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
           0
                                                                                                                                    Ы
                  1
                             67.00
                                    Others
                                            91.00
                                                   Others
                                                          Commerce
                                                                          58.00
                                                                                    Sci&Tech
                                                                                                 No
                                                                                                         55.0
                                                                                                                    Mkt&HR
                                                                                                                              58.80
           1
                  2
                              79.33
                                             78.33
                                                   Others
                                                                          77.48
                                                                                    Sci&Tech
                                                                                                         86.5
                                                                                                                    Mkt&Fin
                                                                                                                              66.28
                                                                                                                                   PΙ
                                   Central
                                                              Science
                                                                                                 Yes
           2
                  3
                              65.00
                                    Central
                                            68.00
                                                   Central
                                                                          64.00
                                                                                Comm&Mgmt
                                                                                                 No
                                                                                                         75.0
                                                                                                                    Mkt&Fin
                                                                                                                              57.80
           3
                  4
                              56.00
                                            52.00
                                                                          52.00
                                                                                    Sci&Tech
                                                                                                         66.0
                                                                                                                   Mkt&HR
                                    Central
                                                   Central
                                                             Science
                                                                                                 No
                                                                                                                              59.43
                                                                                                                                    ΡI
                                                                                                                              55.50 PI
           4
                  5
                             85.80
                                   Central
                                            73.60 Central Commerce
                                                                          73.30 Comm&Mgmt
                                                                                                 No
                                                                                                         96.8
                                                                                                                    Mkt&Fin
         210
                211
                              80.60
                                    Others
                                            82.00
                                                   Others
                                                           Commerce
                                                                          77.60
                                                                                Comm&Mgmt
                                                                                                 No
                                                                                                         91.0
                                                                                                                    Mkt&Fin
                                                                                                                              74.49 PI
         211
                212
                              58.00
                                    Others
                                            60.00
                                                   Others
                                                              Science
                                                                          72.00
                                                                                    Sci&Tech
                                                                                                 No
                                                                                                         74.0
                                                                                                                    Mkt&Fin
                                                                                                                              53.62 PI
         212
                213
                                                                                                                    Mkt&Fin
                                                                                                                              69.72 PI
                             67.00
                                    Others
                                            67.00
                                                   Others
                                                           Commerce
                                                                          73.00
                                                                                Comm&Mgmt
                                                                                                 Yes
                                                                                                         59.0
         213
                214
                              74.00
                                    Others
                                            66.00
                                                                          58.00
                                                                                Comm&Mamt
                                                                                                         70.0
                                                                                                                    Mkt&HR
                                                                                                                              60.23 PI
                                                   Others
                                                           Commerce
                                                                                                 No
                                                                                                                   Mkt&HR
                                                                                                                              60.22 <sub>PI</sub>
         214
                215
                             62.00 Central
                                            58.00
                                                   Others
                                                             Science
                                                                          53.00 Comm&Mgmt
                                                                                                 No
                                                                                                         89.0
        215 rows × 15 columns
         #we check the NaN values present
         dataset.isna().sum()
                               0
Out[3]: sl no
         gender
                              0
                              0
         ssc p
         ssc b
                              0
         hsc_p
                              0
         hsc b
                              0
                              0
         hsc s
         degree_p
                              0
         degree t
                              0
         workex
                              0
                              0
         etest p
         specialisation
                              0
         mba p
                              0
         status
                              0
         salarv
                              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=='0'):
                      qual.append(columnName)
                      quan.append(columnName)
             return quan, qual
         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]
```

 $C: \Users SowmiGanesh App Data \Local \Temp \ ip ykernel_11572 \ 1788620666.py : 3: Future \Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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 \indexer , "col"] = values instead, to perform the assignment in a single step and ensure this ke eps updating the original \indexer .

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

descriptive[columnName]["Mean"]=dataset[columnName].mean()

 $\verb|C:\Users| SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py: 4: Future \verb|Warning: Chained Assignment Error: behaviour will change in pandas 3.0! |$

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

df["col"][row_indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

Use $\dot{df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this keeps updating the original <math>\dot{df}$.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $`df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>`df`.$

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

descriptive[columnName]["Mean"]=dataset[columnName].mean()

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\1788620666.py: 4: Future \verb|Warning: Chained Assignment Error: behaviour will change in pandas 3.0! |$

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

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

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

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

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

df["col"][row indexer] = value

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

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

descriptive[columnName]["Mean"]=dataset[columnName].mean()

 $\verb|C:\Users\\| SowmiGanesh\\| AppData\\| Local\\| Temp\\| ipykernel_11572\\| 1788620666.py:4: Future Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

df["col"][row_indexer] = value

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

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

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

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

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["Mean"]=dataset[columnName].mean()

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\1788620666.py: 4: Future \verb|Warning: Chained Assignment Error: behaviour will change in pandas 3.0! |$

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#returning-a-view-versus-a-copy

descriptive[columnName]["Median"]=dataset[columnName].median()

 $\hbox{C:\Users} SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:5: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0! \\$

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

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row_indexer] = value

Use $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this keeps updating the original <math>\dot{`df}`.$

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

descriptive[columnName]["Mean"]=dataset[columnName].mean()

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

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

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

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

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

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

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

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

df["col"][row indexer] = value

Use $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\dot{`df}`.$

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

descriptive[columnName]["Mean"]=dataset[columnName].mean()

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\1788620666.py: 4: Future \verb|Warning: Chained Assignment Error: behaviour will change in pandas 3.0! |$

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#returning-a-view-versus-a-copy

descriptive[columnName]["Median"]=dataset[columnName].median()

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\1788620666.py:5: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0! |$

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

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

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

 $\hbox{C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\1788620666.py:3: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0! }$

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]: salary sl no hsc p degree p etest p mba p ssc p 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

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

df["col"][row indexer] = value

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

See the 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()
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

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

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

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\182625973.py:7: FutureWarning: ChainedAssignmentError: black the property of the p$

You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on -Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF

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

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

C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel 11572\182625973.py:8: FutureWarning: ChainedAssignmentError: b

You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on -Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF

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

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

C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel 11572\182625973.py:9: FutureWarning: ChainedAssignmentError: b

You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on -Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF

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

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

rame or Series, because the intermediate object on which we are setting values will behave as a copy.

rame or Series, because the intermediate object on which we are setting values will behave as a copy.

rame or Series, because the intermediate object on which we are setting values will behave as a copy.

rame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

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

descriptive[columnName]["Q1:25%"]=dataset.describe()[columnName]["25%"]

descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]

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

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

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

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

df["col"][row indexer] = value

eps updating the original `df`.

ehaviour will change in pandas 3.0!

df["col"][row indexer] = value

eps updating the original `df`.

ehaviour will change in pandas 3.0!

df["col"][row_indexer] = value

eps updating the original `df`.

ehaviour will change in pandas 3.0!

df["col"][row indexer] = value

eps updating the original `df`.

rning-a-view-versus-a-copy

rning-a-view-versus-a-copy

rning-a-view-versus-a-copy

rning-a-view-versus-a-copy

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

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

descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\182625973.py: 11: Future \verb|Warning: Chained AssignmentError: behaviour will change in pandas 3.0! |$

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

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

Use $\dot{df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\dot{df}$.

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

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

 $\label{thm:conversed} C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:14: Future\Warning: Chained\Assignment\Error: behaviour will change in pandas 3.0!$

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

df["col"][row_indexer] = value

Use $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\dot{`df}$ `.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this keeps updating the original <math>\dot{`df}`.$

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

descriptive[columnName]["Greater"]=descriptive[columnName]["03:75%"]+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#returning-a-view-versus-a-copy

descriptive[columnName]["Min"]=dataset[columnName].min()

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

df["col"][row indexer] = value

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

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

descriptive[columnName]["Max"]=dataset[columnName].max()

 $C: \Users SowmiGanesh AppData \Local Temp ipykernel_11572 \182625973.py: 18: Future Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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: $https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html \# returning-a-view-versus-a-copy$

descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()

 $\hbox{C:\Users} SowmiGanesh \App Data \Local \Temp \ ipy kernel_11572 \ 182625973.py:19: Future \Warning: Chained \Assignment \Error: behaviour will change in pandas 3.0! \\$

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#returning-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#returning-a-view-versus-a-copy

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

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

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

df["col"][row_indexer] = value

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

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

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

 $C: \ Users \ SowmiGanesh \ App Data \ Local \ Temp \ ipykernel_11572 \ 182625973.py: 4: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

df["col"][row indexer] = value

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

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

descriptive[columnName]["Median"]=dataset[columnName].median()

 $\hbox{C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:6: FutureWarning: ChainedAssignmentError: be a chaviour will change in pandas 3.0! }$

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

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

df["col"][row_indexer] = value

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

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

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

 $C: \ Users \ SowmiGanesh \ AppData \ Local \ Temp \ ipykernel_11572 \ 182625973.py: 7: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

df["col"][row indexer] = value

Use $\indexcolor{\ 'col"}] = values' instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\indexcolor{\ 'df'}.$

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

descriptive[columnName]["01:25%"]=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 $`df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>`df`.$

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $`df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>`df`.$

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\182625973.py: 11: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0! \\$

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 $\dot{df.loc[row_indexer, "col"]} = values \dot{instead}$, to perform the assignment in a single step and ensure this ke eps updating the original \dot{df} .

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

descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]

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

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

df["col"][row_indexer] = value

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

See the 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]["03:75%"]+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()

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\182625973.py: 17: Future \verb|Warning: Chained Assignment Error: Partial Part$ 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`.

 $See \ the \ caveats \ in \ the \ documentation: \ https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html \# return the documentation in the$ 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 \# return the \ documentation in the \ 
rning-a-view-versus-a-copy
  descriptive[columnName]["skew"]=dataset[columnName].skew()
\verb|C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel\_11572\182625973.py: 20: Future Warning: Chained Assignment Error: \\
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
```

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

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["Mean"]=dataset[columnName].mean()

 $C: \ Users \ SowmiGanesh \ App Data \ Local \ Temp \ ipykernel_11572 \ 182625973.py: 5: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row_indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\dot{`df}$ `.

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

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

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\182625973.py: 10: Future \verb|Warning: Chained Assignment Error: behaviour will change in pandas 3.0! |$

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.

```
df["col"][row_indexer] = value
Use `df.loc[row indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke
eps updating the original `df`.
See the 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"]
\verb|C:\Users\\| SowmiGanesh\\| AppData\\| Local\\| Temp\\| ipykernel\_11572\\| 182625973.py: 12: Future \verb|Warning: Chained Assignment Error: Part AppData|| Temp\\| App
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"]
\verb|C:\Users\\| SowmiGanesh\\| AppData\\| Local\\| Temp\\| ipykernel\_11572\\| 182625973.py: 15: Future \verb|Warning: Chained Assignment Error: Part AppData|| Temp\\| App
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
```

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

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

descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)

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

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

 $\verb|C:\USers\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\182625973.py: 11: Future \verb|Warning: Chained Assignment Error: Partial Part$

You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on -Write (which will become the default behaviour in pandas 3.0) this will never work to update the original DataF

rame or Series, because the intermediate object on which we are setting values will behave as a copy.

df["col"][row indexer] = value

eps updating the original `df`.

behaviour will change in pandas 3.0!

rning-a-view-versus-a-copy

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

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

descriptive[columnName]["Greater"]=descriptive[columnName]["03:75%"]+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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

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

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

descriptive[columnName]["Min"]=dataset[columnName].min()

 $\verb| C:\Users\\| SowmiGanesh\\| AppData\\| Local\\| Temp\\| ipykernel_11572\\| 182625973.py:17: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!$

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

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

descriptive[columnName]["Max"]=dataset[columnName].max()

 $\hbox{C:\Users} SowmiGanesh App Data Local Temp ipy kernel_11572 \ 182625973.py:18: Future Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\$

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

df["col"][row indexer] = value

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

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

descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()

 $C: \Users SowmiGanesh AppData \Local Temp ipykernel_11572 \182625973.py: 19: Future Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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 $\dot{df.loc[row_indexer, "col"]} = values \dot{instead}$, to perform the assignment in a single step and ensure this ke eps updating the original \dot{df} .

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this keeps updating the original <math>\dot{`df}$ `.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $`df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>`df`.$

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

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

 $C: \ Users \ SowmiGanesh \ App Data \ Local \ Temp \ ipykernel_11572 \ 182625973.py: 4: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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 $\dot{\dot} df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\dot{\dot} df`.$

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

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

 $C: \ Users \ SowmiGanesh \ AppData \ Local \ Temp \ ipykernel_11572 \ 182625973.py: 7: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

df["col"][row_indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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%"]

 $C: \ Users \ SowmiGanesh \ App Data \ Local \ Temp \ ipykernel_11572 \ 182625973.py: 9: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

df["col"][row indexer] = value

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

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

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

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\182625973.py: 10: Future \verb|Warning: Chained AssignmentError: behaviour will change in pandas 3.0! |$

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

df["col"][row indexer] = value

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

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

descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)

 $\hbox{$C:\Users\SowmiGanesh\AppData\SoelTemp\ipykernel_11572\S05973.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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\182625973.py:12: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0! |$

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

df["col"][row indexer] = value

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

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

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 cert

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

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

df["col"][row indexer] = value

Use $\dot{df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this keeps updating the original <math>\dot{df}$.

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

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

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

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

df["col"][row indexer] = value

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

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

descriptive[columnName]["Lesser"]= descriptive[columnName]["01:25%"]- 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#returning-a-view-versus-a-copy

descriptive[columnName]["Greater"]=descriptive[columnName]["03:75%"]+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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

descriptive[columnName]["Min"]=dataset[columnName].min()

 $\verb|C:\Users\\| SowmiGanesh\\| AppData\\| Local\\| Temp\\| ipykernel_11572\\| 182625973.py:17: Future Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()

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

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["skew"]=dataset[columnName].skew()

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

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

df["col"][row_indexer] = value

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

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

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

 $\hbox{C:\Users} SowmiGanesh App Data Local Temp ipy kernel_11572 \ 182625973.py: 21: Future Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\$

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

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row_indexer] = value

Use $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this keeps updating the original <math>\dot{`df}`.$

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

descriptive[columnName]["Mean"]=dataset[columnName].mean()

 $C: \ Users \ SowmiGanesh \ AppData \ Local \ Temp \ ipykernel_11572 \ 182625973.py: 5: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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()

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]

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\182625973.py:7: FutureWarning: ChainedAssignmentError: but the property of the pr$ 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%"]

 $\texttt{C:} \\ \texttt{Users} \\ \texttt{SowmiGanesh} \\ \texttt{AppData} \\ \texttt{Local} \\ \texttt{Temp} \\ \texttt{ipykernel 11572} \\ \texttt{182625973.py:9: FutureWarning: ChainedAssignmentError: b.} \\ \texttt{b.} \\ \texttt{c.} \\ \texttt{d.} \\ \texttt$ 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#returning-a-view-versus-a-copy

descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)

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

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]

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

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["IQR"]= 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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

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

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\182625973.py:14: Future \verb|Warning: Chained Assignment Error: behaviour will change in pandas 3.0! |$

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

df["col"][row indexer] = value

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

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

descriptive[columnName]["Lesser"]= descriptive[columnName]["01:25%"]- 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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

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

See the 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()
```

 $\verb|C:\Users\\| SowmiGanesh\\| AppData\\| Local\\| Temp\\| ipykernel_11572\\| 182625973.py: 17: Future \verb|Warning: Chained Assignment Error: Part AppData|| Temp\\| App$ 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()

 $\verb|C:\Users\\| SowmiGanesh\\| AppData\\| Local\\| Temp\\| ipykernel_11572\\| 182625973.py: 18: Future \verb|Warning: Chained Assignment Error: Part AppData|| Temp\\| App$ 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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

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

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

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

 $C: \ Users \ SowmiGanesh \ App Data \ Local \ Temp \ ipykernel_11572 \ 182625973.py: 4: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

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

descriptive[columnName]["Mean"]=dataset[columnName].mean()

 $C: \ Users \ SowmiGanesh \ App Data \ Local \ Temp \ ipykernel_11572 \ 182625973.py:5: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

Use $\dot{df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\dot{df}$.

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

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

 $C: \ Users \ SowmiGanesh \ App Data \ Local \ Temp \ ipykernel_11572 \ 182625973.py:7: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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 $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\dot{`df}$ `.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $`df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>`df`.$

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row indexer] = value

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

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

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

 $\hbox{C:\Users} SowmiGanesh\AppData\Local\Temp\ipykernel_11572\182625973.py:10: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0! \\$

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row indexer] = value

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

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

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#returning-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#returning-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#returning-a-view-versus-a-copy

descriptive[columnName]["Greater"]=descriptive[columnName]["03:75%"]+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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

descriptive[columnName]["Min"]=dataset[columnName].min()

 $\hbox{C:\Users} SowmiGanesh \App Data \Local \Temp \ ipy kernel_11572 \ 182625973.py:17: Future \Warning: Chained Assignment \Error: behaviour will change in pandas 3.0! \\$

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

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

descriptive[columnName]["Max"]=dataset[columnName].max()

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\182625973.py:18: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0! |$

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

df["col"][row indexer] = value

Use $\indexcolor{\ 'col"}] = values' instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\indexcolor{\ 'df'}.$

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

descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()

 $\verb| C:\Users\\| SowmiGanesh\\| AppData\\| Local\\| Temp\\| ipykernel_11572\\| 182625973.py:19: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!$

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

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

descriptive[columnName]["skew"]=dataset[columnName].skew()

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

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

df["col"][row_indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["Mean"]=dataset[columnName].mean()

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

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

df["col"][row_indexer] = value

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

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

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#returning-a-view-versus-a-copy

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

 $C: \ Users \ SowmiGanesh \ App Data \ Local \ Temp \ ipykernel_11572 \ 182625973.py: 7: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row_indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $`df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>`df`.$

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

descriptive[columnName]["02:50%"]=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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

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

 $\verb| C:\Users\\| SowmiGanesh\\| AppData\\| Local\\| Temp\\| ipykernel_11572\\| 182625973.py:10: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!$

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

df["col"][row indexer] = value

Use $\dot{ `df.loc[row_indexer, "col"] = values ` instead, to perform the assignment in a single step and ensure this keeps updating the original `df`.}$

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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\Loca\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 Cony-on.
```

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["IQR"]= 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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

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

 $C: \Users SowmiGanesh AppData \Local Temp ipykernel_11572 \182625973.py: 14: Future Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["Lesser"]= descriptive[columnName]["01:25%"]- 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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

descriptive[columnName]["Min"]=dataset[columnName].min()

 $\hbox{C:\Users} SowmiGanesh App Data Local Temp ipy kernel_11572 \ 182625973.py:17: Future Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\$

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

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $`df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>`df`.$

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

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

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\182625973.py: 21: Future \verb|Warning: Chained Assignment Error: behaviour will change in pandas 3.0! |$

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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row_indexer] = value

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

See the 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
```

:	sl_no	ssc_p	hsc_p	degree_p	etest_p	mba_p	salary
Mea	n 108.0	67.303395	66.333163	66.370186	72.100558	62.278186	288655.405405
Media	n 108.0	67.0	65.0	66.0	71.0	62.0	265000.0
Mod	e 1	62.0	63.0	65.0	60.0	56.7	300000.0
Q1:25%	6 54.5	60.6	60.9	61.0	60.0	57.945	240000.0
Q2:50%	6 108.0	67.0	65.0	66.0	71.0	62.0	265000.0
Q3:75%	6 161.5	75.7	73.0	72.0	83.5	66.255	300000.0
99%	2 12.86	87.0	91.86	83.86	97.0	76.1142	NaN
Q4:100%	6 215.0	89.4	97.7	91.0	98.0	77.89	940000.0
IQI	R 107.0	15.1	12.1	11.0	23.5	8.31	60000.0
1.5rul	e 160.5	22.65	18.15	16.5	35.25	12.465	90000.0
Lesse	r -106.0	37.95	42.75	44.5	24.75	45.48	150000.0
Greate	r 322.0	98.35	91.15	88.5	118.75	78.72	390000.0
Mi	n 1	40.89	37.0	50.0	50.0	51.21	200000.0
Ma	x 215	89.4	97.7	91.0	98.0	77.89	940000.0
kurtosi	- 1.2	-0.60751	0.450765	0.052143	-1.08858	-0.470723	18.544273
skev	v 0.0	-0.132649	0.163639	0.244917	0.282308	0.313576	3.569747
Va	r 3870.0	117.228377	118.755706	54.151103	176.251018	34.028376	8734295412.759695
Sto	d 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"]):</pre>
                 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","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]["03:75%"]+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)

 $C: \ Users \ SowmiGanesh \ AppData \ Local \ Temp \ ipykernel_11572 \ 895335533.py:5: \ Future \ Warning: \ Chained \ Assignment \ Error: \ behaviour will change in pandas 3.0!$

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

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

descriptive[columnName]["Mean"]=dataset[columnName].mean()

 $C: \ Users \ SowmiGanesh \ AppData \ Local \ Temp \ ipykernel_11572 \ 895335533.py: 6: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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#returning-a-view-versus-a-copy

descriptive[columnName]["Median"]=dataset[columnName].median()

 $C: \ Users \ SowmiGanesh \ App Data \ Local \ Temp \ ipykernel_11572 \ 895335533.py:7: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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 $\dot{df.loc[row_indexer, "col"]} = values \dot{instead}$, to perform the assignment in a single step and ensure this ke eps updating the original \dot{df} .

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

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

 $C: \ Users \ SowmiGanesh \ AppData \ Local \ Temp \ ipykernel_11572 \ 895335533.py: 8: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]

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

df["col"][row_indexer] = value

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

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

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

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)

 $\hbox{C:\Users} SowmiGanesh App Data Local Temp ipy kernel_11572 \ 895335533.py:12: Future Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\$

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

df["col"][row indexer] = value

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

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

descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]

 $\label{local-Temp-ipy-kernel} C: \label{local-Temp-ipy-kernel} In State of the control of the$

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

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

df["col"][row_indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this keeps updating the original <math>\dot{`df}$ `.

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

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

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

descriptive[columnName]["Lesser"]= descriptive[columnName]["01:25%"]- 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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use $\dot{\dot} df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\dot{\dot} df`.$

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

descriptive[columnName]["Greater"]=descriptive[columnName]["03:75%"]+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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

descriptive[columnName]["Max"]=dataset[columnName].max()

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

df["col"][row_indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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()

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

df["col"][row indexer] = value

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

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

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

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\895335533.py: 22: Future \verb|Warning: Chained AssignmentError: behaviour will change in pandas 3.0! |$

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

df["col"][row indexer] = value

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

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

 $\tt descriptive[columnName]["Std"] = dataset[columnName].std()$

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

df["col"][row indexer] = value

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

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

descriptive[columnName]["Mean"]=dataset[columnName].mean()

 $C: \ Users \ SowmiGanesh \ App Data \ Local \ Temp \ ipykernel_11572 \ 895335533.py: 6: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

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

descriptive[columnName]["Median"]=dataset[columnName].median()

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#returning-a-view-versus-a-copy

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

 $C: \ Users \ SowmiGanesh \ AppData \ Local \ Temp \ ipykernel_11572 \ 895335533.py: 8: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

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

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

descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]

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

df["col"][row indexer] = value

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

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

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

 $\label{thm:c:UsersSowmiGaneshAppDataLocalTemp\ipykernel_11572\895335533.py: 11: Future Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

df["col"][row_indexer] = value

Use $\dot{df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\dot{df}$.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $\dot{df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\dot{df}$.

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

descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]

 $\verb| C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\895335533.py:13: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0! |$

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#returning-a-view-versus-a-copy

descriptive[columnName]["IQR"]= 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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

descriptive[columnName]["Lesser"]= descriptive[columnName]["01:25%"]- 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 $\dot{"col"} = \dot{"col"} = \dot{"col"} = \dot{"col"} = \dot{"col"}$ instead, to perform the assignment in a single step and ensure this ke eps updating the original $\dot{"df}$.

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

descriptive[columnName]["Greater"]=descriptive[columnName]["03:75%"]+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 $\dot{df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this keeps updating the original <math>\dot{df}$.

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

descriptive[columnName]["Max"]=dataset[columnName].max()

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

df["col"][row indexer] = value

Use $\dot{"col"} = \dot{"col"} = \dot{"col"} = \dot{"col"} = \dot{"col"}$ instead, to perform the assignment in a single step and ensure this ke eps updating the original $\dot{"df}$.

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

descriptive[columnName]["skew"]=dataset[columnName].skew()

 $\verb|C:\Users\\| SowmiGanesh\\| AppData\\| Loca\\| Temp\\| ipykernel_11572\\| 895335533.py: 21: Future Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\| AppData\\| Loca\\| Temp\\| Institute Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\| AppData\\| Institute Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\| AppData\\| Institute Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\| AppData\\| Institute Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\| AppData\\| Institute Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\| AppData\\| Institute Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\| AppData\\| Institute Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\| AppData\\| Institute Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\| AppData\\| Institute Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\| AppData\\| Institute Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\| AppData\\| Institute Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\| AppData\\| Institute Warning: Chained Assignment Error: behaviour will be a simple wi$

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

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

df["col"][row_indexer] = value

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

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

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

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\895335533.py: 22: Future \verb|Warning: Chained AssignmentError: behaviour will change in pandas 3.0! |$

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#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row_indexer] = value

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

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

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $`df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>`df`.$

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $`df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>`df`.$

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

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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row_indexer] = value

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

 $See \ the \ caveats \ in \ the \ documentation: \ https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html \# return the documentation in the$

```
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%"]
\verb|C:\Users\\| SowmiGanesh\\| AppData\\| Local\\| Temp\\| ipykernel\_11572\\| 895335533.py: 11: Future \verb|Warning: Chained Assignment Error: Part AppData|| Temp\\| App
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)
\verb|C:\Users\\| SowmiGanesh\\| AppData\\| Local\\| Temp\\| ipykernel\_11572\\| 895335533.py: 12: Future \verb|Warning: Chained Assignment Error: Part AppData|| Temp\\| App
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 $`df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>`df`.$

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

descriptive[columnName]["Lesser"]= descriptive[columnName]["01:25%"]- 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: $https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy$

descriptive[columnName]["Greater"]=descriptive[columnName]["03:75%"]+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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

descriptive[columnName]["Min"]=dataset[columnName].min()

 $\hbox{C:\sc SowmiGanesh\appData\encolor} \label{thm:cond} \label{thm:cond} \hbox{C:\sc SowmiGanesh\appData\encolor} \label{thm:cond} \hbox{C:\sc SowmiGanesh\appData\encolor} \label{thm:cond} \hbox{C:\sc SowmiGanesh\appData\encolor} \label{thm:condition} \hbox{ChainedAssignmentError:} \\ \hbox{behaviour will change in pandas 3.0!} \\ \hbox{C:\sc SowmiGanesh\appData\encolor} \label{thm:condition}$

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

df["col"][row indexer] = value

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

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

descriptive[columnName]["Max"]=dataset[columnName].max()

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\895335533.py: 20: Future \verb|Warning: Chained AssignmentError: behaviour will change in pandas 3.0! |$

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 $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\dot{`df}$ `.

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

descriptive[columnName]["skew"]=dataset[columnName].skew()

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\895335533.py: 21: Future \verb|Warning: Chained Assignment Error: behaviour will change in pandas 3.0! |$

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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row indexer] = value

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

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

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

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

df["col"][row indexer] = value

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

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

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

 $C: \ Users \ SowmiGanesh \ AppData \ Local \ Temp \ ipykernel_11572 \ 895335533.py:5: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

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

descriptive[columnName]["Mean"]=dataset[columnName].mean()

 $\hbox{C:} Users \onote that $$C: Users \onote t$

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

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)

 ${\tt C:\Users\backslash SowmiGanesh\backslash AppData\backslash Local\backslash Temp\backslash ipykernel~11572\backslash 895335533.py: 12:~Future Warning:~Chained Assignment Error:~C:\Users\backslash SowmiGanesh\backslash AppData\backslash Local\backslash Temp\backslash ipykernel~11572\backslash 895335533.py: 12:~Future Warning:~Chained Assignment Error:~C:\Users\backslash SowmiGanesh\backslash AppData\backslash Local\backslash Temp\backslash ipykernel~11572\backslash 895335533.py: 12:~Future Warning:~Chained Assignment Error:~C:\Users\backslash SowmiGanesh\ AppData\backslash Local\ AppData\backslash Local\ AppData\ AppData\$ 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 $\indexcolor{\ 'col"}] = values' instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\indexcolor{\ 'df'}.$

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

descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]

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

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

df["col"][row indexer] = value

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

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

descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]

 $\verb| C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\895335533.py:14: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0! |$

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

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

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

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

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["Lesser"]= descriptive[columnName]["01:25%"]- 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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

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#returning-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#returning-a-view-versus-a-copy

descriptive[columnName]["Max"]=dataset[columnName].max()

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

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["kurtosis"]=dataset[columnName].kurtosis()

 $\hbox{C:\Users} SowmiGanesh App Data Local Temp ipy kernel_11572 \times 95335533.py: 20: Future Warning: Chained Assignment Error: behaviour will change in pandas 3.0! \\$

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

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

descriptive[columnName]["skew"]=dataset[columnName].skew()

 $\hbox{$C:\Users\SowmiGanesh\AppData\SormentError: 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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

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

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

df["col"][row indexer] = value

Use $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\dot{`df}`.$

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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()
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 \# return the \ documentation in the \$ rning-a-view-versus-a-copy

descriptive[columnName]["Median"]=dataset[columnName].median()

 $\texttt{C:} \\ \texttt{Users} \\ \texttt{SowmiGanesh} \\ \texttt{AppData} \\ \texttt{Local} \\ \texttt{Temp} \\ \texttt{ipykernel 11572} \\ \texttt{895335533.py:7:} \\ \texttt{FutureWarning: ChainedAssignmentError: b} \\ \texttt{b} \\ \texttt{constant} \\ \texttt{$ 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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row_indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

descriptive[columnName]["IQR"]= descriptive[columnName]["Q3:75%"]-descriptive[columnName]["Q1:25%"]

 $\hbox{C:\Users} SowmiGanesh\AppData\Local\Temp\ipykernel_11572\895335533.py:14: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0! \\$

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 $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\dot{`df}$ `.

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

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

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\895335533.py:15: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0! |$

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"]
\verb|C:\Users\\| SowmiGanesh\\| AppData\\| Local\\| Temp\\| ipykernel\_11572\\| 895335533.py: 16: Future \verb|Warning: Chained Assignment Error: Part AppData|| Temp\\| App
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"]
\verb|C:\Users\\| SowmiGanesh\\| AppData\\| Local\\| Temp\\| ipykernel\_11572\\| 895335533.py: 17: Future \verb|Warning: Chained Assignment Error: Part AppData|| Temp\\| App
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()
\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel 11572\\895335533.py: 18: Future \verb|Warning: Chained Assignment Error: Part Solution of the property of the
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()
\verb|C:\Users\SowmiGanesh\AppData\Local\Temp\ipykernel\_11572\895335533.py:19: Future Warning: Chained Assignment Error: \\
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()
\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel\_11572\\895335533.py: 20: Future \verb|Warning: Chained Assignment Error: Partial Control of the Control of 
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#returning-a-view-versus-a-copy

descriptive[columnName]["skew"]=dataset[columnName].skew()

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

df["col"][row_indexer] = value

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

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

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

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

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

A typical example is when you are setting values in a column of a

df["col"][row_indexer] = value

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

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

descriptive[columnName]["Mean"]=dataset[columnName].mean()

 $C: \ Users \ SowmiGanesh \ App Data \ Local \ Temp \ ipykernel_11572 \ 895335533.py: 6: Future \ Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this keeps updating the original <math>\dot{`df}$ `.

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

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 $\dot{ df.loc[row_indexer, "col"] = values ` instead, to perform the assignment in a single step and ensure this keeps updating the original <math>\dot{ df }$.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>\dot{`df}`.$

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

descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]

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

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

df["col"][row indexer] = value

Use $\indexcolor{\ }$ use $\indexcolor{\ }$ uses instead, to perform the assignment in a single step and ensure this ke eps updating the original $\indexcolor{\ }$ df.

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

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

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)

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

df["col"][row_indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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%"]
{\tt C:\Users\backslash SowmiGanesh\backslash AppData\backslash Local\backslash Temp\backslash ipykernel~11572\backslash 895335533.py:14:~Future Warning:~Chained Assignment Error:~Chained Assignment Err
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 $`df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>`df`.$

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

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

df["col"][row_indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row indexer] = value

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

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

descriptive[columnName]["Greater"]=descriptive[columnName]["03:75%"]+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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

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

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

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

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

descriptive[columnName]["Max"]=dataset[columnName].max()

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

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

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

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

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

descriptive[columnName]["skew"]=dataset[columnName].skew()

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

df["col"][row indexer] = value

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

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

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

 $\verb|C:\Users\\| SowmiGanesh\\| AppData\\| Local\\| Temp\\| ipykernel_11572\\| 895335533.py: 22: Future Warning: Chained Assignment Error: behaviour will change in pandas 3.0!$

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

df["col"][row_indexer] = value

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

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

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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["Median"]=dataset[columnName].median()

 $C: \ Users \ SowmiGanesh \ AppData \ Local \ Temp \ ipykernel_11572 \ 895335533.py:7: \ Future \ Warning: \ Chained \ Assignment \ Error: \ behaviour will change in pandas 3.0!$

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

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $\dot{`df.loc[row_indexer, "col"]} = values` instead, to perform the assignment in a single step and ensure this keeps updating the original <math>\dot{`df}`.$

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

descriptive[columnName]["Q2:50%"]=dataset.describe()[columnName]["50%"]

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

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

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

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

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

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

descriptive[columnName]["99%"]=np.percentile(dataset[columnName],99)

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

df["col"][row_indexer] = value

Use $\dot{"col"} = \dot{"col"} = \dot{"col"} = \dot{"col"} = \dot{"col"}$ instead, to perform the assignment in a single step and ensure this ke eps updating the original $\dot{"df}$.

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

descriptive[columnName]["Q4:100%"]=dataset.describe()[columnName]["max"]

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

df["col"][row_indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 $`df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this ke eps updating the original <math>`df`.$

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

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

 $\verb|C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\895335533.py:15: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0! \\$

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

df["col"][row_indexer] = value

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

eps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy. A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

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

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

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

df["col"][row_indexer] = value

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

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

descriptive[columnName]["Min"]=dataset[columnName].min()

 $\verb| C:\Users\\SowmiGanesh\\AppData\\Local\\Temp\\ipykernel_11572\\895335533.py:18: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0! |$

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

df["col"][row indexer] = value

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

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

descriptive[columnName]["Max"]=dataset[columnName].max()

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

df["col"][row indexer] = value

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

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

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

df["col"][row indexer] = value

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

See the 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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

df["col"][row indexer] = value

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-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 DataFrame or Series, because the intermediate object on which we are setting values will behave as a copy.

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

hec n dograe n

df["col"][row_indexer] = value

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

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

mha n

calary

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

		 7.1	E 1	
	u i	т,		

	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	[]:	