romantic"].apply(lambda toLabel: 0 if toLabel == 'no' else 1)


# Verify changes
  # Isolate columns for verification
test.loc[:, ['schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', '
```

	schoolsup	famsup	paid	activities	nursery	higher	internet	romantic
0	1	0	0	0	1	1	0	0
1	0	1	0	0	0	1	1	0
2	1	0	0	0	1	1	1	0
3	0	1	0	1	1	1	1	1
4	0	1	0	0	1	1	0	0

```
# Replace School with binary values (GP = 0 | MS = 1)
test["school"] = test["school"].apply(lambda toLabel: 0 if toLabel == 'GP' else 1)

# Replace Gender with binary values (F = 0 | M = 1)
test["sex"] = test["sex"].apply(lambda toLabel: 0 if toLabel == 'F' else 1)

# Replace Address with binary values (U = 0 | R = 1)
test["address"] = test["address"].apply(lambda toLabel: 0 if toLabel == 'U' else 1)

# Replace Family Size with binary values (LE3 = 0 | GT3 = 1)
test["famsize"] = test["famsize"].apply(lambda toLabel: 0 if toLabel == 'LE3' else 1)

# Replace Parent Cohabitation Status with binary values (T = 0 | A = 1)
test["Pstatus"] = test["Pstatus"].apply(lambda toLabel: 0 if toLabel == 'T' else 1)

# Replace Parent's Occupation with numerical values (teacher = 0 | health = 1 | services = 2
  # Identify the word to be replaced and values to replace it with
replacements = {'teacher': '0', 'health': '1', 'services': '2', 'at_home': '3', 'other': '4'

  # Use the .map() function to replace the words with numerical values
test["Mjob"] = test["Mjob"].map(replacements).fillna(data["Mjob"])

  # Repeat with Father
test["Fjob"] = test["Fjob"].map(replacements).fillna(data["Fjob"])


# Verify changes
  # Isolate columns for verification
test.loc[:, ['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob']].head(20)
```

	school	sex	address	famsize	Pstatus	Mjob	Fjob
0	0	0	0	1	1	3	0
1	0	0	0	1	0	3	4
2	0	0	0	0	0	3	4
3	0	0	0	1	0	1	2
4	0	0	0	1	0	4	4
5	0	1	0	0	0	2	4
6	0	1	0	0	0	4	4
7	0	0	0	1	1	4	0
8	0	1	0	0	1	2	4
9	0	1	0	1	0	4	4
10	0	0	0	1	0	0	1
11	0	0	0	1	0	2	4
12	0	1	0	0	0	1	2
13	0	1	0	1	0	0	4
14	0	1	0	1	1	4	4
15	0	0	0	1	0	1	4
16	0	0	0	1	0	2	2
17	0	0	0	1	0	4	4
18	0	1	0	1	0	2	2
19	0	1	0	0	0	1	4

Data Verification

```
# Test Attribute Verification
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 649 entries, 0 to 648
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   school      649 non-null    int64
 1   sex         649 non-null    int64
 2   age         649 non-null    int64
 3   address     649 non-null    int64
```

```
 4   famsize     649 non-null    int64
 5   Pstatus     649 non-null    int64
 6   Medu        649 non-null    int64
 7   Fedu        649 non-null    int64
 8   Mjob        649 non-null    object
 9   Fjob        649 non-null    object
 10  reason      649 non-null    object
 11  guardian    649 non-null    object
 12  traveltime  649 non-null    int64
 13  studytime   649 non-null    int64
 14  failures    649 non-null    int64
 15  schoolsup   649 non-null    int64
 16  famsup      649 non-null    int64
 17  paid        649 non-null    int64
 18  activities  649 non-null    int64
 19  nursery     649 non-null    int64
 20  higher      649 non-null    int64
 21  internet    649 non-null    int64
 22  romantic    649 non-null    int64
 23  famrel      649 non-null    int64
 24  freetime    649 non-null    int64
 25  goout       649 non-null    int64
 26  Dalc        649 non-null    int64
 27  Walc        649 non-null    int64
 28  health      649 non-null    int64
 29  absences    649 non-null    int64
 30  G1          649 non-null    int64
 31  G2          649 non-null    int64
 32  G3          649 non-null    int64
dtypes: int64(29), object(4)
memory usage: 167.4+ KB
```

```
# Test Content Verification
test.head(50)
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	fre
0	0	0	18	0	1	1	4	4	3	0	...	4	
1	0	0	17	0	1	0	1	1	3	4	...	5	
2	0	0	15	0	0	0	1	1	3	4	...	4	
3	0	0	15	0	1	0	4	2	1	2	...	3	
4	0	0	16	0	1	0	3	3	4	4	...	4	
5	0	1	16	0	0	0	4	3	2	4	...	5	
6	0	1	16	0	0	0	2	2	4	4	...	4	
7	0	0	17	0	1	1	4	4	4	0	...	4	
8	0	1	15	0	0	1	3	2	2	4	...	4	
9	0	1	15	0	1	0	3	4	4	4	...	5	
10	0	0	15	0	1	0	4	4	0	1	...	3	
11	0	0	15	0	1	0	2	1	2	4	...	5	
12	0	1	15	0	0	0	4	4	1	2	...	4	
13	0	1	15	0	1	0	4	3	0	4	...	5	
14	0	1	15	0	1	1	2	2	4	4	...	4	
15	0	0	16	0	1	0	4	4	1	4	...	4	
16	0	0	16	0	1	0	4	4	2	2	...	3	
17	0	0	16	0	1	0	3	3	4	4	...	5	
18	0	1	17	0	1	0	3	2	2	2	...	5	
19	0	1	16	0	0	0	4	3	1	4	...	3	
20	0	1	15	0	1	0	4	3	0	4	...	4	
21	0	1	15	0	1	0	4	4	1	1	...	5	
22	0	1	16	0	0	0	4	2	0	4	...	4	
23	0	1	16	0	0	0	2	2	4	4	...	5	
24	0	0	15	1	1	0	2	4	2	1	...	4	
25	0	0	16	0	1	0	2	2	2	2	...	1	
26	0	1	15	0	1	0	2	2	4	4	...	4	
27	0	1	15	0	1	0	4	2	1	2	...	2	
28	0	1	16	0	0	1	3	4	2	4	...	5	
29	0	1	16	0	1	0	4	4	0	0	...	4	

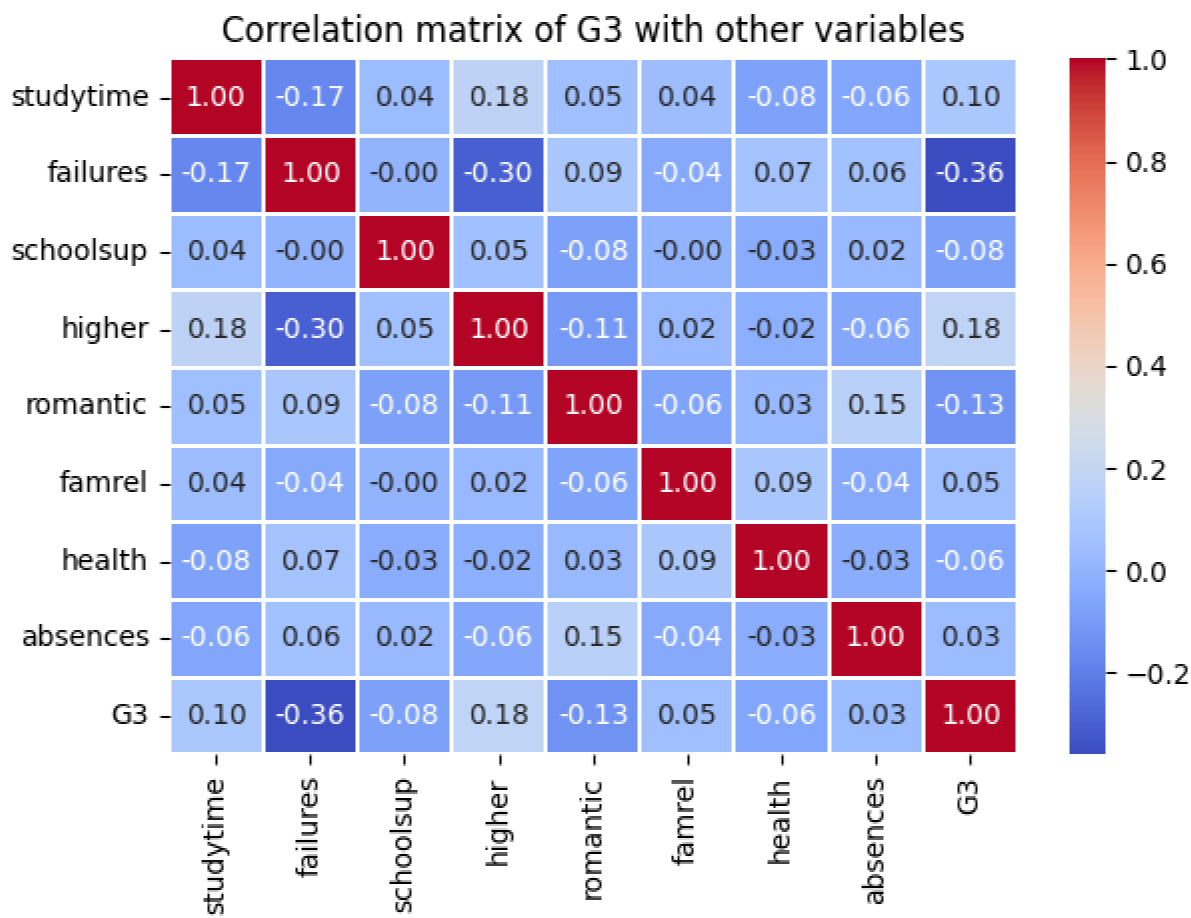
30	0	1	15	0	1	0	4	4	1	2	...	5
31	0	1	15	0	1	0	4	4	2	2	...	4
32	0	1	15	1	1	0	4	3	0	3	...	4
33	0	1	15	0	0	0	3	3	4	4	...	5
34	0	1	16	0	1	0	3	2	4	4	...	5
35	0	0	15	0	1	0	2	3	4	4	...	3
36	0	1	15	0	0	0	4	3	0	2	...	5
37	0	1	16	1	1	1	4	4	4	0	...	2
38	0	0	15	1	1	0	3	4	2	1	...	4
39	0	0	15	1	1	0	2	2	3	4	...	4
40	0	0	16	0	0	0	2	2	4	4	...	3
41	0	1	15	0	0	0	4	4	0	4	...	5
42	0	1	15	0	1	0	4	4	2	0	...	4
43	0	1	15	0	1	0	2	2	2	2	...	5
44	0	0	16	0	0	0	2	2	4	3	...	4
45	0	0	15	0	0	1	4	3	4	4	...	5
46	0	0	16	0	0	1	3	3	4	2	...	2
47	0	1	16	0	1	0	4	3	1	2	...	4
48	0	1	15	0	1	0	4	2	0	4	...	4
49	0	0	15	0	1	0	4	4	2	0	...	4

50 rows × 33 columns

Linear Regression

Singular Linear Regression

```
corr_matrix = data[['studytime', 'failures', 'schoolsup', 'higher', 'romantic', 'famrel', 'h

# creating a heatmap of the correlation matrix
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=.05)

# adding labels and title
plt.title("Correlation matrix of G3 with other variables")
plt.tight_layout()
plt.show()
```



The variables that we want to identify are "studytime", "failures", "schoolsup", "higher", "romantic", "famrel", "health", and "absences". The diagonal elements of the matrix are all 1, as a variable is always perfectly correlated with itself. The other values in the matrix represent the correlation coefficients between the different variables. For example, the value of -0.173563 in the "studytime" row and "failures" column indicates a weak negative correlation between "studytime" and "failures".

Create a Linear Regression Model

```
# Import required libraries
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Define predictors and target variable
X = data[['studytime']]  # Independent variables
y = data['G3']  # Dependent variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=23)

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
# Create a linear regression model and fit it
LinR = LinearRegression()
LinR.fit(input, target_var)

# Use the fitted model to make predictions
predictions = LinR.predict(input)

# Create a new DataFrame that includes the predicted value and actual value (for checking pu
predictions = pd.DataFrame({"Prediction": predictions, "Actual Value": target_var})

# Display the prediction dataframe
print(predictions.head())
```

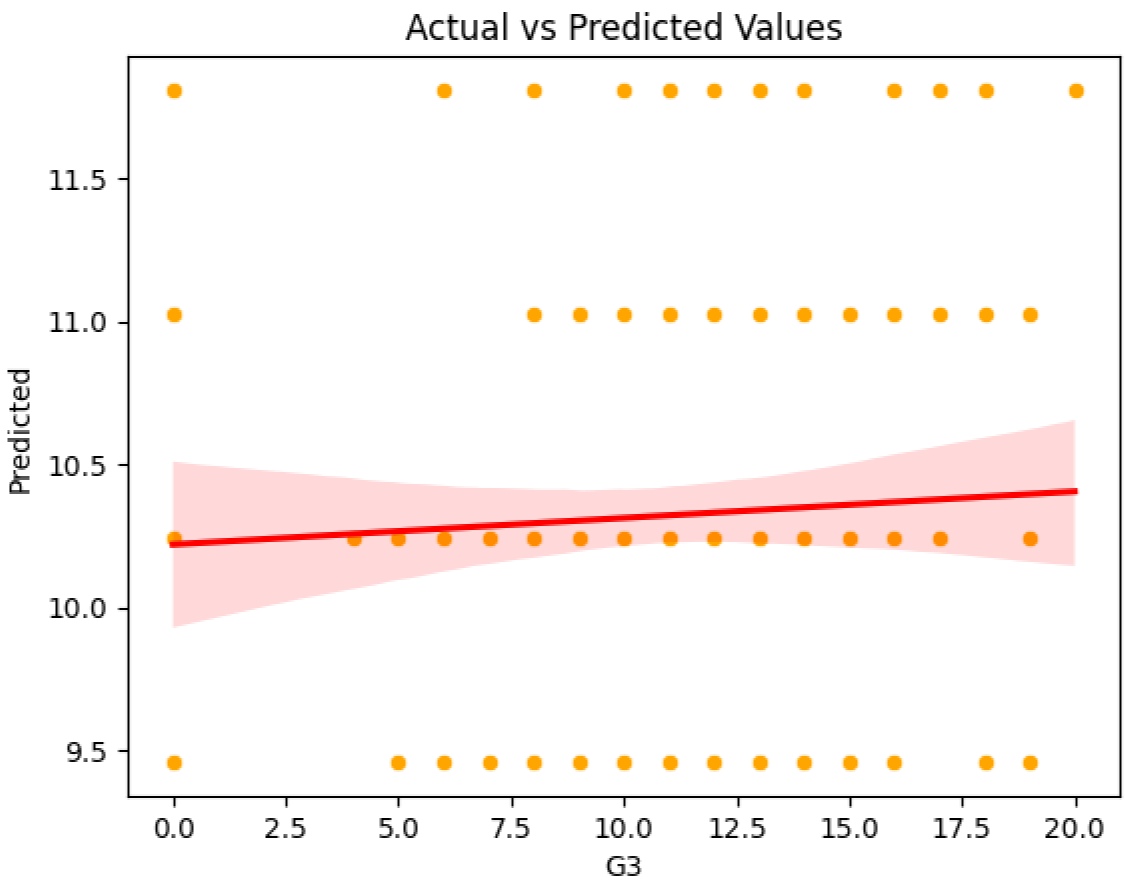
```
   Prediction  Actual Value
0   10.396263             6
1   10.396263             6
2   10.396263            10
3   10.930264            15
4   10.396263            10
```

Apply the Linear Regression Model

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

# Select the features the model will base on for prediction
attributes = ["studytime", "failures", "schoolsup", "higher", "romantic", "famrel", "health'

# Load the values of the features
x_input = data[list(attributes)].values

# Apply the model on the dataset
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x_input)

# Select the target variable
y_target = data["G3"].values

# Split the dataset into training and testing sets
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_scaled, y_target, test_size=0.2, rando

# Apply the model on the training dataset
model = LinearRegression()
model.fit(x_train, y_train)

# Apply the model on the testing dataset
predictions = model.predict(x_test)

# Create a new DataFrame that includes the predicted value and actual value (for checking pu
predictions = pd.DataFrame({"Prediction": predictions, "Actual Value": y_test})

# Display the prediction dataframe
predictions.head(25)
```

	Prediction	Actual Value
0	1.939016	10
1	8.133564	12
2	8.520085	5
3	11.188151	10
4	8.023107	9
5	10.956217	13
6	11.178688	18
7	11.787335	6
8	11.862462	0
9	11.584314	14
10	11.321628	15
11	11.008617	7
12	11.323256	15
13	10.622757	10
14	11.886315	14
15	10.646472	8
16	10.353048	8
17	11.449105	11
18	11.675196	15
19	12.103651	0
20	11.243121	14
21	10.688817	16
22	10.373155	16
23	10.590794	6
24	10.665190	0

```
# Plot actual vs predicted values
sns.scatterplot(x=y_test, y=y_pred, color='orange')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted Values')

# Plot regression line
sns.regplot(x=y_test, y=y_pred, scatter=False, color='red')
plt.show()
```



Evaluation of Linear Regression Model

The code provides a simple linear regression model that can predict the value of the dependent variable 'G3' based on the independent variable 'studytime'. In the table, 'y_test' represent the actual values, and 'y_pred' represent the predicted values. The scatterplot show that the actual values on the x-axis and the predicted values on the y-axis, with each point representing an observation. The regression line will then be fit to these points to show the general trend of the data. The regression line may have a negative slope, as the actual values appear to be lower than the predicted values. The scatterplot may show a pattern where the predicted values are generally higher than the actual values, suggesting that the model may be overestimating the actual values.

```
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Multiple Linear Regression

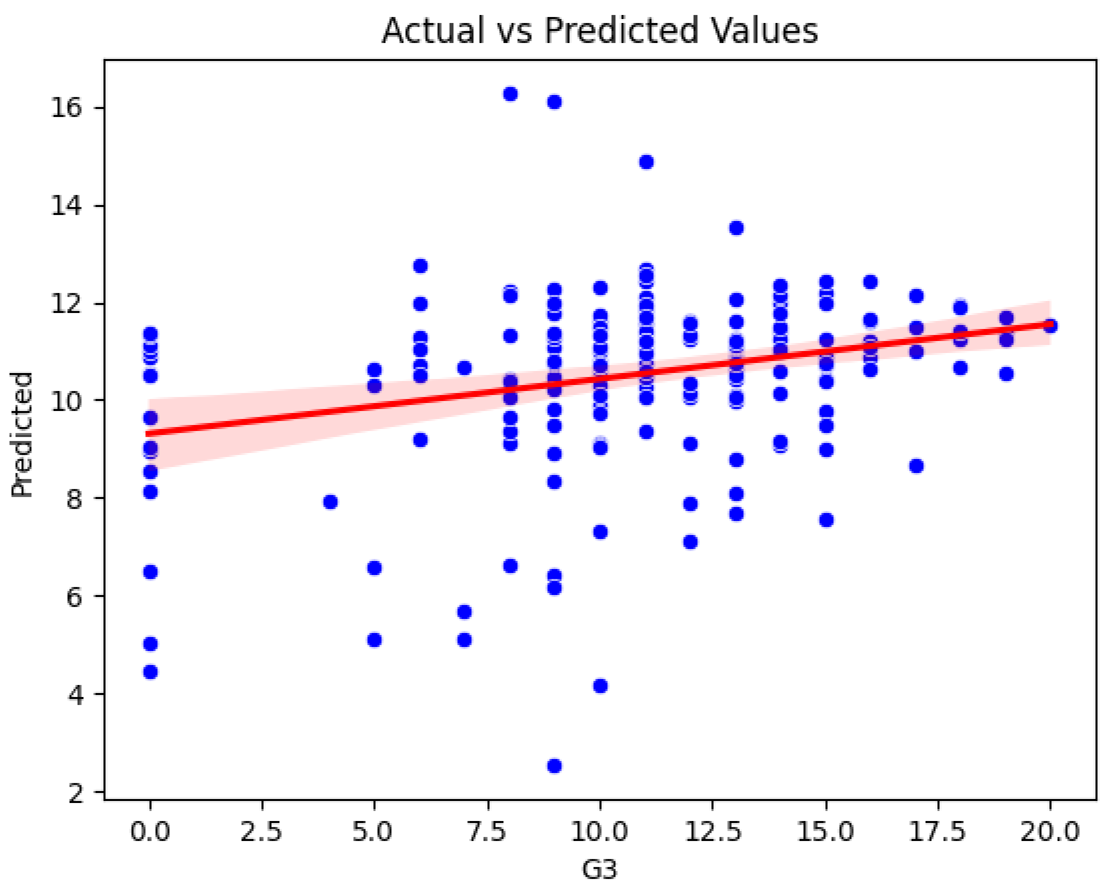
Create a Multiple Linear Regression Model

```
#Import data visualization libraries:
import matplotlib.pyplot as plt
import seaborn as sns

#Import model libraries:
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression

#Import Metrics libraries:
from sklearn.metrics import r2_score,mean_squared_error


# Define predictors and target variable
X = data[['studytime', 'failures', 'schoolsup', 'higher', 'romantic', 'famrel', 'health', 'a
y = data['G3']  # Dependent variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=23)

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)
```

Apply the Multiple Linear Regression Model

```
# Note: The Mean Squared Error (MSE) is a measure of the quality of the predictor.
##It is the average squared difference between the estimated values and the actual value.
# Note: The R-squared Score is a statistical measure that represents the proportion of the v
##for a dependent variable that's explained by an independent variable or variables in a reg
# Note: The scatterplot shows the actual values (y_test) against the predicted values (y_pre
# Note: The regression line is a straight line that best fits the data points. It is a line
##that minimizes the sum of the squared differences between the actual values and the predic

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r_squared = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R-squared Score:", r_squared)

# Plot actual vs predicted values
sns.scatterplot(x=y_test, y=y_pred, color='blue')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted Values')

# Plot regression line
sns.regplot(x=y_test, y=y_pred, scatter=False, color='red')
plt.show()
```

```
Mean Squared Error: 19.089004110475752
R-squared Score: 0.05774972911304077
```



Evaluation of Multiple Linear Regression Model

The R-squared Score is a measure of how well the independent variables in a regression model explain the variance in the dependent variable. It ranges from 0 to 1, where 1 indicates perfect explanation and 0 means no explanation. In this case, the R-squared Score is 0.0577, indicating that only a small fraction of the variance is explained by the independent variables. This suggests that the multiple linear regression model is not accurate in predicting the dependent variable. The high MSE further confirms the presence of unexplained variance. Hence, exploring additional variables or alternative models might improve predictions.

Polynomial Linear Regression

Create a Polynomial Linear Regression Model

```
from sklearn.linear_model import LinearRegression

# Identify the target variable to predict
target_var = data["G3"]

# Select the features the model will base on for prediction
attributes = ["studytime", "failures", "schoolsup", "higher", "romantic", "famrel", "health'

# Load the values of the features
input = data[list(attributes)].values


LinR = LinearRegression()
LinR.fit(input, target_var)
```

```
▾ LinearRegression
LinearRegression()
```

```
# Fitting Polynomial Regression to the dataset
from sklearn.preprocessing import PolynomialFeatures

PLR = PolynomialFeatures(degree=3)
Pinput = PLR.fit_transform(input)

PLR.fit(Pinput, target_var)
LinR2 = LinearRegression()
LinR2.fit(Pinput, target_var)
```

```
▾ LinearRegression
LinearRegression()
```

```
print(LinR.score(input, target_var))
print(LinR2.score(Pinput, target_var))
```

```
0.16324591796647403
-1.906278982376222
```

Apply the Polynomial Linear Regression Model

```
# Select the features the model will base on for prediction
attributes = ["studytime", "failures", "schoolsup", "higher", "romantic", "famrel", "health'

# Load the values of the features
x_input = test[list(attributes)].values

# Apply the model on the test dataset
predictions = LinR2.predict(PLR.fit_transform(x_input))

# Create a new DataFrame that includes the predicted value and actual value (for checking pu
predictions = pd.DataFrame({"Prediction": predictions, "Actual Value": test["G3"]})

# Display the prediction dataframe
predictions.head(25)
```

	Prediction	Actual Value
0	3.245930	11
1	11.533730	11
2	4.429955	12
3	11.318359	14
4	11.629276	13
5	16.747768	13
6	6.588108	13
7	11.452629	13
8	11.852386	17
9	15.459606	13
10	13.613083	14
11	13.037735	13
12	8.618889	12
13	9.852901	13
14	15.012421	15
15	4.978653	17
16	21.078674	14
17	4.412533	14
18	11.561165	7
19	12.094276	12
20	11.852386	14
21	14.816578	12
22	11.629276	14
23	16.413040	10
24	8.555965	10

```
from sklearn.metrics import mean_absolute_error as abs_err

# Using the Mean Absolute Error to evaluate the model
abs_err(predictions["Actual Value"], predictions["Prediction"])
```

```
10.018544560037151
```

Evaluation of Polynomial Linear Regression Model

The MAE is approximately 10.02. It implies that, on average, the model's predictions are off by about 10 grade points. This level of prediction error suggests that the model might not be capturing all the nuances of the relationship between the selected features and the final grade (G3).

Logistic Regression

Create a Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression

# Identify the target variable to predict
target_var = data["G3"]

# Select the features the model will base on for prediction
attributes = ["studytime", "failures", "schoolsup", "higher", "romantic", "famrel", "health'

# Load the values of the features
input = data[list(attributes)].values


# Create a Logistic Regression Object
LR = LogisticRegression(random_state = 0, solver = 'lbfgs', multi_class = 'ovr')

# Note: random_state determines the random seed that will be used, a fixed value ensures rep
# Note: solver determines the optimization algorithm that Logistic Regression will use, lbfg
# Note: multi_class determines multiclass classification handling, ovr means One-vs-Rest, wh

# Fit the model with the dataset
LR.fit(input, target_var)

# Display the score of the trained dataset
LR.score(input, target_var)

    0.2481012658227848
```

Apply the Logistic Regression Model

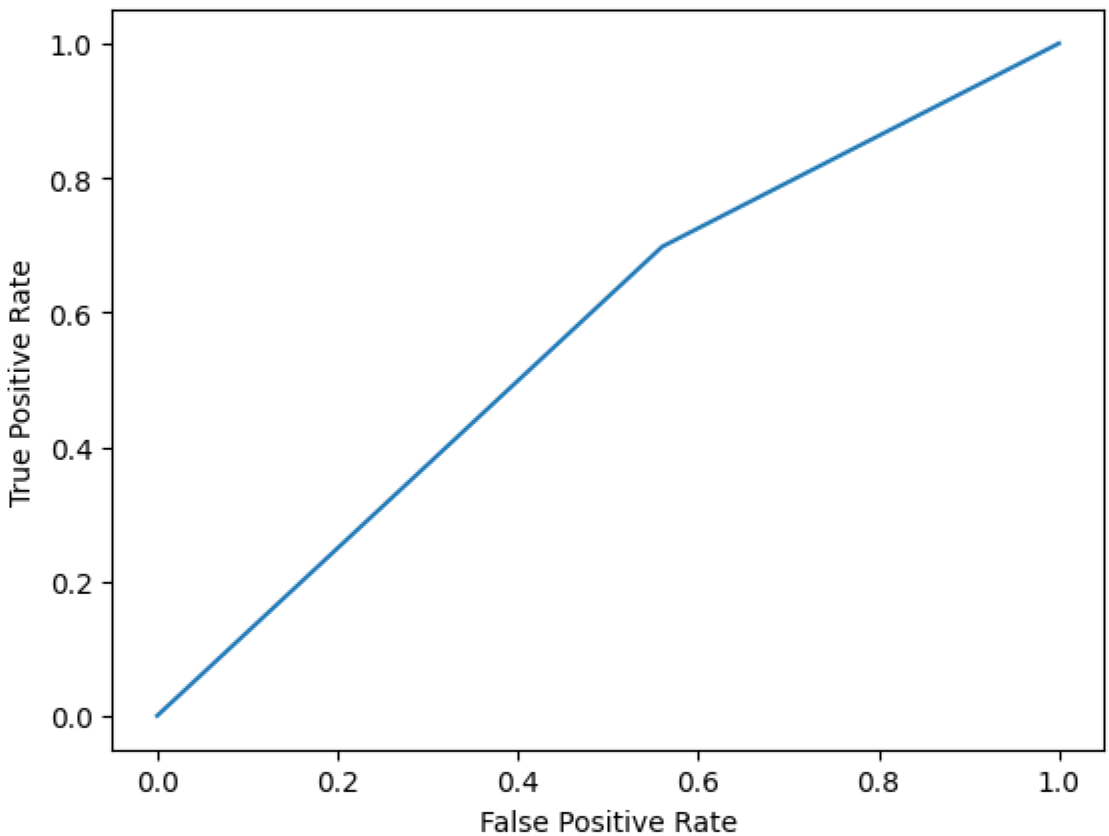
```
# Select the features the model will base on for prediction
attributes = ["studytime", "failures", "schoolsup", "higher", "romantic", "famrel", "health'

# Load the values of the features
x_input = test[list(attributes)].values

# Apply the Logistic Regression Model on the test dataset
predictions = LR.predict(x_input)

# Create a new DataFrame that includes the predicted value and actual value (for checking pu
predictions = pd.DataFrame({"Prediction": predictions, "Actual Value": test["G3"]})

# Display the prediction dataframe
predictions.head(25)
```

	ID	Prediction	Actual Value
0	0	10	11
1	1	10	11
2	2	10	12
3	3	0	14
4	4	0	13
5	5	10	13
6	6	0	13
7	7	10	13
8	8	0	17
9	9	10	13
10	10	10	14
11	11	10	13
12	12	0	12
13	13	10	13
14	14	0	15
15	15	10	17
16	16	11	14
17	17	10	14
18	18	10	7
19	19	11	12
20	20	0	14
21	21	10	12
22	22	0	14
23	23	10	10
24	24	11	10

```
# Display the accuracy of the model of both training and test data sets
print("Training score:" + str(LR.score(input, target_var)))
print("Testing score:" + str(LR.score(x_input, test["G3"])))
```

```
Training score:0.2481012658227848
Testing score:0.12172573189522343
```

```
from sklearn import metrics

# Normalize the data
predictions["Actual Value"] = predictions["Actual Value"].apply(lambda toLabel: 0 if toLabel
predictions["Prediction"] = predictions["Prediction"].apply(lambda toLabel: 0 if toLabel < 1


# Visualize the data
fpr, tpr, _ = metrics.roc_curve(predictions["Actual Value"], predictions["Prediction"])

plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



The plot generated from the code displays the Receiver Operating Characteristic (ROC) curve. This curve illustrates the True Positive Rate (TPR) on the y-axis and the False Positive Rate (FPR) on the x-axis. Each point on the curve represents a sensitivity/specificity pair corresponding to a particular discrimination threshold.

The proximity of the curve to the top-left corner indicates higher test accuracy. A curve closer to the 45-degree line suggests a weaker model, while further deviation from this line signifies better class distinction.

In summary, the ROC curve visually assesses the model's ability to distinguish between positive and negative classes, with closer proximity to the top-left corner indicating superior performance.

Evaluation of Logistic Regression Model

The accuracy for the training dataset is 0.248, and for the testing dataset, it is 0.122. Accuracy measures the proportion of correct predictions out of the total predictions made by the model. A low testing score (0.122) suggests overfitting to the training data and poor generalization to unseen data. This may result from a complex model with many parameters or a small dataset. To mitigate overfitting, techniques like regularization, cross-validation, and simplifying the model can be employed.

Decision Tree

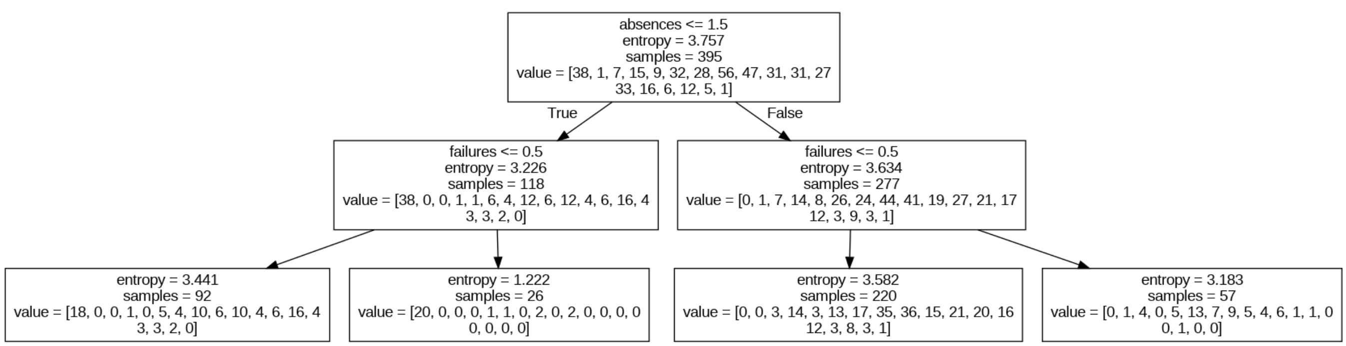
Creating a Decision Tree

```
# Import necessary libraries and packages
from sklearn import tree
from six import StringIO
from IPython.display import Image

# Set the target variable along with its data
target_var = data["G3"].values

# Identify the input variables
attributes = ["studytime", "failures", "schoolsup", "higher", "romantic", "famrel", "health'

# List the input variable data
input = data[list(attributes)].values
```

```
# Create a decision tree classifier object
dec_tree = tree.DecisionTreeClassifier(criterion = "entropy", max_depth = 2)

# Note: "Entropy" represents the criterion in which the tree bases its decision or predictic
# Note: max_depth represents how many 'children' does the root node can have.

# Train the object using the fit() function and passing the input and target variable
dec_tree = dec_tree.fit(input, target_var)

# Evaluation of model through its accuracy
dec_tree.score(input, target_var)
```

```
0.22025316455696203
```

```
# Convert the decision tree into an image
with open("decision.dot", 'w') as y:
  y = tree.export_graphviz(dec_tree, out_file = y, feature_names = attributes)


# Convert the dot file to a png
!dot -Tpng decision.dot -o decision.png


# Display the image
Image("decision.png")
```



It's determining the best attribute to split the data based on entropy reduction. Entropy measures data impurity, ranging from 0 to 4.5. Lower entropy indicates clearer separation of classes. For each attribute, entropy is calculated for different conditions. The attribute with the lowest entropy becomes the split. This process repeats, forming a tree. The resulting tree aids in classification or regression tasks by maximizing class separation.

```
data.info()
```

```
<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    int64 
 1   sex         395 non-null    int64 
 2   age         395 non-null    int64 
 3   address     395 non-null    int64 
 4   famsize     395 non-null    int64 
 5   Pstatus     395 non-null    int64 
 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    int64 
 16  famsup      395 non-null    int64 
 17  paid        395 non-null    int64 
 18  activities  395 non-null    int64 
 19  nursery     395 non-null    int64 
 20  higher      395 non-null    int64 
 21  internet    395 non-null    int64 
 22  romantic    395 non-null    int64 
 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(29), object(4)
memory usage: 102.0+ KB
```

Apply the Decision Tree

```
# Identify the attributes of the input variables
attributes = ["studytime", "failures", "schoolsup", "higher", "romantic", "famrel", "health'

# Apply the trained model to the input variables
prediction = dec_tree.predict(x_input)

# Create a new dataframe containing the prediction and actual value (for checking purposes)
prediction = pd.DataFrame({"Prediction": prediction, "Actual Value": test["G3"]})

# Display the new dataframe
prediction.head(25)
```

	Prediction	Actual Value
0	11	11
1	11	11