FE handling missing data

March 30, 2023

1 Handling missing data

1.1 Why are there missing values in the datasets?

1.1.1 1) Data entry errors:- Sometimes, data may not be entered correctly, leading to missing values.

1.1.2 2) Non-response or non-participation:- Participants may not respond to certain questions, resulting in missing data.

1.1.3 3) Data not collected:- In some cases, certain data points may not have been collected due to technical or logistical reasons.

1.1.4 4) Data cleaning or preprocessing:- Missing data may be intentionally introduced during data cleaning or preprocessing, such as when outliers or invalid values are removed.

1.1.5 5) Confidentiality or privacy concerns: - In some cases, certain data points may be withheld to protect the confidentiality or privacy of individuals or organizations.

1.2 What are the different types of Missing Data?

1.2.1 1) Missing Completely at Random (MCAR):- MCAR occurs when the missingness of data is completely unrelated to any other variable in the dataset, including the variable being measured. In other words, the missingness occurs randomly and there is no systematic reason why certain values are missing. This type of missing data is considered to be the least problematic because it does not introduce bias into the analysis.

1.2.2 2) Missing at Random (MAR):- MAR occurs when the missingness of data is related to other variables in the dataset, but not to the variable being measured. In other words, the missingness is related to other measured variables in the dataset, but not to the variable that is missing. This type of missing data can introduce bias into the analysis if the relationship between the missing data and the other variables in the dataset is not properly accounted for.

1.2.3 3) Missing Not at Random (MNAR):- MNAR occurs when the missingness of data is related to the value of the variable that is missing. In other words, the missingness is related to the unmeasured value of the variable that is missing. This type of missing data can introduce significant bias into the analysis because the missing values may be systematically different from the non-missing values, and simply ignoring the missing data can lead to incorrect conclusions.

[1]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
df=sns.load_dataset("titanic")
```

[3]:
```
df.head()
```

[3]:
```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

[4]:
```python
df.isnull().mean()
```

[4]:
```
survived       0.000000
pclass         0.000000
sex            0.000000
age            0.198653
sibsp          0.000000
parch          0.000000
fare           0.000000
embarked       0.002245
class          0.000000
who            0.000000
adult_male     0.000000
deck           0.772166
embark_town    0.002245
alive          0.000000
alone          0.000000
dtype: float64
```

2 mean / median / mode imputation

2.0.1 When should we apply? Mean/median imputation has the assumption that the data are missing completely at random(MCAR). We solve this by replacing the NAN with the most frequent occurance of the variables.

[5]:
```python
#To reduce impact of outliers, we use median.
def impute_median(df,variable,median):
    df[variable+"_median"]=df[variable].fillna(median)
```

[6]:
```python
median=df["age"].median()
impute_median(df,"age",median)
df.head()
```

[6]:
```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone  age_median
0    man        True  NaN  Southampton    no  False        22.0
1  woman       False    C    Cherbourg   yes  False        38.0
2  woman       False  NaN  Southampton   yes   True        26.0
3  woman       False    C  Southampton   yes  False        35.0
4    man        True  NaN  Southampton    no   True        35.0
```

[7]:
```
df["age"].std()
```

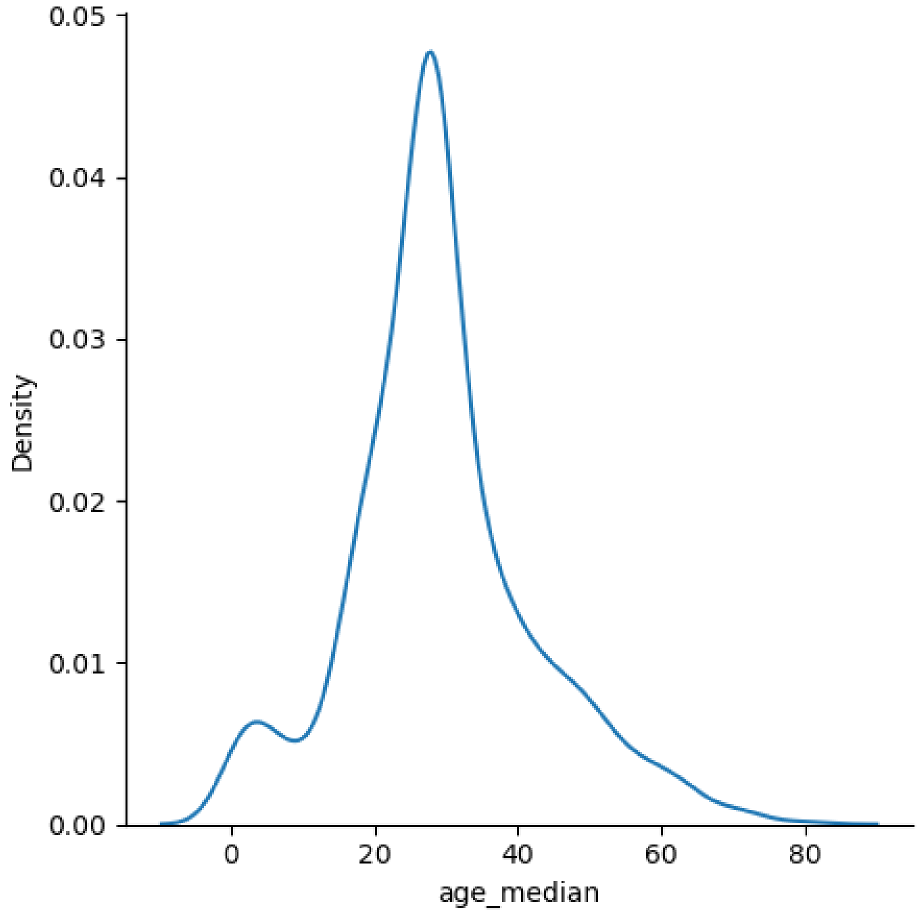
[7]: 14.526497332334044

[8]:
```
df["age_median"].std()
```

[8]: 13.019696550973194

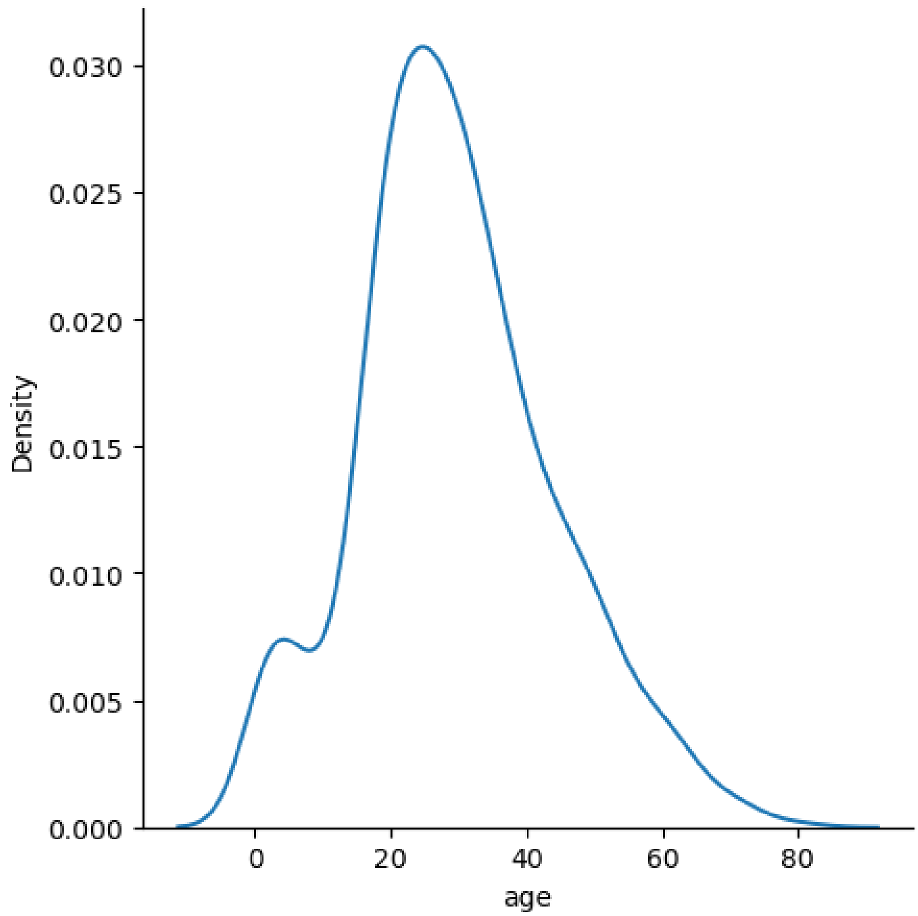
[9]:
```
sns.displot(df["age_median"],kind="kde")
```

[9]: <seaborn.axisgrid.FacetGrid at 0x7efd63876950>



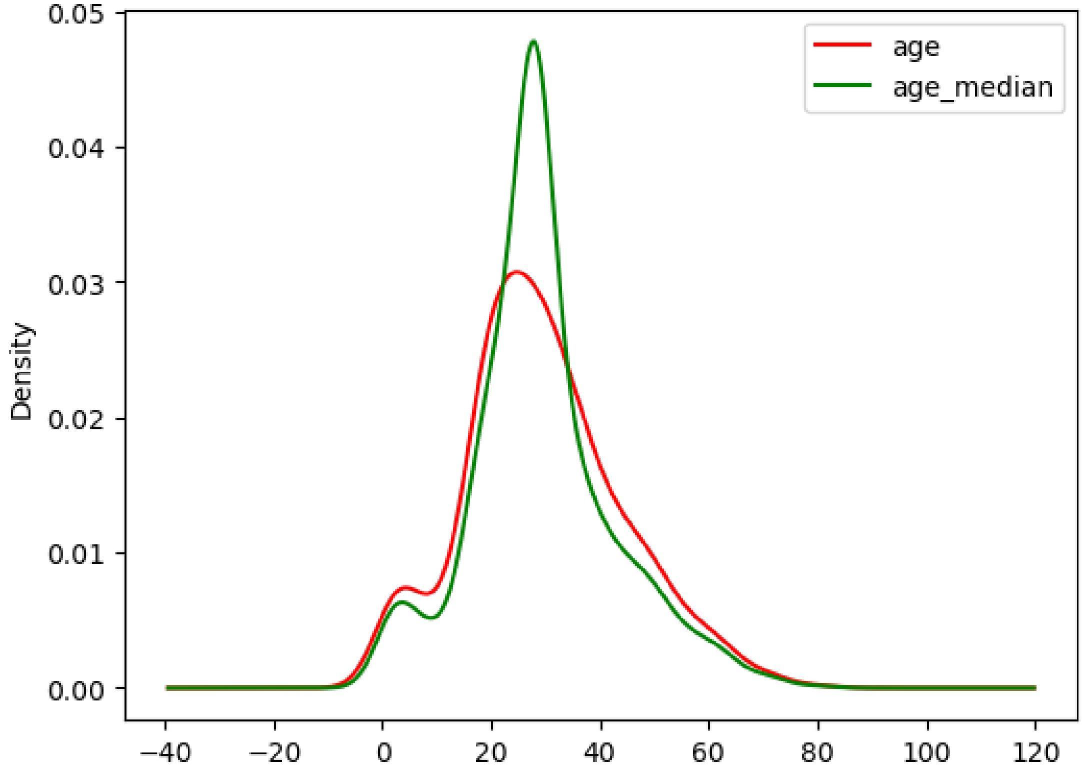
[10]:
```
sns.displot(df["age"],kind="kde")
```

[10]: `<seaborn.axisgrid.FacetGrid at 0x7efd5b5ad7b0>`



[11]:

```
fig=plt.figure()
ax=fig.add_subplot(111)

df["age"].plot(kind="kde",ax=ax,color="red")
df["age_median"].plot(kind="kde",ax=ax,color="green")

lines, lables=ax.get_legend_handles_labels()
ax.legend(lines, lables,loc="best")
plt.show()                              # kde = kernel density estimation
```



2.1 Advantages

2.1.1 Easy to implement

2.1.2 Preserves sample size

2.1.3 Works well for variables with a normal distribution

2.2 Disadvantages

2.2.1 Change or Distortion in the original variance

2.2.2 Impacts Correlation

[ ]:
```

```

3 random sample imputation

3.0.1 Aim: Random sample imputation consists of taking random observation from the dataset and we use this observation to replace the nan values

3.0.2 When should it be used? It assumes that the data are missing completely at random(MCAR)

[12]:
```
df["age"].dropna().sample(df["age"].isnull().sum(),random_state=29)
```

[12]:
```
771    48.0
402    21.0
112    22.0
682    20.0
315    26.0
       …
670    40.0
120    21.0
275    63.0
786    18.0
708    22.0
Name: age, Length: 177, dtype: float64
```

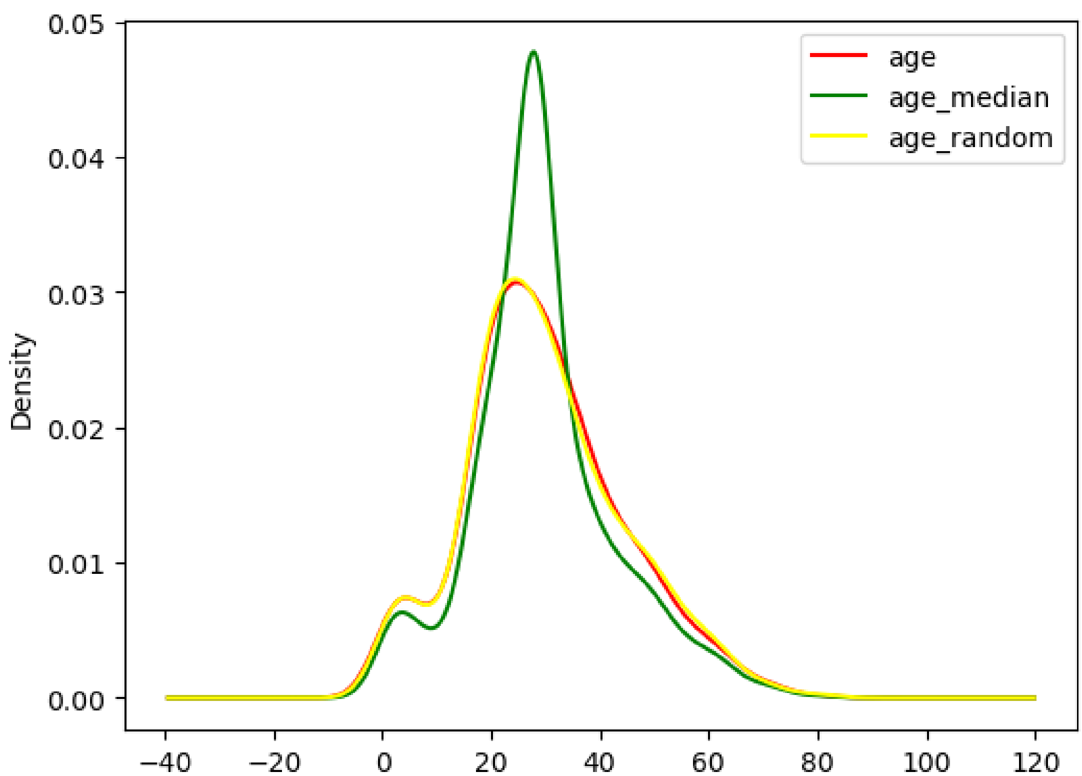
[13]:
```
def impute_random(df,variable):
    df[variable+"_random"]=df[variable]

    # it will have random sample to fill the na
    random_sample=df[variable].dropna().sample(df[variable].isnull().
 ↪sum(),random_state=29)

    #it needs same index in oreder to mearge the dataset
    random_sample.index=df[df[variable].isnull()].index

    df.loc[df[variable].isnull(),variable+"_random"]=random_sample
```

[14]:
```
impute_random(df,"age")
df.head()
```

[14]:
```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone  age_median  age_random
0    man        True  NaN  Southampton    no  False        22.0        22.0
1  woman       False    C    Cherbourg   yes  False        38.0        38.0
```

```
2  woman        False  NaN  Southampton   yes   True         26.0         26.0
3  woman        False    C  Southampton   yes  False         35.0         35.0
4    man         True  NaN  Southampton    no   True         35.0         35.0
```

[15]:
```
fig=plt.figure()
ax=fig.add_subplot(111)

df["age"].plot(kind="kde",ax=ax,color="red")
df["age_median"].plot(kind="kde",ax=ax,color="green")
df["age_random"].plot(kind="kde",ax=ax,color="yellow")

lines, lables=ax.get_legend_handles_labels()
ax.legend(lines, lables,loc="best")
plt.show()
```



3.0.3 Advantages

3.0.4 Easy To implement

3.0.5 There is less distortion in variance

3.0.6 Disadvantages

3.0.7 Every situation randomness wont work

[ ]:

4 capturing NAN value with new feature

4.0.1 It works well if the data are not missing completely at random

[16]:
```
import numpy as np
```

[17]:
```
# if there is missing value, we put 1 otherwise 0.
df["age_NAN"]=np.where(df["age"].isnull(),1,0)
df.head(10)
```

[17]:
```
   survived  pclass     sex   age  sibsp  parch     fare embarked   class  \
0         0       3    male  22.0      1      0   7.2500        S   Third
1         1       1  female  38.0      1      0  71.2833        C   First
2         1       3  female  26.0      0      0   7.9250        S   Third
3         1       1  female  35.0      1      0  53.1000        S   First
4         0       3    male  35.0      0      0   8.0500        S   Third
5         0       3    male   NaN      0      0   8.4583        Q   Third
6         0       1    male  54.0      0      0  51.8625        S   First
7         0       3    male   2.0      3      1  21.0750        S   Third
8         1       3  female  27.0      0      2  11.1333        S   Third
9         1       2  female  14.0      1      0  30.0708        C  Second

     who  adult_male deck  embark_town alive  alone  age_median  age_random  \
0    man        True  NaN  Southampton    no  False        22.0        22.0
1  woman       False    C    Cherbourg   yes  False        38.0        38.0
2  woman       False  NaN  Southampton   yes   True        26.0        26.0
3  woman       False    C  Southampton   yes  False        35.0        35.0
4    man        True  NaN  Southampton    no   True        35.0        35.0
5    man        True  NaN   Queenstown    no   True        28.0        48.0
6    man        True    E  Southampton    no   True        54.0        54.0
7  child       False  NaN  Southampton    no  False         2.0         2.0
8  woman       False  NaN  Southampton   yes  False        27.0        27.0
9  child       False  NaN    Cherbourg   yes  False        14.0        14.0

   age_NAN
0        0
1        0
```

```
2          0
3          0
4          0
5          1
6          0
7          0
8          0
9          0
```

4.1 Advantages

4.1.1 Easy to implement

4.1.2 Captures the importance of missing values

4.2 Disadvantages

4.2.1 Creating Additional Features(Curse of Dimensionality)

```
[ ]:
```

5 End of Distribution imputation

5.0.1 missing values are replaced with values that are at the extreme end of the distribution

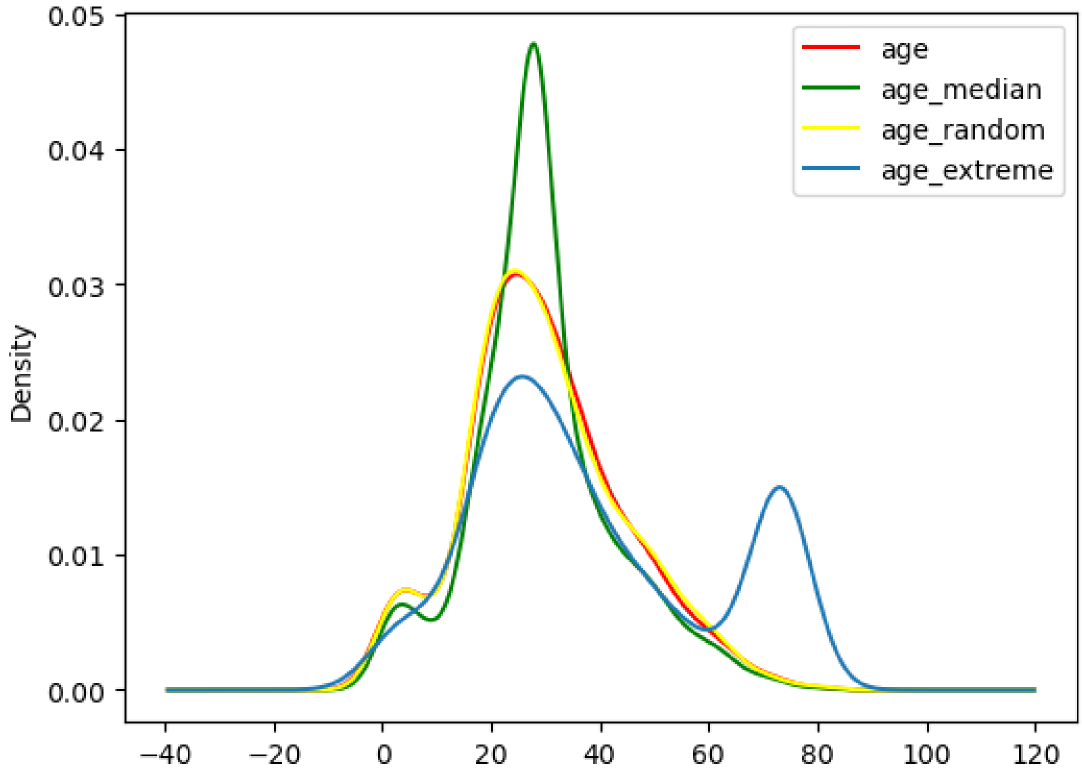
```
[18]: def impute_NAN(df,variable,median,extreme_value):
          df[variable+"_extreme"]=df["age"].fillna(extreme_value)
```

```
[19]: extreme=df["age"].mean() + (3*(df["age"].std()))    # extreme is generally␣
       ↪value lies above 3 standard deviation
      impute_NAN(df,"age",df["age"].median(),extreme)

      df.head(10)
```

```
[19]:   survived  pclass     sex   age  sibsp  parch     fare embarked   class  \
      0         0       3    male  22.0      1      0   7.2500        S   Third
      1         1       1  female  38.0      1      0  71.2833        C   First
      2         1       3  female  26.0      0      0   7.9250        S   Third
      3         1       1  female  35.0      1      0  53.1000        S   First
      4         0       3    male  35.0      0      0   8.0500        S   Third
      5         0       3    male   NaN      0      0   8.4583        Q   Third
      6         0       1    male  54.0      0      0  51.8625        S   First
      7         0       3    male   2.0      3      1  21.0750        S   Third
      8         1       3  female  27.0      0      2  11.1333        S   Third
      9         1       2  female  14.0      1      0  30.0708        C  Second

           who  adult_male deck  embark_town alive  alone  age_median  age_random  \
```

```
0    man        True  NaN  Southampton   no  False        22.0        22.0
1  woman       False    C    Cherbourg  yes  False        38.0        38.0
2  woman       False  NaN  Southampton  yes   True        26.0        26.0
3  woman       False    C  Southampton  yes  False        35.0        35.0
4    man        True  NaN  Southampton   no   True        35.0        35.0
5    man        True  NaN   Queenstown   no   True        28.0        48.0
6    man        True    E  Southampton   no   True        54.0        54.0
7  child       False  NaN  Southampton   no  False         2.0         2.0
8  woman       False  NaN  Southampton  yes  False        27.0        27.0
9  child       False  NaN    Cherbourg  yes  False        14.0        14.0

   age_NAN  age_extreme
0        0     22.00000
1        0     38.00000
2        0     26.00000
3        0     35.00000
4        0     35.00000
5        1     73.27861
6        0     54.00000
7        0      2.00000
8        0     27.00000
9        0     14.00000
```

[20]:
```
fig=plt.figure()
ax=fig.add_subplot(111)

df["age"].plot(kind="kde",ax=ax,color="red")
df["age_median"].plot(kind="kde",ax=ax,color="green")
df["age_random"].plot(kind="kde",ax=ax,color="yellow")
df["age_extreme"].plot(kind="kde",ax=ax)

lines, lables=ax.get_legend_handles_labels()
ax.legend(lines, lables,loc="best")
plt.show()
```



5.1 Advantages

5.1.1 Easy to implement

5.1.2 Preserves sample size

5.1.3 Can be useful for variables with a skewed distribution

5.2 Disadvantages

5.2.1 Can introduce bias

5.2.2 Ignores relationships between variables

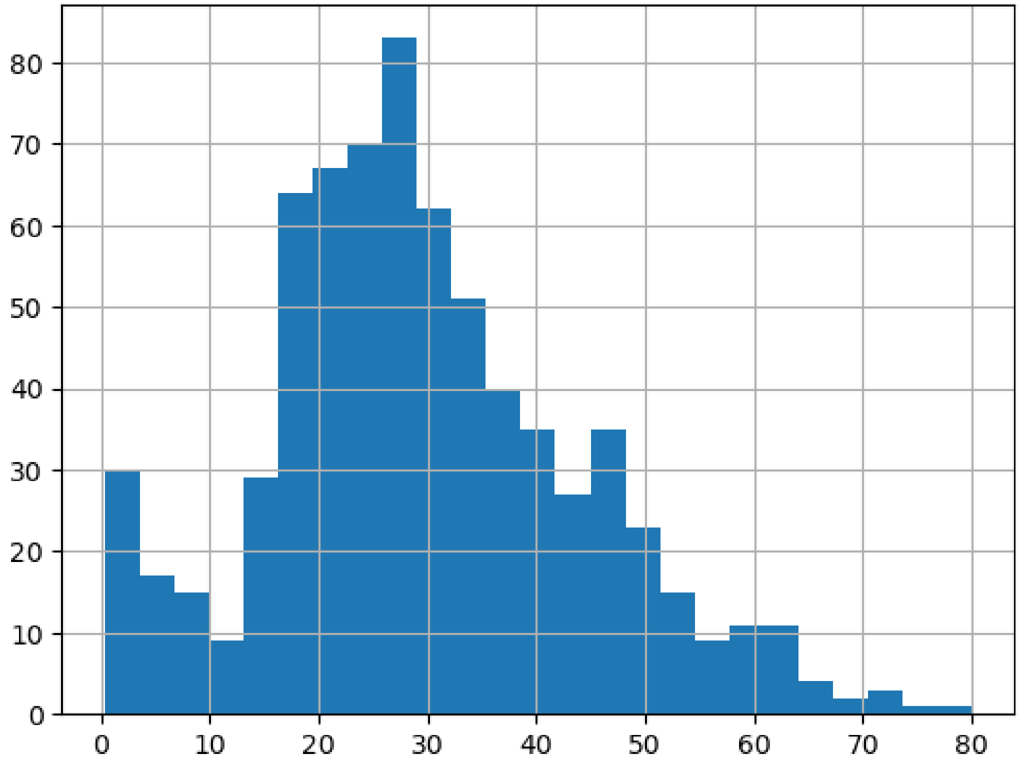
5.2.3 Results in loss of information

[ ]:
```

```

5.3 imputed columns vs original

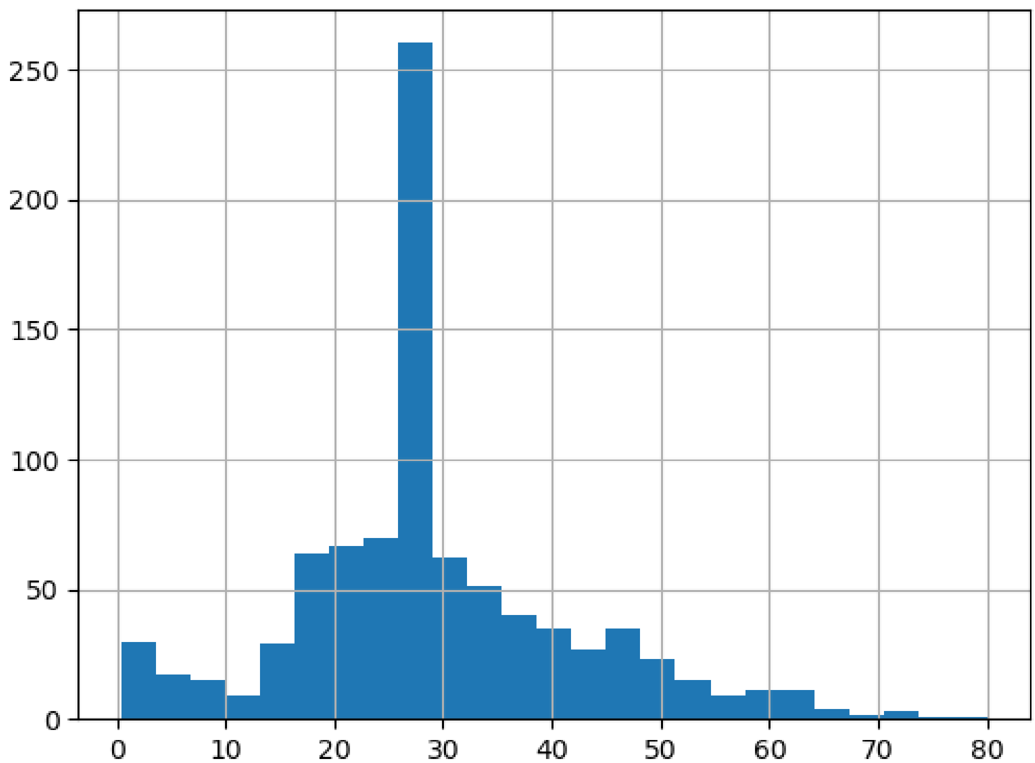
[21]:
```
df["age"].hist(bins=25)
```

[21]: <AxesSubplot: >



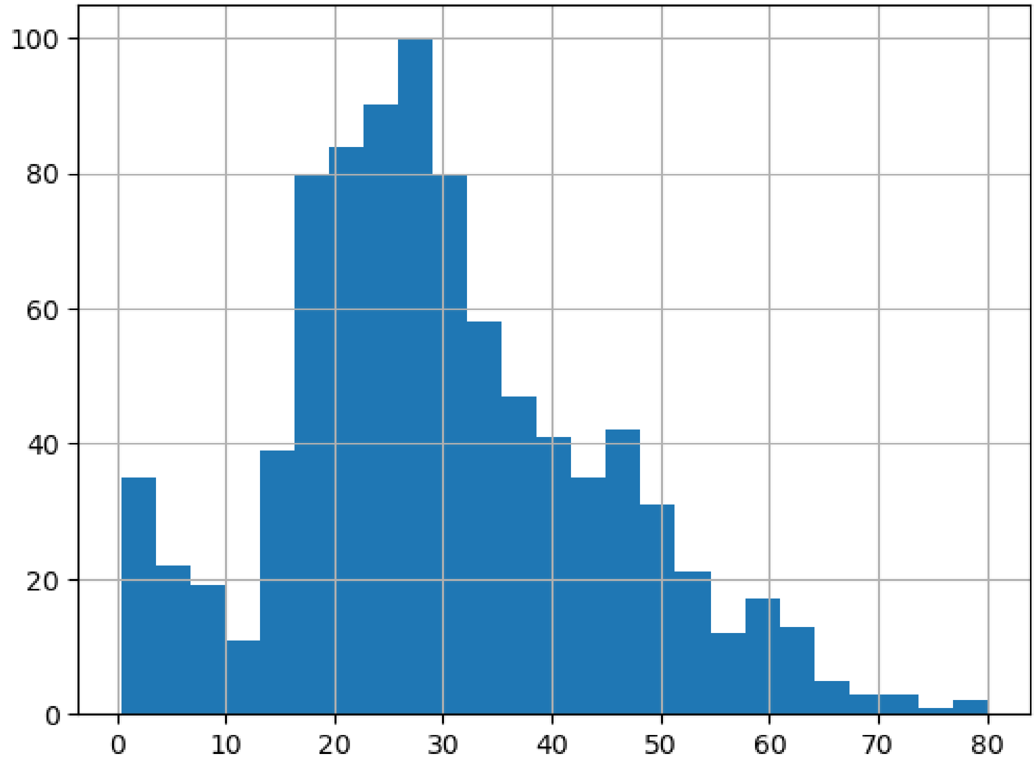

```
[22]: df["age_median"].hist(bins=25)
```

```
[22]: <AxesSubplot: >
```



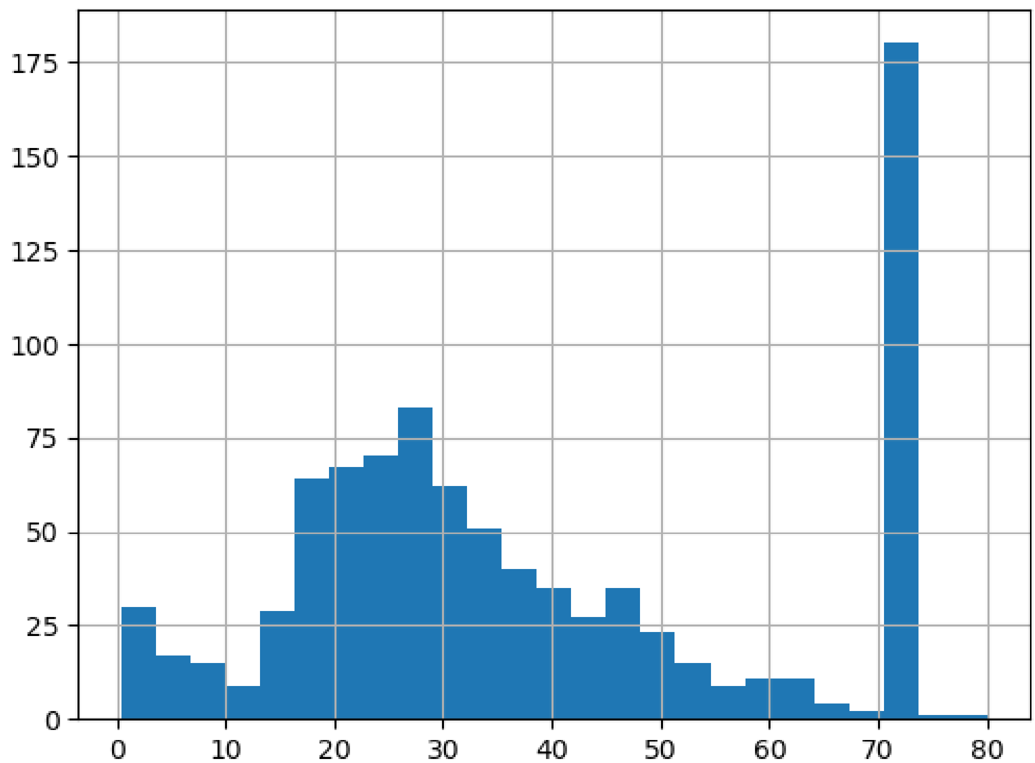
[23]:
```
df["age_random"].hist(bins=25)
```

[23]: <AxesSubplot: >



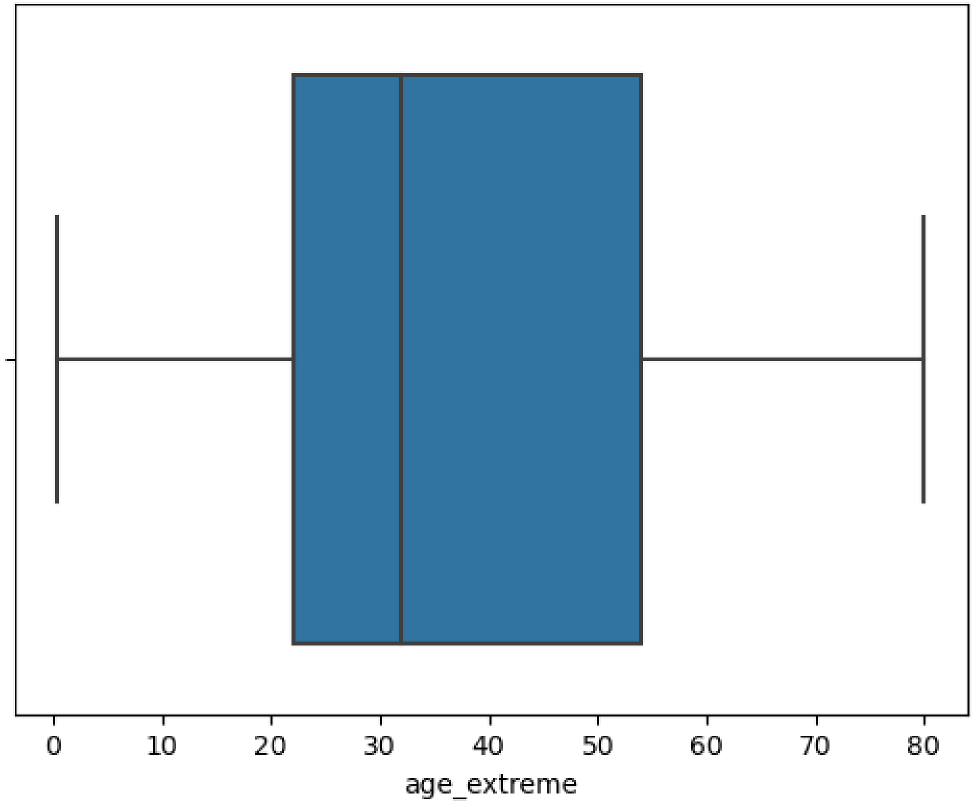
[24]:
```
df["age_extreme"].hist(bins=25)
```

[24]: <AxesSubplot: >




```
[25]: sns.boxplot(x="age_extreme",data=df)
```

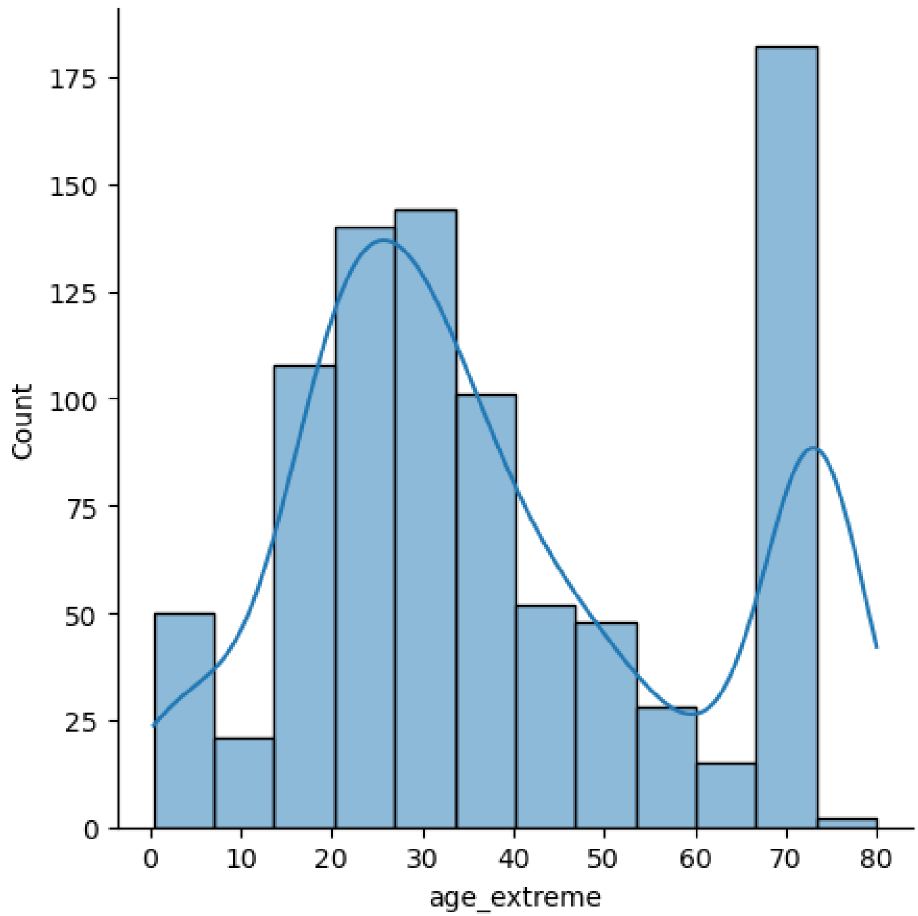
```
[25]: <AxesSubplot: xlabel='age_extreme'>
```



5.3.1 End of Distribution imputation removes outliers

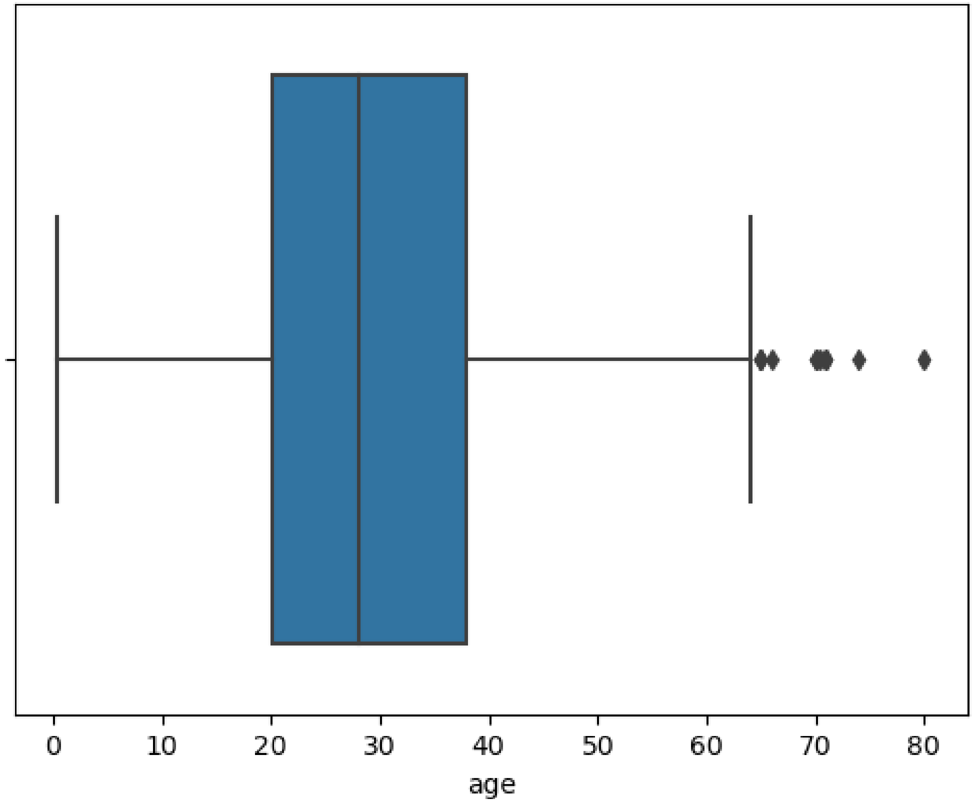
[26]:
```
sns.displot(df["age_extreme"],kde=True)
```

[26]:
```
<seaborn.axisgrid.FacetGrid at 0x7efd5ae44d90>
```



[27]:
```
sns.boxplot(x=df["age"])
```

[27]: <AxesSubplot: xlabel='age'>



[ ]:

```

```

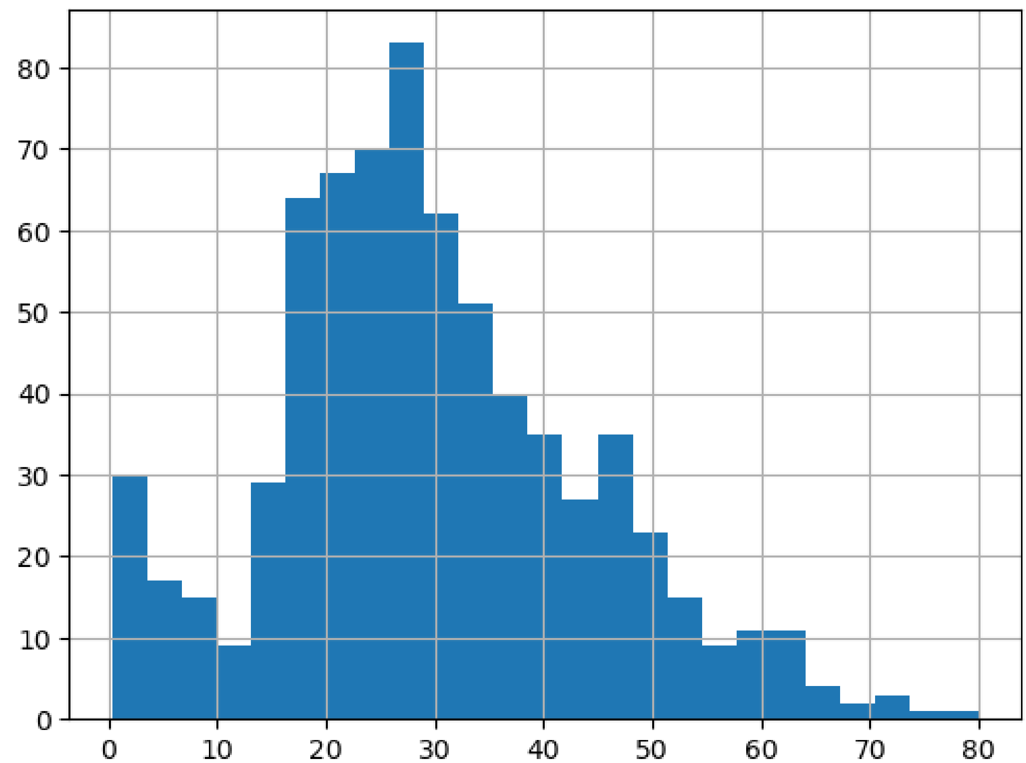
6 arbitrary value imputation

6.0.1 This technique was derived from kaggle competition It consists of replacing NAN by an arbitrary value

[28]:
```
def impute_nan(df,variable):
    df[variable+'_zero']=df[variable].fillna(0)
    df[variable+'_hundred']=df[variable].fillna(100)
```

[29]:
```
df['age'].hist(bins=25)
```

[29]: <AxesSubplot: >



6.1 Advantages

6.1.1 Easy to implement

6.1.2 Captures the importance of missingess if there is one

6.2 Disadvantages

6.2.1 Distorts the original distribution of the variable

6.2.2 If missingess is not important, it may mask the predictive power of the original variable by distorting its distribution

6.2.3 Hard to decide which value to use

[ ]:
```

```

7 handling categorical values

7.1 frequent category imputation

[30]:
```
import pandas as pd
df1=pd.read_csv("train.
    ↪csv",usecols=["BsmtQual","FireplaceQu","GarageType","SalePrice"])
df1.head()
# df1.columns
```

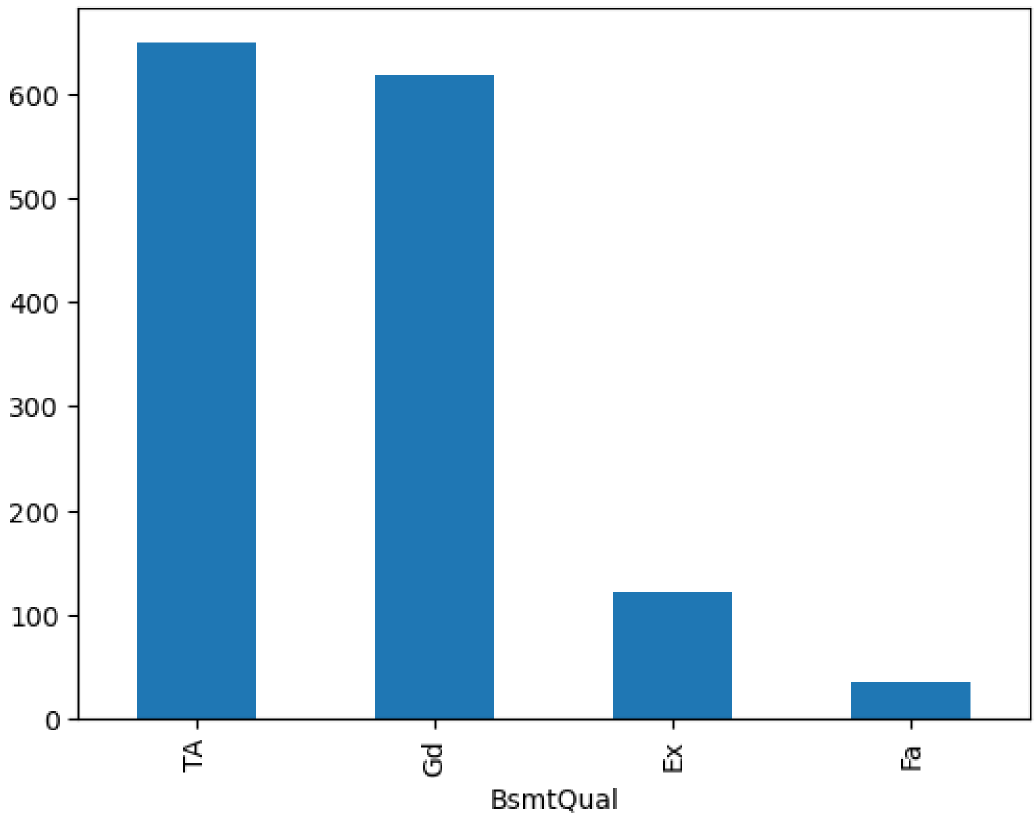
[30]:
```
  BsmtQual FireplaceQu GarageType  SalePrice
0       Gd         NaN     Attchd     208500
1       Gd          TA     Attchd     181500
2       Gd          TA     Attchd     223500
3       TA          Gd     Detchd     140000
4       Gd          TA     Attchd     250000
```

[31]:
```
df1.isnull().mean().sort_values(ascending=True)
```

[31]:
```
SalePrice      0.000000
BsmtQual       0.025342
GarageType     0.055479
FireplaceQu    0.472603
dtype: float64
```

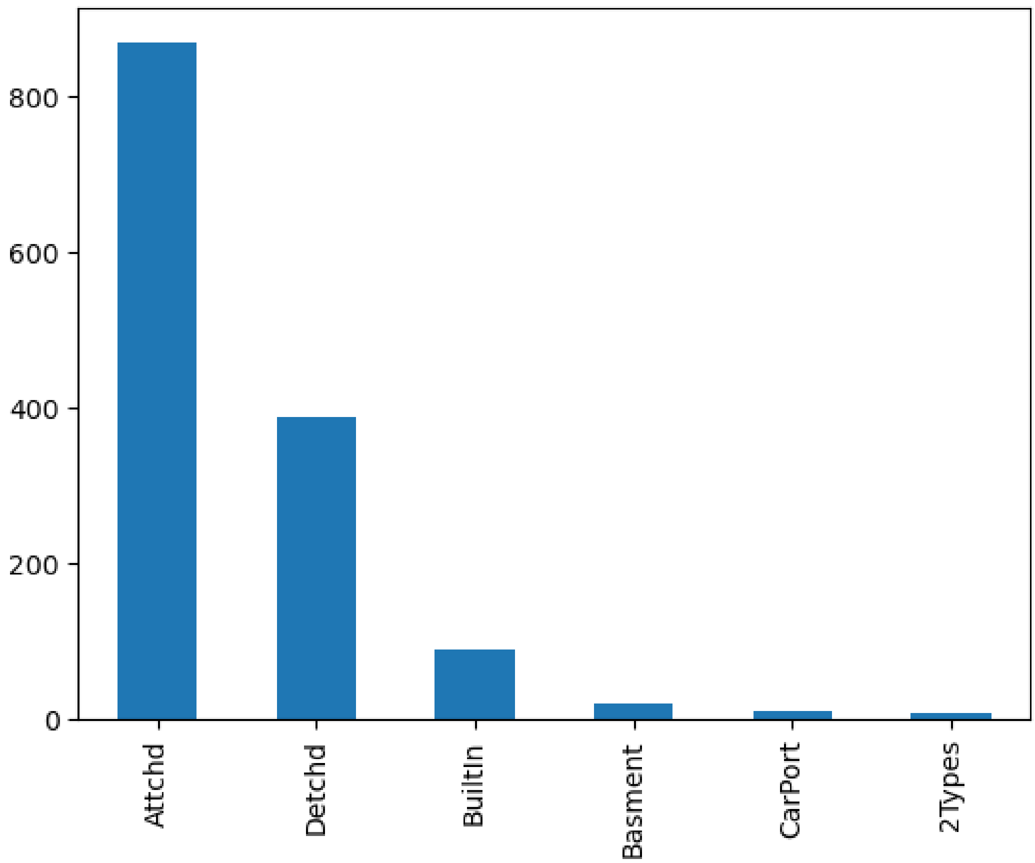
[32]:
```
df1.groupby("BsmtQual")["BsmtQual"].count().sort_values(ascending=False).plot.
    ↪bar()
```

[32]:
```
<AxesSubplot: xlabel='BsmtQual'>
```



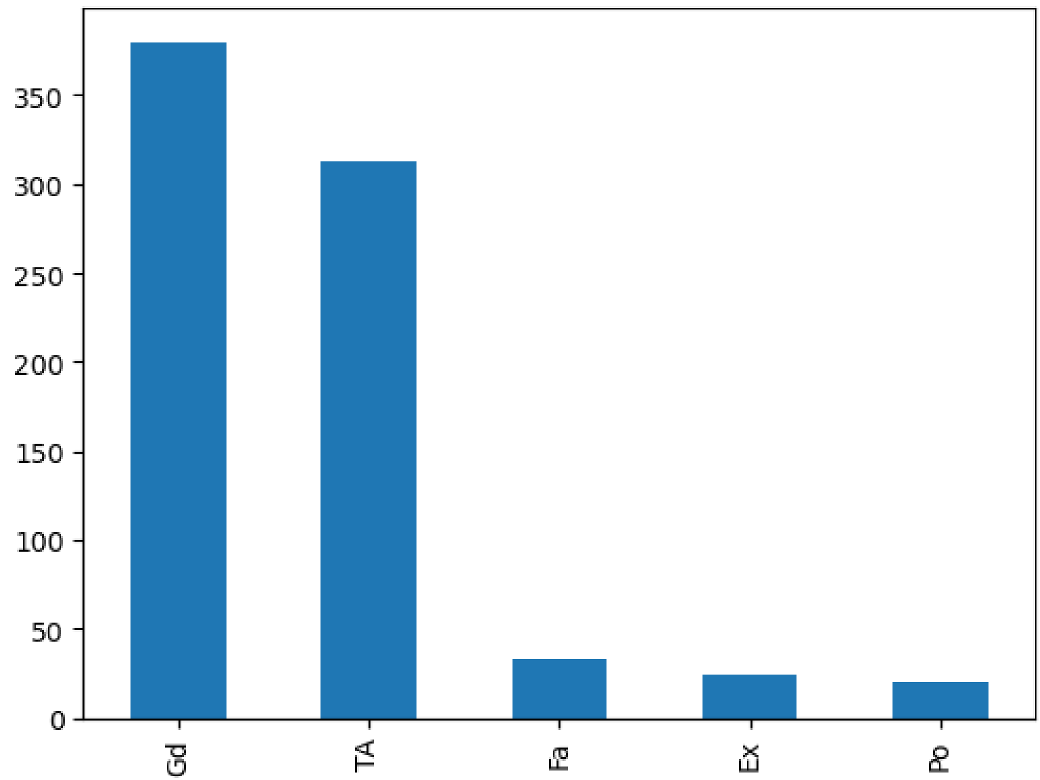
[33]:
```
df1["GarageType"].value_counts().plot.bar()
```

[33]: <AxesSubplot: >




```
[34]: df1["FireplaceQu"].value_counts().plot.bar()
```

[34]: <AxesSubplot: >



[35]:
```
def impute_nan_cat(df1,variable):
    most_frequent_category=df1[variable].mode()[0]
    df1[variable].fillna(most_frequent_category,inplace=True)
```

[36]:
```
for feature in ['BsmtQual','FireplaceQu','GarageType']:
    impute_nan_cat(df1,feature)
```

[37]:
```
df1.isnull().sum()
```

[37]:
```
BsmtQual       0
FireplaceQu    0
GarageType     0
SalePrice      0
dtype: int64
```

[38]:
```
df1.head()
```

[38]:

	BsmtQual	FireplaceQu	GarageType	SalePrice
0	Gd	Gd	Attchd	208500
1	Gd	TA	Attchd	181500
2	Gd	TA	Attchd	223500
3	TA	Gd	Detchd	140000

```
4      Gd        TA    Attchd    250000
```

7.2 Advantages

7.2.1 Easy To implement

7.2.2 Fater way to implement

7.3 Disadvantages

7.3.1 Since we are using the more frequent labels, it may use them in an over respresented way, if there are many nan's

7.3.2 It distorts the relation of the most frequent label

[ ]:

```

```