LONDON METROPOLITAN UNIVERSITY

Islington college
(इस्लिङ्टन कलेज)

CC7182NI Programming for Data Analytics

100% Individual Coursework

Autumn Semester 2021

Student Name: Pravash Karki

London Met ID: 11071480

College ID: NP01MS7S210070

Assignment Due Date: 28 January 2022

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

Table of Contents

List of Figures

List of Tables

Introduction

Data analytics is the study and analysis of large databases in order to ascertain the information contained within them. The strategies enable us to transform raw data into actionable insights by recognising patterns and analysing the data using the Python computer language. (McKinney, 2017)

The coursework for this module includes student achievement data from two Portuguese secondary schools, Gabriel Pereira and Mousinho da Silveira. The second data set includes livestock data from different districts and regions in Nepal, along with their commodities.

This assignment displays the semester's learning outcomes for Python programming and analytics methodologies. Data visualisation is just as critical as analytics. Python is an ideal language for data analysis due to its usage of built-in libraries such as Pandas, NumPy, Matplotlib, and other analytics tools.

Part 1. Analysis of a Student Performance Dataset

1. Data Understanding

This coursework will analyse data on secondary students' performance at two Portuguese schools: Gabriel Pereira and Mourinho da Silveira. The data will be gathered through school reports and questionnaires. The dataset "student.csv" contains 395 student records and 33 attributes containing information about students. The student's grades, socioeconomic, social, and school-related statistics are among the data attributes. G2 and G1 are highly correlated with the desired attribute G3. G3 is the final year grade (issued during the third period), whereas G1 and G2 are first and second-period grades. Predicting G3 in the absence of G2 and G1 is more complicated, but the benefits are far more significant.

Attribute Information

Attributes for "student.csv" dataset

SN	Column Name	Description	Variable Type
1.	school	Information about the student's school name ('GP' stands for Gabriel Pereira and 'MS' stands for Mousinho da Silveira).	Categorical
2.	sex	Student's gender 'F' indicates female and 'M' indicates male.	Categorical
3.	age	Student's actual age in years between the ages of 15 and 22	Discrete
4.	address	Type of student's residential address 'U' - urban or 'R' - rural	Categorical
5.	famsize	'LE3' denotes a family of less than or equal to three members; 'GT3' denotes a family of more than three members.	Categorical
6.	Pstatus	Cohabitation status of parents 'T' - living together or 'A' - living apart	Categorical
7.	Medu	Mother's educational status is classified into four categories: none, primary education (4th grade), 5th to 9th grade, secondary education, higher education.	Discrete
8.	Fedu	Father's educational status is classified into four categories: none, primary education (4th grade), 5th to 9th grade, secondary education, higher education.	Discrete
9.	Mjob	Mothers' job categories include 'teacher', 'health' care related, civil services (e.g. administrative or police), 'at home', and 'other'.	Discrete

10.	Fjob	Fathers' job categories include 'teacher', 'health' care related, civil services (e.g. administrative or police), 'at home', and 'other'.	Discrete
11.	reason	The reasons for selecting the school include proximity to home, school reputation, course preference, or other.	Discrete
12.	guardian	The registered guardian of a student is either mother, father, or other.	Discrete
13.	traveltime	Travel time from home to school (15 minutes, 15 to 30 minutes, 30 minutes to 1 hour, >1 hour)	Discrete
14.	studytime	Students' weekly study time (2 hours, 2 to 5 hours, 5 to 10 hours, or more than 10 hours)	Discrete
15.	failures	Number of past class failures (n if 1<=n<3, else 4)	Numerical
16.	schoolsup	Extra educational support (yes or no)	Categorical
17.	famsup	Family educational support (yes or no)	Categorical
18.	paid	Additional paid classes within the subject of the course (Math or Portuguese) (yes or no))	Categorical
19.	activities	Extra-curricular activities (yes or no)	Categorical
20.	nursery	Attended nursery school (yes or no)	Categorical
21.	higher	Desires to pursue higher education (yes or no))	Categorical
22.	internet	Internet access at home (yes or no)	Categorical
23.	romantic	With a romantic relationship (yes or no)	Categorical
24.	famrel	Family relationships' quality (from 1 - very bad to 5 - excellent))	Discrete
25.	freetime	After-school free time (from 1 - extremely low to 5 - extremely high))	Categorical
26.	goout	Socializing with friends (from 1 - extremely low to 5 - extremely high)	Categorical
27.	Dalc	Alcohol consumption during the workday (from 1 - very low to 5 - very high)	Categorical
28.	Walc	Weekend alcohol consumption (ranging from 1 – extremely low to 5 – extremely high))	Categorical
29.	health	Current health status of students (from 1 - very bad to 5 - very good)	Categorical
30.	absences	Number of absences from school (range: 0 to 93))	Discrete
31.	G1	First period grade of students (from 0 to 20)	Discrete
32.	G2	Second period grade of students (from 0 to 20)	Discrete
33.	G3	Final grade students (from 0 to 20, output target)	Discrete

Table 1: Attributes description and variable type.

2. Data Transformation

After understanding the data and requirnments loading all required package and converting all of the data provided in the student.csv data set into a pandas data frame using the read_csv() function.

```
# Import Python packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
```

```
# Making dataframe
student_csv_url = 'Dataset/Part 1/student.csv'
df = pd.read_csv(student_csv_url)
```

Figure 1: Importing Packages & making pandas dataframe 'df'

```
# Print the dataframe
df
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	go
0	GP	F	18	U	GT3	A	higher education	higher education	at_home	teacher	...	very good	medium	h
1	GP	F	17	U	GT3	T	primary education (4th grade)	primary education (4th grade)	at_home	other	...	excellent	medium	med
2	GP	F	15	U	LE3	T	primary education (4th grade)	primary education (4th grade)	at_home	other	...	very good	medium	
3	GP	F	15	U	GT3	T	higher education	5th to 9th grade	health	services	...	good	low	
4	GP	F	16	U	GT3	T	secondary education	secondary education	other	other	...	very good	medium	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
390	MS	M	20	U	LE3	A	5th to 9th grade	5th to 9th grade	services	services	...	excellent	very high	h
391	MS	M	17	U	LE3	T	secondary education	primary education (4th grade)	services	services	...	bad	high	v h
392	MS	M	21	R	GT3	T	primary education (4th grade)	primary education (4th grade)	other	other	...	excellent	very high	med
393	MS	M	18	R	LE3	T	secondary education	5th to 9th grade	services	other	...	very good	high	v
394	MS	M	19	U	LE3	T	primary education (4th grade)	primary education (4th grade)	other	at_home	...	good	low	med

395 rows × 33 columns

Figure 2: Printing the Dataframe Data

```
# Full summary of the Dataframe
df.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    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    object
 7   Fedu        395 non-null    object
 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    object
 13  studytime   395 non-null    object
 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    object
 24  freetime    395 non-null    object
 25  goout       395 non-null    object
 26  Dalc        395 non-null    object
 27  Walc        395 non-null    object
 28  health      395 non-null    object
 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(6), object(27)
memory usage: 102.0+ KB
```

Figure 3: Full summary of the Dataframe

2.a Transform Variables - Binary

Understanding the unique value and binary conversion (binary; 0 or 1) and creating new columns of school, sex, address, famsize, Pstatus, schoolsup, famsup, paid, activities, nursery, higher, internet and romantic without overwriting the existing columns.

Transform Variable 'school'

```
""" Defining Functions
Concats (joins) original column and column with replaced value
"""
def concat_cols(df, col, newcol):
    return pd.concat((
        df[col],
        newcol.rename(f'new_{col}')),
        axis=1
    )

# Returns Array of unique values in any column
def unique_cols(col):
    return col.unique()
```

Figure 4: Defining two function one to concat original column and new column and other to return array of unique values.

```
# Understand unique value of column
df.school.unique()
```

```
array(['GP', 'MS'], dtype=object)
```

```
col = df.school
data = col.unique() # Unique Value
vals = (0,1) # Binary Value

# Replace Values
replace_vals = col.replace(data, vals)

# Create new column
new_col_school = concat_cols(df, 'school', replace_vals)

# Unique values in column
new_col_school.apply(unique_cols)
```

	school	new_school
0	GP	0
1	MS	1

Figure 5: Unique value of column 'school' and binary transformation.

Transform Variable 'sex'

```
# Understand unique value of column sex
df.sex.unique()
```

```
array(['F', 'M'], dtype=object)
```

```
col = df.sex
data = col.unique() # Unique Value
vals = (0,1) # Binary Value

# Replace Values
replace_vals = col.replace(data, vals)

# Create new coloumn
new_col_sex = concat_cols(df, 'sex', replace_vals)

# Unique values in column
new_col_sex.apply(unique_cols)
```

	sex	new_sex
0	F	0
1	M	1

Figure 6: Unique value of column 'sex' and binary transformation.

Transform Variable 'address'

```
# Understand unique value of column address
df.address.unique()
```

```
array(['U', 'R'], dtype=object)
```

```
col = df.address
data = col.unique() # Unique Value
vals = (0,1) # Binary Value

# Replace Values
replace_vals = col.replace(data, vals)

# Create new coloumn
new_col_address = concat_cols(df, 'address', replace_vals)

# Unique values in column
new_col_address.apply(unique_cols)
```

	address	new_address
0	U	0
1	R	1

Figure 7: Unique value of column 'address' and binary transformation.

Transform Variable 'famsize'

```
# Understand unique value of column famsize
df.famsize.unique()
```

```
array(['GT3', 'LE3'], dtype=object)
```

```
col = df.famsize
data = col.unique() # Unique Value
vals = (0,1) # Binary Value

# Replace Values
replace_vals = col.replace(data, vals)

# Create new coloumn
new_col_famsize = concat_cols(df, 'famsize', replace_vals)

# Unique values in column
new_col_famsize.apply(unique_cols)
```

	famsize	new_famsize
0	GT3	0
1	LE3	1

Figure 8: Unique value of column 'famsize' and binary transformation.

Transform Variable 'Pstatus'

```
# Understand unique value of column Pstatus
df.Pstatus.unique()
```

```
array(['A', 'T'], dtype=object)
```

```
col = df.Pstatus
data = col.unique() # Unique Value
vals = (0,1) # Binary Value

# Replace Values
replace_vals = col.replace(data, vals)

# Create new coloumn
new_col_pstatus = concat_cols(df, 'Pstatus', replace_vals)

# Unique values in column
new_col_pstatus.apply(unique_cols)
```

	Pstatus	new_Pstatus
0	A	0
1	T	1

Figure 9: Unique value of column 'Pstatus' and binary transformation

Transform Variable 'schoolsup'

```
# Understand unique value of column schoolsup
df.schoolsup.unique()
```

```
array(['yes', 'no'], dtype=object)
```

```
col = df.schoolsup
data = col.unique() # Unique Value
vals = (0,1) # Binary Value

# Replace Values
replace_vals = col.replace(data, vals)

# Create new coloumn
new_col_schoolsup = concat_cols(df, 'schoolsup', replace_vals)

# Unique values in column
new_col_schoolsup.apply(unique_cols)
```

	schoolsup	new_schoolsup
0	yes	0
1	no	1

Figure 10: Unique value of column 'schoolsup' and binary transformation.

Transform Variable 'famsup'

```
# Understand unique value of column famsup
df.famsup.unique()
```

```
array(['no', 'yes'], dtype=object)
```

```
col = df.famsup
data = col.unique() # Unique Value
vals = (0,1) # Binary Value

# Replace Values
replace_vals = col.replace(data, vals)

# Create new coloumn
new_col_famsup = concat_cols(df, 'famsup', replace_vals)

# Unique values in column
new_col_famsup.apply(unique_cols)
```

	famsup	new_famsup
0	no	0
1	yes	1

Figure 11: Unique value of column 'famsup' and binary transformation.

Transform Variable 'paid'

```
#Understand unique value of column paid
df.paid.unique()
```

```
array(['no', 'yes'], dtype=object)
```

```
col = df.paid
data = col.unique() # Unique Value
vals = (0,1) # Binary Value

# Replace Values
replace_vals = col.replace(data, vals)

# Create new coloumn
new_col_paid = concat_cols(df, 'paid', replace_vals)

# Unique values in column
new_col_paid.apply(unique_cols)
```

	paid	new_paid
0	no	0
1	yes	1

Figure 12: Unique value of column 'paid' and binary transformation.

Transform Variable 'activities'

```
# Understand unique value of column activities
df.activities.unique()
```

```
array(['no', 'yes'], dtype=object)
```

```
col = df.activities
data = col.unique() # Unique Value
vals = (0,1) # Binary Value

# Replace Values
replace_vals = col.replace(data, vals)

# Create new coloumn
new_col_activities = concat_cols(df, 'activities', replace_vals)

# Unique values in column
new_col_activities.apply(unique_cols)
```

	activities	new_activities
0	no	0
1	yes	1

Figure 13: Unique value of column 'activities' and binary transformation.

Transform Variable 'nursery'

```
# Understand unique value of column nursery
df.nursery.unique()
```

```
array(['yes', 'no'], dtype=object)
```

```
col = df.nursery
data = col.unique() # Unique Value
vals = (0,1) # Binary Value

# Replace Values
replace_vals = col.replace(data, vals)

# Create new coloumn
new_col_nursery = concat_cols(df, 'nursery', replace_vals)

# Unique values in column
new_col_nursery.apply(unique_cols)
```

	nursery	new_nursery
0	yes	0
1	no	1

Figure 14: Unique value of column 'nursery' and binary transformation.

Transform Variable 'higher'

```
# Understand unique value of column higher
df.higher.unique()
```

```
array(['yes', 'no'], dtype=object)
```

```
col = df.higher
data = col.unique() # Unique Value
vals = (0,1) # Binary Value

# Replace Values
replace_vals = col.replace(data, vals)

# Create new coloumn
new_col_higher = concat_cols(df, 'higher', replace_vals)

# Unique values in column
new_col_higher.apply(unique_cols)
```

	higher	new_higher
0	yes	0
1	no	1

Figure 15: Unique value of column 'higher' and binary transformation.

Transform Variable 'internet'

```
# Understand unique value of column internet
col = df.internet
```

```
col = df.internet
data = col.unique() # Unique Value
vals = (0,1) # Binary Value

# Replace Values
replace_vals = col.replace(data, vals)

# Create new coloumn
new_col_internet = concat_cols(df, 'internet', replace_vals)

# Unique values in column
new_col_internet.apply(unique_cols)
```

	internet	new_internet
0	no	0
1	yes	1

Figure 16: Unique value of column 'internet' and binary transformation

Transform Variable 'romantic'

```
# Understand unique value of column romantic
df.romantic.unique()
```

```
array(['no', 'yes'], dtype=object)
```

```
col = df.romantic
data = col.unique() # Unique Value
vals = (0,1) # Binary Value

# Replace Values
replace_vals = col.replace(data, vals)

# Create new coloumn
new_col_romantic = concat_cols(df, 'romantic', replace_vals)

# Unique values in column
new_col_romantic.apply(unique_cols)
```

	romantic	new_romantic
0	no	0
1	yes	1

Figure 17: Unique value of column 'romantic' and binary transformation

2.b Transform Variables - Ordinal numbers

Understanding the unique value and converting Medu, Fedu, Mjob, Fjob, reason, guardian, traveltime, studytime, famrel, freetime, gout, Dalc, Walc and health into ordinal numbers based on number of cases in the data set by creating new columns without overwriting the existing ones.

Transform Variable 'Medu'

```
# Understand unique value of column Medu
df.Medu.unique()
```

```
array(['higher education', 'primary education (4th grade)',
       'secondary education', '5th to 9th grade', 'none'], dtype=object)
```

```
col = df.Medu # Dataframe column
data = col.unique() # Unique Value
data = tuple(data) # Tuple of unique value
vals = tuple(range(0, len(data))) #Tuple of ordinal number based on data

replace_vals = col.replace(data, vals) # Transform to ordinal numbers

# Create new ordinal number column
new_col_medu = concat_cols(df, 'Medu', replace_vals)

# Unique values in columns
new_col_medu.apply(unique_cols)
```

	Medu	new_Medu
0	higher education	0
1	primary education (4th grade)	1
2	secondary education	2
3	5th to 9th grade	3
4	none	4

Figure 18: Unique value of column 'Medu' and ordinal transformation

Transform Variable 'Fedu'

```
# Understand unique value of column Fedu
df.Fedu.unique()
```

```
array(['higher education', 'primary education (4th grade)',
       '5th to 9th grade', 'secondary education', 'none'], dtype=object)
```

```
col = df.Fedu # Dataframe column
data = col.unique() # Unique Value
data = tuple(data) # Tuple of unique value
vals = tuple(range(0, len(data))) #Tuple of ordinal number based on data

replace_vals = col.replace(data, vals) # Transform to ordinal numbers

# Create new ordinal number column
new_col_fedu = concat_cols(df, 'Fedu', replace_vals)

# Unique values in columns
new_col_fedu.apply(unique_cols)
```

	Fedu	new_Fedu
0	higher education	0
1	primary education (4th grade)	1
2	5th to 9th grade	2
3	secondary education	3
4	none	4

Figure 19: Unique value of column 'Fedu' and ordinal transformation

Transform Variable 'Mjob'

```
# Understand unique value of column Mjob
df.Mjob.unique()
```

```
array(['at_home', 'health', 'other', 'services', 'teacher'], dtype=object)
```

```
col = df.Mjob # Dataframe column
data = col.unique() # Unique Value
data = tuple(data) # Tuple of unique value
vals = tuple(range(0, len(data))) #Tuple of ordinal number based on data

replace_vals = col.replace(data, vals) # Transform to ordinal numbers

# Create new ordinal number column
new_col_mjob = concat_cols(df, 'Mjob', replace_vals)

# Unique values in columns
new_col_mjob.apply(unique_cols)
```

	Mjob	new_Mjob
0	at_home	0
1	health	1
2	other	2
3	services	3
4	teacher	4

Figure 20: Unique value of column 'Mjob' and ordinal transformation

Transform Variable 'Fjob'

```
# Understand unique value of column Fjob
df.Fjob.unique()
```

```
array(['teacher', 'other', 'services', 'health', 'at_home'], dtype=object)
```

```
col = df.Fjob # Dataframe column
data = col.unique() # Unique Value
data = tuple(data) # Tuple of unique value
vals = tuple(range(0, len(data))) #Tuple of ordinal number based on data

replace_vals = col.replace(data, vals) # Transform to ordinal numbers

# Create new ordinal number column
new_col_fjob = concat_cols(df, 'Fjob', replace_vals)

# Unique values in columns
new_col_fjob.apply(unique_cols)
```

	Fjob	new_Fjob
0	teacher	0
1	other	1
2	services	2
3	health	3
4	at_home	4

Figure 21: Unique value of column 'Fjob' and ordinal transformation

Transform Variable 'reason'

```
# Understand unique value of column reason
df.reason.unique()
```

```
array(['course', 'other', 'home', 'reputation'], dtype=object)
```

```
col = df.reason # Dataframe column
data = col.unique() # Unique Value
data = tuple(data) # Tuple of unique value
vals = tuple(range(0, len(data))) #Tuple of ordinal number based on data

replace_vals = col.replace(data, vals) # Transform to ordinal numbers

# Create new ordinal number column
new_col_reason = concat_cols(df, 'reason', replace_vals)

# Unique values in columns
new_col_reason.apply(unique_cols)
```

	reason	new_reason
0	course	0
1	other	1
2	home	2
3	reputation	3

Figure 22: Unique value of column 'reason' and ordinal transformation

Transform Variable 'guardian'

```
# Understand unique value of column guardian
df.guardian.unique()
```

```
array(['mother', 'father', 'other'], dtype=object)
```

```
col = df.guardian # Dataframe column
data = col.unique() # Unique Value
data = tuple(data) # Tuple of unique value
vals = tuple(range(0, len(data))) #Tuple of ordinal number based on data

replace_vals = col.replace(data, vals) # Transform to ordinal numbers

# Create new ordinal number column
new_col_guardian = concat_cols(df, 'guardian', replace_vals)

# Unique values in columns
new_col_guardian.apply(unique_cols)
```

	guardian	new_guardian
0	mother	0
1	father	1
2	other	2

Figure 23: Unique value of column 'guardian' and ordinal transformation

Transform Variable 'traveltime'

```
# Understand unique value of column traveltime
df.traveltime.unique()
```

```
array(['15 to 30 min.', '<15 min.', '30 min. to 1 hour', '>1 hour'],
      dtype=object)
```

```
col = df.traveltime # Dataframe column
data = col.unique() # Unique Value
data = tuple(data) # Tuple of unique value
vals = tuple(range(0, len(data))) #Tuple of ordinal number based on data

replace_vals = col.replace(data, vals) # Transform to ordinal numbers

# Create new ordinal number column
new_col_traveltime = concat_cols(df, 'traveltime', replace_vals)

# Unique values in columns
new_col_traveltime.apply(unique_cols)
```

	traveltime	new_traveltime
0	15 to 30 min.	0
1	<15 min.	1
2	30 min. to 1 hour	2
3	>1 hour	3

Figure 24: Unique value of column 'traveltime' and ordinal transformation

Transform Variable 'studytime'

```
# Understand unique value of column studytime
df.studytime.unique()
```

```
array(['2 to 5 hours', '5 to 10 hours', '<2 hours', '>10 hours'],
      dtype=object)
```

```
col = df.studytime # Dataframe column
data = col.unique() # Unique Value
data = tuple(data) # Tuple of unique value
vals = tuple(range(0, len(data))) #Tuple of ordinal number based on data

replace_vals = col.replace(data, vals) # Transform to ordinal numbers

# Create new ordinal number column
new_col_studytime = concat_cols(df, 'studytime', replace_vals)

# Unique values in columns
new_col_studytime.apply(unique_cols)
```

	studytime	new_studytime
0	2 to 5 hours	0
1	5 to 10 hours	1
2	<2 hours	2
3	>10 hours	3

Figure 25: Unique value of column 'studytime' and ordinal transformation

Transform Variable 'famrel'

```
# Understand unique value of column famrel
df.famrel.unique()
```

```
array(['very good', 'excellent', 'good', 'very bad', 'bad'], dtype=object)
```

```
col = df.famrel # Dataframe column
data = col.unique() # Unique Value
data = tuple(data) # Tuple of unique value
vals = tuple(range(0, len(data))) #Tuple of ordinal number based on data

replace_vals = col.replace(data, vals) # Transform to ordinal numbers

# Create new ordinal number column
new_col_famrel = concat_cols(df, 'famrel', replace_vals)

# Unique values in columns
new_col_famrel.apply(unique_cols)
```

	famrel	new_famrel
0	very good	0
1	excellent	1
2	good	2
3	very bad	3
4	bad	4

Figure 26: Unique value of column 'famrel' and ordinal transformation

Transform Variable 'freetime'

```
#Understand unique value of column freetime
df.freetime.unique()
```

```
array(['medium', 'low', 'high', 'very low', 'very high'], dtype=object)
```

```
col = df.freetime # Dataframe column
data = col.unique() # Unique Value
data = tuple(data) # Tuple of unique value
vals = tuple(range(0, len(data))) #Tuple of ordinal number based on data

replace_vals = col.replace(data, vals) # Transform to ordinal numbers

# Create new ordinal number column
new_col_freetime = concat_cols(df, 'freetime', replace_vals)

# Unique values in columns
new_col_freetime.apply(unique_cols)
```

	freetime	new_freetime
0	medium	0
1	low	1
2	high	2
3	very low	3
4	very high	4

Figure 27: Unique value of column 'freetime' and ordinal transformation

Transform Variable 'goout'

```
# Understand unique value of column gout
df.goout.unique()
```

```
array(['high', 'medium', 'low', 'very low', 'very high'], dtype=object)
```

```
col = df.goout # Dataframe column
data = col.unique() # Unique Value
data = tuple(data) # Tuple of unique value
vals = tuple(range(0, len(data))) #Tuple of ordinal number based on data

replace_vals = col.replace(data, vals) # Transform to ordinal numbers

# Create new ordinal number column
new_col_goout = concat_cols(df, 'goout', replace_vals)

# Unique values in columns
new_col_goout.apply(unique_cols)
```

	goout	new_goout
0	high	0
1	medium	1
2	low	2
3	very low	3
4	very high	4

Figure 28: Unique value of column 'goout' and ordinal transformation

Transform Variable 'Dalc'

```
# Understand unique value of column Dalc
df.Dalc.unique()
```

```
array(['very low', 'low', 'very high', 'medium', 'high'], dtype=object)
```

```
col = df.Dalc # Dataframe column
data = col.unique() # Unique Value
data = tuple(data) # Tuple of unique value
vals = tuple(range(0, len(data))) #Tuple of ordinal number based on data

replace_vals = col.replace(data, vals) # Transform to ordinal numbers

# Create new ordinal number column
new_col_dalc = concat_cols(df, 'Dalc', replace_vals)

# Unique values in columns
new_col_dalc.apply(unique_cols)
```

	Dalc	new_Dalc
0	very low	0
1	low	1
2	very high	2
3	medium	3
4	high	4

Figure 29: Unique value of column 'Dale' and ordinal transformation

Transform Variable 'Walc'

```
# Understand unique value of column Walc
df.Walc.unique()
```

```
array(['very low', 'medium', 'low', 'high', 'very high'], dtype=object)
```

```
col = df.Walc # Dataframe column
data = col.unique() # Unique Value
data = tuple(data) # Tuple of unique value
vals = tuple(range(0, len(data))) #Tuple of ordinal number based on data

replace_vals = col.replace(data, vals) # Transform to ordinal numbers

# Create new ordinal number column
new_col_wcal = concat_cols(df, 'Walc', replace_vals)

# Unique values in columns
new_col_wcal.apply(unique_cols)
```

	Walc	new_Walc
0	very low	0
1	medium	1
2	low	2
3	high	3
4	very high	4

Figure 30: Unique value of column 'Walc' and ordinal transformation

Transform Variable 'health'

```
# Understand unique value of column health
df.health.unique()
```

```
array(['good', 'excellent', 'very bad', 'bad', 'very good'], dtype=object)
```

```
col = df.health # Dataframe column
data = col.unique() # Unique Value
data = tuple(data) # Tuple of unique value
vals = tuple(range(0, len(data))) #Tuple of ordinal number based on data

replace_vals = col.replace(data, vals) # Transform to ordinal numbers

# Create new ordinal number column
new_col_health = concat_cols(df, 'health', replace_vals)

# Unique values in columns
new_col_health.apply(unique_cols)
```

	health	new_health
0	good	0
1	excellent	1
2	very bad	2
3	bad	3
4	very good	4

Figure 31: Unique value of column 'health' and ordinal transformation

2.c Create a New Column - age_category

Values should be based on the values in the age column, divide the values into 3 ordinal numbers; 1 – 15 to 17, 2 – 18 to 20, 3 – 21 and over.

```
# Understand unique value of column age
df.age.unique()
```

```
array([18, 17, 15, 16, 19, 22, 20, 21])
```

```
# Creating function which categorize data according to age.

def transform_age(age):
    if age >= 21:
        return 3
    elif age >=18:
        return 2
    elif age >= 15:
        return 1
```

Figure 32: Understanding the unique value and categorizing according to age.

```
# Applying the transformed value based on the value of age

df['age_category'] = df['age'].apply(transform_age)
# transform_age function is used to convert the age in ordinal numbers

df[['age', 'age_category']]
```

	age	age_category
0	18	2
1	17	1
2	15	1
3	15	1
4	16	1
...	...	...
390	20	2
391	17	1
392	21	3
393	18	2
394	19	2

395 rows × 2 columns

Figure 33: Dividing the values into 3 ordinal numbers

2.d Create a New Column - passed (yes or no)

Create a new column named passed (yes or no) whose values should be based on the values present in the G3 column (>=8 – yes, <8 – no)

```
# Understand unique value of column G3
df.G3.unique()
```

```
array([ 6, 10, 15, 11, 19,  9, 12, 14, 16,  5,  8, 17, 18, 13, 20,  7,  0,
        4])
```

```
# Creating function which categorize data according to G3.

def category(G3):
    if G3 >= 8:
        return "yes"
    elif G3 < 8:
        return "no"

# Applying the transformed value based on the value of G3
df["passed"] = df["G3"].apply(category)
```

Figure 34: Understanding unique value of G3, Creating function to categorize the data and applying the transofrmed value

```
# G3 Value
df[["G3", "passed"]]
```

	G3	passed
0	6	no
1	6	no
2	10	yes
3	15	yes
4	10	yes
...	...	...
390	9	yes
391	16	yes
392	7	no
393	10	yes
394	9	yes

395 rows × 2 columns

Figure 35: Value of G3

3. Initial Data Analysis

3.1 Summary Statistics

Summary statistics (sum, mean, median, standard deviation, max and min) of the variables age, absences, G1, G2 and G3.

```
# Create function to show summary statistics.
def calculate_describe(col, keyword):
    col_sum  = col.sum() # Sum
    col_mean = col.mean() # Mean
    col_median = col.median() # Median
    col_std =  col.std() # Standard deviation
    col_max = col.max() # Max
    col_min = col.min() # Min

    print(f"Total sum of {keyword} is: {col_sum}")
    print(f"Mean of {keyword} is: {col_mean}")
    print(f"Median of {keyword} is: {col_median}")
    print(f"Standard deviation of {keyword}: {col_std}")
    print(f"Maximum value of {keyword} is: {col_max}")
    print(f"Minimum value of {keyword} is: {col_min}")
```

Figure 36: Creating function to show summary statistics

3.1.a Variable 'age'

```
col = df.age # Dataframe
keyword = 'age' # Keyword used on print

calculate_describe(col, keyword) # Printing Summary
```

```
Total sum of age is: 6595
Mean of age is: 16.696202531645568
Median of age is: 17.0
Standard deviation of age: 1.2760427246056245
Maximum value of age is: 22
Minimum value of age is: 15
```

Figure 37: Statistics of age

- Total sum of age is: 6595
- Mean of age is: 16.696202531645568
- Median of age is: 6595
- Standard deviation of age: 1.2760427246056245
- Maximum age of student is: 22
- Minimum age of student is: 15

3.1.b Variable 'absences'

```
col = df.absences # Dataframe
keyword = 'absences' # Keyword used on print

calculate_describe(col, keyword) # Printing Summary
```

```
Total sum of absences is: 2255
Mean of absences is: 5.708860759493671
Median of absences is: 4.0
Standard deviation of absences: 8.003095687108177
Maximum value of absences is: 75
Minimum value of absences is: 0
```

Figure 38: Statistics of absences'

- Total sum of absences is: 2255
- Mean of absences is: 5.708860759493671
- Median of absences is: 4.0
- Standard deviation of absences: 8.003095687108177
- Maximum value of absences is: 75
- Minimum value of absences is: 0

3.1.c Variable 'G1'

```
col = df.G1 # Dataframe
keyword = 'G1' # Keyword used on print

calculate_describe(col, keyword) # Printing Summary
```

```
Total sum of G1 is: 4309
Mean of G1 is: 10.90886075949367
Median of G1 is: 11.0
Standard deviation of G1: 3.3191946715076686
Maximum value of G1 is: 19
Minimum value of G1 is: 3
```

Figure 39: Statistics of G1

- Total sum of 'G1' is: 4309
- Mean of 'G1' is: 10.90886075949367
- Median of 'G1' is: 11.0
- Standard deviation of 'G1' is: 3.3191946715076686
- Maximum value of 'G1' is: 19
- Minimum value of 'G1' is: 3

3.1.d Variable of 'G2'

```
col = df.G2 # Dataframe
keyword = 'G2' # Keyword used on print

calculate_describe(col, keyword) # Printing Summary
```

```
Total sum of G2 is: 4232
Mean of G2 is: 10.713924050632912
Median of G2 is: 11.0
Standard deviation of G2: 3.761504659556034
Maximum value of G2 is: 19
Minimum value of G2 is: 0
```

Figure 40: Statistics of G2

- Total sum of 'G2' is: 4232
- Mean of 'G2' is: 10.713924050632912
- Median of 'G2' is: 11.0
- Standard deviation of 'G2' is: 3.761504659556034
- Maximum value of 'G2' is: 19
- Minimum value of ' G2' is: 0

3.1.e Variable 'G3'

Summary statistics of variable 'G3'

```
col = df.G3 # Dataframe
keyword = 'G3' # Keyword used on print

calculate_describe(col, keyword) # Printing Summary
```

```
Total sum of G3 is: 4114
Mean of G3 is: 10.415189873417722
Median of G3 is: 11.0
Standard deviation of G3: 4.5814426109978434
Maximum value of G3 is: 20
Minimum value of G3 is: 0
```

Figure 41: Statistics of G3

3.2 Calculation and Correlation Between Variables

Correlation is a statistic that indicates the linear relationship between two variables. There is inbuilt function in pandas known as **dataframe.corr()** which will find the pairwise correlation of all columns in dataframe.

Calculate and correlation between the variables absences, failures, G1, G2 and G3. Present the result using a heatmap and interpret the results.

```
# Create the correlation matrix
d = df[['absences', 'failures', 'G1', 'G2', 'G3']]
corr = d.corr()

corr # Printing Data
```

	absences	failures	G1	G2	G3
absences	1.000000	0.063726	-0.031003	-0.031777	0.034247
failures	0.063726	1.000000	-0.354718	-0.355896	-0.360415
G1	-0.031003	-0.354718	1.000000	0.852118	0.801468
G2	-0.031777	-0.355896	0.852118	1.000000	0.904868
G3	0.034247	-0.360415	0.801468	0.904868	1.000000

Figure 42: Coorelation matrix

```
# Set up the matplotlib figure
figsize = plt.figure(figsize = (11,9))

heatmap = sns.heatmap(
    corr, # The data to plot
    cmap = "YlGnBu",
    annot = True
)

heatmap.set_title('Correlation Heatmap between absences, failures, G1, G2
G3', fontsize =16, loc='left') # Title with font size

plt.show() # Display a figure
```

Figure 43: Generate heatmap

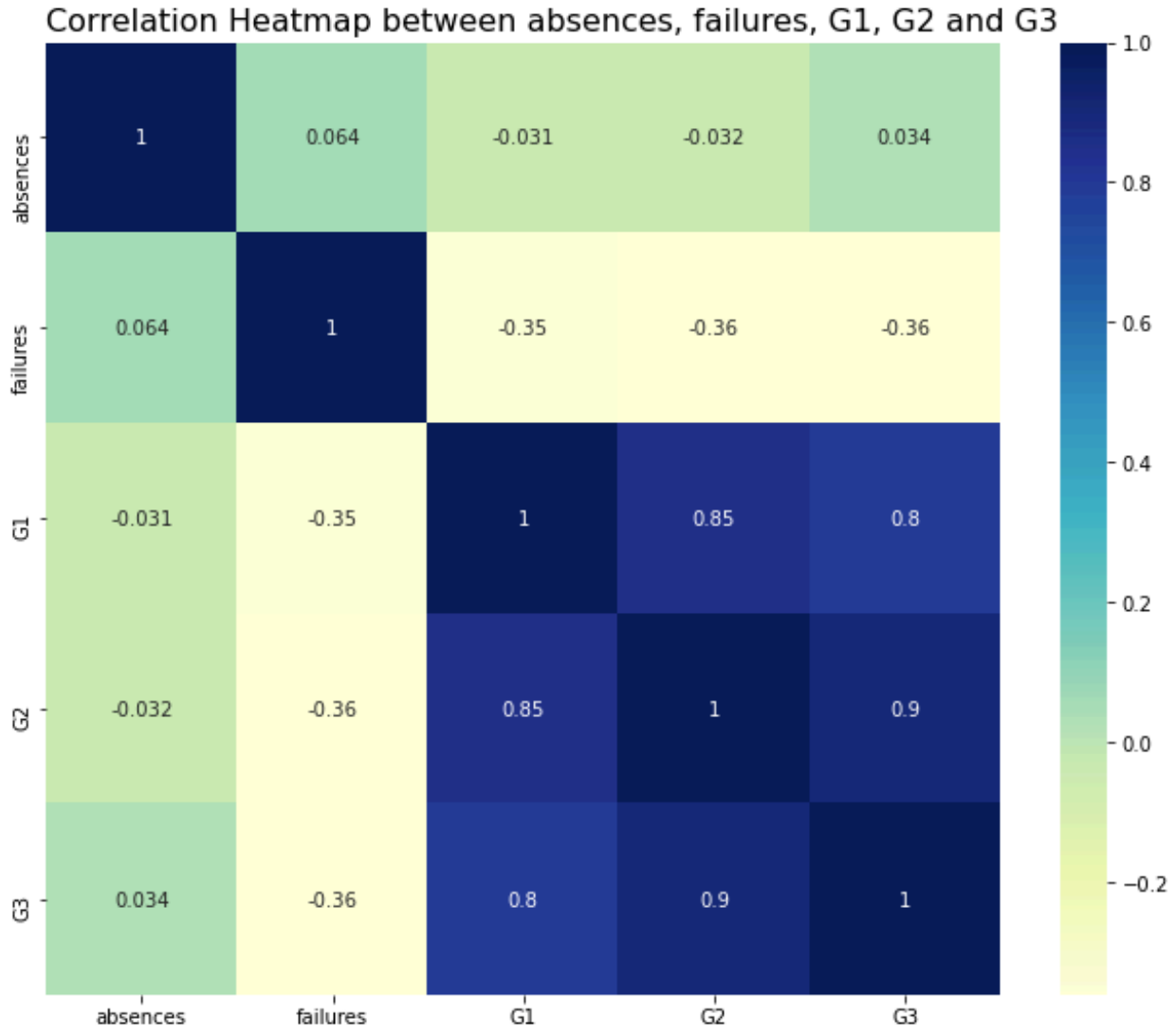

Figure 44: Heatmap correlation between variables (absences, failures, G1, G2 and G3)

The correlation between five different variables (absences, failures, G1, G2, and G3) is depicted in the heatmap above. If the colour is dark, there is a strong correlation between the variables; if the colour is light, there is a weak correlation between the variables. Thus, we can see that G3 and G2 have a strong correlation because the colour representing them is dark, whereas absences and G2 have a weak correlation because the colour representing them is light in comparison to the others.

4. Data Exploration and Visualization

4.1 Histogram Plots and Boxplots

Histogram plots and boxplots to visualize the distribution of the variables age, absences and G3. Interpretation of the results and comment about the distribution of each variable.

4.1.a Histogram plot for 'age'

```
# Set up the histogram figure size
plt.figure(figsize = (11,9))

start = df['age'].unique().min() # Minimum Age
stop = df['age'].unique().max() + 1 # Maximum Age +1 is list highest.

ages_list = [i for i in range(start, stop)]

plt.hist(
    df['age'],
    edgecolor = "white",
    linewidth = 2,
    color = 'skyblue',
    bins = ages_list
)

plt.xticks(ages_list)

# Title of Histogram
plt.title("Histogram Plot for 'age'", fontsize=16, loc='left')

# X Label
plt.xlabel("Student's Age")

# Y Label
plt.ylabel("Student's Number")

plt.show() # Display a figure
```

Figure 45: Generating histogram plot

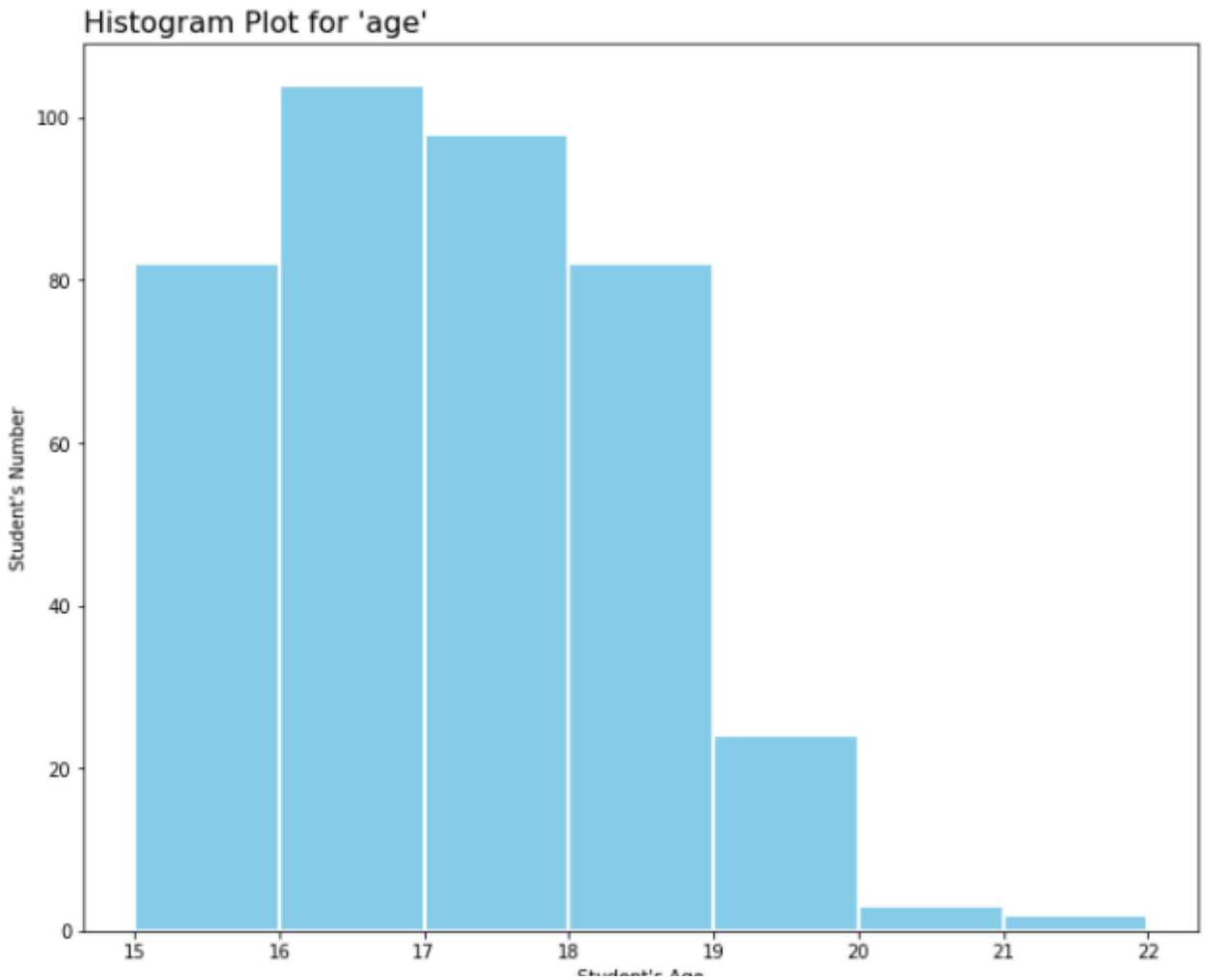

Figure 46: Histogram plot for 'age'

Interpretation: From above histogram plot we can see that the majority of students age is in between 16 and 18 and minority of age is from 20 to 22.

4.1.b Box plot for 'age'

```
# Set up the box plot figure
plt.figure(figsize = (11,9))

sns.boxplot(x = df["age"], color='skyblue')

# Title
plt.title("Box Plot for 'age'", fontsize=16, loc='left')

# X Label
plt.xlabel("Student's Age")

plt.show() # Display a figure
```

Figure 47: Generating box plot of age

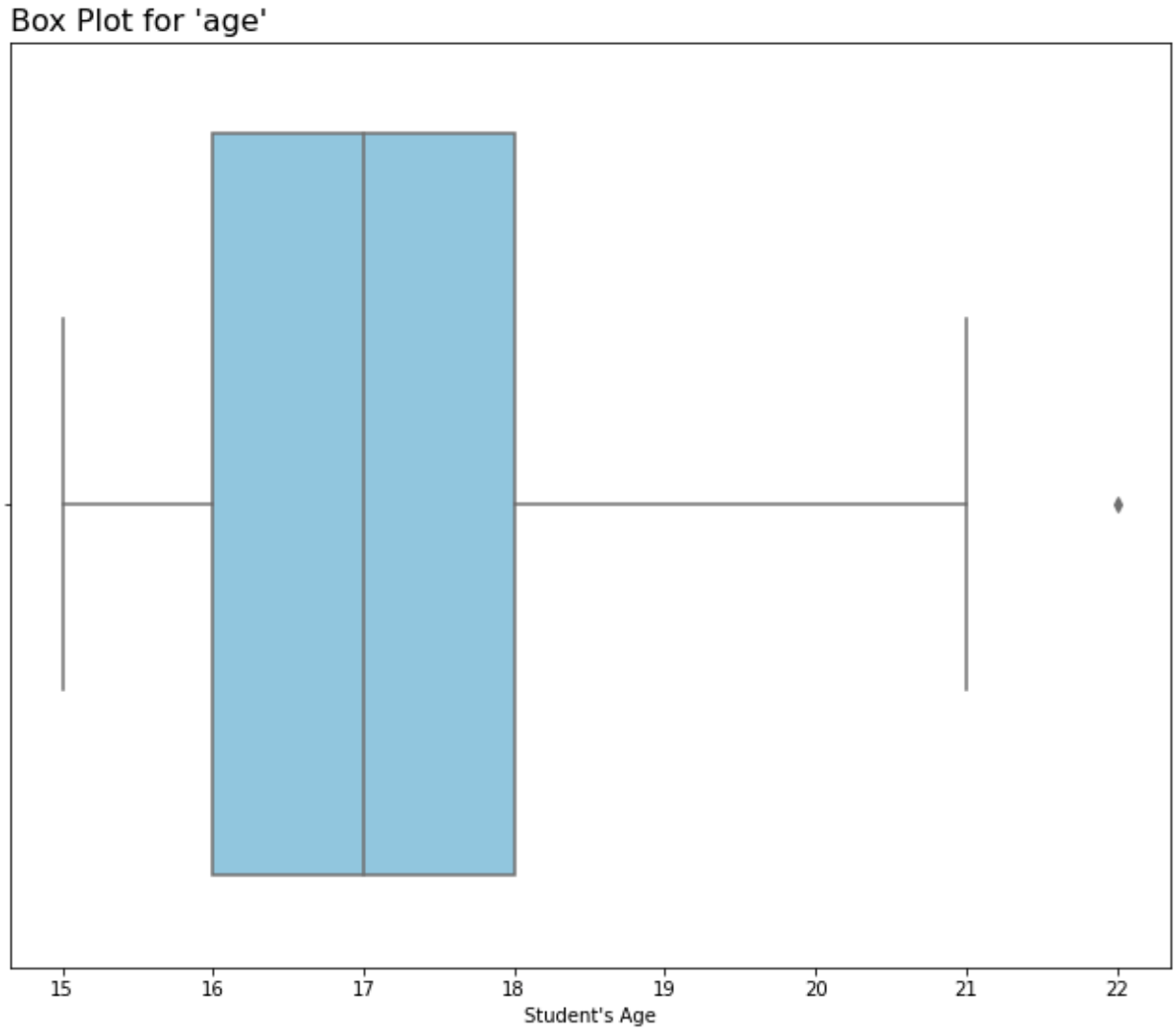

Figure 48: Box plot for 'age'

Interpretation: The above boxplot shows us that the minimum age of student in the school is 15 and the first quartile start from age of 16 and median is 17. Third quartile lies in age of 18 and maximum age is 22.

4.1.c Histogram plot for ‘absences’

```
# Set up the histogram figure
plt.figure(figsize = (11,9))

start = df['absences'].unique().min()
stop = df['absences'].unique().max()+10

absences_list = [i for i in range(start, stop, 5)]

plt.hist(
    df['absences'],
    edgecolor = "white",
    linewidth = 1,
    color = 'skyblue',
    bins = absences_list
)

plt.xticks(absences_list)

# Title of Histogram
plt.title("Histogram Plot for 'absences'", fontsize=14, loc='left')

# X Label
plt.xlabel("Student's Absense")

# Y Label
plt.ylabel("Student's Number")

plt.show() # Display a figure
```

Figure 49: Generating Histogram plot for 'absences'

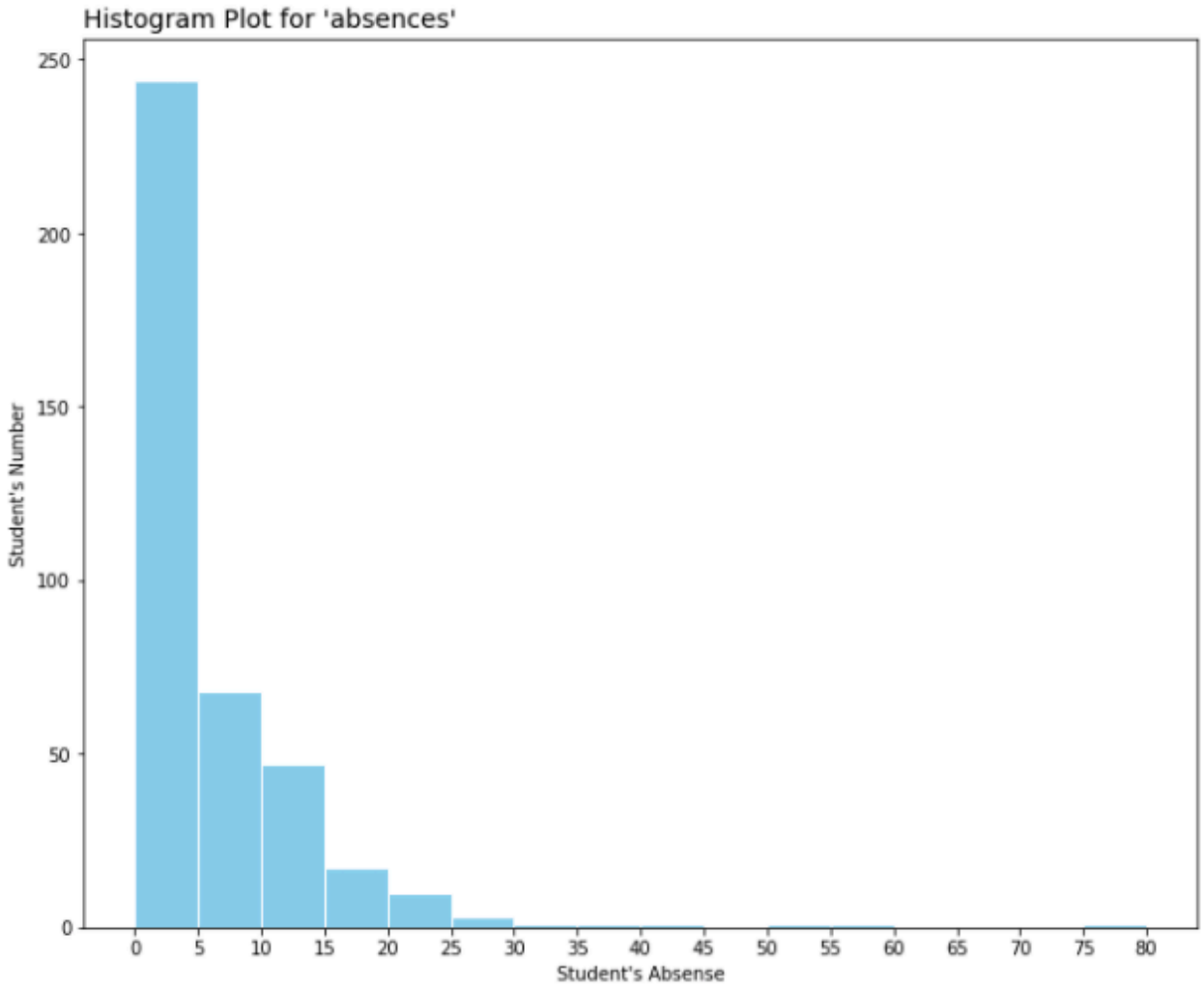

Figure 50: Histogram plot for 'absences'

Interpretation: From above histogram plot we can see that the majority of students are absent for around 5 days and minority of students are absent for 75 to 80 days.

4.1.d Box plot for ‘absences’

```
# Set up the histogram figure
plt.figure(figsize = (11,9))

# Title
plt.title(
    "Box Plot for 'absences'",
    fontsize = 14,
    loc = "left"
)

sns.boxplot(
    x = df["absences"],
    color = "skyblue"
) # Box Plot

plt.show() # Display a figure
```

Figure 51: Generating box plot for 'absences'

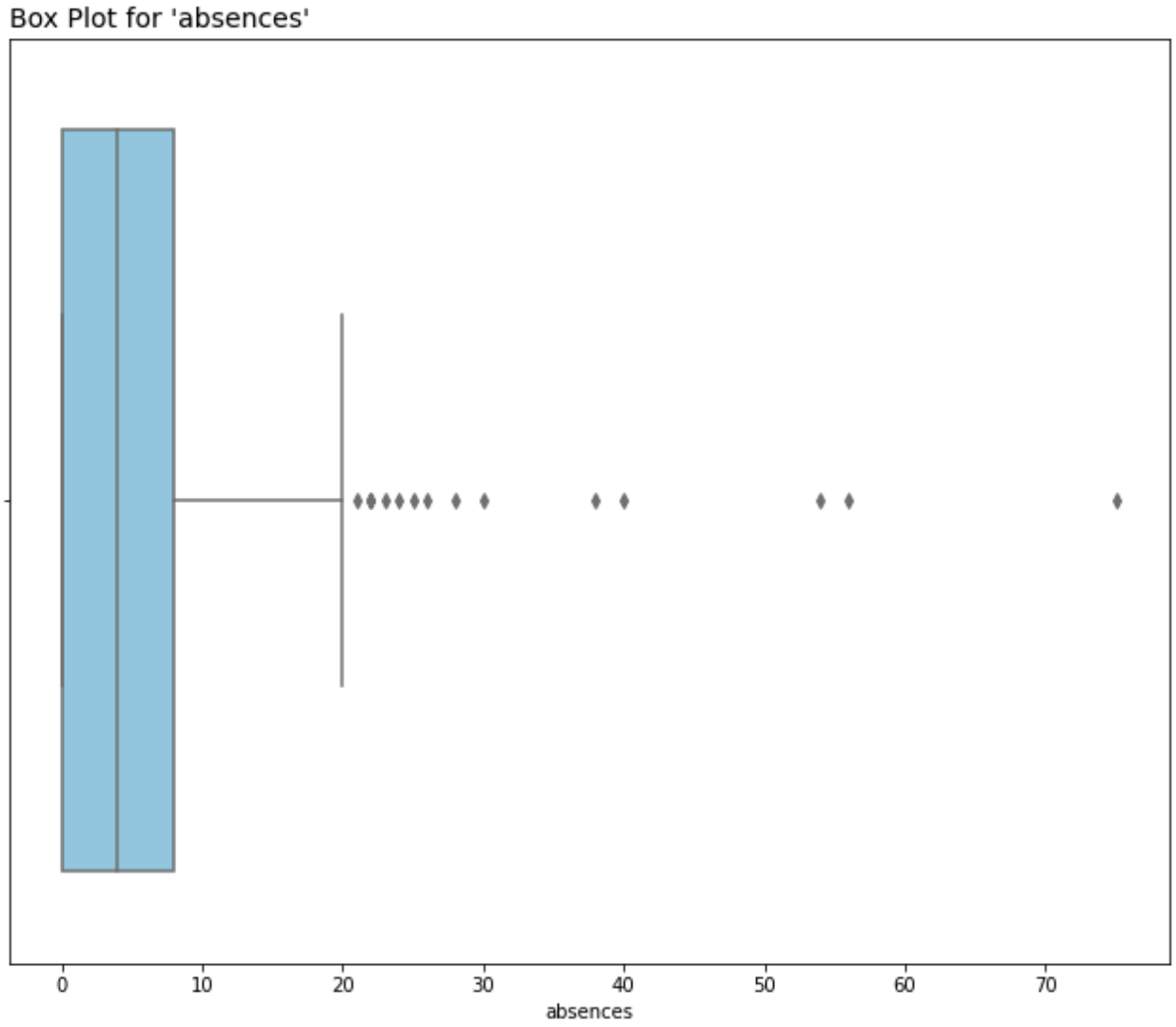

Figure 52: Box plot for 'absences'

Interpretation: From above boxplot we can see that minimum number of absent days of student is 0 whereas median value, first quartile and third quartile lie between 0 and 10 and maximum value is above 70.

4.1.d Histogram plot for ‘G3

```
# Set up the histogram figure
plt.figure(figsize = (11,9))

start = df['G3'].unique().min()
stop = df['G3'].unique().max()+10

absences_list = [i for i in range(start, stop, 5)]

plt.hist(
    df['G3'],
    edgecolor = "white",
    linewidth = 1,
    color = 'skyblue',
    bins = absences_list
)

plt.xticks(absences_list)

# Title of Histogram
plt.title("Histogram Plot for 'G3'", fontsize=14, loc='left')

# X Label
plt.xlabel("Student's Final Grade")

# Y Label
plt.ylabel("Student's Number")

plt.show() # Display a figure
```

Figure 53: Generate histogram plot for G3

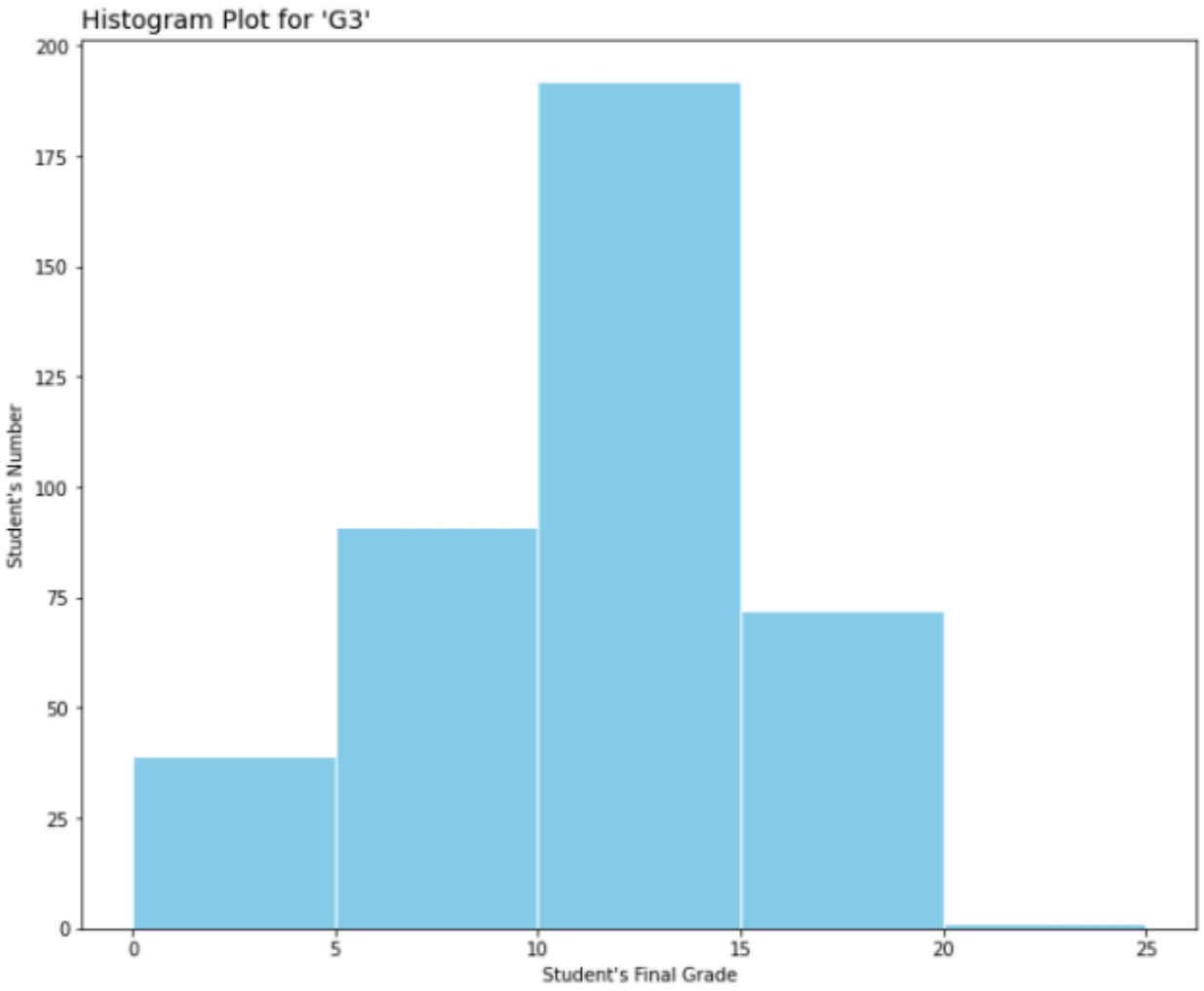

Figure 54: Histogram plot for G3

Interpretation: From above histogram plot we can see that the majority of students have secured final grades between 10 and 15 whereas there are few numbers of students who have secured grades between 20 and 25.

4.1.d Box plot for ‘G3

```
# Set up the histogram figure
plt.figure(figsize = (6,9))

plt.title("Box Plot for 'G3'", fontsize=14, loc='left') # Title

sns.boxplot(x=df["G3"], color='skyblue') # Box Plot

plt.show() # Display a figure
```

Figure 55: Generating box plot for G3

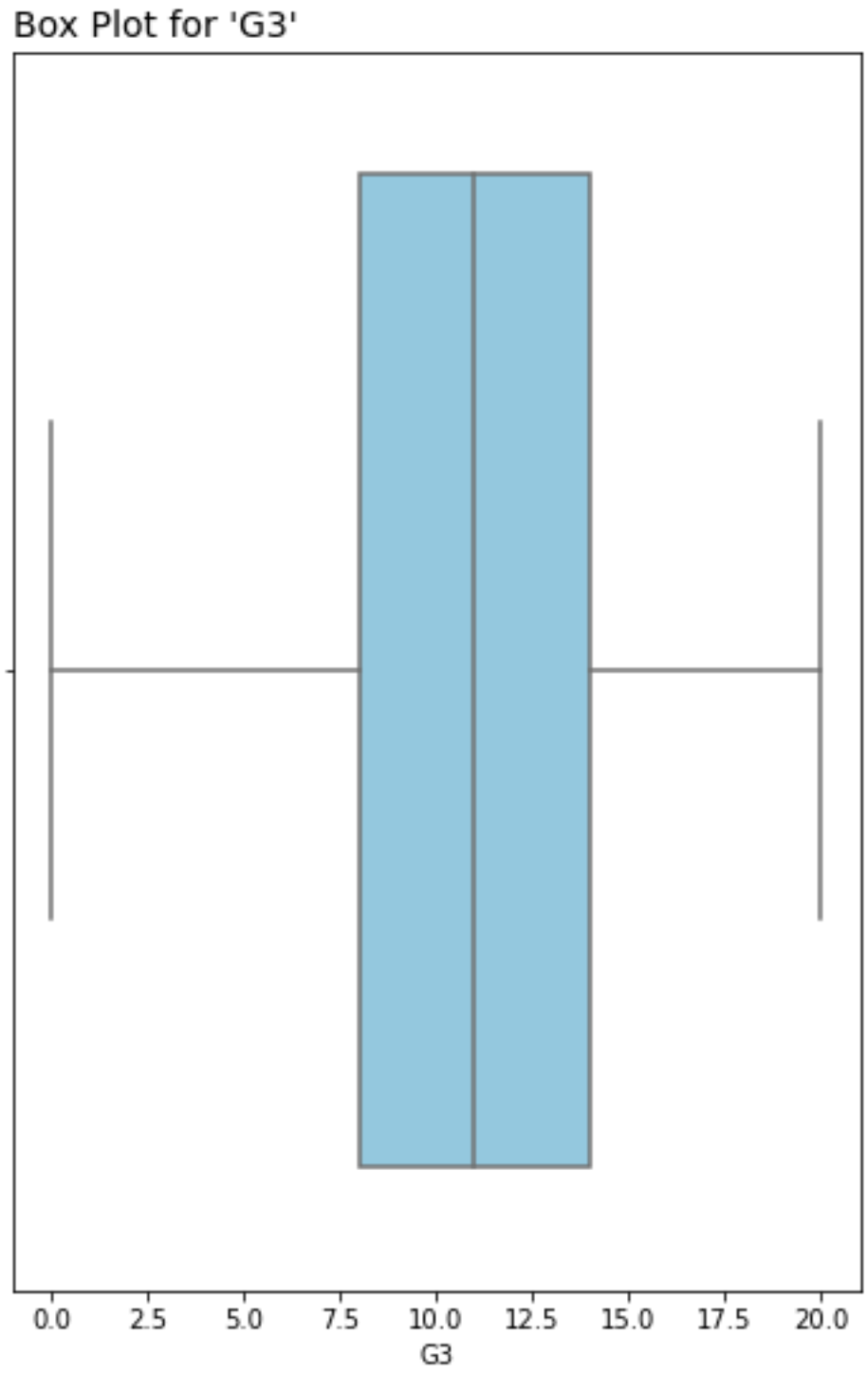

Figure 56: Box plot for G3

Interpretation: From above boxplot we can see that minimum number of grades secured by student is 0 whereas median value, first quartile and third quartile lie between 7.5 and 15 and maximum grade secured is 20.

4.2 Bar Graph of Passed Students Grouped by School Name

Total students passed the final term grouped according to the school they belongs.

```
plt.figure(figsize=(6,6)) # Define figure size

df[df['passed']=='yes'].groupby('school')['G3'].count().plot(kind ="bar",
edgecolor = "white", color = "skyblue")

# Title
plt.title("Number of passed student in final term", fontsize=14, loc='left

plt.xlabel("School Name") # X label

plt.ylabel("Number of Passed Students") # Y label

plt.show() # Display a figure
```

Figure 57: Generating passed students and grouping

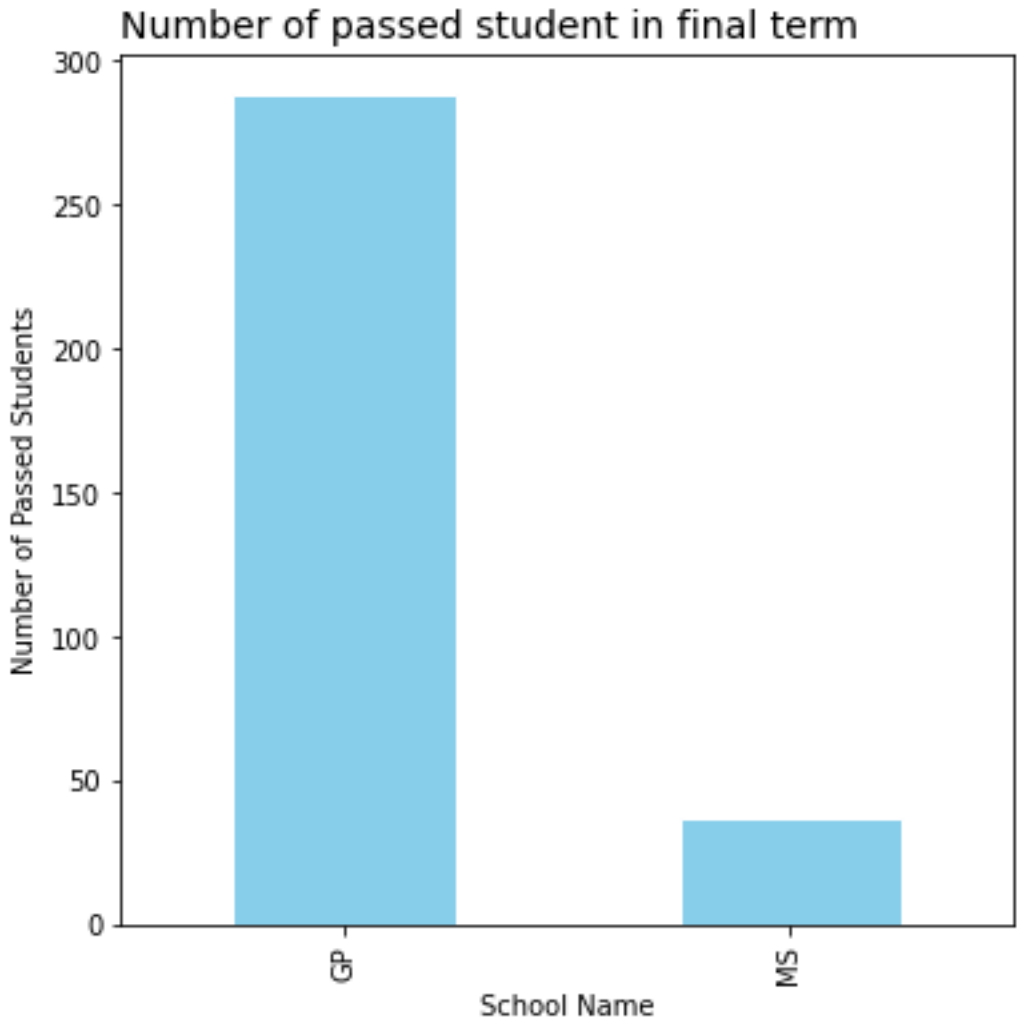

Figure 58: 4.2 Bar Graph of Passed Students

Interpretation: The above bar graph provides us the information about the number of students who has secured passing grades in both GP and MS school. Here, the students from GP has secured more passing grades compared to MS.

4.3 Bar Graph of Failed Students Based on Weekly Study Time

Total number of students who failed the final term grouped according to their weekly study time.

```
plt.figure(figsize=(11,9)) # Define figure size

df[df['passed']=='no'].groupby('studytime')['G3'].count().plot(kind ="bar"
edgecolor = "white", color = "skyblue")

# Title
plt.title("Number of students who failed in final term based on weekly stu
time", fontsize=14, loc='left')

plt.xlabel("Student's weekly study time") # X label

plt.ylabel("Student's failed number") # Y label

plt.show() # Display a figure
```

Figure 59: Generating bar graph of failed students based on study time.

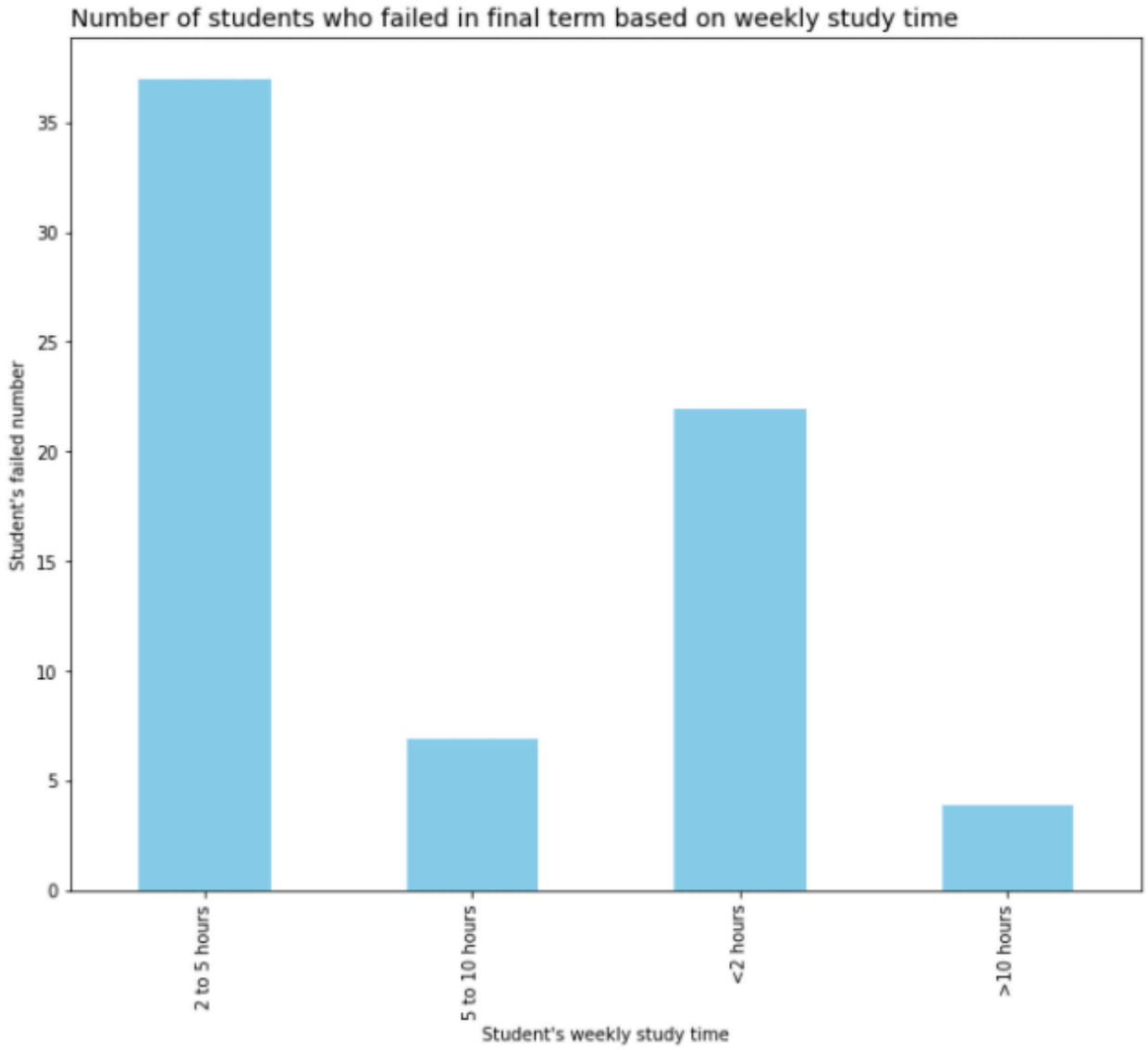

Figure 60: bar graph of failed students based on study time

Interpretation: According to above bar graph we can see that the maximum number of students who have the studytime from 2 to 5 hours per week are failing their subjects. Whereas the students who have studytime of more than 10 hours are failing less.

5. Further Analysis

Most parents care about their childern's academic performance oppose to what they want. Here are the understanding and analysis based on the student's grades, socioeconomic, social, and school-related statistics.

5.1 Pass Rate in GP and MS School

To perform analysis and plotting in pie chart the values of passed and failed student from GP and MS are grouped together and saved in dictionary with key as school name and value as passed student of that particular school.

5.1.a Grouping values of GP and MS based on passed

```
# Calculate total passed student based on school
tpass = df[df['passed'] == 'yes'].groupby('school')['G3'].count()
tpass
```

```
school
GP    288
MS     37
Name: G3, dtype: int64
```

Figure 61: Find total pass on both school

5.1.b Assigning key to school name and creating dictionary with key value pair for passed number of students¶

```
# Assign key to school name
df_tpass = {}
keynum = 0
for i in tpass.index:
    df_tpass[keynum] = tpass[i]
    print(i, "is", keynum)
    keynum += 1
df_tpass
```

```
GP is 0
MS is 1
{0: 288, 1: 37}
```

Figure 62: Assign key value pair for pass

5.1.c Grouping values of GP and MS based on failed

```
# Calculating total number failed student based on school
tfail = df[df['passed'] == 'no'].groupby('school')['G3'].count()
tfail
```

```
school
GP    61
MS     9
Name: G3, dtype: int64
```

Figure 63: Find total fail on both school

5.1.d Pass and failed Rate in GP (Gabriel Pereira) School

```
plt.figure(figsize=(5,5)) # Define figure size

# Create a set of colors
colors = ['#99CCFF', '#ffcc99']

# Pie chart showing pass and fail rate of GP
plt.pie([df_tpass[0], df_tfail[0]],
        labels = ["Pass","Fail"],
        autopct = "%.2f%%",
        radius = 1.5,
        wedgeprops = { 'linewidth' : 3, 'edgecolor' : 'white' },
        colors = colors
       )

plt.show() # Display a figure
```

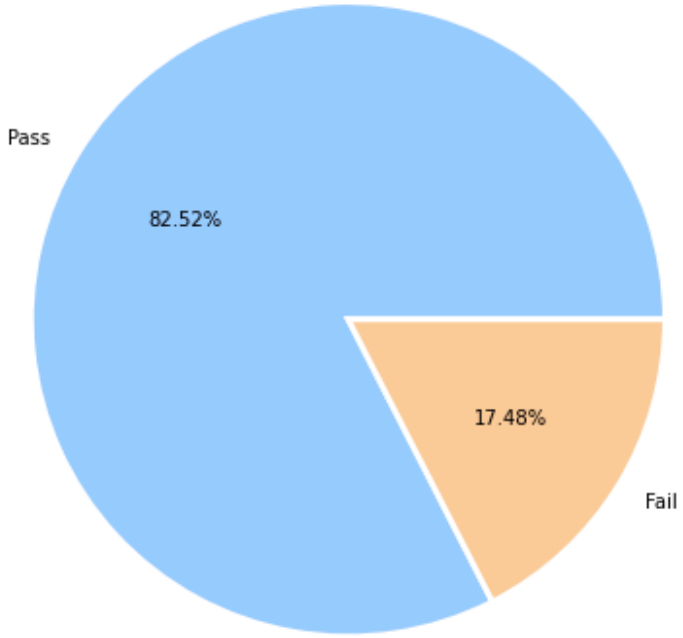

Figure 64: Piechart - Pass and failed Rate in GP (Gabriel Pereira) School

5.1.e Pass and failed Rate in MS (Mousinho da Silveira) School

```
plt.figure(figsize=(5,5)) # Define figure size

# Pie chart showing pass and fail rate of GP
plt.pie([df_tpass[1], df_tfail[1]],
        labels = ["Pass","Fail"],
        autopct = "%.2f%%",
        radius = 1.5,
        wedgeprops = { 'linewidth' : 3, 'edgecolor' : 'white' },
        colors = colors
       )

plt.show() # Display a figure
```

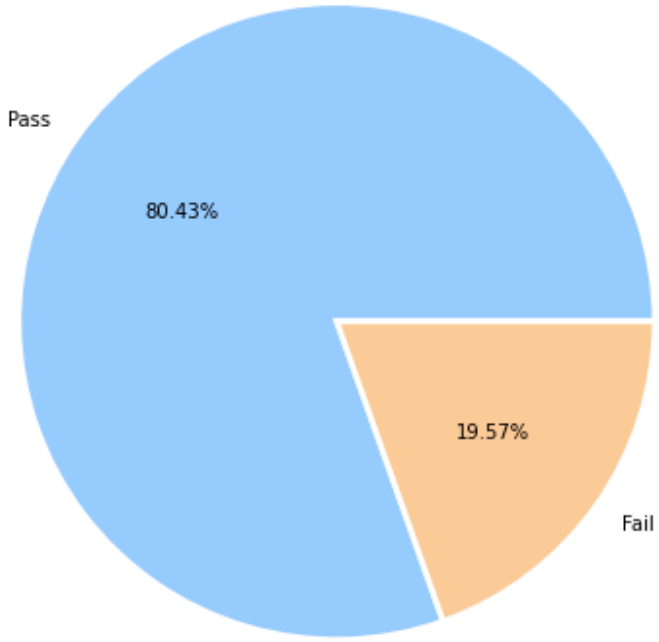

Figure 65: Piechart - Pass and failed Rate in MS (Mousinho da Silveira) School

Conclusions: According to the above findings, GP school's pass rate is higher than its MS school's pass rate, while MS school's fail rate is higher than GP school's fail rate. In light of the findings presented here, we can conclude that the best students come from GP.

5.2 Grade Of Students as per the Gender

Calculating number of passed students based on their gender

```
# Calculating total number passed student based on gender
gpass = df[df['passed']=='yes'].groupby('sex')['G3'].count()
gpass
```

```
sex
F    164
M    161
Name: G3, dtype: int64
```

Converting above result into array for further analysis

```
# Converts into numpy array
np.array(list(df[df['passed']=='yes'].groupby('sex')['G3'].count()))

# Converts into numpy array
np.array(list(df[df['passed']=='yes'].groupby('sex')['G3'].count().index))
```

```
array(['F', 'M'], dtype='<U1')
```

Calculating number of failed students based on their gender

```
# Calculating total number failed student based on gender
gfail = df[df['passed']=='no'].groupby('sex')['G3'].count()
gfail
```

```
sex
F    44
M    26
Name: G3, dtype: int64
```

Figure 66: Grouping and assigning key value pair of students grade per gender

Gender-specific final grade values are grouped together and then converted into a set of key value pairs for plotting on a bar graph. Using the provided dataset, the presentation plots the final grades earned by females and males.

```
# Bar graph showing passed and fail number based on gender
plt.figure(figsize=(6,6))

plt.bar(np.array(list(df[df['passed']=='yes'].groupby(new_col_sex['new_sex
['G3'].count().index)),
        np.array(list(df[df['passed']=='yes'].groupby(new_col_sex['new_sex
['G3'].count())), edgecolor = "white", linewidth = 2, color ="skyblue",
width=0.5, label="Pass")

plt.bar(np.array(list(df[df['passed']=='no'].groupby(new_col_sex['new_sex'
['G3'].count().index))+ 0.5,
        np.array(list(df[df['passed']=='no'].groupby(new_col_sex['new_sex'
['G3'].count())),
        color="orange", edgecolor = "white", linewidth = 2, width=0.5,
label="Fail")

plt.xticks(np.array(list(df[df['passed']=='yes'].groupby(new_col_sex['new_
)['G3'].count().index))+ 0.15,["Female","Male"])


# Title
plt.title("Number of failed students based on gender ", fontsize=14,
loc='left')

plt.xlabel("Gender")

plt.ylabel("Number of Students")

plt.legend()

plt.show() # Display a figure
```

Figure 67: Create bar graph final grade based on gender.

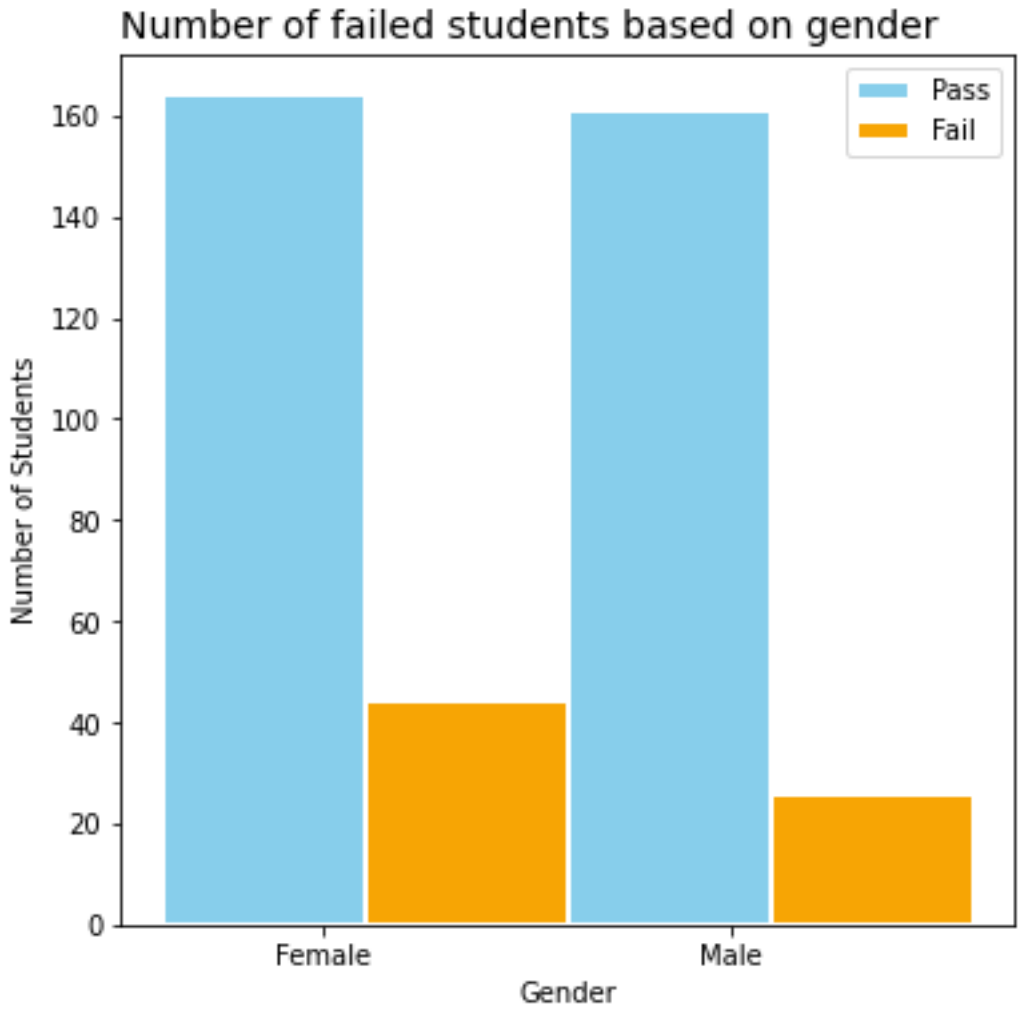

Figure 68: Bar graph to show failed students based on gender

Conclusions: We can see from the bar graph above that in both schools the number of passed students per male and female is high, but the female passed rate is slightly higher than the males. When compared to the female failure rate, the male failure rate is nearly half as high. Accordingly, we can conclude that male students are slightly more successful in the classroom when compared to female students' passing and failing rates.

5.3 Family Background Affect on Students Final Grades

Grade of students as per father's job

```
# Countplot showing passed and fail number based on father's occupation
plt.figure(figsize=(11,9))

foccup = sns.countplot(
    x="Fjob",
    data = df,
    hue="passed",
    palette="tab20b_r"
)

plt.xlabel("Father Occupation")
plt.ylabel("Number of Students")
plt.show()
```

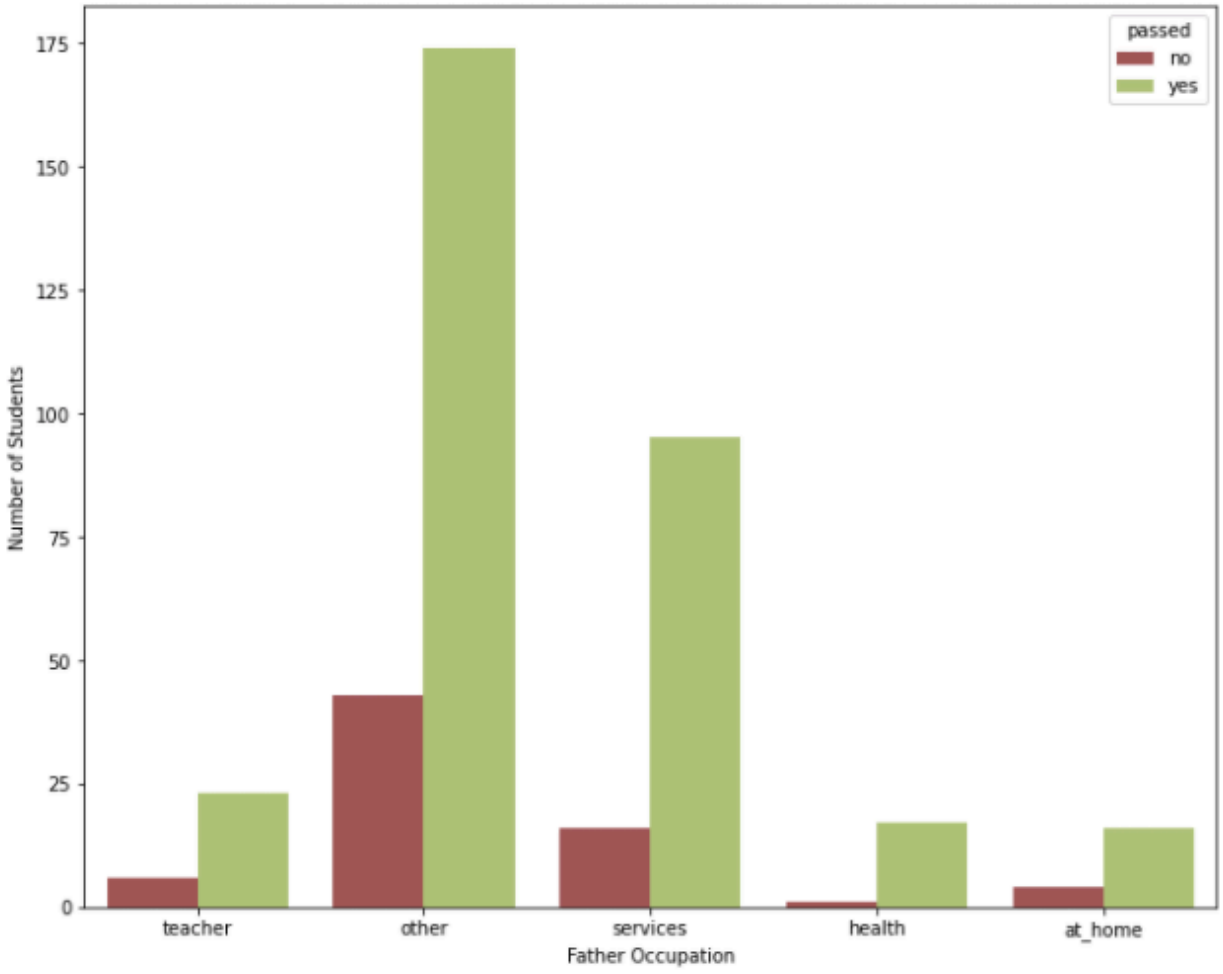

Figure 69: Countplot of passed and failed students per father's occupation

Conclusions: The countplot presentation of the number of students who passed or failed based on their father's job can be seen in the above code. Students who passed and failed are more likely to have fathers who work in other sectors, but students who passed are less likely to have fathers who work in the health sector or don't have any jobs at all. However, when considering all job categories, the percentage of students who fail is lower for those whose fathers work in the health care industry.

Grade of students as per mother's job

```
# Countplot showing passed and fail number based on father's occupation
plt.figure(figsize = (11,9))

foccup = sns.countplot(
    x = "Mjob",
    data = df,
    hue = "passed",
    palette = "coolwarm"
)

plt.xlabel("Mother Occupation")
plt.ylabel("Number of Students")
plt.show()
```

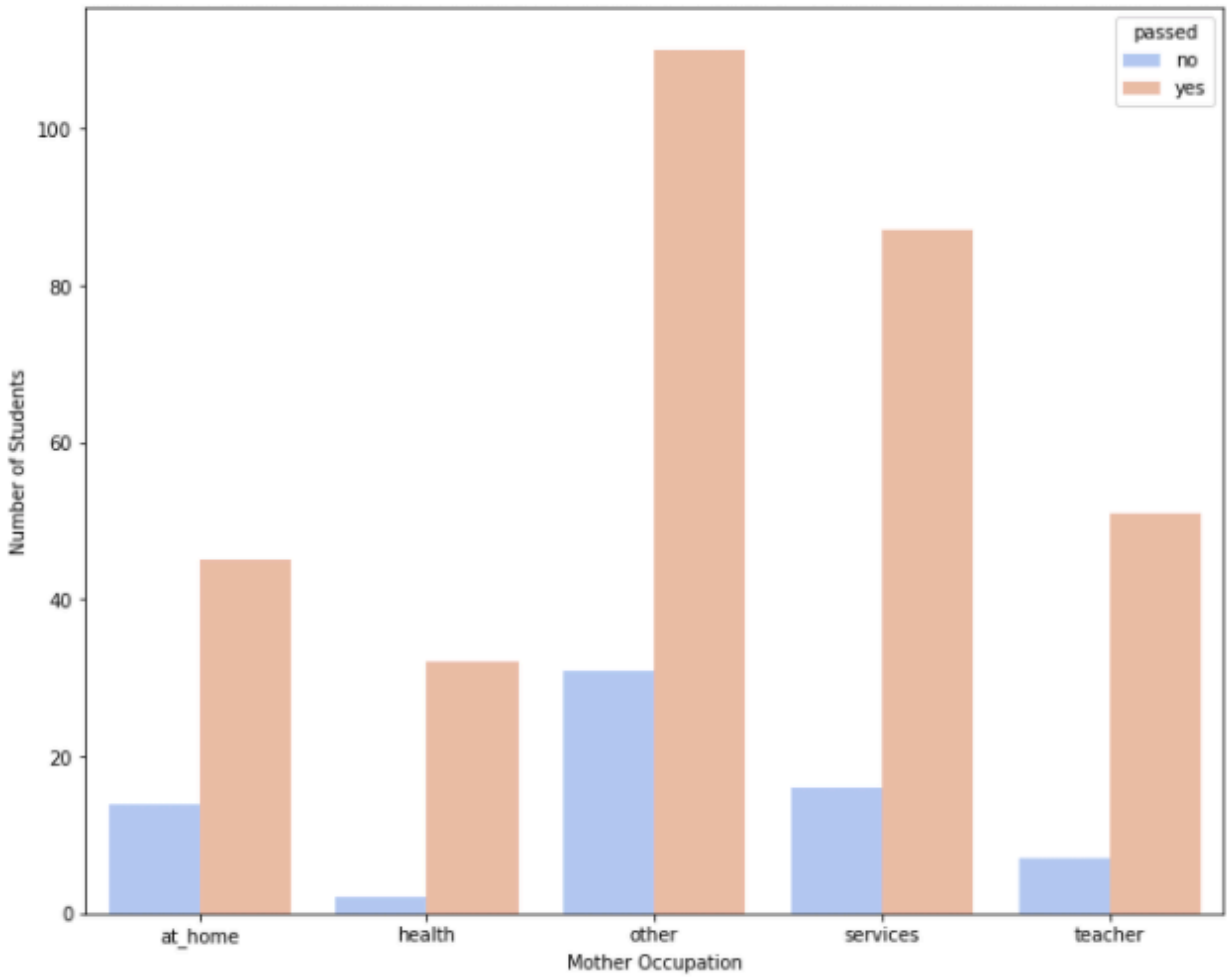

Figure 70: Countplot of passed and failed students per mother's occupation

Conclusions: The above code displays the number of students who passed or failed based on their mother's job in a countplot format. We can see from the countplot above that students whose mothers work in other categories have more passed and failed records and students whose mothers work in health categories have fewer passed and failed records. Services-worker students are slightly more likely to pass and fail than those from other occupations. Finally, the number of students who passed or failed has been higher for fathers and mothers who work in other fields.

5.4 Parents Relationship Status Affect on Students Pass Rate

```
# Countplot showing passed and fail number based on parent's relationship
status
plt.figure(figsize =( 11,9))
par_rel = sns.countplot(
    x = "famrel",
    data = df,
    hue = "passed",
    palette = "BuPu",
    edgecolor = "black"
)

par_rel.set(
    xlabel = "Family Relationship",
    ylabel = "Number of Students"
)

plt.show()
```

Figure 71: Code to countplot based on parent's relationship status

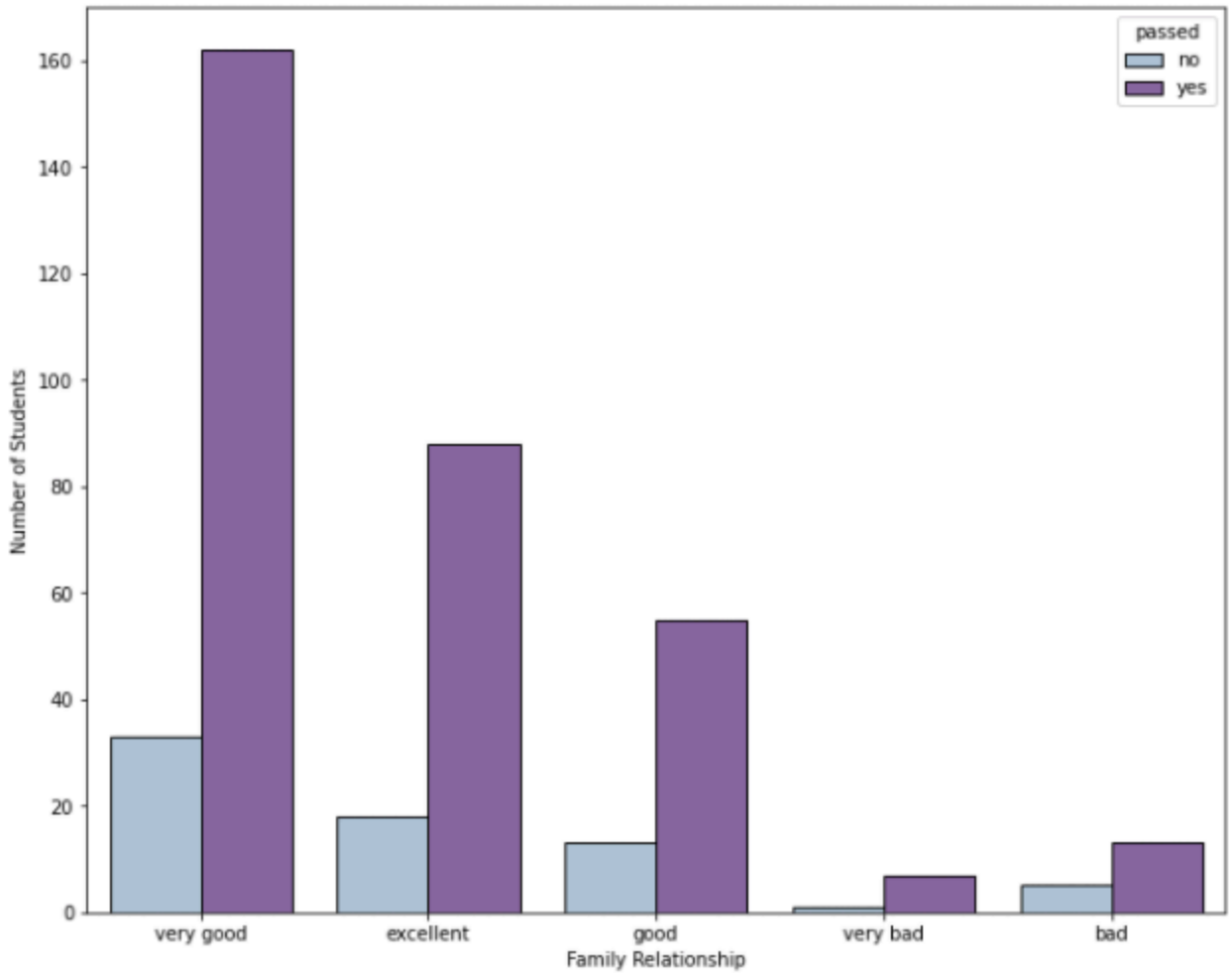

Figure 72: Countplot based on Family relationship status

Conclusions: Students with parents who are considered to have a perfect relationship have a higher percentage of passing and failing grades, as can be seen. In contrast, the pass/fail record of students with excellent parental relationship status lags behind that of students with excellent parental relationship status. Parents with a poor relationship status, on the other hand, have a low pass/fail record. Finally, we can say that a student's final grade is directly influenced by his or her relationship with his or her parents.

5.5 Parents Education Background Affect on Students Pass Rate

```
# Count Plot showing passed and fail number based on father's & mother's
education
f, axes = plt.subplots(2,1, figsize = (11,9))

passgroup_one=sns.barplot(
    y = "Medu",
    x = "G3",
    data = df,
    hue = "passed",
    ax = axes[0],
    ci = None,
    palette = "rainbow"
)

passgroup_one.set(
    xlabel = "Final Grade",
    ylabel = "Mother Education Details"
)

passgroup_two = sns.barplot(
    y="Fedu",
    x="G3",
    data=df,
    hue="passed",
    ax=axes[1],
    ci=None,
    palette="spring"
)

passgroup_two.set(
    xlabel="Final Grade",
    ylabel="Father Education Details"
)

plt.subplots_adjust(top=0.9)
plt.show()
```

Figure 73: Code to plot pass and fail based on parents education background.

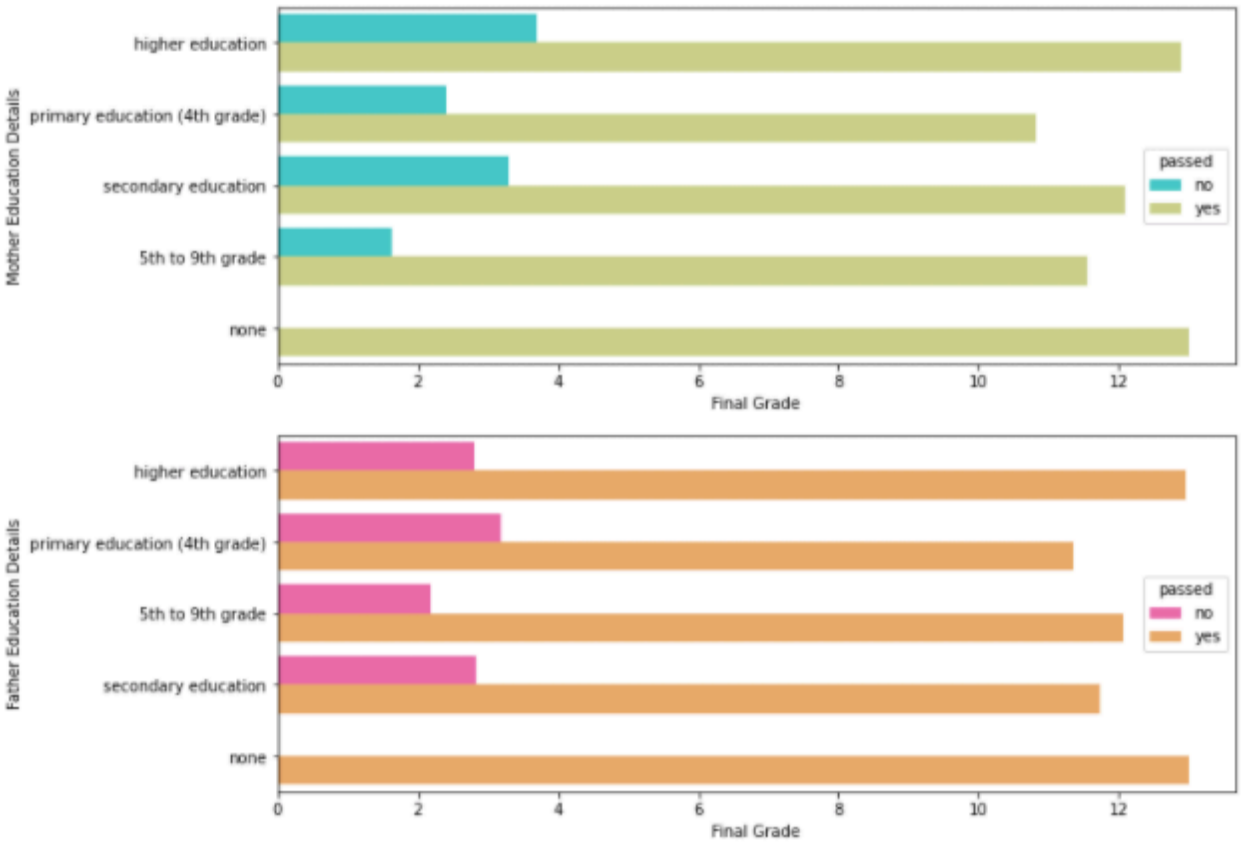

Figure 74: Barplot of pass and fail student based on mother's eduction (top) and father's education (bottom)

Conclusions: According to the barplots above, students whose parents' education level falls into the "none" category have a higher passing rate than those whose parents' education level falls into the "higher" category. In addition, the number of students who fail when their fathers have only a primary education is higher than in other categories. Looking at the mother's educational background, the failure rate for students whose mother has a college degree is relatively high.

5.6 Affect of Frequency of Going Out on Final

```
"""
Bar Plot showing passed and fail number based on frequency
students going out
"""
pct = (lambda col: col/col.sum())
index = ['yes', 'no']
out_tab = pd.crosstab(index = df.passed, columns = df.goout)
out_pct = out_tab.apply(pct).reindex(index)

out_pct.plot.bar(
    colormap = "Spectral",
    figsize = (11,9),
    edgecolor = 'white'
)

plt.title("Final Grade By Frequency of Going Out", fontsize = 14)
plt.ylabel('Percentage of Student')
plt.xlabel('Passed')
plt.show()
```

Figure 75: Code frequency of going out.

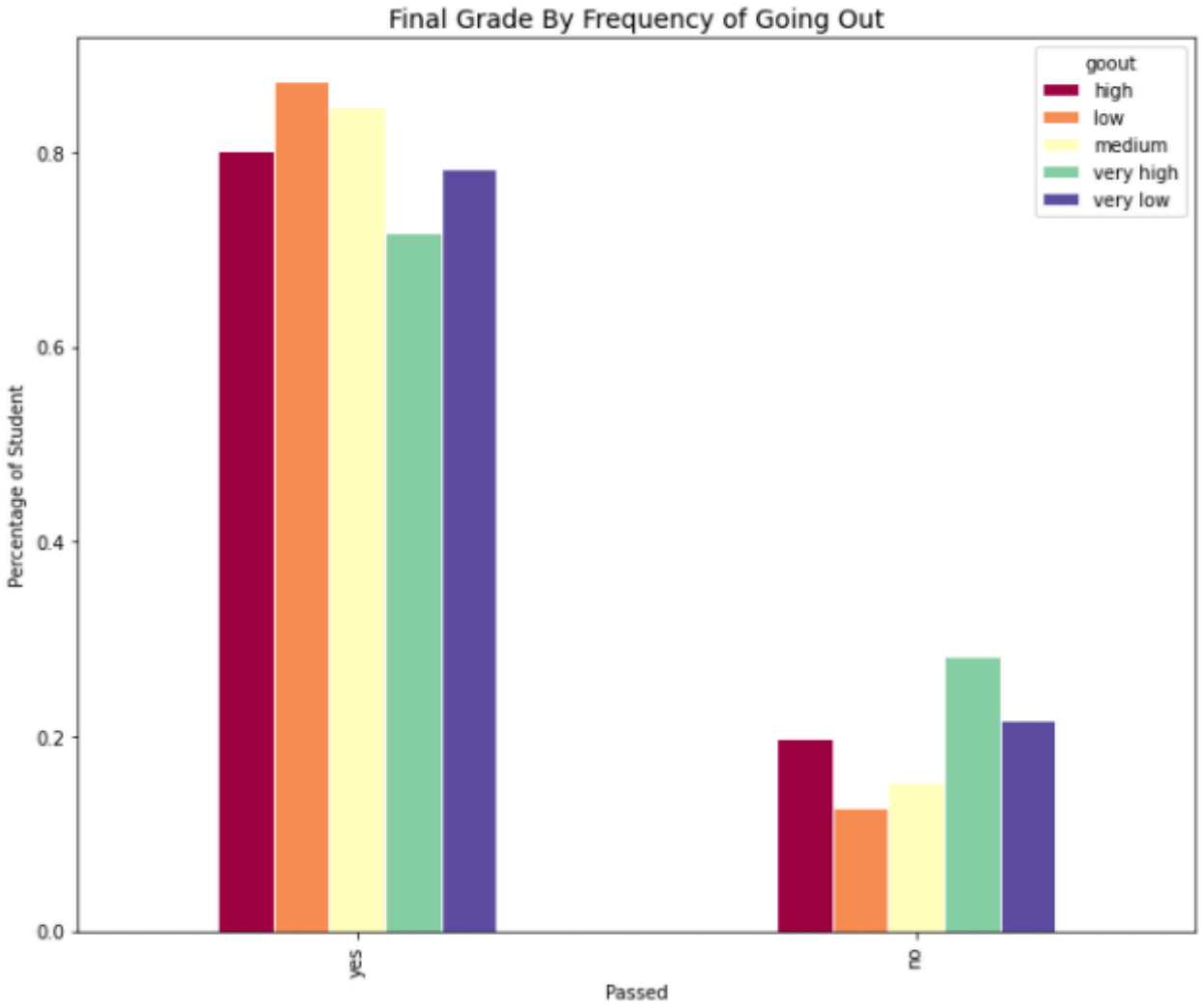

Figure 76: Barplot - Percentage of pass and fail based on going out

Conclusions: For example, students with low gout values are more likely to pass, and those with low values are less likely to fail, as shown by the bar graph above. As a result, students who have high levels of gout have a high percentage of passing grades, just like students who have medium levels of gout. As a result, we can unequivocally state that students who frequently go on outings have a negative impact on their academic performance. As a result, students should be discouraged from going out as much.

5.7 Final Grade Affect by Travel Time

```
# Countplot showing passed and fail number based on student's traveltime
plt.figure(figsize = (11,9))

stu_time_detail = sns.countplot(
    x = "traveltime",
    data = df,
    hue = "passed",
    palette = "twilight_shifted_r",
    edgecolor = "white"
)

stu_time_detail.set(
    xlabel = "Travel Time",
    ylabel = "Number of Studetns"
)

plt.title(
    "Final Grade Affect by Travel Time",
    fontsize = 14
)

plt.show()
```

Figure 77: Code to generate affect of travel time.

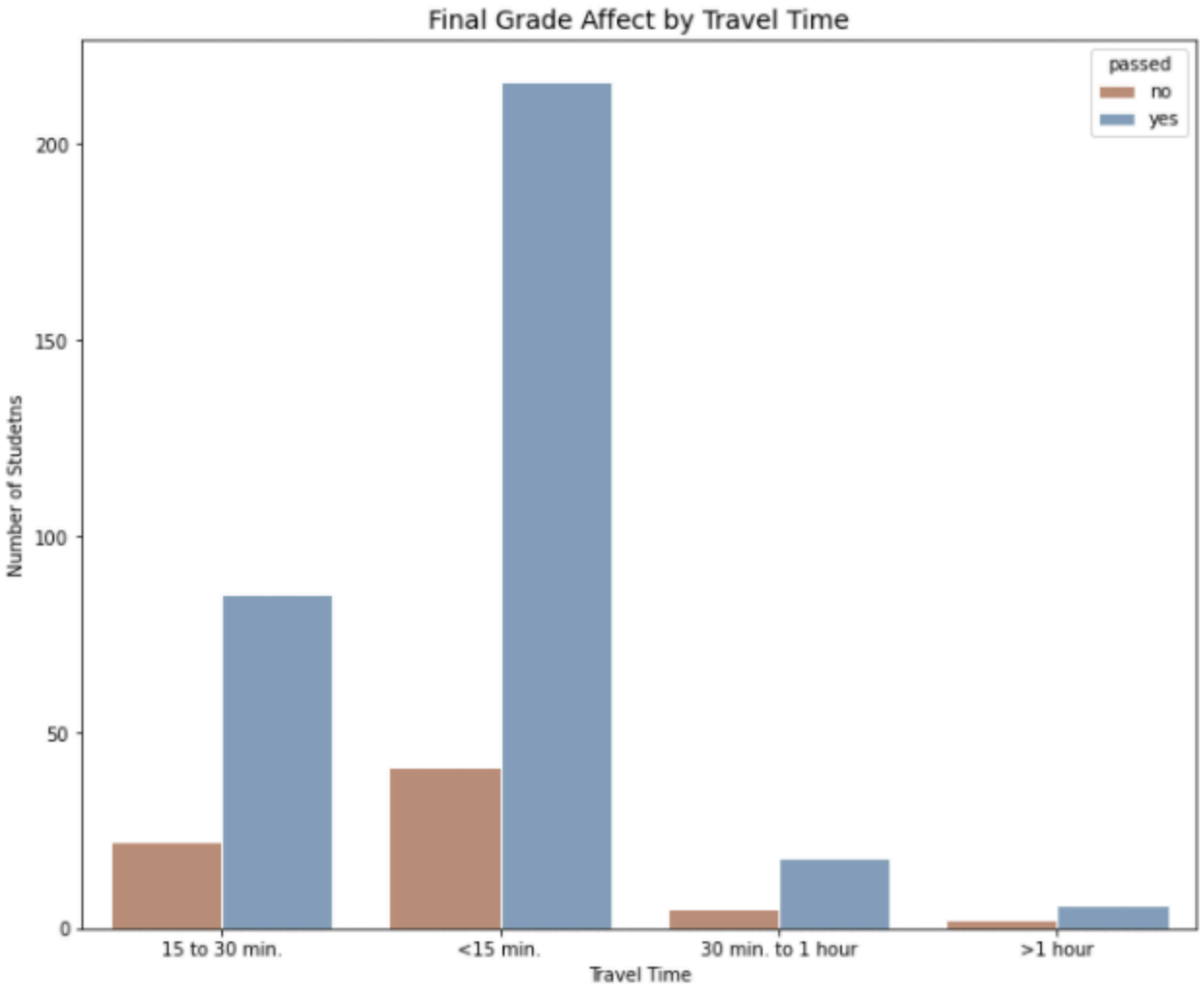

Figure 78: Final grade affect by travel time

Conclusions: As shown in the above graph, travel time has a direct impact on students' grades, as students who take more than an hour to get to class have a lower percentage of passing records than students who take less than 15 minutes to get to class. Also, students who travel for 30 minutes to an hour and a half are less likely to pass. As a result, students' ability to pass exams is directly impacted by the time it takes them to commute between school and home.

5.8 Final Grade by Romantic Status

```
# Romantic status
roa_col = pd.crosstab(index = df.passed, columns = df.romantic)
roa_tab = np.log(roa_col)
roa_pct = roa_tab.apply(pct).reindex(index)

# Plot showing passed and fail number based on students romantic status.
roa_pct.plot.bar(colormap = "brg", figsize = (8,8))
plt.title('Final Grade By Romantic Status', fontsize = 14)
plt.ylabel('Percentage of Logarithm Student Counts')
plt.xlabel('Passed')
plt.show()
```

Figure 79: Code to plot barplot students pass and fail rate

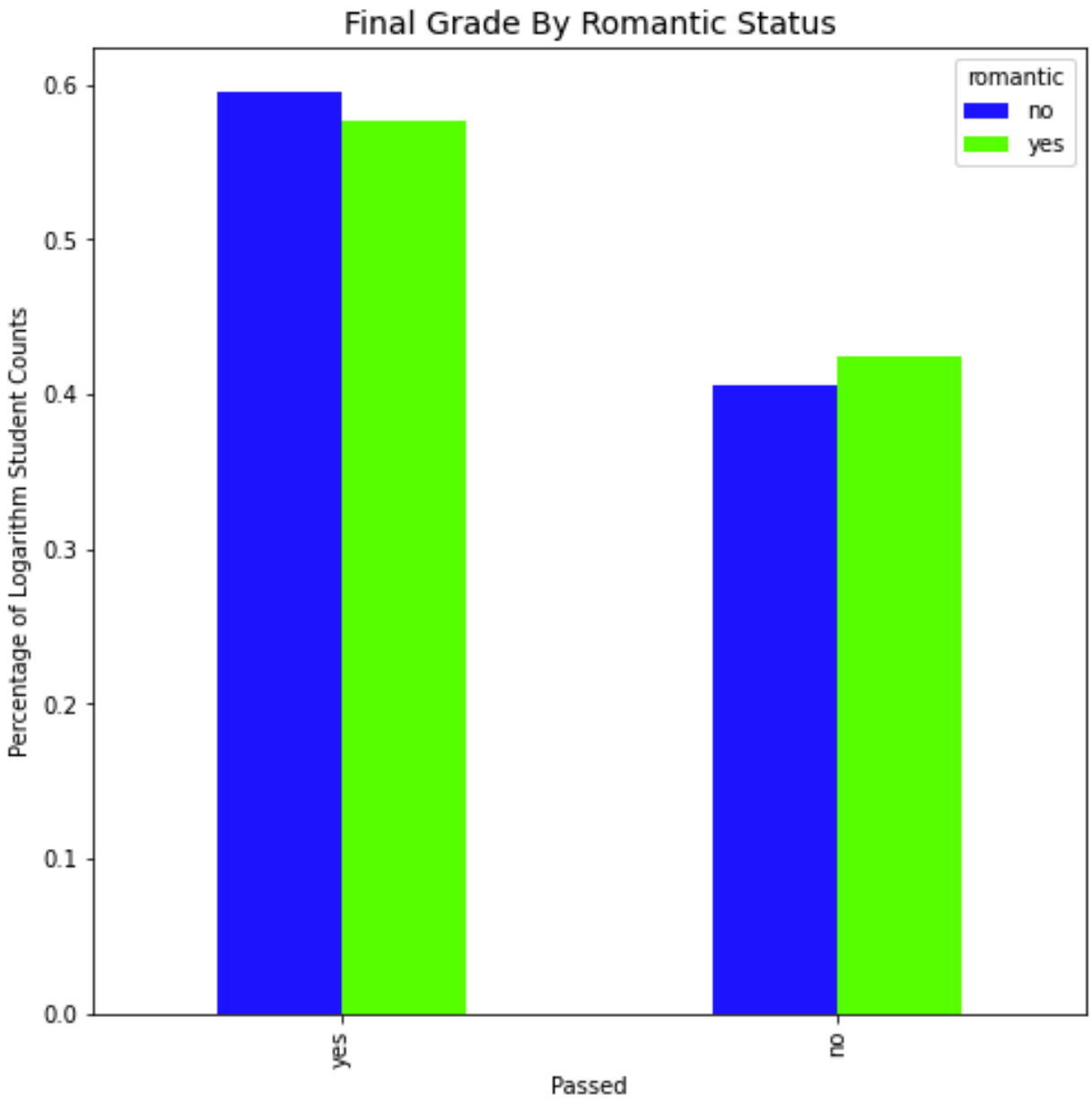

Figure 80: Barplot to show final grade of students based on romantic status

Conclusions: As we can see, students with a romantic relationship have a lower percentage of passing grades than students without a romantic relationship. When it comes to the number of students who fail, we can see that those who have a romantic relationship have a higher percentage than those who do not. This means that students' grades will suffer if they are in a romantic relationship.

Part 2: Analysis of Livestock Data of Nepal

1. Data Understanding

The data we have been provided with pertains to livestock raised throughout Nepal, categorised by districts/regions and the commodities produced by them. The aggregated data is segmented according to the products of meat, milk, and eggs, as well as the number of animals, cotton, and wool. All of this data is contained in multiple.csv files. The following details pertain to data in tabular format:

Attribute Information

SN	Column Name	Description	Variable Type
1.	District	It encompasses all of Nepal's districts.	Categorical
2.	Horses/Asses	This table summarises the total number of horses and asses in various districts.	Discrete
3.	Milking Cows no.	This attribute contains total number milking cows in different districts	Discrete
4.	Milking Buffaloes No.	Contains total number of milking buffaloes in different districts	Discrete
5.	Cow Milk	Contains information about total milk produced by cow in different districts	Discrete
6.	Buff Milk	Contains information about total milk produced by buffalo in different districts	Discrete
7.	Total Milk produced	Contains information about total milk produced by both cow and buffalo in different districts	Discrete
8.	Buff	This attribute contains total weight of buff meats produced in different districts.	Discrete
9.	Mutton	This attribute contains total weight of mutton meats produced in different districts.	Discrete
10.	Chevon	This attribute contains total weight of chevon meats produced in different districts.	Discrete
11.	Pork	This attribute contains total weight of pork meats produced in different districts.	Discrete
12.	Chicken	This attribute contains total weight of chicken meats produced in different districts.	Discrete
13.	Duck Meat	This attribute contains total weight of duck meats produced in different districts.	Discrete
14.	Total Meat	It contains information about total number of all kinds of meats in different districts	Discrete
15.	Area	It gives us the information about total area in hectare allocated for cotton production in different districts	Discrete
16.	Prod.(Mt.)	It gives us the information about total production of cottons in different districts.	Discrete

17.	Yield Kg/Ha	This attribute contains total yield of cotton produced in different districts.	Discrete
18.	Laying Hen	This attribute contains total number of hen laying eggs in different districts.	Discrete
19.	Laying Duck	This attribute contains total number of ducks laying eggs in different districts.	Discrete
20.	Hen Egg	It gives us the information about total production of hen eggs in different districts.	Discrete
21.	Duck Egg	It gives us the information about total production of duck eggs in different districts.	Discrete
22.	Total Egg	It gives us the information about total number of eggs production in different districts.	Discrete
23.	Rabbit	It gives us the information about total number of rabbits in farm in different districts.	Discrete
24.	Sheeps No.	This attribute contains total number of sheeps in farms of different districts.	Discrete
25.	Sheep Wool Produced	This attribute contains about sheep wool produced in different districts.	Discrete
26.	Yak/Nak/Chauri	This attribute contains total number of Yak/Nak/Chauri in different districts.	Discrete

Table 2: Attributes description and variable type Livestock Data of Nepal

2. Data Merging and Cleaning

2.1. Creating Pandas Dataframe for each Dataset

```
# Creating pandas dataframe for all dataset

# Making dataframe
horse_csv_url = 'Dataset/Part 2/horseasses-population-in-nepal-by-district.csv'
milk_csv_url = 'Dataset/Part 2/milk-animals-and-milk-production-in-nepal-by-
district.csv'
meat_csv_url = 'Dataset/Part 2/net-meat-production-in-nepal-by-district.csv'
cotton_csv_url = 'Dataset/Part 2/production-of-cotton-in-nepal-by-district.csv'
egg_csv_url = 'Dataset/Part 2/production-of-egg-in-nepal-by-district.csv'
rabbit_csv_url = 'Dataset/Part 2/rabbit-population-in-nepal-by-district.csv'
wool_csv_url = 'Dataset/Part 2/wool-production-in-nepal-by-district.csv'
yakchauri_csv_url = 'Dataset/Part 2/yak-nak-chauri-population-in-nepal-by-
district.csv'

df_horse = pd.read_csv(horse_csv_url)
df_milk = pd.read_csv(milk_csv_url)
df_meat = pd.read_csv(meat_csv_url)
df_cotton = pd.read_csv(cotton_csv_url)
df_egg = pd.read_csv(egg_csv_url)
df_rabbit = pd.read_csv(rabbit_csv_url)
df_wool = pd.read_csv(wool_csv_url)
df_yakchauri = pd.read_csv(yakchauri_csv_url)
```

Figure 81: Create Pandas dataframe for each Dataset

Verifying if the Given Total in Dataset is Correct or Not

```
'''
Changing column name to calculate original total
'''
df_horse.rename(columns={'Horses/Asses':'Horses_Asses'}, inplace = True)
```

```
df_horse.tail()
```

	DISTRICT	Horses_Asses
55	DOTI	252
56	BAITADI	484
57	DADELDHURA	241
58	FW.REGION	3811
59	Total	55808

Figure 82: Change column name to Horses_Asses

Confirming that the total value given is not correct.

```
total_horse = df_horse.Horses_Asses.sum() # finds out the sum of values of
Horses_asses
total_horse
```

```
167424
```

```
total_horse = df_horse.Horses_Asses.sum() # finds out the sum of values of
Horses_asses
total_horse
```

```
167424
```

```
'''
Removing total because the actual total is not correct compared with the
dataset.
'''
df_horse.drop(labels=59, axis=0, inplace = True)
df_horse.tail()
```

	DISTRICT	Horses_Asses
54	ACHHAM	95
55	DOTI	252
56	BAITADI	484
57	DADELDHURA	241
58	FW.REGION	3811

Figure 83: Removing total row from the dataset

2.2. Data Cleaning and Merging

After a thorough examination of the dataset, it became clear that some cleaning was required prior to merging it. Because the districts had different names, the difference is identified using the find_difference() function, and all of the names are renamed to the same name.

Creating function to find different district in two dataframe to perform Merge

```
# Function to find out the different district name in two dataframes

def find_difference(list1, list2):
    list_difference = [] # stores the differences
    for item in list1:
        if item not in list2:
            list_difference.append(item) #appends the different values found
    return list_difference
```

Figure 84: Create function

```
"""Creating list of districts to use it in above function."""

horse_dis = list(df_horse.DISTRICT) # df_horse converted in list to find out
the list of districts
milk_dis = list(df_milk.DISTRICT) # df_milk converted in list to find out the
list of districts
```

```
"""Using above function to find different value in both the list"""

find_difference(horse_dis, milk_dis)
```

```
['TERATHUM', 'E.REGION', 'C.REGION', 'W.REGION', 'MW.REGION', 'FW.REGION']
```

Figure 85: Find difference in two dataframe

Transforming the value in both df_horse and df_milk to make it equal for merge

```
"""Transforming above different district in to same spelling"""

df_milk['DISTRICT'] = df_milk['DISTRICT'].replace(['TERHATHUM'],'TERATHUM')
df_milk['DISTRICT'] = df_milk['DISTRICT'].replace(['E. REGION'],'E.REGION')
df_milk['DISTRICT'] = df_milk['DISTRICT'].replace(['C. REGION'],'C.REGION')
df_milk['DISTRICT'] = df_milk['DISTRICT'].replace(['W. REGION'],'W.REGION')
df_milk['DISTRICT'] = df_milk['DISTRICT'].replace(['MW. REGION'],'MW.REGION')
df_milk['DISTRICT'] = df_milk['DISTRICT'].replace(['FW. REGION'],'FW.REGION')
```

Figure 86: Making districts name same

Merging df_horse and df_milk and create new dataframe df_merge1

```
df_merge1 = pd.merge(df_horse, df_milk, on="DISTRICT", how='outer')
df_merge1.head()
```

	DISTRICT	Horses_Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED
0	TAPLEJUNG	543.0	8123	4987	5389	4257	9645.0
1	SANKHUWASHAVA	358.0	15342	13367	6988	10589	17577.0
2	SOLUKHUMBU	1775.0	7819	13501	2948	5493	8441.0
3	PANCHTHAR	15.0	14854	11331	8511	9835	18346.0
4	ILLAM	2815.0	26821	5759	19735	15261	34996.0

Figure 87: Merge and create single dataframe

Finding Difference between df_merge1 and df_meat before merging

```
"""Using find_difference function to find difference between two dataframes"""

horseassess = list(df_merge1['DISTRICT'])
milkproduction = list(df_meat['DISTRICT'])
find_difference(horseassess, milkproduction)
```

```
['SANKHUWASHAVA', 'TERATHUM']
```

Figure 88: Difference between df_merge1 and df_meat

Renaming necessary district name

```
"""Renaming different district name to merge"""

df_meat['DISTRICT'] = df_meat['DISTRICT'].replace(['TERHATHUM'],'TERATHUM')
df_meat['DISTRICT'] =
df_meat['DISTRICT'].replace(['SANKHUWASABHA'],'SANKHUWASHAVA')
```

Figure 89: Renaming necessary district name

Merging df_merge1 and df_meat create new dataframe df_merge2

```
df_merge2 = pd.merge(df_merge1, df_meat, on="DISTRICT", how='outer')
df_merge2.head()
```

	DISTRICT	Horses_Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEV
0	TAPLEJUNG	543.0	8123	4987	5389	4257	9645.0	607	31	
1	SANKHUWASHAVA	358.0	15342	13367	6988	10589	17577.0	1646	41	
2	SOLUKHUMBU	1775.0	7819	13501	2948	5493	8441.0	1123	28	
3	PANCHTHAR	15.0	14854	11331	8511	9835	18346.0	1496	4	
4	ILLAM	2815.0	26821	5759	19735	15261	34996.0	1974	1	

Figure 90: Create new dataframe after merge df_merge2

Finding diffrence between df_merge2 and df_cotton

```
"""Using find_difference function to find difference between two dataframes"""

meatproduction = list(df_merge2['DISTRICT'])
cottonproduction = list(df_cotton['DISTRICT'])
find_difference(cottonproduction, meatproduction)
```

```
['Dang', 'Banke', 'Bardiya']
```

```
"""Renaming different district name to merge"""

df_cotton['DISTRICT'] = df_cotton['DISTRICT'].replace(['Dang'],'DANG')
df_cotton['DISTRICT'] = df_cotton['DISTRICT'].replace(['Banke'],'BANKE')
df_cotton['DISTRICT'] = df_cotton['DISTRICT'].replace(['Bardiya'],'BARDIYA')
```

Figure 91: Diffrence between df_merge2 and df_cotton¶

Merging df_merge2 and df_cotton to create df_merge3

```
df_merge3 = pd.merge(df_merge2, df_cotton, on="DISTRICT", how='outer') # merges
horseassess and milkproduction and creates new dataframe df9
df_merge3.head()
```

	DISTRICT	Horses_Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEV
0	TAPLEJUNG	543.0	8123	4987	5389	4257	9645.0	607	31	
1	SANKHUWASHAVA	358.0	15342	13367	6988	10589	17577.0	1646	41	
2	SOLUKHUMBU	1775.0	7819	13501	2948	5493	8441.0	1123	28	
3	PANCHTHAR	15.0	14854	11331	8511	9835	18346.0	1496	4	
4	ILLAM	2815.0	26821	5759	19735	15261	34996.0	1974	1	

Figure 92: Merge two dataframe can create new dataframe df_merge3

Finding difference between df_merge3 and df_egg

```
"""Using find_difference function to find difference between two dataframes"""

eggproduction = list(df_merge3['DISTRICT'])
rabbitpopulation = list(df_egg['DISTRICT'])
find_difference(eggproduction, rabbitpopulation)
```

```
['TERATHUM']
```

```
"""Renaming different district name to merge"""

df_egg['DISTRICT'] = df_egg['DISTRICT'].replace(['TERHATHUM'],'TERATHUM')
```

Figure 93: Find difference between df_merge3 and df_egg

Merging df_merge3 and df_egg to create df_merge4

```
df_merge4 = pd.merge(df_merge3, df_egg, on="DISTRICT", how='outer')
df_merge4.head()
```

	DISTRICT	Horses_Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEVON	...	DUCK MEAT
0	TAPLEJUNG	543.0	8123	4987	5389	4257	9645.0	607	31	491	...	0
1	SANKHUWASHAVA	358.0	15342	13367	6988	10589	17577.0	1646	41	958	...	1
2	SOLUKHUMBU	1775.0	7819	13501	2948	5493	8441.0	1123	28	416	...	0
3	PANCHTHAR	15.0	14854	11331	8511	9835	18346.0	1496	4	940	...	1
4	ILLAM	2815.0	26821	5759	19735	15261	34996.0	1974	1	870	...	0

5 rows × 22 columns

Figure 94: Merging df_merge3 and df_egg and creating df_merge4

Finding difference in df_merge4 and df_rabbit

```
"""Using find_difference function to find difference between two dataframes"""

wool_dis = list(df_merge4['DISTRICT'])
rabbit_dis = list(df_rabbit['DISTRICT'])
find_difference(rabbit_dis, wool_dis)
```

```
['TERHATHUM', 'RAMECHHAP', 'Total']
```

```
"""Removing total because the actual total is not correct compared with the
dataset."""

df_rabbit.drop(labels=54, axis=0, inplace = True)
df_rabbit.head()
```

	DISTRICT	Rabbit
0	TAPLEJUNG	506
1	SANKHUWASHAVA	313
2	SOLUKHUMBU	105
3	PANCHTHAR	29
4	ILLAM	240

```
"""Renaming different district name to merge"""

df_rabbit['DISTRICT'] = df_rabbit['DISTRICT'].replace(['TERHATHUM'],'TERATHUM')
df_rabbit['DISTRICT'] = df_rabbit['DISTRICT'].replace(['RAMECHHAP'],'RAMECHAP')
```

Figure 95: Find difference in df_merge4 and df_rabbit

Merging df_merge4 and df_rabbit to create df_merge5

```
df_merge5 = pd.merge(df_merge4, df_rabbit, on="DISTRICT", how='outer')
df_merge5.head()
```

	DISTRICT	Horses_Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEVON	...	TOTA MEA
0	TAPLEJUNG	543.0	8123	4987	5389	4257	9645.0	607	31	491	...	174
1	SANKHUWASHAVA	358.0	15342	13367	6988	10589	17577.0	1646	41	958	...	345
2	SOLUKHUMBU	1775.0	7819	13501	2948	5493	8441.0	1123	28	416	...	216
3	PANCHTHAR	15.0	14854	11331	8511	9835	18346.0	1496	4	940	...	341
4	ILLAM	2815.0	26821	5759	19735	15261	34996.0	1974	1	870	...	336

5 rows × 23 columns

Figure 96: Merging df_merge4 and df_rabbit to create df_merge5

Finding difference in df_merge5 and df_wool

```
"""Using find_difference function to find difference between two dataframes"""

df_merge5_diff = list(df_merge5['DISTRICT'])
horseassess_diff = list(df_wool['DISTRICT'])
find_difference(horseassess_diff, df_merge5_diff)
```

```
['TERHATHUM']
```

```
"""Renaming different district name to merge"""

df_wool['DISTRICT'] = df_wool['DISTRICT'].replace(['TERHATHUM'],'TERATHUM')
```

Figure 97: Finding difference in df_merge5 and df_wool

Merging df_merge5 and df_wool to create new df_merge6

```
df_merge6 = pd.merge(df_merge5, df_wool, on="DISTRICT", how='outer')
df_merge6.head()
```

	DISTRICT	Horses_Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEVON
0	TAPLEJUNG	543.0	8123	4987	5389	4257	9645.0	607	31	491
1	SANKHUWASHAVA	358.0	15342	13367	6988	10589	17577.0	1646	41	958
2	SOLUKHUMBU	1775.0	7819	13501	2948	5493	8441.0	1123	28	416
3	PANCHTHAR	15.0	14854	11331	8511	9835	18346.0	1496	4	940
4	ILLAM	2815.0	26821	5759	19735	15261	34996.0	1974	1	870

5 rows × 25 columns

Figure 98: Merge df_merge5 and df_wool to create new df_merge6

Finding difference between df_merge6 and df_yakchauri

```
"""Using find_difference function to find difference between two dataframes"""

df_merge6_diff = list(df_merge6['DISTRICT'])
yakchauri_diff = list(df_yakchauri['DISTRICT'])
find_difference(yakchauri_diff, df_merge6_diff)
```

```
['Total']
```

```
"""Removing total because the actual total is not correct compared with the
dataset."""

df_yakchauri.drop(labels=34, axis=0, inplace = True)
df_yakchauri.head()
```

	DISTRICT	YAK/NAK/CHAURI
0	TAPLEJUNG	3465
1	SANKHUWASHAVA	3945
2	SOLUKHUMBU	12235
3	PANCHTHAR	1075
4	ILLAM	165

Figure 99: Finding difference between df_merge6 and df_yakchauri

Merging df_merge6 and df_yakchauri which will create final Dataframe df_finalmerge

```
df_finalmerge = pd.merge(df_merge6, df_yakchauri, on="DISTRICT", how='outer')
df_finalmerge
```

	DISTRICT	Horses_Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHE
0	TAPLEJUNG	543.0	8123	4987	5389	4257	9645.0	607	31	
1	SANKHUWASHAVA	358.0	15342	13367	6988	10589	17577.0	1646	41	
2	SOLUKHUMBU	1775.0	7819	13501	2948	5493	8441.0	1123	28	
3	PANCHTHAR	15.0	14854	11331	8511	9835	18346.0	1496	4	
4	ILLAM	2815.0	26821	5759	19735	15261	34996.0	1974	1	
...	...	...	...	...	...	...	...	...	...	
91	FW.HILLS	NaN	45036	39569	22850	33505	56355.0	5692	14	
92	KAILALI	NaN	27758	41103	27905	36677	64582.0	5962	71	
93	KANCHANPUR	NaN	20164	27812	23146	25876	49022.0	3816	27	
94	FW.TERAI	NaN	47922	68915	51051	62553	113604.0	9778	98	
95	NEPAL	NaN	1026135	1355384	643806	1210441	NaN	175005	2684	

96 rows × 26 columns

Figure 100: Create final Dataframe df_finalmerge

All of the dataframes are combined together to make a new final dataframe after they have been cleaned up. Now, all of the necessary data analysis is done with this final merged dataframe.

3. Exploratory Data Analysis

The following is a presentation of insights obtained from the dataset using various analysis and visualisation techniques.

3.1 Milk Production Analysis

```
# Extracting and creating dataframe of five different regions
data_milk=
df_finalmerge[df_finalmerge['DISTRICT'].str.startswith(("FW.","MW.","W.","E.","
C."))]
data_milk
```

Figure 101: Create dataframe of total milk

	DISTRICT	Horses_Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEVON	...	YIELD Kg/Ha	LAYING HEN	LAYING DUCK	HEN EGG	DUCK EGG	TOTAL EGG	Rabbit	SHEEPS NO.	SHEEP WOOL PRODUCED	YAK/NAK/CHAURI
13	E.REGION	7616.0	332384	292178	196708	263199	459907.0	41220	269	15729	...	NaN	1780554.06	60193	214241	4712	218953	5124.0	79815	58002	22797.0
18	C.REGION	1468.0	263728	377741	177815	358483	536299.0	50244	256	16893	...	NaN	7118554.32	49572	756783	3907	760690	5304.0	77126	57295	12484.0
34	W.REGION	7789.0	154560	341323	105190	315616	420806.0	40476	561	12540	...	NaN	1745955.36	54256	163058	3963	167021	8915.0	144089	101731	14823.0
50	MW.REGION	35124.0	144868	211885	76157	160705	236862.0	24911	1263	13528	...	NaN	1170714.06	10534	119341	820	120161	11483.0	397057	295006	17835.0
58	FW.REGION	3811.0	130595	132257	87936	112438	200374.0	18154	335	6893	...	NaN	537737.00	6372	40743	504	41247	1387.0	102571	76314	892.0
59	E.MOUNTAIN	NaN	31284	31855	15324	20339	35663.0	3376	100	1865	...	NaN	135548.00	1180	10271	87	10358	NaN	26419	18855	NaN
61	E.HILLS	NaN	123976	109431	74587	95837	170424.0	15440	146	5484	...	NaN	549366.00	5916	78878	458	79336	NaN	46238	33361	NaN
64	E.TERAI	NaN	177124	150892	106797	147023	253820.0	22404	23	8380	...	NaN	1095640.00	53097	125092	4167	129259	NaN	7158	5786	NaN
68	C.MOUNTAIN	NaN	21380	32607	13173	30261	43434.0	4486	85	1821	...	NaN	232271.00	3423	16482	265	16747	NaN	25100	18649	NaN
77	C.HILLS	NaN	125519	187803	78958	187149	266107.0	23305	147	6777	...	NaN	3222902.00	17417	276415	1338	277753	NaN	44251	32874	NaN
82	C.TERAI	NaN	116829	157331	85684	141074	226758.0	22453	24	8295	...	NaN	3663381.00	28732	463886	2304	466190	NaN	7775	5772	NaN
83	W.MOUNTAIN	NaN	1561	49	894	35	929.0	6	34	195	...	NaN	3571.00	13	616	1	617	NaN	12257	9231	NaN
85	W.HILLS	NaN	94009	225270	64947	230740	295687.0	27487	422	7995	...	NaN	1129538.00	21433	106718	1458	108176	NaN	97499	72231	NaN
86	W.TERAI	NaN	58990	116004	39349	84840	124189.0	12983	105	4350	...	NaN	612847.00	32810	55724	2504	58228	NaN	34333	20269	NaN
87	MW.MOUNTAIN	NaN	18414	7963	7878	5880	13758.0	1221	702	1016	...	NaN	21256.00	495	2368	34	2402	NaN	210274	156231	NaN
88	MW.HILLS	NaN	77832	107908	38547	77936	116483.0	13522	369	5750	...	NaN	420065.00	4744	33666	369	34035	NaN	128058	95144	NaN
89	MW.TERAI	NaN	48622	96014	29732	76889	106621.0	10168	192	6762	...	NaN	729393.00	5295	83307	417	83724	NaN	58725	43631	NaN
90	FW.MOUNTAIN	NaN	37637	23773	14035	16380	30415.0	2684	223	1460	...	NaN	25484.00	487	2615	35	2650	NaN	68985	51254	NaN
91	FW.HILLS	NaN	45036	39569	22850	33505	56355.0	5692	14	3103	...	NaN	48735.00	535	7717	39	7756	NaN	4366	3350	NaN
94	FW.TERAI	NaN	47922	68915	51051	62553	113604.0	9778	98	2330	...	NaN	463517.80	5350	30411	430	30841	NaN	29220	21710	NaN

20 rows × 26 columns

Figure 102: Separate dataframe containg information on regions only

The above dataset is divided into subplots for further analysis of milk production in various regions.

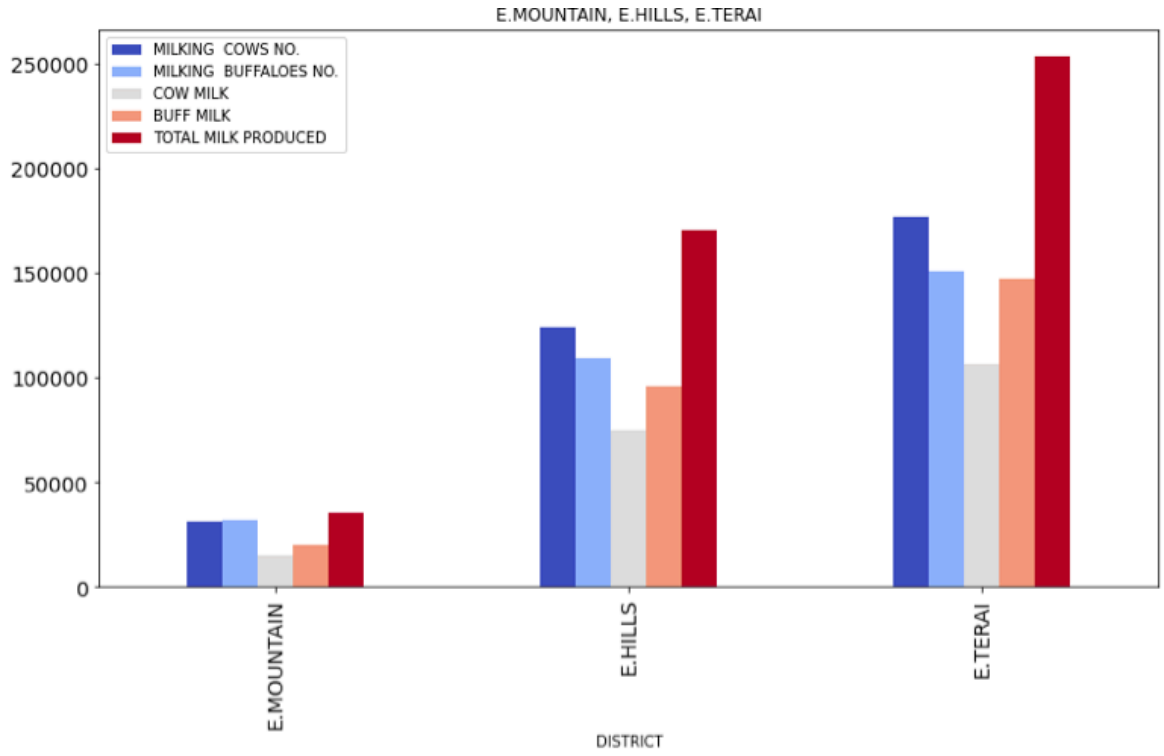

Figure 103: Plot - E.MOUNTAIN, E.HILLS, E.TERAI

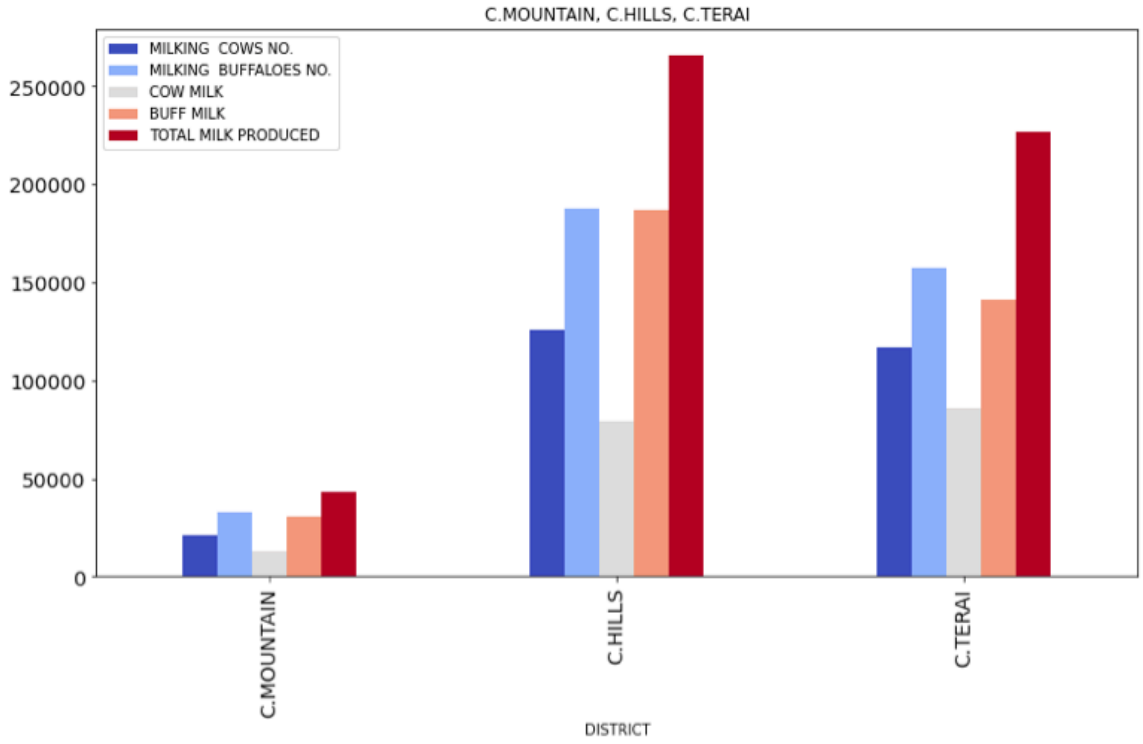

Figure 104: Plot - C.MOUNTAIN, C.HILLS, C.TERAI

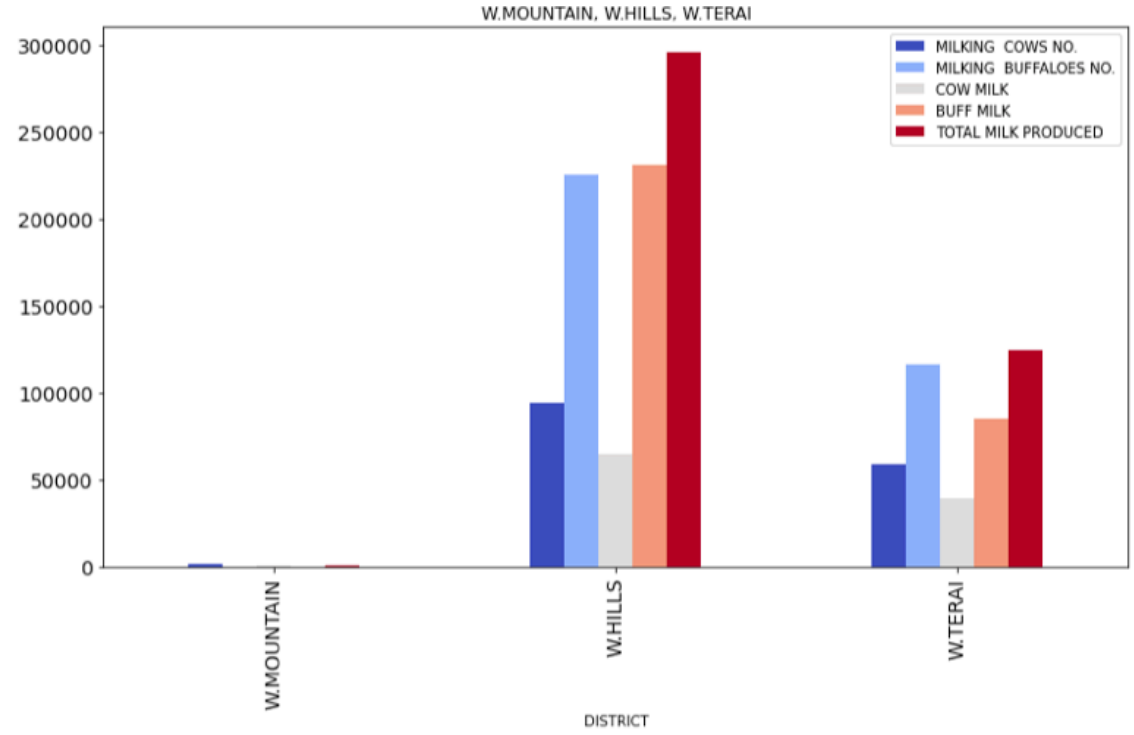

Figure 105: Plot - W.MOUNTAIN, W.HILLS, W.TERAI

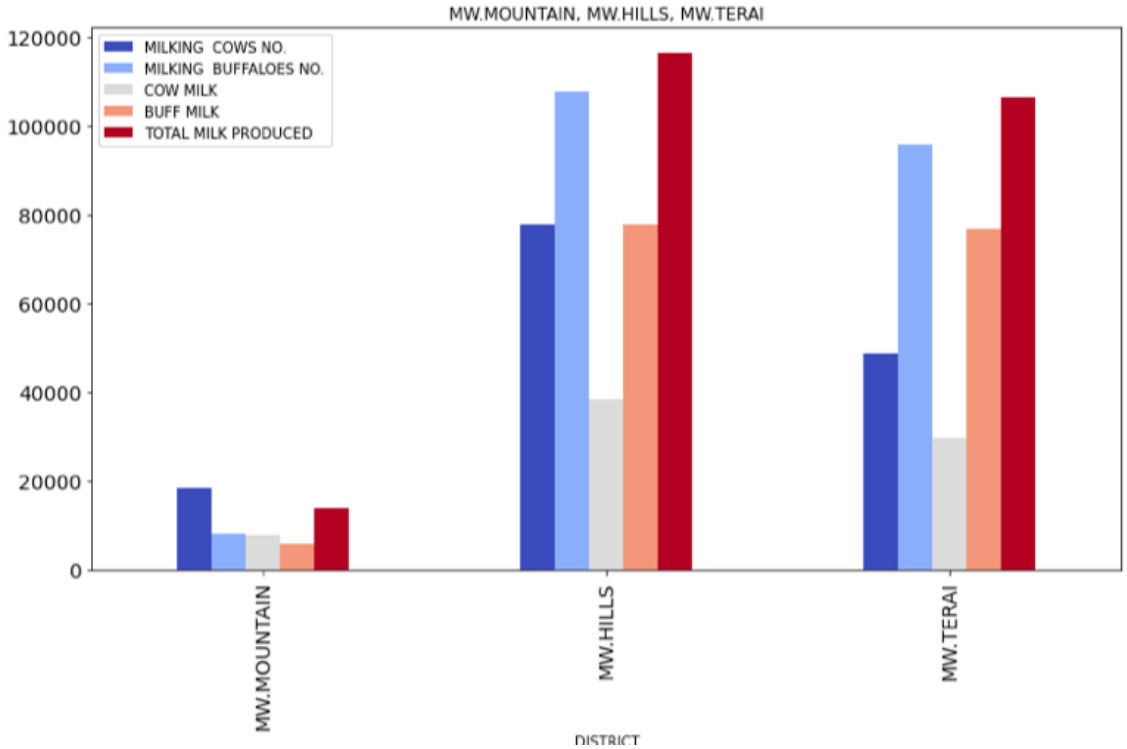

Figure 106: Plot - MW.MOUNTAIN, MW.HILLS, MW.TERAI

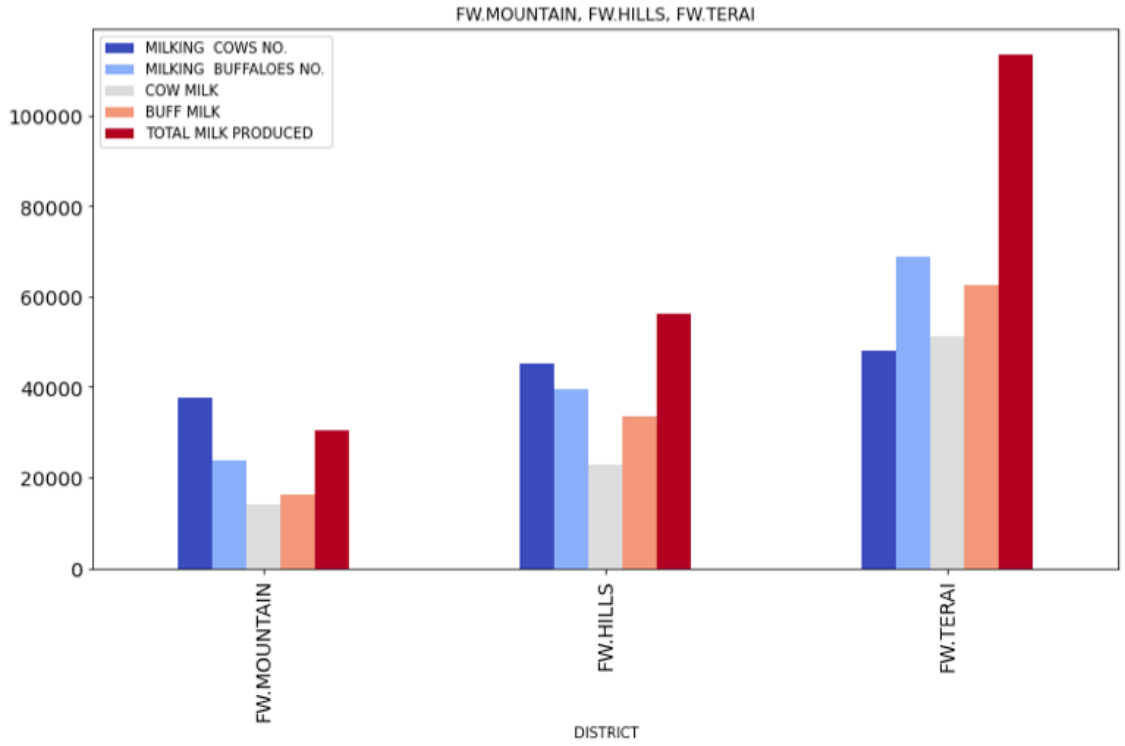

Figure 107: Plot - FW.MOUNTAIN, FW.HILLS, FW.TERAI

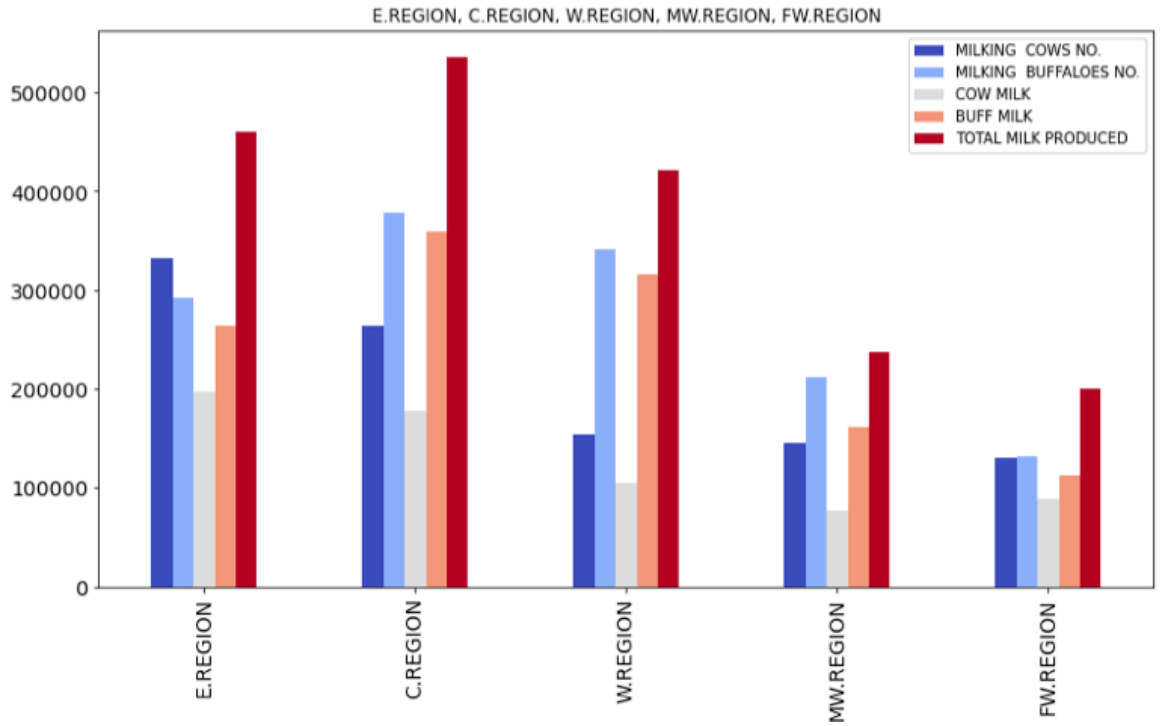

Figure 108: Plot - E.REGION, C.REGION, W.REGION, MW.REGION, FW.REGION

All subplots are divided into five regions of Nepal: eastern, central, western, mid-western, and far-western. And the data is divided into three regions of Nepal: the Mountain region, the Hilly region, and the Terai region. By examining the data, we can see that western hills region of Nepal produces the most milk, while the mid-western mountain region of Nepal produces the least. However, the capacity for producing milk is high in Nepal's mid-western hills region because the number of milking cows and buffaloes in this region is higher than in others.

Looking at the five different regions, we can see that the C.region produces the most milk, while the far-western region produces the least.

3.2 Duck Egg Analysis

```
# Extracting and creating dataframe of five different regions
duck_numbers =
df_finalmerge[df_finalmerge['DISTRICT'].str.startswith(("FW.","MW.","W.","E.","
C."))]
duck_numbers
```

	DISTRICT	Horses_Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEVON
13	E.REGION	7616.0	332384	292178	196708	263199	459907.0	41220	269	15729
18	C.REGION	1468.0	263728	377741	177815	358483	536299.0	50244	256	16893
34	W.REGION	7789.0	154560	341323	105190	315616	420806.0	40476	561	12540
50	MW.REGION	35124.0	144868	211885	76157	160705	236862.0	24911	1263	13528
58	FW.REGION	3811.0	130595	132257	87936	112438	200374.0	18154	335	6893
59	E.MOUNTAIN	NaN	31284	31855	15324	20339	35663.0	3376	100	1865
61	E.HILLS	NaN	123976	109431	74587	95837	170424.0	15440	146	5484
64	E.TERAI	NaN	177124	150892	106797	147023	253820.0	22404	23	8380
68	C.MOUNTAIN	NaN	21380	32607	13173	30261	43434.0	4486	85	1821
77	C.HILLS	NaN	125519	187803	78958	187149	266107.0	23305	147	6777
82	C.TERAI	NaN	116829	157331	85684	141074	226758.0	22453	24	8295

Figure 109: Create datafrme of totla numbers of duck

```
#Total duck egg production in different hill, mountains, terai regions in Nepal
duck_egg = duck_numbers[['DISTRICT','LAYING DUCK','DUCK EGG']]

#Reshaping
duck_reshape= pd.melt(duck_egg, id_vars="DISTRICT",value_name="Laying
Egg",var_name="Duck and Egg")
plt.figure(figsize=(30,10))

#barplot to visualize the data
sns.barplot(x="DISTRICT", y="Laying Egg", data= duck_reshape, hue="Duck and
Egg", palette="autumn")
plt.title('Total duck egg production in different hill, mountains, terai
regions in Nepal', fontsize=16, loc='left')
plt.xlabel('Development Region')
plt.show() # Display a figure
```

Figure 110: Create Barplot to show actual ducks and duck eggs production

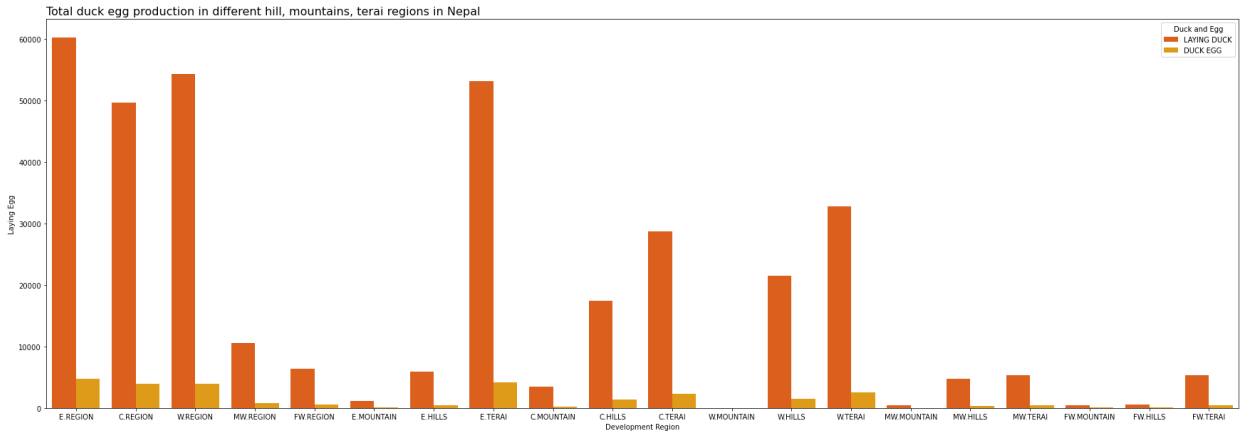

Figure 111: Barplot showing actual ducks and duck eggs produced in different regions

According to the above barplot, the most ducks are found in farms in Nepal's E.region, but egg production is deficient in comparison to the total number of ducks. The western region of Nepal has the second highest number of ducks, but egg production is also lower in comparison to the number of ducks present in farms. Skimming through other regions, the situation is similar; there are more ducks, but egg production is deficient. So, in conclusion, despite having the most significant number of egg-producing ducks, actual egg production is deficient in every region of Nepal.

3.3 Meat Production Analysis

```
meat_productions=
df_finalmerge[df_finalmerge['DISTRICT'].str.endswith(("REGION"))]
meat_productions
```

	DISTRICT	Horses_Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEVON	..
13	E.REGION	7616.0	332384	292178	196708	263199	459907.0	41220	269	15729	.
18	C.REGION	1468.0	263728	377741	177815	358483	536299.0	50244	256	16893	.
34	W.REGION	7789.0	154560	341323	105190	315616	420806.0	40476	561	12540	.
50	MW.REGION	35124.0	144868	211885	76157	160705	236862.0	24911	1263	13528	.
58	FW.REGION	3811.0	130595	132257	87936	112438	200374.0	18154	335	6893	.

5 rows × 26 columns

Figure 112: All meat production dataframe

```
#Total meat production in different Development Region of Nepal
meat_prod =
meat_productions[['DISTRICT','BUFF','MUTTON','CHEVON','CHICKEN','DUCK
MEAT','TOTAL MEAT']]

#Reshaping
meat_reshape= pd.melt(meat_prod, id_vars="DISTRICT",value_name="Meat
Production",var_name="Type of Meat")
plt.figure(figsize=(20,5))

#barplot to visualize the data
sns.lineplot(x="DISTRICT", y="Meat Production", data= meat_reshape, hue="Type
of Meat", palette="YlOrRd")
plt.title('Bar Diagram to show meat production in different region of Nepal',
fontsize=16, loc='left')
plt.xlabel('Development Region')
plt.show() # Display a figure
```

Figure 113: Create plot of total meat production

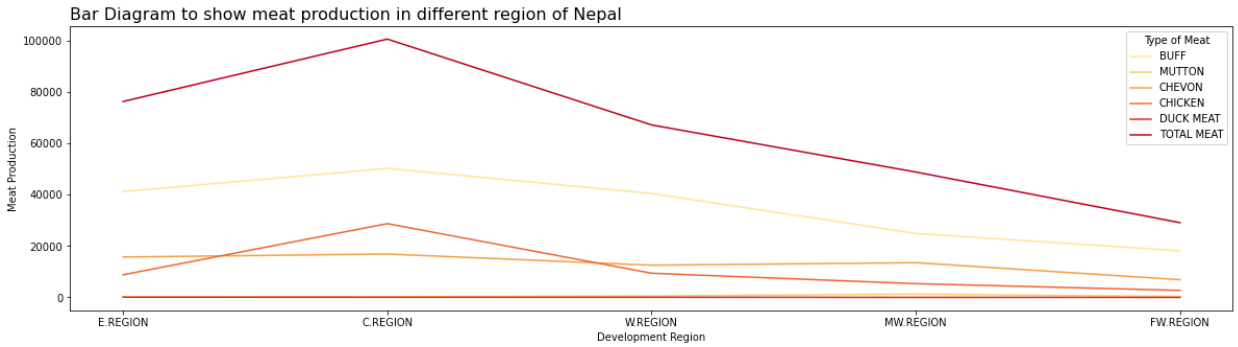

Figure 114: Lineplot showing total meat produced in different regions

Take a close look at the above line plot, which is divided into five different regions of Nepal. As we can see, the total number meat of production, which includes all mutton, buff, chevon, and chicken, is highest in the central region of Nepal and lowest in the far western region. However, if we look closely at meat production, we can see that mutton and chicken meat production is highest in the central region, while chevon meat production is similar in all regions of Nepal.

3.4 Wool Production Analysis

```
# Extracting data based on five different regions
wool_productions=
df_finalmerge[df_finalmerge['DISTRICT'].str.endswith(("REGION"))]
wool_productions
```

	DISTRICT	Horses_Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEVON	..
13	E.REGION	7616.0	332384	292178	196708	263199	459907.0	41220	269	15729	..
18	C.REGION	1468.0	263728	377741	177815	358483	536299.0	50244	256	16893	..
34	W.REGION	7789.0	154560	341323	105190	315616	420806.0	40476	561	12540	..
50	MW.REGION	35124.0	144868	211885	76157	160705	236862.0	24911	1263	13528	..
58	FW.REGION	3811.0	130595	132257	87936	112438	200374.0	18154	335	6893	..

5 rows × 26 columns

Figure 115: Create dataframe of total wool

```
# Pie chart representing wool production in 5 different regions of Nepal
pie, ax = plt.subplots(figsize=[6,6])
plt.pie(wool_productions['SHEEP WOOL PRODUCED'],
        labels = wool_productions['DISTRICT'],
        colors = ["#9E6240", "#DEA47E","#CD4631","#F8F2DC","#81ADC8"], autopct
= "%.2f%%",radius = 1,shadow=True)
plt.title("SHEEP WOOL PRODUCTION ACCORDING TO REGIONS", fontsize=16)
ax.legend(title="WOOL PRODUCTION",bbox_to_anchor =(1.25,1))
plt.show() # Display a figure
```

Figure 116: Create pie plot of total wool

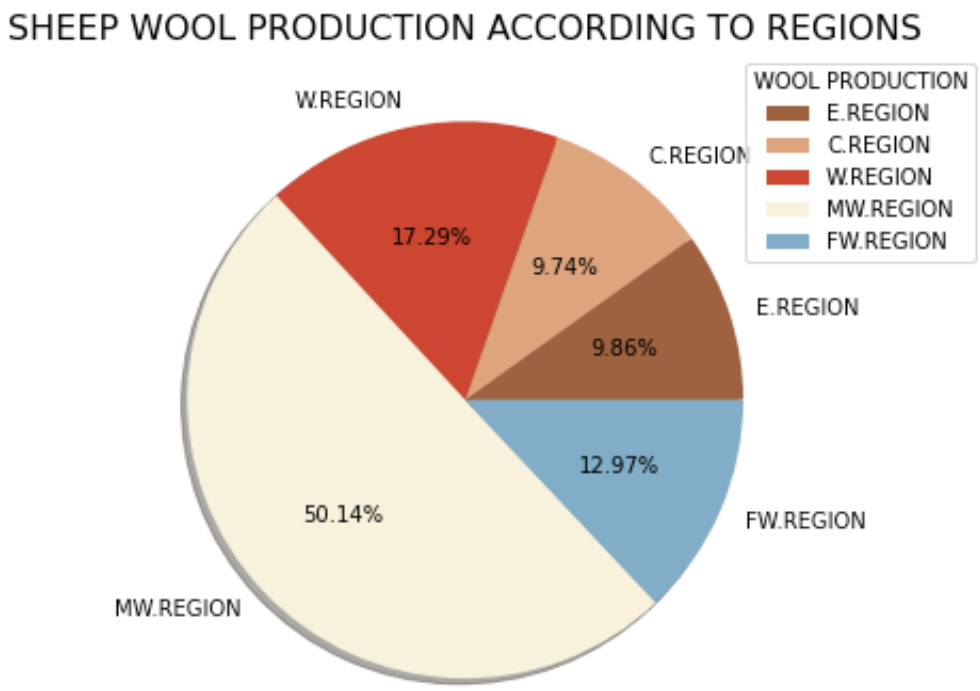

Figure 117: Piechart showing wool production in different regions

According to the data in the pie chart above, 50 percent of Nepal's wool production is done in the mid-western region, while the lowest number, 9.74 percent, is done in the central region. The eastern region of Nepal has a comparable number of wool productions as the central region of Nepal. The difference in production between Nepal's western and far-western regions is only about 5 percent.

3.5 Cotton Production Analysis

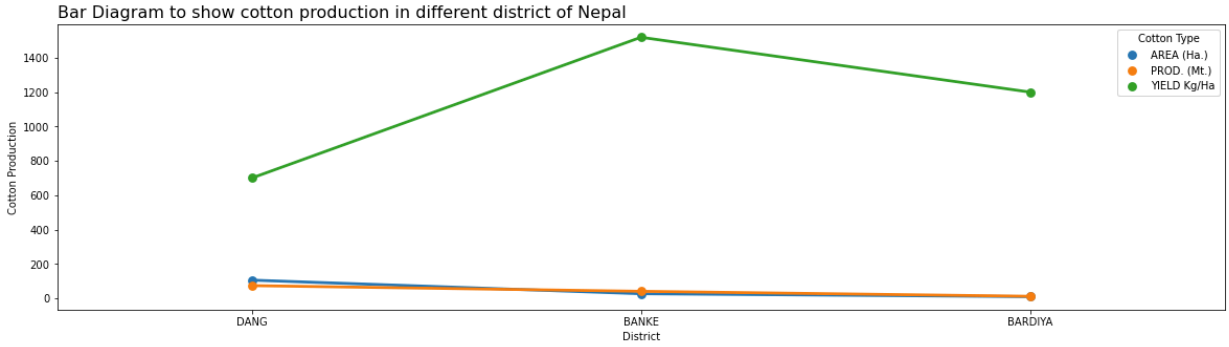

Figure 118: Figure 160: Pointplot showing cotton production in three districts

Looking at the pointplot presentation, we can see that the final yield of cotton is higher in the Banke district of Nepal, whereas production per land area is comparable in all three districts of Nepal. However, Dang has the lowest number of cotton production, and Bardiya is located between Banke and Dang in terms of actual cotton production. So, despite having similar production numbers in all three districts, the actual yield in the Banke district is high.

3.6 HorseAsses Analysis based on Regions

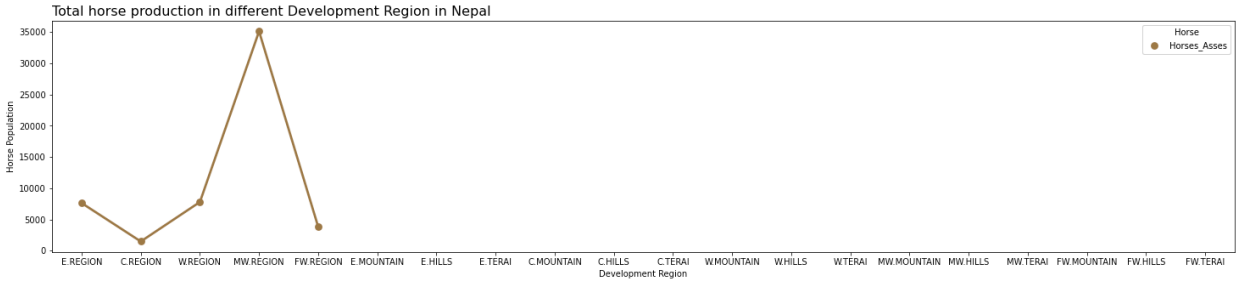

Figure 119: Pointplot showing horseasses population in different regions

Looking at the pointplot presentation, we can see that the mid-western region of Nepal has the highest number of horseasses. In contrast, the central region of Nepal has the lowest number of horseasses compared to other regions. However, the number of horseasses in Nepal's eastern and western regions is comparable. The far-western region ranks second to last in terms of the number of horseasses.

3.7 Heatmap Of Total productions in different Districts

```
# correlation between the different production items
correlation= df_finalmerge[['TOTAL MILK PRODUCED','TOTAL EGG','SHEEP WOOL
PRODUCED','TOTAL MEAT']].corr()

plt.figure(figsize=(11,9))

# plot the heatmap
sns.heatmap(correlation,annot =True,cmap="YlGnBu")
plt.title('Heatmap- Total Production of all Items ',fontsize=16, loc='left')
plt.xlabel('Production Items',fontsize=10)
plt.ylabel('Production Items',fontsize=10)

plt.show() # Display a figure
```

Figure 120: Create correlation between milk, egg, wool, meat

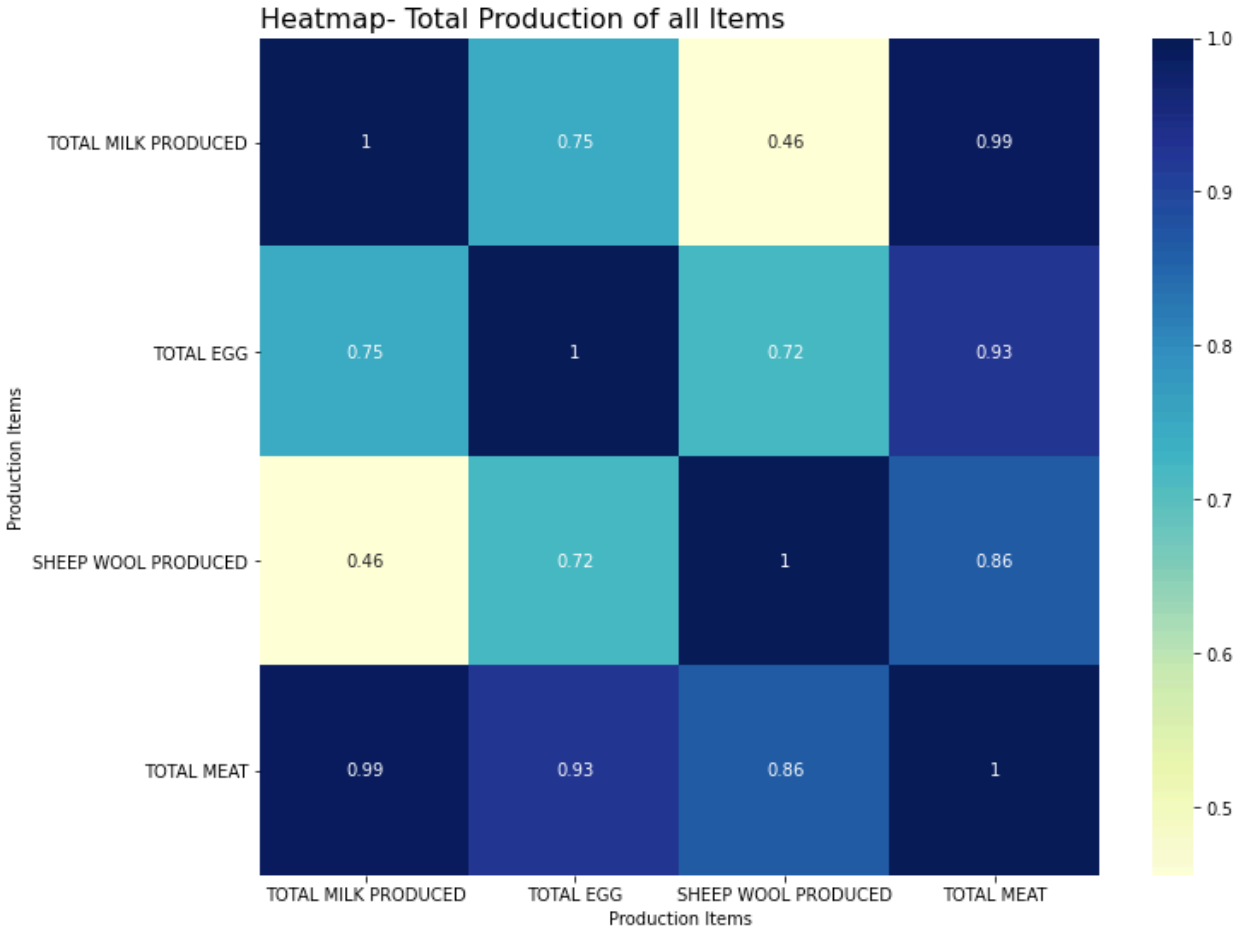

Figure 121: Heatmap showing correlation between total production.

The heatmap depicts the relationship between various types of data. If the colour is dark, the correlation between the two attributes is vital; if the colour is light, the correlation is very weak.

So, based on the heat map above, Total Meat Production has a strong correlation with Total Milk Production. In contrast, Total Milk Production has a very weak correlation with Total Wool Production.

3.8 YAK/NAK/CHAURI Population Analysis

```
# Extracting yak/nak/chauri data of different regions
yak_numbers= df_finalmerge[df_finalmerge['DISTRICT'].str.endswith(("REGION",
"HILLS","MOUNTAIN","TERAI"))]
yak_numbers
```

	DISTRICT	Horses_Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEVON	...	YI Kg
13	E.REGION	7616.0	332384	292178	196708	263199	459907.0	41220	269	15729	...	
18	C.REGION	1468.0	263728	377741	177815	358483	536299.0	50244	256	16893	...	
34	W.REGION	7789.0	154560	341323	105190	315616	420806.0	40476	561	12540	...	
50	MW.REGION	35124.0	144868	211885	76157	160705	236862.0	24911	1263	13528	...	
58	FW.REGION	3811.0	130595	132257	87936	112438	200374.0	18154	335	6893	...	
59	E.MOUNTAIN	NaN	31284	31855	15324	20339	35663.0	3376	100	1865	...	
61	E.HILLS	NaN	123976	109431	74587	95837	170424.0	15440	146	5484	...	
64	E.TERAI	NaN	177124	150892	106797	147023	253820.0	22404	23	8380	...	
68	C.MOUNTAIN	NaN	21380	32607	13173	30261	43434.0	4486	85	1821	...	
77	C.HILLS	NaN	125519	187803	78958	187149	266107.0	23305	147	6777	...	
82	C.TERAI	NaN	116829	157331	85684	141074	226758.0	22453	24	8295	...	
83	W.MOUNTAIN	NaN	1561	49	894	35	929.0	6	34	195	...	
85	W.HILLS	NaN	94009	225270	64947	230740	295687.0	27487	422	7995	...	
86	W.TERAI	NaN	58990	116004	39349	84840	124189.0	12983	105	4350	...	
87	MW.MOUNTAIN	NaN	18414	7963	7878	5880	13758.0	1221	702	1016	...	
88	MW.HILLS	NaN	77832	107908	38547	77936	116483.0	13522	369	5750	...	
89	MW.TERAI	NaN	48622	96014	29732	76889	106621.0	10168	192	6762	...	
90	FW.MOUNTAIN	NaN	37637	23773	14035	16380	30415.0	2684	223	1460	...	
91	FW.HILLS	NaN	45036	39569	22850	33505	56355.0	5692	14	3103	...	
94	FW.TERAI	NaN	47922	68915	51051	62553	113604.0	9778	98	2330	...	

20 rows × 26 columns

Figure 122: Total yak/nak/chauri data from different regions.

```
# Total Yak/Nak/Chauri population in different region in Nepal
yak_num = yak_numbers[['DISTRICT','YAK/NAK/CHAURI']]

# Reshaping
yak_reshape= pd.melt(yak_num, id_vars="DISTRICT",value_name="Yak
Population",var_name="YAK/NAK/CHAURI")
plt.figure(figsize=(30,10))

# barplot to visualize the above data
sns.barplot(x="DISTRICT", y="Yak Population", data= yak_reshape,
hue="YAK/NAK/CHAURI", palette="afmhot")

plt.title('Total Yak/Nak/Chauri population in different region in
Nepal',fontsize=16, loc='left')

plt.xlabel('Development Region')

plt.show() # Display a figure
```

Figure 123: Barplot code for Yak/Nak/Chauri

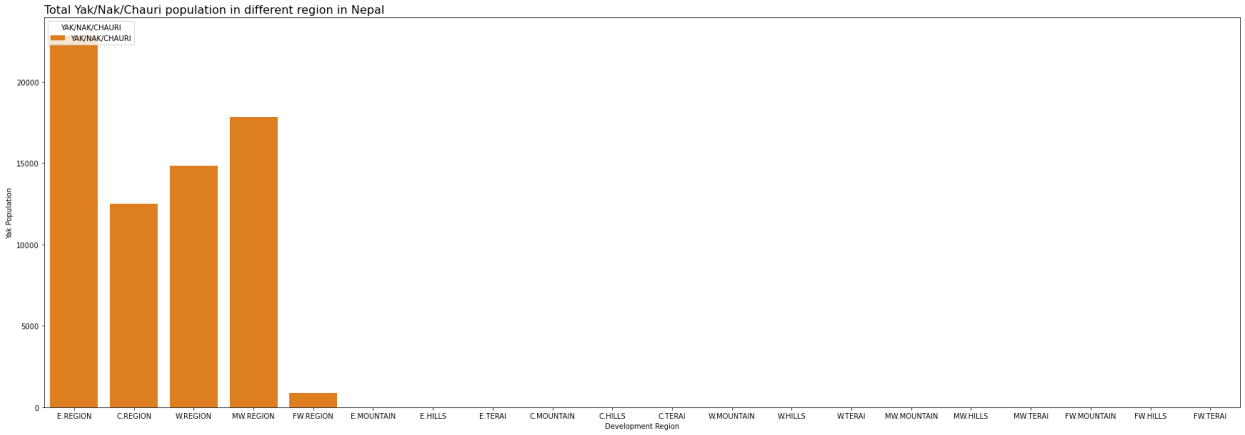

Figure 124: Barplot showing yak/nak/chauri population in different regions

The above barplot depicts the distribution of yak/nak/chauri populations in Nepal's various regions. We can see that the number of yak/nak/chauri is very high in the eastern region of Nepal, whereas it is shallow in the far-western region. The second highest number is found in Nepal's mid-western region, while the second lowest is found in the country's central region. And the western region's total population is located between the eastern and mid-western regions.

Conclusion

The preceding report analyses two distinct fields education (students' achievement from the two different schools named "Gabriel Pereira", and "Moushinho da Silveira") and livestock data from Nepal. Although all of the necessary datasets were provided, the necessary cleaning and merging were performed to conduct further analysis. This coursework has aided in improving knowledge about slicing, list comprehension, creating different functions, and many other topics. We also learned about different libraries used in the analysis process, such as NumPy, Seaborn, Pandas, and matplotlib. During the analysis, various logic was developed and implemented.

There were various activities that had to be completed as part of this coursework. First, all of the provided datasets were saved as .csv files, then converted into pandas dataframes. The data was then refined for analysis by merging and cleaning as needed. The final merged dataframes were then used for data exploration and visualisation. Finally, the analysis was carried out using various plotting functions of matplotlib, and the resulting representations were interpreted. Furthermore, information about graph analysis was obtained from this coursework by researching. As a result, our overall coursework has been highly beneficial in developing our technical and analytical skills.

Thank you!