Data analysis assignment in placement dataset of students

1) Replace the Nan values with the correct value and justify why you have chosen the same.

In [1]:
```
# 1.Replace the NaN values with the correct value and justify why you chosen the same.
# import pandas library
import pandas as pd
import numpy as np
```

In [2]:
```
# assign the file to the variable
dataset=pd.read_csv("Placement.csv")
dataset
```

Out[2]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	st
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	Mkt&HR	58.80	Pl
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	Mkt&Fin	66.28	Pl
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	Mkt&Fin	57.80	Pl
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0	Mkt&HR	59.43	Pl
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	96.8	Mkt&Fin	55.50	Pl
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
210	211	M	80.60	Others	82.00	Others	Commerce	77.60	Comm&Mgmt	No	91.0	Mkt&Fin	74.49	Pl
211	212	M	58.00	Others	60.00	Others	Science	72.00	Sci&Tech	No	74.0	Mkt&Fin	53.62	Pl
212	213	M	67.00	Others	67.00	Others	Commerce	73.00	Comm&Mgmt	Yes	59.0	Mkt&Fin	69.72	Pl
213	214	F	74.00	Others	66.00	Others	Commerce	58.00	Comm&Mgmt	No	70.0	Mkt&HR	60.23	Pl
214	215	M	62.00	Central	58.00	Others	Science	53.00	Comm&Mgmt	No	89.0	Mkt&HR	60.22	Pl

215 rows × 15 columns

In [3]:
```
#we check the NaN values present
dataset.isna().sum()
```

Out[3]:
```
sl_no              0
gender             0
ssc_p              0
ssc_b              0
hsc_p              0
hsc_b              0
hsc_s              0
degree_p           0
degree_t           0
workex             0
etest_p            0
specialisation     0
mba_p              0
status             0
salary            67
dtype: int64
```

In [4]:
```
# function  we separte the numerical and categorical dataset.
def quanQual(dataset):
    quan=[]
    qual=[]
    for columnName in dataset.columns:
        if (dataset[columnName].dtypes=='O'):
            qual.append(columnName)
        else:
            quan.append(columnName)
    return quan,qual
```

In [5]:
```
quan,qual=quanQual(dataset)
```

In [6]:
```
dataset[quan]
```

Out[6]:

	sl_no	ssc_p	hsc_p	degree_p	etest_p	mba_p	salary
0	1	67.00	91.00	58.00	55.0	58.80	270000.0
1	2	79.33	78.33	77.48	86.5	66.28	200000.0
2	3	65.00	68.00	64.00	75.0	57.80	250000.0
3	4	56.00	52.00	52.00	66.0	59.43	NaN
4	5	85.80	73.60	73.30	96.8	55.50	425000.0
...	...	...	...	...	...	...	...
210	211	80.60	82.00	77.60	91.0	74.49	400000.0
211	212	58.00	60.00	72.00	74.0	53.62	275000.0
212	213	67.00	67.00	73.00	59.0	69.72	295000.0
213	214	74.00	66.00	58.00	70.0	60.23	204000.0
214	215	62.00	58.00	53.00	89.0	60.22	NaN

215 rows × 7 columns

In [7]:
```
descriptive=pd.DataFrame()
descriptive
```

Out[7]: —

In [8]:
```
descriptive=pd.DataFrame(index=["Mean","Median","Mode"],columns=quan)
for columnName in quan:
    descriptive[columnName]["Mean"]=dataset[columnName].mean()
    descriptive[columnName]["Median"]=dataset[columnName].median()
    descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
```

```
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:3: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:4: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:5: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:3: FutureWarning: ChainedAssignmentError:
```

```
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:4: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:5: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:3: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:4: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:5: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
```

```
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:3: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:4: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:5: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:3: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:4: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:
```

```
df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:5: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:3: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:4: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:5: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:3: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
```

```
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:4: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:5: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
```

In [9]:
```
descriptive
```

Out[9]:

	sl_no	ssc_p	hsc_p	degree_p	etest_p	mba_p	salary
Mean	108.0	67.303395	66.333163	66.370186	72.100558	62.278186	288655.405405
Median	108.0	67.0	65.0	66.0	71.0	62.0	265000.0
Mode	1	62.0	63.0	65.0	60.0	56.7	300000.0

In [10]:
```
descriptive=pd.DataFrame(index=["Mean","Median","Mode","Q1:25%","Q2:50%",
                        "Q3:75%","99%","Q4:100%","IQR","1.5rule","Lesser","Greater","Min","Max","kurtosis","ske
for columnName in quan:
        descriptive[columnName]["Mean"]=dataset[columnName].mean()
        descriptive[columnName]["Median"]=dataset[columnName].median()
        descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
        descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
        descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
        descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
        descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
        descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
        descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
        descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
        descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"
        descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
        descriptive[columnName]["Min"]=dataset[columnName].min()
        descriptive[columnName]["Max"]=dataset[columnName].max()
        descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
        descriptive[columnName]["skew"]=dataset[columnName].skew()
        descriptive[columnName]["Var"]=dataset[columnName].var()
        descriptive[columnName]["Std"]=dataset[columnName].std()
```

```
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:4: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
```

```
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:5: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:6: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:7: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:8: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:9: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
```

```
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:10: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:11: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:12: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:13: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:14: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:15: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
```

```
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:16: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Min"]=dataset[columnName].min()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:17: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Max"]=dataset[columnName].max()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:18: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:19: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["skew"]=dataset[columnName].skew()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:20: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:
```

```
df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Var"]=dataset[columnName].var()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:21: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Std"]=dataset[columnName].std()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:4: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:5: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:6: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:7: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value
```

```
Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:8: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:9: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:10: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:11: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:12: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:13: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:14: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:15: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:16: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Min"]=dataset[columnName].min()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:17: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
```

```
  descriptive[columnName]["Max"]=dataset[columnName].max()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:18: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:19: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["skew"]=dataset[columnName].skew()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:20: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Var"]=dataset[columnName].var()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:21: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Std"]=dataset[columnName].std()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:4: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:5: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
```

```
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:6: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:7: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:8: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:9: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:10: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
```

```
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:11: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:12: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:13: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:14: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:15: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value
```

```
Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:16: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Min"]=dataset[columnName].min()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:17: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Max"]=dataset[columnName].max()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:18: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:19: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["skew"]=dataset[columnName].skew()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:20: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Var"]=dataset[columnName].var()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:21: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Std"]=dataset[columnName].std()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:4: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:5: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:6: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:7: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
```

```
  descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:8: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:9: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:10: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:11: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:12: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:13: FutureWarning: ChainedAssignmentError:
```

```
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:14: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:15: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:16: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Min"]=dataset[columnName].min()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:17: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Max"]=dataset[columnName].max()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:18: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
```

```
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:19: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["skew"]=dataset[columnName].skew()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:20: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Var"]=dataset[columnName].var()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:21: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Std"]=dataset[columnName].std()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:4: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:5: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:
```

```
df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:6: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:7: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:8: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:9: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:10: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
```

```
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:11: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:12: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:13: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:14: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:15: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
```

```
rning-a-view-versus-a-copy

  descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:16: FutureWarning: ChainedAssignmentError: 
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Min"]=dataset[columnName].min()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:17: FutureWarning: ChainedAssignmentError: 
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Max"]=dataset[columnName].max()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:18: FutureWarning: ChainedAssignmentError: 
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:19: FutureWarning: ChainedAssignmentError: 
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["skew"]=dataset[columnName].skew()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:20: FutureWarning: ChainedAssignmentError: 
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Var"]=dataset[columnName].var()
```

```
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:21: FutureWarning: ChainedAssignmentError: 
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Std"]=dataset[columnName].std()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:4: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:5: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:6: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:7: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:8: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
```

```
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:9: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:10: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:11: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:12: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:13: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:
```

```
df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:14: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:15: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:16: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Min"]=dataset[columnName].min()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:17: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Max"]=dataset[columnName].max()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:18: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value
```

```
Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:19: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["skew"]=dataset[columnName].skew()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:20: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Var"]=dataset[columnName].var()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:21: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Std"]=dataset[columnName].std()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:4: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:5: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:6: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:7: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:8: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:9: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:10: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
```

```
  descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:11: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:12: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:13: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:14: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:15: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:16: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
```

```
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Min"]=dataset[columnName].min()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:17: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Max"]=dataset[columnName].max()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:18: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:19: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["skew"]=dataset[columnName].skew()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:20: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Var"]=dataset[columnName].var()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:21: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
```

```
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Std"]=dataset[columnName].std()
```

In [11]:
```
descriptive
```

Out[11]:

	sl_no	ssc_p	hsc_p	degree_p	etest_p	mba_p	salary
Mean	108.0	67.303395	66.333163	66.370186	72.100558	62.278186	288655.405405
Median	108.0	67.0	65.0	66.0	71.0	62.0	265000.0
Mode	1	62.0	63.0	65.0	60.0	56.7	300000.0
Q1:25%	54.5	60.6	60.9	61.0	60.0	57.945	240000.0
Q2:50%	108.0	67.0	65.0	66.0	71.0	62.0	265000.0
Q3:75%	161.5	75.7	73.0	72.0	83.5	66.255	300000.0
99%	212.86	87.0	91.86	83.86	97.0	76.1142	NaN
Q4:100%	215.0	89.4	97.7	91.0	98.0	77.89	940000.0
IQR	107.0	15.1	12.1	11.0	23.5	8.31	60000.0
1.5rule	160.5	22.65	18.15	16.5	35.25	12.465	90000.0
Lesser	-106.0	37.95	42.75	44.5	24.75	45.48	150000.0
Greater	322.0	98.35	91.15	88.5	118.75	78.72	390000.0
Min	1	40.89	37.0	50.0	50.0	51.21	200000.0
Max	215	89.4	97.7	91.0	98.0	77.89	940000.0
kurtosis	-1.2	-0.60751	0.450765	0.052143	-1.08858	-0.470723	18.544273
skew	0.0	-0.132649	0.163639	0.244917	0.282308	0.313576	3.569747
Var	3870.0	117.228377	118.755706	54.151103	176.251018	34.028376	8734295412.759695
Std	62.209324	10.827205	10.897509	7.358743	13.275956	5.833385	93457.45242

find outliers

In [12]:
```
lesser=[]
greater=[]
for colunName in quan:
    if(descriptive[colunName]["Min"]<descriptive[colunName]["Lesser"]):
        lesser.append(colunName)
    if(descriptive[colunName]["Max"]>descriptive[colunName]["Greater"]):
        greater.append(colunName)
```

In [13]:
```
lesser
```

Out[13]:
```
['hsc_p']
```

In [14]:
```
greater
```

Out[14]:
```
['hsc_p', 'degree_p', 'salary']
```

In [15]:
```
def Univariate(dataset,quan):
    descriptive=pd.DataFrame(index=["Mean","Median","Mode","Q1:25%","Q2:50%",
                                    "Q3:75%","99%","Q4:100%","IQR","1.5rule","Lesser","Greater","Min","Max","kurtosis","skew
    for columnName in quan:
        descriptive[columnName]["Mean"]=dataset[columnName].mean()
        descriptive[columnName]["Median"]=dataset[columnName].median()
        descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
        descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
        descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
        descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
        descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
        descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
        descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
        descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
        descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"
        descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
        descriptive[columnName]["Min"]=dataset[columnName].min()
        descriptive[columnName]["Max"]=dataset[columnName].max()
        descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
        descriptive[columnName]["skew"]=dataset[columnName].skew()
        descriptive[columnName]["Var"]=dataset[columnName].var()
        descriptive[columnName]["Std"]=dataset[columnName].std()
    return descriptive
```

In [16]:
```
Univariate(dataset,quan)
```

```
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:5: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:6: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:7: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:8: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:9: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value
```

```
Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:10: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:11: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:12: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:13: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:14: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:15: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:16: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:17: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Min"]=dataset[columnName].min()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:18: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Max"]=dataset[columnName].max()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:19: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
```

```
  descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:20: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["skew"]=dataset[columnName].skew()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:21: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Var"]=dataset[columnName].var()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:22: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Std"]=dataset[columnName].std()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:5: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:6: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:7: FutureWarning: ChainedAssignmentError: b
```

```
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:8: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:9: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:10: FutureWarning: ChainedAssignmentError: 
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:11: FutureWarning: ChainedAssignmentError: 
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:12: FutureWarning: ChainedAssignmentError: 
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
```

```
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:13: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:14: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:15: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:16: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:17: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:
```

```
df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Min"]=dataset[columnName].min()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:18: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Max"]=dataset[columnName].max()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:19: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:20: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["skew"]=dataset[columnName].skew()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:21: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Var"]=dataset[columnName].var()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:22: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
```

```
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Std"]=dataset[columnName].std()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:5: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:6: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:7: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:8: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:9: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
```

```
rning-a-view-versus-a-copy

  descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:10: FutureWarning: ChainedAssignmentError: 
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:11: FutureWarning: ChainedAssignmentError: 
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:12: FutureWarning: ChainedAssignmentError: 
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:13: FutureWarning: ChainedAssignmentError: 
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:14: FutureWarning: ChainedAssignmentError: 
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
```

```
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:15: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:16: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:17: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Min"]=dataset[columnName].min()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:18: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Max"]=dataset[columnName].max()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:19: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:20: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
```

```
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["skew"]=dataset[columnName].skew()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:21: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Var"]=dataset[columnName].var()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:22: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Std"]=dataset[columnName].std()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:5: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:6: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:7: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:
```

```
df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:8: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:9: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:10: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:11: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:12: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value
```

```
Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:13: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:14: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:15: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:16: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:17: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Min"]=dataset[columnName].min()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:18: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Max"]=dataset[columnName].max()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:19: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:20: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["skew"]=dataset[columnName].skew()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:21: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Var"]=dataset[columnName].var()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:22: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
```

```
  descriptive[columnName]["Std"]=dataset[columnName].std()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:5: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:6: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:7: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:8: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:9: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:10: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
```

```
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:11: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:12: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:13: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:14: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:15: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
```

```
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:16: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:17: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Min"]=dataset[columnName].min()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:18: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Max"]=dataset[columnName].max()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:19: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:20: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value
```

```
Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["skew"]=dataset[columnName].skew()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:21: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Var"]=dataset[columnName].var()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:22: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Std"]=dataset[columnName].std()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:5: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:6: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:7: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:8: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:9: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:10: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:11: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:12: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
```

```
  descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:13: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:14: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:15: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:16: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:17: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Min"]=dataset[columnName].min()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:18: FutureWarning: ChainedAssignmentError:
```

```
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Max"]=dataset[columnName].max()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:19: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:20: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["skew"]=dataset[columnName].skew()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:21: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Var"]=dataset[columnName].var()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:22: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Std"]=dataset[columnName].std()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:5: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
```

```
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mean"]=dataset[columnName].mean()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:6: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Median"]=dataset[columnName].median()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:7: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Mode"]=dataset[columnName].mode()[0]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:8: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:9: FutureWarning: ChainedAssignmentError: b
ehaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:10: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:
```

```
df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q3:75%"]=dataset.describe()[columnName]["75%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:11: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:12: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:13: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:14: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["1.5rule"]=1.5*descriptive[columnName]["IQR"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:15: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
```

```
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Lesser"]= descriptive[columnName]["Q1:25%"]- descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:16: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Greater"]=descriptive[columnName]["Q3:75%"]+descriptive[columnName]["1.5rule"]
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:17: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Min"]=dataset[columnName].min()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:18: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Max"]=dataset[columnName].max()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:19: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:20: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
```

```
rning-a-view-versus-a-copy

  descriptive[columnName]["skew"]=dataset[columnName].skew()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:21: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Var"]=dataset[columnName].var()
C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:22: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on
-Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF
rame or Series, because the intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy

  descriptive[columnName]["Std"]=dataset[columnName].std()
```

Out[16]:

	sl_no	ssc_p	hsc_p	degree_p	etest_p	mba_p	salary
Mean	108.0	67.303395	66.333163	66.370186	72.100558	62.278186	288655.405405
Median	108.0	67.0	65.0	66.0	71.0	62.0	265000.0
Mode	1	62.0	63.0	65.0	60.0	56.7	300000.0
Q1:25%	54.5	60.6	60.9	61.0	60.0	57.945	240000.0
Q2:50%	108.0	67.0	65.0	66.0	71.0	62.0	265000.0
Q3:75%	161.5	75.7	73.0	72.0	83.5	66.255	300000.0
99%	212.86	87.0	91.86	83.86	97.0	76.1142	NaN
Q4:100%	215.0	89.4	97.7	91.0	98.0	77.89	940000.0
IQR	107.0	15.1	12.1	11.0	23.5	8.31	60000.0
1.5rule	160.5	22.65	18.15	16.5	35.25	12.465	90000.0
Lesser	-106.0	37.95	42.75	44.5	24.75	45.48	150000.0
Greater	322.0	98.35	91.15	88.5	118.75	78.72	390000.0
Min	1	40.89	37.0	50.0	50.0	51.21	200000.0
Max	215	89.4	97.7	91.0	98.0	77.89	940000.0
kurtosis	-1.2	-0.60751	0.450765	0.052143	-1.08858	-0.470723	18.544273
skew	0.0	-0.132649	0.163639	0.244917	0.282308	0.313576	3.569747
Var	3870.0	117.228377	118.755706	54.151103	176.251018	34.028376	8734295412.759695
Std	62.209324	10.827205	10.897509	7.358743	13.275956	5.833385	93457.45242

In [ ]:

In [ ]:

In [ ]:

In [ ]: