Topic_4_Basic_programming_concepts

March 4, 2025

0.1 Data Series

1. In Pandas, series are labeled/indexed one-dimensional arrays.
2. The elements of the series could be integer, float, string, objects, etc.
3. The index may not be unique.
4. A collection of Data Series is called as Data Frame.
5. Our focus in this chapter will be on Data Frames.

1 DataFrame (DF)

Collection of series, where each column is a series.

1.1 Creating DataFrame

In order to create a data from, following things are typically needed:

1. Data in n-dimensional array format.
2. Optional column title/header

An optional row index can be given too. In addition to the above, there are many parameters, which can be read from pydata docs.

1.1.1 Create DF with Given n-D Array

Data:

$$\begin{bmatrix} a11 & a12 \\ a21 & a22 \end{bmatrix} \tag{1}$$

col_names: [‘col 1’, ‘col 2’]
row_index: [‘row 1’, ‘row 2’]

[9]:
```
%pip install pandas
```

```
Requirement already satisfied: pandas in
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (2.2.3)
Requirement already satisfied: numpy>=1.23.2 in
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (from
pandas) (2.2.3)
Requirement already satisfied: python-dateutil>=2.8.2 in
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (from
```

```
pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (from
pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (from
pandas) (2025.1)
Requirement already satisfied: six>=1.5 in
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (from
python-dateutil>=2.8.2->pandas) (1.17.0)

[notice] A new release of pip is
available: 24.0 -> 25.0.1
[notice] To update, run:
pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

[10]:

```
import pandas as pd
df = pd.DataFrame([['a11', 'a12'], ['a21', 'a22']],
                  index=['row 1', 'row 2'],
                  columns=['Course', 'Major'])

display(df)
```

```
      Course Major
row 1    a11   a12
row 2    a21   a22
```

1.1.2 Create DF with Random n-D Array

Data: random 3 columns 1000 rows numbers between 0 and 1.
col_names: ['col 1', 'col 2', 'col 3']

[11]:

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.rand(1000,3),
                  columns=['col 1', 'col 2', 'col 3'])

display(df)
```

```
        col 1     col 2     col 3
0    0.882961  0.598013  0.231091
1    0.476830  0.598566  0.082018
2    0.524957  0.443243  0.648673
3    0.849683  0.824520  0.425529
4    0.868353  0.729919  0.060088
..        …         …         …
995  0.737923  0.805233  0.051131
```

```
996   0.381251  0.677263  0.458775
997   0.056614  0.426475  0.718898
998   0.398317  0.911468  0.236637
999   0.462512  0.609815  0.950087

[1000 rows x 3 columns]
```

1.1.3 Create DF from CSV

Data: Basic-1.csv
col_names: Use from the first row

1. Display the above data.
2. Obtain the summary statistics:

- For numeric data (count, mean, std, min, max)
- For non-numeric data (count, unique-values, most occurring and its frequency)

[12]:
```
#1. Display the above data.
import pandas as pd
df = pd.read_csv('data/Basic-1.csv', delimiter=',')#the path to the data file.

## Path relative to the .ipynb file
# df = pd.read_csv('Basic-101.csv',␣
↪delimiter=',',header=None,names=['Gender','Job Type',"Province"])

# display data
display(df)

#2. Obtain the summary statistics:
# summary statistics for non-numeric columns is available by default only when␣
↪all columns are non-numeric
display(df.describe())
```

```
     Gender      Job Type Province
0         M   Pink-collar    Hejaz
1         M  White-collar  Central
2         M   Pink-collar    Hejaz
3         M   Pink-collar    Hejaz
4         M   Gold-collar  Eastern
…       …             …       …
1273      M   Blue-collar  Central
1274      M   Pink-collar    Hejaz
1275      F   Blue-collar    Hejaz
1276      F  White-collar  Central
1277      F   Blue-collar  Eastern

[1278 rows x 3 columns]

       Gender      Job Type Province
```

```
count    1278          1278      1278
unique      2             4         3
top         M  White-collar   Central
freq      875           568       531
```

1.1.4 Create DF from XLSX

Data: Basic-2.xlsx
col_names: There are 17 columns, defined respectively as: 1. dur: duration of agreement 2. wage1.wage : wage increase in first year of contract 3. wage2.wage : wage increase in second year of contract 4. wage3.wage : wage increase in third year of contract 5. cola : cost of living allowance 6. hours.hrs : number of working hours during week 7. pension : employer contributions to pension plan 8. stby_pay : standby pay 9. shift_diff : shift differencial : supplement for work on II and III shift 10. educ_allw.boolean : education allowance 11. holidays : number of statutory holidays 12. vacation : number of paid vacation days 13. lngtrm_disabil.boolean : employer's help during employee longterm disability 14. dntl_ins : employers contribution towards the dental plan 15. bereavement.boolean : employer's financial contribution towards the covering the costs of bereavement 16. empl_hplan : employer's contribution towards the health plan 17. empl_hplan : employer's contribution towards the health plan

For the above data, do the following: 1. Display the above data. 2. Obtain the summary statistics: - For numeric data (count, mean, std, min, max) - For non-numeric data (count, unique-values, most occurring and its frequency)

[13]:
```
%pip install openpyxl
```

```
Requirement already satisfied: openpyxl in
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (3.1.5)
Requirement already satisfied: et-xmlfile in
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (from
openpyxl) (2.0.0)

[notice] A new release of pip is
available: 24.0 -> 25.0.1
[notice] To update, run:
pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

[14]:
```
# 1. Display the above data.
import pandas as pd

df = pd.read_excel('data/Basic-2.xlsx')

#display(df.head())
display(df.tail())
#df.sample(5)
```

```
     c1   c2   c3   c4    c5    c6        c7   c8   c9  c10   c11  \
35  2.0  2.0  2.0  NaN  none  40.0      none  NaN  NaN   no  11.0
36  1.0  2.0  NaN  NaN    tc  40.0  ret_allw  4.0  0.0   no  11.0
```

```
37  1.0  2.8  NaN  NaN  none  38.0  empl_contr  2.0  3.0   no   9.0
38  3.0  2.0  2.5  2.0   NaN  37.0  empl_contr  NaN  NaN  NaN  10.0
39  2.0  4.5  4.0  NaN  none  40.0         NaN  NaN  4.0  NaN  12.0

              c12  c13   c14  c15   c16   c17
35        average  yes  none  yes  full   bad
36       generous   no  none   no  none   bad
37  below average  yes  half  NaN  none   bad
38        average  NaN   NaN  yes  none   bad
39        average  yes  full  yes  half  good
```

[15]:
```
# 2. Obtain the summary statistics:

#display(df.describe())
# by default the describe gives summary of the numeric columns, when there are␣
↪mixed columns
#df.describe(include='all') # can be used, but too many NaNs
```

[16]:
```
# to get summary of non-numeric columns

# Idenfity non-numeric columns
cat_columns= df.select_dtypes(include='object').columns

# Show summary of non-numeric columns
df[cat_columns].describe()

## or in one step
# df.select_dtypes(include='object').describe()
```

[16]:

	c5	c7	c10	c12	c13	c14	c15	c16	c17
count	24	18	18	37	16	25	20	24	40
unique	3	3	2	3	2	3	2	3	2
top	none	none	no	below average	yes	half	yes	full	good
freq	14	8	11	14	11	11	18	12	26

1.2 Accessing Elements

.loc and *.iloc* methods can be used for accessing the elements of a DF.

1.2.1 Accessing via .loc

Consider the dataset given in Basic-1.csv. Display second row, second column element.

[17]:
```
import pandas as pd
df = pd.read_csv('data/Basic-1.csv', delimiter=',')
print(df.loc[1,'Job Type'])
```

```
White-collar
```

1.2.2 Accessing via .iloc

Consider the dataset given in Basic-1.csv. Display second row, second column element.

[18]:
```
import pandas as pd
df = pd.read_csv('data/Basic-1.csv', delimiter=',')
display(df.head())
print(df.iloc[1,1])
print(df.loc[1,'Job Type'])
```

```
  Gender      Job Type Province
0      M   Pink-collar    Hejaz
1      M  White-collar  Central
2      M   Pink-collar    Hejaz
3      M   Pink-collar    Hejaz
4      M   Gold-collar  Eastern
```

```
White-collar
White-collar
```

1.3 DF Slicing

1. Slicing is nothing but extracting a part of the DF.
2. The part could be contiguous or broken chunks of the DF.

1.3.1 Single Column Slice

A single column can be extracted using the following methods: 1. df.COL_NAME 2. df[COL_NAME]

Consider the dataset given in Basic-2.xlsx. Display first column.

[19]:
```
import pandas as pd

df = pd.read_excel('data/Basic-2.xlsx')
display(df.head())
display(df['c1'])

## Other styles to select a column
# display(df.iloc[:,0])
# display(df.loc[:,'c1'])

## Following code can be used to create column names when dataset does not␣
↪contain any column names.
# col_name=[''.join(['c',str(i+1)])  for i in range(17)]
## Following options can be used to add column names to the dataset
# header=None, names=col_name
```

```
    c1   c2   c3   c4   c5    c6          c7  c8   c9  c10   c11  \
0  1.0  5.0  NaN  NaN  NaN  40.0         NaN NaN  2.0  NaN  11.0
1  2.0  4.5  5.8  NaN  NaN  35.0    ret_allw NaN  NaN  yes  11.0
```

```
2  NaN  NaN  NaN  NaN  NaN  38.0  empl_contr NaN  5.0  NaN  11.0
3  3.0  3.7  4.0  5.0   tc   NaN         NaN NaN  NaN  yes   NaN
4  3.0  4.5  4.5  5.0  NaN  40.0         NaN NaN  NaN  NaN  12.0

             c12  c13   c14  c15   c16   c17
0        average  NaN   NaN  yes   NaN  good
1  below average  NaN  full  NaN  full  good
2       generous  yes  half  yes  half  good
3            NaN  NaN   NaN  yes   NaN  good
4        average  NaN  half  yes  half  good
```

```
0     1.0
1     2.0
2     NaN
3     3.0
4     3.0
5     2.0
6     3.0
7     3.0
8     2.0
9     1.0
10    3.0
11    2.0
12    2.0
13    3.0
14    1.0
15    2.0
16    1.0
17    1.0
18    1.0
19    2.0
20    2.0
21    2.0
22    3.0
23    2.0
24    1.0
25    3.0
26    2.0
27    2.0
28    2.0
29    3.0
30    3.0
31    3.0
32    2.0
33    2.0
34    3.0
35    2.0
36    1.0
```

```
37     1.0
38     3.0
39     2.0
Name: c1, dtype: float64
```

1.3.2 Slicing via .loc

Consider the dataset given in Basic-2.xlsx. Display first 10 rows from the second, third, fourth and fifth column.

[20]:
```python
import pandas as pd
df = pd.read_excel('data/Basic-2.xlsx')

display(df.head())

display(df.loc[0:9,'c2':'c5'])  # ends are inclusive

# display(df.loc[0:9:2,'c2':'c5':2])  # ends are inclusive and steps can be␣
↪used too
```

```
    c1   c2   c3   c4   c5    c6          c7   c8   c9  c10   c11  \
0  1.0  5.0  NaN  NaN  NaN  40.0         NaN  NaN  2.0  NaN  11.0
1  2.0  4.5  5.8  NaN  NaN  35.0    ret_allw  NaN  NaN  yes  11.0
2  NaN  NaN  NaN  NaN  NaN  38.0  empl_contr  NaN  5.0  NaN  11.0
3  3.0  3.7  4.0  5.0   tc   NaN         NaN  NaN  NaN  yes   NaN
4  3.0  4.5  4.5  5.0  NaN  40.0         NaN  NaN  NaN  NaN  12.0

             c12  c13   c14  c15   c16   c17
0        average  NaN   NaN  yes   NaN  good
1  below average  NaN  full  NaN  full  good
2       generous  yes  half  yes  half  good
3            NaN  NaN   NaN  yes   NaN  good
4        average  NaN  half  yes  half  good
```

```
    c2   c3   c4    c5
0  5.0  NaN  NaN   NaN
1  4.5  5.8  NaN   NaN
2  NaN  NaN  NaN   NaN
3  3.7  4.0  5.0    tc
4  4.5  4.5  5.0   NaN
5  2.0  2.5  NaN   NaN
6  4.0  5.0  5.0    tc
7  6.9  4.8  2.3   NaN
8  3.0  7.0  NaN   NaN
9  5.7  NaN  NaN  none
```

[21]:
```python
#pip install openpyxl
```

1.3.3 Slicing via .iloc

Consider the dataset given in Basic-2.xlsx. Display first 10 rows from the second, third, fourth and fifth column.

[22]:
```
import pandas as pd
df = pd.read_excel('data/Basic-2.xlsx')

display(df.iloc[:10,1:5])
```

```
    c2   c3   c4    c5
0  5.0  NaN  NaN   NaN
1  4.5  5.8  NaN   NaN
2  NaN  NaN  NaN   NaN
3  3.7  4.0  5.0    tc
4  4.5  4.5  5.0   NaN
5  2.0  2.5  NaN   NaN
6  4.0  5.0  5.0    tc
7  6.9  4.8  2.3   NaN
8  3.0  7.0  NaN   NaN
9  5.7  NaN  NaN  none
```

[23]:
```
## Example of loc and iloc
df = pd.DataFrame([['a11', 'a12'], ['a21', 'a22']],
                  index=['row 1', 'row 2'],
                  columns=['col 1', 'col-2'])

display(df)


# display(df.loc[:,'col 1'])  # Select one column

# display(df['col 1'])  # Select one column

# display(df.col-2) # # Select one column, problem is Spaces or Special␣
 ↪characters

# display(df.loc['row 2','col 1'])

# display(df.loc['row 2',:])

# display(df['row 2']) ## check...


# display(df.iloc[:,0])  # Select one column

# display(df.iloc[1,0]) # row 2 column 1
```

```
      col 1 col-2
```

```
row 1   a11   a12
row 2   a21   a22
```

1.4 DF Sorting

1. DataFrames can be sorted w.r.t values in the columns.
2. One or more columns can be used for sorting.

Consider the dataset given in Basic-2.xlsx. Sort the rows according to the first column, in ascending order. Repeat the sort in descending order.

[24]:
```
import pandas as pd

#Read data
df = pd.read_excel('data/Basic-2.xlsx')
display(df.head())

# # sort by single column in ascending order
df.sort_values(by=['c1'], inplace=True)
display(df.head())

# # sort by single column in descending order
df.sort_values(by=['c1'], inplace=True, ascending=False)
display(df.head())
```

```
    c1   c2   c3   c4   c5    c6          c7  c8   c9  c10   c11  \
0  1.0  5.0  NaN  NaN  NaN  40.0         NaN NaN  2.0  NaN  11.0
1  2.0  4.5  5.8  NaN  NaN  35.0    ret_allw NaN  NaN  yes  11.0
2  NaN  NaN  NaN  NaN  NaN  38.0  empl_contr NaN  5.0  NaN  11.0
3  3.0  3.7  4.0  5.0   tc   NaN         NaN NaN  NaN  yes   NaN
4  3.0  4.5  4.5  5.0  NaN  40.0         NaN NaN  NaN  NaN  12.0

             c12  c13   c14  c15   c16   c17
0        average  NaN   NaN  yes   NaN  good
1  below average  NaN  full  NaN  full  good
2       generous  yes  half  yes  half  good
3            NaN  NaN   NaN  yes   NaN  good
4        average  NaN  half  yes  half  good
     c1   c2  c3  c4    c5    c6          c7  c8    c9  c10   c11  \
0   1.0  5.0 NaN NaN   NaN  40.0         NaN NaN   2.0  NaN  11.0
16  1.0  2.8 NaN NaN   NaN  35.0         NaN NaN   2.0  NaN  12.0
14  1.0  3.0 NaN NaN  none  36.0         NaN NaN  10.0   no  11.0
9   1.0  5.7 NaN NaN  none  40.0  empl_contr NaN   4.0  NaN  11.0
18  1.0  2.0 NaN NaN  none  38.0        none NaN   NaN  yes  11.0

              c12  c13   c14  c15   c16   c17
0         average  NaN   NaN  yes   NaN  good
16  below average  NaN   NaN  NaN   NaN  good
```

```
14        generous  NaN   NaN  NaN   NaN  good
9         generous  yes  full  NaN   NaN  good
18         average   no  none   no  none   bad

     c1   c2   c3   c4    c5    c6          c7   c8   c9  c10   c11  \
22  3.0  3.5  4.0  4.6   tcf  27.0         NaN  NaN  NaN  NaN   NaN
34  3.0  2.0  2.5  2.1    tc  40.0        none  2.0  1.0   no  10.0
29  3.0  2.0  2.5  NaN   NaN  35.0        none  NaN  NaN  NaN  10.0
38  3.0  2.0  2.5  2.0   NaN  37.0  empl_contr  NaN  NaN  NaN  10.0
30  3.0  4.5  4.5  5.0  none  40.0         NaN  NaN  NaN   no  11.0


              c12  c13   c14  c15   c16   c17
22            NaN  NaN   NaN  NaN   NaN  good
34  below average   no  half  yes  full   bad
29        average  NaN   NaN  yes  full   bad
38        average  NaN   NaN  yes  none   bad
30        average  NaN  half  NaN   NaN  good
```

Consider the dataset given in Basic-2.xlsx. Sort the rows according to the first and then sixth column, where ascending in the first and descending in the sixth column. Display only first 6 columns.

[25]:
```python
# sort by multiple columns

df.sort_values(by=['c1','c6'], inplace=True, ascending=[True, False])

display(df.iloc[:,0:6])
```

```
     c1   c2   c3   c4    c5    c6
0   1.0  5.0  NaN  NaN   NaN  40.0
17  1.0  2.1  NaN  NaN    tc  40.0
9   1.0  5.7  NaN  NaN  none  40.0
36  1.0  2.0  NaN  NaN    tc  40.0
24  1.0  6.0  NaN  NaN   NaN  38.0
18  1.0  2.0  NaN  NaN  none  38.0
37  1.0  2.8  NaN  NaN  none  38.0
14  1.0  3.0  NaN  NaN  none  36.0
16  1.0  2.8  NaN  NaN   NaN  35.0
23  2.0  4.5  4.0  NaN   NaN  40.0
35  2.0  2.0  2.0  NaN  none  40.0
12  2.0  3.5  4.0  NaN  none  40.0
33  2.0  4.0  5.0  NaN  none  40.0
21  2.0  2.5  3.0  NaN   NaN  40.0
39  2.0  4.5  4.0  NaN  none  40.0
11  2.0  6.4  6.4  NaN   NaN  38.0
8   2.0  3.0  7.0  NaN   NaN  38.0
32  2.0  2.5  2.5  NaN   NaN  38.0
20  2.0  4.3  4.4  NaN   NaN  38.0
28  2.0  5.0  4.0  NaN  none  37.0
```

```
15  2.0  4.5  4.0  NaN  none  37.0
19  2.0  4.0  5.0  NaN   tcf  35.0
1   2.0  4.5  5.8  NaN   NaN  35.0
5   2.0  2.0  2.5  NaN   NaN  35.0
27  2.0  3.0  3.0  NaN  none  33.0
26  2.0  4.5  4.5  NaN   tcf   NaN
34  3.0  2.0  2.5  2.1    tc  40.0
30  3.0  4.5  4.5  5.0  none  40.0
31  3.0  3.0  2.0  2.5    tc  40.0
7   3.0  6.9  4.8  2.3   NaN  40.0
4   3.0  4.5  4.5  5.0   NaN  40.0
25  3.0  2.0  2.0  2.0  none  40.0
38  3.0  2.0  2.5  2.0   NaN  37.0
13  3.0  3.5  4.0  5.1   tcf  37.0
10  3.0  3.5  4.0  4.6  none  36.0
29  3.0  2.0  2.5  NaN   NaN  35.0
22  3.0  3.5  4.0  4.6   tcf  27.0
3   3.0  3.7  4.0  5.0    tc   NaN
6   3.0  4.0  5.0  5.0    tc   NaN
2   NaN  NaN  NaN  NaN   NaN  38.0
```

1.5 DF Selecting

Selecting is similar to the slicing. Typically, it deals with the following: 1. We are typically interested in extracting rows based on conditions. 2. Conditions are based on the values of one or more columns.

Consider the dataset given in Basic-1.csv. Display all rows related to White-collar jobs.

[26]:
```
import pandas as pd
df = pd.read_csv('data/Basic-1.csv', delimiter=',')
display(df.head())

seleted_rows = df['Job Type']=='White-collar'
display(df.loc[seleted_rows,:])
```

```
  Gender      Job Type Province
0      M   Pink-collar    Hejaz
1      M  White-collar  Central
2      M   Pink-collar    Hejaz
3      M   Pink-collar    Hejaz
4      M   Gold-collar  Eastern
```

```
     Gender      Job Type Province
1         M  White-collar  Central
5         M  White-collar    Hejaz
8         M  White-collar  Eastern
9         F  White-collar    Hejaz
10        M  White-collar    Hejaz
```

```
…     …             …       …
1266       M  White-collar    Hejaz
1267       M  White-collar    Hejaz
1271       F  White-collar  Central
1272       M  White-collar  Central
1276       F  White-collar  Central

[568 rows x 3 columns]
```

Consider the dataset given in Basic-1.csv. 1. Display all rows related to White-collar jobs in Eastern Province. 2. Display the statistical summary of Gender column in the above selection.

[27]:
```python
import pandas as pd
df = pd.read_csv('data/Basic-1.csv', delimiter=',')

seleted_rows = (df['Province']=='Eastern') & (df['Job Type']=='White-collar')
display(df.loc[seleted_rows,:])


# Summary
display(df.loc[seleted_rows,:].describe().iloc[:,0])
# display(df.loc[seleted_rows,:].describe())
# display(df.loc[seleted_rows,:].describe().loc[:,'Gender'])
```

```
     Gender      Job Type Province
8         M  White-collar  Eastern
18        F  White-collar  Eastern
27        M  White-collar  Eastern
31        M  White-collar  Eastern
37        M  White-collar  Eastern
…     …             …       …
1226      M  White-collar  Eastern
1232      M  White-collar  Eastern
1238      F  White-collar  Eastern
1241      M  White-collar  Eastern
1256      M  White-collar  Eastern

[138 rows x 3 columns]

count     138
unique      2
top         M
freq       98
Name: Gender, dtype: object
```

1.5.1 Example: Displaying Data

Question-A: Consider the dataset given in Basic-1.csv. 1. Display all rows related to White-collar and Blue-collar jobs. 2. Display all rows related to White-collar and Blue-collar jobs for Females.

3. Display the statistical summary of Province column in the above selection. 4. Among all the White-collar and Blue-collar job Females, what proportion works in Eastern Province.

[28]:
```
#1. Display all rows related to White-collar and Blue-collar jobs.

import pandas as pd
df = pd.read_csv('data/Basic-1.csv', delimiter=',')
display(df)

seleted_rows = df['Job Type'].isin(['White-collar','Blue-collar'])

# seleted_rows = (df['Job Type']=='White-collar') | (df['Job␣
↪Type']=='Blue-collar')

display(df.loc[seleted_rows,:])
```

```
     Gender      Job Type Province
0         M   Pink-collar    Hejaz
1         M  White-collar  Central
2         M   Pink-collar    Hejaz
3         M   Pink-collar    Hejaz
4         M   Gold-collar  Eastern
…      …           …       …
1273      M   Blue-collar  Central
1274      M   Pink-collar    Hejaz
1275      F   Blue-collar    Hejaz
1276      F  White-collar  Central
1277      F   Blue-collar  Eastern

[1278 rows x 3 columns]
```

```
     Gender      Job Type Province
1         M  White-collar  Central
5         M  White-collar    Hejaz
6         M   Blue-collar  Eastern
7         M   Blue-collar  Central
8         M  White-collar  Eastern
…      …           …       …
1272      M  White-collar  Central
1273      M   Blue-collar  Central
1275      F   Blue-collar    Hejaz
1276      F  White-collar  Central
1277      F   Blue-collar  Eastern

[984 rows x 3 columns]
```

[29]:
```
# 2. Display all rows related to White-collar and Blue-collar jobs for Females.
```

```
import pandas as pd
df = pd.read_csv('data/Basic-1.csv', delimiter=',')

seleted_rows = df['Job Type'].isin(['White-collar','Blue-collar'])  & ␣
↪(df['Gender']=='F')

display(df.loc[seleted_rows,:])

# 3. Display the statistical summary of Province column in the above selection.
df.loc[seleted_rows,['Province']].describe()
```

```
     Gender      Job Type Province
9         F  White-collar    Hejaz
11        F   Blue-collar  Central
13        F  White-collar  Central
14        F  White-collar    Hejaz
15        F  White-collar  Central
…      …           …       …
1254      F  White-collar    Hejaz
1271      F  White-collar  Central
1275      F   Blue-collar    Hejaz
1276      F  White-collar  Central
1277      F   Blue-collar  Eastern

[311 rows x 3 columns]
```

[29]:

	Province
count	311
unique	3
top	Central
freq	136

[30]:

```
# 4. Among all the White-collar and Blue-collar job Females, what proportion␣
↪works in Eastern Province.
ndf = df.loc[seleted_rows,'Province']
display(ndf)
print(ndf.value_counts())

proportion = ndf.value_counts()[2]/len(ndf.index)*100

## to get length of dataframe
# print(len(ndf.index))
# print(ndf.shape[0])
# print(len(ndf))
```

```
print(f'The proportion of Females working as blue/white collar in Eastern␣
↪province is: {proportion: 0.2f}%')
```

```
9         Hejaz
11      Central
13      Central
14        Hejaz
15      Central
         …
1254      Hejaz
1271    Central
1275      Hejaz
1276    Central
1277    Eastern
Name: Province, Length: 311, dtype: object
```

```
Province
Central    136
Hejaz      104
Eastern     71
Name: count, dtype: int64
The proportion of Females working as blue/white collar in Eastern province is:
22.83%
```

```
/tmp/ipykernel_21727/2696248615.py:6: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  proportion = ndf.value_counts()[2]/len(ndf.index)*100
```

1.6 Addition DF Methods

1.6.1 Count Rows and Columns

Consider the dataset given in Basic-2.csv. 1. Count the number of rows and columns. 2. Count the number of non-null rows for each column.

[31]:
```
import pandas as pd
df = pd.read_excel('data/Basic-2.xlsx')
display(df.head())

print(f'The number of rows are {len(df.index)}, and the number of columns are␣
↪{len(df.columns)}')

print(f'The number of non-null rows for each column are:\n{df.count()}')

print(f'The number of null rows for each column are:\n{df.isna().sum()}')

print(df.shape)
```

```
    c1   c2   c3   c4   c5    c6          c7  c8   c9  c10   c11  \
0  1.0  5.0  NaN  NaN  NaN  40.0         NaN NaN  2.0  NaN  11.0
1  2.0  4.5  5.8  NaN  NaN  35.0    ret_allw NaN  NaN  yes  11.0
2  NaN  NaN  NaN  NaN  NaN  38.0  empl_contr NaN  5.0  NaN  11.0
3  3.0  3.7  4.0  5.0   tc   NaN         NaN NaN  NaN  yes   NaN
4  3.0  4.5  4.5  5.0  NaN  40.0         NaN NaN  NaN  NaN  12.0

             c12  c13   c14  c15   c16   c17
0        average  NaN   NaN  yes   NaN  good
1  below average  NaN  full  NaN  full  good
2       generous  yes  half  yes  half  good
3            NaN  NaN   NaN  yes   NaN  good
4        average  NaN  half  yes  half  good
The number of rows are 40, and the number of columns are 17
The number of non-null rows for each column are:
c1     39
c2     39
c3     30
c4     12
c5     24
c6     37
c7     18
c8      7
c9     24
c10    18
c11    38
c12    37
c13    16
c14    25
c15    20
c16    24
c17    40
dtype: int64
The number of null rows for each column are:
c1      1
c2      1
c3     10
c4     28
c5     16
c6      3
c7     22
c8     33
c9     16
c10    22
c11     2
c12     3
c13    24
```

```
c14    15
c15    20
c16    16
c17     0
dtype: int64
(40, 17)
```

1.6.2 Apply Lambda Functions

Lambda functions can be applied to each row or column using *.apply()* method.

Consider the data in file Basic-2-Clean.csv. 1. In columns c1, c6 and c11, convert every number to the nearest integer. 2. In column c12, replace the space between two words by underscore.

[32]:

```python
# 1. In columns c1, c6 and c11, convert every number to the nearest integer.
import pandas as pd
df = pd.read_csv('data/Basic-2-Clean.csv')
display(df.head())

df['c1']=df['c1'].apply(lambda x: int(round(x))) # apply for single column
# df['c1']= (lambda x: round(x))(df['c1'])  #another style

df.loc[:,['c6','c11']]=df.loc[:,['c6','c11']].applymap(lambda x: int(round(x))␣
 ↪) #for multiple columns

display(df.head())
```

```
         c1        c2        c3    c5         c6        c9        c11  \
0  1.000000  5.000000  3.913333  none  40.000000  2.000000  11.000000
1  2.000000  4.500000  5.800000  none  35.000000  4.583333  11.000000
2  2.102564  3.620513  3.913333  none  38.000000  5.000000  11.000000
3  3.000000  3.700000  4.000000    tc  37.810811  4.583333  11.105263
4  3.000000  4.500000  4.500000  none  40.000000  4.583333  12.000000

             c12  c14  c16  c17
0        average  half  full  good
1  below average  full  full  good
2       generous  half  half  good
3  below average  half  full  good
4        average  half  half  good

/tmp/ipykernel_21727/3215444923.py:9: FutureWarning: DataFrame.applymap has been
deprecated. Use DataFrame.map instead.
  df.loc[:,['c6','c11']]=df.loc[:,['c6','c11']].applymap(lambda x: int(round(x))
) #for multiple columns

   c1        c2        c3    c5    c6        c9   c11            c12   c14  \
0   1  5.000000  3.913333  none  40.0  2.000000  11.0        average  half
1   2  4.500000  5.800000  none  35.0  4.583333  11.0  below average  full
2   2  3.620513  3.913333  none  38.0  5.000000  11.0       generous  half
```

```
3   3  3.700000  4.000000    tc  38.0  4.583333  11.0  below average  half
4   3  4.500000  4.500000  none  40.0  4.583333  12.0        average  half

    c16   c17
0  full  good
1  full  good
2  half  good
3  full  good
4  half  good
```

[33]:
```
# 2. In column c12, replace the space between two words by underscore.
display(df.head())
df['c12']=df['c12'].apply(lambda x: x.replace(' ','_'))
display(df.head())
```

```
   c1        c2        c3    c5    c6        c9   c11            c12   c14  \
0   1  5.000000  3.913333  none  40.0  2.000000  11.0        average  half
1   2  4.500000  5.800000  none  35.0  4.583333  11.0  below average  full
2   2  3.620513  3.913333  none  38.0  5.000000  11.0       generous  half
3   3  3.700000  4.000000    tc  38.0  4.583333  11.0  below average  half
4   3  4.500000  4.500000  none  40.0  4.583333  12.0        average  half

    c16   c17
0  full  good
1  full  good
2  half  good
3  full  good
4  half  good
```

```
   c1        c2        c3    c5    c6        c9   c11            c12   c14  \
0   1  5.000000  3.913333  none  40.0  2.000000  11.0        average  half
1   2  4.500000  5.800000  none  35.0  4.583333  11.0  below_average  full
2   2  3.620513  3.913333  none  38.0  5.000000  11.0       generous  half
3   3  3.700000  4.000000    tc  38.0  4.583333  11.0  below_average  half
4   3  4.500000  4.500000  none  40.0  4.583333  12.0        average  half

    c16   c17
0  full  good
1  full  good
2  half  good
3  full  good
4  half  good
```

1.6.3 Apply General Functions

Similar to Lambda functions, user defined or inbuilt functions can be applied to each row or column using *.apply()* method. Moreover, *.applymap()* method can be used for multiple columns.

Consider the data in file Basic-2-Clean.csv.

In columns c2, c3 and c9, round the values to the nearest 0, 0.5 or 1. If any value is negative, then replace it with zero.

[34]:
```python
def custom_round(x):
    if x <=0:
        return 0
    int_x = int(x)
    if (x <= int_x+0.25):
        return int_x
    elif  (x > int_x+0.25) and (x <= int_x+0.75):
        return int_x+0.5
    else:
        return int_x+1

import pandas as pd
df = pd.read_csv('data/Basic-2-Clean.csv')
display(df.head())
# df['c2']=df['c2'].apply(custom_round)  # for single column
df.loc[:,['c2','c3','c9']]=df.loc[:,['c2','c3','c9']].applymap(custom_round)  #
↪for multiple columns
display(df.head())
```

```
         c1        c2        c3    c5         c6        c9        c11  \
0  1.000000  5.000000  3.913333  none  40.000000  2.000000  11.000000
1  2.000000  4.500000  5.800000  none  35.000000  4.583333  11.000000
2  2.102564  3.620513  3.913333  none  38.000000  5.000000  11.000000
3  3.000000  3.700000  4.000000    tc  37.810811  4.583333  11.105263
4  3.000000  4.500000  4.500000  none  40.000000  4.583333  12.000000

             c12  c14  c16  c17
0        average  half  full  good
1  below average  full  full  good
2       generous  half  half  good
3  below average  half  full  good
4        average  half  half  good
```

```
/tmp/ipykernel_21727/1219133502.py:16: FutureWarning: DataFrame.applymap has
been deprecated. Use DataFrame.map instead.
  df.loc[:,['c2','c3','c9']]=df.loc[:,['c2','c3','c9']].applymap(custom_round)
# for multiple columns
```

```
         c1   c2   c3    c5         c6   c9        c11            c12   c14  \
0  1.000000  5.0  4.0  none  40.000000  2.0  11.000000        average  half
1  2.000000  4.5  6.0  none  35.000000  4.5  11.000000  below average  full
2  2.102564  3.5  4.0  none  38.000000  5.0  11.000000       generous  half
3  3.000000  3.5  4.0    tc  37.810811  4.5  11.105263  below average  half
4  3.000000  4.5  4.5  none  40.000000  4.5  12.000000        average  half
```

```
   c16   c17
0  full  good
1  full  good
2  half  good
3  full  good
4  half  good
```

2 Data Visualization

Graphical representation of data

2.1 Why Visualization

1. When data is small (no magic numbers, but say 5 rows and 2 columns), then one may extract meaningful patterns by looking at the data.
2. However, when we have large data (again no magic numbers, but say 5000 rows and 2000 columns), then mere observation may not be fruitful in identifying patterns.
3. Visualization is the first typical step in data analysis.

2.2 Typical Visualizations

1. Histogram
2. Box Plots
3. Scatter Plots
4. Line Plots

2.3 Pandas Plots

1. Pandas provide in-built basic plots using the numpy and matplotlib libraries.
2. Several plots from pandas are under df.plot, which can be accessed via *kind* option.
3. In addition to that, specific plots like histogram (df.hist()), and boxplot (df.boxplot()) are also available.
4. However, pandas basic plots are not enough, and in the course we look at the seaborn library for plotting.

In the following cells, we will look at most typical plots that can be constructed using seaborn plots.

2.4 Seaborn Plots

1. Seaborn library provides a variety of plots, inlcluding histogram, boxplots, lineplots, scatterplots, countplot, violinplot, swarmplot, pairplot, catplot etc.
2. These plots can be accessed through *.countplot, .violinplot, catplot, etc.*
3. Typical parameters include
$$x, y, \; and/or \; hue$$
where x, y refers to the x and y axis, and *hue* defines subsets of the data, which will be drawn on separate facets in the grid.
4. It also comes with huge customization.

5. The Seaborn Plots are aesthetically better than basic plots from pandas library.
6. The library requires numpy and matplotlib libraries.
7. It is integrated with pandas data.

In the following cell, we will look at most typical plots from the seaborn library.

2.4.1 Example: Seaborn's Basic Plots

Question-B: Consider the data in file Basic-3.csv.

1. Read and display information the data.
2. Add a new column to dataframe that represents total score. The total score is sum of score of math, reading, and writing scores.
3. Draw histograms of both numeric and non-numeric columns.
4. Draw box-plots for the numeric columns, and differentiate by *test preparation course* column.
5. Draw three overlapping scatter plots of columns *math score, reading score* and *writing score* w.r.t *Total score*. Set the markers transparency level (alpha) to 0.5.
6. Draw scatter plot of columns *math score* and *writing score*, where the size of the marker is based on column *Total score*. In addition to that, reduce the marker transparency (alpha) to 0.5.
7. Plot *Total score* in ascending order on the x axis, and the corresponding pairwise absolute differences of *math score, reading score* and *writing score* on the y axis.

[35]:
```python
# 1. Read and display information the data.

import pandas as pd
df = pd.read_csv('data/Basic-3.csv')

display(df.head())
display(df.info())
```

```
   gender race/ethnicity parental level of education         lunch  \
0  female        group B           bachelor's degree      standard
1  female        group C                some college      standard
2  female        group B             master's degree      standard
3    male        group A          associate's degree  free/reduced
4    male        group C                some college      standard

  test preparation course  math score  reading score  writing score
0                    none          72             72             74
1               completed          69             90             88
2                    none          90             95             93
3                    none          47             57             44
4                    none          76             78             75

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
```

```
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental level of education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test preparation course      1000 non-null   object
 5   math score                   1000 non-null   int64
 6   reading score                1000 non-null   int64
 7   writing score                1000 non-null   int64
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
```

```
None
```

[36]:

```python
# 2. Add a new column to dataframe that represents total score.
# The total score is sum of score of math, reading, and writing scores

df['Total score']=df['math score']+df['reading score']+df['writing score']

df
```

[36]:

```
     gender race/ethnicity parental level of education         lunch  \
0    female        group B           bachelor's degree      standard
1    female        group C                some college      standard
2    female        group B             master's degree      standard
3      male        group A          associate's degree  free/reduced
4      male        group C                some college      standard
..      …            …                         …           …
995  female        group E             master's degree      standard
996    male        group C                 high school  free/reduced
997  female        group C                 high school  free/reduced
998  female        group D                some college      standard
999  female        group D                some college  free/reduced

    test preparation course  math score  reading score  writing score  \
0                      none          72             72             74
1                 completed          69             90             88
2                      none          90             95             93
3                      none          47             57             44
4                      none          76             78             75
..                      …          …             …             …
995               completed          88             99             95
996                    none          62             55             55
997               completed          59             71             65
998               completed          68             78             77
999                    none          77             86             86

     Total score
0            218
```

```
1          247
2          278
3          148
4          229
..         …
995        282
996        172
997        195
998        223
999        249

[1000 rows x 9 columns]
```

[37]:

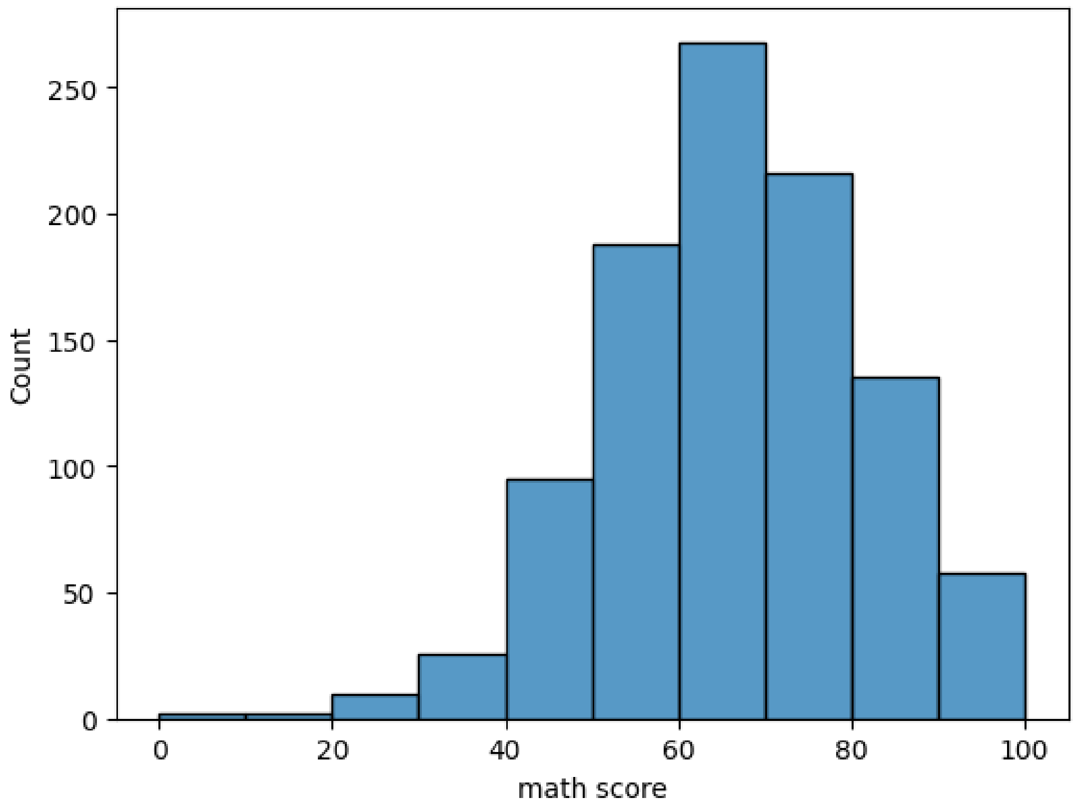
```
%pip install seaborn
```

```
Collecting seaborn
  Downloading seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (from
seaborn) (2.2.3)
Requirement already satisfied: pandas>=1.2 in
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (from
seaborn) (2.2.3)
Collecting matplotlib!=3.6.1,>=3.4 (from seaborn)
  Downloading matplotlib-3.10.1-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib!=3.6.1,>=3.4->seaborn)
  Downloading contourpy-1.3.1-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (5.4 kB)
Collecting cycler>=0.10 (from matplotlib!=3.6.1,>=3.4->seaborn)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib!=3.6.1,>=3.4->seaborn)
  Downloading fonttools-4.56.0-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (101 kB)
                          101.9/101.9 kB
682.0 kB/s eta 0:00:00a 0:00:01
Collecting kiwisolver>=1.3.1 (from matplotlib!=3.6.1,>=3.4->seaborn)
  Downloading kiwisolver-1.4.8-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (6.2 kB)
Requirement already satisfied: packaging>=20.0 in
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (24.2)
Collecting pillow>=8 (from matplotlib!=3.6.1,>=3.4->seaborn)
  Downloading pillow-11.1.0-cp311-cp311-manylinux_2_28_x86_64.whl.metadata (9.1
kB)
Collecting pyparsing>=2.3.1 (from matplotlib!=3.6.1,>=3.4->seaborn)
  Downloading pyparsing-3.2.1-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in
```

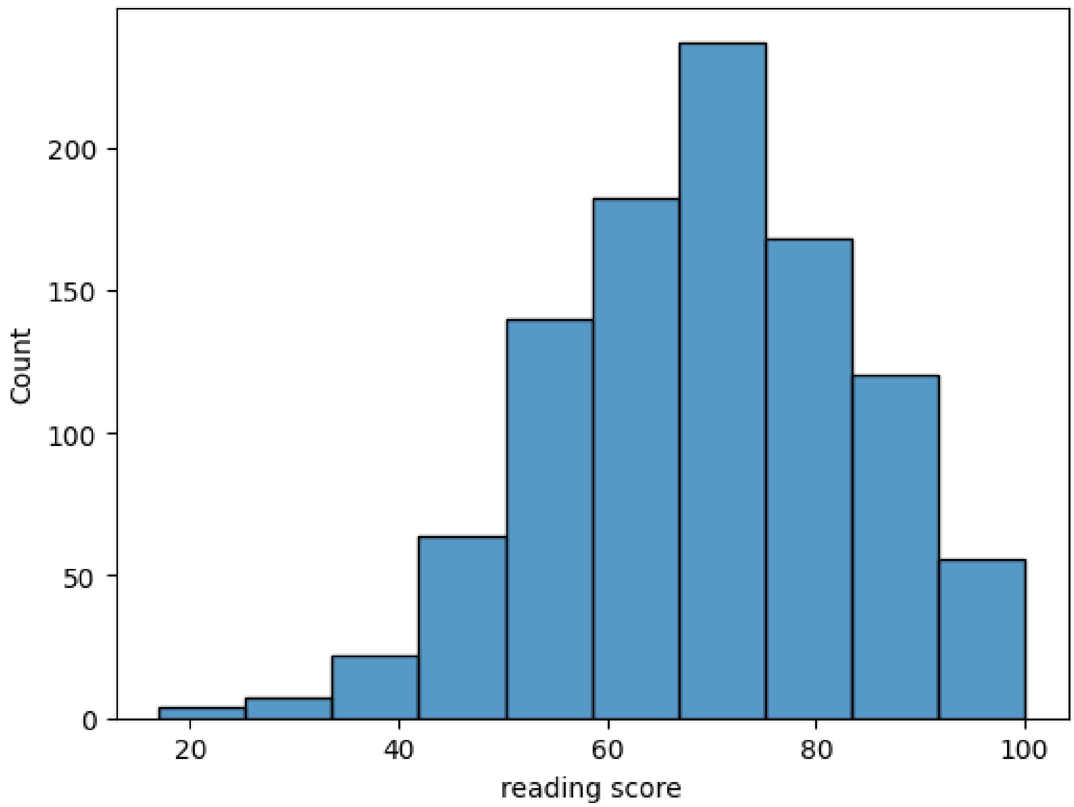
```
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (from
pandas>=1.2->seaborn) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (from
pandas>=1.2->seaborn) (2025.1)
Requirement already satisfied: six>=1.5 in
/home/motid/kfupm/assignment2-ICS/.venv/lib/python3.11/site-packages (from
python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.17.0)
Downloading seaborn-0.13.2-py3-none-any.whl (294 kB)
                         294.9/294.9 kB
2.1 MB/s eta 0:00:00a 0:00:01
Downloading
matplotlib-3.10.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(8.6 MB)
                         8.6/8.6 MB
10.4 MB/s eta 0:00:0000:0100:01
Downloading
contourpy-1.3.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (326
kB)
                         326.2/326.2 kB
9.5 MB/s eta 0:00:00ta 0:00:01
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading
fonttools-4.56.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (4.9
MB)
                         4.9/4.9 MB
13.1 MB/s eta 0:00:0000:0100:01
Downloading
kiwisolver-1.4.8-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.4
MB)
                         1.4/1.4 MB
12.2 MB/s eta 0:00:0000:0100:01
Downloading pillow-11.1.0-cp311-cp311-manylinux_2_28_x86_64.whl (4.5 MB)
                         4.5/4.5 MB
13.7 MB/s eta 0:00:0000:0100:01
Downloading pyparsing-3.2.1-py3-none-any.whl (107 kB)
                         107.7/107.7 kB
5.4 MB/s eta 0:00:00
Installing collected packages: pyparsing, pillow, kiwisolver, fonttools,
cycler, contourpy, matplotlib, seaborn
Successfully installed contourpy-1.3.1 cycler-0.12.1 fonttools-4.56.0
kiwisolver-1.4.8 matplotlib-3.10.1 pillow-11.1.0 pyparsing-3.2.1 seaborn-0.13.2

[notice] A new release of pip is
available: 24.0 -> 25.0.1
```

```
[notice] To update, run:
pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

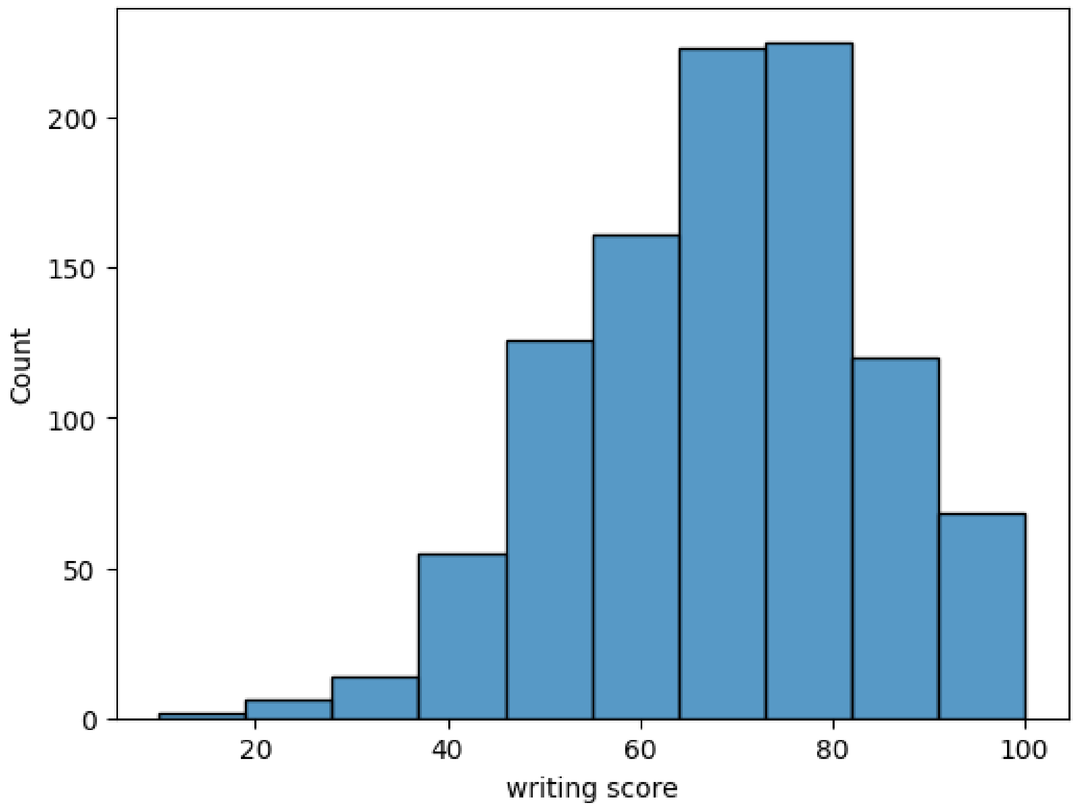
[38]:
```python
%matplotlib inline
# 3. (a) Draw histograms of both numeric and non-numeric columns.

# Load all the libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Identify numeric columns
num_columns = df.select_dtypes(exclude='object').columns

for c in num_columns:
    plt.figure()
    sns.histplot(x=c,bins=10,data=df);
    plt.show()

# sns.histplot(x=df["column_name"], bins=15, kde=True, color='red',␣
 ↪stat='density')
# plt.show()
## To draw a single column
# plt.figure()
# sns.histplot(x='math score',bins=10,data=df);
# plt.show()
```

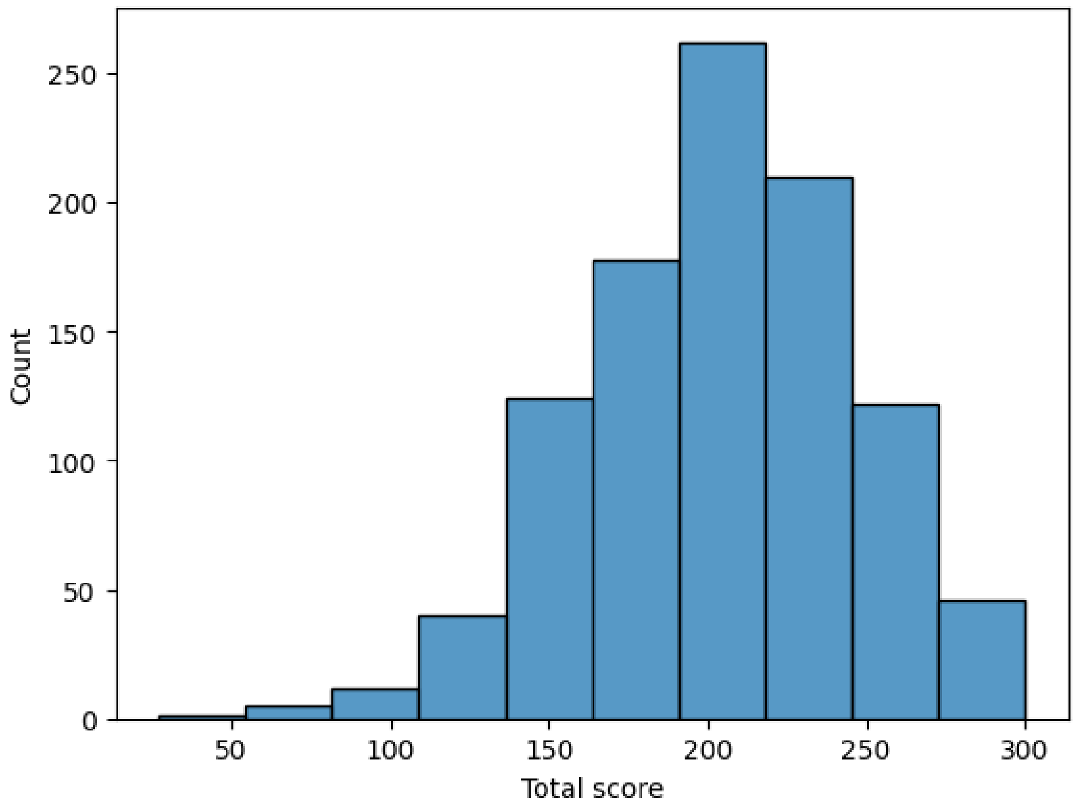


Count
math score
0
50
100
150
200
250
0
20
40
60
80
100



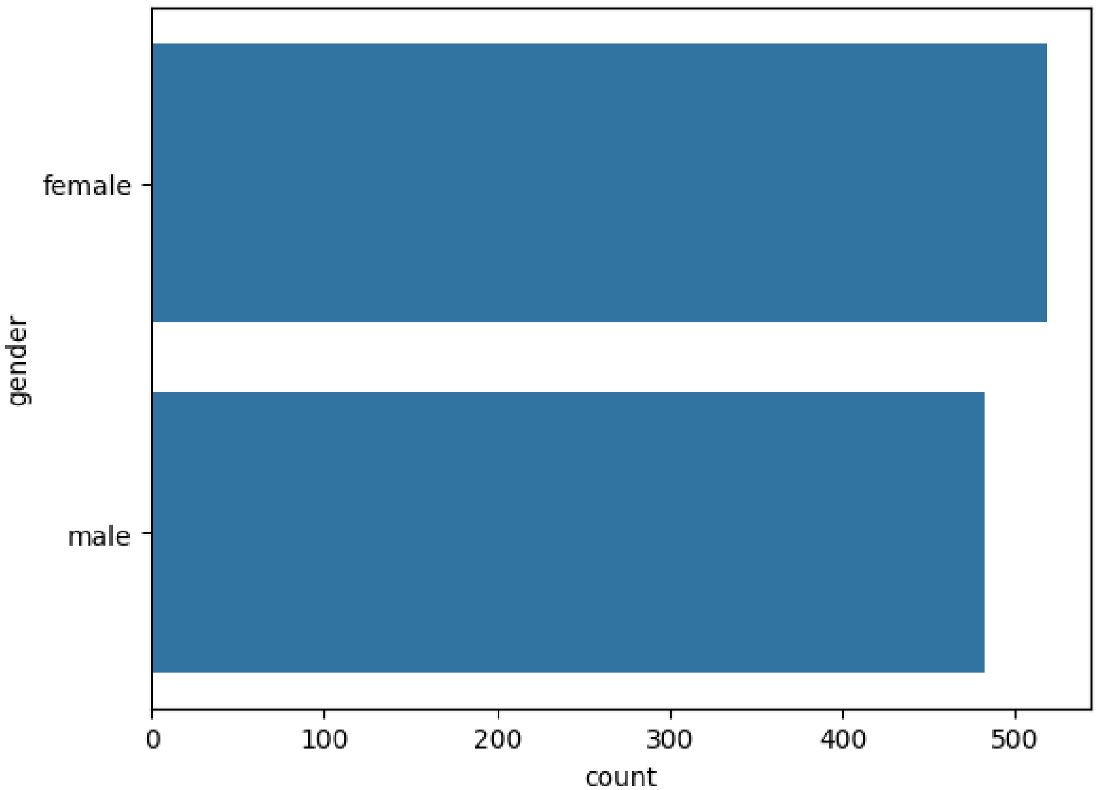
200
150
100
50
0
Count
20
40
60
80
100
reading score

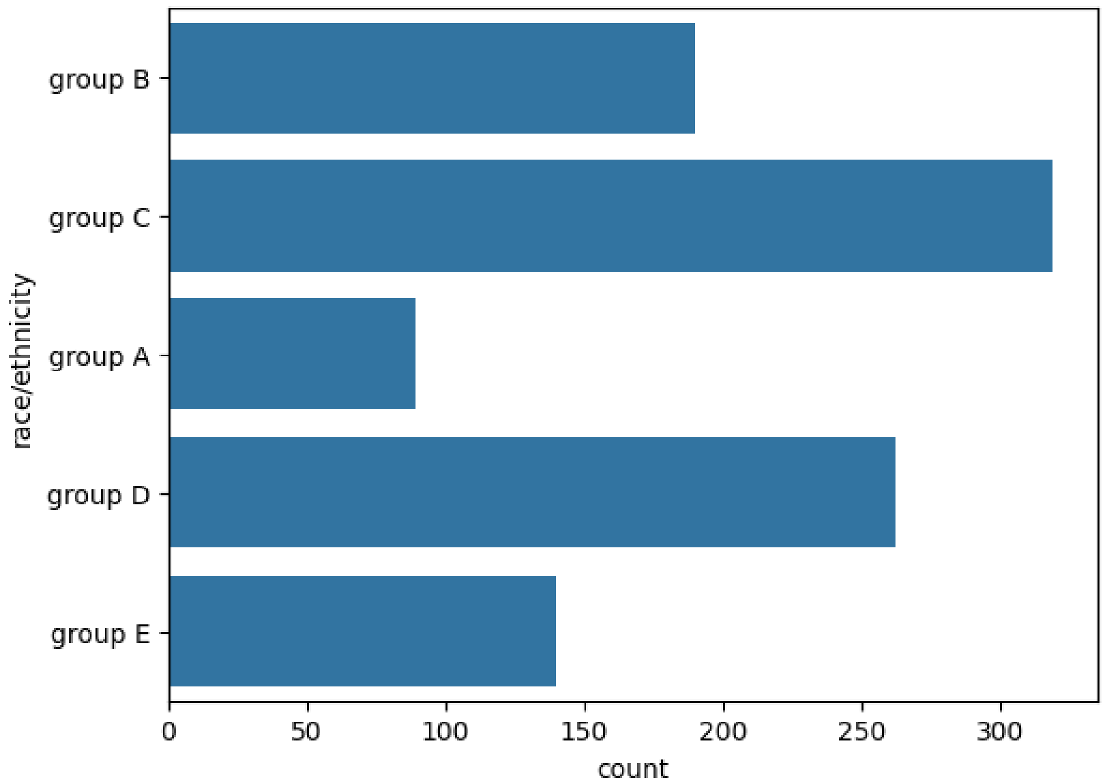

Count
writing score
0
50
100
150
200
20
40
60
80
100

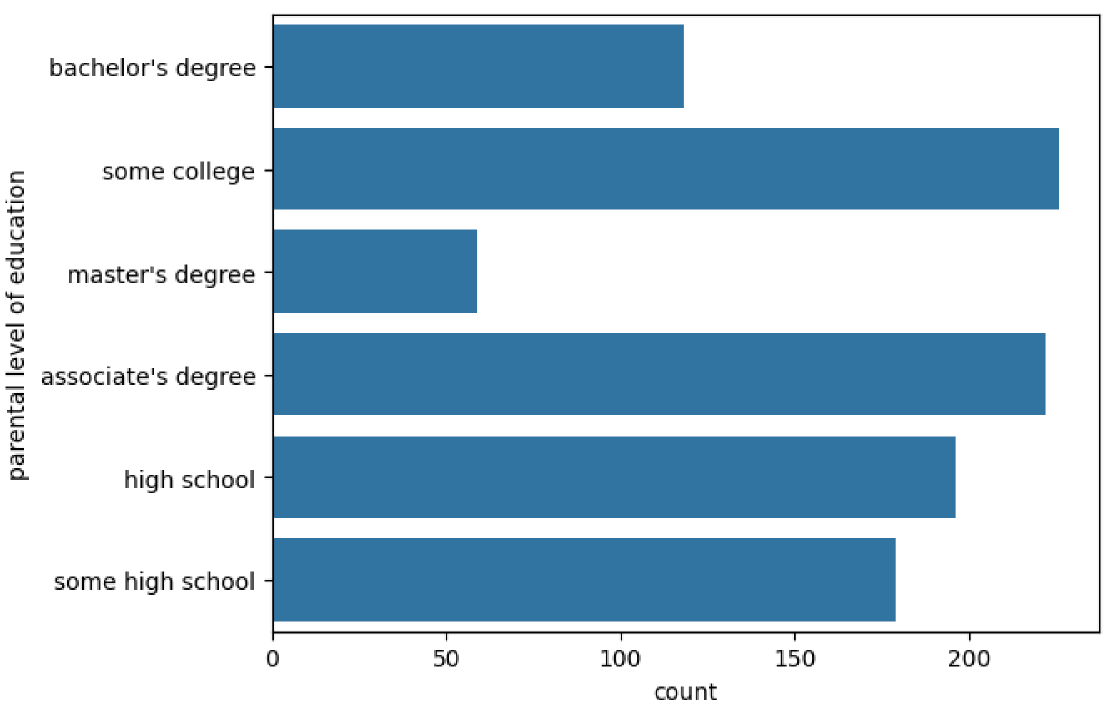


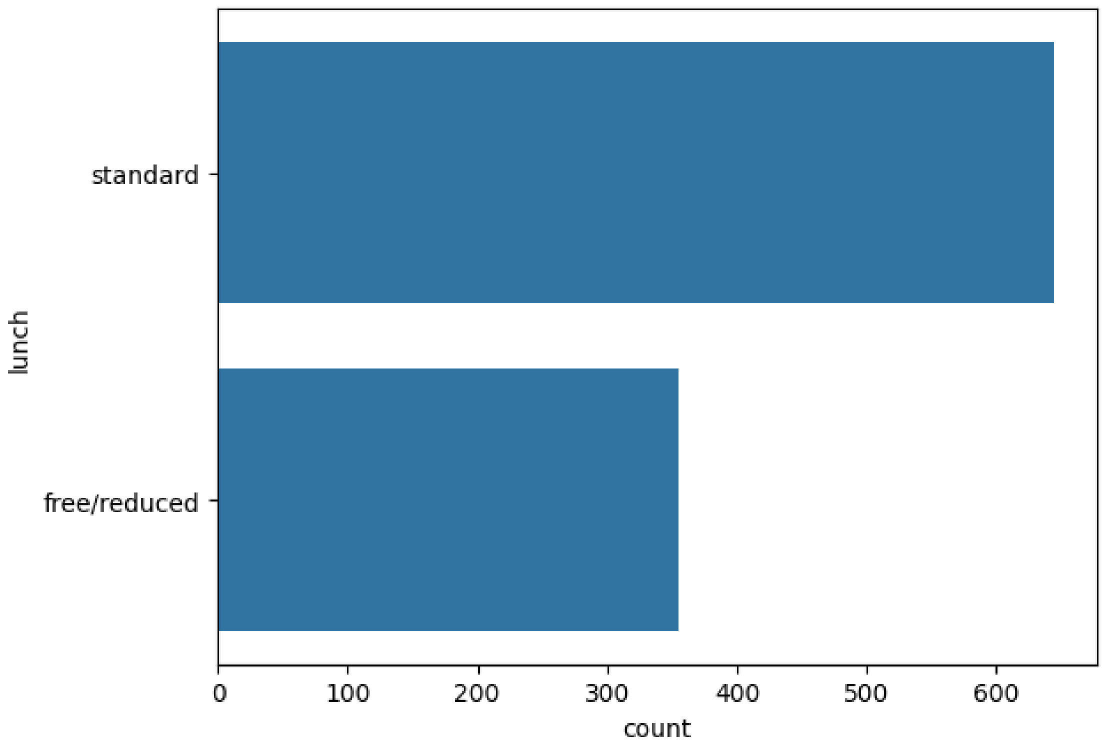
[39]:
```
# pip install seaborn
```

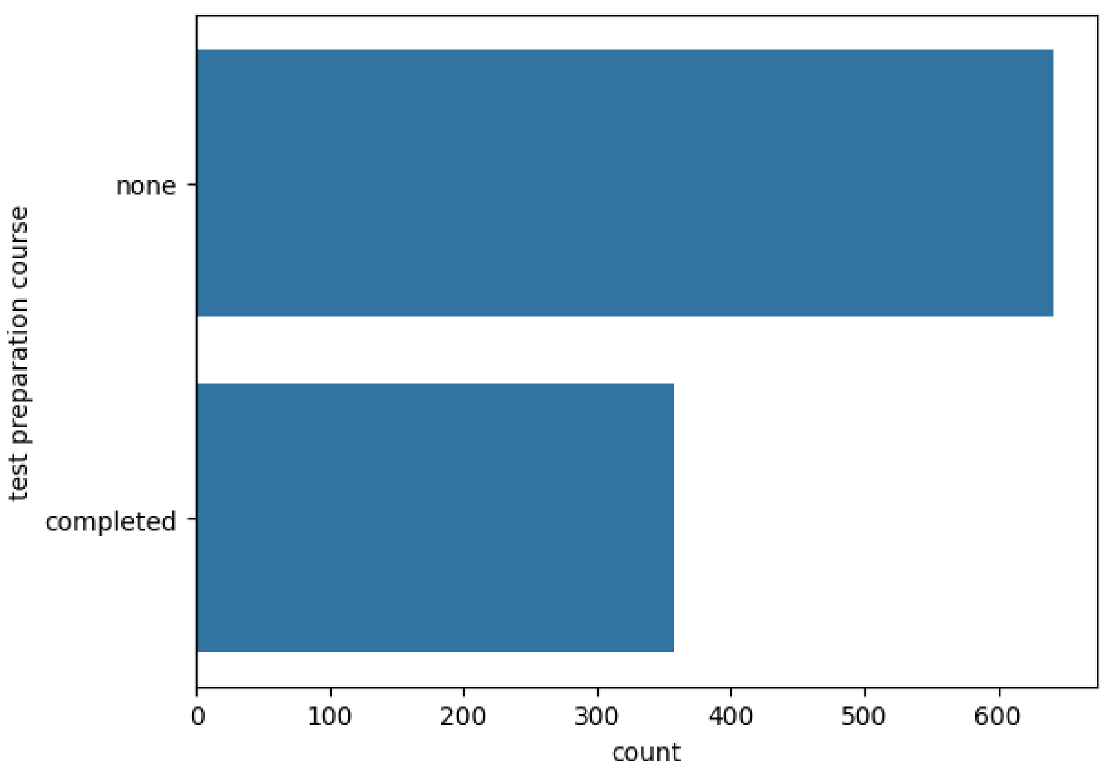
[40]:
```
# (b) For non-numeric columns, count plot option may be used.

# Identify non-numeric columns
obj_columns = df.select_dtypes('object').columns

# Plot for each identified column
for c in obj_columns:
    plt.figure()
#     sns.histplot(y=c,data=df);
    sns.countplot(y=c,data=df);
    plt.show()

## To draw a single column
# plt.figure()
# sns.countplot(y='gender',data=df);
# plt.show()
```


female
male
gender
0
100
200
300
400
500
count


group B
group C
group A
group D
group E
race/ethnicity
0
50
100
150
200
250
300
count

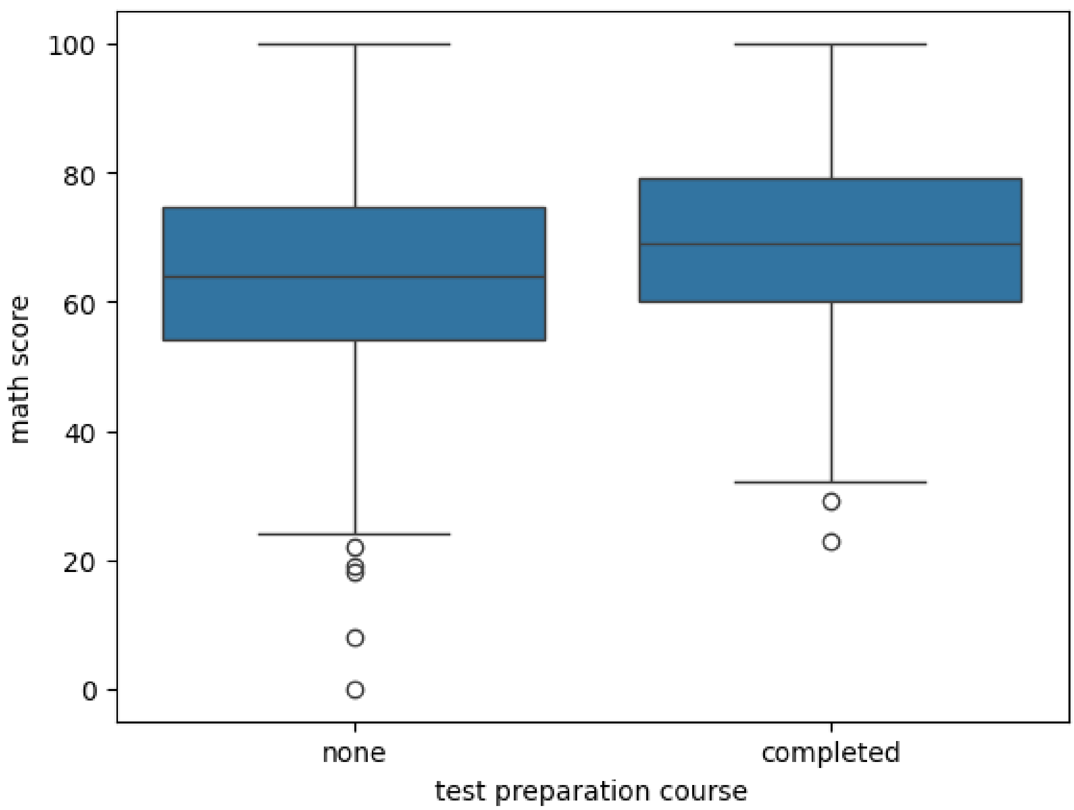

bachelor's degree
some college
master's degree
associate's degree
high school
some high school
parental level of education
0
50
100
150
200
count

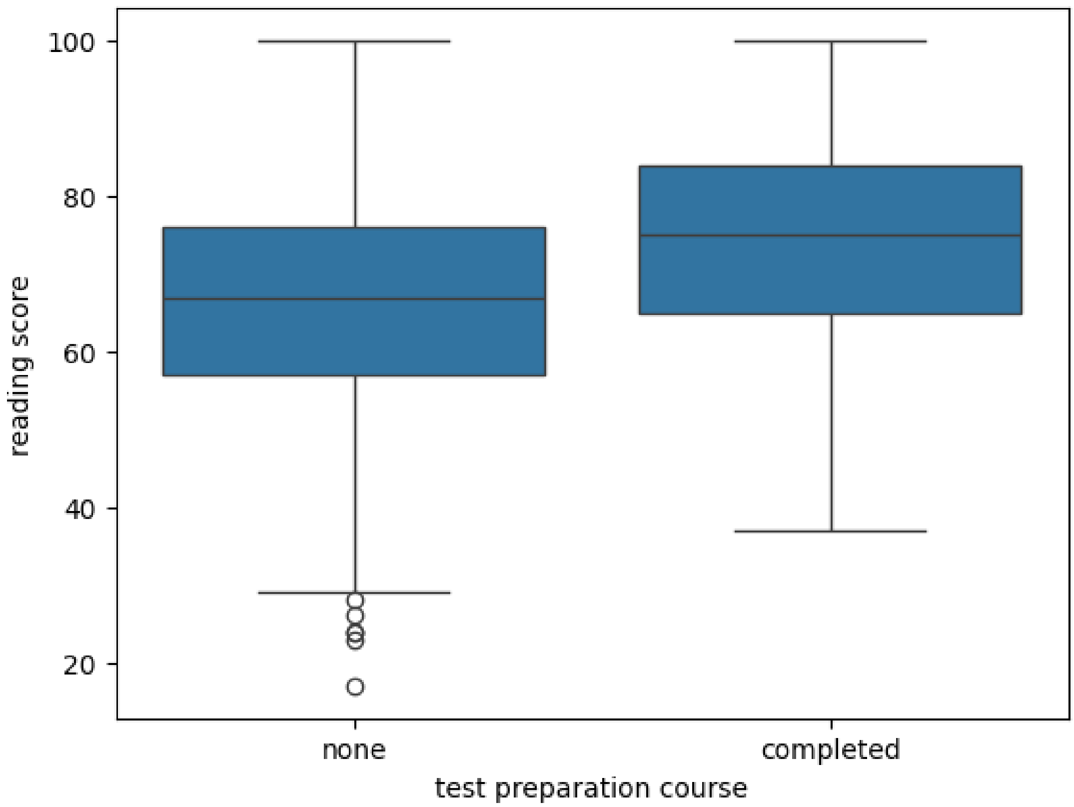

standard
free/reduced
lunch
0
100
200
300
400
500
600
count


none
completed
test preparation course
0
100
200
300
400
500
600
count

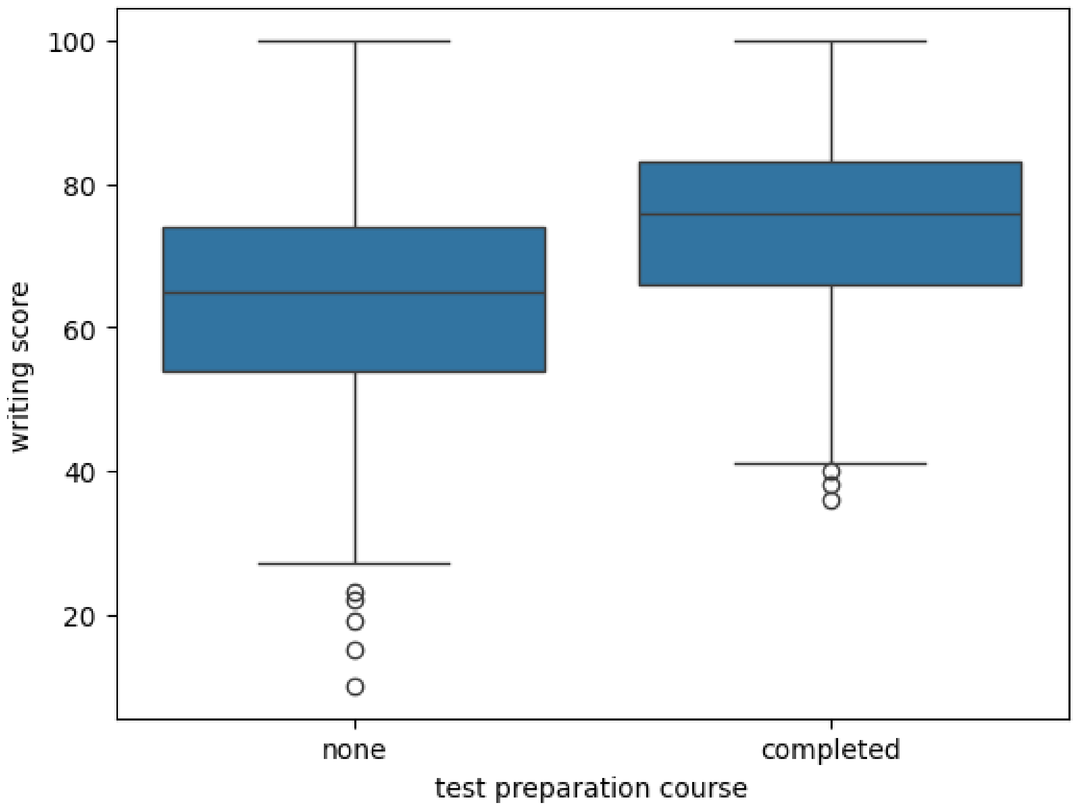
[41]:

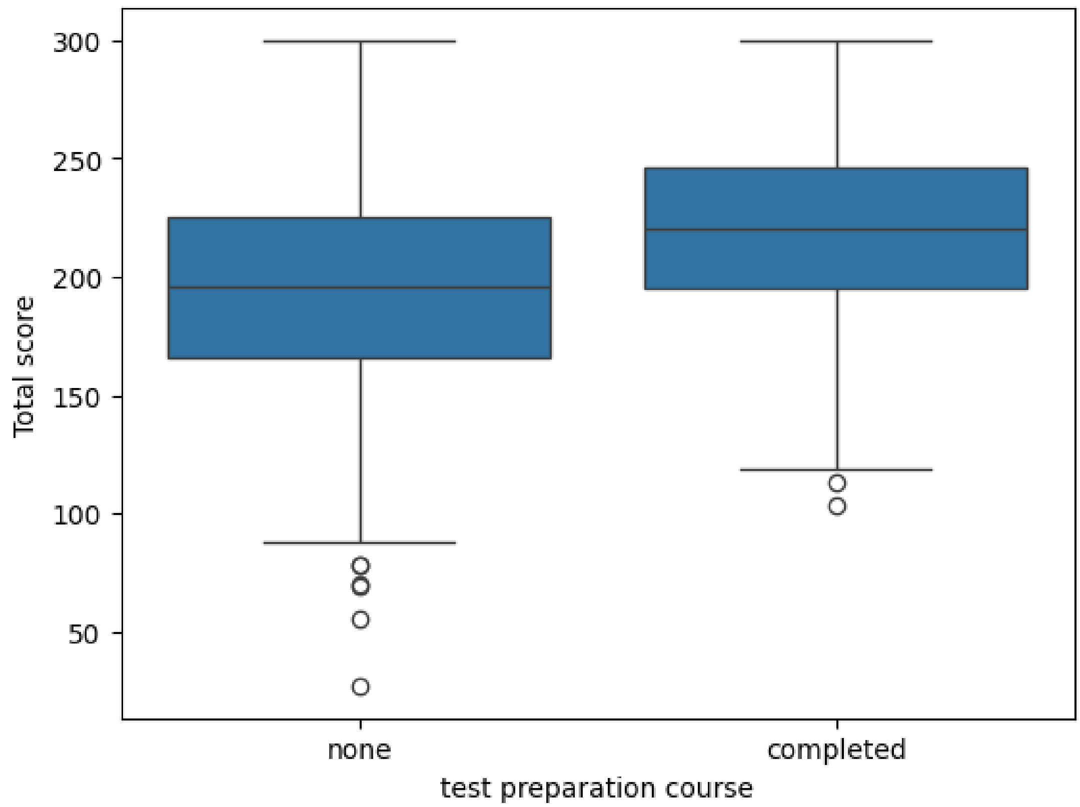
```
# 4. Draw box-plots for the numeric columns, and differentiate by *test␣
↪preparation course* column.

import matplotlib.pyplot as plt
import seaborn as sns

# Identify numeric columns
num_columns = df.select_dtypes(exclude='object').columns

for c in num_columns:
    plt.figure()
    sns.boxplot(y=c,x='test preparation course',data=df);
    plt.show()


# plt.figure()
# sns.boxplot(y='math score',x='test preparation course',data=df);
# plt.show()
```



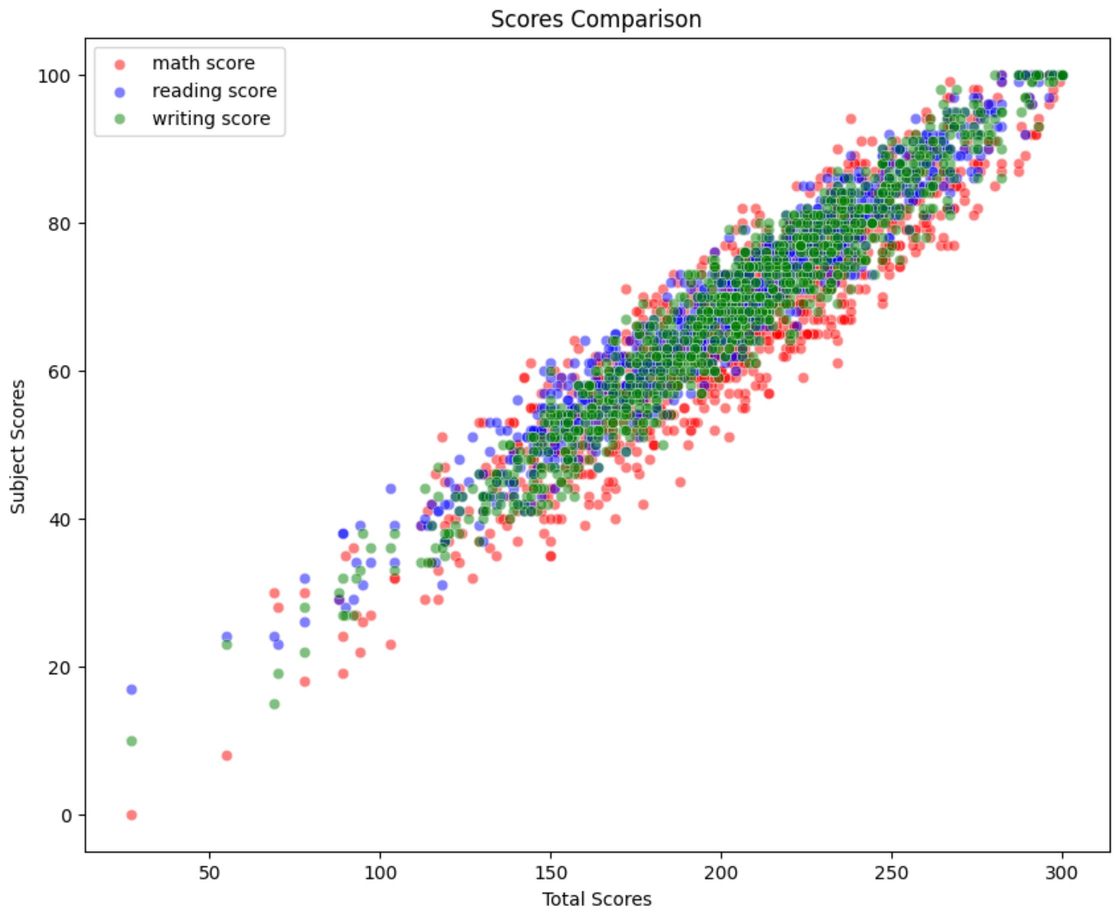


100
80
60
40
20
reading score
none
completed
test preparation course

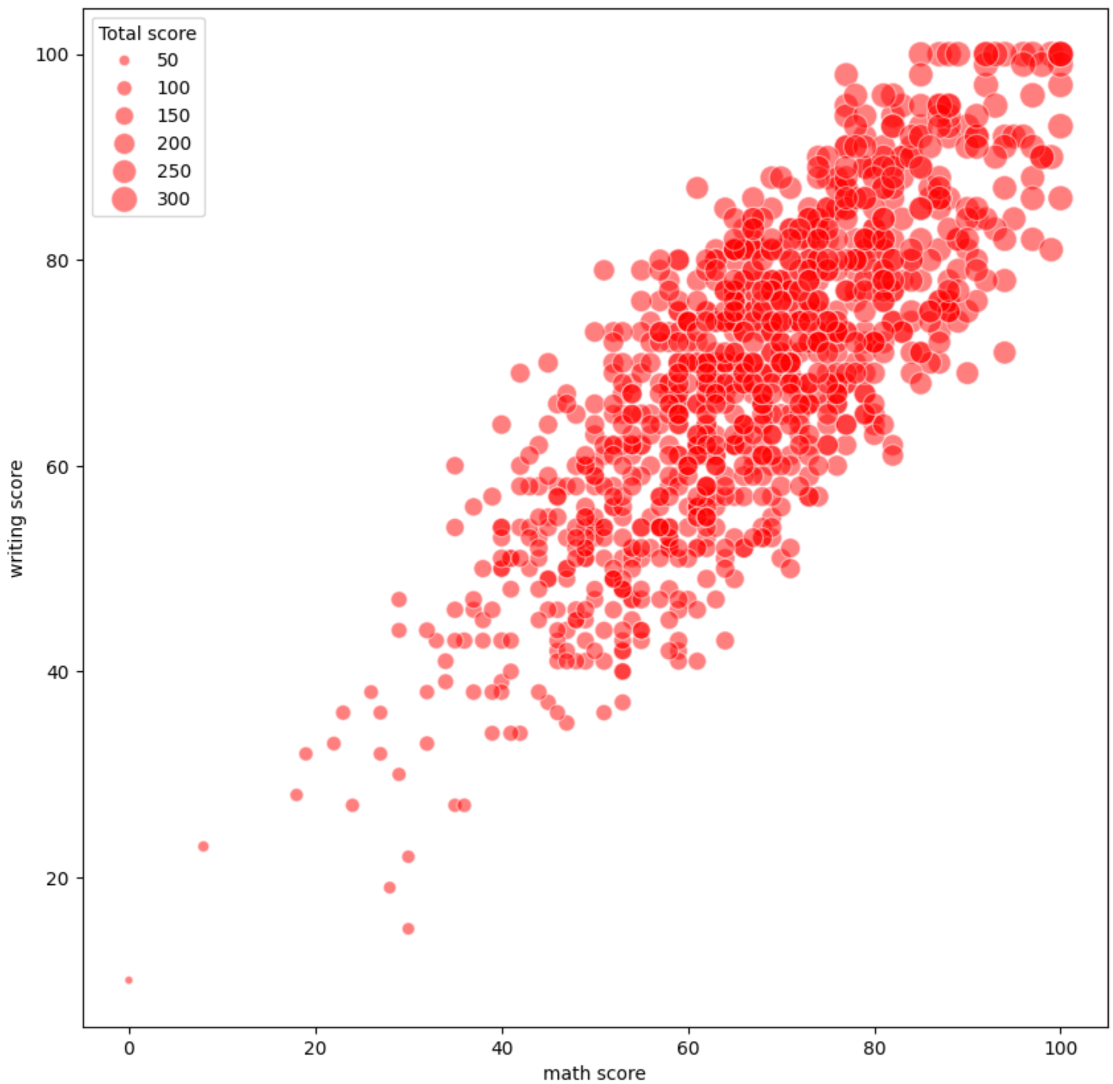

100
80
60
40
20
writing score
none
completed
test preparation course




```
[42]: # 5. Draw three overlapping scatter plots of columns *math score, reading␣
      ↪score* and *writing score*
      #w.r.t *Total score*. Set the markers transparency level (alpha) to 0.5.


      plt.figure(figsize=(10,8))
      sns.scatterplot(x='Total score', y='math score', color='red', label='math␣
      ↪score',  alpha=0.5, data=df)
      sns.scatterplot(x='Total score', y='reading score',color='blue',label='reading␣
      ↪score', alpha=0.5, data=df)
      sns.scatterplot(x='Total score', y='writing score',color='green',label='writing␣
      ↪score', alpha=0.5, data=df)
      plt.ylabel("Subject Scores");
      plt.xlabel("Total Scores");
      plt.title("Scores Comparison");
      plt.show()
```



[43]:
```
# 6. Draw scatter plot of columns *math score* and *writing score*,
#where the size of the marker is based on column *Total score*.
#In addition to that, reduce the marker transparency (alpha) to 0.5.

plt.figure(figsize=(10,10))
sns.scatterplot(x='math score', y='writing score',
                size='Total score',sizes=(20,200),
                alpha=0.5, color='red',
                data=df)
plt.show()
```



2.4.2 Example: Seaborn's Advanced Plots

Question-C: Consider the data in file Basic-4-Clean.csv.

1. Read and display information the data. Remove all rows containing NA values.
2. Depict *Category* column by count. In another plot, depict *Category* counts (differentiated) by *Type*.
3. Draw a boxplot plot of *Installs* vs *Rating* columns, ordered by *Installs*.
4. Draw a violinplot plot of *Content Rating* vs *Rating* columns, differentiated w.r.t *Type* .
5. Plot a barplot depicting count of *Category* that have *Installs* above 0.5B.
6. Depicting the *Size* of *Category* that have *Installs* above 0.5B.
7. For all the paid apps, depict *Genres* by count diferentiated by *Content Rating*. In another plot depict *Genres* by *Rating*.

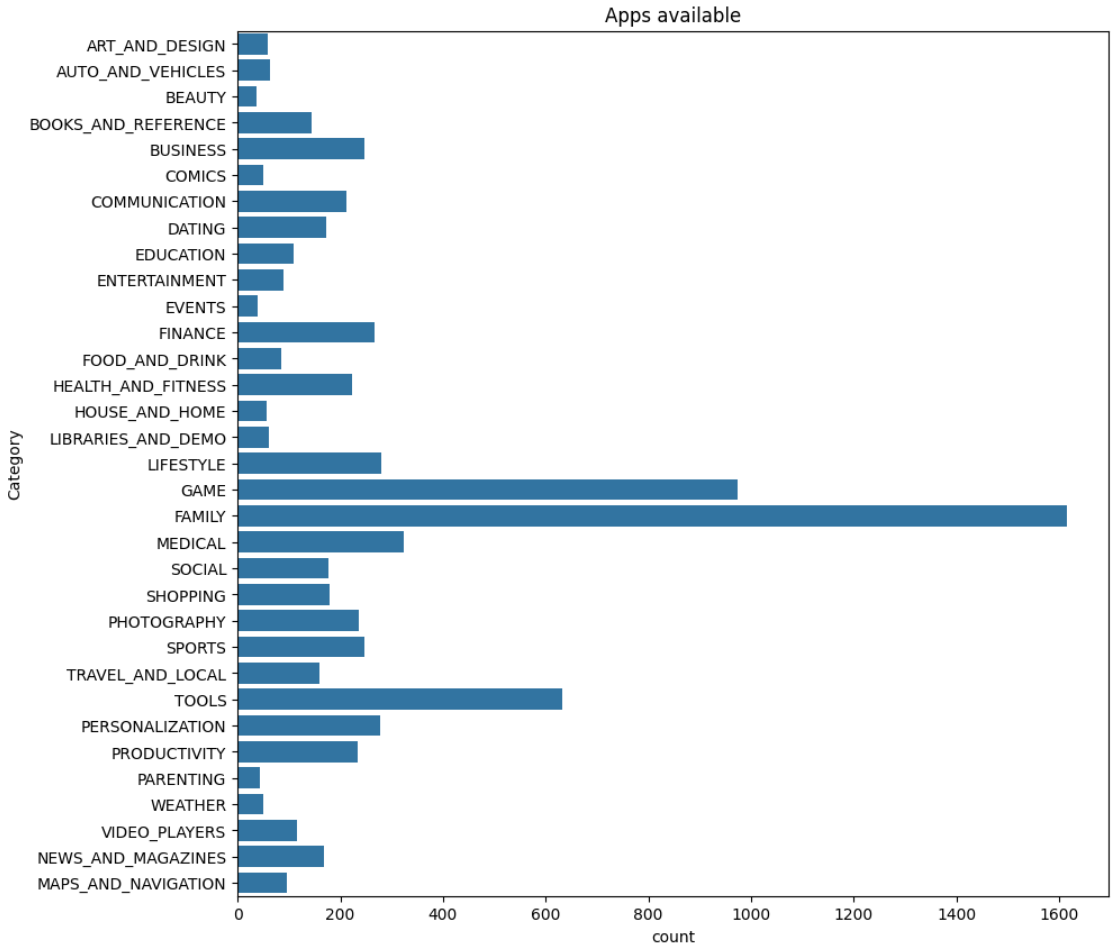
[44]:
```python
# 1. Read and display information the data. Remove all rows containing NA␣
↪values.

import pandas as pd
df = pd.read_csv('data/Basic-4-Clean.csv')
print(df.info())
#print(df.shape)# Can also be obtained from info
#print(df.count())# Can also be obtained from info
display(df.sample(5))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7723 entries, 0 to 7722
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   App             7723 non-null   object
 1   Category        7723 non-null   object
 2   Rating          7723 non-null   float64
 3   Reviews         7723 non-null   int64
 4   Size            7723 non-null   float64
 5   Installs        7723 non-null   object
 6   Type            7723 non-null   object
 7   Price           7723 non-null   float64
 8   Content Rating  7723 non-null   object
 9   Genres          7723 non-null   object
 10  Last Updated    7723 non-null   object
 11  Current Ver     7723 non-null   object
 12  Android Ver     7723 non-null   object
dtypes: float64(3), int64(1), object(9)
memory usage: 784.5+ KB
None

                                  App  Category  Rating  Reviews     Size  \
322                    OkCupid Dating    DATING     4.1   285726  15000.0
1446         Learn To Draw Glow Flower    FAMILY     4.4     7320  10000.0
5229                      CJ WOW SHOP  SHOPPING     4.2     2099      4.0
7409                        FG Mobile    FAMILY     3.3      130  13000.0
6077  Download Manager - File & Video     TOOLS     3.9     8780      5.0

       Installs  Type  Price Content Rating                 Genres  \
322   10000000+  Free    0.0     Mature 17+                 Dating
1446   1000000+  Free    0.0       Everyone  Entertainment;Education
5229    100000+  Free    0.0       Everyone               Shopping
7409      5000+  Free    0.0       Everyone              Education
6077   1000000+  Free    0.0       Everyone                  Tools

     Last Updated Current Ver   Android Ver
322     30-Jul-18     11.10.1    4.1 and up
```

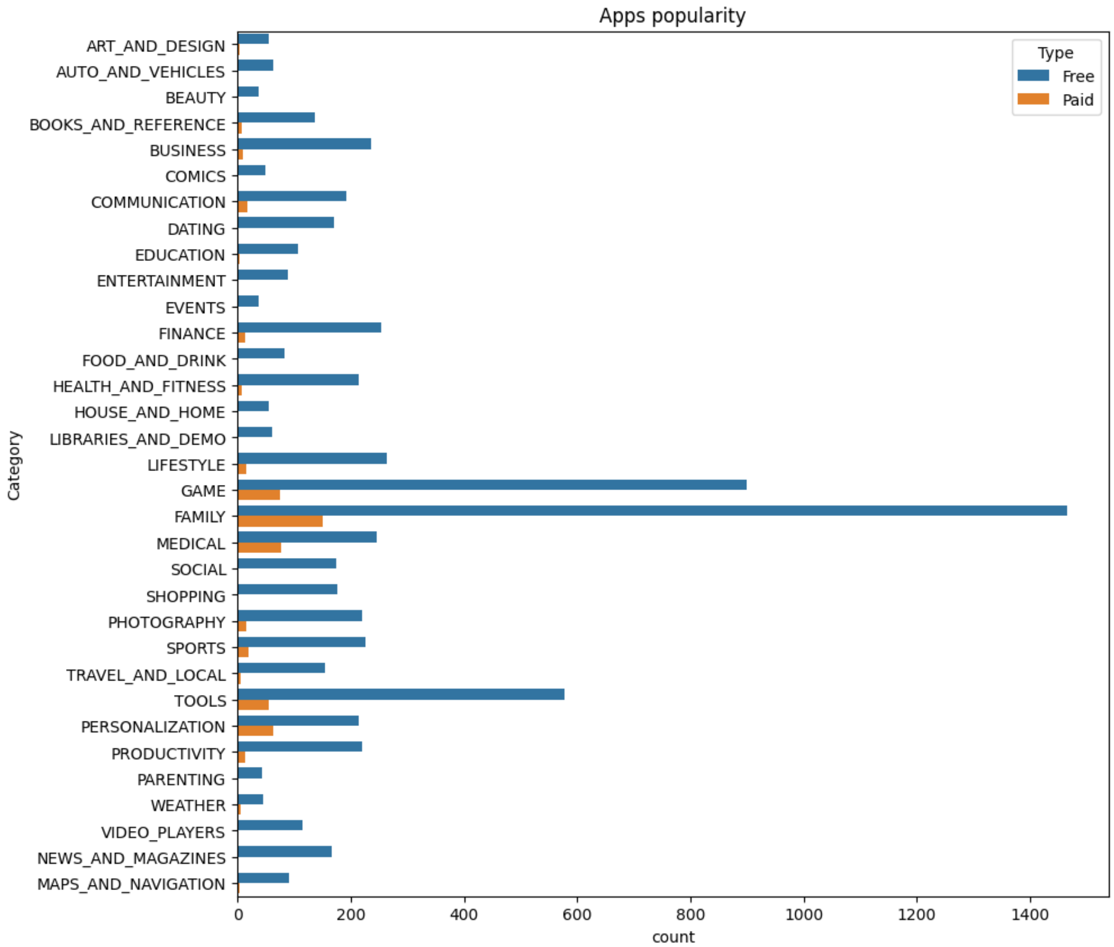
```
1446     7-Jul-17       1.0.1  4.0.3 and up
5229    23-Dec-17       1.1.1  4.0.3 and up
7409    21-Dec-17       2.0.0    4.0 and up
6077    13-Jun-18       2.7.5    4.2 and up
```

[45]:

```python
# 2. Depict *Category* column by count. In another plot, depict *Category*␣
↪counts (differentiated) by *Type*.

#plot category counts
import seaborn as sns
plt.figure(figsize=(10,10))
ax = sns.countplot(y="Category", data=df)
plt.title('Apps available')
plt.show()

#plot category with type
import seaborn as sns
plt.figure(figsize=(10,10))
ax = sns.countplot(y='Category',hue='Type',data=df)
plt.title('Apps popularity')
plt.show()
```

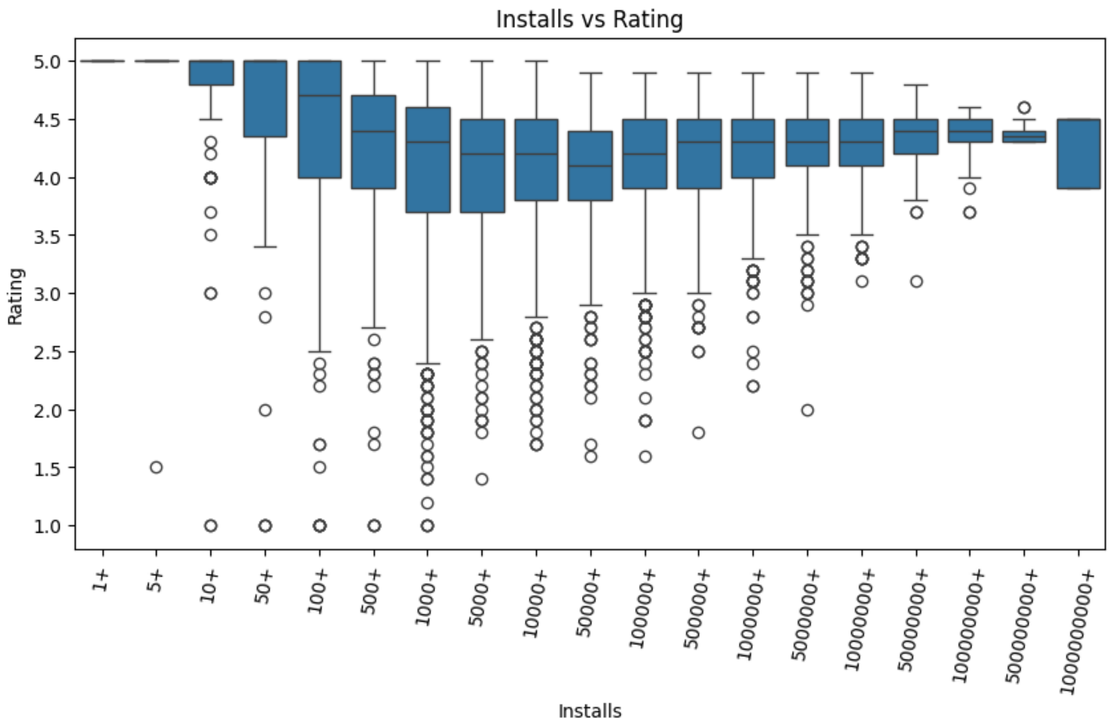


Apps available
ART_AND_DESIGN
AUTO_AND_VEHICLES
BEAUTY
BOOKS_AND_REFERENCE
BUSINESS
COMICS
COMMUNICATION
DATING
EDUCATION
ENTERTAINMENT
EVENTS
FINANCE
FOOD_AND_DRINK
HEALTH_AND_FITNESS
HOUSE_AND_HOME
LIBRARIES_AND_DEMO
LIFESTYLE
GAME
FAMILY
MEDICAL
SOCIAL
SHOPPING
PHOTOGRAPHY
SPORTS
TRAVEL_AND_LOCAL
TOOLS
PERSONALIZATION
PRODUCTIVITY
PARENTING
WEATHER
VIDEO_PLAYERS
NEWS_AND_MAGAZINES
MAPS_AND_NAVIGATION
Category
0
200
400
600
800
1000
1200
1400
1600
count



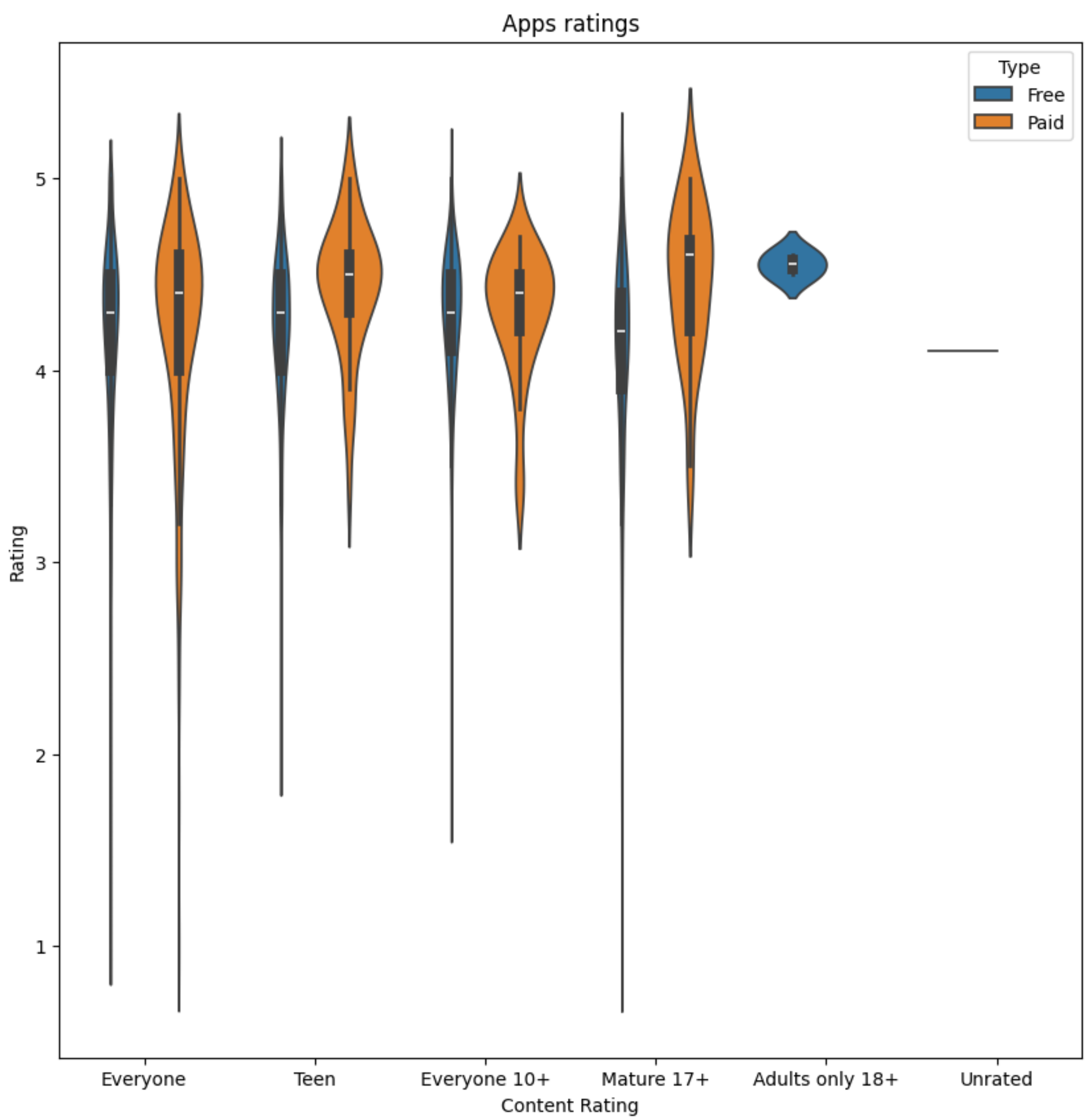
[46]:

```
#3. Draw a boxplot plot of *Installs* vs *Rating* columns, ordered by␣
↪*Installs*.

#adding a new column
df['Installs_num']=df['Installs'].apply(lambda x:x.replace('+','')).apply(pd.
↪to_numeric)

#the box plot
ax = plt.figure(figsize=(10,5))
sns.boxplot(x="Installs", y="Rating", data=df.sort_values(by=['Installs_num']))
#,palette="Set2"
plt.title("Installs vs Rating")
plt.xticks(rotation=80)
plt.show()
```



[47]:

```
#4. Draw a violinplot plot of *Content Rating* vs *Rating* columns,␣
↪differentiated w.r.t *Type* .

import seaborn as sns
plt.figure(figsize=(10,10))
sns.violinplot(x ='Content Rating', y ='Rating', hue='Type', data = df)
plt.title('Apps ratings')
plt.show()
#violinplot plot also depitcs the distribution.


# # contrast with box plots
# plt.figure(figsize=(10,10))
# sns.boxplot(x ='Content Rating', y ='Rating', hue='Type', data = df)
# plt.title('Apps ratings')
# plt.show()
```

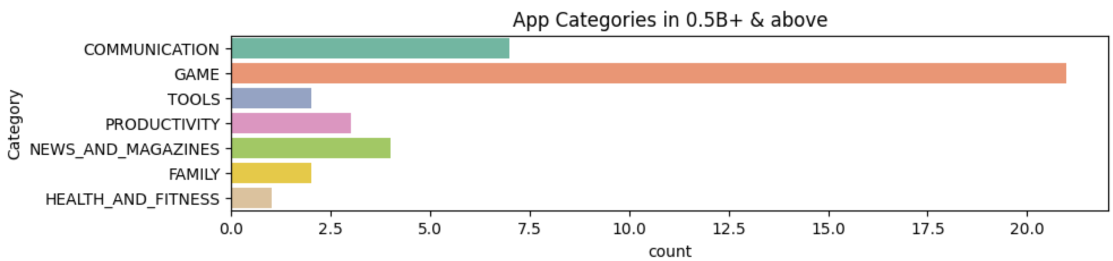


[48]:
```
#5. Plot a barplot depicting count of *Category* that have *Installs* above 0.
 ↪5B.
import seaborn as sns
seleted_rows = (df['Installs']=='1000000000+') | (df['Installs']=='500000000+')

plt.figure(figsize=(10,2))
sns.countplot(y ='Category', data = df.loc[seleted_rows,:],palette="Set2")
plt.title('App Categories in 0.5B+ & above')
plt.show()
```

```
/tmp/ipykernel_21727/394882582.py:6: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(y ='Category', data = df.loc[seleted_rows,:],palette="Set2")

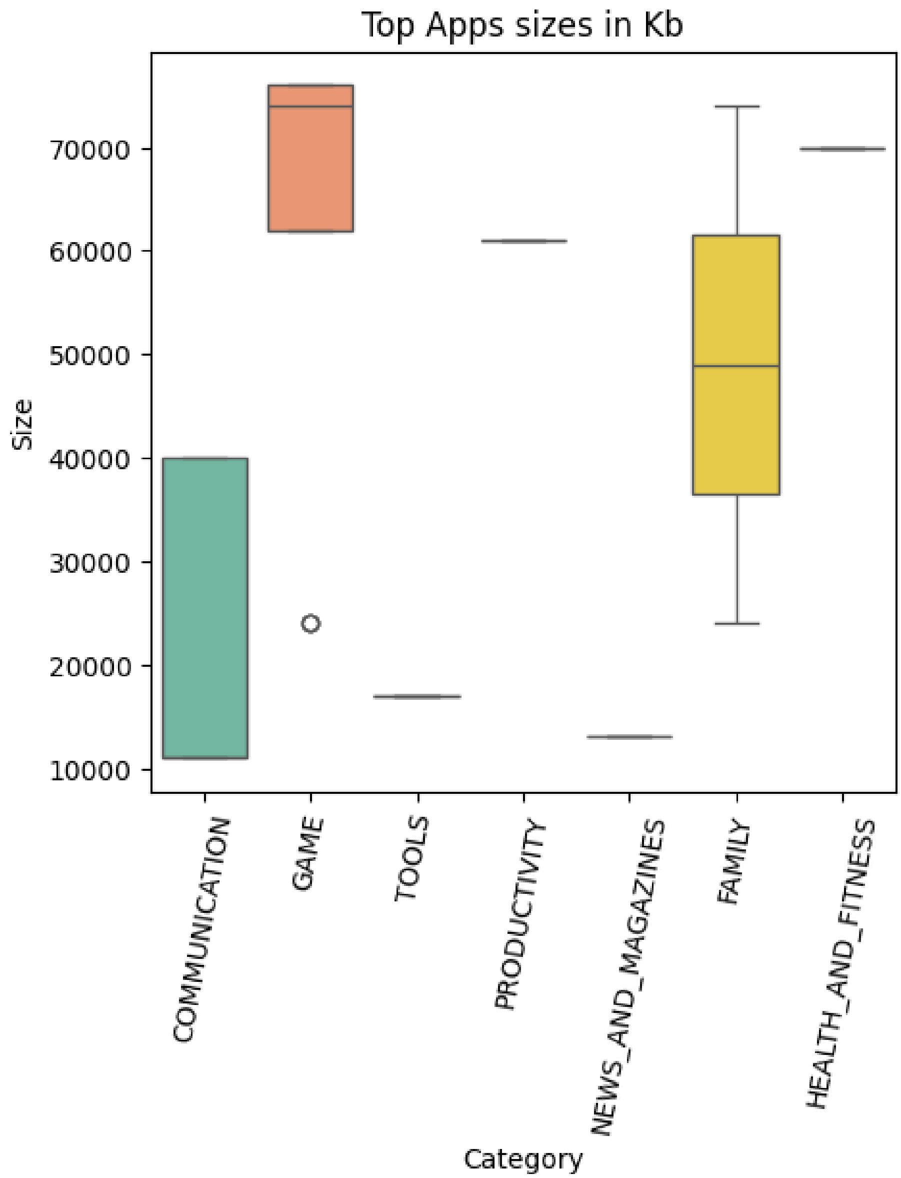


[49]:
```
#6. Depicting the *Size* of *Category* that have *Installs* above 0.5B.

import seaborn as sns

plt.figure(figsize=(5,5))
# ax=sns.swarmplot(x ='Category', y ='Size', data = df.loc[seleted_rows,:])
# ax=sns.violinplot(x ='Category', y ='Size', data = df.loc[seleted_rows,:])
ax=sns.boxplot(x ='Category', y ='Size', data = df.loc[seleted_rows,:
 ↪],palette="Set2")
plt.xticks(rotation=80)
plt.title('Top Apps sizes in Kb')
plt.show()
```

/tmp/ipykernel_21727/3301675080.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  ax=sns.boxplot(x ='Category', y ='Size', data =
df.loc[seleted_rows,:],palette="Set2")



3 References:

Main ref. ISE 291

3.1 Data Sets:

1. Basic-2: Labor Relations Data, UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science. Published in: Bergadano, F., Matwin, S., Michalski, R., Zhang, J., Measuring Quality of Concept Descriptions, Procs. of the 3rd European Working Sessions on Learning, Glasgow, October 1988.

2. Basic-3: https://www.kaggle.com/spscientist/students-performance-in-exams
3. Basic-4: *modified*, https://www.kaggle.com/lava18/google-play-store-apps

3.2 Others:

1. Series: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html
2. DataFrames: https://pandas.pydata.org/pandas-docs/stable/reference/frame.html
3. Read CSV: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html
4. Read XLSX: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_excel.html
5. Visualization: https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html
6. Seaborn: https://seaborn.pydata.org/