oqureywey

October 8, 2024

1 Analysis of AMCAT Data

1.1 Introduction

1.1.1 Objective

The goal of this Exploratory Data Analysis (EDA) is to extensively investigate the provided dataset, with a particular emphasis on understanding the link between various variables and the target variable, Salary.

The key aims of this analysis include:

- Providing a detailed explanation of the dataset's features.
- Find any observable patterns or trends in the data.
- Investigating the relationships between the independent factors and the target variable (salary).
- Identify any outliers or abnormalities in the dataset.
- Offering practical insights and recommendations based on the analysis.

1.2 Importing the data and dislaying the head, shape, descriptions, etc.

[ ]:
```
import pandas as pd
import numpy as np
pd.set_option('display.max_columns',100)
```

[ ]:
```
ameo_data = pd.read_excel('/content/data.xlsx')
df1 = ameo_data.copy()
df1.head()
```

[ ]:
```
  Unnamed: 0      ID   Salary        DOJ                 DOL  \
0      train  203097   420000 2012-06-01             present
1      train  579905   500000 2013-09-01             present
2      train  810601   325000 2014-06-01             present
3      train  267447  1100000 2011-07-01             present
4      train  343523   200000 2014-03-01  2015-03-01 00:00:00

                Designation    JobCity Gender        DOB  10percentage  \
0   senior quality engineer  Bangalore      f 1990-02-19          84.3
1         assistant manager     Indore      m 1989-10-04          85.4
2          systems engineer    Chennai      f 1992-08-03          85.0
```

```
3  senior software engineer    Gurgaon      m 1989-12-05          85.6
4                       get    Manesar      m 1991-02-27          78.0

                          10board  12graduation  12percentage  \
0  board ofsecondary education,ap          2007          95.8
1                             cbse          2007          85.0
2                             cbse          2010          68.2
3                             cbse          2007          83.6
4                             cbse          2008          76.8

                              12board  CollegeID  CollegeTier       Degree  \
0  board of intermediate education,ap       1141            2  B.Tech/B.E.
1                                cbse       5807            2  B.Tech/B.E.
2                                cbse         64            2  B.Tech/B.E.
3                                cbse       6920            1  B.Tech/B.E.
4                                cbse      11368            2  B.Tech/B.E.

                             Specialization  collegeGPA  CollegeCityID  \
0                       computer engineering       78.00           1141
1  electronics and communication engineering       70.06           5807
2                     information technology       70.00             64
3                       computer engineering       74.64           6920
4  electronics and communication engineering       73.90          11368

   CollegeCityTier    CollegeState  GraduationYear  English  Logical  Quant  \
0                0  Andhra Pradesh            2011      515      585    525
1                0  Madhya Pradesh            2012      695      610    780
2                0   Uttar Pradesh            2014      615      545    370
3                1           Delhi            2011      635      585    625
4                0   Uttar Pradesh            2012      545      625    465

     Domain  ComputerProgramming  ElectronicsAndSemicon  ComputerScience  \
0  0.635979                  445                     -1               -1
1  0.960603                   -1                    466               -1
2  0.450877                  395                     -1               -1
3  0.974396                  615                     -1               -1
4  0.124502                   -1                    233               -1

   MechanicalEngg  ElectricalEngg  TelecomEngg  CivilEngg  conscientiousness  \
0              -1              -1           -1         -1             0.9737
1              -1              -1           -1         -1            -0.7335
2              -1              -1           -1         -1             0.2718
3              -1              -1           -1         -1             0.0464
4              -1              -1           -1         -1            -0.8810

   agreeableness  extraversion  nueroticism  openess_to_experience
0         0.8128        0.5269      1.35490                -0.4455
```

```
1         0.3789        1.2396     -0.10760                0.8637
2         1.7109        0.1637     -0.86820                0.6721
3         0.3448       -0.3440     -0.40780               -0.9194
4        -0.2793       -1.0697      0.09163               -0.1295
```

[ ]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3998 entries, 0 to 3997
Data columns (total 39 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             3998 non-null   object
 1   ID                     3998 non-null   int64
 2   Salary                 3998 non-null   int64
 3   DOJ                    3998 non-null   datetime64[ns]
 4   DOL                    3998 non-null   object
 5   Designation            3998 non-null   object
 6   JobCity                3998 non-null   object
 7   Gender                 3998 non-null   object
 8   DOB                    3998 non-null   datetime64[ns]
 9   10percentage           3998 non-null   float64
 10  10board                3998 non-null   object
 11  12graduation           3998 non-null   int64
 12  12percentage           3998 non-null   float64
 13  12board                3998 non-null   object
 14  CollegeID              3998 non-null   int64
 15  CollegeTier            3998 non-null   int64
 16  Degree                 3998 non-null   object
 17  Specialization         3998 non-null   object
 18  collegeGPA             3998 non-null   float64
 19  CollegeCityID          3998 non-null   int64
 20  CollegeCityTier        3998 non-null   int64
 21  CollegeState           3998 non-null   object
 22  GraduationYear         3998 non-null   int64
 23  English                3998 non-null   int64
 24  Logical                3998 non-null   int64
 25  Quant                  3998 non-null   int64
 26  Domain                 3998 non-null   float64
 27  ComputerProgramming    3998 non-null   int64
 28  ElectronicsAndSemicon  3998 non-null   int64
 29  ComputerScience        3998 non-null   int64
 30  MechanicalEngg         3998 non-null   int64
 31  ElectricalEngg         3998 non-null   int64
 32  TelecomEngg            3998 non-null   int64
 33  CivilEngg              3998 non-null   int64
 34  conscientiousness      3998 non-null   float64
```

```
 35  agreeableness          3998 non-null   float64
 36  extraversion           3998 non-null   float64
 37  nueroticism            3998 non-null   float64
 38  openess_to_experience  3998 non-null   float64
dtypes: datetime64[ns](2), float64(9), int64(18), object(10)
memory usage: 1.2+ MB
```

[ ]:
```
df1.shape
```

[ ]:
```
(3998, 39)
```

[ ]:
```
df1.isnull().sum()
```

[ ]:
```
Unnamed: 0              0
ID                      0
Salary                  0
DOJ                     0
DOL                     0
Designation             0
JobCity                 0
Gender                  0
DOB                     0
10percentage            0
10board                 0
12graduation            0
12percentage            0
12board                 0
CollegeID               0
CollegeTier             0
Degree                  0
Specialization          0
collegeGPA              0
CollegeCityID           0
CollegeCityTier         0
CollegeState            0
GraduationYear          0
English                 0
Logical                 0
Quant                   0
Domain                  0
ComputerProgramming     0
ElectronicsAndSemicon   0
ComputerScience         0
MechanicalEngg          0
ElectricalEngg          0
TelecomEngg             0
CivilEngg               0
```

```
conscientiousness        0
agreeableness            0
extraversion             0
nueroticism              0
openess_to_experience    0
dtype: int64
```

[ ]:
```
df1.duplicated().sum()
```

[ ]:
```
0
```

[ ]:
```
df1.columns
```

[ ]:
```
Index(['Unnamed: 0', 'ID', 'Salary', 'DOJ', 'DOL', 'Designation', 'JobCity',
       'Gender', 'DOB', '10percentage', '10board', '12graduation',
       '12percentage', '12board', 'CollegeID', 'CollegeTier', 'Degree',
       'Specialization', 'collegeGPA', 'CollegeCityID', 'CollegeCityTier',
       'CollegeState', 'GraduationYear', 'English', 'Logical', 'Quant',
       'Domain', 'ComputerProgramming', 'ElectronicsAndSemicon',
       'ComputerScience', 'MechanicalEngg', 'ElectricalEngg', 'TelecomEngg',
       'CivilEngg', 'conscientiousness', 'agreeableness', 'extraversion',
       'nueroticism', 'openess_to_experience'],
      dtype='object')
```

[ ]:
```
df1.nunique()
```

[ ]:
```
Unnamed: 0                1
ID                     3998
Salary                  177
DOJ                      81
DOL                      67
Designation             419
JobCity                 339
Gender                    2
DOB                    1872
10percentage            851
10board                 275
12graduation             16
12percentage            801
12board                 340
CollegeID              1350
CollegeTier               2
Degree                    4
Specialization           46
collegeGPA             1282
CollegeCityID          1350
CollegeCityTier           2
```

```
CollegeState               26
GraduationYear             11
English                   111
Logical                   107
Quant                     138
Domain                    243
ComputerProgramming        79
ElectronicsAndSemicon      29
ComputerScience            20
MechanicalEngg             42
ElectricalEngg             31
TelecomEngg                26
CivilEngg                  23
conscientiousness         141
agreeableness             149
extraversion              154
nueroticism               217
openess_to_experience     142
dtype: int64
```

[ ]:
```
df1 = df1.drop(columns = ['Unnamed: 0', 'ID','CollegeID','CollegeCityID'])
df1.head()
```

[ ]:
```
    Salary        DOJ                 DOL              Designation  \
0   420000 2012-06-01             present   senior quality engineer
1   500000 2013-09-01             present         assistant manager
2   325000 2014-06-01             present          systems engineer
3  1100000 2011-07-01             present  senior software engineer
4   200000 2014-03-01  2015-03-01 00:00:00                      get

     JobCity Gender        DOB  10percentage                         10board  \
0  Bangalore      f 1990-02-19          84.3  board ofsecondary education,ap
1     Indore      m 1989-10-04          85.4                            cbse
2    Chennai      f 1992-08-03          85.0                            cbse
3    Gurgaon      m 1989-12-05          85.6                            cbse
4    Manesar      m 1991-02-27          78.0                            cbse

   12graduation  12percentage                              12board  \
0          2007          95.8  board of intermediate education,ap
1          2007          85.0                                cbse
2          2010          68.2                                cbse
3          2007          83.6                                cbse
4          2008          76.8                                cbse

   CollegeTier       Degree                              Specialization  \
0            2  B.Tech/B.E.                         computer engineering
1            2  B.Tech/B.E.  electronics and communication engineering
```

```
2            2  B.Tech/B.E.                     information technology
3            1  B.Tech/B.E.                       computer engineering
4            2  B.Tech/B.E.  electronics and communication engineering

   collegeGPA  CollegeCityTier    CollegeState  GraduationYear  English  \
0       78.00               0  Andhra Pradesh            2011      515
1       70.06               0  Madhya Pradesh            2012      695
2       70.00               0   Uttar Pradesh            2014      615
3       74.64               1           Delhi            2011      635
4       73.90               0   Uttar Pradesh            2012      545

   Logical  Quant    Domain  ComputerProgramming  ElectronicsAndSemicon  \
0      585    525  0.635979                  445                     -1
1      610    780  0.960603                   -1                    466
2      545    370  0.450877                  395                     -1
3      585    625  0.974396                  615                     -1
4      625    465  0.124502                   -1                    233

   ComputerScience  MechanicalEngg  ElectricalEngg  TelecomEngg  CivilEngg  \
0               -1              -1              -1           -1         -1
1               -1              -1              -1           -1         -1
2               -1              -1              -1           -1         -1
3               -1              -1              -1           -1         -1
4               -1              -1              -1           -1         -1

   conscientiousness  agreeableness  extraversion  nueroticism  \
0             0.9737         0.8128        0.5269      1.35490
1            -0.7335         0.3789        1.2396     -0.10760
2             0.2718         1.7109        0.1637     -0.86820
3             0.0464         0.3448       -0.3440     -0.40780
4            -0.8810        -0.2793       -1.0697      0.09163

   openess_to_experience
0                -0.4455
1                 0.8637
2                 0.6721
3                -0.9194
4                -0.1295
```

1.2.1 Datatypes Conversion

1. DOL - Date of Leaving. The survey was conducted back in 2015 and therefore making an assumption that the respondents who responded as **present** for DOL actually left the company within 2015 only. So, we will replace **present** value in DOL with 2024-02-17.

Then we convert the datetype of `DOJ` and `DOL` to datetime.

[ ]:
```
df1['DOL'].replace('present','2015-12-31', inplace = True)
df1['DOL'] = pd.to_datetime(df1['DOL'])
df1['DOJ'] = pd.to_datetime(df1['DOJ'])
df1.head()
```

```
<ipython-input-9-a4dd542eefee>:1: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.
```

```
  df1['DOL'].replace('present','2015-12-31', inplace = True)
```

[ ]:

	Salary	DOJ	DOL	Designation	JobCity	Gender	\
0	420000	2012-06-01	2015-12-31	senior quality engineer	Bangalore	f	
1	500000	2013-09-01	2015-12-31	assistant manager	Indore	m	
2	325000	2014-06-01	2015-12-31	systems engineer	Chennai	f	
3	1100000	2011-07-01	2015-12-31	senior software engineer	Gurgaon	m	
4	200000	2014-03-01	2015-03-01	get	Manesar	m	

	DOB	10percentage	10board	12graduation	\
0	1990-02-19	84.3	board ofsecondary education,ap	2007	
1	1989-10-04	85.4	cbse	2007	
2	1992-08-03	85.0	cbse	2010	
3	1989-12-05	85.6	cbse	2007	
4	1991-02-27	78.0	cbse	2008	

	12percentage	12board	CollegeTier	Degree	\
0	95.8	board of intermediate education,ap	2	B.Tech/B.E.	
1	85.0	cbse	2	B.Tech/B.E.	
2	68.2	cbse	2	B.Tech/B.E.	
3	83.6	cbse	1	B.Tech/B.E.	
4	76.8	cbse	2	B.Tech/B.E.	

	Specialization	collegeGPA	CollegeCityTier	\
0	computer engineering	78.00	0	
1	electronics and communication engineering	70.06	0	
2	information technology	70.00	0	
3	computer engineering	74.64	1	
4	electronics and communication engineering	73.90	0	

```
     CollegeState  GraduationYear  English  Logical  Quant    Domain  \
0  Andhra Pradesh            2011      515      585    525  0.635979
1  Madhya Pradesh            2012      695      610    780  0.960603
2   Uttar Pradesh            2014      615      545    370  0.450877
3           Delhi            2011      635      585    625  0.974396
4   Uttar Pradesh            2012      545      625    465  0.124502

   ComputerProgramming  ElectronicsAndSemicon  ComputerScience  \
0                  445                     -1               -1
1                   -1                    466               -1
2                  395                     -1               -1
3                  615                     -1               -1
4                   -1                    233               -1

   MechanicalEngg  ElectricalEngg  TelecomEngg  CivilEngg  conscientiousness  \
0              -1              -1           -1         -1             0.9737
1              -1              -1           -1         -1            -0.7335
2              -1              -1           -1         -1             0.2718
3              -1              -1           -1         -1             0.0464
4              -1              -1           -1         -1            -0.8810

   agreeableness  extraversion  nueroticism  openess_to_experience
0         0.8128        0.5269      1.35490                -0.4455
1         0.3789        1.2396     -0.10760                 0.8637
2         1.7109        0.1637     -0.86820                 0.6721
3         0.3448       -0.3440     -0.40780                -0.9194
4        -0.2793       -1.0697      0.09163                -0.1295
```

[ ]:
```
categorical = ['Designation','JobCity',␣
 ↪'Gender','10board','12board','CollegeTier','Degree',
              'Specialization','CollegeCityTier','CollegeState']
for cat in categorical:
    df1[cat] = df1[cat].astype('category')
```

[ ]:
```
df1.dtypes
```

[ ]:
```
Salary                          int64
DOJ                    datetime64[ns]
DOL                    datetime64[ns]
Designation                  category
JobCity                      category
Gender                       category
DOB                    datetime64[ns]
10percentage                  float64
10board                      category
12graduation                    int64
12percentage                  float64
```

```
12board                      category
CollegeTier                  category
Degree                       category
Specialization               category
collegeGPA                    float64
CollegeCityTier              category
CollegeState                 category
GraduationYear                  int64
English                         int64
Logical                         int64
Quant                           int64
Domain                        float64
ComputerProgramming             int64
ElectronicsAndSemicon           int64
ComputerScience                 int64
MechanicalEngg                  int64
ElectricalEngg                  int64
TelecomEngg                     int64
CivilEngg                       int64
conscientiousness             float64
agreeableness                 float64
extraversion                  float64
nueroticism                   float64
openess_to_experience         float64
dtype: object
```

2. Checking if the DOL (Date of leaving) is actually greater than DOJ (Date of joining).

[ ]:
```
dates = df1[(df1['DOL'] < df1['DOJ'])].shape[0]
print(f'DOL is earlier than DOJ for {dates} observations.')
print(df1.shape)
```

```
DOL is earlier than DOJ for 40 observations.
(3998, 35)
```

These observations might be typos and hence we will drop those 40 rows.

[ ]:
```
df1 = df1.drop(df1[~(df1['DOL'] > df1['DOJ'])].index)
print(df1.shape)
```

```
(3943, 35)
```

3. Making the entries for Gender column more descriptive

[ ]:
```
df1['Gender'].replace({'f':'Female','m':'Male'}, inplace = True)
df1.head()
```

```
<ipython-input-14-d3db9241f287>:1: FutureWarning: A value is trying to be set on
```

```
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df1['Gender'].replace({'f':'Female','m':'Male'}, inplace = True)
<ipython-input-14-d3db9241f287>:1: FutureWarning: The behavior of Series.replace
(and DataFrame.replace) with CategoricalDtype is deprecated. In a future
version, replace will only be used for cases that preserve the categories. To
change the categories, use ser.cat.rename_categories instead.
  df1['Gender'].replace({'f':'Female','m':'Male'}, inplace = True)
```

[ ]:

	Salary	DOJ	DOL	Designation	JobCity	Gender	\
0	420000	2012-06-01	2015-12-31	senior quality engineer	Bangalore	Female	
1	500000	2013-09-01	2015-12-31	assistant manager	Indore	Male	
2	325000	2014-06-01	2015-12-31	systems engineer	Chennai	Female	
3	1100000	2011-07-01	2015-12-31	senior software engineer	Gurgaon	Male	
4	200000	2014-03-01	2015-03-01	get	Manesar	Male	

	DOB	10percentage	10board	12graduation	\
0	1990-02-19	84.3	board ofsecondary education,ap	2007	
1	1989-10-04	85.4	cbse	2007	
2	1992-08-03	85.0	cbse	2010	
3	1989-12-05	85.6	cbse	2007	
4	1991-02-27	78.0	cbse	2008	

	12percentage	12board	CollegeTier	Degree	\
0	95.8	board of intermediate education,ap	2	B.Tech/B.E.	
1	85.0	cbse	2	B.Tech/B.E.	
2	68.2	cbse	2	B.Tech/B.E.	
3	83.6	cbse	1	B.Tech/B.E.	
4	76.8	cbse	2	B.Tech/B.E.	

	Specialization	collegeGPA	CollegeCityTier	\
0	computer engineering	78.00	0	
1	electronics and communication engineering	70.06	0	
2	information technology	70.00	0	
3	computer engineering	74.64	1	
4	electronics and communication engineering	73.90	0	

	CollegeState	GraduationYear	English	Logical	Quant	Domain	\

```
0  Andhra Pradesh            2011      515      585    525  0.635979
1  Madhya Pradesh            2012      695      610    780  0.960603
2   Uttar Pradesh            2014      615      545    370  0.450877
3           Delhi            2011      635      585    625  0.974396
4   Uttar Pradesh            2012      545      625    465  0.124502

   ComputerProgramming  ElectronicsAndSemicon  ComputerScience  \
0                  445                     -1               -1
1                   -1                    466               -1
2                  395                     -1               -1
3                  615                     -1               -1
4                   -1                    233               -1

   MechanicalEngg  ElectricalEngg  TelecomEngg  CivilEngg  conscientiousness  \
0              -1              -1           -1         -1             0.9737
1              -1              -1           -1         -1            -0.7335
2              -1              -1           -1         -1             0.2718
3              -1              -1           -1         -1             0.0464
4              -1              -1           -1         -1            -0.8810

   agreeableness  extraversion  nueroticism  openess_to_experience
0         0.8128        0.5269      1.35490                -0.4455
1         0.3789        1.2396     -0.10760                 0.8637
2         1.7109        0.1637     -0.86820                 0.6721
3         0.3448       -0.3440     -0.40780                -0.9194
4        -0.2793       -1.0697      0.09163                -0.1295
```

4. Validating if the results are in percentages and not in CGPA or otherwise.

[ ]:
```
print((df1['10percentage'] <=10).sum())
print((df1['12percentage'] <=10).sum())
print((df1['collegeGPA'] <=10).sum())
```

```
0
0
12
```

`10percentage` and `12percentage` are fine but `collegeGPA` has 12 obvservations which need to be deal with.

[ ]:
```
df1.loc[df1['collegeGPA']<=10,'collegeGPA'].index
```

[ ]:
```
Index([7, 138, 788, 1419, 1439, 1767, 2151, 2229, 2293, 2662, 2691, 3308],
dtype='int64')
```

[ ]:
```
df1.loc[df1['collegeGPA']<=10,'collegeGPA'] = (df1.
  ↪loc[df1['collegeGPA']<=10,'collegeGPA']/10)*100
df1.head()
```

[ ]:

```
    Salary        DOJ        DOL                Designation    JobCity  Gender  \
0   420000 2012-06-01 2015-12-31   senior quality engineer  Bangalore  Female
1   500000 2013-09-01 2015-12-31         assistant manager     Indore    Male
2   325000 2014-06-01 2015-12-31          systems engineer    Chennai  Female
3  1100000 2011-07-01 2015-12-31  senior software engineer    Gurgaon    Male
4   200000 2014-03-01 2015-03-01                       get    Manesar    Male

         DOB  10percentage                         10board  12graduation  \
0 1990-02-19          84.3  board ofsecondary education,ap          2007
1 1989-10-04          85.4                            cbse          2007
2 1992-08-03          85.0                            cbse          2010
3 1989-12-05          85.6                            cbse          2007
4 1991-02-27          78.0                            cbse          2008

   12percentage                              12board CollegeTier       Degree  \
0          95.8  board of intermediate education,ap           2  B.Tech/B.E.
1          85.0                                cbse           2  B.Tech/B.E.
2          68.2                                cbse           2  B.Tech/B.E.
3          83.6                                cbse           1  B.Tech/B.E.
4          76.8                                cbse           2  B.Tech/B.E.

                           Specialization  collegeGPA CollegeCityTier  \
0                     computer engineering       78.00               0
1  electronics and communication engineering       70.06               0
2                    information technology       70.00               0
3                     computer engineering       74.64               1
4  electronics and communication engineering       73.90               0

     CollegeState  GraduationYear  English  Logical  Quant    Domain  \
0  Andhra Pradesh            2011      515      585    525  0.635979
1  Madhya Pradesh            2012      695      610    780  0.960603
2   Uttar Pradesh            2014      615      545    370  0.450877
3           Delhi            2011      635      585    625  0.974396
4   Uttar Pradesh            2012      545      625    465  0.124502

   ComputerProgramming  ElectronicsAndSemicon  ComputerScience  \
0                  445                     -1               -1
1                   -1                    466               -1
2                  395                     -1               -1
3                  615                     -1               -1
4                   -1                    233               -1

   MechanicalEngg  ElectricalEngg  TelecomEngg  CivilEngg  conscientiousness  \
0              -1              -1           -1         -1             0.9737
1              -1              -1           -1         -1            -0.7335
2              -1              -1           -1         -1             0.2718
3              -1              -1           -1         -1             0.0464
```

```
4              -1              -1           -1         -1            -0.8810

   agreeableness  extraversion  nueroticism  openess_to_experience
0         0.8128        0.5269      1.35490                -0.4455
1         0.3789        1.2396     -0.10760                 0.8637
2         1.7109        0.1637     -0.86820                 0.6721
3         0.3448       -0.3440     -0.40780                -0.9194
4        -0.2793       -1.0697      0.09163                -0.1295
```

5. Validating if there exist 0 or -1 in the data

[ ]:
```
print((df1==0).sum()[(df1==0).sum() > 0])
```

```
10board             349
12board             358
CollegeCityTier    2761
GraduationYear        1
dtype: int64
```

[ ]:
```
(df1==-1).sum()[(df1==-1).sum()>0]/len(df1)*100
```

[ ]:
```
JobCity                 11.361907
Domain                   6.137459
ComputerProgramming     21.836165
ElectronicsAndSemicon   71.392341
ComputerScience         77.605884
MechanicalEngg          94.040071
ElectricalEngg          96.094344
TelecomEngg             90.565559
CivilEngg               98.934821
dtype: float64
```

According to the description of the columns:

1. 10board
2. 12board
3. GraduationYear
4. JobCity
5. Domain

The above columns cannot have 0 or -1 as their inputs and hence they should be considered as null values and therefore imputed

The following columns describes subjects which are optional for the exam and that is why they have large number of -1(null values). Hence we will be dropping the columns out of analysis in which the percentage for -1 values is greater than or equal to 80% and for the rest of them, we will impute the values as zero.

Sr.No.	Column Name	Null Score

Sr.No.	Column Name	Null Score
1	ElectronicsAndSemicon	71.392341
2	ComputerScience	77.605884
3	MechanicalEngg	94.040071
4	ElectricalEngg	96.094344
5	TelecomEngg	90.565559
6	CivilEngg	98.934821

[ ]:
```
df1 = df1.drop(columns = ['MechanicalEngg', 'ElectricalEngg', 'TelecomEngg',␣
↪'CivilEngg'])
df1.head()
```

[ ]:
```
    Salary        DOJ        DOL                Designation    JobCity  Gender  \
0   420000 2012-06-01 2015-12-31   senior quality engineer  Bangalore  Female
1   500000 2013-09-01 2015-12-31         assistant manager     Indore    Male
2   325000 2014-06-01 2015-12-31          systems engineer    Chennai  Female
3  1100000 2011-07-01 2015-12-31  senior software engineer    Gurgaon    Male
4   200000 2014-03-01 2015-03-01                       get    Manesar    Male

         DOB  10percentage                          10board  12graduation  \
0 1990-02-19          84.3  board ofsecondary education,ap          2007
1 1989-10-04          85.4                            cbse          2007
2 1992-08-03          85.0                            cbse          2010
3 1989-12-05          85.6                            cbse          2007
4 1991-02-27          78.0                            cbse          2008

   12percentage                              12board CollegeTier       Degree  \
0          95.8  board of intermediate education,ap           2  B.Tech/B.E.
1          85.0                                cbse           2  B.Tech/B.E.
2          68.2                                cbse           2  B.Tech/B.E.
3          83.6                                cbse           1  B.Tech/B.E.
4          76.8                                cbse           2  B.Tech/B.E.

                          Specialization  collegeGPA CollegeCityTier  \
0                    computer engineering       78.00               0
1  electronics and communication engineering       70.06               0
2                   information technology       70.00               0
3                    computer engineering       74.64               1
4  electronics and communication engineering       73.90               0

     CollegeState  GraduationYear  English  Logical  Quant    Domain  \
0  Andhra Pradesh            2011      515      585    525  0.635979
1  Madhya Pradesh            2012      695      610    780  0.960603
```

```
2   Uttar Pradesh            2014      615      545    370  0.450877
3           Delhi            2011      635      585    625  0.974396
4   Uttar Pradesh            2012      545      625    465  0.124502

   ComputerProgramming  ElectronicsAndSemicon  ComputerScience  \
0                  445                     -1               -1
1                   -1                    466               -1
2                  395                     -1               -1
3                  615                     -1               -1
4                   -1                    233               -1

   conscientiousness  agreeableness  extraversion  nueroticism  \
0             0.9737         0.8128        0.5269      1.35490
1            -0.7335         0.3789        1.2396     -0.10760
2             0.2718         1.7109        0.1637     -0.86820
3             0.0464         0.3448       -0.3440     -0.40780
4            -0.8810        -0.2793       -1.0697      0.09163

   openess_to_experience
0                -0.4455
1                 0.8637
2                 0.6721
3                -0.9194
4                -0.1295
```

[ ]:
```
df1['10board'] = df1['10board'].astype(str)
df1['12board'] = df1['12board'].astype(str)
df1['JobCity'] = df1['JobCity'].astype(str)
```

[ ]:
```
df1['10board'] = df1['10board'].replace({'0':np.nan})
df1['12board'] = df1['12board'].replace({'0':np.nan})
df1['GraduationYear'] = df1['GraduationYear'].replace({0:np.nan})
df1['JobCity'] = df1['JobCity'].replace({'-1':np.nan})
df1['Domain'] = df1['Domain'].replace({-1:np.nan})
df1['ElectronicsAndSemicon'] = df1['ElectronicsAndSemicon'].replace({-1:0})
df1['ComputerScience'] = df1['ComputerScience'].replace({-1:0})
df1['ComputerProgramming'] = df1['ComputerProgramming'].replace({-1:np.nan})
```

[ ]:
```
df1['10board'] = df1['10board'].astype('category')
df1['12board'] = df1['12board'].astype('category')
df1['JobCity'] = df1['JobCity'].astype('category')
```

[ ]:
```
df1
```

[ ]:
```
    Salary        DOJ        DOL                  Designation  \
0   420000 2012-06-01 2015-12-31      senior quality engineer
1   500000 2013-09-01 2015-12-31            assistant manager
```

```
2      325000 2014-06-01 2015-12-31             systems engineer
3     1100000 2011-07-01 2015-12-31     senior software engineer
4      200000 2014-03-01 2015-03-01                         get
…         …          …          …                          …
3992   800000 2014-04-01 2015-04-01                     manager
3993   280000 2011-10-01 2012-10-01           software engineer
3995   320000 2013-07-01 2015-12-31  associate software engineer
3996   200000 2014-07-01 2015-01-01           software developer
3997   400000 2013-02-01 2015-12-31      senior systems engineer

               JobCity  Gender        DOB  10percentage  \
0            Bangalore  Female 1990-02-19         84.30
1               Indore    Male 1989-10-04         85.40
2              Chennai  Female 1992-08-03         85.00
3              Gurgaon    Male 1989-12-05         85.60
4              Manesar    Male 1991-02-27         78.00
…                  …     …         …             …
3992            Rajkot    Male 1990-06-22         73.00
3993         New Delhi     Male 1987-04-15         52.09
3995          Bangalore    Male 1991-07-03         81.86
3996  Asifabadbanglore  Female 1992-03-20         78.72
3997            Chennai  Female 1991-02-26         70.60

                             10board  12graduation  12percentage  \
0     board ofsecondary education,ap          2007         95.80
1                               cbse          2007         85.00
2                               cbse          2010         68.20
3                               cbse          2007         83.60
4                               cbse          2008         76.80
…                                …             …            …
3992                             NaN          2008         54.00
3993                            cbse          2006         55.50
3995                    bse,odisha            2008         65.50
3996                   state board            2010         69.88
3997                            cbse          2008         68.00

                                  12board CollegeTier       Degree  \
0     board of intermediate education,ap           2  B.Tech/B.E.
1                                   cbse           2  B.Tech/B.E.
2                                   cbse           2  B.Tech/B.E.
3                                   cbse           1  B.Tech/B.E.
4                                   cbse           2  B.Tech/B.E.
…                                    …           …          …
3992                                NaN            2  B.Tech/B.E.
3993                                cbse           2  B.Tech/B.E.
3995                           chse,odisha         2  B.Tech/B.E.
3996                           state board         2  B.Tech/B.E.
```

```
3997                              cbse           2  B.Tech/B.E.

                              Specialization  collegeGPA CollegeCityTier  \
0                          computer engineering       78.00               0
1     electronics and communication engineering       70.06               0
2                        information technology       70.00               0
3                          computer engineering       74.64               1
4     electronics and communication engineering       73.90               0
…                                          …         …              …
3992                           civil engineering       79.00               0
3993                     information technology       61.50               0
3995                       computer engineering       70.00               0
3996              computer science & engineering       70.42               1
3997                     information technology       68.00               1

        CollegeState  GraduationYear  English  Logical  Quant    Domain  \
0     Andhra Pradesh          2011.0      515      585    525  0.635979
1     Madhya Pradesh          2012.0      695      610    780  0.960603
2      Uttar Pradesh          2014.0      615      545    370  0.450877
3              Delhi          2011.0      635      585    625  0.974396
4      Uttar Pradesh          2012.0      545      625    465  0.124502
…                 …             …      …        …      …        …
3992          Orissa          2012.0      405      345    525  0.938588
3993         Haryana          2010.0      365      334    475  0.276047
3995          Orissa          2012.0      475      475    465  0.488348
3996       Karnataka          2014.0      450      410    320  0.744758
3997      Tamil Nadu          2012.0      565      515    464  0.600057

      ComputerProgramming  ElectronicsAndSemicon  ComputerScience  \
0                   445.0                      0                0
1                     NaN                    466                0
2                   395.0                      0                0
3                   615.0                      0                0
4                     NaN                    233                0
…                     …                      …                …
3992                  NaN                      0                0
3993                345.0                      0                0
3995                405.0                      0                0
3996                445.0                      0              438
3997                435.0                      0                0

      conscientiousness  agreeableness  extraversion  nueroticism  \
0                0.9737         0.8128        0.5269      1.35490
1               -0.7335         0.3789        1.2396     -0.10760
2                0.2718         1.7109        0.1637     -0.86820
3                0.0464         0.3448       -0.3440     -0.40780
4               -0.8810        -0.2793       -1.0697      0.09163
```

```
…                 …             …             …           …
3992             0.3555       -0.9033        0.9623      0.64983
3993            -0.1082        0.3448        0.2366      0.64980
3995            -1.5765       -1.5273       -1.5051     -1.31840
3996            -0.1590        0.0459       -0.4511     -0.36120
3997            -1.1128       -0.2793       -0.6343      1.32553

      openess_to_experience
0                   -0.4455
1                    0.8637
2                    0.6721
3                   -0.9194
4                   -0.1295
…                     …
3992                -0.4229
3993                -0.9194
3995                -0.7615
3996                -0.0943
3997                -0.6035

[3943 rows x 31 columns]
```

Imputing categorical columns with mode values for their respective columns.

[ ]:
```
df1['10board'].fillna(df1['10board'].mode()[0], inplace = True)
df1['12board'].fillna(df1['12board'].mode()[0], inplace = True)
df1['GraduationYear'].fillna(df1['GraduationYear'].mode()[0], inplace = True)
df1['JobCity'].fillna(df1['JobCity'].mode()[0], inplace = True)

df1
```

```
<ipython-input-25-49b0504dbf6c>:1: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df1['10board'].fillna(df1['10board'].mode()[0], inplace = True)
<ipython-input-25-49b0504dbf6c>:2: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
```

```
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df1['12board'].fillna(df1['12board'].mode()[0], inplace = True)
<ipython-input-25-49b0504dbf6c>:3: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df1['GraduationYear'].fillna(df1['GraduationYear'].mode()[0], inplace = True)
<ipython-input-25-49b0504dbf6c>:4: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df1['JobCity'].fillna(df1['JobCity'].mode()[0], inplace = True)
```

[ ]:

	Salary	DOJ	DOL	Designation	\
0	420000	2012-06-01	2015-12-31	senior quality engineer	
1	500000	2013-09-01	2015-12-31	assistant manager	
2	325000	2014-06-01	2015-12-31	systems engineer	
3	1100000	2011-07-01	2015-12-31	senior software engineer	
4	200000	2014-03-01	2015-03-01	get	
…	…	…	…	…	
3992	800000	2014-04-01	2015-04-01	manager	
3993	280000	2011-10-01	2012-10-01	software engineer	
3995	320000	2013-07-01	2015-12-31	associate software engineer	
3996	200000	2014-07-01	2015-01-01	software developer	

```
3997   400000 2013-02-01 2015-12-31      senior systems engineer

               JobCity  Gender        DOB  10percentage  \
0            Bangalore  Female 1990-02-19         84.30
1               Indore    Male 1989-10-04         85.40
2              Chennai  Female 1992-08-03         85.00
3              Gurgaon    Male 1989-12-05         85.60
4              Manesar    Male 1991-02-27         78.00
…                  …     …         …            …
3992            Rajkot    Male 1990-06-22         73.00
3993         New Delhi    Male 1987-04-15         52.09
3995          Bangalore    Male 1991-07-03         81.86
3996  Asifabadbanglore  Female 1992-03-20         78.72
3997            Chennai  Female 1991-02-26         70.60

                             10board  12graduation  12percentage  \
0     board ofsecondary education,ap          2007         95.80
1                               cbse          2007         85.00
2                               cbse          2010         68.20
3                               cbse          2007         83.60
4                               cbse          2008         76.80
…                               …             …            …
3992                            cbse          2008         54.00
3993                            cbse          2006         55.50
3995                     bse,odisha           2008         65.50
3996                    state board           2010         69.88
3997                            cbse          2008         68.00

                                12board CollegeTier       Degree  \
0     board of intermediate education,ap           2  B.Tech/B.E.
1                                   cbse           2  B.Tech/B.E.
2                                   cbse           2  B.Tech/B.E.
3                                   cbse           1  B.Tech/B.E.
4                                   cbse           2  B.Tech/B.E.
…                                   …              …          …
3992                                cbse           2  B.Tech/B.E.
3993                                cbse           2  B.Tech/B.E.
3995                           chse,odisha         2  B.Tech/B.E.
3996                           state board         2  B.Tech/B.E.
3997                                cbse           2  B.Tech/B.E.

                                  Specialization  collegeGPA CollegeCityTier  \
0                           computer engineering       78.00               0
1     electronics and communication engineering       70.06               0
2                         information technology       70.00               0
3                           computer engineering       74.64               1
4     electronics and communication engineering       73.90               0
```

```
…                                        …         …              …
3992                         civil engineering       79.00              0
3993                     information technology       61.50              0
3995                       computer engineering       70.00              0
3996             computer science & engineering       70.42              1
3997                     information technology       68.00              1

        CollegeState  GraduationYear  English  Logical  Quant    Domain  \
0     Andhra Pradesh          2011.0      515      585    525  0.635979
1     Madhya Pradesh          2012.0      695      610    780  0.960603
2      Uttar Pradesh          2014.0      615      545    370  0.450877
3              Delhi          2011.0      635      585    625  0.974396
4      Uttar Pradesh          2012.0      545      625    465  0.124502
…                 …             …        …        …      …         …
3992          Orissa          2012.0      405      345    525  0.938588
3993         Haryana          2010.0      365      334    475  0.276047
3995          Orissa          2012.0      475      475    465  0.488348
3996       Karnataka          2014.0      450      410    320  0.744758
3997      Tamil Nadu          2012.0      565      515    464  0.600057

      ComputerProgramming  ElectronicsAndSemicon  ComputerScience  \
0                   445.0                      0                0
1                     NaN                    466                0
2                   395.0                      0                0
3                   615.0                      0                0
4                     NaN                    233                0
…                     …                      …              …
3992                  NaN                      0                0
3993                345.0                      0                0
3995                405.0                      0                0
3996                445.0                      0              438
3997                435.0                      0                0

      conscientiousness  agreeableness  extraversion  nueroticism  \
0                0.9737         0.8128        0.5269      1.35490
1               -0.7335         0.3789        1.2396     -0.10760
2                0.2718         1.7109        0.1637     -0.86820
3                0.0464         0.3448       -0.3440     -0.40780
4               -0.8810        -0.2793       -1.0697      0.09163
…                 …             …             …          …
3992             0.3555        -0.9033        0.9623      0.64983
3993            -0.1082         0.3448        0.2366      0.64980
3995            -1.5765        -1.5273       -1.5051     -1.31840
3996            -0.1590         0.0459       -0.4511     -0.36120
3997            -1.1128        -0.2793       -0.6343      1.32553

      openess_to_experience
```

```
0                   -0.4455
1                    0.8637
2                    0.6721
3                   -0.9194
4                   -0.1295
…                     …
3992                -0.4229
3993                -0.9194
3995                -0.7615
3996                -0.0943
3997                -0.6035

[3943 rows x 31 columns]
```

Imputing the numerical columns with median values for their respective columns.

[ ]:
```
df1['Domain'].fillna(df1['Domain'].median(), inplace = True)
df1['ComputerProgramming'].fillna(df1['ComputerProgramming'].median(), inplace␣
↪= True)
df1.head()
```

```
<ipython-input-26-880c50f4b894>:1: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df1['Domain'].fillna(df1['Domain'].median(), inplace = True)
<ipython-input-26-880c50f4b894>:2: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df1['ComputerProgramming'].fillna(df1['ComputerProgramming'].median(), inplace
= True)
```

[ ]:

	Salary	DOJ	DOL	Designation	JobCity	Gender
0	420000	2012-06-01	2015-12-31	senior quality engineer	Bangalore	Female
1	500000	2013-09-01	2015-12-31	assistant manager	Indore	Male
2	325000	2014-06-01	2015-12-31	systems engineer	Chennai	Female
3	1100000	2011-07-01	2015-12-31	senior software engineer	Gurgaon	Male
4	200000	2014-03-01	2015-03-01	get	Manesar	Male

	DOB	10percentage	10board	12graduation
0	1990-02-19	84.3	board ofsecondary education,ap	2007
1	1989-10-04	85.4	cbse	2007
2	1992-08-03	85.0	cbse	2010
3	1989-12-05	85.6	cbse	2007
4	1991-02-27	78.0	cbse	2008

	12percentage	12board	CollegeTier	Degree
0	95.8	board of intermediate education,ap	2	B.Tech/B.E.
1	85.0	cbse	2	B.Tech/B.E.
2	68.2	cbse	2	B.Tech/B.E.
3	83.6	cbse	1	B.Tech/B.E.
4	76.8	cbse	2	B.Tech/B.E.

	Specialization	collegeGPA	CollegeCityTier
0	computer engineering	78.00	0
1	electronics and communication engineering	70.06	0
2	information technology	70.00	0
3	computer engineering	74.64	1
4	electronics and communication engineering	73.90	0

	CollegeState	GraduationYear	English	Logical	Quant	Domain
0	Andhra Pradesh	2011.0	515	585	525	0.635979
1	Madhya Pradesh	2012.0	695	610	780	0.960603
2	Uttar Pradesh	2014.0	615	545	370	0.450877
3	Delhi	2011.0	635	585	625	0.974396
4	Uttar Pradesh	2012.0	545	625	465	0.124502

	ComputerProgramming	ElectronicsAndSemicon	ComputerScience
0	445.0	0	0
1	455.0	466	0
2	395.0	0	0
3	615.0	0	0
4	455.0	233	0

	conscientiousness	agreeableness	extraversion	nueroticism
0	0.9737	0.8128	0.5269	1.35490
1	-0.7335	0.3789	1.2396	-0.10760
2	0.2718	1.7109	0.1637	-0.86820
3	0.0464	0.3448	-0.3440	-0.40780

```
4            -0.8810        -0.2793        -1.0697       0.09163

   openess_to_experience
0                -0.4455
1                 0.8637
2                 0.6721
3                -0.9194
4                -0.1295
```

6. Correcting string data in columns

[ ]:
```
def correct_string_data(data):
    '''
    Convert the textual categories to lower case
    and remove the leading or trailing spaces if any.

    '''
    df1[data] = df1[data].str.lower().str.strip()
```

[ ]:
```
textual_columns =␣
 ↪['Designation','JobCity','10board','12board','Specialization','CollegeState']
```

[ ]:
```
for col in textual_columns:
    print(f'Number of unique values in {col} with inconsistency : {df1[col].
 ↪nunique()}')
```

```
Number of unique values in Designation with inconsistency : 416
Number of unique values in JobCity with inconsistency : 337
Number of unique values in 10board with inconsistency : 274
Number of unique values in 12board with inconsistency : 339
Number of unique values in Specialization with inconsistency : 46
Number of unique values in CollegeState with inconsistency : 26
```

[ ]:
```
for col in textual_columns:
    correct_string_data(col)
```

[ ]:
```
for col in textual_columns:
    print(f'Number of unique values in {col} without inconsistency : {df1[col].
 ↪nunique()}')
```

```
Number of unique values in Designation without inconsistency : 416
Number of unique values in JobCity without inconsistency : 230
Number of unique values in 10board without inconsistency : 272
Number of unique values in 12board without inconsistency : 336
Number of unique values in Specialization without inconsistency : 46
Number of unique values in CollegeState without inconsistency : 26
```

Since the number of categories are large enough to deal with, we keep the top 10

categories.

1.2.2 Collapsing Categories

Keeping only the top 10 frequent categories and classifying others as **other**.

[ ]:
```
def collapsing_categories(df1, data):
    for Designation in df1[data].unique():
        min_count = df1[data].value_counts()[:10].min()
        if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
            df1.loc[df1[data] == Designation, data] = 'other'
```

[ ]:
```
for cols in textual_columns:
    collapsing_categories(df1, cols)
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
```

```
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

```
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
```

```
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
```

```
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
<ipython-input-32-455db4d5b38a>:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  if df1[df1[data] == Designation][data].value_counts()[0] < min_count:
```

[ ]:
```
for cols in textual_columns:
    print('')
    print('Top 10 categories in:', cols)
    print('')
    print(df1[cols].value_counts())
    print('')
    print('*'*100)
```

```
Top 10 categories in: Designation

Designation
other                          2259
software engineer               535
software developer              262
system engineer                 202
programmer analyst              139
systems engineer                117
java software engineer          109
software test engineer          100
project engineer                 76
technical support engineer       73
senior software engineer         71
Name: count, dtype: int64

********************************************************************************
```

```
********************

Top 10 categories in: JobCity

JobCity
bangalore    1109
other         807
noida         382
hyderabad     361
pune          322
chennai       310
gurgaon       212
new delhi     203
mumbai        119
kolkata       118
Name: count, dtype: int64

********************************************************************************
********************

Top 10 categories in: 10board

10board
cbse                           1726
state board                    1140
other                           498
icse                            276
ssc                             121
up board                         85
matriculation                    38
rbse                             21
board of secondary education     20
up                               18
Name: count, dtype: int64

********************************************************************************
********************

Top 10 categories in: 12board

12board
cbse                             1737
state board                      1229
other                             595
icse                              128
up board                           87
isc                                45
board of intermediate              38
```

```
board of intermediate education      31
up                                   19
mp board                             17
rbse                                 17
Name: count, dtype: int64

**********************************************************************
********************

Top 10 categories in: Specialization

Specialization
electronics and communication engineering    865
computer science & engineering               731
information technology                       654
computer engineering                         593
other                                        268
computer application                         241
mechanical engineering                       201
electronics and electrical engineering       191
electronics & telecommunications             120
electrical engineering                        79
Name: count, dtype: int64

**********************************************************************
********************

Top 10 categories in: CollegeState

CollegeState
uttar pradesh     902
other             772
karnataka         369
tamil nadu        363
telangana         312
maharashtra       257
andhra pradesh    222
west bengal       192
madhya pradesh    189
punjab            188
haryana           177
Name: count, dtype: int64

**********************************************************************
********************
```

[ ]:
```
df1
```

```
[ ]:        Salary        DOJ        DOL               Designation    JobCity  \
     0      420000 2012-06-01 2015-12-31                     other  bangalore
     1      500000 2013-09-01 2015-12-31                     other      other
     2      325000 2014-06-01 2015-12-31          systems engineer    chennai
     3     1100000 2011-07-01 2015-12-31  senior software engineer    gurgaon
     4      200000 2014-03-01 2015-03-01                     other      other
     …       …        …         …                          …        …
     3992   800000 2014-04-01 2015-04-01                     other      other
     3993   280000 2011-10-01 2012-10-01         software engineer  new delhi
     3995   320000 2013-07-01 2015-12-31                     other  bangalore
     3996   200000 2014-07-01 2015-01-01        software developer      other
     3997   400000 2013-02-01 2015-12-31                     other    chennai

           Gender        DOB  10percentage      10board  12graduation  \
     0     Female 1990-02-19         84.30        other          2007
     1       Male 1989-10-04         85.40         cbse          2007
     2     Female 1992-08-03         85.00         cbse          2010
     3       Male 1989-12-05         85.60         cbse          2007
     4       Male 1991-02-27         78.00         cbse          2008
     …      …        …            …           …            …
     3992    Male 1990-06-22         73.00         cbse          2008
     3993    Male 1987-04-15         52.09         cbse          2006
     3995    Male 1991-07-03         81.86        other          2008
     3996  Female 1992-03-20         78.72  state board          2010
     3997  Female 1991-02-26         70.60         cbse          2008

           12percentage      12board CollegeTier       Degree  \
     0            95.80        other           2  B.Tech/B.E.
     1            85.00         cbse           2  B.Tech/B.E.
     2            68.20         cbse           2  B.Tech/B.E.
     3            83.60         cbse           1  B.Tech/B.E.
     4            76.80         cbse           2  B.Tech/B.E.
     …            …           …          …           …
     3992         54.00         cbse           2  B.Tech/B.E.
     3993         55.50         cbse           2  B.Tech/B.E.
     3995         65.50        other           2  B.Tech/B.E.
     3996         69.88  state board           2  B.Tech/B.E.
     3997         68.00         cbse           2  B.Tech/B.E.

                                    Specialization  collegeGPA CollegeCityTier  \
     0                          computer engineering       78.00               0
     1     electronics and communication engineering       70.06               0
     2                        information technology       70.00               0
     3                          computer engineering       74.64               1
     4     electronics and communication engineering       73.90               0
     …                                          …         …             …
     3992                                   other       79.00               0
```

```
3993                    information technology       61.50              0
3995                      computer engineering       70.00              0
3996             computer science & engineering       70.42              1
3997                    information technology       68.00              1

        CollegeState  GraduationYear  English  Logical  Quant    Domain  \
0     andhra pradesh          2011.0      515      585    525  0.635979
1     madhya pradesh          2012.0      695      610    780  0.960603
2      uttar pradesh          2014.0      615      545    370  0.450877
3              other          2011.0      635      585    625  0.974396
4      uttar pradesh          2012.0      545      625    465  0.124502
…                …               …      …        …      …        …
3992           other          2012.0      405      345    525  0.938588
3993         haryana          2010.0      365      334    475  0.276047
3995           other          2012.0      475      475    465  0.488348
3996       karnataka          2014.0      450      410    320  0.744758
3997      tamil nadu          2012.0      565      515    464  0.600057

      ComputerProgramming  ElectronicsAndSemicon  ComputerScience  \
0                   445.0                      0                0
1                   455.0                    466                0
2                   395.0                      0                0
3                   615.0                      0                0
4                   455.0                    233                0
…                   …                        …                …
3992                455.0                      0                0
3993                345.0                      0                0
3995                405.0                      0                0
3996                445.0                      0              438
3997                435.0                      0                0

      conscientiousness  agreeableness  extraversion  nueroticism  \
0                0.9737         0.8128        0.5269      1.35490
1               -0.7335         0.3789        1.2396     -0.10760
2                0.2718         1.7109        0.1637     -0.86820
3                0.0464         0.3448       -0.3440     -0.40780
4               -0.8810        -0.2793       -1.0697      0.09163
…                 …             …             …           …
3992             0.3555        -0.9033        0.9623      0.64983
3993            -0.1082         0.3448        0.2366      0.64980
3995            -1.5765        -1.5273       -1.5051     -1.31840
3996            -0.1590         0.0459       -0.4511     -0.36120
3997            -1.1128        -0.2793       -0.6343      1.32553

      openess_to_experience
0                   -0.4455
1                    0.8637
```

```
2                    0.6721
3                   -0.9194
4                   -0.1295
…                      …
3992                -0.4229
3993                -0.9194
3995                -0.7615
3996                -0.0943
3997                -0.6035

[3943 rows x 31 columns]
```

1.2.3 Feature Engineering

1. Since the dataset was release in 2015, we add a age column by subtracting DOB year from 2015. This will add the age as of 2015.

[ ]:
```
df1['DOB'] = pd.to_datetime(df1['DOB'])
df1['Age'] = 2015 - df1['DOB'].dt.year
df1.head()
```

[ ]:
```
    Salary        DOJ        DOL               Designation    JobCity  Gender  \
0   420000 2012-06-01 2015-12-31                     other  bangalore  Female
1   500000 2013-09-01 2015-12-31                     other      other    Male
2   325000 2014-06-01 2015-12-31          systems engineer    chennai  Female
3  1100000 2011-07-01 2015-12-31  senior software engineer    gurgaon    Male
4   200000 2014-03-01 2015-03-01                     other      other    Male

         DOB  10percentage 10board  12graduation  12percentage 12board  \
0 1990-02-19          84.3   other          2007          95.8   other
1 1989-10-04          85.4    cbse          2007          85.0    cbse
2 1992-08-03          85.0    cbse          2010          68.2    cbse
3 1989-12-05          85.6    cbse          2007          83.6    cbse
4 1991-02-27          78.0    cbse          2008          76.8    cbse

  CollegeTier       Degree                                Specialization  \
0           2  B.Tech/B.E.                          computer engineering
1           2  B.Tech/B.E.  electronics and communication engineering
2           2  B.Tech/B.E.                       information technology
3           1  B.Tech/B.E.                          computer engineering
4           2  B.Tech/B.E.  electronics and communication engineering

   collegeGPA CollegeCityTier    CollegeState  GraduationYear  English  \
0       78.00               0  andhra pradesh          2011.0      515
1       70.06               0  madhya pradesh          2012.0      695
2       70.00               0   uttar pradesh          2014.0      615
3       74.64               1           other          2011.0      635
4       73.90               0   uttar pradesh          2012.0      545
```

```
   Logical  Quant    Domain  ComputerProgramming  ElectronicsAndSemicon  \
0      585    525  0.635979                445.0                      0
1      610    780  0.960603                455.0                    466
2      545    370  0.450877                395.0                      0
3      585    625  0.974396                615.0                      0
4      625    465  0.124502                455.0                    233

   ComputerScience  conscientiousness  agreeableness  extraversion  \
0                0             0.9737         0.8128        0.5269
1                0            -0.7335         0.3789        1.2396
2                0             0.2718         1.7109        0.1637
3                0             0.0464         0.3448       -0.3440
4                0            -0.8810        -0.2793       -1.0697

   nueroticism  openess_to_experience  Age
0      1.35490                -0.4455   25
1     -0.10760                 0.8637   26
2     -0.86820                 0.6721   23
3     -0.40780                -0.9194   26
4      0.09163                -0.1295   24
```

2. Adding a tenure column by subtracting the DOL from DOJ

[ ]:
```
delta = (df1['DOL'] - df1['DOJ'])
tenure = np.zeros(len(df1))
for i, date in enumerate(delta):
    tenure[i] = round(date.days/365,2)
df1['Tenure'] = tenure
```

3. Dropping the rows where the graduationyear is greater than or equal to date of joining

[ ]:
```
len(df1[(df1['GraduationYear'] > df1['DOJ'].dt.year)].index)
```

[ ]:
```
79
```

[ ]:
```
df1 = df1.drop(df1[(df1['GraduationYear'] > df1['DOJ'].dt.year)].index)
```

4. Function to calculate CDF

[ ]:
```
def cdf(data):
    x = np.sort(data)
    y = np.arange(1, len(x)+1)/len(x)
    return x, y
```

1.3 Univariate Analysis

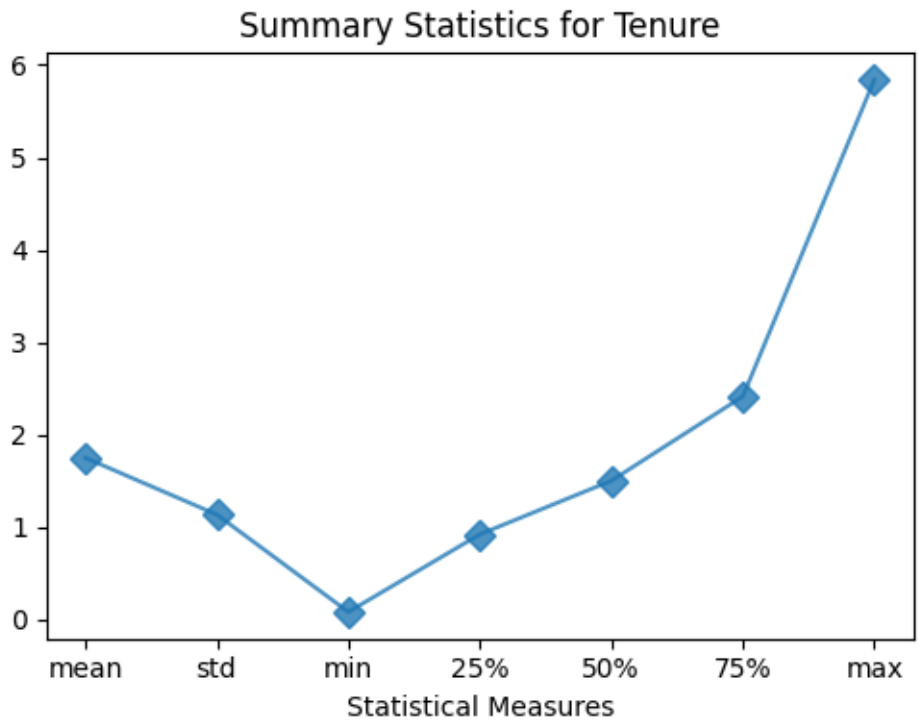
1.3.1 1. Continuous Features

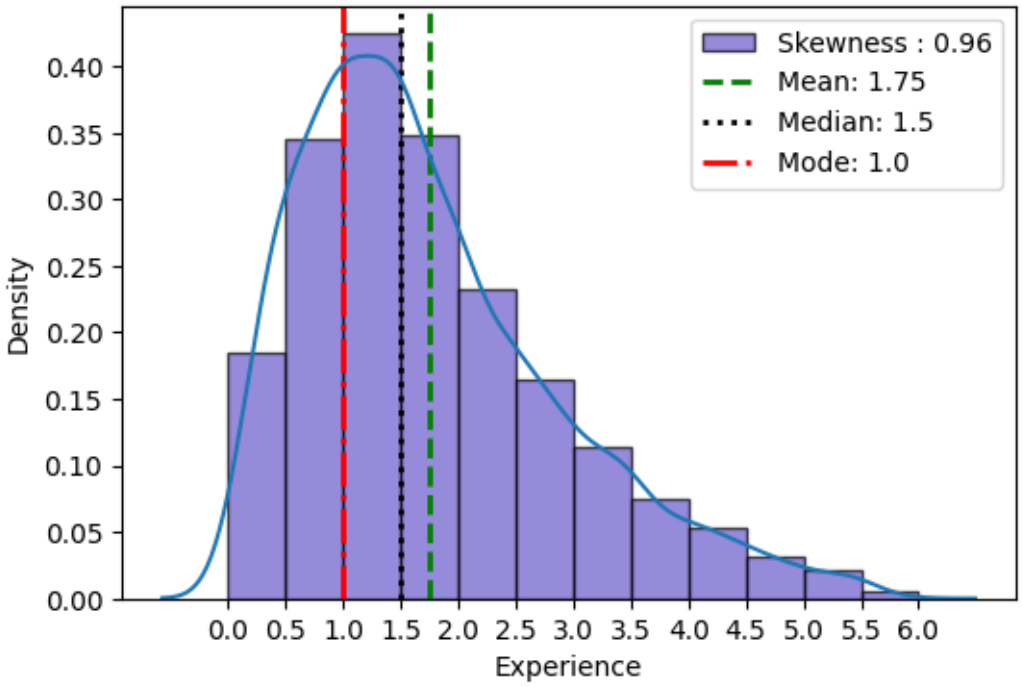
[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
%matplotlib inline
```

1.1 Tenure

[ ]:
```
# Summary Plot

plt.figure(figsize=(5, 4))
df1['Tenure'].describe()[1:].plot(alpha = 0.8,
                                  marker = 'D', markersize = 8)
plt.title('Summary Statistics for Tenure')
plt.xlabel('Statistical Measures')
plt.tight_layout()
plt.show()

# Histogram

plt.figure(figsize = (6,4))
plt.hist(df1['Tenure'],
        ec = 'k',
        bins = np.arange(0, df1['Tenure'].max()+0.5, 0.5),
        color = 'slateblue',
        alpha = 0.7,
        label = f"Skewness : {round(df1['Tenure'].skew(),2)}",
        density = True)
plt.xticks(ticks = np.arange(0, df1['Tenure'].max()+0.5, 0.5))
plt.xlabel('Experience')
plt.ylabel('Density')
plt.axvline(df1['Tenure'].mean(), label = f"Mean: {round(df1['Tenure'].
 ↪mean(),2)}",
           linestyle = '--',
          color = 'green', linewidth = 2)
plt.axvline(df1['Tenure'].median(), label = f"Median: {round(df1['Tenure'].
 ↪median(),2)}",
           linestyle = ':',
          color = 'k', linewidth = 2)
plt.axvline(df1['Tenure'].mode()[0], label = f"Mode: {round(df1['Tenure'].
 ↪mode()[0],2)}"
           , linestyle = '-.',
          color = 'red', linewidth = 2)
sns.kdeplot(df1['Tenure'])
plt.legend()
```

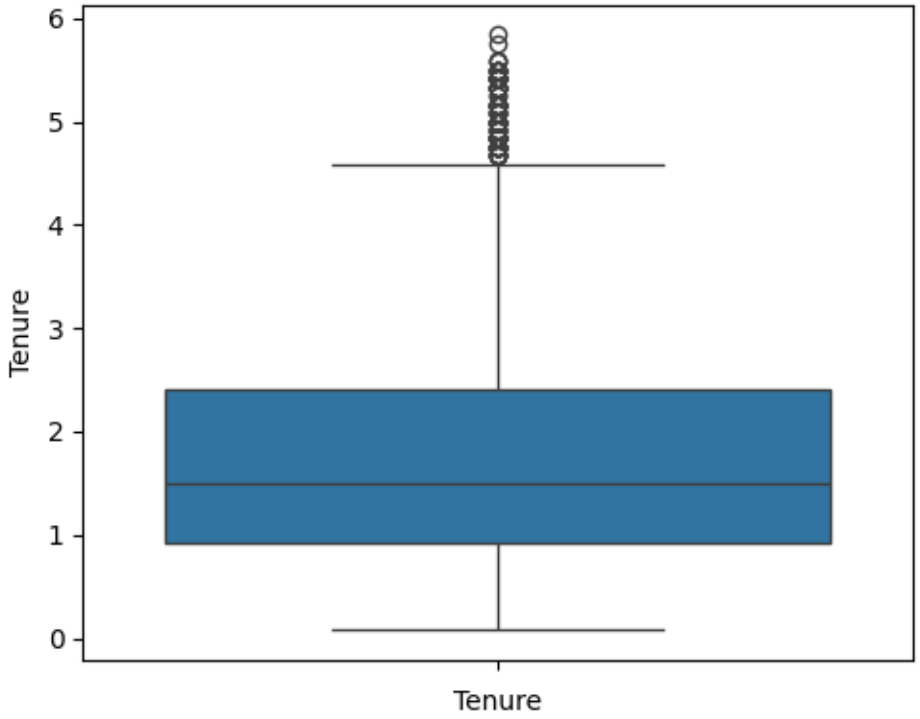
```
plt.show()

# Box Plot

plt.figure(figsize=(5, 4))
sns.boxplot(df1['Tenure'])
plt.xlabel('Tenure')
plt.tight_layout()
plt.show()

# CDF

plt.figure(figsize=(5, 4))
x_tenure, y_tenure = cdf(df1['Tenure'])
x_sample_tenure, y_sample_tenure = cdf(np.random.normal(df1['Tenure'].mean(),␣
 ↪df1['Tenure'].std(), size = len(df1['Tenure'])))
plt.plot(x_tenure, y_tenure, linestyle = 'None',
        marker = '.', color = 'orange',
         alpha = 0.7, label = 'Tenure')
plt.plot(x_sample_tenure, y_sample_tenure, linestyle = 'None',
        marker ='.', color = 'red',
        alpha = 0.7, label = 'Normal Distribution')
plt.xlabel('Tenure')
plt.ylabel('CDF')
plt.legend()
plt.tight_layout()
plt.show()
```

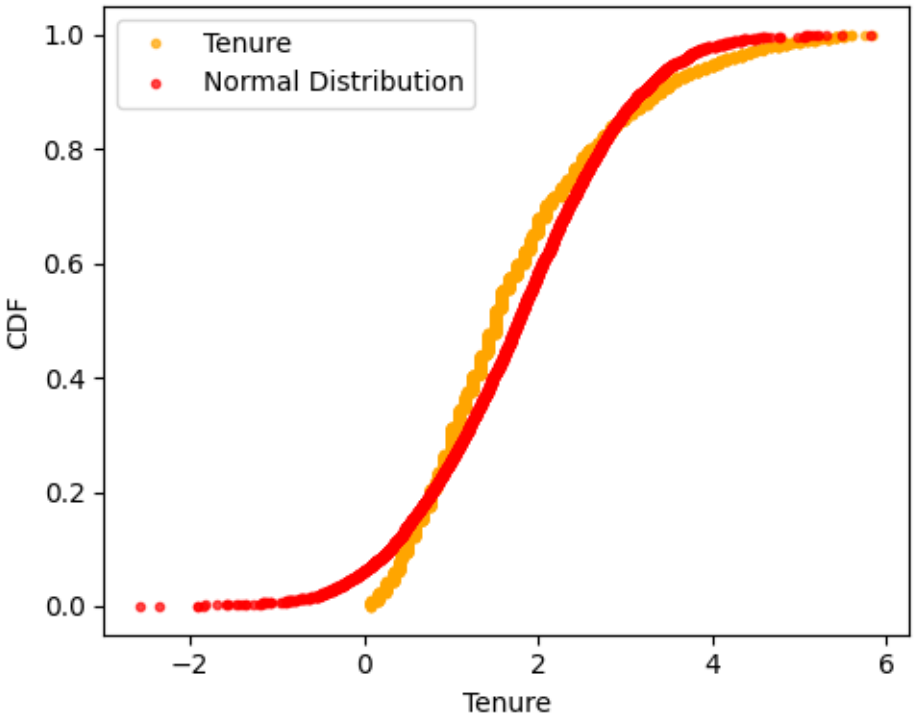





```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:640:
```

```
FutureWarning: SeriesGroupBy.grouper is deprecated and will be removed in a
future version of pandas.
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```





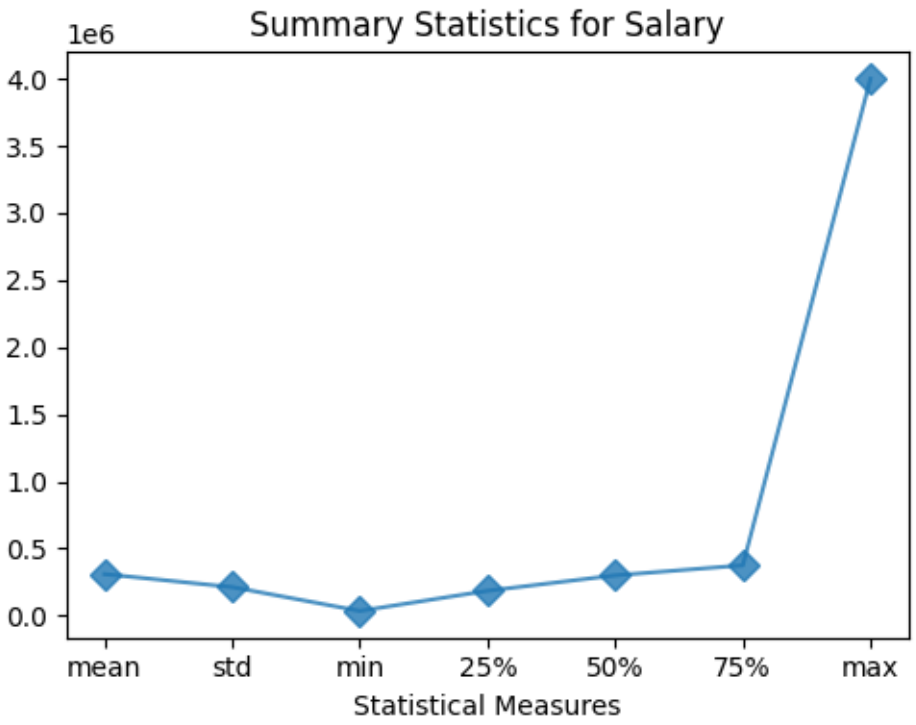
Sr.No.	Conclusion	Inferences
1.	Summary Plot	- The range for experience is 4 years.
2.	Histogram	- The data is positively skewed i.e there exists larger number of respondents with low tenure, 50% data points are below 1.5 years, Average tenure is 1.5 years, The mean, median, and mode lie very close to each other and skewness (0.6) is close to that of a normal (0).
3.	Box Plot	- There are few values with large tenure i.e outliers
4.	CDF	- The data is not normally distributed, We can say that tenure is not normally distributed.

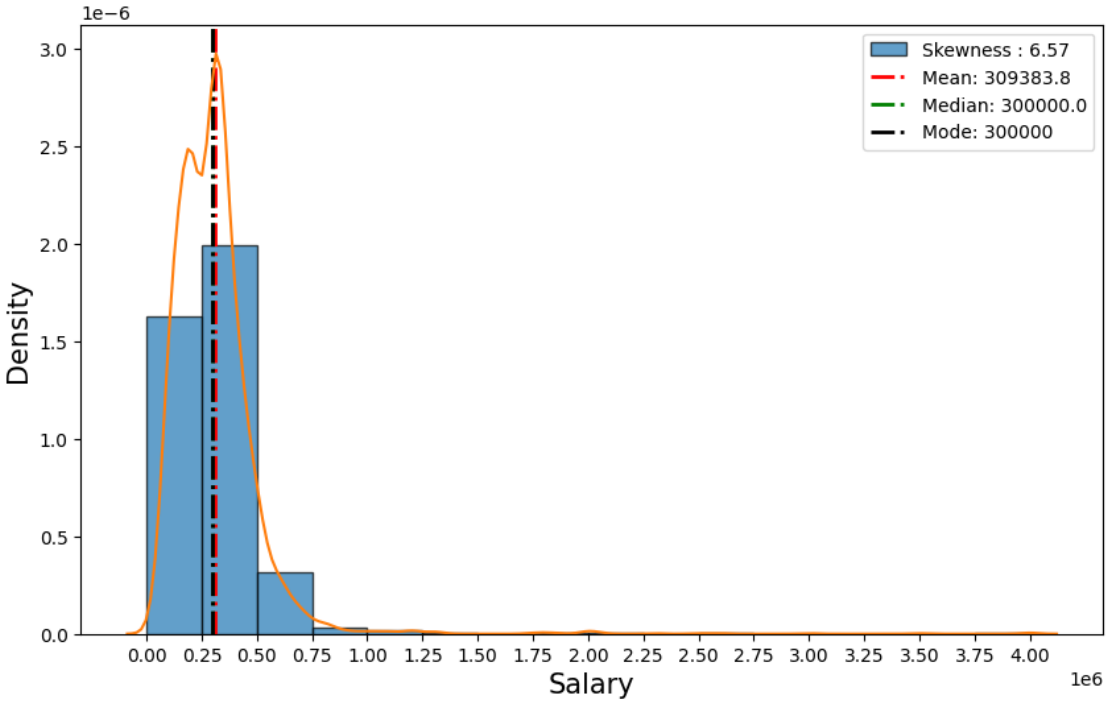
Observations

1.2 Salary

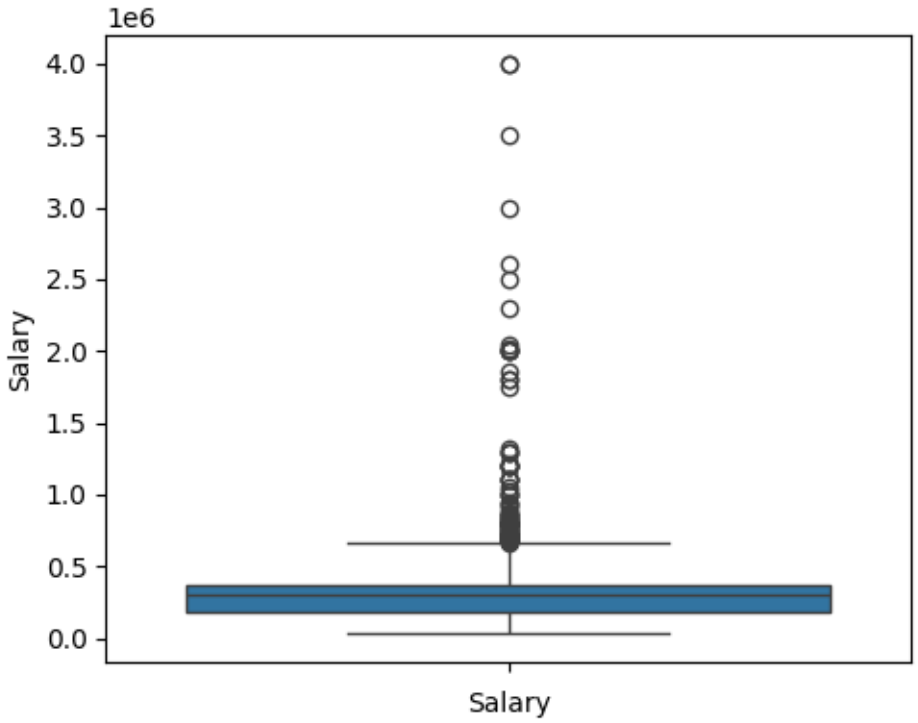
[ ]:
```
# Summary Plot

plt.figure(figsize=(5,4))
df1['Salary'].describe()[1:].plot(alpha = 0.8,
                                  marker = 'D', markersize = 8)
plt.title('Summary Statistics for Salary')
plt.xlabel('Statistical Measures')
plt.tight_layout()
plt.show()

# Histogram

bins = np.arange(0, df1['Salary'].max()+250000, 250000)
plt.figure(figsize = (10,6))
plt.hist(df1['Salary'], ec = 'k',
        bins = bins,
        label = f"Skewness : {round(df1['Salary'].skew(),2)}",
        alpha = 0.7,
        density = True)
plt.xticks(bins)
plt.xlabel('Salary', size = 15)
plt.ylabel('Density', size = 15)

plt.axvline(df1['Salary'].mean(), label = f"Mean: {round(df1['Salary'].
 ↪mean(),2)}"
            , linestyle = '-.',
           color = 'red', linewidth = 2)
plt.axvline(df1['Salary'].median(), label = f"Median: {round(df1['Salary'].
 ↪median(),2)}"
            , linestyle = '-.',
           color = 'green', linewidth = 2)
plt.axvline(df1['Salary'].mode()[0], label = f"Mode: {round(df1['Salary'].
 ↪mode()[0],2)}"
            , linestyle = '-.',
           color = 'k', linewidth = 2)
sns.kdeplot(df1['Salary'])
plt.legend()
plt.show()

# Box Plot

plt.figure(figsize=(5,4))
sns.boxplot(df1['Salary'])
plt.xlabel('Salary')
plt.tight_layout()
plt.show()
```

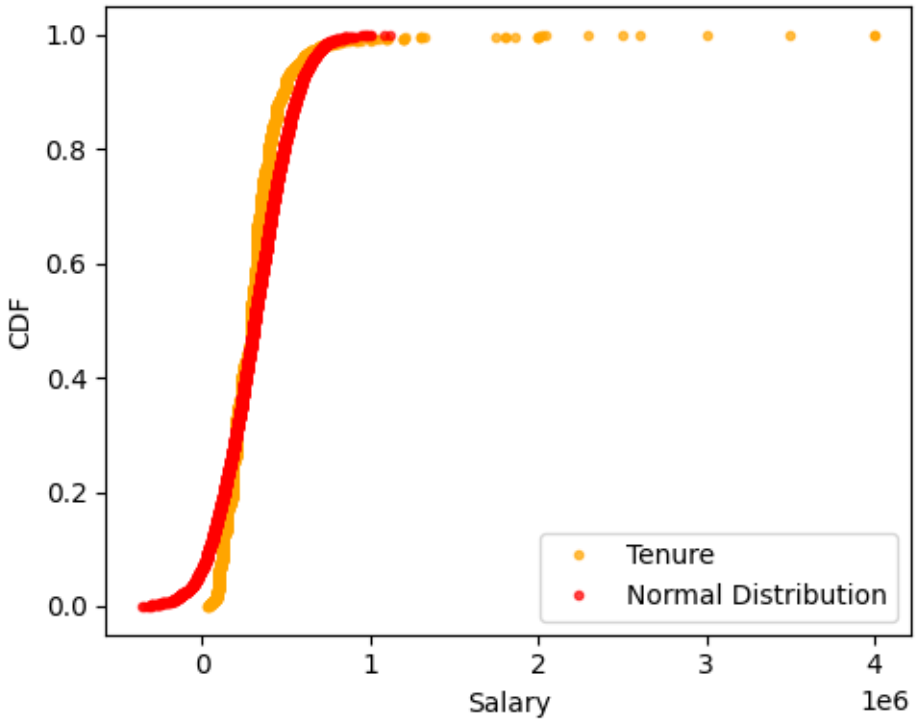
```
# CDF

plt.figure(figsize=(5,4))
x_salary, y_salary = cdf(df1['Salary'])
x_sample_salary, y_sample_salary = \
cdf(np.random.normal(df1['Salary'].mean(), df1['Salary'].std(), size =␣
 ↪len(df1['Salary'])))
plt.plot(x_salary, y_salary, linestyle = 'None',
        marker = '.', color = 'orange',
         alpha = 0.7, label = 'Tenure')
plt.plot(x_sample_salary, y_sample_salary, linestyle = 'None',
        marker ='.', color = 'red',
        alpha = 0.7, label = 'Normal Distribution')
plt.xlabel('Salary')
plt.ylabel('CDF')
plt.legend()
plt.tight_layout()
plt.show()
```






```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:640:
FutureWarning: SeriesGroupBy.grouper is deprecated and will be removed in a
future version of pandas.
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```


1e6
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Salary
Salary


1.0
0.8
0.6
0.4
0.2
0.0
CDF
Tenure
Normal Distribution
0
1
2
3
4
Salary
1e6

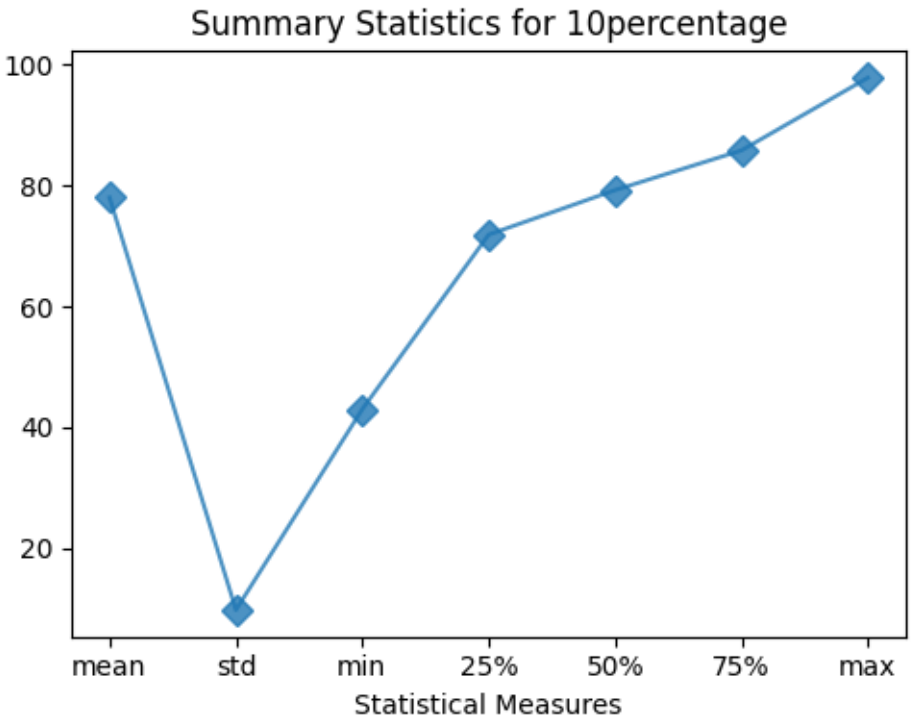
Conclusion	Inferences
1. Summary Plot	There is substantial variation in salary across the dataset.
2. Histogram	The data exhibits significant positive skewness, with a skewness value around 6 (approximately), indicating a departure from a normal distribution. The measures of central tendency (mean, median, and mode) are approximately equal.
3. Box Plot	There is a notable concentration of data points with high salaries, as depicted by the box plot.
4. CDF	The cumulative distribution function (CDF) reveals a high degree of skewness in the data, with considerable deviation from a normal distribution pattern.

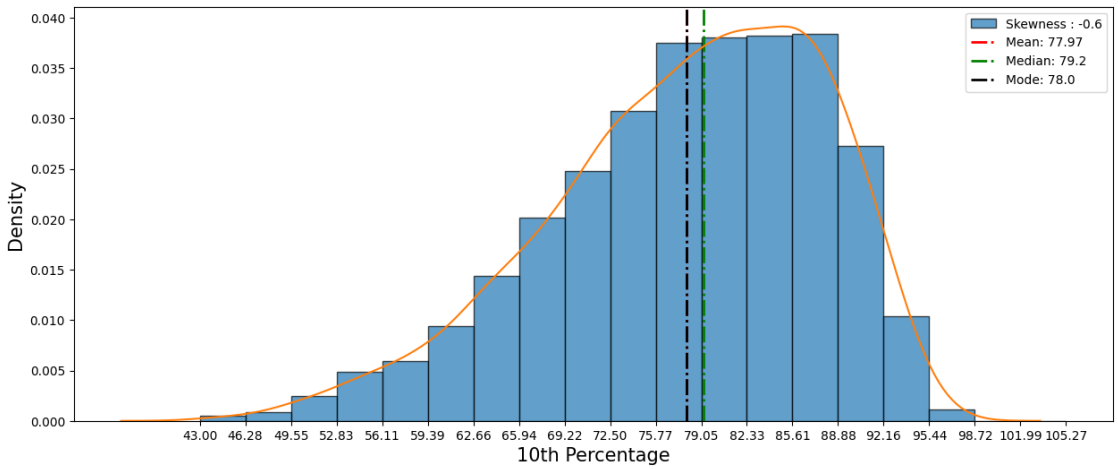
Observations

1.3 10th Percentage

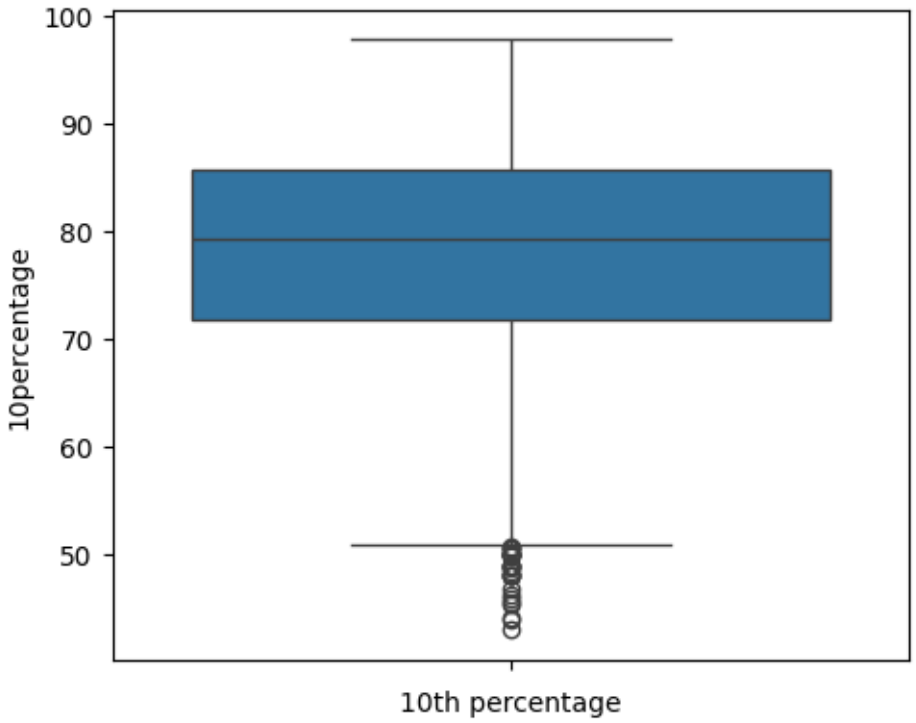
[ ]:
```
# Summary Plot

plt.figure(figsize=(5,4))
df1['10percentage'].describe()[1:].plot(alpha = 0.8,
                                    marker = 'D', markersize = 8)
plt.title('Summary Statistics for 10percentage')
plt.xlabel('Statistical Measures')
plt.tight_layout()
plt.show()

#Histogram


bins = np.arange(df1['10percentage'].min(), df1['10percentage'].
 ↪max()+df1['10percentage'].std(),
                  df1['10percentage'].std()/3)
plt.figure(figsize = (15,6))
plt.hist(df1['10percentage'], ec = 'k',
         bins = bins,
         label = f"Skewness : {round(df1['10percentage'].skew(),2)}",
         alpha = 0.7,
         density = True)
plt.xticks(bins)
plt.xlabel('10th Percentage', size = 15)
plt.ylabel('Density', size = 15)
```

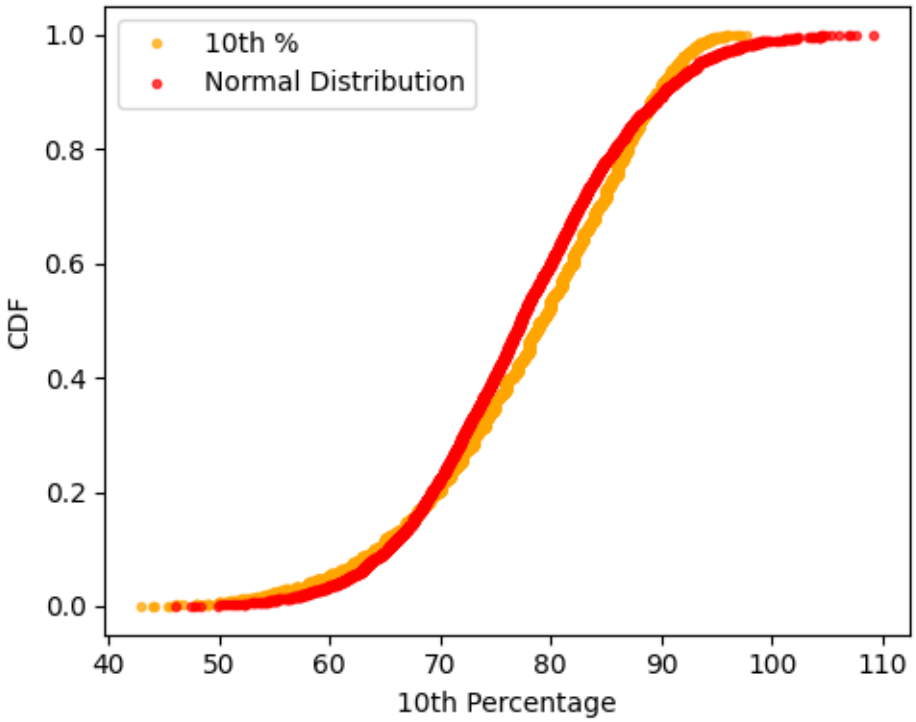
```
plt.axvline(df1['10percentage'].mean(), label = f"Mean:␣
 ↪{round(df1['10percentage'].mean(),2)}"
            , linestyle = '-.',
           color = 'red', linewidth = 2)
plt.axvline(df1['10percentage'].median(), label = f"Median:␣
 ↪{round(df1['10percentage'].median(),2)}"
            , linestyle = '-.',
           color = 'green', linewidth = 2)
plt.axvline(df1['10percentage'].mode()[0], label = f"Mode:␣
 ↪{round(df1['10percentage'].mode()[0],2)}"
            , linestyle = '-.',
           color = 'k', linewidth = 2)
sns.kdeplot(df1['10percentage'])
plt.legend()
plt.show()

#Box Plot

plt.figure(figsize=(5,4))
sns.boxplot(df1['10percentage'])
plt.xlabel('10th percentage')
plt.tight_layout()
plt.show()

# CDF

plt.figure(figsize=(5,4))
x_10, y_10 = cdf(df1['10percentage'])
x_sample_10 , y_sample_10 = \
cdf(np.random.normal(df1['10percentage'].mean(), df1['10percentage'].std(),␣
 ↪size = len(df1['10percentage'])))
plt.plot(x_10, y_10, linestyle = 'None',
        marker = '.', color = 'orange',
         alpha = 0.7, label = '10th %')
plt.plot(x_sample_10, y_sample_10, linestyle = 'None',
        marker ='.', color = 'red',
        alpha = 0.7, label = 'Normal Distribution')
plt.xlabel('10th Percentage')
plt.ylabel('CDF')
plt.legend()
plt.tight_layout()
plt.show()
```






```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:640:
FutureWarning: SeriesGroupBy.grouper is deprecated and will be removed in a
future version of pandas.
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```


100
90
80
70
60
50
10percentage
10th percentage


10th %
Normal Distribution
1.0
0.8
0.6
0.4
0.2
0.0
CDF
40
50
60
70
80
90
100
110
10th Percentage

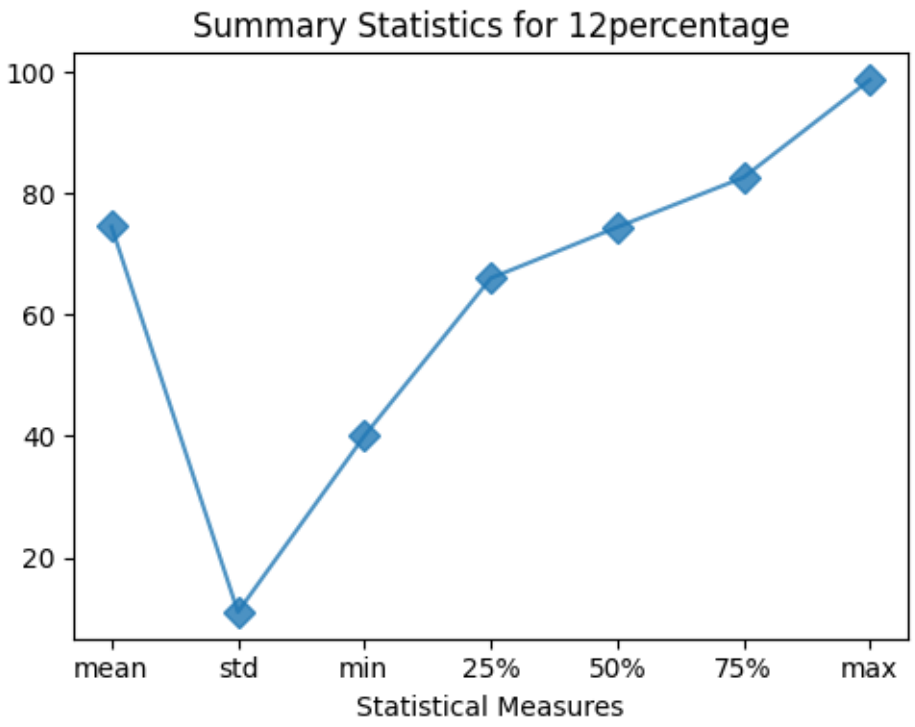
Conclusions	Inferences
1. Summary Plot	Around 50% of students achieved scores of approximately 80% or less.
2. Histogram	The histogram depicts a scarcity of students with low percentages, with the majority falling within the 75% to 90% range. The peak frequency occurs at 78%, and the average score hovers around 77%.
3. Box Plot	The presence of a few extreme outliers is evident from the box plot.
4. CDF	The data exhibits some skewness and does not conform to a normal distribution pattern.

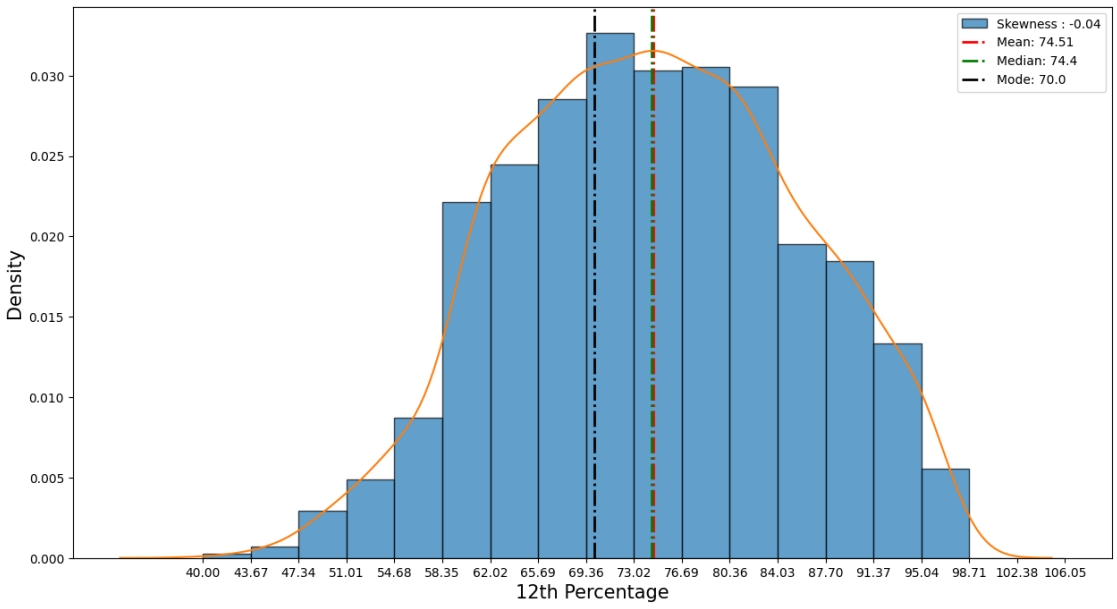
Observations

1.4 12th Percentage

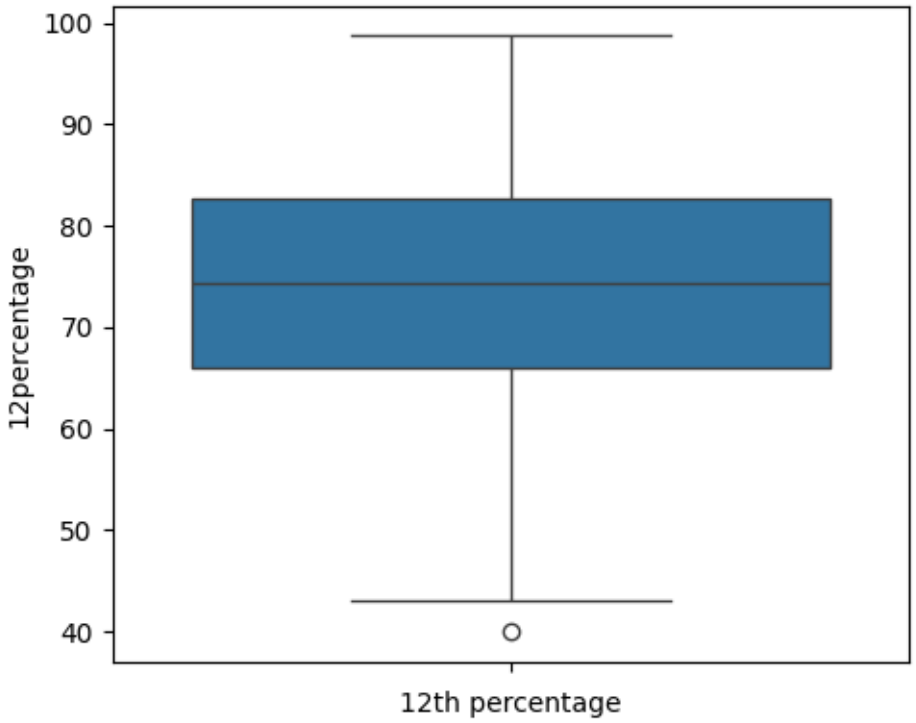
[ ]:
```
# Summary Plot

plt.figure(figsize=(5,4))
df1['12percentage'].describe()[1:].plot(alpha = 0.8,
                                  marker = 'D', markersize = 8)
plt.title('Summary Statistics for 12percentage')
plt.xlabel('Statistical Measures')
plt.tight_layout()
plt.show()

# Histogram

bins = np.arange(df1['12percentage'].min(), df1['12percentage'].
 ↪max()+df1['12percentage'].std(),
                 df1['12percentage'].std()/3)
plt.figure(figsize = (15,8))
plt.hist(df1['12percentage'], ec = 'k',
        bins = bins,
        label = f"Skewness : {round(df1['12percentage'].skew(),2)}",
        alpha = 0.7,
        density = True)
plt.xticks(bins)
plt.xlabel('12th Percentage', size = 15)
plt.ylabel('Density', size = 15)

plt.axvline(df1['12percentage'].mean(), label = f"Mean:␣
 ↪{round(df1['12percentage'].mean(),2)}"
            , linestyle = '-.',
           color = 'red', linewidth = 2)
```

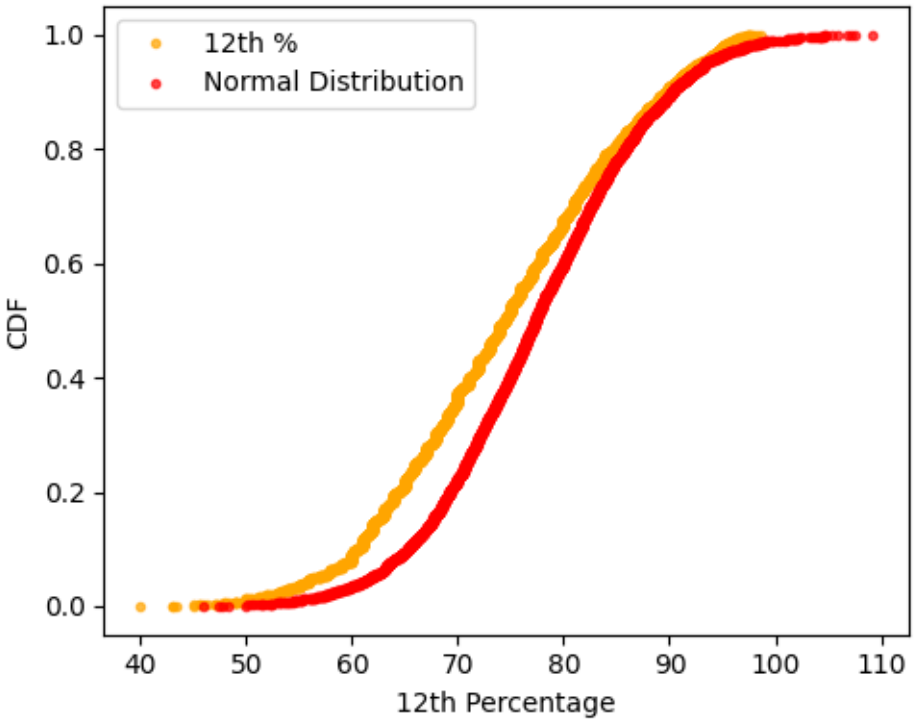
```
plt.axvline(df1['12percentage'].median(), label = f"Median:␣
 ↪{round(df1['12percentage'].median(),2)}"
            , linestyle = '-.',
           color = 'green', linewidth = 2)
plt.axvline(df1['12percentage'].mode()[0], label = f"Mode:␣
 ↪{round(df1['12percentage'].mode()[0],2)}"
            , linestyle = '-.',
           color = 'k', linewidth = 2)
sns.kdeplot(df1['12percentage'])
plt.legend()
plt.show()

#Box Plot

plt.figure(figsize=(5,4))
sns.boxplot(df1['12percentage'])
plt.xlabel('12th percentage')
plt.tight_layout()
plt.show()

# CDF

plt.figure(figsize=(5,4))
x_12, y_12 = cdf(df1['12percentage'])
x_sample_12 , y_sample_12 = \
cdf(np.random.normal(df1['12percentage'].mean(), df1['12percentage'].std(),␣
 ↪size = len(df1['12percentage'])))
plt.plot(x_12, y_12, linestyle = 'None',
        marker = '.', color = 'orange',
         alpha = 0.7, label = '12th %')
plt.plot(x_sample_10, y_sample_10, linestyle = 'None',
        marker ='.', color = 'red',
        alpha = 0.7, label = 'Normal Distribution')
plt.xlabel('12th Percentage')
plt.ylabel('CDF')
plt.legend()
plt.tight_layout()
plt.show()
```






```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:640:
FutureWarning: SeriesGroupBy.grouper is deprecated and will be removed in a
future version of pandas.
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```


100
90
80
70
60
50
40
12percentage
12th percentage


12th %
Normal Distribution
1.0
0.8
0.6
0.4
0.2
0.0
CDF
40
50
60
70
80
90
100
110
12th Percentage

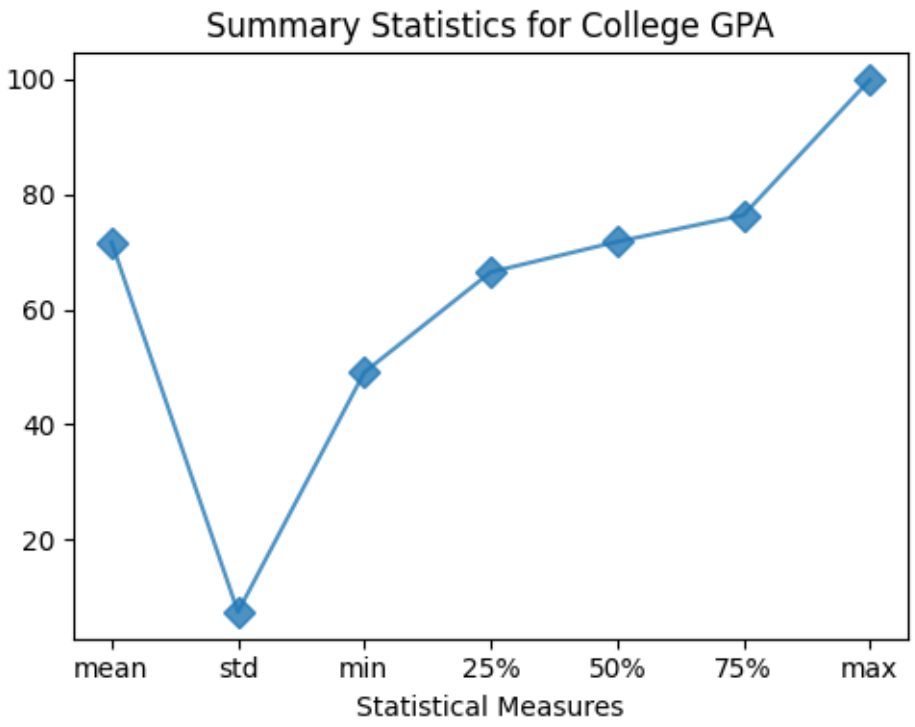
Conclusions	Inferences
1. Summary Plot	Roughly half of the students achieved scores of approximately 78% or lower.
2. Histogram	The histogram illustrates a scarcity of students with low percentages, with the majority scoring between 69% and 84%. The peak frequency occurs at 70%, and the average score is around 74%.
3. Box Plot	The box plot indicates only one data point with an extremely low score.
4. CDF	The data does not follow a normal distribution pattern.

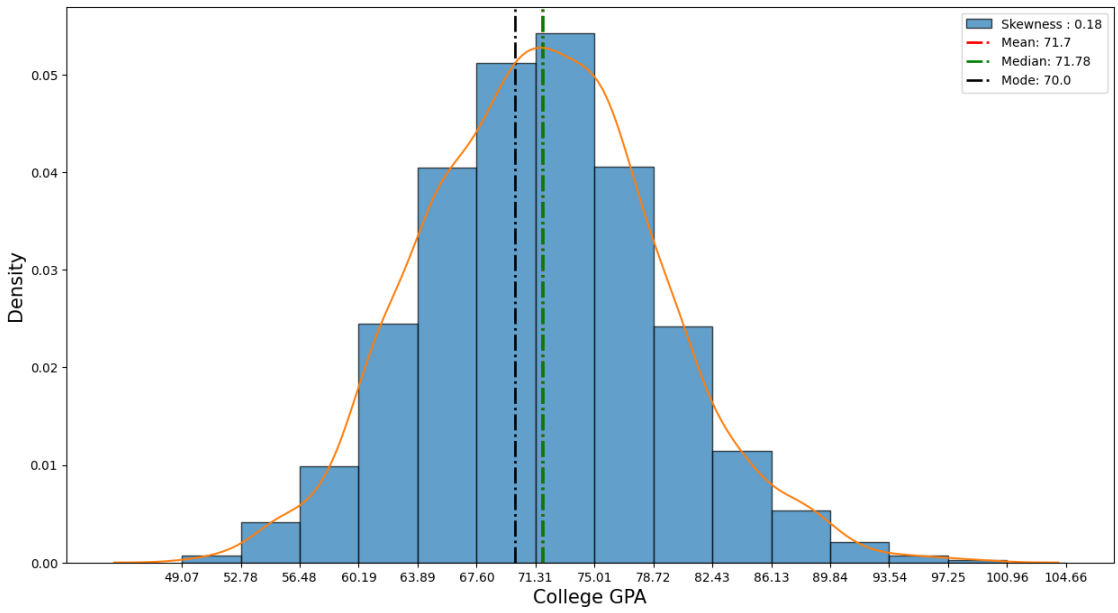
Observations

1.5 CollegeGPA

[ ]:
```
#Summary Plot

plt.figure(figsize=(5,4))
df1['collegeGPA'].describe()[1:].plot(alpha = 0.8,
                                      marker = 'D', markersize = 8)
plt.title('Summary Statistics for College GPA')
plt.xlabel('Statistical Measures')
plt.tight_layout()
plt.show()

# Histogram

bins = np.arange(df1['collegeGPA'].min(), df1['collegeGPA'].
 ↪max()+df1['collegeGPA'].std(),
                  df1['collegeGPA'].std()/2)
plt.figure(figsize = (15,8))
plt.hist(df1['collegeGPA'], ec = 'k',
         bins = bins,
         label = f"Skewness : {round(df1['collegeGPA'].skew(),2)}",
         alpha = 0.7,
         density = True)
plt.xticks(bins)
plt.xlabel('College GPA', size = 15)
plt.ylabel('Density', size = 15)

plt.axvline(df1['collegeGPA'].mean(), label = f"Mean: {round(df1['collegeGPA'].
 ↪mean(),2)}"
            , linestyle = '-.',
           color = 'red', linewidth = 2)
```

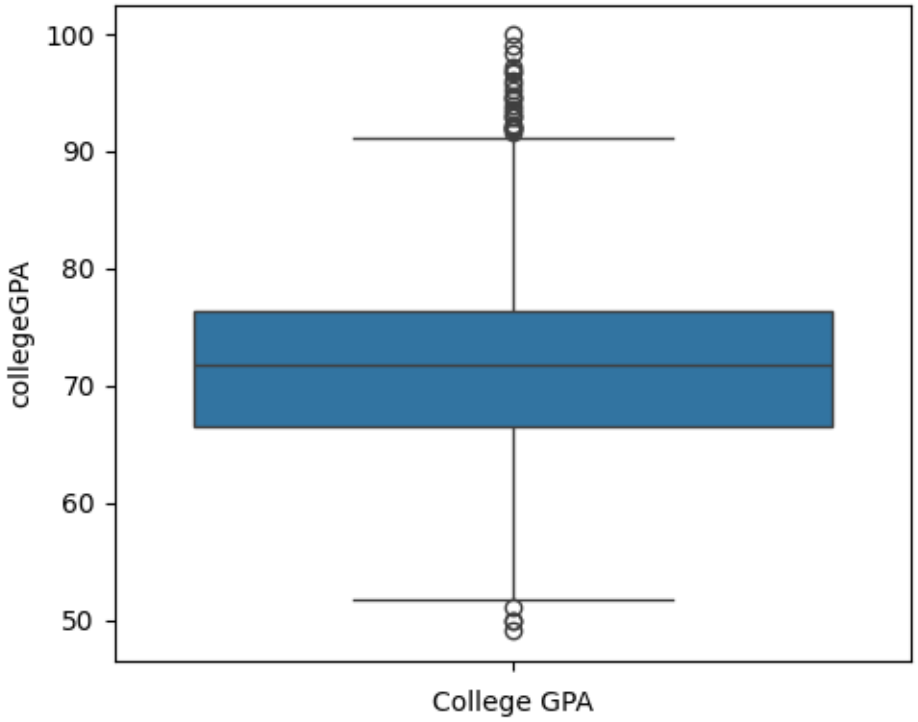
```
plt.axvline(df1['collegeGPA'].median(), label = f"Median:␣
 ↪{round(df1['collegeGPA'].median(),2)}"
            , linestyle = '-.',
           color = 'green', linewidth = 2)
plt.axvline(df1['collegeGPA'].mode()[0], label = f"Mode:␣
 ↪{round(df1['collegeGPA'].mode()[0],2)}"
            , linestyle = '-.',
           color = 'k', linewidth = 2)
sns.kdeplot(df1['collegeGPA'])
plt.legend()
plt.show()

# Box Plot

plt.figure(figsize=(5,4))
sns.boxplot(df1['collegeGPA'])
plt.xlabel('College GPA')
plt.tight_layout()
plt.show()

# CDF

plt.figure(figsize=(5,4))
x_gpa, y_gpa = cdf(df1['collegeGPA'])
x_sample_gpa , y_sample_gpa = \
cdf(np.random.normal(df1['collegeGPA'].mean(), df1['collegeGPA'].std(), size =␣
 ↪len(df1['12percentage'])))
plt.plot(x_gpa, y_gpa, linestyle = 'None',
        marker = '.', color = 'orange',
         alpha = 0.7, label = 'GPA')
plt.plot(x_sample_gpa, y_sample_gpa, linestyle = 'None',
        marker ='.', color = 'red',
        alpha = 0.7, label = 'Normal Distribution')
plt.xlabel('College GPA')
plt.ylabel('CDF')
plt.legend()
plt.tight_layout()
plt.show()
```

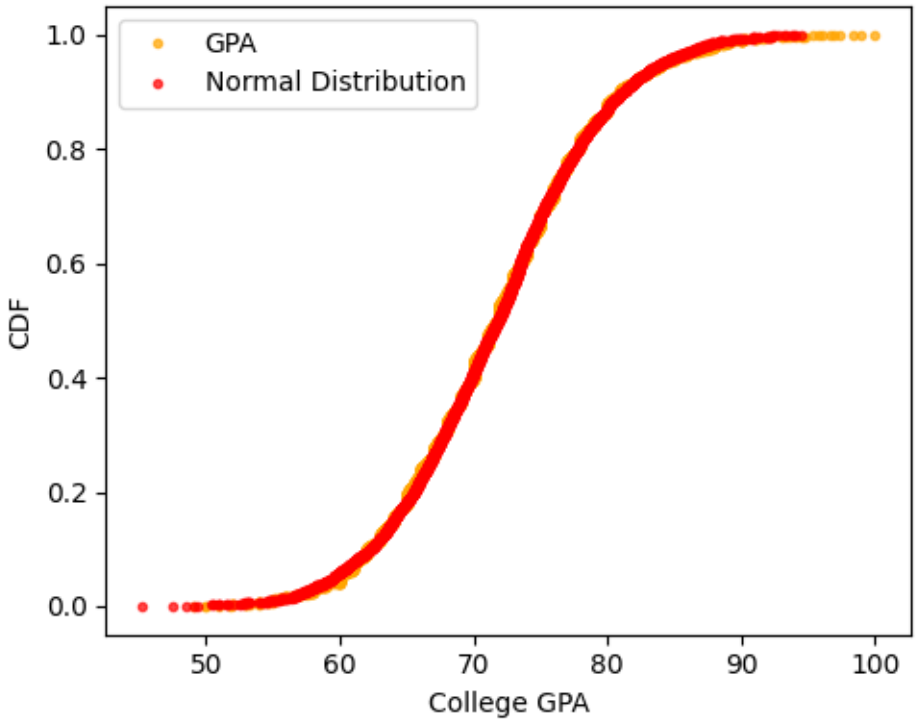





```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:640:
FutureWarning: SeriesGroupBy.grouper is deprecated and will be removed in a
future version of pandas.
```

```
positions = grouped.grouper.result_index.to_numpy(dtype=float)
```





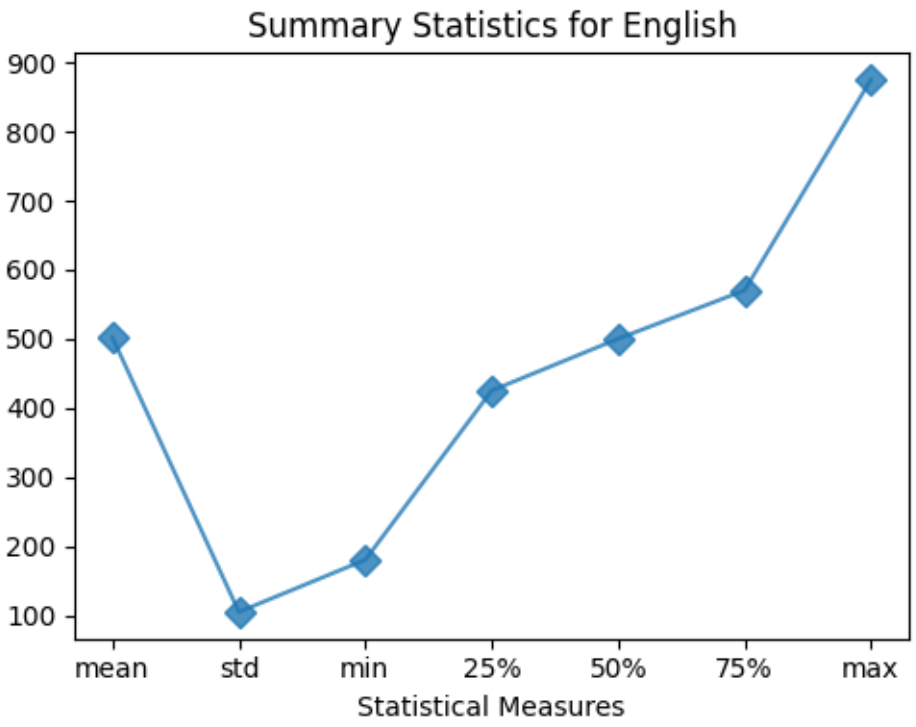
Conclusions	Inferences
1. Summary Plot	75% of students had a GPA of approximately 80% or lower.
2. Histogram	The majority of students had GPAs ranging between 63% and 78%. The highest frequency of students scored 70%, and the average GPA was 74%.
3. Box Plot	The box plot reveals the presence of both low and high extreme values within the dataset.
4. CDF	The data is deemed to be sufficiently normally distributed.

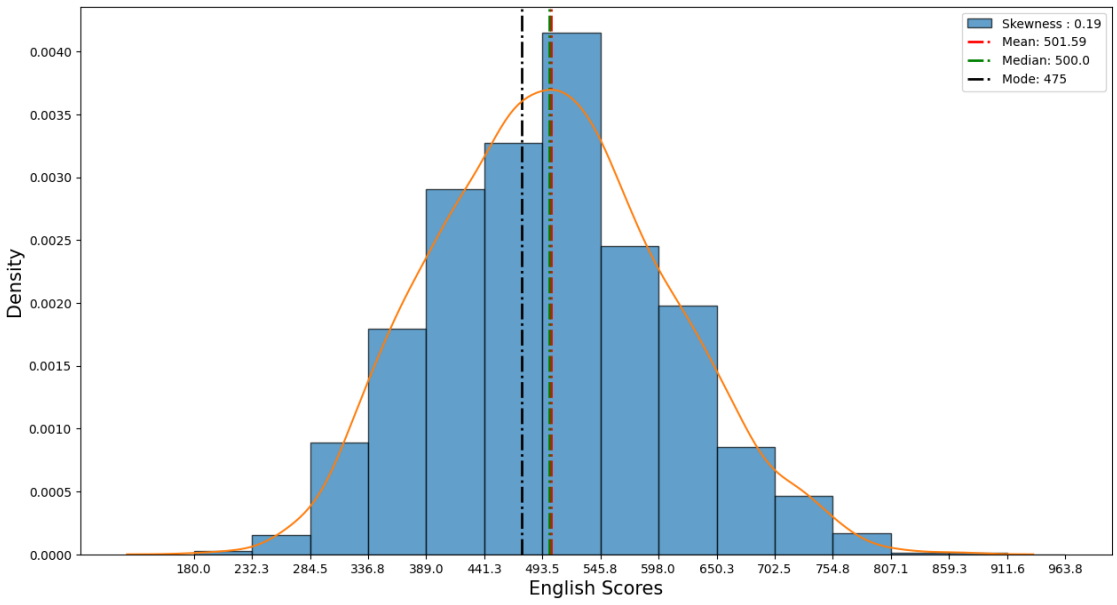
Observations

1.6 English

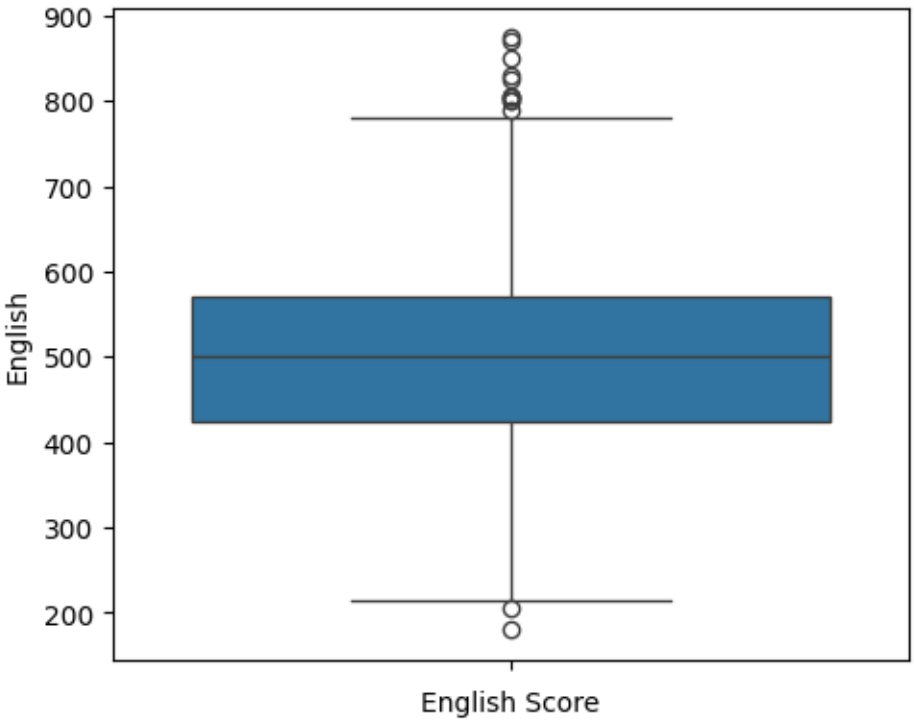
[ ]:
```
# Summary Plot

plt.figure(figsize=(5,4))
df1['English'].describe()[1:].plot(alpha = 0.8,
                                   marker = 'D', markersize = 8)
plt.title('Summary Statistics for English')
plt.xlabel('Statistical Measures')
plt.tight_layout()
plt.show()

# Histogram

bins = np.arange(df1['English'].min(), df1['English'].max()+df1['English'].
 ↪std(),
                 df1['English'].std()/2)
plt.figure(figsize = (15,8))
plt.hist(df1['English'], ec = 'k',
        bins = bins,
        label = f"Skewness : {round(df1['English'].skew(),2)}",
        alpha = 0.7,
        density = True)
plt.xticks(bins)
plt.xlabel('English Scores', size = 15)
plt.ylabel('Density', size = 15)

plt.axvline(df1['English'].mean(), label = f"Mean: {round(df1['English'].
 ↪mean(),2)}"
            , linestyle = '-.',
```

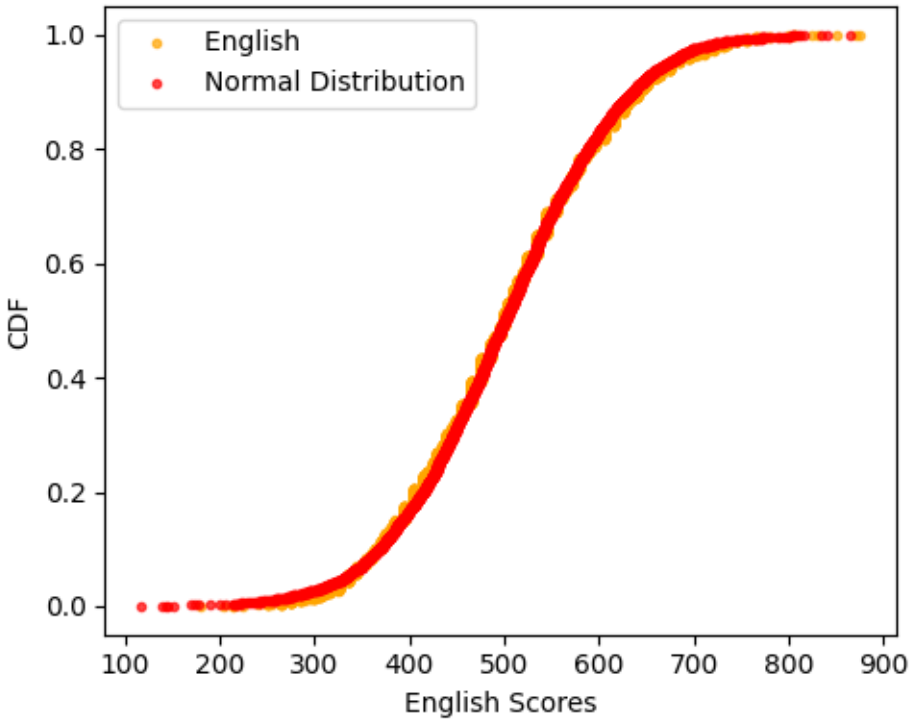
```
           color = 'red', linewidth = 2)
plt.axvline(df1['English'].median(), label = f"Median: {round(df1['English'].
 ↪median(),2)}"
            , linestyle = '-.',
           color = 'green', linewidth = 2)
plt.axvline(df1['English'].mode()[0], label = f"Mode: {round(df1['English'].
 ↪mode()[0],2)}"
            , linestyle = '-.',
           color = 'k', linewidth = 2)
sns.kdeplot(df1['English'])
plt.legend()
plt.show()

# Box Plot

plt.figure(figsize=(5,4))
sns.boxplot(df1['English'])
plt.xlabel('English Score')
plt.tight_layout()
plt.show()

# CDF

plt.figure(figsize=(5,4))
x_eng, y_eng = cdf(df1['English'])
x_sample_eng , y_sample_eng = \
cdf(np.random.normal(df1['English'].mean(), df1['English'].std(), size =␣
 ↪len(df1['English'])))
plt.plot(x_eng, y_eng, linestyle = 'None',
        marker = '.', color = 'orange',
         alpha = 0.7, label = 'English ')
plt.plot(x_sample_eng, y_sample_eng, linestyle = 'None',
        marker ='.', color = 'red',
        alpha = 0.7, label = 'Normal Distribution')
plt.xlabel('English Scores')
plt.ylabel('CDF')
plt.legend()
plt.tight_layout()
plt.show()
```






```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:640:
FutureWarning: SeriesGroupBy.grouper is deprecated and will be removed in a
future version of pandas.
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```


900
800
700
600
500
400
300
200
English
English Score


English
Normal Distribution
1.0
0.8
0.6
0.4
0.2
0.0
CDF
100
200
300
400
500
600
700
800
900
English Scores

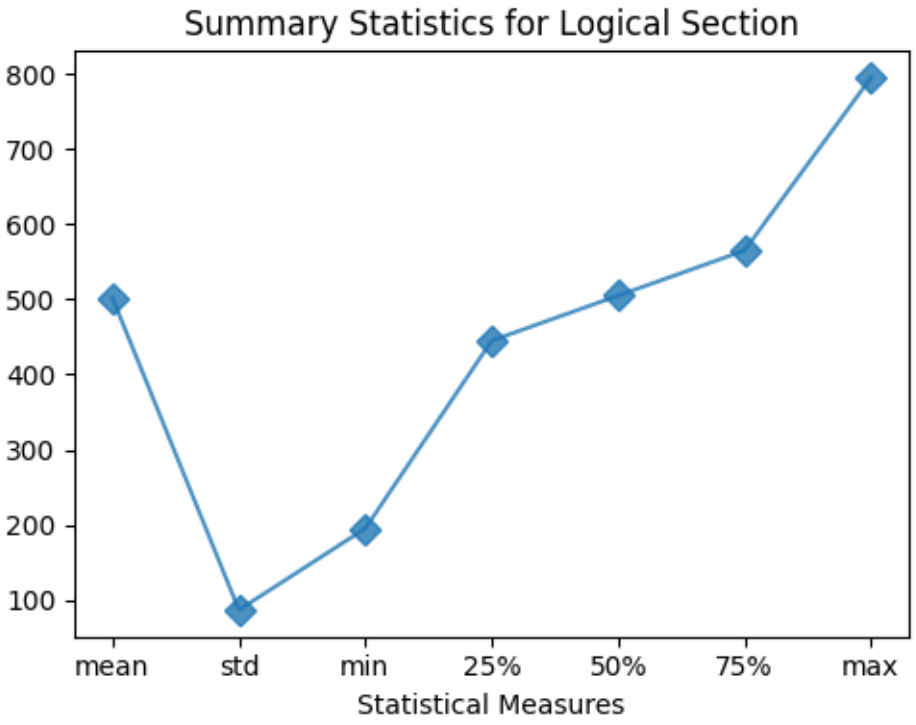
Conclusions	Paraphrased Version
Summary Plot	Half of the students scored below 500 in their English exams.
Histogram	The bulk of the scores fell within the range of 389 to 545. The peak occurred at 475, with an average score of 502.
Box Plot	Both lower and higher extreme values are evident from the distribution representation.
CDF	The data follows a reasonably normal distribution pattern.

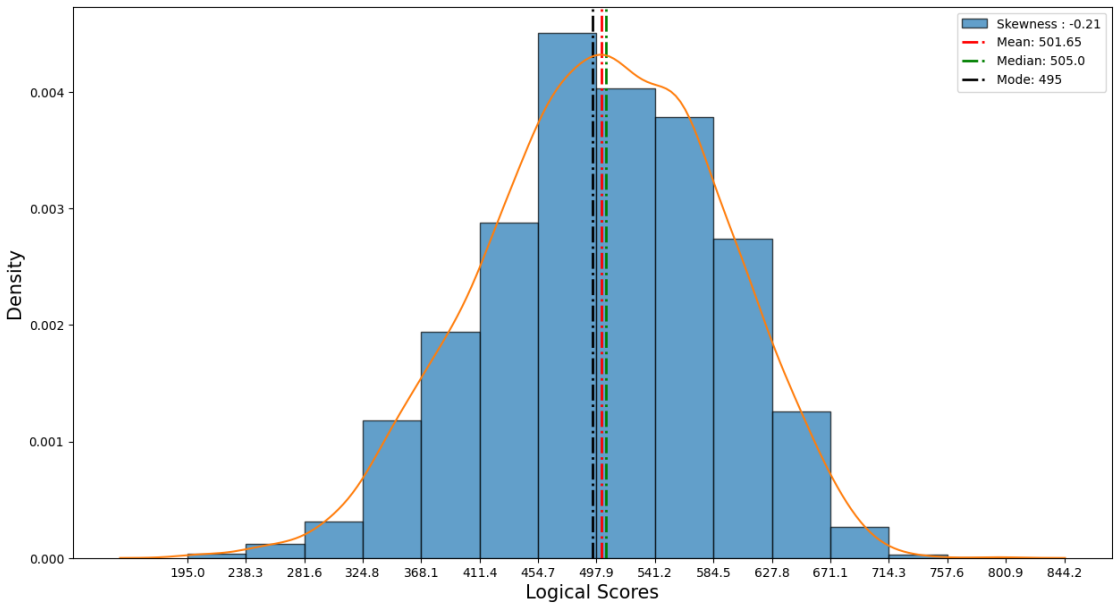
Observations

1.7 Logical

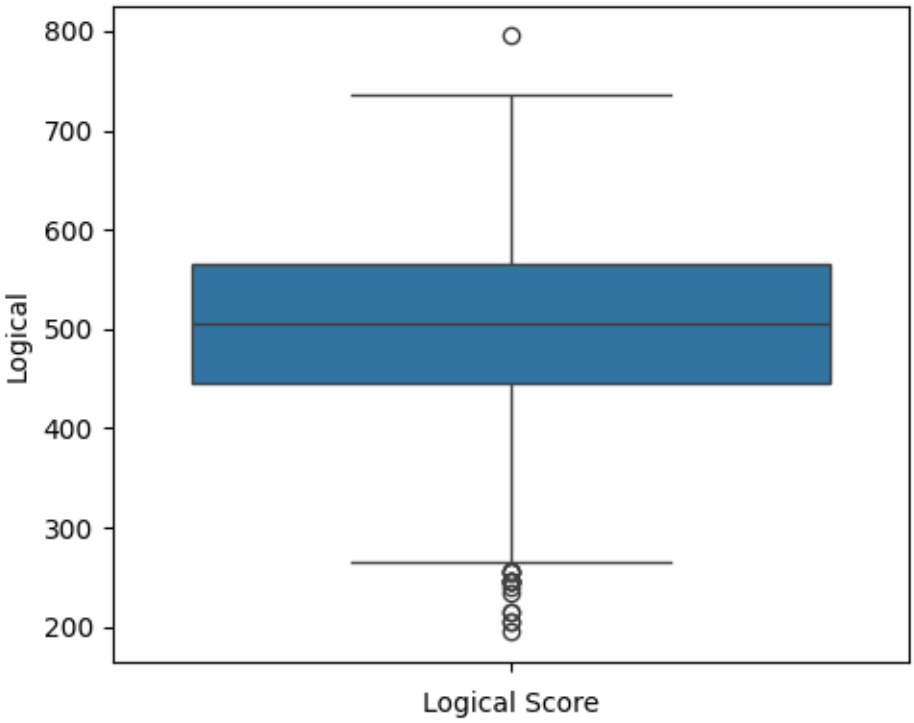
[ ]:
```
# Summary Plot

plt.figure(figsize=(5,4))
df1['Logical'].describe()[1:].plot(alpha = 0.8,
                                   marker = 'D', markersize = 8)
plt.title('Summary Statistics for Logical Section')
plt.xlabel('Statistical Measures')
plt.tight_layout()
plt.show()

# Histogram

bins = np.arange(df1['Logical'].min(), df1['Logical'].max()+df1['Logical'].
 ↪std(),
                  df1['Logical'].std()/2)
plt.figure(figsize = (15,8))
plt.hist(df1['Logical'], ec = 'k',
        bins = bins,
        label = f"Skewness : {round(df1['Logical'].skew(),2)}",
        alpha = 0.7,
        density = True)
plt.xticks(bins)
plt.xlabel('Logical Scores', size = 15)
plt.ylabel('Density', size = 15)

plt.axvline(df1['Logical'].mean(), label = f"Mean: {round(df1['Logical'].
 ↪mean(),2)}"
            , linestyle = '-.',
           color = 'red', linewidth = 2)
plt.axvline(df1['Logical'].median(), label = f"Median: {round(df1['Logical'].
 ↪median(),2)}"
            , linestyle = '-.',
           color = 'green', linewidth = 2)
```

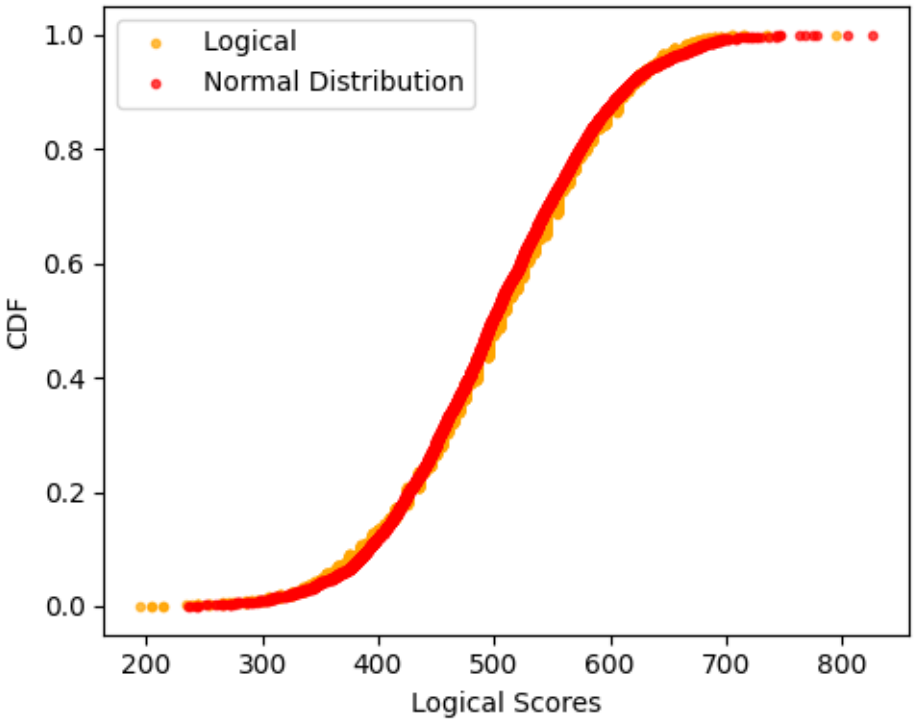
```
plt.axvline(df1['Logical'].mode()[0], label = f"Mode: {round(df1['Logical'].
 ↪mode()[0],2)}"
            , linestyle = '-.',
           color = 'k', linewidth = 2)
sns.kdeplot(df1['Logical'])
plt.legend()
plt.show()

# Box Plot

plt.figure(figsize=(5,4))
sns.boxplot(df1['Logical'])
plt.xlabel('Logical Score')
plt.tight_layout()
plt.show()

# CDF

plt.figure(figsize=(5,4))
x_log, y_log = cdf(df1['Logical'])
x_sample_log , y_sample_log = \
cdf(np.random.normal(df1['Logical'].mean(), df1['Logical'].std(), size =␣
 ↪len(df1['Logical'])))
plt.plot(x_log, y_log, linestyle = 'None',
        marker = '.', color = 'orange',
         alpha = 0.7, label = 'Logical ')
plt.plot(x_sample_log, y_sample_log, linestyle = 'None',
        marker ='.', color = 'red',
        alpha = 0.7, label = 'Normal Distribution')
plt.xlabel('Logical Scores')
plt.ylabel('CDF')
plt.legend()
plt.tight_layout()
plt.show()
```






```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:640:
FutureWarning: SeriesGroupBy.grouper is deprecated and will be removed in a
future version of pandas.
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```


800
700
600
500
400
300
200
Logical
Logical Score


Logical
Normal Distribution
1.0
0.8
0.6
0.4
0.2
0.0
CDF
200
300
400
500
600
700
800
Logical Scores

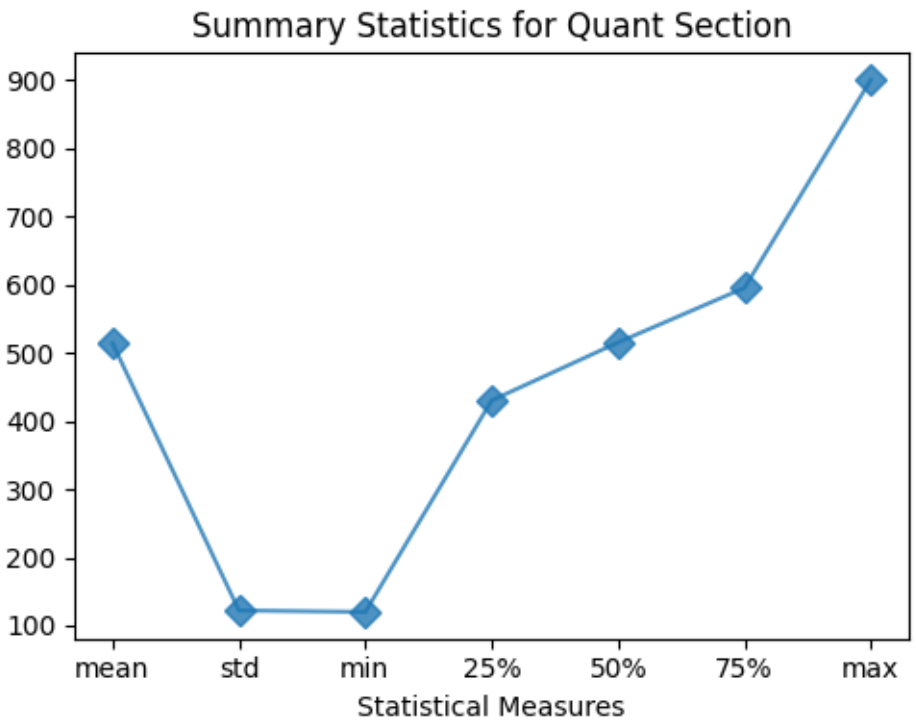
Conclusions	Inferences
Summary Plot	Half of the students scored below 500 in the logical exams.
Histogram	Most scores fell within the range of 454 to 584, peaking at 495, with an average of 502.
Box Plot	Presence of lower extreme values, with only one high extreme value being notable.
CDF	Data closely approximates a normal distribution pattern.

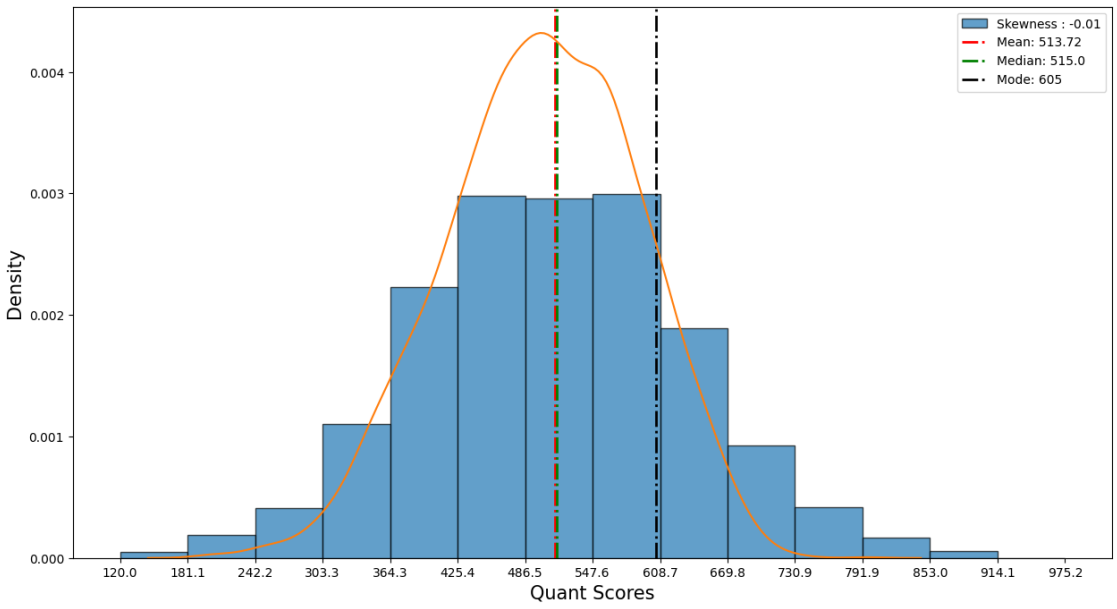
Observations

1.8 Quant

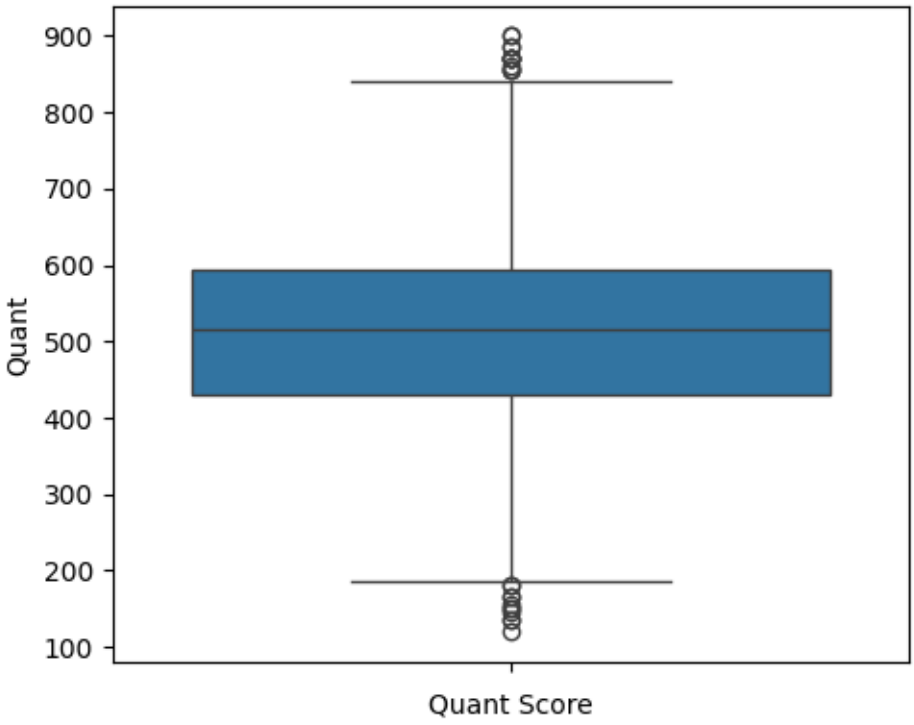
[ ]:
```
# Summary Plot

plt.figure(figsize=(5,4))
df1['Quant'].describe()[1:].plot(alpha = 0.8,
                                 marker = 'D', markersize = 8)
plt.title('Summary Statistics for Quant Section')
plt.xlabel('Statistical Measures')
plt.tight_layout()
plt.show()

# Histogram

bins = np.arange(df1['Quant'].min(), df1['Quant'].max()+df1['Quant'].std(),
                 df1['Quant'].std()/2)
plt.figure(figsize = (15,8))
plt.hist(df1['Quant'], ec = 'k',
        bins = bins,
        label = f"Skewness : {round(df1['Quant'].skew(),2)}",
        alpha = 0.7,
        density = True)
plt.xticks(bins)
plt.xlabel('Quant Scores', size = 15)
plt.ylabel('Density', size = 15)

plt.axvline(df1['Quant'].mean(), label = f"Mean: {round(df1['Quant'].mean(),2)}"
            , linestyle = '-.',
           color = 'red', linewidth = 2)
plt.axvline(df1['Quant'].median(), label = f"Median: {round(df1['Quant'].
 ↪median(),2)}"
            , linestyle = '-.',
           color = 'green', linewidth = 2)
plt.axvline(df1['Quant'].mode()[0], label = f"Mode: {round(df1['Quant'].
 ↪mode()[0],2)}"
```

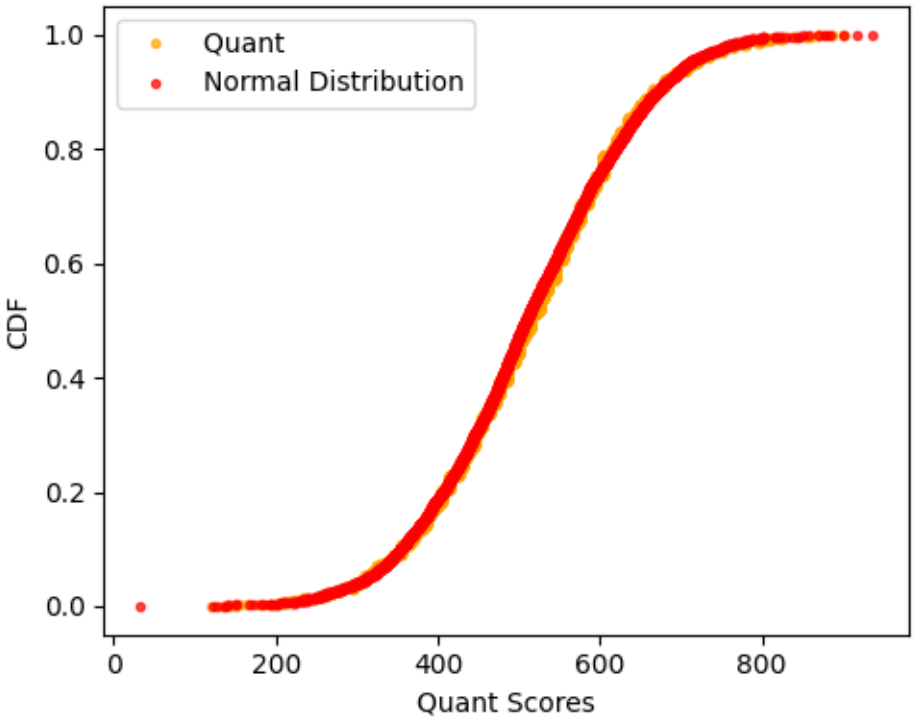
```
            , linestyle = '-.',
           color = 'k', linewidth = 2)
sns.kdeplot(df1['Logical'])
plt.legend()
plt.show()

# Box Plot

plt.figure(figsize=(5,4))
sns.boxplot(df1['Quant'])
plt.xlabel('Quant Score')
plt.tight_layout()
plt.show()

# CDF

plt.figure(figsize=(5,4))
x_q, y_q = cdf(df1['Quant'])
x_sample_q , y_sample_q = \
cdf(np.random.normal(df1['Quant'].mean(), df1['Quant'].std(), size =␣
 ↪len(df1['Quant'])))
plt.plot(x_q, y_q, linestyle = 'None',
        marker = '.', color = 'orange',
         alpha = 0.7, label = 'Quant ')
plt.plot(x_sample_q, y_sample_q, linestyle = 'None',
        marker ='.', color = 'red',
        alpha = 0.7, label = 'Normal Distribution')
plt.xlabel('Quant Scores')
plt.ylabel('CDF')
plt.legend()
plt.tight_layout()
plt.show()
```






```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:640:
FutureWarning: SeriesGroupBy.grouper is deprecated and will be removed in a
future version of pandas.
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```


900
800
700
600
500
400
300
200
100
Quant
Quant Score


Quant
Normal Distribution
1.0
0.8
0.6
0.4
0.2
0.0
CDF
0
200
400
600
800
Quant Scores

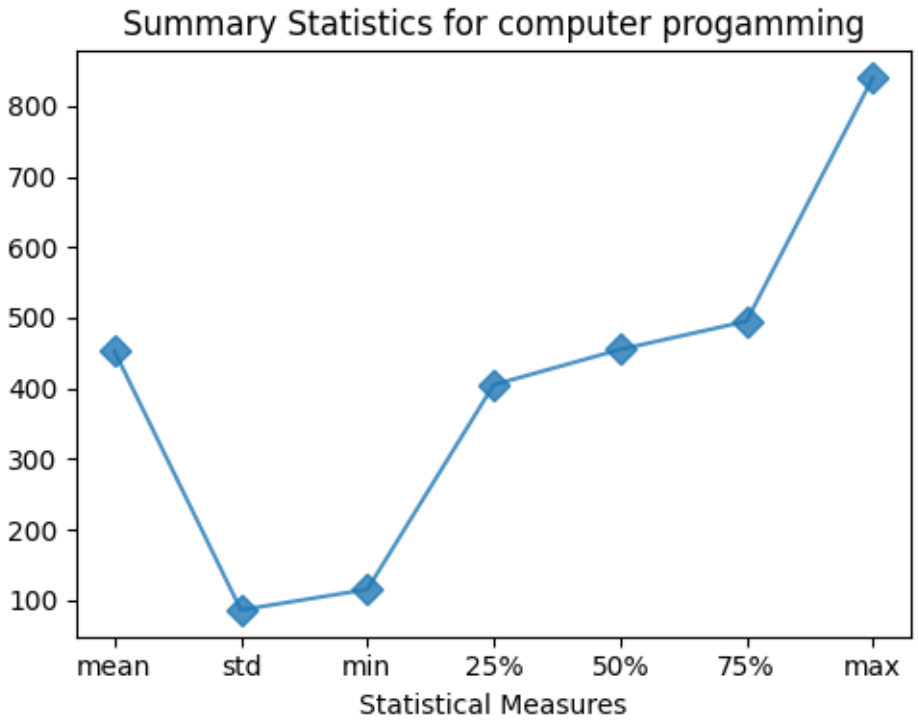
Conclusions	Inferences
Summary Plot	75% of students' logical score was less than 600.
Histogram	Majority of the scores were in between 425-608. The maximum number of students scored 605 with an average of 513.
Box Plot	The box plot shows the presence of both low and high extreme values.
CDF	The data is sufficiently close to normally distributed.

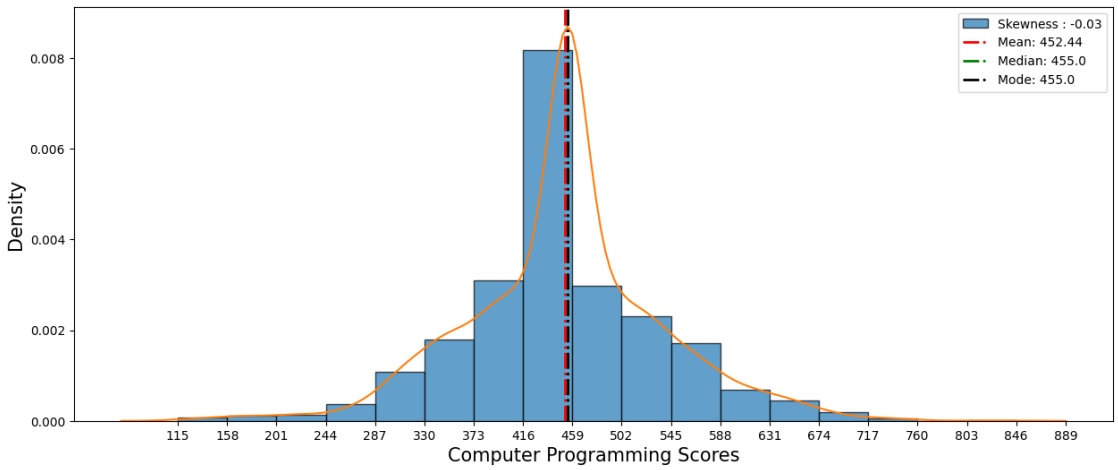
Observations

1.9 Computer Programming

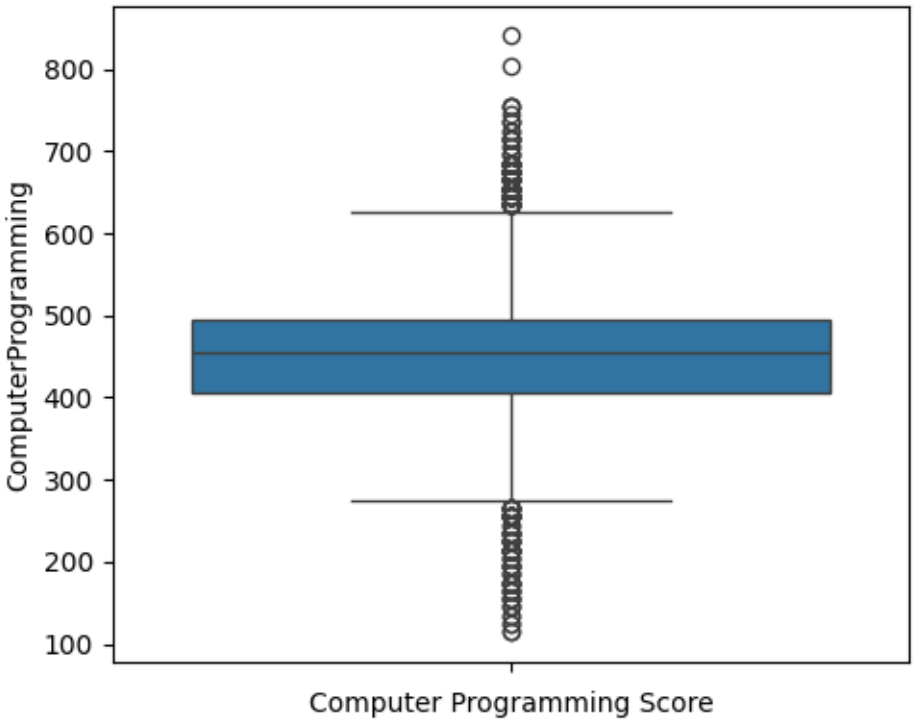
[ ]:
```
# Summary Plot

plt.figure(figsize=(5,4))
df1['ComputerProgramming'].describe()[1:].plot(alpha = 0.8,
                                  marker = 'D', markersize = 8)
plt.title('Summary Statistics for computer progamming')
plt.xlabel('Statistical Measures')
plt.tight_layout()
plt.show()

# Histogram

bins = np.arange(df1['ComputerProgramming'].min(), df1['ComputerProgramming'].
 ↪max()+df1['ComputerProgramming'].std(),
                 df1['ComputerProgramming'].std()/2)
plt.figure(figsize = (15,6))
plt.hist(df1['ComputerProgramming'], ec = 'k',
        bins = bins,
        label = f"Skewness : {round(df1['ComputerProgramming'].skew(),2)}",
        alpha = 0.7,
        density = True)
plt.xticks(bins)
plt.xlabel('Computer Programming Scores', size = 15)
plt.ylabel('Density', size = 15)

plt.axvline(df1['ComputerProgramming'].mean(), label = f"Mean:␣
 ↪{round(df1['ComputerProgramming'].mean(),2)}"
            , linestyle = '-.',
           color = 'red', linewidth = 2)
plt.axvline(df1['ComputerProgramming'].median(), label = f"Median:␣
 ↪{round(df1['ComputerProgramming'].median(),2)}"
            , linestyle = '-.',
           color = 'green', linewidth = 2)
plt.axvline(df1['ComputerProgramming'].mode()[0], label = f"Mode:␣
 ↪{round(df1['ComputerProgramming'].mode()[0],2)}"
```

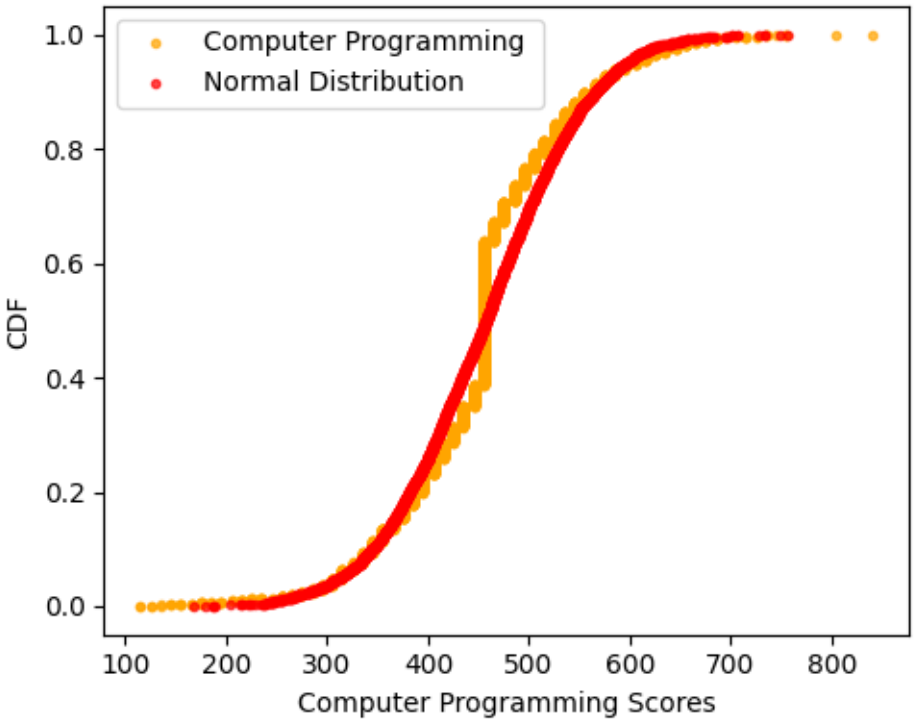
```
            , linestyle = '-.',
           color = 'k', linewidth = 2)
sns.kdeplot(df1['ComputerProgramming'])
plt.legend()
plt.show()

# Box Plot

plt.figure(figsize=(5,4))
sns.boxplot(df1['ComputerProgramming'])
plt.xlabel('Computer Programming Score')
plt.tight_layout()
plt.show()

# CDF

plt.figure(figsize=(5,4))
x_cp, y_cp = cdf(df1['ComputerProgramming'])
x_sample_cp , y_sample_cp = \
cdf(np.random.normal(df1['ComputerProgramming'].mean(),␣
 ↪df1['ComputerProgramming'].std(), size = \
                     len(df1['ComputerProgramming'])))
plt.plot(x_cp, y_cp, linestyle = 'None',
        marker = '.', color = 'orange',
         alpha = 0.7, label = 'Computer Programming ')
plt.plot(x_sample_cp, y_sample_cp, linestyle = 'None',
        marker ='.', color = 'red',
        alpha = 0.7, label = 'Normal Distribution')
plt.xlabel('Computer Programming Scores')
plt.ylabel('CDF')
plt.legend()
plt.tight_layout()
plt.show()
```






```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:640:
FutureWarning: SeriesGroupBy.grouper is deprecated and will be removed in a
future version of pandas.
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```


ComputerProgramming
800
700
600
500
400
300
200
100
Computer Programming Score


Computer Programming
Normal Distribution
CDF
1.0
0.8
0.6
0.4
0.2
0.0
100
200
300
400
500
600
700
800
Computer Programming Scores

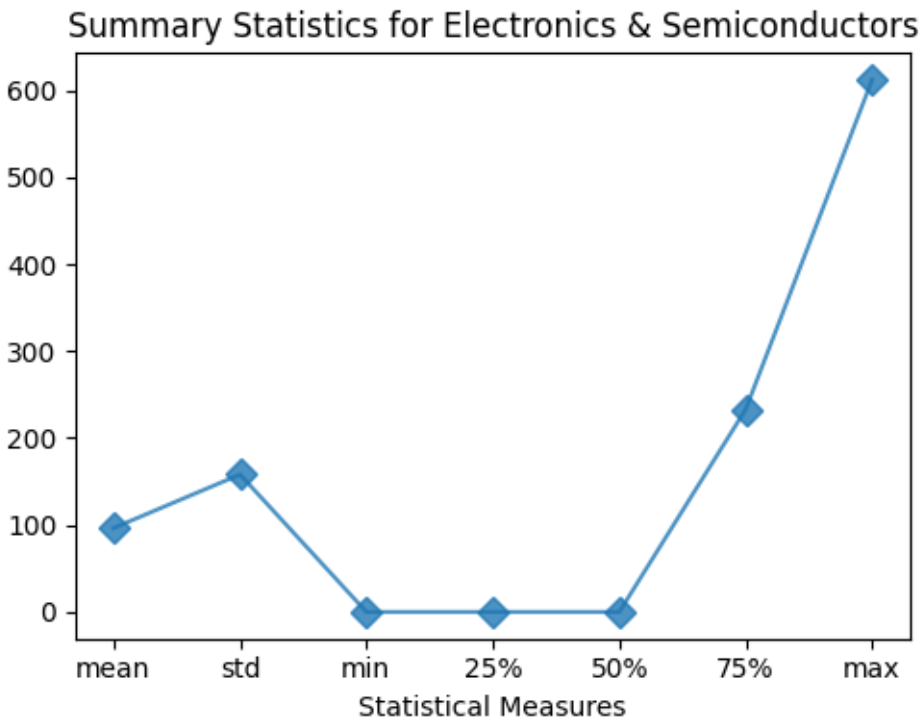
Conclusions	**Inferences** Version
Summary Plot	50% of students' scores were below 500.
Histogram	The majority of scores ranged between 416 and 459. The peak occurred at 455, with an average score of 452.
Box Plot	The box plot illustrates the presence of numerous low extreme values as well as high extreme values.
CDF	The data does not follow a normal distribution pattern.

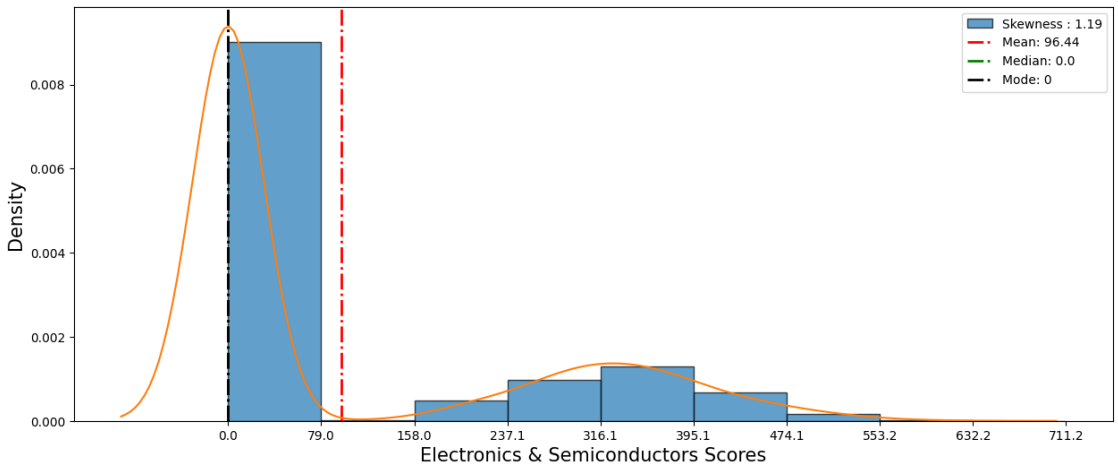
Observations

1.10 Electronics & Semiconductors

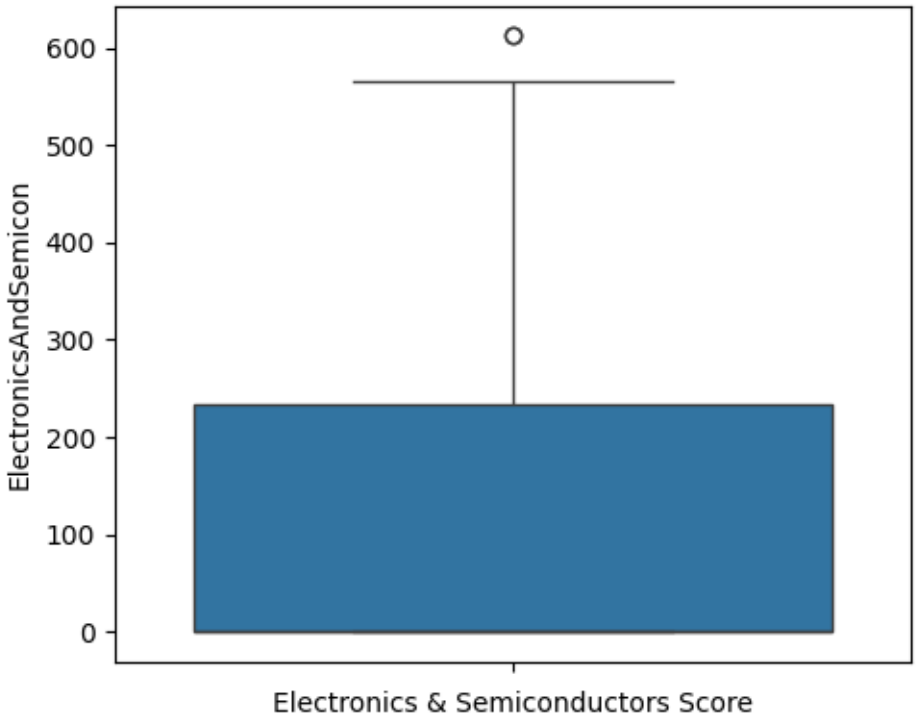
[ ]:
```
# Summary Plot

plt.figure(figsize=(5,4))
df1['ElectronicsAndSemicon'].describe()[1:].plot(alpha = 0.8,
                                  marker = 'D', markersize = 8)
plt.title('Summary Statistics for Electronics & Semiconductors')
plt.xlabel('Statistical Measures')
plt.tight_layout()
plt.show()

# Histogram

bins = np.arange(df1['ElectronicsAndSemicon'].min(),␣
 ↪df1['ElectronicsAndSemicon'].max()+df1['ElectronicsAndSemicon'].std(),
                 df1['ElectronicsAndSemicon'].std()/2)
plt.figure(figsize = (15,6))
plt.hist(df1['ElectronicsAndSemicon'], ec = 'k',
        bins = bins,
        label = f"Skewness : {round(df1['ElectronicsAndSemicon'].skew(),2)}",
        alpha = 0.7,
        density = True)
plt.xticks(bins)
plt.xlabel('Electronics & Semiconductors Scores', size = 15)
plt.ylabel('Density', size = 15)

plt.axvline(df1['ElectronicsAndSemicon'].mean(), label = f"Mean:␣
 ↪{round(df1['ElectronicsAndSemicon'].mean(),2)}"
            , linestyle = '-.',
           color = 'red', linewidth = 2)
plt.axvline(df1['ElectronicsAndSemicon'].median(), label = f"Median:␣
 ↪{round(df1['ElectronicsAndSemicon'].median(),2)}"
            , linestyle = '-.',
           color = 'green', linewidth = 2)
plt.axvline(df1['ElectronicsAndSemicon'].mode()[0], label = f"Mode:␣
 ↪{round(df1['ElectronicsAndSemicon'].mode()[0],2)}"
```

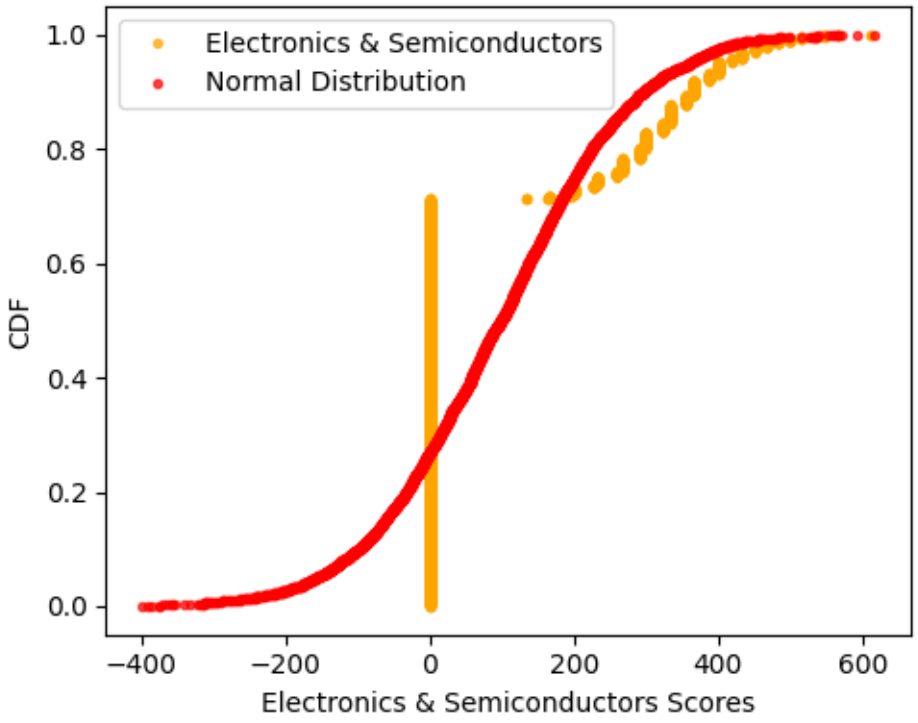
```
            , linestyle = '-.',
           color = 'k', linewidth = 2)
sns.kdeplot(df1['ElectronicsAndSemicon'])
plt.legend()
plt.show()

# Box Plot

plt.figure(figsize=(5,4))
sns.boxplot(df1['ElectronicsAndSemicon'])
plt.xlabel('Electronics & Semiconductors Score')
plt.tight_layout()
plt.show()

# CDF

plt.figure(figsize=(5,4))
x_cp, y_cp = cdf(df1['ElectronicsAndSemicon'])
x_sample_cp , y_sample_cp = \
cdf(np.random.normal(df1['ElectronicsAndSemicon'].mean(),␣
 ↪df1['ElectronicsAndSemicon'].std(), size = \
                     len(df1['ElectronicsAndSemicon'])))
plt.plot(x_cp, y_cp, linestyle = 'None',
        marker = '.', color = 'orange',
         alpha = 0.7, label = 'Electronics & Semiconductors')
plt.plot(x_sample_cp, y_sample_cp, linestyle = 'None',
        marker ='.', color = 'red',
        alpha = 0.7, label = 'Normal Distribution')
plt.xlabel('Electronics & Semiconductors Scores')
plt.ylabel('CDF')
plt.legend()
plt.tight_layout()
plt.show()
```






```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:640:
FutureWarning: SeriesGroupBy.grouper is deprecated and will be removed in a
future version of pandas.
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```


600
500
400
300
200
100
0
ElectronicsAndSemicon
Electronics & Semiconductors Score


Electronics & Semiconductors
Normal Distribution
1.0
0.8
0.6
0.4
0.2
0.0
CDF
−400
−200
0
200
400
600
Electronics & Semiconductors Scores

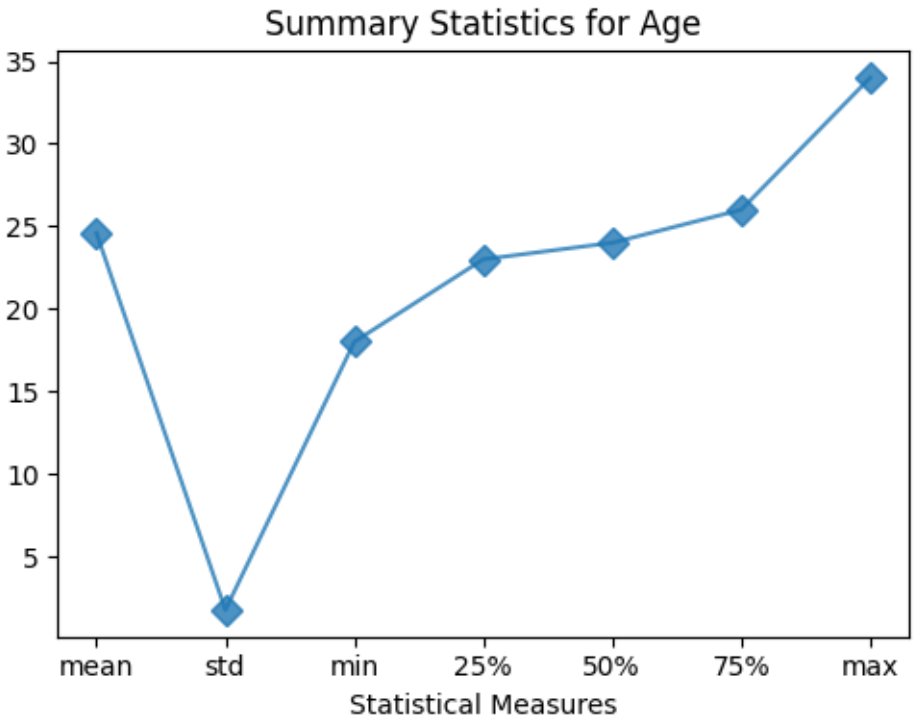
Conclusions	Inferences
Summary Plot	About 75% of students scored less than 250.
Histogram	Most scores fell between 0 and 79. The highest number of students scored 0, with an average score of 96.
Box Plot	The lowest score is equal to the median of the dataset.
CDF	The data does not conform to a normal distribution pattern.

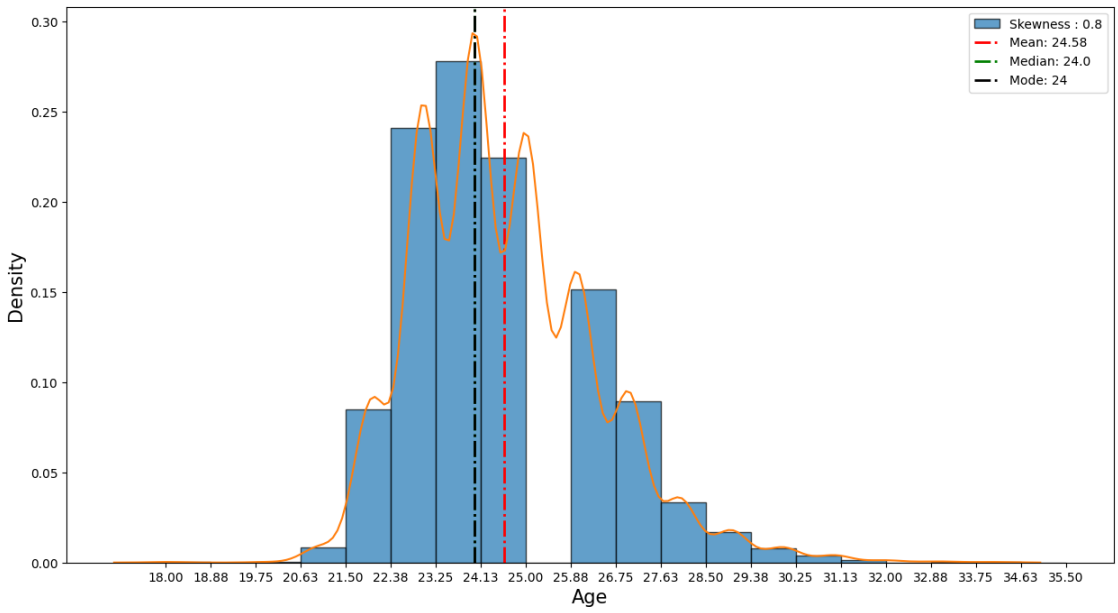
Observations

1.11 Age

[ ]:
```
# Summary Plot

plt.figure(figsize=(5,4))
df1['Age'].describe()[1:].plot(alpha = 0.8,
                                marker = 'D', markersize = 8)
plt.title('Summary Statistics for Age')
plt.xlabel('Statistical Measures')
plt.tight_layout()
plt.show()

# Histogram

bins = np.arange(df1['Age'].min(), df1['Age'].max()+df1['Age'].std(),
                 df1['Age'].std()/2)
plt.figure(figsize = (15,8))
plt.hist(df1['Age'], ec = 'k',
        bins = bins,
        label = f"Skewness : {round(df1['Age'].skew(),2)}",
        alpha = 0.7,
        density = True)
plt.xticks(bins)
plt.xlabel('Age', size = 15)
plt.ylabel('Density', size = 15)

plt.axvline(df1['Age'].mean(), label = f"Mean: {round(df1['Age'].mean(),2)}"
            , linestyle = '-.',
           color = 'red', linewidth = 2)
plt.axvline(df1['Age'].median(), label = f"Median: {round(df1['Age'].
 ↪median(),2)}"
            , linestyle = '-.',
           color = 'green', linewidth = 2)
plt.axvline(df1['Age'].mode()[0], label = f"Mode: {round(df1['Age'].
 ↪mode()[0],2)}"
            , linestyle = '-.',
           color = 'k', linewidth = 2)
sns.kdeplot(df1['Age'])
```

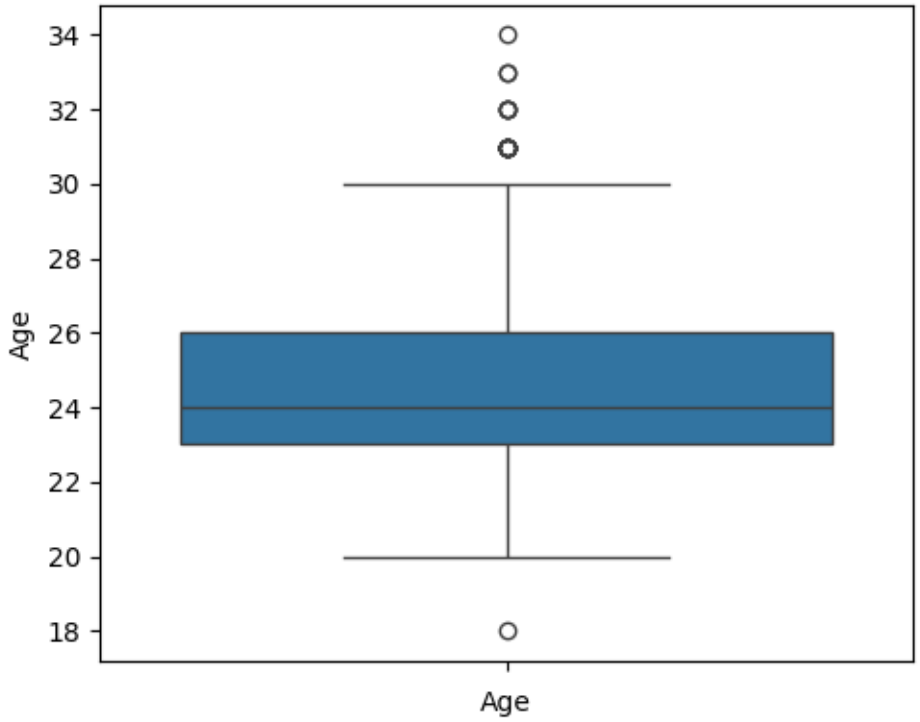
```
plt.legend()
plt.show()

# Box Plot

plt.figure(figsize=(5,4))
sns.boxplot(df1['Age'])
plt.xlabel('Age')
plt.tight_layout()
plt.show()

# CDF

plt.figure(figsize=(5,4))
x_cp, y_cp = cdf(df1['Age'])
x_sample_cp , y_sample_cp = \
cdf(np.random.normal(df1['Age'].mean(), df1['Age'].std(), size = \
                     len(df1['Age'])))
plt.plot(x_cp, y_cp, linestyle = 'None',
        marker = '.', color = 'orange',
         alpha = 0.7, label = 'Age')
plt.plot(x_sample_cp, y_sample_cp, linestyle = 'None',
        marker ='.', color = 'red',
        alpha = 0.7, label = 'Normal Distribution')
plt.xlabel('Age')
plt.ylabel('CDF')
plt.legend()
plt.tight_layout()
plt.show()
```

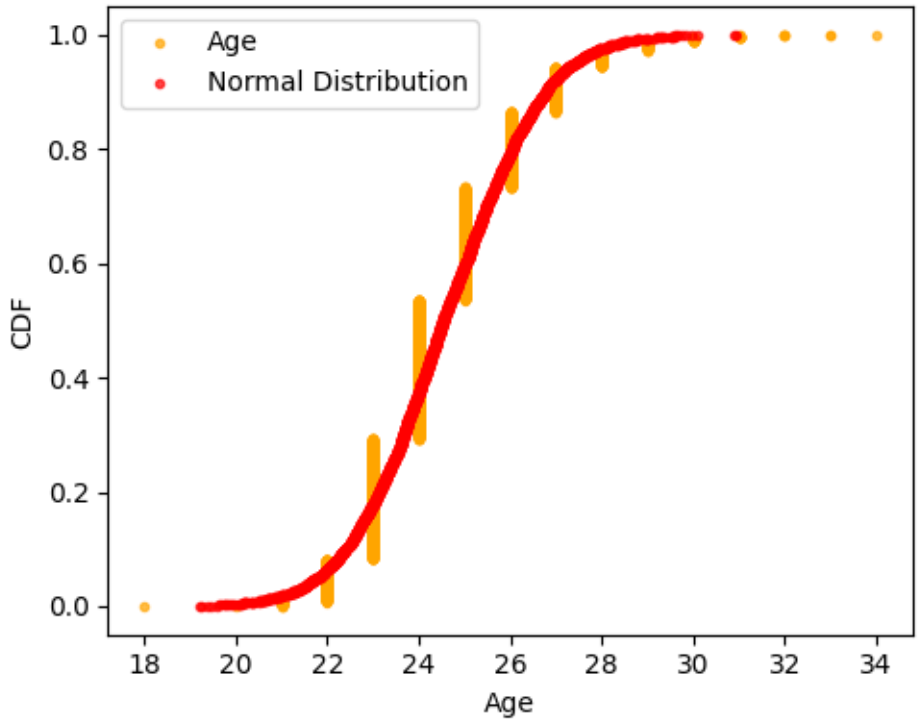





```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:640:
FutureWarning: SeriesGroupBy.grouper is deprecated and will be removed in a
future version of pandas.
```

```
positions = grouped.grouper.result_index.to_numpy(dtype=float)
```





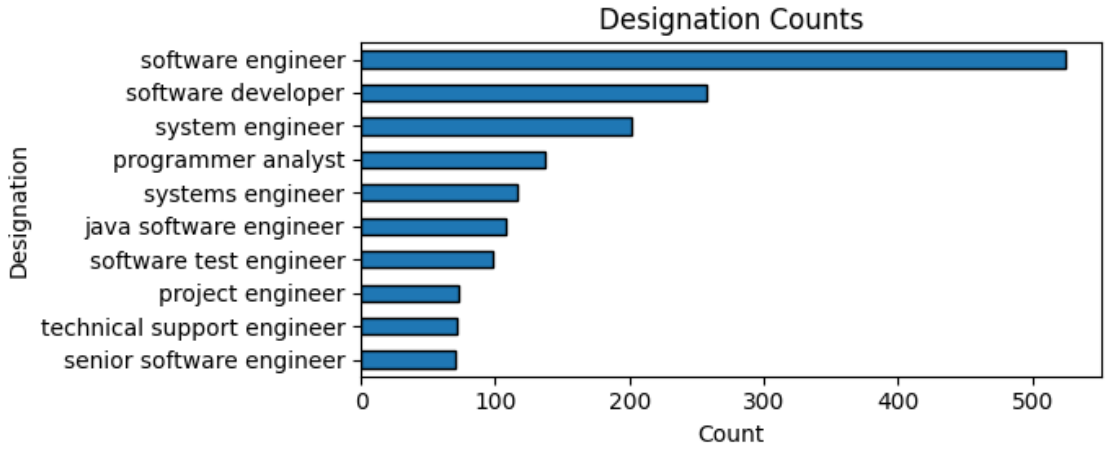
Conclusions	Inferences
Summary Plot	Approximately 75% of students are under 26 years old.
Histogram	The majority of students' ages ranged between 22 and 25. The mean, median, and mode ages are approximately 25.
Box Plot	The box plot indicates the presence of 4 students with very high ages and one with a very low age compared to other data points.
CDF	The age data does not follow a normal distribution pattern.

Observations

1.3.2 2. Categorical Features

2.1 Designation

```
[ ]: df1['Designation'].value_counts()[1:].sort_values(
         ascending=True
     ).plot(
         kind='barh',
         title='Designation Counts',
         figsize=(7, 3),
         ec='k'
     )
     plt.ylabel('Designation')
     plt.xlabel('Count')
     plt.tight_layout()
     plt.show()
```

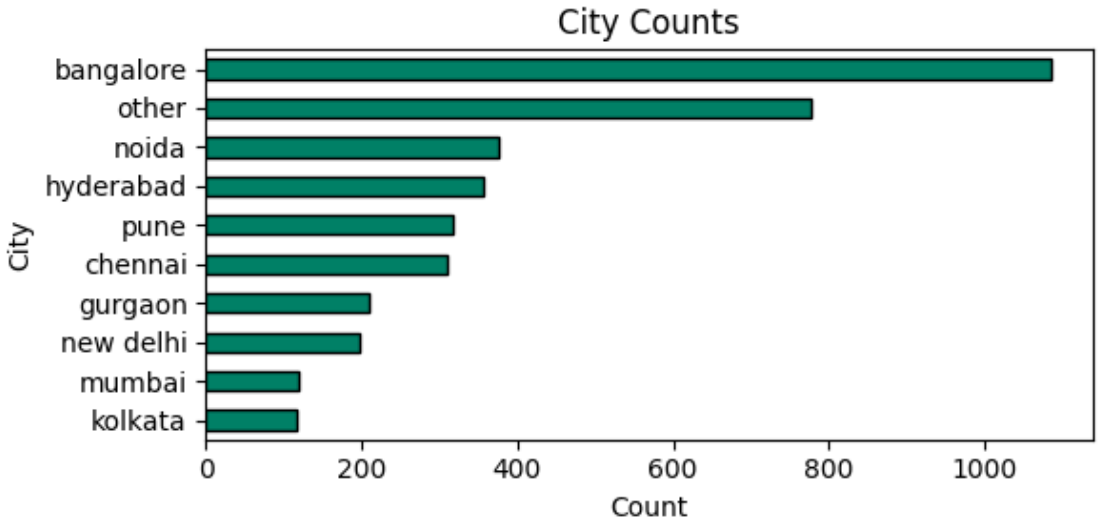


Observations Software engineer is the most common desgination of all, followed by system engineer and software developer.

NOTE : This graph contains the most common designations. There exists *OTHER* category too.

2.2 JobCity

```
[ ]: df1['JobCity'].value_counts().sort_values(ascending=True).plot(
         kind='barh',
         cmap='summer',
         title='City Counts',
         figsize=(6,3),
         ec='k'
     )
     plt.ylabel('City')
     plt.xlabel('Count')
     plt.tight_layout()
     plt.show()
```

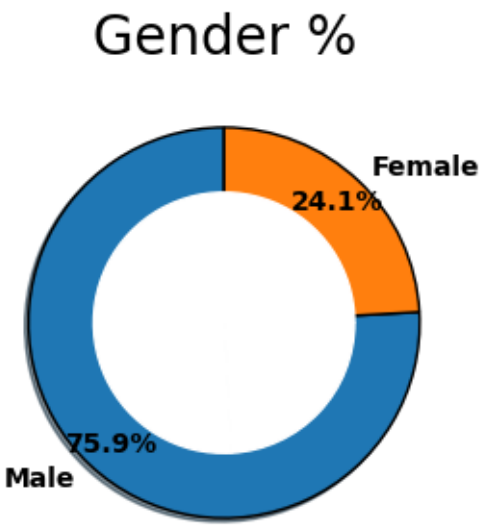


Observations The most favourable city for job placements is bangalore, followed by Noid, Hyderabad and pune. Mumbai and kolkata being least favourable.

2.3 Gender

```
[ ]: plt.figure(figsize=(3,3))
     plt.pie(df1['Gender'].value_counts().tolist(),
             labels=df1['Gender'].value_counts().index,
             autopct='%1.1f%%',
             radius=1.5,
             wedgeprops={'edgecolor': 'k'},
             textprops={'fontsize': 10, 'fontweight': 'bold'},
             shadow=True,
```

```
        startangle=90,
        pctdistance=0.85)
plt.pie(df1['Gender'].value_counts().tolist(),
        colors=['white'],
        wedgeprops={'edgecolor': 'white'},
        radius=1)
plt.title('Gender %', pad=40, size=20)
plt.tight_layout()
plt.show()
```

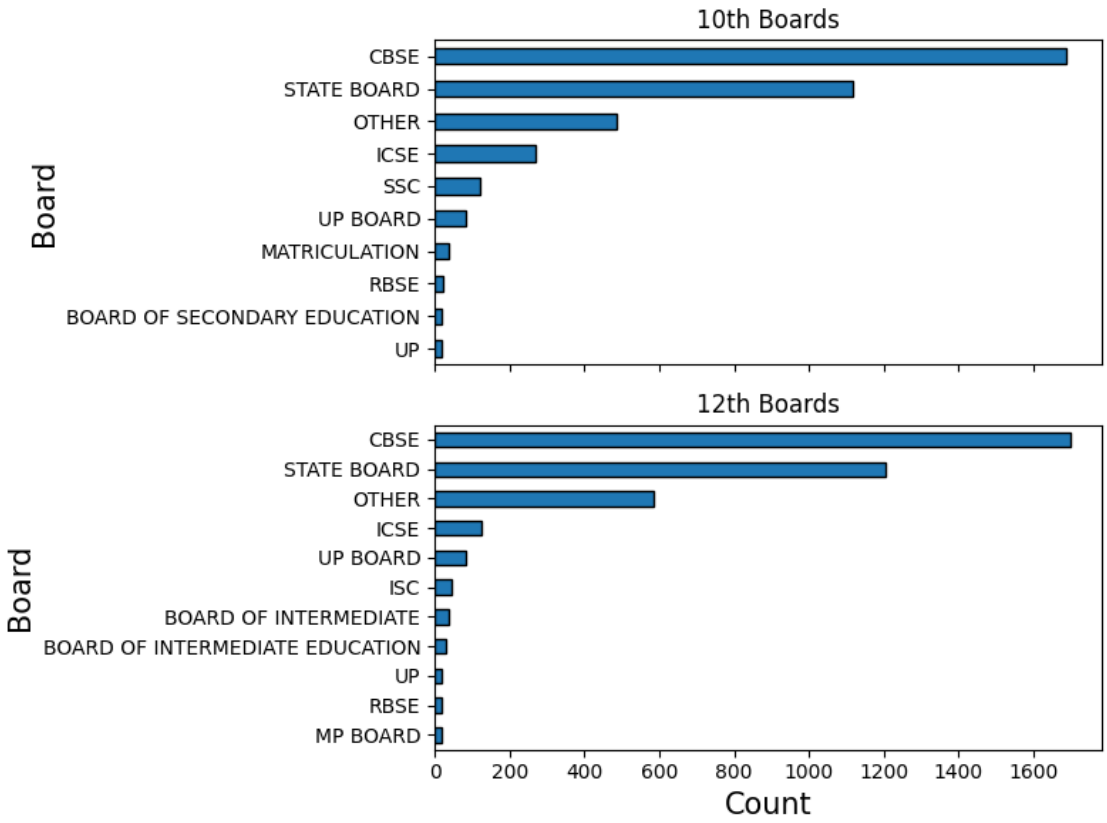


Observations The dataset is not balanced in terms of gender as the population of Male is really larger as compared to the female one.

2.4 10board & 12board

[ ]:
```
fig, ax = plt.subplots(2, 1, figsize=(8, 6), sharex=True)

df1['10board'].str.upper().value_counts().sort_values(ascending=True).plot(
    kind='barh',
    ax=ax[0],
    ec='k',
    title='10th Boards'
)
ax[0].set_ylabel('Board', size=15)

df1['12board'].str.upper().value_counts().sort_values(ascending=True).plot(
    kind='barh',
    ax=ax[1],
    ec='k',
```

```
    title='12th Boards'
)
ax[1].set_ylabel('Board', size=15)
ax[1].set_xlabel('Count', size=15)

plt.tight_layout()
plt.show()
```

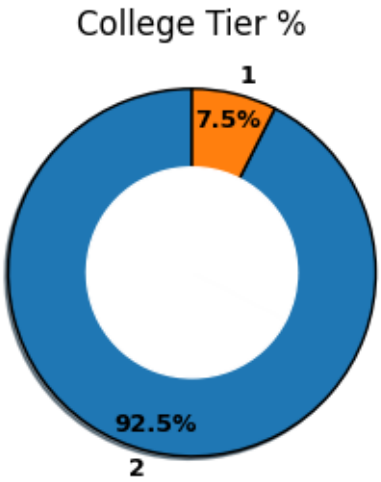


Observations CBSE is the most common school board for both 12th and 10th.

2.5 CollegeTier

[ ]:
```
plt.figure(figsize=(3,3))
plt.pie(df1['CollegeTier'].value_counts().tolist(), labels = df1['CollegeTier'].
 ↪value_counts().index,
       autopct = '%1.1f%%',
       radius = 1.75,
       wedgeprops = {'edgecolor':'k'},
       textprops = {'fontsize':9,'fontweight':'bold'},
       shadow = True,
       startangle = 90,
```

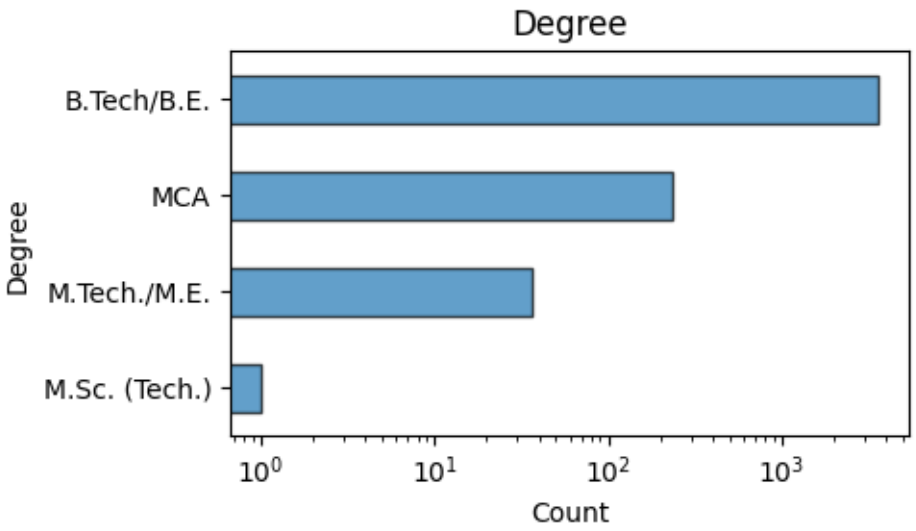
```
       pctdistance = 0.85)
plt.pie(df1['CollegeTier'].value_counts().tolist(), colors = ['white'],
        wedgeprops = {'edgecolor':'white'},
       radius = 1)
plt.title('College Tier %',pad = 40, size = 12)
plt.margins(0.02)
plt.tight_layout()
plt.show()
```



Observations Almost all the college belongs to Tier 1 only with a percentage of 92.5

2.6 Degree

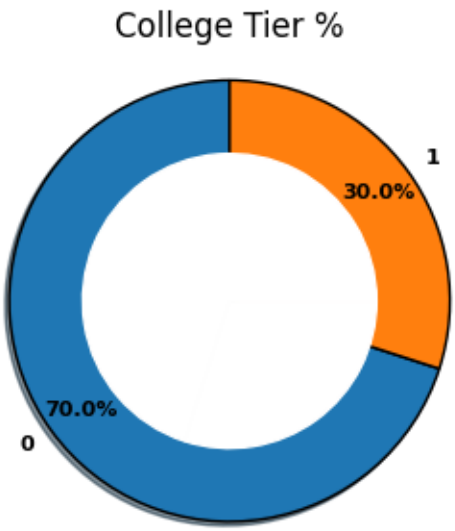
[ ]:
```
df1['Degree'].value_counts().sort_values(ascending=True).plot(
    kind='barh',
    title='Degree',
    figsize=(5, 3),
    ec='k',
    alpha=0.7
)
plt.ylabel('Degree')
plt.xlabel('Count')
plt.xscale('log')
plt.tight_layout()
plt.show()
```



Observations Most of the students have done their graduation in B.Tech and there are very less students from M.Sc(Tech)

2.7 CollegeCityTier

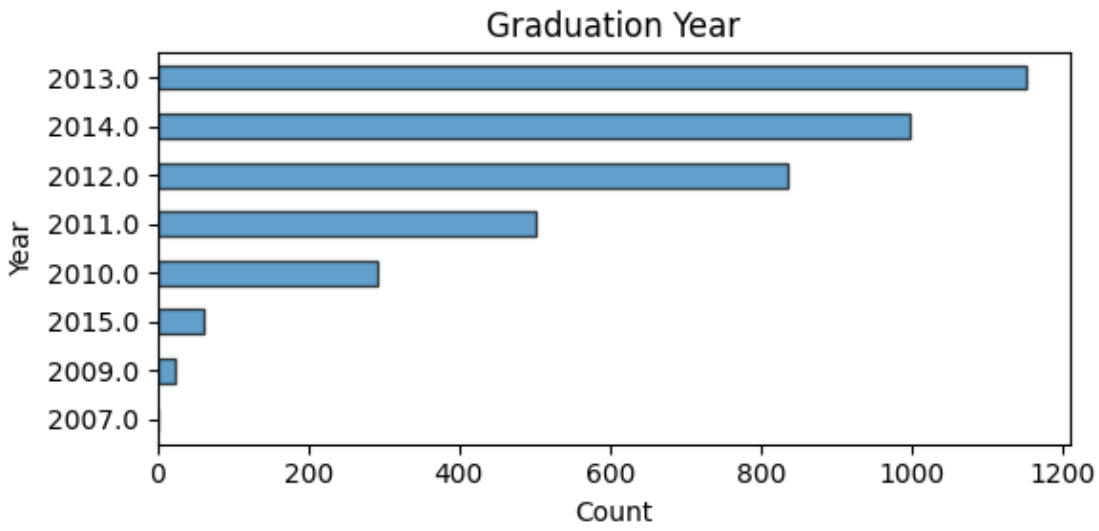
[ ]:
```
plt.figure(figsize=(3,3))
plt.pie(df1['CollegeCityTier'].value_counts().tolist(), labels =␣
 ↪df1['CollegeCityTier'].value_counts().index,
       autopct = '%1.1f%%',
       radius = 1.5,
       wedgeprops = {'edgecolor':'k'},
       textprops = {'fontsize':8,'fontweight':'bold'},
       shadow = True,
       startangle = 90,
       pctdistance = 0.84)
plt.pie(df1['CollegeCityTier'].value_counts().tolist(), colors = ['white'],
        wedgeprops = {'edgecolor':'white'},
       radius = 1)
plt.title('College Tier %',pad = 30, size = 12)
plt.margins(0.02)
plt.tight_layout()
plt.show()
```



Observations Majority of the colleges are form Tier 0 city.

2.8 GraduationYear

[ ]:
```
df1['GraduationYear'].value_counts().sort_values(ascending=True).plot(
    kind='barh',
    title='Graduation Year',
    figsize=(6, 3),
    ec='k',
    alpha=0.7
)
plt.ylabel('Year')
plt.xlabel('Count')
plt.tight_layout()
plt.show()
```



Observations

Maximum number of students were graduated in 2013, followed by the year 2014 and 2012.

1.3.3 Removing Outliers

[ ]:
```
def outlier_treatment(datacolumn):
    sorted(datacolumn)
    Q1,Q3 = np.percentile(datacolumn , [25,75])
    IQR = Q3 - Q1
    lower_range = Q1 - (1.5 * IQR)
    upper_range = Q3 + (1.5 * IQR)
    return lower_range,upper_range
```

[ ]:
```
df1.columns
```

[ ]:
```
Index(['Salary', 'DOJ', 'DOL', 'Designation', 'JobCity', 'Gender', 'DOB',
       '10percentage', '10board', '12graduation', '12percentage', '12board',
       'CollegeTier', 'Degree', 'Specialization', 'collegeGPA',
       'CollegeCityTier', 'CollegeState', 'GraduationYear', 'English',
       'Logical', 'Quant', 'Domain', 'ComputerProgramming',
       'ElectronicsAndSemicon', 'ComputerScience', 'conscientiousness',
       'agreeableness', 'extraversion', 'nueroticism', 'openess_to_experience',
       'Age', 'Tenure'],
      dtype='object')
```

[ ]:
```
columns = ['Salary','10percentage','12percentage','English',
           'Logical','Quant','Domain', 'ComputerProgramming',
       'ElectronicsAndSemicon', 'ComputerScience', 'conscientiousness',
```

```
       'agreeableness', 'extraversion', 'nueroticism', 'openess_to_experience',
       'Age', 'Tenure']
df2 = df1.copy()
```

[ ]:
```
for cols in columns:
    lowerbound, upperbound = outlier_treatment(df2[cols])

    df2 = df2.drop(df2[(df2[cols] < lowerbound) | (df2[cols] > upperbound)].
 ↪index)
```

[ ]:
```
print(f'Number of observation with outliers: {df1.shape[0]}')
print(f'Number of observations without outliers: {df2.shape[0]}')
```

```
Number of observation with outliers: 3864
Number of observations without outliers: 2490
```

1.4 Bivariate Analysis

1.4.1 1. Barplots

1.1 Average Salary for each Designation

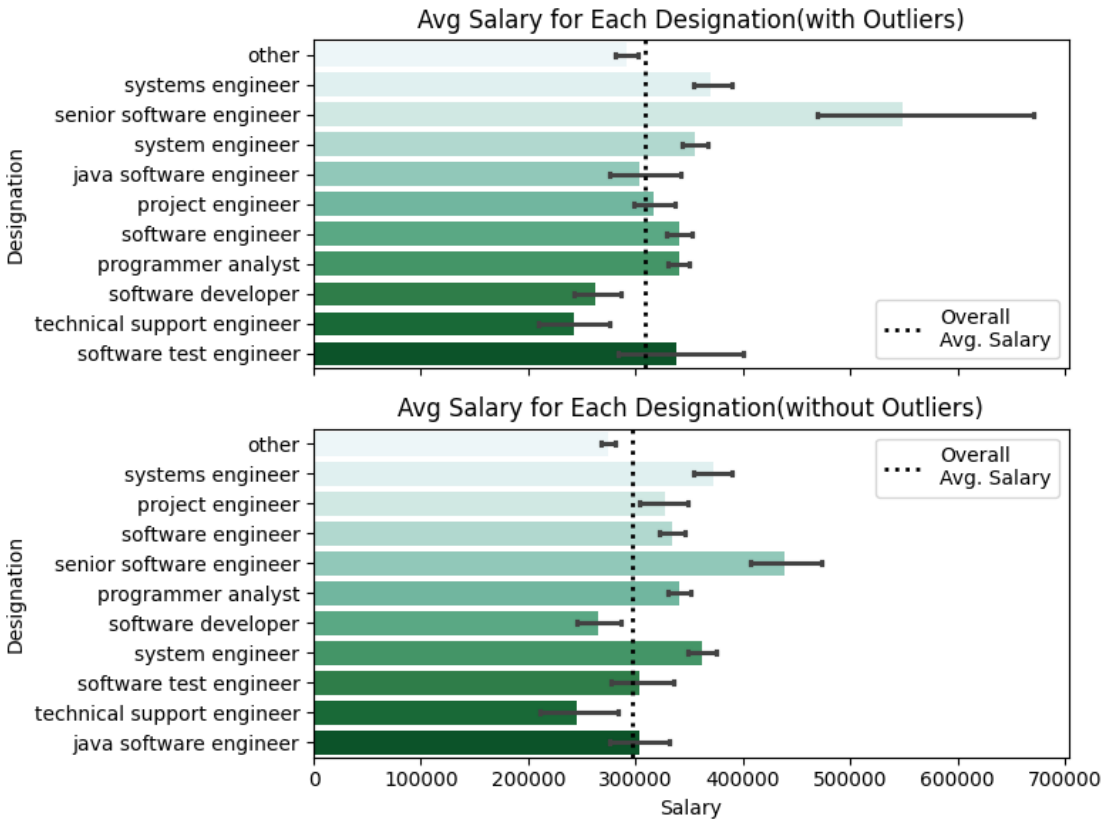
[ ]:
```
fig, ax = plt.subplots(2, 1, figsize = (8,6), sharex = True)
sns.barplot(x = 'Salary', y = 'Designation',
            data = df1,
            palette = 'BuGn',
            capsize = 0.1,
            ax = ax[0])
ax[0].axvline(df1['Salary'].mean(), color = 'k',
            linestyle = ':',
            linewidth = 2, label = 'Overall\nAvg. Salary')
ax[0].set_title('Avg Salary for Each Designation(with Outliers)')
ax[0].legend()
ax[0].set_xlabel('')


sns.barplot(x = 'Salary', y = 'Designation',
            data = df2,
            palette = 'BuGn',
            capsize = 0.1,
            ax = ax[1])
ax[1].axvline(df2['Salary'].mean(), color = 'k',
            linestyle = ':',
            linewidth = 2, label = 'Overall\nAvg. Salary')
ax[1].set_title('Avg Salary for Each Designation(without Outliers)')
ax[1].legend()
ax[1].set_xlabel('Salary')

plt.tight_layout()
```

```
plt.show()
```

```
<ipython-input-66-a02ae9c7fbad>:2: FutureWarning:

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

  sns.barplot(x = 'Salary', y = 'Designation',
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
```

```
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
<ipython-input-66-a02ae9c7fbad>:15: FutureWarning:

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

  sns.barplot(x = 'Salary', y = 'Designation',
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
```

```
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```



Observations Bar plot shows the maximum salary for each Designation. Senior Software Engineer has the highest salary but they also has the maximum standard deviation in their salary. There are only two designations namely, software developer and technical support engineer who has salary lower than average salary.

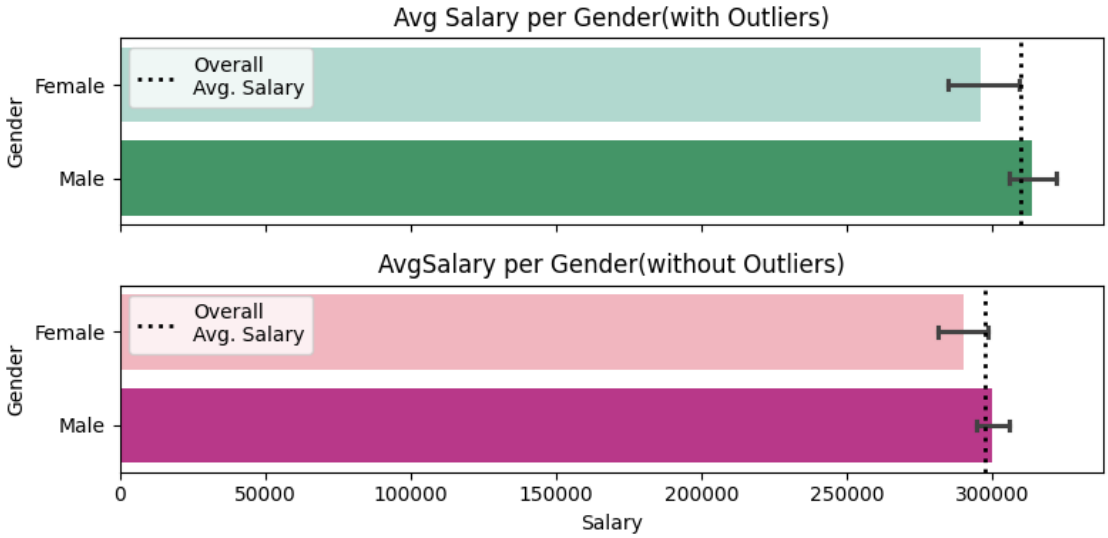
1.2 Average Salary for each Gender

[ ]:
```
fig, ax = plt.subplots(2, 1, figsize = (8,4), sharex = True)
sns.barplot(x = 'Salary', y = 'Gender',
            data = df1,
            palette = 'BuGn',
            capsize = 0.1,
            ax = ax[0])
ax[0].axvline(df1['Salary'].mean(), color = 'k',
            linestyle = ':',
            linewidth = 2, label = 'Overall\nAvg. Salary')
ax[0].set_title('Avg Salary per Gender(with Outliers)')
ax[0].legend()
ax[0].set_xlabel('')
```

```
sns.barplot(x = 'Salary', y = 'Gender',
            data = df2,
            palette = 'RdPu',
            capsize = 0.1,
            ax = ax[1])
ax[1].axvline(df2['Salary'].mean(), color = 'k',
            linestyle = ':',
            linewidth = 2, label = 'Overall\nAvg. Salary')
ax[1].set_title('AvgSalary per Gender(without Outliers)')
ax[1].legend()
ax[1].set_xlabel('Salary')


plt.tight_layout()
plt.show()
```

```
<ipython-input-67-46f5bd1b91dc>:2: FutureWarning:

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

  sns.barplot(x = 'Salary', y = 'Gender',
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
<ipython-input-67-46f5bd1b91dc>:15: FutureWarning:

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

  sns.barplot(x = 'Salary', y = 'Gender',
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a length-1 tuple
```

```
to get_group in a future version of pandas. Pass `(name,)` instead of `name` to
silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```



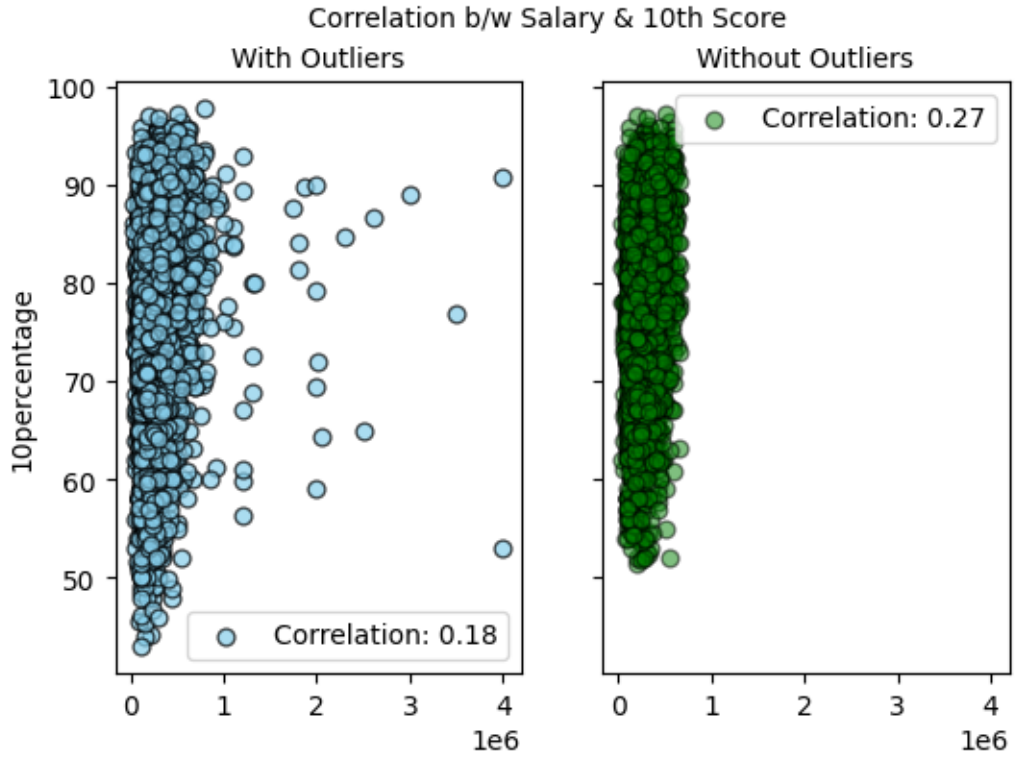
Observations The average salary for both male and female is approximately equal and it implies that there was no gender bias in terms of salary. It is also plausible to say that Female's get salary below the overall average salary.

1.4.2 2. Scatter Plots

2.1 Salary & 10th score

[ ]:
```
fig, ax = plt.subplots(1, 2, figsize = (6,4), sharex = True, sharey = True)

ax[0].scatter(df1['Salary'],df1['10percentage'],
              ec = 'k',
              color = 'skyblue',
              alpha = 0.7,
              s = 40,
              label = f"Correlation: {round(df1[['Salary','10percentage']].
 ↪corr().iloc[1,0],2)}"
              )
ax[0].set_ylabel('10percentage')
ax[0].set_title('With Outliers', size=10)
ax[0].legend()


ax[1].scatter(df2['Salary'],df2['10percentage'],
              ec = 'k',
```

```
                color = 'green',
                alpha = 0.5,
                s = 40,
                label = f"Correlation: {round(df2[['Salary','10percentage']].
 ↪corr().iloc[1,0],2)}"
                )
ax[1].set_title('Without Outliers', size=10)
ax[1].legend()


fig.suptitle('Correlation b/w Salary & 10th Score', size = 10)
plt.show()
```



Observations There does not exist any correlation between Salary and 10th scores.

2.2 Salary & 12th score

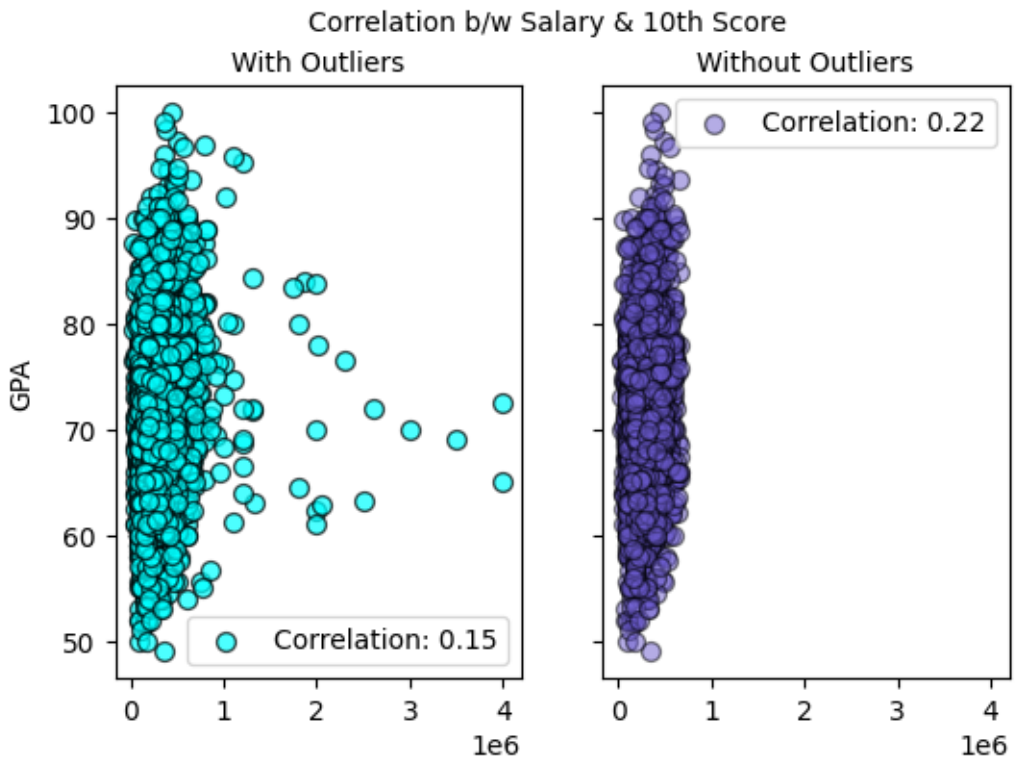
[ ]:
```
fig, ax = plt.subplots(1, 2, figsize = (6,4), sharex = True, sharey = True)

ax[0].scatter(df1['Salary'],df1['12percentage'],
                ec = 'k',
                color = 'red',
```

```
                alpha = 0.7,
                s = 50,
                label = f"Correlation: {round(df1[['Salary','12percentage']].
 ↪corr().iloc[1,0],2)}"
                )
ax[0].set_ylabel('12percentage')
ax[0].set_title('With Outliers', size=10)
ax[0].legend()


ax[1].scatter(df2['Salary'],df2['12percentage'],
                ec = 'k',
                color = 'brown',
                alpha = 0.5,
                s = 50,
                label = f"Correlation: {round(df2[['Salary','12percentage']].
 ↪corr().iloc[1,0],2)}"
                )
ax[1].set_title('Without Outliers', size=10)
ax[1].legend()

fig.suptitle('Correlation b/w Salary & 10th Score', size = 10)
plt.show()
```

Observations There does not exist any correlation between Salary and 10th scores.

2.3 Salary & CollegeGPA score

[ ]:
```
fig, ax = plt.subplots(1, 2, figsize = (6,4), sharex = True, sharey = True)

ax[0].scatter(df1['Salary'],df1['collegeGPA'],
              ec = 'k',
              color = 'cyan',
              alpha = 0.7,
              s = 50,
              label = f"Correlation: {round(df1[['Salary','collegeGPA']].
 ↪corr().iloc[1,0],2)}"
              )
ax[0].set_ylabel('GPA')
ax[0].set_title('With Outliers', size=10)
ax[0].legend()


ax[1].scatter(df2['Salary'],df2['collegeGPA'],
              ec = 'k',
              color = 'slateblue',
              alpha = 0.5,
              s = 50,
              label = f"Correlation: {round(df2[['Salary','collegeGPA']].
 ↪corr().iloc[1,0],2)}"
              )
ax[1].set_title('Without Outliers', size=10)
ax[1].legend()

fig.suptitle('Correlation b/w Salary & 10th Score', size = 10)
plt.show()
```

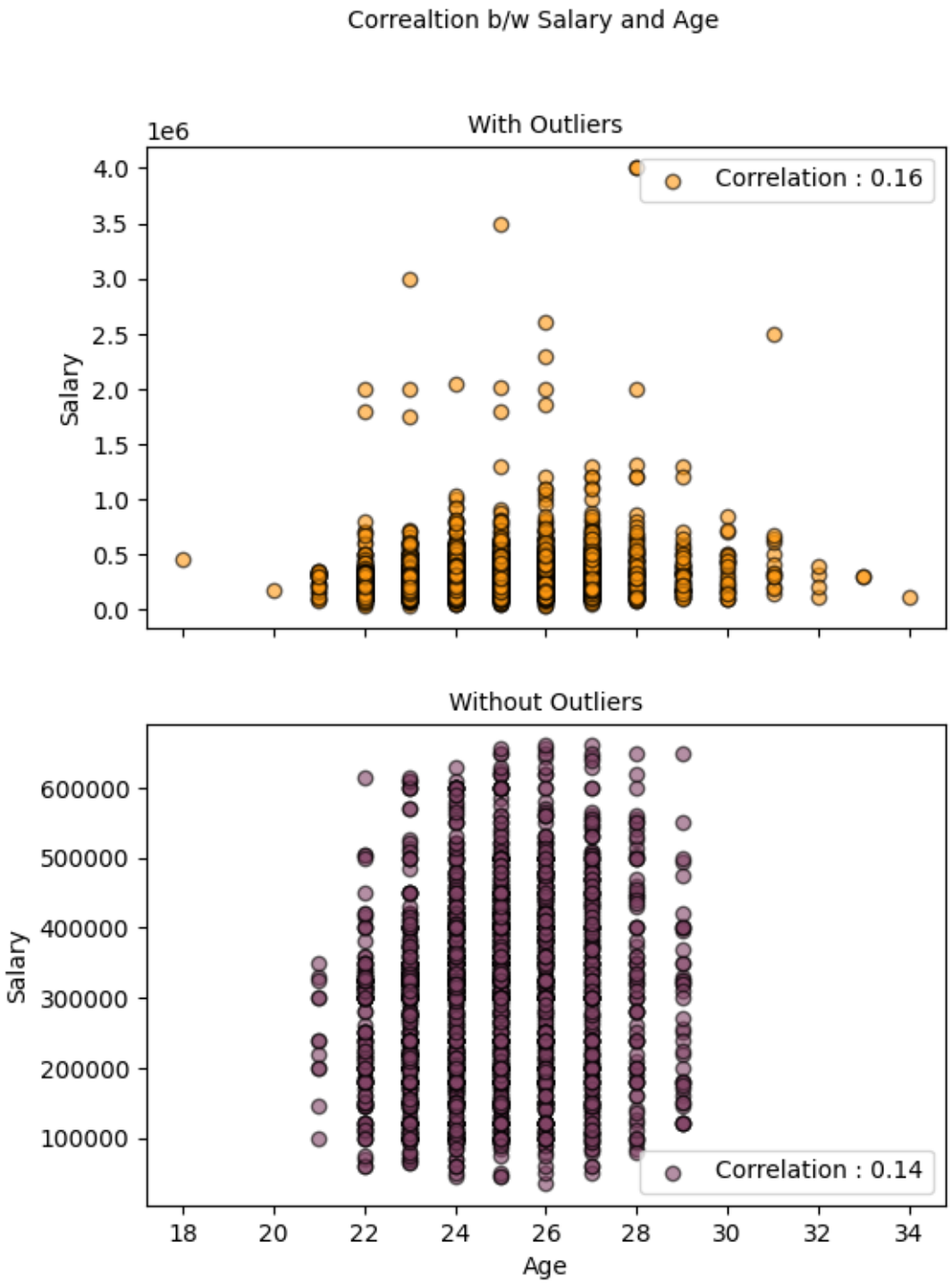


Observations There does not exist any correlation between Salary and 10th scores.

2.4 Salary & Age

[ ]:
```
fig, ax = plt.subplots(2, 1, figsize = (6,8), sharex  = True)
ax[0].scatter(df1['Age'], df1['Salary'],
              ec = 'k',
              color = '#ff9911',
              alpha = 0.6,
              label = f"Correlation : {round(df1[['Age','Salary']].corr().
 ↪iloc[1,0],2)}"
              )
ax[0].legend()
ax[0].set_ylabel('Salary')
ax[0].set_title('With Outliers' , size=10)


ax[1].scatter(df2['Age'], df2['Salary'],
              ec = 'k',
              color = '#834567',
              alpha = 0.6,
```

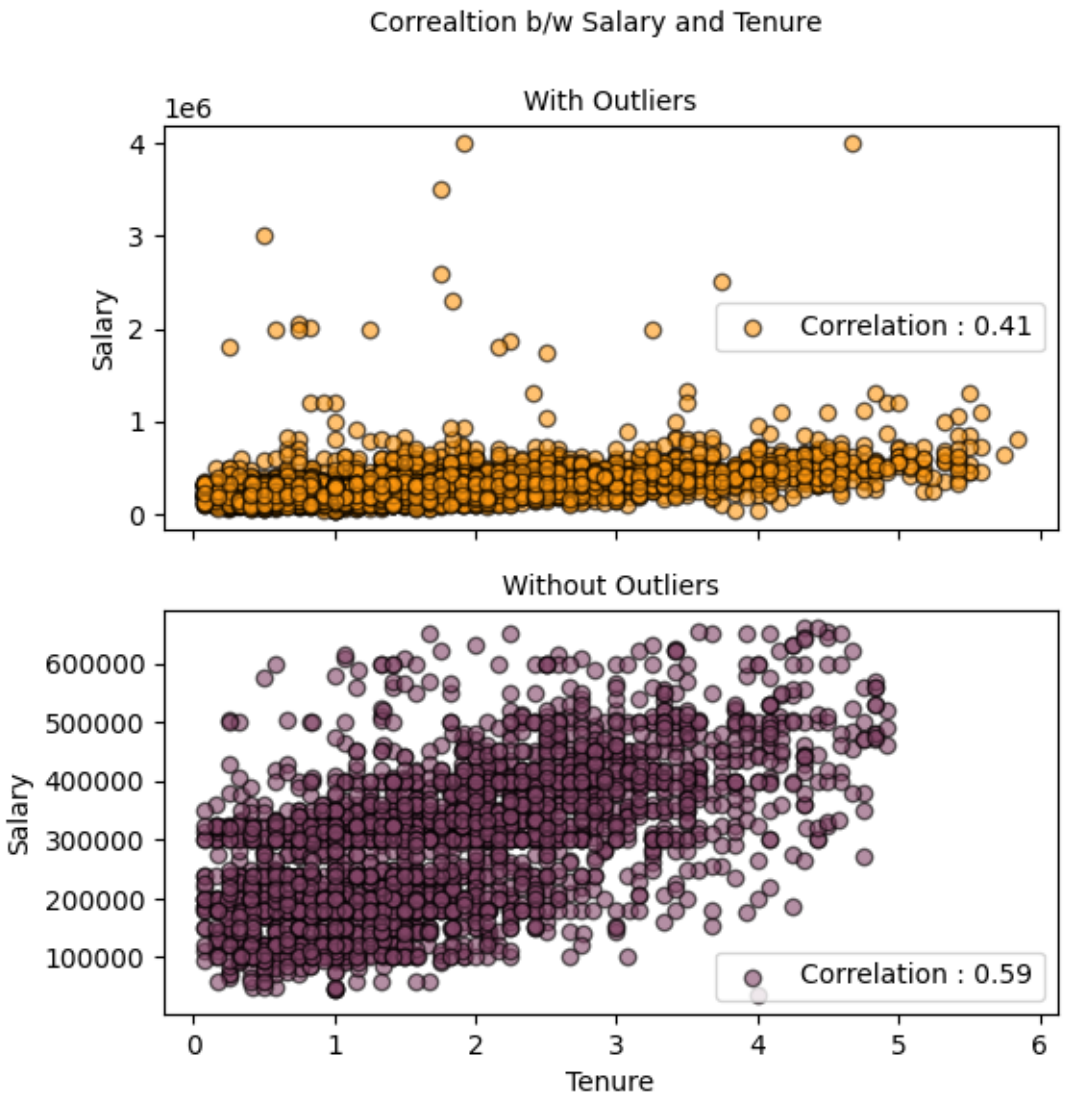
```
            label = f"Correlation : {round(df2[['Age','Salary']].corr().
 ↪iloc[1,0],2)}"
            )
ax[1].legend()
ax[1].set_ylabel('Salary')
ax[1].set_title('Without Outliers' , size=10)
ax[1].set_xlabel('Age')

fig.suptitle('Correaltion b/w Salary and Age', size = 10)

plt.show()
```



Observations After removing the outliers, it is evident that the salary and age are not related to each other.

2.5 Salary & Tenure

[ ]:
```
fig, ax = plt.subplots(2, 1, figsize = (6,6), sharex  = True)
ax[0].scatter(df1['Tenure'], df1['Salary'],
              ec = 'k',
              color = '#ff9911',
              alpha = 0.6,
              label = f"Correlation : {round(df1[['Tenure','Salary']].corr().
 ↪iloc[1,0],2)}"
              )
ax[0].legend()
ax[0].set_ylabel('Salary')
ax[0].set_title('With Outliers' , size=10)


ax[1].scatter(df2['Tenure'], df2['Salary'],
              ec = 'k',
              color = '#834567',
              alpha = 0.6,
              label = f"Correlation : {round(df2[['Tenure','Salary']].corr().
 ↪iloc[1,0],2)}"
              )
ax[1].legend()
ax[1].set_ylabel('Salary')
ax[1].set_title('Without Outliers' , size=10)
ax[1].set_xlabel('Tenure')

fig.suptitle('Correaltion b/w Salary and Tenure', size = 10)

plt.show()
```



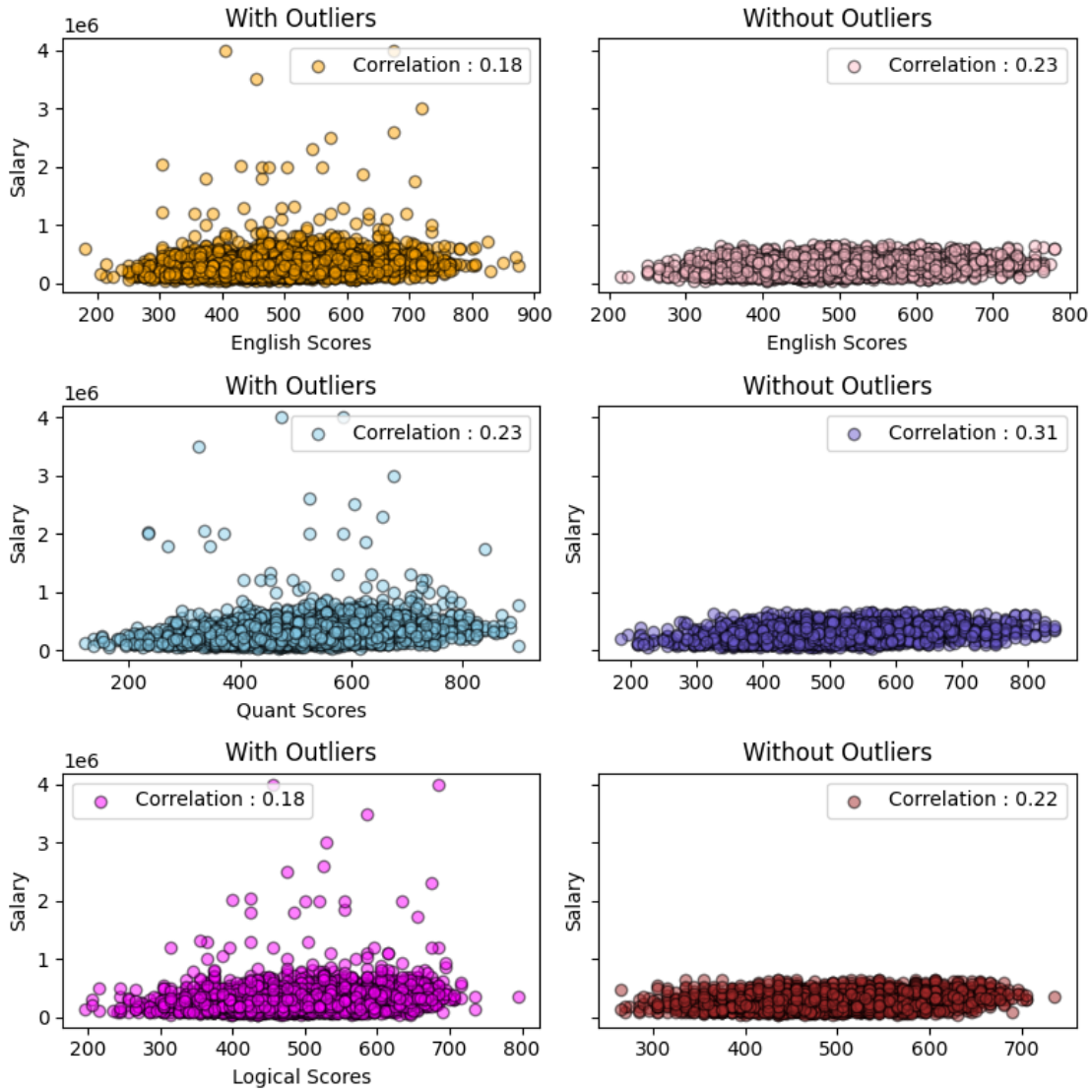
Observations After removing the outliers, it is evident that salary gets about 50% of increment as tenure increase as there is a positive correlation of 0.60.

2.6 Salary with English, Quants, Logical

[ ]:
```
fig, ax = plt.subplots(3, 2, figsize = (8,8), sharey = True)
ax[0,0].scatter(df1['English'],df1['Salary'],
                ec = 'k',
                color = 'orange',
                alpha = 0.5,
                label = f"Correlation : {round(df1[['English','Salary']].corr().
 ↪iloc[1,0],2)}"
                )
```

```
ax[0,0].set_ylabel('Salary')
ax[0,0].set_xlabel('English Scores')
ax[0,0].set_title('With Outliers')
ax[0,0].legend()


ax[0,1].scatter(df2['English'],df2['Salary'],
                ec = 'k',
                color = 'pink',
                alpha = 0.5,
                label = f"Correlation : {round(df2[['English','Salary']].corr().
 ↪iloc[1,0],2)}"
                )
ax[0,1].set_title('Without Outliers')
ax[0,1].set_xlabel('English Scores')
ax[0,1].legend()




ax[1,0].scatter(df1['Quant'],df1['Salary'],
                ec = 'k',
                color = 'skyblue',
                alpha = 0.5,
                label = f"Correlation : {round(df1[['Quant','Salary']].corr().
 ↪iloc[1,0],2)}"
                )
ax[1,0].set_ylabel('Salary')
ax[1,0].set_xlabel('Quant Scores')
ax[1,0].set_title('With Outliers')
ax[1,0].legend()


ax[1,1].scatter(df2['Quant'],df2['Salary'],
                ec = 'k',
                color = 'slateblue',
                alpha = 0.5,
                label = f"Correlation : {round(df2[['Quant','Salary']].corr().
 ↪iloc[1,0],2)}"
                )
ax[1,1].set_ylabel('Salary')
ax[1,1].set_title('Without Outliers')
ax[1,1].legend()



ax[2,0].scatter(df1['Logical'],df1['Salary'],
```

```
                ec = 'k',
                color = 'magenta',
                alpha = 0.5,
                label = f"Correlation : {round(df1[['Logical','Salary']].corr().
 ↪iloc[1,0],2)}"
                )
ax[2,0].set_ylabel('Salary')
ax[2,0].set_xlabel('Logical Scores')
ax[2,0].set_title('With Outliers')
ax[2,0].legend()


ax[2,1].scatter(df2['Logical'],df2['Salary'],
                ec = 'k',
                color = 'brown',
                alpha = 0.5,
                label = f"Correlation : {round(df2[['Logical','Salary']].corr().
 ↪iloc[1,0],2)}"
                )
ax[2,1].set_ylabel('Salary')
ax[2,1].set_title('Without Outliers')
ax[2,1].legend()


plt.tight_layout()
plt.show()
```



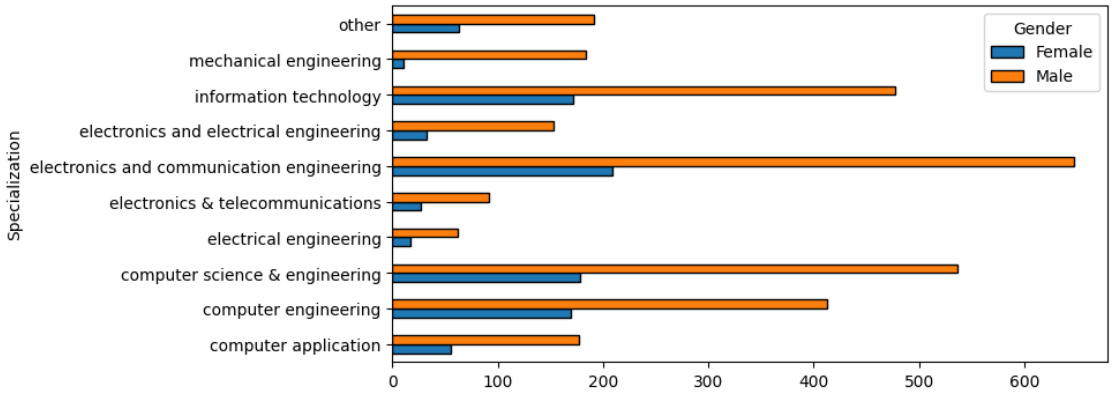
Observations The scatters plots above gives adequate evidence that salary is not effected by any of the above scores.

1.4.3 3. Crosstabs

3.1 Gender and Specialization

[ ]:
```
pd.crosstab(df1['Gender'],df1['Specialization']).T.plot(kind = 'barh',
                                                        ec = 'k',
                                                        figsize = (8,4))
```

[ ]: <Axes: ylabel='Specialization'>

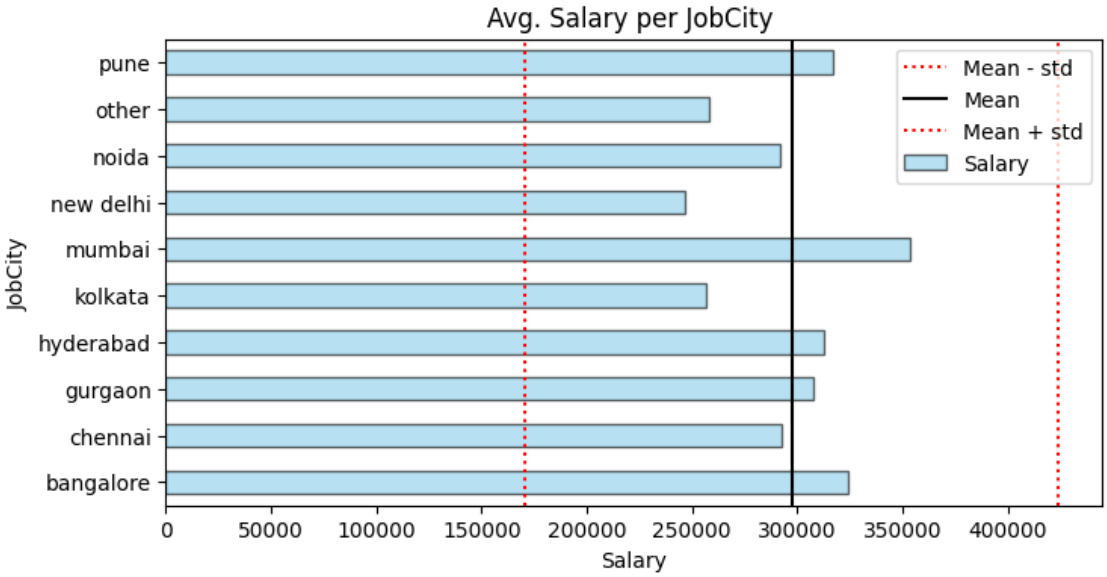


Observations There are almost males 2 times as of females in every specialization. Also, there are very less number of females who opted for mechanical and electronics.

1.4.4 4. Pivot Tables

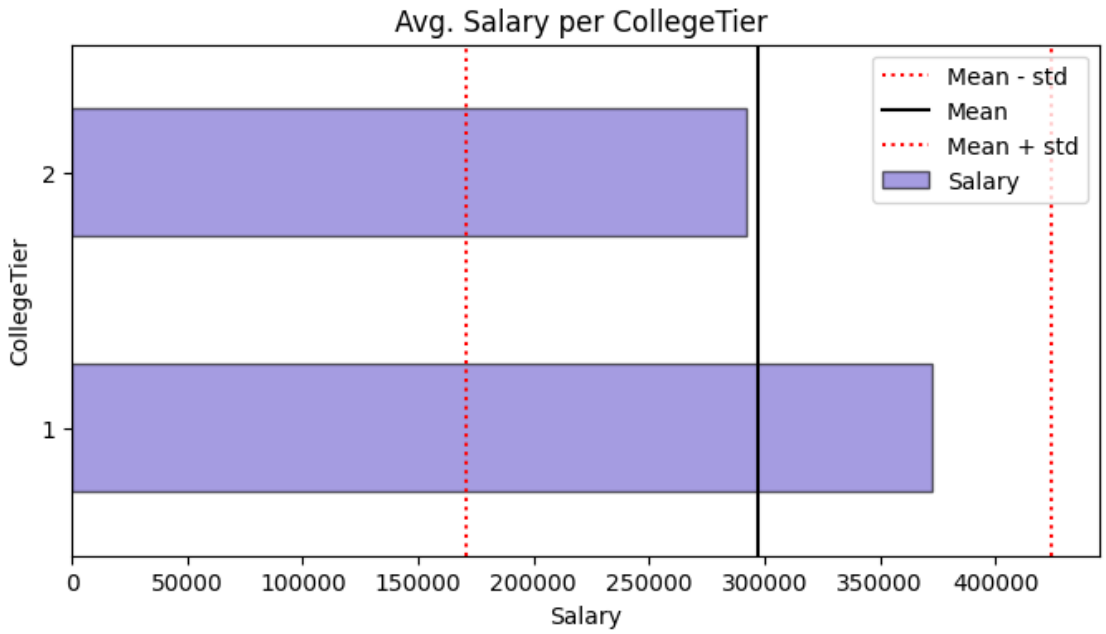
4.1 Average Salary per JobCity

[ ]:
```
pd.pivot_table(index = 'JobCity',
               values = 'Salary',
               data = df2).plot(kind = 'barh',
                                ec = 'k',
                               alpha = 0.6,
                                     color = 'skyblue',
                                     title = 'Avg. Salary per JobCity ',
                                     figsize = (8,4))
plt.xlabel('Salary')
plt.axvline(df2['Salary'].mean() - df2['Salary'].std(),
            color = 'red',
           linestyle = ':',
           label = 'Mean - std')
plt.axvline(df2['Salary'].mean(), color = 'k', label = 'Mean')
plt.axvline(df2['Salary'].mean() + df2['Salary'].std(), color = 'red',
           linestyle = ':',
           label = 'Mean + std')
plt.legend()
plt.show()
```



4.2 Average Salary per CollegeTier

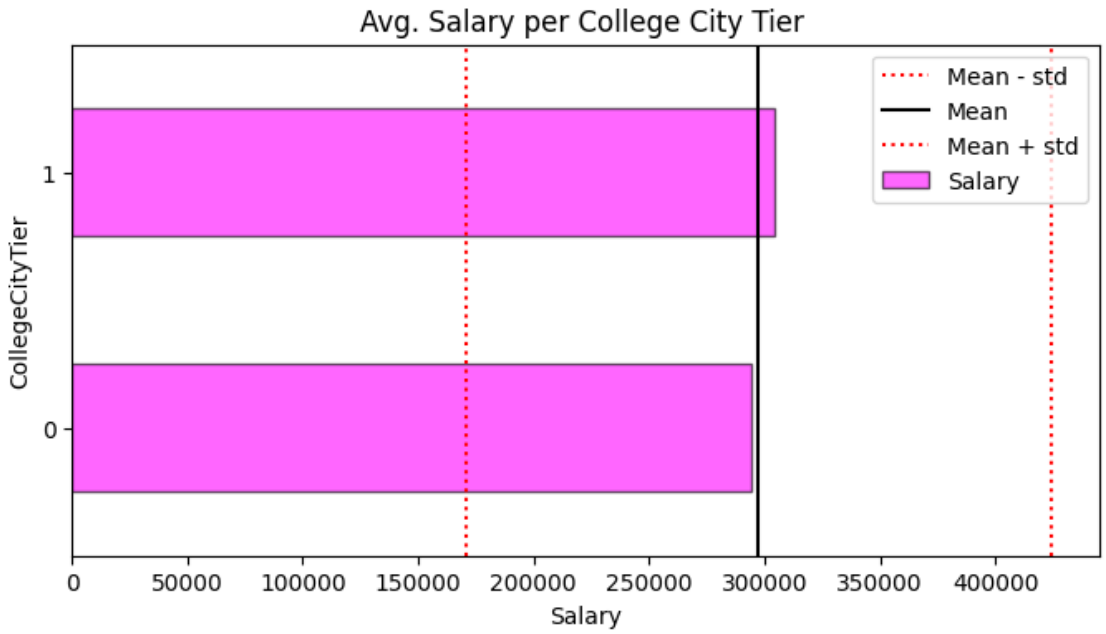
[ ]:
```
pd.pivot_table(index = 'CollegeTier',
               values = 'Salary',
               data = df2).plot(kind = 'barh',
                            alpha = 0.6,
                                  color = 'slateblue',
                                  title = 'Avg. Salary per CollegeTier ',
                                  figsize = (8,4),
                                  ec = 'k')
plt.xlabel('Salary')
plt.axvline(df2['Salary'].mean() - df2['Salary'].std(),
             color = 'red',
            linestyle = ':',
            label = 'Mean - std')
plt.axvline(df2['Salary'].mean(), color = 'k', label = 'Mean')
plt.axvline(df2['Salary'].mean() + df2['Salary'].std(), color = 'red',
            linestyle = ':',
            label = 'Mean + std')
plt.legend()
plt.show()
```



Observations College within Tier 1 offers high salary as compared to the colleges in TIer 2. Colleges in Tier 2 offers below overall average salary.

4.3 Average Salary per CollegeCityTier

[ ]:
```
pd.pivot_table(index = 'CollegeCityTier',
               values = 'Salary',
               data = df2).plot(kind = 'barh',
                            alpha = 0.6,
                                  color = 'magenta',
                                  title = 'Avg. Salary per College City Tier ',
                                  figsize = (8,4),
                                  ec = 'k')
plt.xlabel('Salary')
plt.axvline(df2['Salary'].mean() - df2['Salary'].std(),
             color = 'red',
            linestyle = ':',
            label = 'Mean - std')
plt.axvline(df2['Salary'].mean(), color = 'k', label = 'Mean')
plt.axvline(df2['Salary'].mean() + df2['Salary'].std(), color = 'red',
            linestyle = ':',
            label = 'Mean + std')
plt.legend()
plt.show()
```



Obervations Cities under Tier 1 and 2 offers almost same salaries to students.

1.5 Research Questions

1.5.1 1. Times of India article dated Jan 18, 2019 states that "After doing your Computer Science Engineering if you take up jobs as a Programming Analyst, Software Engineer, Hardware Engineer and Associate Engineer you can earn up to 2.5-3 lakhs as a fresh graduate."

Solution with Visualization

[ ]:
```
designations = ameo_data['Designation'].value_counts().sort_index()
pd.set_option('display.max_rows', None)

print(designations)
```

```
.net developer                          34
.net web developer                       4
account executive                        4
account manager                          1
admin assistant                          2
administrative coordinator               1
administrative support                   1
aircraft technician                      1
android developer                       46
application developer                   52
application engineer                    22
apprentice                               3
```

ase	3
asp.net developer	26
assistant administrator	1
assistant electrical engineer	2
assistant engineer	4
assistant manager	52
assistant professor	12
assistant programmer	3
assistant software engineer	3
assistant store manager	2
assistant system engineer	23
assistant system engineer - trainee	1
assistant system engineer trainee	1
assistant systems engineer	4
associate developer	1
associate engineer	6
associate manager	1
associate qa	1
associate software developer	4
associate software engg	1
associate software engineer	46
associate system engineer	11
associate technical operations	1
associate test engineer	1
asst. manager	2
automation engineer	15
branch manager	2
bss engineer	1
business analyst	49
business analyst consultant	9
business consultant	1
business development executive	5
business development manager	14
business development managerde	1
business intelligence analyst	3
business office manager	2
business process analyst	2
business system analyst	4
business systems analyst	1
business systems consultant	5
business technology analyst	3
c# developer	2
cad designer	3
cad drafter	1
catalog associate	2
civil engineer	3
clerical	2
clerical assistant	1

client services associate	13
cloud engineer	1
cnc programmer	1
co faculty	1
computer faculty	1
continuous improvement engineer	1
controls engineer	1
corporate recruiter	1
customer care executive	1
customer service	17
customer service manager	3
customer service representative	17
customer support engineer	2
data analyst	49
data entry operator	3
data scientist	3
database administrator	5
database developer	7
db2 dba	1
dba	1
dcs engineer	1
delivery software engineer	1
design engineer	28
designer	3
desktop support analyst	3
desktop support engineer	1
desktop support technician	8
developer	2
digital marketing specialist	1
documentation specialist	1
dotnet developer	1
editor	1
educator	2
electrical controls engineer	1
electrical design engineer	8
electrical designer	2
electrical engineer	23
electrical field engineer	5
electrical project engineer	9
electronic field service engineer	5
embedded engineer	1
embedded software engineer	16
engineer	47
engineer trainee	4
engineer- customer support	1
engineer-hws	1
engineering manager	3
engineering technician	2

enterprise solutions developer	1
entry level management trainee	13
entry level sales and marketing	2
environmental engineer	1
etl developer	5
executive administrative assistant	1
executive assistant	4
executive engg	1
executive engineer	2
executive hr	1
executive recruiter	2
faculty	2
field based employee relations manager	1
field business development associate	4
field engineer	3
field service engineer	7
financial analyst	2
financial service consultant	1
firmware engineer	2
front end developer	3
front end web developer	4
full stack developer	1
full-time loss prevention associate	1
game developer	3
general manager	2
get	14
gis/cad engineer	1
graduate apprentice trainee	3
graduate engineer	2
graduate engineer trainee	14
graduate trainee engineer	2
graphic designer	1
hardware engineer	8
help desk analyst	2
help desk technician	2
hr assistant	1
hr executive	1
hr generalist	2
hr manager	2
hr recruiter	5
html developer	2
human resource assistant	1
human resources analyst	1
human resources associate	1
human resources intern	1
implementation engineer	1
industrial engineer	2
information security analyst	6

information technology specialist	1
ios developer	13
it analyst	7
it assistant	1
it business analyst	4
it developer	1
it engineer	3
it executive	4
it operations associate	1
it recruiter	7
it support specialist	13
it technician	6
java developer	67
java software engineer	111
java trainee	1
javascript developer	2
jr. software developer	1
jr. software engineer	3
junior .net developer	1
junior engineer	11
junior engineer product support	3
junior manager	1
junior recruiter	1
junior research fellow	4
junior software developer	4
junior software engineer	9
junior system analyst	1
lead engineer	1
lecturer	20
lecturer & electrical maintenance	1
linux systems administrator	5
logistics executive	1
maintenance engineer	14
maintenance supervisor	1
management trainee	19
manager	6
manual tester	1
manufacturing engineer	3
marketing analyst	4
marketing assistant	2
marketing coordinator	3
marketing executive	1
marketing manager	1
mechanical design engineer	7
mechanical engineer	7
mis executive	1
mobile application developer	5
network administrator	4

network engineer	51
network security engineer	3
network support engineer	1
noc engineer	1
office coordinator	3
online marketing manager	3
operation engineer	1
operation executive	5
operational excellence manager	1
operational executive	1
operations	1
operations analyst	12
operations assistant	6
operations engineer	1
operations engineer and jetty handling	1
operations executive	2
operations manager	5
oracle dba	12
performance engineer	2
phone banking officer	1
php developer	33
planning engineer	2
portfolio analyst	2
principal software engineer	2
process advisor	2
process associate	8
process control engineer	1
process engineer	6
process executive	2
product design engineer	2
product developer	2
product development engineer	13
product engineer	10
product manager	5
production engineer	29
professor	1
program analyst trainee	1
program manager	1
programmer	36
programmer analyst	139
programmer analyst trainee	5
project administrator	1
project assistant	4
project coordinator	8
project engineer	77
project management officer	1
project manager	3
python developer	1

qa analyst	29
qa engineer	2
qa trainee	1
quality analyst	25
quality associate	10
quality assurance	3
quality assurance analyst	1
quality assurance auditor	1
quality assurance automation engineer	6
quality assurance engineer	14
quality assurance test engineer	15
quality assurance tester	3
quality consultant	1
quality control engineer	2
quality control inspection technician	1
quality control inspector	1
quality controller	1
quality engineer	17
r & d	1
r&d engineer	2
recruiter	1
recruitment associate	1
recruitment coordinator	1
research analyst	15
research associate	11
research engineer	7
research scientist	1
research staff member	2
rf engineer	13
rf/dt engineer	1
risk consultant	2
risk investigator	1
ruby on rails developer	1
sales & service engineer	1
sales account manager	2
sales and service engineer	1
sales associate	4
sales coordinator	2
sales development manager	1
sales engineer	13
sales executive	17
sales management trainee	3
sales manager	1
sales support	1
sales trainer	1
salesforce developer	7
sap abap associate consultant	1
sap abap consultant	2

sap analyst	1
sap consultant	3
sap functional consultant	1
sap mm consultant	1
secretary	1
senior .net developer	4
senior business analyst	3
senior design engineer	1
senior developer	1
senior engineer	18
senior java developer	4
senior mechanical engineer	2
senior network engineer	3
senior php developer	4
senior programmer	1
senior project engineer	1
senior quality assurance engineer	1
senior quality engineer	5
senior research fellow	1
senior risk consultant	1
senior sales executive	1
senior software developer	20
senior software engineer	72
senior systems engineer	35
senior test engineer	2
senior web developer	3
seo	3
seo analyst	1
seo engineer	2
seo executive	4
seo trainee	1
service and sales engineer	1
service coordinator	5
service engineer	4
service manager	3
sharepoint developer	1
shift engineer	2
site engineer	2
site manager	5
software analyst	1
software architect	1
software designer	1
software developer	265
software development engineer	17
software devloper	1
software eng	1
software engg	1
software engineer	539

software engineer analyst	1
software engineer associate	2
software engineer trainee	3
software engineere	1
software engineering associate	1
software enginner	2
software executive	1
software programmer	2
software quality assurance analyst	5
software quality assurance tester	17
software test engineer	100
software test engineer (etl)	1
software test engineerte	1
software tester	2
software trainee	7
software trainee engineer	1
sql dba	5
sql developer	4
sr. database engineer	1
sr. engineer	2
staffing recruiter	1
supply chain analyst	1
support engineer	1
system administrator	20
system analyst	1
system engineer	205
system engineer trainee	1
systems administrator	3
systems analyst	11
systems engineer	118
talent acquisition specialist	2
team lead	2
team leader	2
technical analyst	1
technical assistant	2
technical consultant	2
technical engineer	13
technical lead	5
technical operations analyst	1
technical recruiter	8
technical support engineer	76
technical support executive	3
technical support specialist	8
technical writer	3
technology analyst	6
technology lead	1
telecom engineer	9
telecom support engineer	1

```
telecommunication engineer                  1
teradata dba                                1
teradata developer                          1
territory sales manager                     1
test engineer                              57
test technician                             1
testing engineer                            2
trainee decision scientist                  2
trainee engineer                            7
trainee software developer                  1
trainee software engineer                   6
training specialist                         3
ui developer                                5
ux designer                                 2
visiting faculty                            1
web application developer                   6
web designer                                9
web designer and joomla administrator       1
web designer and seo                        1
web developer                              54
web intern                                  1
website developer/tester                    1
windows systems administrator               1
Name: Designation, dtype: int64
```

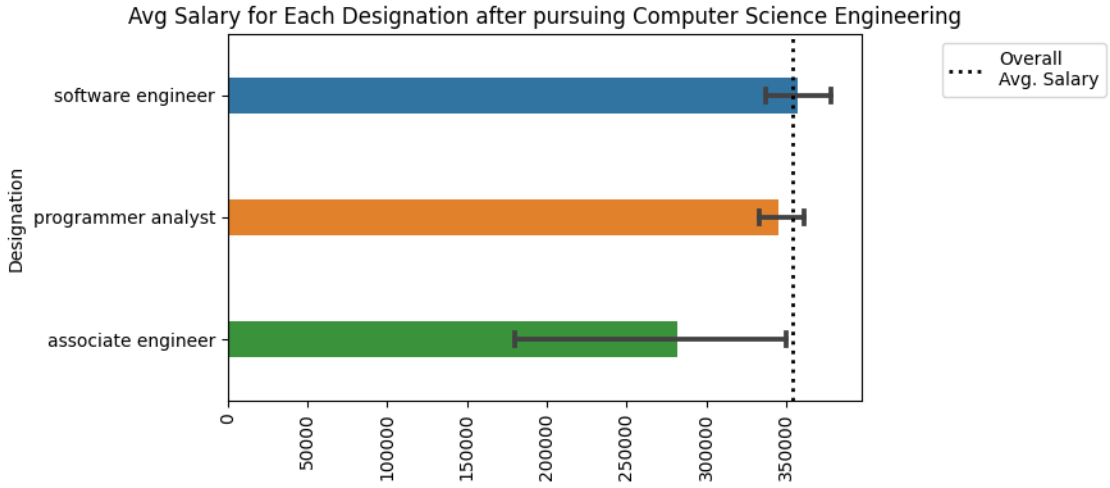
[ ]:
```
ameo_data['Designation'] = ameo_data['Designation'].replace([
    'programmer analyst trainee', 'programmer analyst'
], 'programmer analyst'
)

ameo_data['Designation'] = ameo_data['Designation'].replace([
    'software eng', 'software engg', 'software engineer', 'software engineere',␣
 ↪'software enginner'
], 'software engineer'
)
```

[ ]:
```
df3 = ameo_data[(ameo_data["Designation"].isin(["programmer analyst", "software␣
 ↪engineer", "hardware engineer", "associate engineer"])) &
                (ameo_data["Specialization"].isin(["computer science &␣
 ↪engineering", "computer engineering"]))]
```

[ ]:
```
fig, ax = plt.subplots(figsize=(10, 4))
sns.barplot(x='Salary', y='Designation',
            data=df3,
            capsize=0.1,
            width=0.3,
            ax=ax)
```

```
ax.axvline(df3['Salary'].mean(), color='k',
           linestyle=':',
           linewidth=2, label='Overall\nAvg. Salary')
ax.set_title('Avg Salary for Each Designation after pursuing Computer Science␣
 ↪Engineering')
ax.legend(loc='upper right', bbox_to_anchor=(1.4, 1))
ax.set_xlabel('')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)

plt.tight_layout()
plt.show()
```

```
C:\Users\himan\AppData\Local\Temp\ipykernel_15328\3672005199.py:13: UserWarning:
FixedFormatter should only be used together with FixedLocator
  ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
```



Solution considering all designations at once in Hypothesis

[ ]:
```
import random
n = 40
salary_random = random.sample(df3['Salary'].tolist(),n)
print(salary_random)
```

```
[170000, 400000, 515000, 350000, 360000, 110000, 265000, 420000, 120000, 120000,
315000, 200000, 335000, 305000, 280000, 435000, 300000, 180000, 500000, 200000,
490000, 300000, 930000, 325000, 275000, 450000, 350000, 390000, 345000, 180000,
590000, 560000, 240000, 300000, 800000, 420000, 350000, 310000, 450000, 360000]
```

Function for T-Score

[ ]:
```
def t_score(sample_size, sample_mean, pop_mean, sample_std):
    numerator = sample_mean - pop_mean
    denomenator = sample_std / sample_size**0.5
    return numerator / denomenator
```

Calculating sample values

[ ]:
```
from scipy.stats import t,norm
import statistics

print('Sample Mean: ', statistics.mean(salary_random))
print('Sample Standard Deviation: ', statistics.stdev(salary_random))
```

```
Sample Mean:  357375
Sample Standard Deviation:  167236.1584475908
```

[ ]:
```
sample_size = 40
sample_mean = statistics.mean(salary_random)
pop_mean = 275000
sample_std = statistics.stdev(salary_random)
```

Calculating t_value

[ ]:
```
t_value = t_score(sample_size, sample_mean, pop_mean, sample_std)
print(t_value)
```

```
3.1152667542050074
```

Calculating t_critical

[ ]:
```
confidence_level = 0.95

alpha = 1 - confidence_level

t_critical = t.ppf(1 - alpha/2, df = 99)

print(t_critical)
```

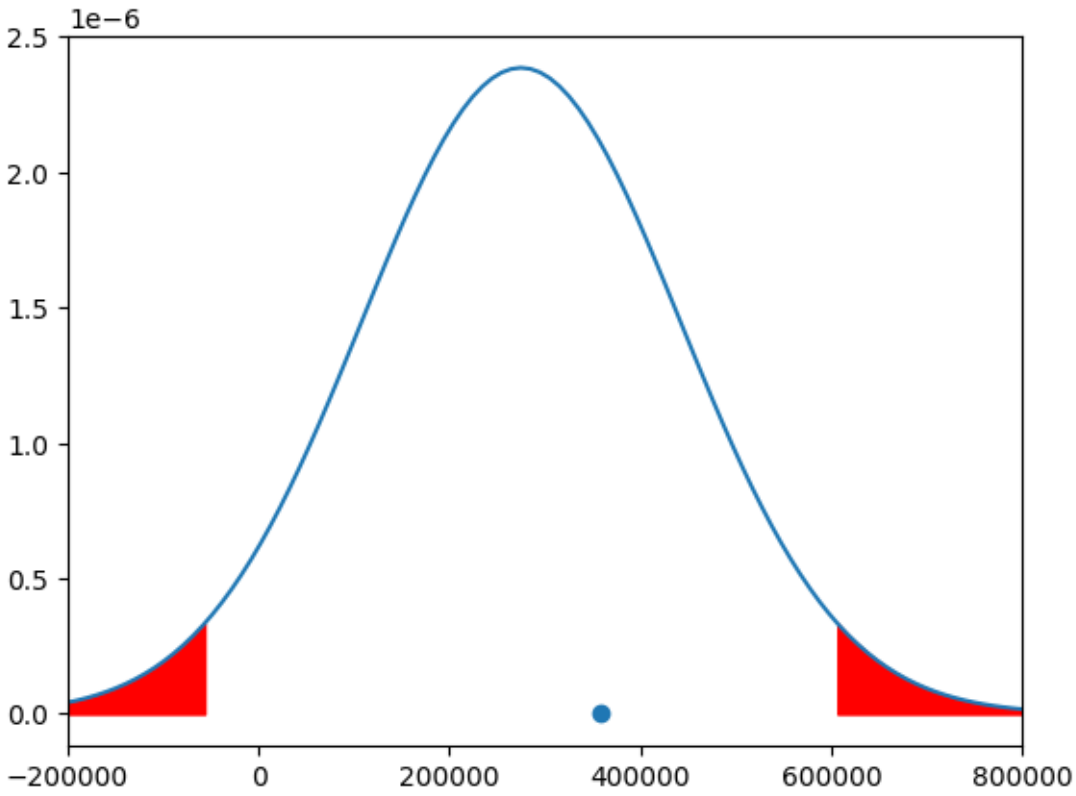
```
1.9842169515086827
```

One Sample t-test Visualization

[ ]:
```
x_min = -200000
x_max = 800000

mean = pop_mean
std = sample_std

x = np.linspace(x_min, x_max, 100)
```

```
y = norm.pdf(x, mean, std)
plt.xlim(x_min, x_max)
plt.plot(x, y)

t_critical_left = pop_mean + (-t_critical * std)
t_critical_right = pop_mean + (t_critical * std)

x1 = np.linspace(x_min, t_critical_left, 100)
y1 = norm.pdf(x1, mean, std)
plt.fill_between(x1, y1, color='red')

x2 = np.linspace(t_critical_right, x_max, 100)
y2 = norm.pdf(x2, mean, std)
plt.fill_between(x2, y2, color='red')

plt.scatter(sample_mean, 0)
plt.annotate("x_bar", (sample_mean, 0.7))
```

[ ]: Text(357375, 0.7, 'x_bar')



[ ]:
```
if(t_value < t_critical):
    print("There is not enough evidence to reject the Null Hypothesis")
else:
    print("There is sufficent evidence to reject the Null Hypothesis")
```

```
There is sufficent evidence to reject the Null Hypothesis
```

[ ]:
```
p_value = 2 * (1.0 - norm.cdf(np.abs(t_value)))

print("p_value = ", p_value)

if(p_value > alpha):
    print("There is not enough evidence to reject the Null Hypothesis")
else:
    print("There is sufficent evidence to reject the Null Hypothesis")
```

```
p_value =  0.0018377862896536978
There is sufficent evidence to reject the Null Hypothesis
```

Test	Value
t_value	4.160393243881741
t_critical	1.9842169515086827
p_value	3.177000951049003e-05

Observations

- As the result of the hypothesis testing we see that the claim is false.
- For this claim Null Hupothesis fails.
- The **t_critical** and probability value i.e. **p_value** claiming it as wrong.

Solution considering Individual Designations Hypothesis

[ ]:
```
job_group = df3.groupby('Designation')
job_salary_mean = job_group['Salary'].mean()
job_salary_std = job_group['Salary'].std()
```

[ ]:
```
print("Mean salaries for different job roles:")
print(job_salary_mean)

print("\nStandard deviation of salaries for different job roles:")
print(job_salary_std)
```

```
Mean salaries for different job roles:
Designation
associate engineer    281666.666667
programmer analyst    345267.857143
software engineer     356820.000000
```

```
Name: Salary, dtype: float64

Standard deviation of salaries for different job roles:
Designation
associate engineer     89768.220063
programmer analyst     55844.098271
software engineer     165473.604102
Name: Salary, dtype: float64
```

[ ]:
```
alpha = 0.05
```

[ ]:
```
from scipy.stats import ttest_1samp

prog_analyst_salaries = df3.loc[df3['Designation'] == 'programmer analyst',␣
 ↪'Salary'].values
software_eng_salaries = df3.loc[df3['Designation'] == 'software engineer',␣
 ↪'Salary'].values
hardware_eng_salaries = df3.loc[df3['Designation'] == 'hardware engineer',␣
 ↪'Salary'].values
assoc_eng_salaries = df3.loc[df3['Designation'] == 'associate engineer',␣
 ↪'Salary'].values

expected_range = (250000, 300000)

for job, salaries in [("programmer analyst", prog_analyst_salaries),
                      ("software engineer", software_eng_salaries),
                      ("hardware engineer", hardware_eng_salaries),
                      ("associate engineer", assoc_eng_salaries)]:

    t_stat, p_val = ttest_1samp(salaries, expected_range[0],␣
 ↪alternative='greater')

    print(f"One-sample t-test for {job}:")
    print(f"  t_critical: {t_stat:.2f}")
    print(f"  p_value: {p_val:.5e}")

    if p_val < 0.05:
        print("  Result: There is sufficent evidence to reject the Null␣
 ↪Hypothesis\n")
    else:
        print("  Result: There is not enough evidence to reject the Null␣
 ↪Hypothesis\n")
```

```
One-sample t-test for programmer analyst:
  t_critical: 12.77
  p_value: 2.20314e-18
  Result: There is sufficent evidence to reject the Null Hypothesis
```

```
One-sample t-test for software engineer:
  t_critical: 10.21
  p_value: 5.81591e-21
  Result: There is sufficent evidence to reject the Null Hypothesis

One-sample t-test for hardware engineer:
  t_critical: nan
  p_value: nan
  Result: There is not enough evidence to reject the Null Hypothesis

One-sample t-test for associate engineer:
  t_critical: 0.61
  p_value: 3.01696e-01
  Result: There is not enough evidence to reject the Null Hypothesis


c:\Users\himan\AppData\Local\Programs\Python\Python311\Lib\site-
packages\numpy\core\fromnumeric.py:3432: RuntimeWarning: Mean of empty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
c:\Users\himan\AppData\Local\Programs\Python\Python311\Lib\site-
packages\numpy\core\_methods.py:190: RuntimeWarning: invalid value encountered
in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

Designation	t_critical	p_value	Result
Programmer Analyst	12.77	2.20314e-18	There is sufficent evidence to reject the Null Hypothesis
Software Engineer	10.21	5.81591e-21	There is sufficent evidence to reject the Null Hypothesis
Hardware Engineer	NaN	NaN	There is not enough evidence to reject the Null Hypothesis
Associate Engineer	0.61	3.01696e-01	There is not enough evidence to reject the Null Hypothesis

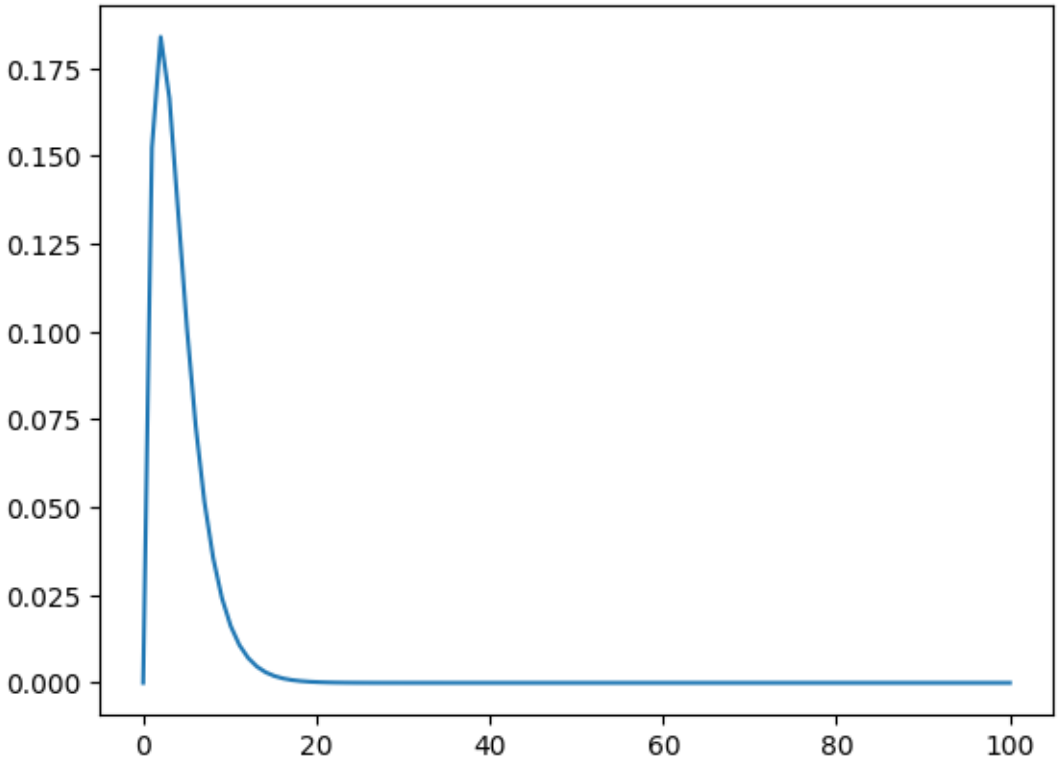
Observations

1.5.2 2. Is there a relationship between gender and specialization? (i.e. Does the preference of Specialisation depend on the Gender?)

[ ]:
```
from scipy.stats import chi2
from scipy.stats import chi2_contingency
```

[ ]:
```
x = np.linspace(0, 100, 100)
y = chi2.pdf(x, df = 4)
plt.plot(x, y)
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x1b45fed5ed0>]
```



[ ]:
```
obsr = pd.crosstab(df2.Specialization,df2.Gender)
obsr
```

[ ]:

Gender	Female	Male
Specialization		
computer application	36	93
computer engineering	142	293
computer science & engineering	57	162
electrical engineering	13	46
electronics & telecommunications	25	80

```
electronics and communication engineering     169   510
electronics and electrical engineering         28   115
information technology                        115   275
mechanical engineering                         10   147
other                                          46   128
```

Computing Chi2 statistics, p_value and dof

[ ]:
```
chi2_statistic, chi2_p_value, chi2_dof, chi2_expected = chi2_contingency(obsr)

print("Statistic            :", chi2_statistic)
print('')
print("p value              :", chi2_p_value)
print('')
print("Degrees of freedom   :", chi2_dof)
print('')
print("Expected frequencies array:\n", chi2_expected)
```

```
Statistic            : 48.62141720904882

p value              : 1.9542895953348004e-07

Degrees of freedom   : 9

Expected frequencies array:
 [[ 33.20843373  95.79156627]
 [111.98192771 323.01807229]
 [ 56.37710843 162.62289157]
 [ 15.18835341  43.81164659]
 [ 27.03012048  77.96987952]
 [174.79477912 504.20522088]
 [ 36.8124498  106.1875502 ]
 [100.39759036 289.60240964]
 [ 40.41646586 116.58353414]
 [ 44.79277108 129.20722892]]
```

Calculating chi2_critical

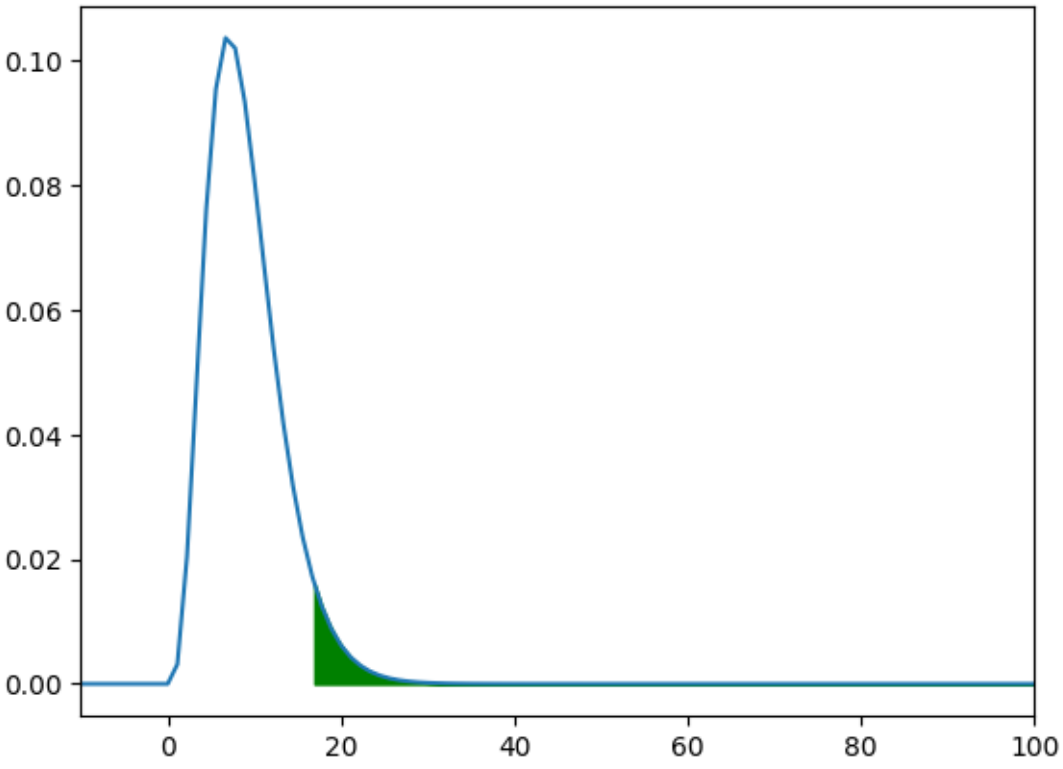
[ ]:
```
confidence_level = 0.95
alpha = 1 - confidence_level

chi2_critical = chi2.ppf(1 - alpha, chi2_dof)

chi2_critical
```

[ ]:
```
16.918977604620448
```

Chi-squared Test Visualization

[ ]:
```
x_min = -10
x_max = 100

x = np.linspace(x_min, x_max, 100)
y = chi2.pdf(x, chi2_dof)
plt.xlim(x_min, x_max)
plt.plot(x, y)

chi2_critical_right = chi2_critical

x1 = np.linspace(chi2_critical_right, x_max, 100)
y1 = chi2.pdf(x1, chi2_dof)
plt.fill_between(x1, y1, color='green')
```

[ ]:
```
<matplotlib.collections.PolyCollection at 0x1b4601b9290>
```



[ ]:
```
if(chi2_statistic > chi2_critical):
   print("There is not enough evidence to reject the Null Hypothesis")
else:
    print("There is sufficent evidence to reject the Null Hypothesis")
```

```
There is not enough evidence to reject the Null Hypothesis
```

[ ]:
```
if(chi2_p_value < alpha):
    print("There is not enough evidence to reject the Null Hypothesis")
else:
    print("There is sufficent evidence to reject the Null Hypothesis")
```

```
There is not enough evidence to reject the Null Hypothesis
```

Test	Value
chi2_critical	16.918977604620448
chi2_statistic	48.62141720904882
chi2_p_value	1.9542895953348e-07

Observations

- As the result of the second research question we see that there is a relationship between Gender and specialization.
- We test this claim through Chi-Square test and find the result that both the categorical variables are dependent on each other.
- Some specialization or working field does not allow some candidates to work in that field due to some risks.

1.6 Conclusion

1. **Data Understanding:**
   - The dataset encompasses the employment outcomes of engineering graduates, focusing on target variable *Salary*.
   - Additionally, it includes standardized scores in three distinct areas: *cognitive skills, technical skills, and personality skills.*
2. **Data Manipulation:**
   - Upon initial observation, the dataset consists of 4000 rows and 40 columns.
   - The dataset exhibits numerous duplicate values, necessitating data manipulation.
   - Initially, we remove redundant rows and columns.
   - Subsequently, we assess for the presence of any missing values (NaN).
   - Following data cleaning, we proceed with visualization.
3. **Data Visualization:**
   - **Univariate Analysis:**
     - Univariate analysis encompasses various plots, including Cumulativee Distribution Functions (CDF), Histograms, Box Plots, and Summary Plots.
     - These visualizations illustrate probability and frequency distributions.
   - **Bivariate Analysis:**
     - Bivariate analysis comprises Scatterplots, Barplots, Crosstabs, Pivot tables, pie charts.

- This analysis helps in comparing percentages across different variables.
- Additionally, it aids in identifying outliers, as observed through Boxplots.
- For instance, Countplots assist in identifying outliers within categorical variables, such as Job City, by highlighting the cities with higher employee counts.

1.7 Making a Research Question

1.7.1 In a comparative study of recruitment practices among leading companies, does AMEO's hiring policy of recruiting candidates with a minimum percentage of 70% and maintaining an average percentage of 80% hold true?

[ ]:
```
average_percentage = df2['12percentage'].mean()
average_percentage
```

[ ]:
```
75.13589558232933
```

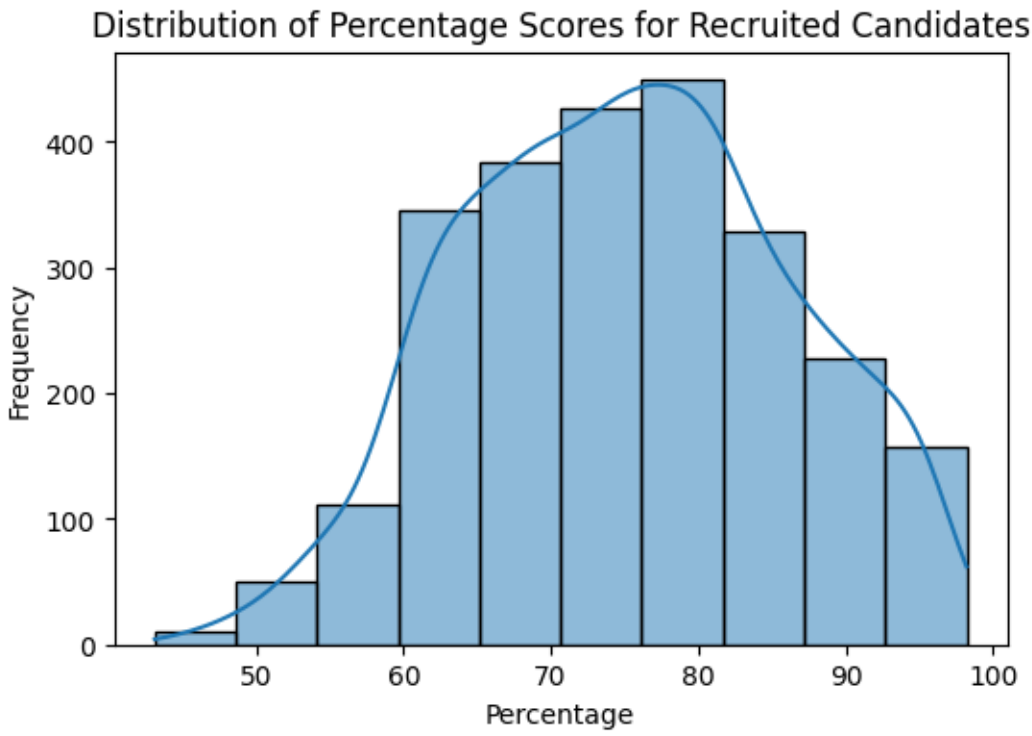
1.7.2 Interpreting the Difference

[ ]:
```
Difference = 80 - 74.51477
Difference
```

[ ]:
```
5.485230000000001
```

Since the difference is positive (5.5), it indicates that the average percentage obtained (74.5%) is below the stated goal of 80%.

[ ]:
```
plt.figure(figsize=(6, 4))
sns.histplot(df2['12percentage'], bins=10, kde=True)
plt.title('Distribution of Percentage Scores for Recruited Candidates')
plt.xlabel('Percentage')
plt.ylabel('Frequency')
plt.show()
```



[ ]:
```
if average_percentage < 80:
    print("Recommendation: AMCAR should consider revising its minimum␣
 ↪percentage requirement or implementing additional screening processes to␣
 ↪improve the quality of recruited candidates.")
else:
    print("No specific recommendation.")
```

```
Recommendation: AMEO should consider revising its minimum percentage requirement
or implementing additional screening processes to improve the quality of
recruited candidates.
```

1.7.3 Observations

AMCAT should consider revising its minimum percentage requirement or implementing additional screening processes to improve the quality of recruited candidates.