steps to follow in Data science 1) collecting raw data from real world web scrapping survey (online,offline) etc.... 2) Transform data into meaningful information structurised the data which we have collected from real world .csv .excel .database .big data 3) Perform Data cleaning Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled data cleaning :- sometimes in our data there is some missing values or null so we need to find correct information for those missing values sometimes in our data there is some not required things available then before we start analysis we will remove those things i have collection of record in that there is one column which contains only 5% record 95% records not available so that column is not useful information so will remove it 4) Analysis to find business insights from the data Exploratory Data Analysis we use statistics to perform analysis on the data 1) measures of central value analysis 2) probability for categorical data 3)inferetial statistics to find meaningful features sometimes we also use statistics to perform data cleaning process i have 1 column which contain continuous datatype values 92% records available 8% records are missing

```
        computer is not capable to deal with null values in machine learning so
        i need to fill 8% records with some values so i calculate mean of 92% record
        that calculated mean i will replace to 8% records this process is known as missing value
    treament
            sqfit   price
            230     10
            340     15
            280     12
            ?       15
            350     20

           ml wants complete data it can not understand null or missing values
           so i make my data cleaning process by replacing missing value with mean or median
           if it is continuous if it is categorical i will use mode
           in above example sqfit is continuous so we will calculate sqfit mean and replace ?
           with mean value
```

EDA process steps

1) identification of datatypes we have collection records and we want to understand which column have which kind of datatype numerical continuous descrete character /categorical data also ordinal nominal 2) analyzing basic metrics dataframe.describe() all numeric columns statistical summery mean,median,min,max,dev,s,variance 3) non- graphical univariate analysis we will take one single column and will perform

analysis on the column mean,mode,sd,variance,skewness checking 4) graphical univariate analysis dist plot,hist plot ,box plot,violinplot etc 5) non-graphical bivariate analysis correlation regression VIF all above statistics helps us to understand two variable relationship and it's distribution 6) graphical bivariate analysis line plot scatter plot heatmap 7) multi variate analysis graphical and non graphical 8) outlier treatments when we have positively and negatively skewd data then we need to perform outlier treatment minmax scaler standered scaler squarroot log10 conversion IQR (Lower whisker,upper whisker) 9) feature enginering 30k columns/features
i want to collect meaning or importent columns for ML for that we can do inferential statistics when we want to understand importance of data for training we will do inferetiral statistics wrapper,embded methods chi2,annova test etc... 22k 10) dimensionality reduction PCA (principal component analysis) 20k above columns

after above processing my data is ready for ML as computer machine can understand only numeric values so it is necesary to do EDA and all data cleaning process mentioned above

Measures of dispersion how much distance is there between data range percentile percentage IQR sd variance deviation

all above is come unde measure of dispersion how much your data is deviated from mean value SD it is also known as measure spread

-1 3 5 7 56 Q1 Q2 q3

here we can see low extream value and high extream value we can ignore IQR = q3-q1 =7 -3 = 4

```
4 is IQR
```

Mean Deviation : to check how our data is deviated from mean

12,12.5,12.3,11.1,10.5

In [43]:
```
x=[12,12.5,12.3,11.1,10.5]
```

In [44]:
```
x
```

Out[44]: `[12, 12.5, 12.3, 11.1, 10.5]`

In [45]:
```
m=sum(x)/5
```

In [46]:
```
m
```

Out[46]:
```
11.68
```

In [47]:
```
dis=[m-i for i in x]
```

In [48]:
```
dis
```

Out[48]:
```
[-0.3200000000000003,
 -0.8200000000000003,
 -0.620000000000001,
 0.5800000000000001,
 1.1799999999999997]
```

In [49]:
```
sum(dis)/2
```

Out[49]:
```
-8.881784197001252e-16
```

Data cleaning - preprocessing and EDA exploratory data analysis row data which we collect from real world that may have impurity so to make those data analyizable or machine learnable we need to perform above task 1) data clearning 2) EDA 1) identification of datatypes 2) analyzing basic metricss 3) univariate analysis 4) graphical univariate analysis 5) bivariate analysis 6) graphical bivariate analysis 7) multi variate analysis non-graphical and graphical 8) missing values treatments 9) outlier treatments 10) Label encoding 11) feature engineering 12) dimensionality reduction all above steps come under EDA and data preprocessing 1) statistics descreat stats mean,median,mode,sd,v,skew,correlation etc.... probability probability oods conditional probability inferential statistics :feature selection 2) python modules non graphical analysis numpy pandas scipy etc... graphical analysis matplotlib.pyplot seaborn

Import all required modules

In [50]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Read Data from csv file or other resources

In [51]:
```
# 1) goto whats up download Automobile_data.csv file
# 2) create folder under your ipynp file current folder
df=pd.read_csv("data.csv")
```

df is dataframe variable which is 2 dimensional array which store all the data from csv file to df

In [52]:
```
df.head(10) # first 10 records will be display
```

Out[52]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engine-size	horsepower	city-mpg	highway-mpg	price
0	3	?	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111	21	27	13495
1	3	?	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111	21	27	16500
2	1	?	alfa-romero	gas	hatchback	rwd	front	65.5	52.4	ohcv	152	154	19	26	16500
3	2	164	audi	gas	sedan	fwd	front	66.2	54.3	ohc	109	102	24	30	13950
4	2	164	audi	gas	sedan	4wd	front	66.4	54.3	ohc	136	115	18	22	17450
5	2	?	audi	gas	sedan	fwd	front	66.3	53.1	ohc	136	110	19	25	15250
6	1	158	audi	gas	sedan	fwd	front	71.4	55.7	ohc	136	110	19	25	17710
7	1	?	audi	gas	wagon	fwd	front	71.4	55.7	ohc	136	110	19	25	18920
8	1	158	audi	gas	sedan	fwd	front	71.4	55.9	ohc	131	140	17	20	23875
9	0	?	audi	gas	hatchback	4wd	front	67.9	52.0	ohc	131	160	16	22	12000

In [53]:
```
df
```

Out[53]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engine-size	horsepower	city-mpg	highway-mpg	price

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engine-size	horsepower	city-mpg	highway-mpg	price
0	3	?	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111	21	27	13495
1	3	?	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111	21	27	16500
2	1	?	alfa-romero	gas	hatchback	rwd	front	65.5	52.4	ohcv	152	154	19	26	16500
3	2	164	audi	gas	sedan	fwd	front	66.2	54.3	ohc	109	102	24	30	13950
4	2	164	audi	gas	sedan	4wd	front	66.4	54.3	ohc	136	115	18	22	17450
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
200	-1	95	volvo	gas	sedan	rwd	front	68.9	55.5	ohc	141	114	23	28	16845
201	-1	95	volvo	gas	sedan	rwd	front	68.8	55.5	ohc	141	160	19	25	19045
202	-1	95	volvo	gas	sedan	rwd	front	68.9	55.5	ohcv	173	134	18	23	21485
203	-1	95	volvo	diesel	sedan	rwd	front	68.9	55.5	ohc	145	106	26	27	22470
204	-1	95	volvo	gas	sedan	rwd	front	68.9	55.5	ohc	141	114	19	25	22625

205 rows × 15 columns

1) Analyzing the Datatypes of each columns

all 15 columns are giving some information so 1) they must have some datatype in python int,float,object,string,boolean 2) they must have some datatye in statistics numeric descrete continuous character / categorical data type ordinal nominal each column we need to analyz

In [54]:

```
# gather information for all columns in term of size,datatype,rows,columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
```

```
 1   normalized-losses  205 non-null    object
 2   make               205 non-null    object
 3   fuel-type          205 non-null    object
 4   body-style         205 non-null    object
 5   drive-wheels       205 non-null    object
 6   engine-location    205 non-null    object
 7   width              205 non-null    float64
 8   height             205 non-null    float64
 9   engine-type        205 non-null    object
 10  engine-size        205 non-null    int64
 11  horsepower         205 non-null    object
 12  city-mpg           205 non-null    int64
 13  highway-mpg        205 non-null    int64
 14  price              205 non-null    int64
dtypes: float64(2), int64(5), object(8)
memory usage: 24.1+ KB
```

numeric symboling -> descrete price -> descrete normalized-losses -> continuous width -> continuous height -> continuous engine-size -> continous horsepower -> continuous city-mpg ->continuous highway-mpg -> continuous categorical make - nominal fuel-type - nominal body-style - nominal drive wheels - nominal engine-location -nominal engine-type -nominal descrete and continuous data type float and integer

2) Data Cleaning

In [55]:

```
# missing value treatment
df['symboling'].value_counts()
```

Out[55]:

```
 0    67
 1    54
 2    32
 3    27
-1    22
-2     3
Name: symboling, dtype: int64
```

In [56]:

```
df['normalized-losses'].value_counts()
```

Out[56]:

```
?      41
161    11
91      8
150     7
134     6
```

104	6
128	6
94	5
65	5
95	5
74	5
168	5
85	5
102	5
103	5
122	4
106	4
148	4
93	4
118	4
125	3
137	3
83	3
154	3
101	3
115	3
129	2
89	2
108	2
188	2
197	2
87	2
153	2
110	2
192	2
194	2
164	2
158	2
119	2
113	2
81	2
145	2
186	1
107	1
90	1
142	1
256	1
77	1
78	1
231	1

```
98      1
121     1
Name: normalized-losses, dtype: int64
```

In [57]:
```
df['normalized-losses'].dtype
```

Out[57]:
```
dtype('O')
```

In [58]:
```
# data cleaning process for continuous datatype variable
# ? is special symbols 41 records have
# replace ? with nan values
df['normalized-losses'].replace('?',np.nan,inplace=True)
# change datatype of normalized-losses from object to integer
df['normalized-losses']=df['normalized-losses'].astype(float)
# find mean of all normalized-losses column and replace nan values with that mean value
df['normalized-losses'].fillna(df['normalized-losses'].mean(),inplace=True)

# df['normalized-losses']=df['normalized-losses'].fillna(df['normalized-losses'].mean())
```

In [59]:
```
df
```

Out[59]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engine-size	horsepower	city-mpg	highway-mpg	price
0	3	122.0	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111	21	27	13495
1	3	122.0	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111	21	27	16500
2	1	122.0	alfa-romero	gas	hatchback	rwd	front	65.5	52.4	ohcv	152	154	19	26	16500
3	2	164.0	audi	gas	sedan	fwd	front	66.2	54.3	ohc	109	102	24	30	13950
4	2	164.0	audi	gas	sedan	4wd	front	66.4	54.3	ohc	136	115	18	22	17450
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
200	-1	95.0	volvo	gas	sedan	rwd	front	68.9	55.5	ohc	141	114	23	28	16845

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engine-size	horsepower	city-mpg	highway-mpg	price
201	-1	95.0	volvo	gas	sedan	rwd	front	68.8	55.5	ohc	141	160	19	25	19045
202	-1	95.0	volvo	gas	sedan	rwd	front	68.9	55.5	ohcv	173	134	18	23	21485
203	-1	95.0	volvo	diesel	sedan	rwd	front	68.9	55.5	ohc	145	106	26	27	22470
204	-1	95.0	volvo	gas	sedan	rwd	front	68.9	55.5	ohc	141	114	19	25	22625

205 rows × 15 columns

In [60]:

```
print("mean => ",df['normalized-losses'].mean())
print("median => ",df['normalized-losses'].median())

# becasue mean and median is same so we willreplace missing values with mean only
# whenever mean and median have diff then we will go with median
```

```
mean =>  122.0
median =>  122.0
```

In [61]:

```
# missing value function
# x is column
# s is symbols
def missingvalues(x,s):
    df[x].replace(s,np.nan,inplace=True)
    df[x]=df[x].astype(float)
    if df[x].mean() == df[x].median():

        df[x].fillna(df[x].mean(),inplace=True)
    else:
         df[x].fillna(df[x].median(),inplace=True)
    print(df[x].value_counts())

# missingvalues('normalized-losses','?')
```

In [62]:

```
df['horsepower'].value_counts()
```

Out[62]:
```
68     19
70     11
69     10
116     9
110     8
95      7
114     6
62      6
160     6
101     6
88      6
82      5
76      5
145     5
84      5
97      5
102     5
111     4
86      4
123     4
92      4
90      3
73      3
85      3
121     3
207     3
152     3
182     3
?       2
162     2
52      2
155     2
184     2
94      2
112     2
161     2
56      2
156     2
176     2
100     2
262     1
134     1
60      1
64      1
48      1
```

```
200      1
106      1
72       1
78       1
140      1
142      1
143      1
135      1
288      1
115      1
175      1
58       1
154      1
55       1
120      1
Name: horsepower, dtype: int64
```

In [63]:

```
missingvalues('horsepower','?')
```

```
68.0      19
70.0      11
69.0      10
95.0       9
116.0      9
110.0      8
88.0       6
114.0      6
160.0      6
101.0      6
62.0       6
82.0       5
84.0       5
97.0       5
76.0       5
145.0      5
102.0      5
86.0       4
123.0      4
111.0      4
92.0       4
121.0      3
73.0       3
152.0      3
207.0      3
```

```
85.0      3
90.0      3
182.0     3
100.0     2
112.0     2
176.0     2
161.0     2
156.0     2
56.0      2
52.0      2
155.0     2
184.0     2
162.0     2
94.0      2
48.0      1
140.0     1
115.0     1
154.0     1
200.0     1
58.0      1
60.0      1
78.0      1
262.0     1
135.0     1
288.0     1
64.0      1
120.0     1
72.0      1
134.0     1
175.0     1
143.0     1
55.0      1
142.0     1
106.0     1
Name: horsepower, dtype: int64
```

3) see General Metrics

In [64]:

```
df.head(10)
```

Out[64]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engine-size	horsepower	city-mpg	highway-mpg	price
0	3	122.0	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111.0	21	27	13495
1	3	122.0	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111.0	21	27	16500
2	1	122.0	alfa-romero	gas	hatchback	rwd	front	65.5	52.4	ohcv	152	154.0	19	26	16500
3	2	164.0	audi	gas	sedan	fwd	front	66.2	54.3	ohc	109	102.0	24	30	13950
4	2	164.0	audi	gas	sedan	4wd	front	66.4	54.3	ohc	136	115.0	18	22	17450
5	2	122.0	audi	gas	sedan	fwd	front	66.3	53.1	ohc	136	110.0	19	25	15250
6	1	158.0	audi	gas	sedan	fwd	front	71.4	55.7	ohc	136	110.0	19	25	17710
7	1	122.0	audi	gas	wagon	fwd	front	71.4	55.7	ohc	136	110.0	19	25	18920
8	1	158.0	audi	gas	sedan	fwd	front	71.4	55.9	ohc	131	140.0	17	20	23875
9	0	122.0	audi	gas	hatchback	4wd	front	67.9	52.0	ohc	131	160.0	16	22	12000

In [65]:

```
# all numeric columns general metrics we can get using DataFrame describe function
df.describe()
```

Out[65]:

	symboling	normalized-losses	width	height	engine-size	horsepower	city-mpg	highway-mpg	price
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	0.834146	122.000000	65.907805	53.724878	126.907317	104.165854	25.219512	30.751220	13227.478049
std	1.245307	31.681008	2.145204	2.443522	41.642693	39.529733	6.542142	6.886443	7902.651615
min	-2.000000	65.000000	60.300000	47.800000	61.000000	48.000000	13.000000	16.000000	5118.000000
25%	0.000000	101.000000	64.100000	52.000000	97.000000	70.000000	19.000000	25.000000	7788.000000
50%	1.000000	122.000000	65.500000	54.100000	120.000000	95.000000	24.000000	30.000000	10345.000000
75%	2.000000	137.000000	66.900000	55.500000	141.000000	116.000000	30.000000	34.000000	16500.000000
max	3.000000	256.000000	72.300000	59.800000	326.000000	288.000000	49.000000	54.000000	45400.000000

In [66]:
```
corr=df.corr()
```

In [67]:
```
corr
```

Out[67]:

	symboling	normalized-losses	width	height	engine-size	horsepower	city-mpg	highway-mpg	price
symboling	1.000000	0.465190	-0.232919	-0.541038	-0.105790	0.071064	-0.035823	0.034606	-0.085781
normalized-losses	0.465190	1.000000	0.084195	-0.370706	0.110997	0.203380	-0.218749	-0.178221	0.133424
width	-0.232919	0.084195	1.000000	0.279210	0.735433	0.641337	-0.642704	-0.677218	0.718253
height	-0.541038	-0.370706	0.279210	1.000000	0.067149	-0.109286	-0.048640	-0.107358	0.132444
engine-size	-0.105790	0.110997	0.735433	0.067149	1.000000	0.810216	-0.653658	-0.677470	0.852995
horsepower	0.071064	0.203380	0.641337	-0.109286	0.810216	1.000000	-0.802170	-0.770780	0.747445
city-mpg	-0.035823	-0.218749	-0.642704	-0.048640	-0.653658	-0.802170	1.000000	0.971337	-0.654611
highway-mpg	0.034606	-0.178221	-0.677218	-0.107358	-0.677470	-0.770780	0.971337	1.000000	-0.679048
price	-0.085781	0.133424	0.718253	0.132444	0.852995	0.747445	-0.654611	-0.679048	1.000000

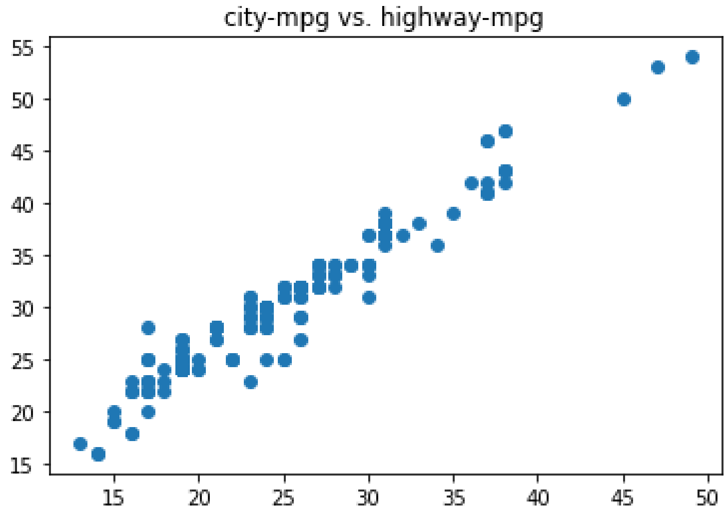
1= perfect correlation 0.90 or near to 1 positive number high,low,perfect

- means negatively high,low,perfect correlation 0 means no correlation correlation is comparision or relationship between two variable it comes under buvariate analysis but we can have general metrics like given method

Graphical Analysis with scatter plot to check correlation between numeric variable or features

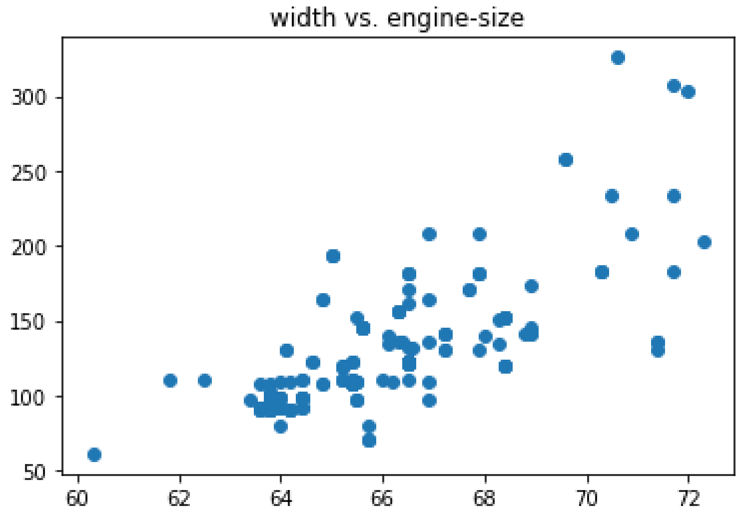
In [68]:
```
x=df['city-mpg']
y=df['highway-mpg']
plt.title("city-mpg vs. highway-mpg")
plt.scatter(x,y)
plt.show()
```

```
# so we can see there is low high correlation
```



In [69]:

```
x=df['width']
y=df['engine-size']
plt.title("width vs. engine-size")
plt.scatter(x,y)
plt.show()
```



Univariate Analysis

In [70]:

```
# categorical data analysis
df['make'].value_counts()
```

Out[70]:

```
toyota           32
nissan           18
mazda            17
honda            13
mitsubishi       13
subaru           12
volkswagen       12
volvo            11
peugot           11
dodge             9
bmw               8
mercedes-benz     8
plymouth          7
audi              7
saab              6
porsche           5
isuzu             4
alfa-romero       3
```

```
chevrolet         3
jaguar            3
renault           2
mercury           1
Name: make, dtype: int64
```

In [71]:
```
df['fuel-type'].value_counts()
```

Out[71]:
```
gas       185
diesel     20
Name: fuel-type, dtype: int64
```

In [72]:
```
df['body-style'].value_counts()
```

Out[72]:
```
sedan          96
hatchback      70
wagon          25
hardtop         8
convertible     6
Name: body-style, dtype: int64
```

In [73]:
```
# univariate for continuous data
```

In [74]:
```
df['highway-mpg'].value_counts()
```

Out[74]:
```
25    19
24    17
38    17
30    16
32    16
34    14
37    13
28    13
29    10
33     9
31     8
22     8
23     7
27     5
```

```
43      4
41      3
42      3
26      3
20      2
19      2
18      2
16      2
36      2
39      2
46      2
47      2
53      1
50      1
17      1
54      1
Name: highway-mpg, dtype: int64
```

In [75]:

```
df['price'].value_counts()
```

Out[75]:

```
12000    2
8845     2
7609     2
5572     2
8495     2
        ..
12964    1
16430    1
7126     1
36000    1
40960    1
Name: price, Length: 189, dtype: int64
```

In [76]:

```
print("average car price using mean => ",df['price'].mean())
print("average car price using median => ",df['price'].median())
```

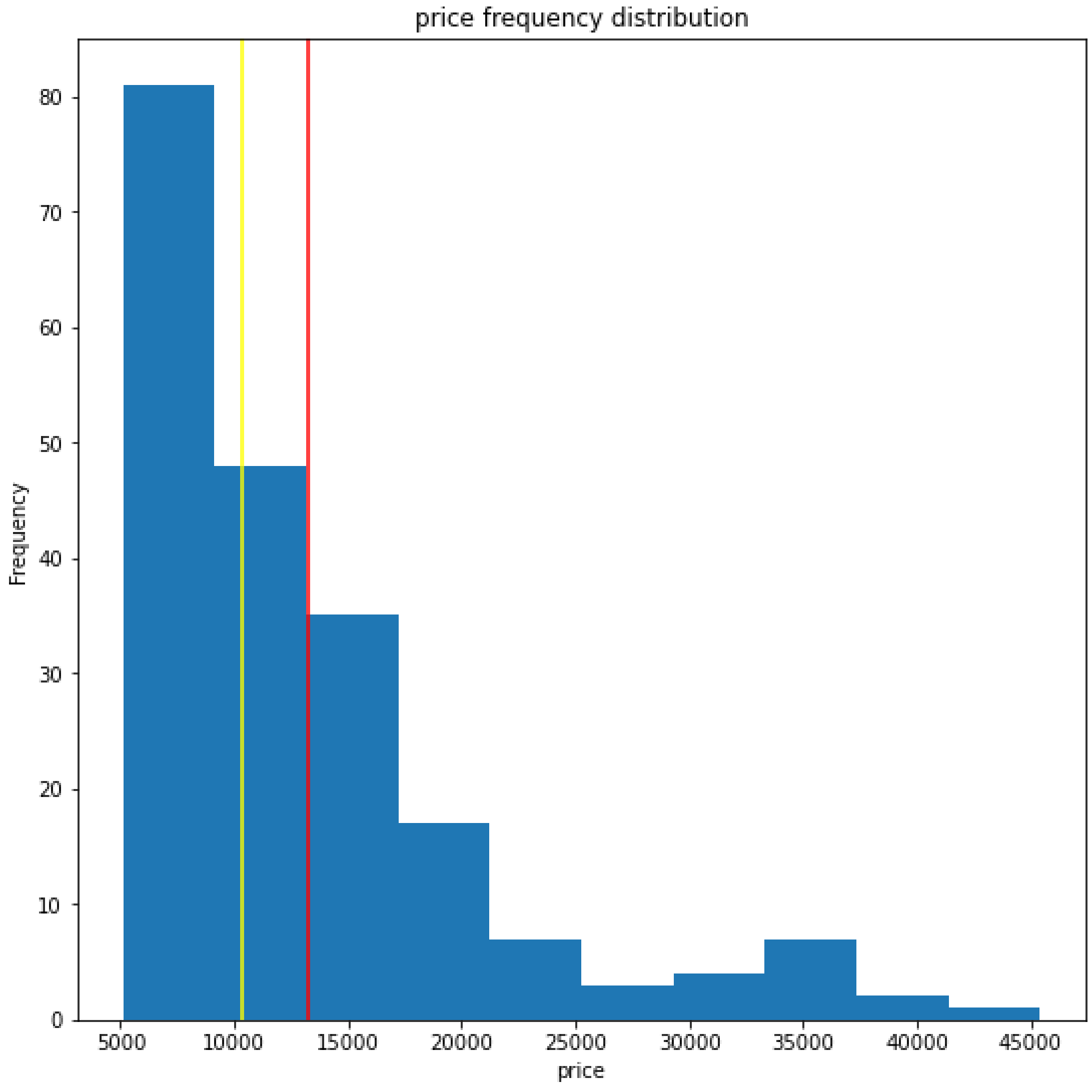
```
average car price using mean =>  13227.478048780487
average car price using median =>  10345.0
```

In [77]:

```
# i can see data have some outlier as mean and median is diff shows no - normalization in data
```

In [78]:

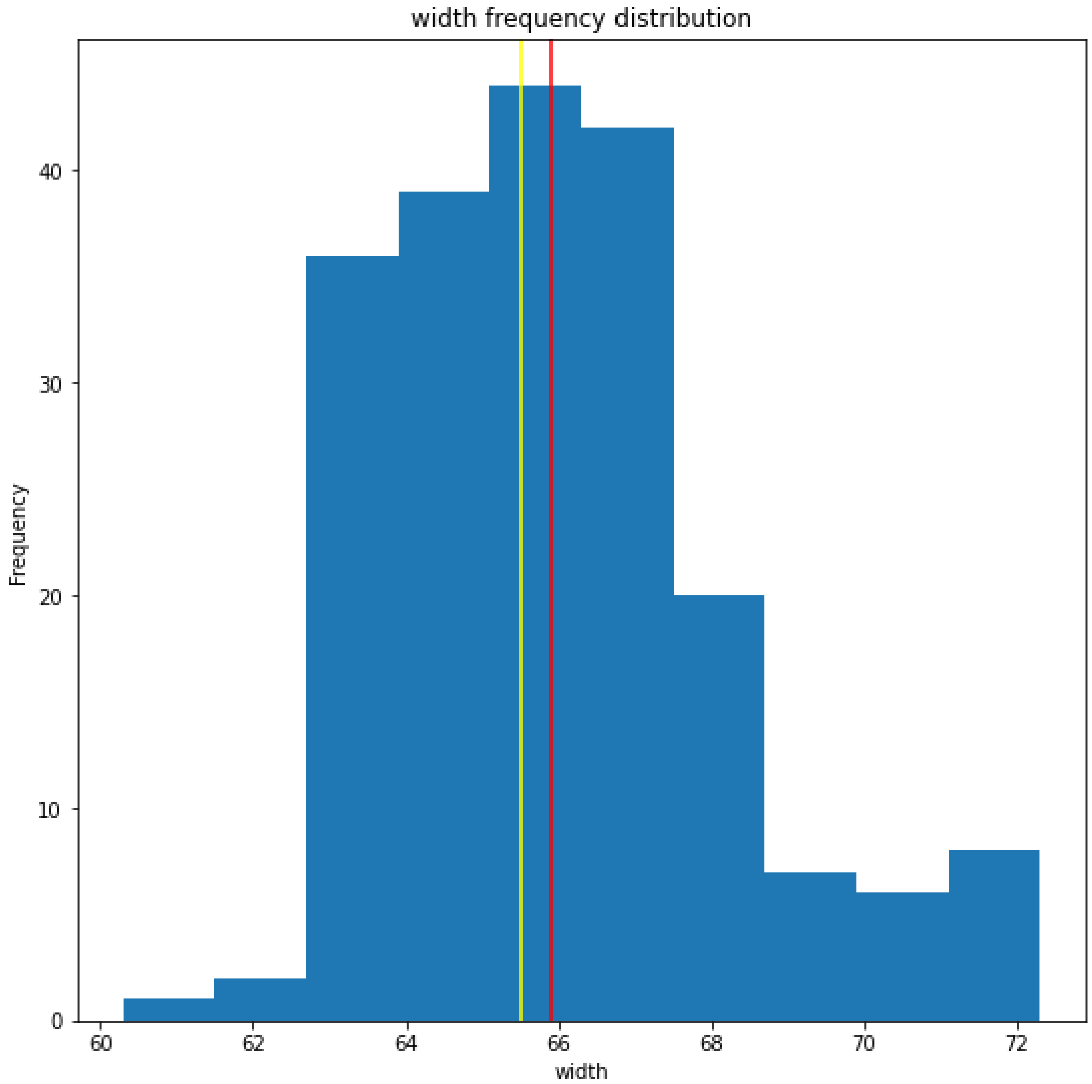
```
# using histogram we will check price column distribution
plt.figure(figsize=(9,9))
plt.hist(df['price'],bins=10)
plt.title("price frequency distribution")
plt.axvline(df["price"].mean(),color="red")
plt.axvline(df["price"].median(),color="yellow")
plt.ylabel("Frequency")
plt.xlabel("price")
plt.show()
```



above grpah showing right side skewness in price distribution that means we have positive extream values it is also known as outliers

In [79]:
```
# using histogram we will check price column distribution
plt.figure(figsize=(9,9))
plt.hist(df['width'],bins=10)
plt.title("width frequency distribution")
plt.axvline(df["width"].mean(),color="red")
```

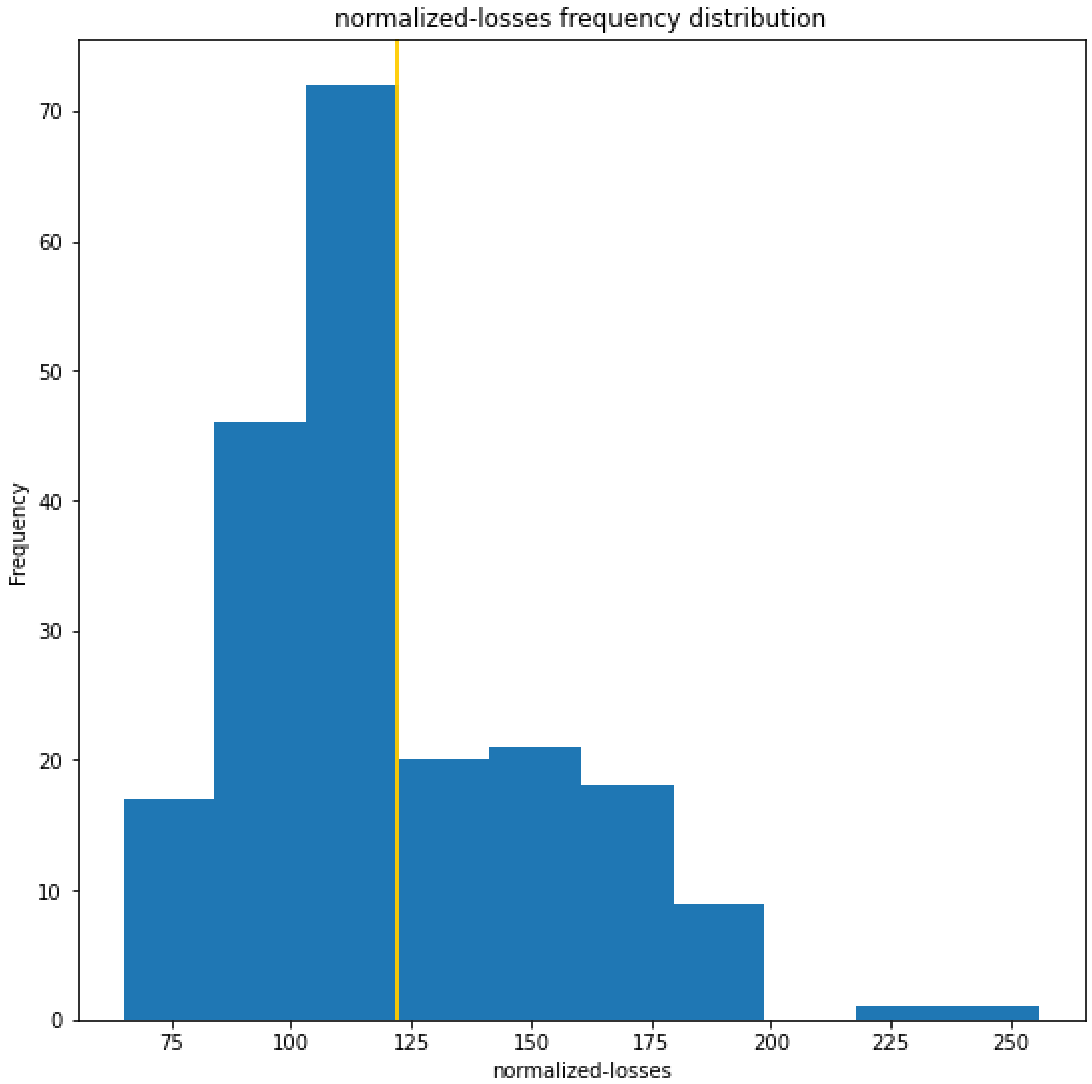
```
plt.axvline(df["width"].median(),color="yellow")
plt.ylabel("Frequency")
plt.xlabel("width")
plt.show()
```



In [80]:
```
df.columns
```

Out[80]:
```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'body-style',
       'drive-wheels', 'engine-location', 'width', 'height', 'engine-type',
       'engine-size', 'horsepower', 'city-mpg', 'highway-mpg', 'price'],
      dtype='object')
```

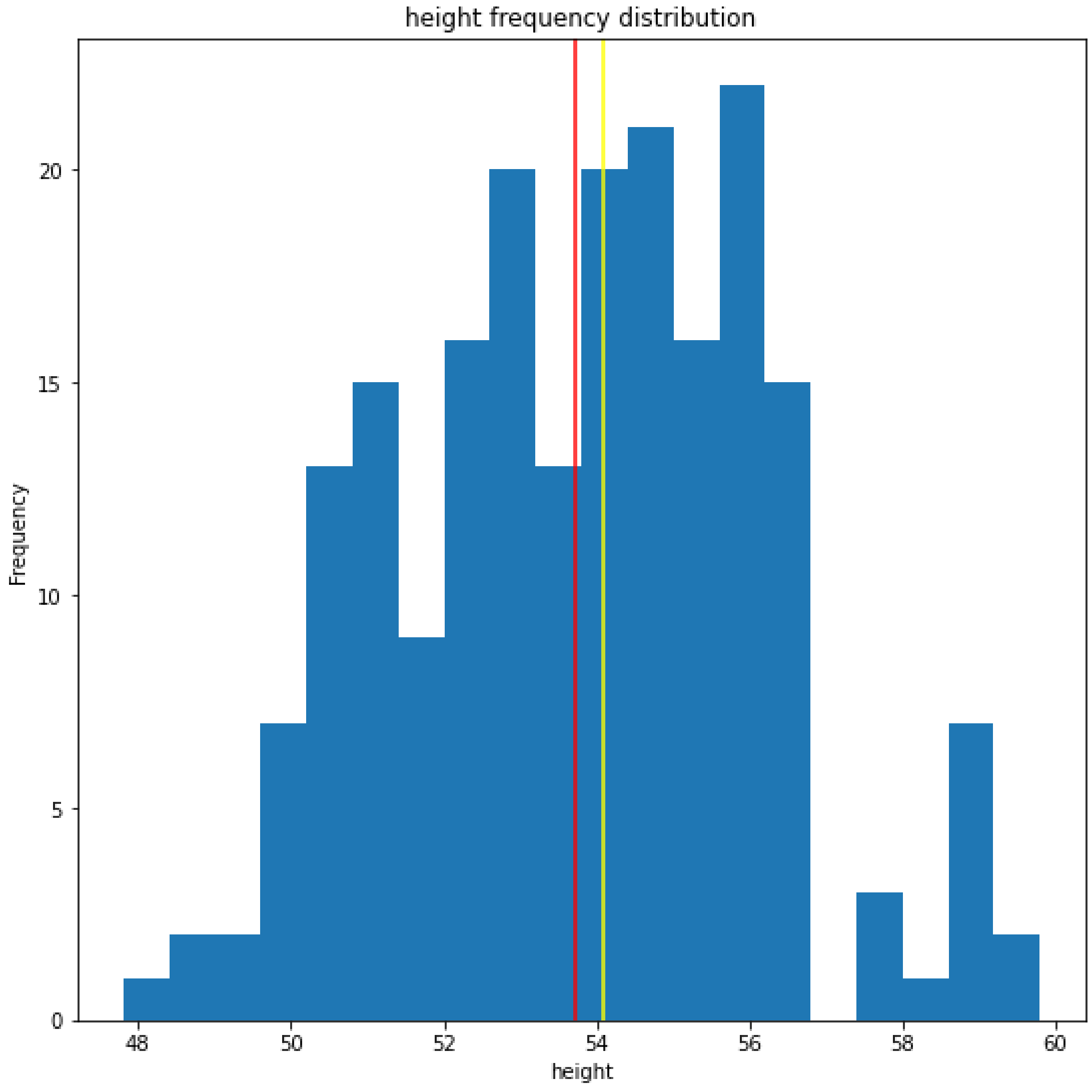
In [81]:
```
# using histogram we will check price column distribution
plt.figure(figsize=(9,9))
plt.hist(df['normalized-losses'],bins=10)
plt.title("normalized-losses frequency distribution")
plt.axvline(df["normalized-losses"].mean(),color="red")
plt.axvline(df["normalized-losses"].median(),color="yellow")
plt.ylabel("Frequency")
plt.xlabel("normalized-losses")
plt.show()
```



In [82]:

```
# using histogram we will check price column distribution
plt.figure(figsize=(9,9))
plt.hist(df['height'],bins=20)
plt.title("height frequency distribution")
plt.axvline(df["height"].mean(),color="red")
plt.axvline(df["height"].median(),color="yellow")
```

```
plt.ylabel("Frequency")
plt.xlabel("height")
plt.show()
```



1) perform scatter plot for all numeric 2) perform univariate analysis for all numeric and categorical columns 3) correlation make example of correlation using numpy and visualise correlation with matplotlib take any random data Till now recap 1) EDA and Data preprocessing 2) steps and task which we need to perform 3) understand your datatypes 4) data cleaning where learning to get rid of missing values can be special symbols or nan type values 4) univariate graphical and

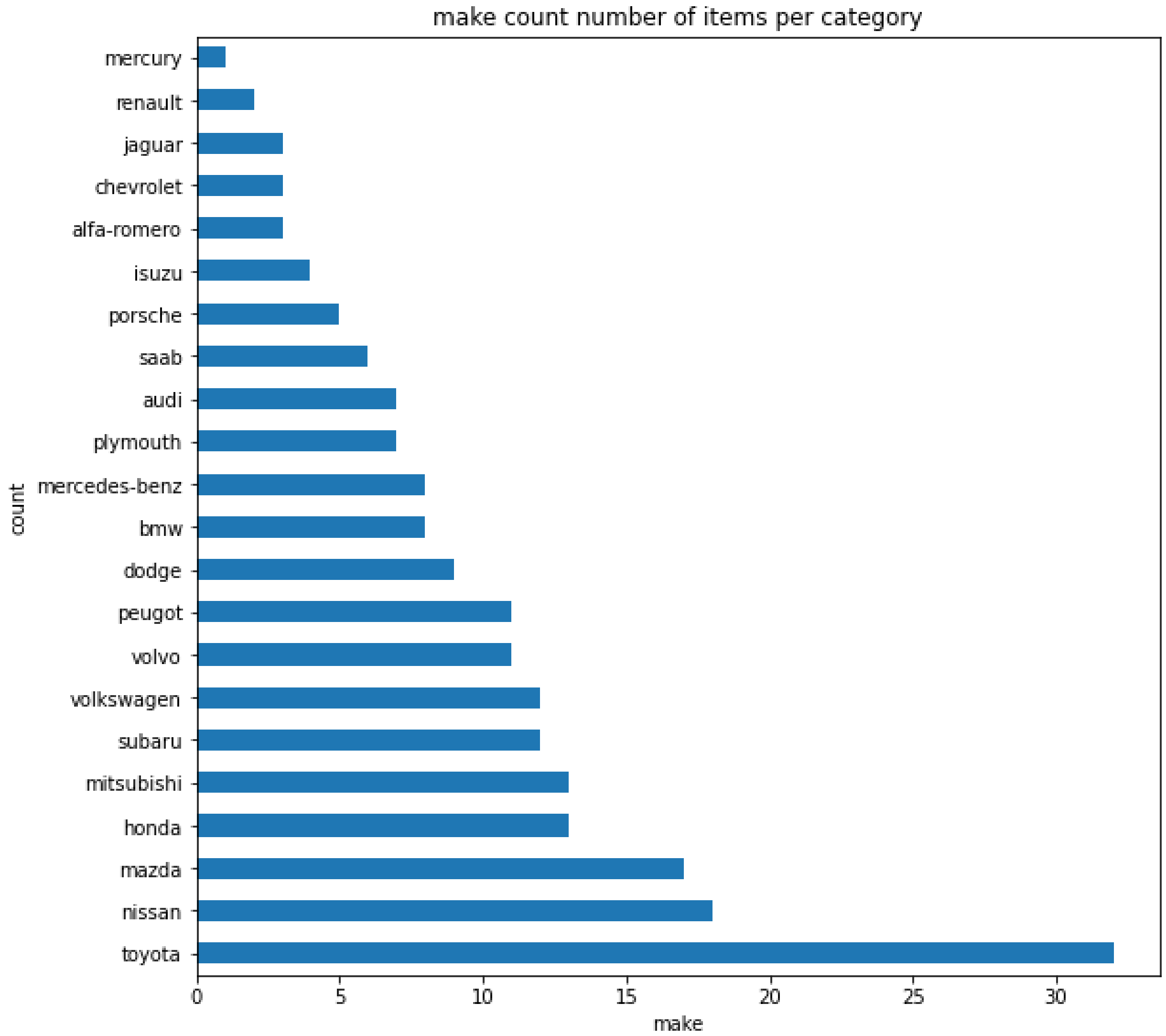
non graphical analysis 5) bivariate graphical and non graphical 6) basic metrics covers in this -1 , 2 ,3 ,4, 2.5, 3.5,4.2,15,12,15 mean 7.8 median 4 so above data is not normally distributed so at that mean is influanced by extream values -1 15 12 when mean and median is same or near to each other then data is normally distributed when mean and median have huge diff then data is not normally distributed i replace missing values using mean then i am making my missing data also bias so at that time we will use median

Categorical Data Univariate analysis

1) Dist plot see the distribution of our continuous or descrete dataset outlier analysis then boxplot,violin plot # 2) bar plot when we want to observe catagorical data distribution we will go for barplot when we want to check percentage wise ratio of categorical data we can use pie plot

In [83]:

```
# using bar plot we can perform categorical data univariate analysis
plt.figure(figsize=(9,9))
#df["make"].value_counts().plot(kind="bar") # vertical bar
df["make"].value_counts().plot(kind="barh") # vertical bar
plt.title("make count number of items per category")
plt.ylabel("count")
plt.xlabel("make")
plt.show()
```



In [84]:

```
df["make"].value_counts()
```

Out[84]:

```
toyota          32
nissan          18
mazda           17
honda           13
```

```
mitsubishi       13
subaru           12
volkswagen       12
volvo            11
peugot           11
dodge             9
bmw               8
mercedes-benz     8
plymouth          7
audi              7
saab              6
porsche           5
isuzu             4
alfa-romero       3
chevrolet         3
jaguar            3
renault           2
mercury           1
Name: make, dtype: int64
```

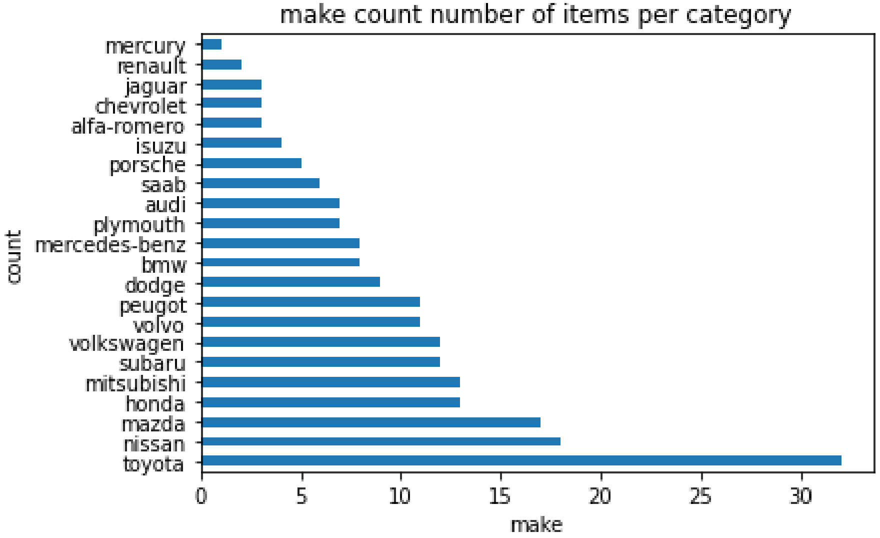
In [85]:

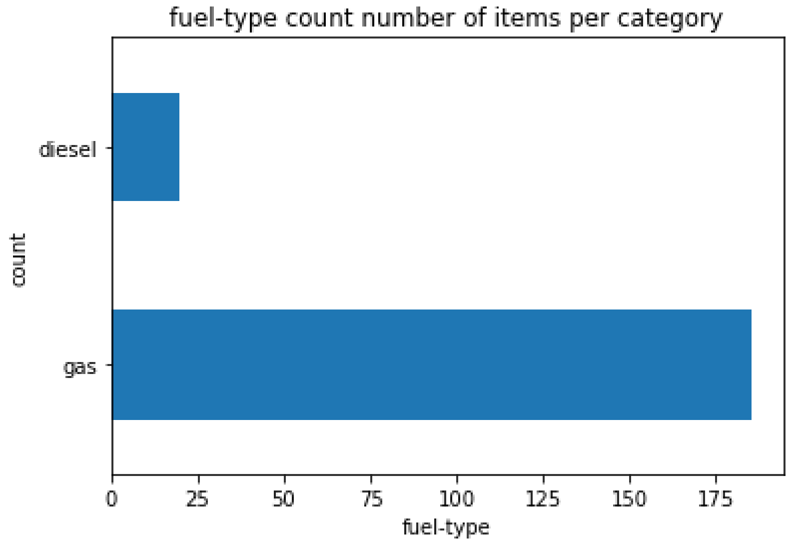
```
df.info()
```

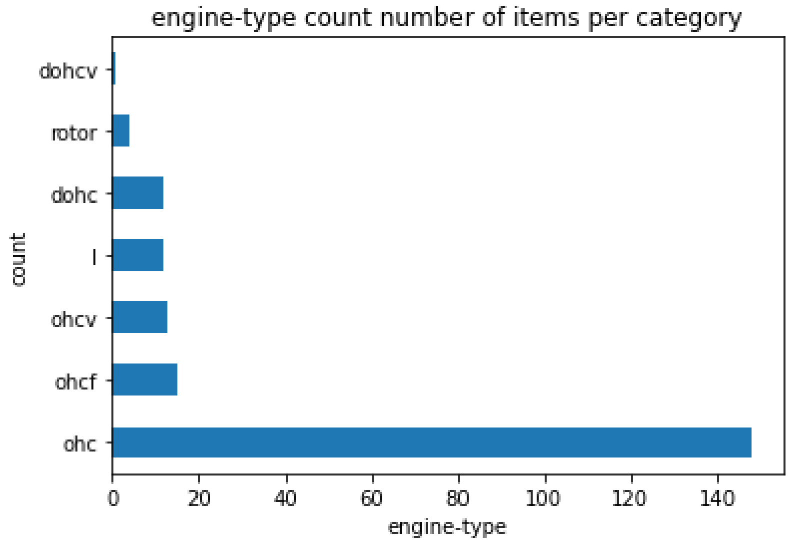
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          205 non-null    int64  
 1   normalized-losses  205 non-null    float64
 2   make               205 non-null    object 
 3   fuel-type          205 non-null    object 
 4   body-style         205 non-null    object 
 5   drive-wheels       205 non-null    object 
 6   engine-location    205 non-null    object 
 7   width              205 non-null    float64
 8   height             205 non-null    float64
 9   engine-type        205 non-null    object 
 10  engine-size        205 non-null    int64  
 11  horsepower         205 non-null    float64
 12  city-mpg           205 non-null    int64  
 13  highway-mpg        205 non-null    int64  
 14  price              205 non-null    int64  
dtypes: float64(4), int64(5), object(6)
memory usage: 24.1+ KB
```

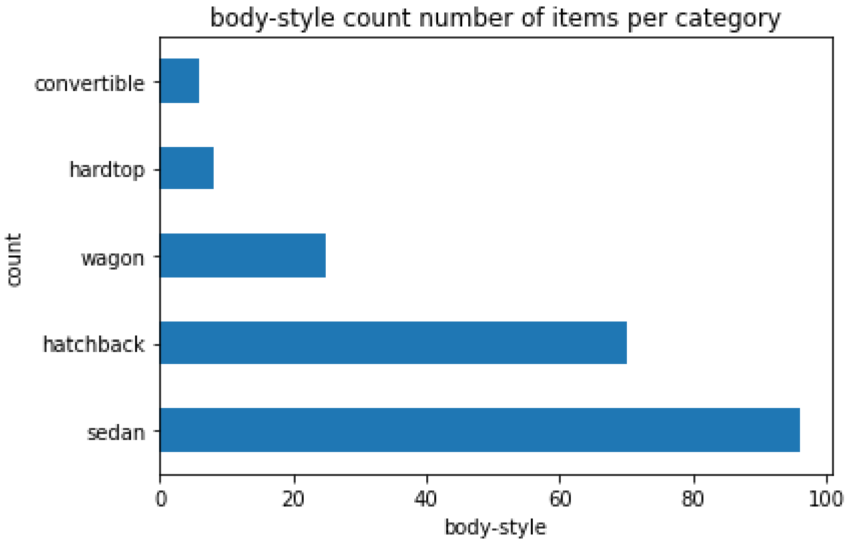
In [86]:

```
cat=["make","fuel-type","engine-type","body-style"]
for i in cat:
    df[i].value_counts().plot(kind="barh") # horizontal
    plt.title("{} count number of items per category".format(i))
    plt.ylabel("count")
    plt.xlabel(i)
    plt.show()
```




fuel-type count number of items per category
diesel
gas
count
0
25
50
75
100
125
150
175
fuel-type

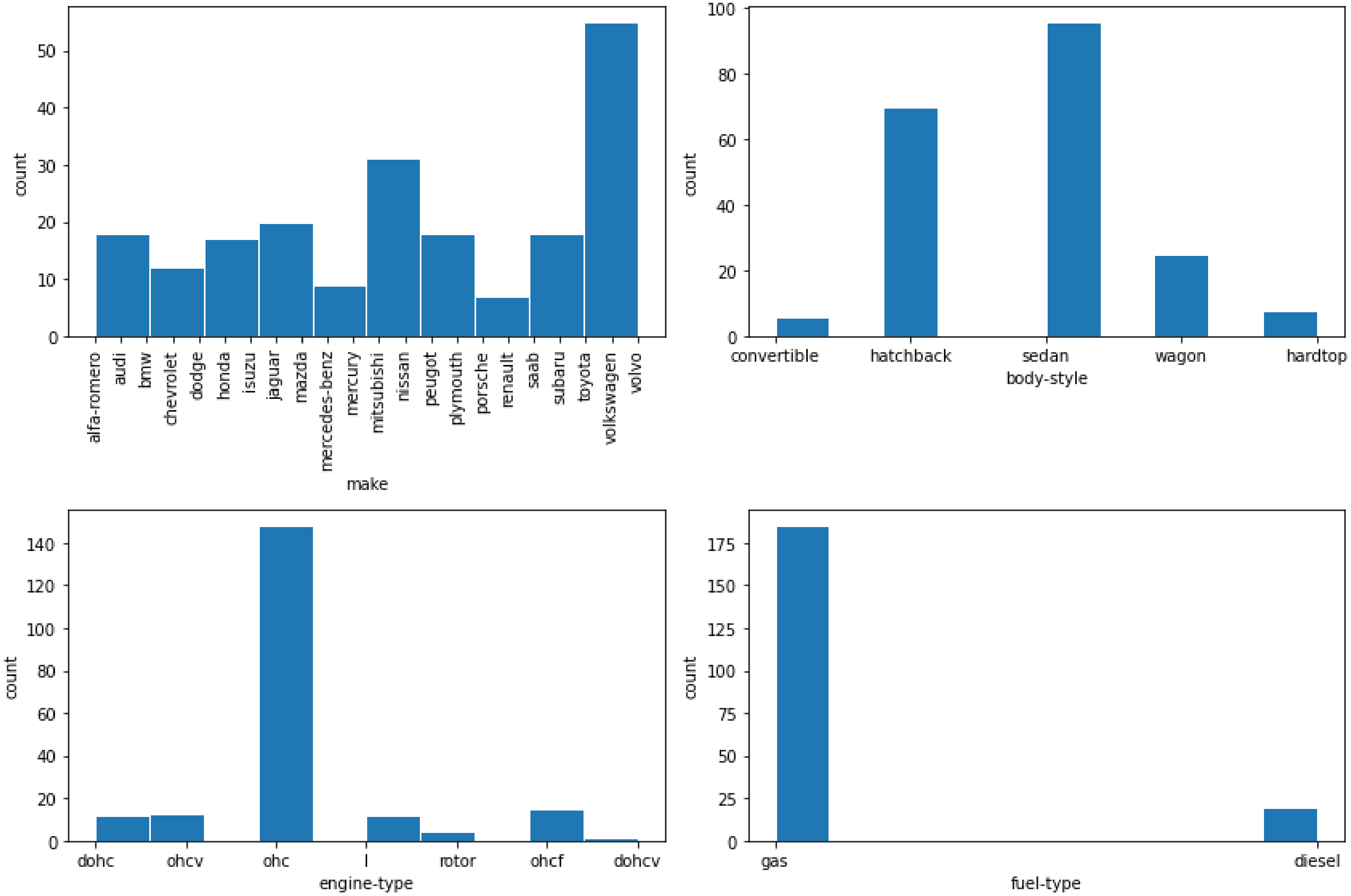

engine-type count number of items per category
dohcv
rotor
dohc
l
ohcv
ohcf
ohc
count
0
20
40
60
80
100
120
140
engine-type



In [87]:

```
# we can also use subplot mechanism for univariate analysis

fig,((ax1,ax2),(ax3,ax4))=plt.subplots(2,2, figsize = (12,8))

# i want 2 rows and 2 columns subplot 1 row contain 2 subplots like that

ax1.hist(df['make'],edgecolor="white",align='mid')
ax1.set_xlabel("make")
ax1.set_ylabel("count")

ax2.hist(df['body-style'],edgecolor="white",align='mid')
ax2.set_xlabel("body-style")
ax2.set_ylabel("count")

ax3.hist(df['engine-type'],edgecolor="white",align='mid')
ax3.set_xlabel("engine-type")
ax3.set_ylabel("count")

ax4.hist(df['fuel-type'],edgecolor="white",align='mid')
ax4.set_xlabel("fuel-type")
ax4.set_ylabel("count")

plt.setp(ax1.xaxis.get_majorticklabels(),rotation=90)
```

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



outlier analysis using boxplot

In [88]:

```
df["make"].unique()
```

Out[88]:

```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'mazda', 'mercedes-benz', 'mercury',
       'mitsubishi', 'nissan', 'peugot', 'plymouth', 'porsche', 'renault',
       'saab', 'subaru', 'toyota', 'volkswagen', 'volvo'], dtype=object)
```

In [89]:

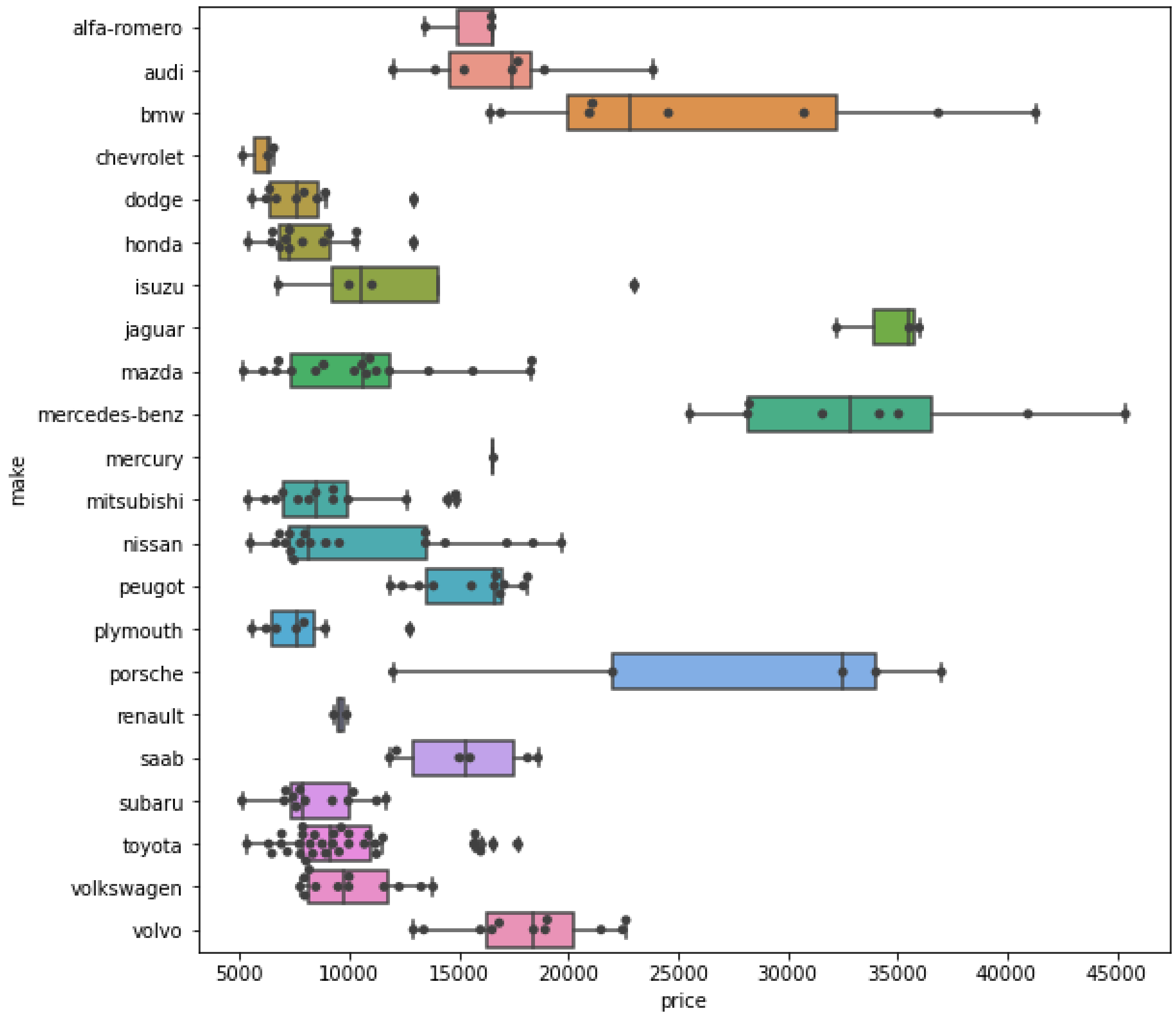
```
labels=df["make"].unique()
print(labels)
plt.figure(figsize=(9,9))
sns.boxplot(x="price",y="make",data=df)
#plt.xticks(df["make"],labels,rotation="vertical")
#showing data points
sns.swarmplot(x="price",y="make",data=df,color=".25")
plt.plot()
```

```
['alfa-romero' 'audi' 'bmw' 'chevrolet' 'dodge' 'honda' 'isuzu' 'jaguar'
 'mazda' 'mercedes-benz' 'mercury' 'mitsubishi' 'nissan' 'peugot'
 'plymouth' 'porsche' 'renault' 'saab' 'subaru' 'toyota' 'volkswagen'
 'volvo']
```

```
C:\Users\Student 12\Anaconda3\lib\site-packages\seaborn\categorical.py:1296: UserWarning: 5.6% of the points cannot b
e placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\Student 12\Anaconda3\lib\site-packages\seaborn\categorical.py:1296: UserWarning: 6.2% of the points cannot b
e placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\Student 12\Anaconda3\lib\site-packages\seaborn\categorical.py:1296: UserWarning: 8.3% of the points cannot b
e placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

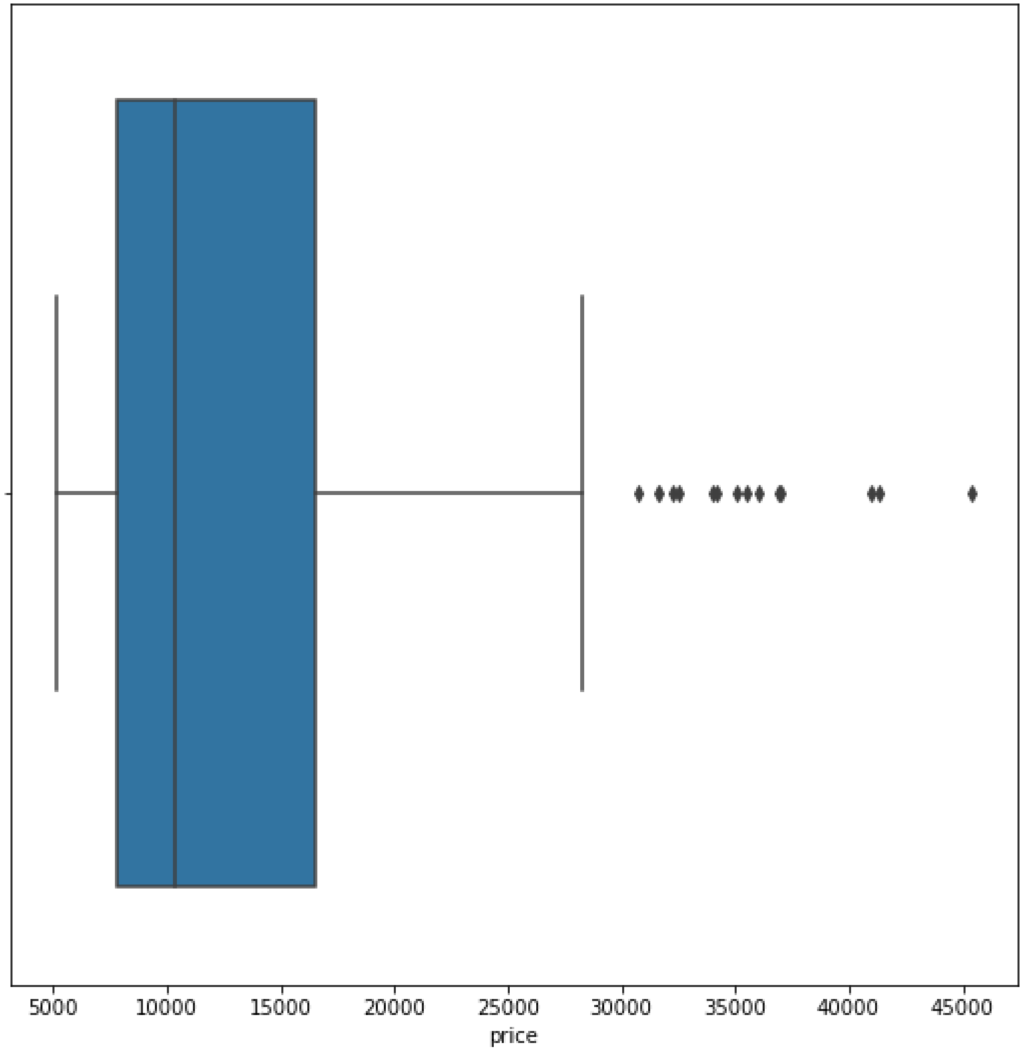
Out[89]:

```
[]
```



In [90]:

```
plt.figure(figsize=(9,9))
sns.boxplot(x="price",data=df)
plt.show()
```



In [91]:

```
outlierdata = df[df["price"]>30000]
```

In [92]:

```
outlierdata.columns
```

Out[92]:

```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'body-style',
```

```
       'drive-wheels', 'engine-location', 'width', 'height', 'engine-type',
       'engine-size', 'horsepower', 'city-mpg', 'highway-mpg', 'price'],
      dtype='object')
```

In [93]:
```
outlierdata["price"].value_counts()
```

Out[93]:
```
31600    1
35550    1
32250    1
34184    1
45400    1
36000    1
32528    1
36880    1
35056    1
34028    1
30760    1
37028    1
41315    1
40960    1
Name: price, dtype: int64
```

if i do skewness reducing process on price data after that i will check again outlier # after that outlier will be reduces 1) univariate 2) bivariate 3) outlier analyze on above data

EDA process on Housing DataSet

In [94]:
```
house=pd.read_csv("house_price.csv")
```

In [95]:
```
house.columns
```

Out[95]:
```
Index(['Unnamed: 0', 'Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea',
       'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'GarageArea', 'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [96]:
```
house.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 26 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Unnamed: 0     1460 non-null   int64  
 1   Id             1460 non-null   int64  
 2   MSSubClass     1460 non-null   int64  
 3   MSZoning       1460 non-null   object 
 4   LotFrontage    1201 non-null   float64
 5   LotArea        1460 non-null   int64  
 6   Street         1460 non-null   object 
 7   Alley          91 non-null     object 
 8   LotShape       1460 non-null   object 
 9   LandContour    1460 non-null   object 
 10  Utilities      1460 non-null   object 
 11  LotConfig      1460 non-null   object 
 12  LandSlope      1460 non-null   object 
 13  Neighborhood   1460 non-null   object 
 14  Condition1     1460 non-null   object 
 15  Condition2     1460 non-null   object 
 16  BldgType       1460 non-null   object 
 17  HouseStyle     1460 non-null   object 
 18  OverallQual    1460 non-null   int64  
 19  OverallCond    1460 non-null   int64  
 20  YearBuilt      1460 non-null   int64  
 21  YearRemodAdd   1460 non-null   int64  
 22  RoofStyle      1460 non-null   object 
 23  GarageArea     1460 non-null   int64  
 24  SaleCondition  1460 non-null   object 
 25  SalePrice      1460 non-null   int64  
dtypes: float64(1), int64(10), object(15)
memory usage: 296.7+ KB
```

In [97]:

```
house # first 5 and last 5 records will be visualized
```

Out[97]:

	Unnamed: 0	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	...	BldgType	HouseStyle	OverallQua
0	0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	...	1Fam	2Story	
1	1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	...	1Fam	1Story	

	Unnamed: 0	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	...	BldgType	HouseStyle	OverallQua
2	2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	...	1Fam	2Story	7
3	3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	...	1Fam	2Story	7
4	4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	...	1Fam	2Story	8
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1455	1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	...	1Fam	2Story	6
1456	1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	...	1Fam	1Story	6
1457	1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	...	1Fam	2Story	7
1458	1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	...	1Fam	1Story	5
1459	1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	...	1Fam	1Story	5

1460 rows × 26 columns

In [98]:

```
# Home work understand each column english meaning and its datatypes
```

Data Cleaning Process

In [99]:

```
# we come to know Unnamed: 0 column is working as record index for ml it is not required becasue
# pandas dataframe also give us indexes for each row by default so we will remove that column
house.drop('Unnamed: 0',axis=1,inplace=True)
#house=house.drop('Unnamed: 0',axis=1)
```

In [100…

```
house.columns
```

Out[100…

```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
```

```
       'RoofStyle', 'GarageArea', 'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [101…

```
house.head()
```

Out[101…

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	BldgType	HouseStyle	OverallQual	Overa
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	1Fam	2Story	7	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	1Fam	1Story	6	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	1Fam	2Story	7	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	1Fam	2Story	7	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	1Fam	2Story	8	

5 rows × 25 columns

In [102…

```
# make Id column as dataframe index itself
house.set_index("Id",inplace=True)
```

In [103…

```
house.columns
```

Out[103…

```
Index(['MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street', 'Alley',
       'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope',
       'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle',
       'GarageArea', 'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [104…

```
house.head()
```

Out[104…

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	...	BldgType	HouseStyle	OverallQua
Id														
1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	...	1Fam	2Story	7

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	...	BldgType	HouseStyle	OverallQua
Id														
2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	...	1Fam	1Story	6
3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	...	1Fam	2Story	7
4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	...	1Fam	2Story	7
5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	...	1Fam	2Story	8

5 rows × 24 columns

Null or missing value treatments

In [105…

```
house.isnull().sum()
```

Out[105…

```
MSSubClass         0
MSZoning           0
LotFrontage      259
LotArea            0
Street             0
Alley           1369
LotShape           0
LandContour        0
Utilities          0
LotConfig          0
LandSlope          0
Neighborhood       0
Condition1         0
Condition2         0
BldgType           0
HouseStyle         0
OverallQual        0
OverallCond        0
YearBuilt          0
YearRemodAdd       0
RoofStyle          0
GarageArea         0
```

```
SaleCondition       0
SalePrice           0
dtype: int64
```

In [106…

```
# in our dataframe LotFrontage,Alley two columns have null values
# so if 90% or above column data are null then we will remove that column
nullper=(house.isnull().sum()/len(house))*100
```

In [107…

```
nullper
```

Out[107…

```
MSSubClass        0.000000
MSZoning          0.000000
LotFrontage      17.739726
LotArea           0.000000
Street            0.000000
Alley            93.767123
LotShape          0.000000
LandContour       0.000000
Utilities         0.000000
LotConfig         0.000000
LandSlope         0.000000
Neighborhood      0.000000
Condition1        0.000000
Condition2        0.000000
BldgType          0.000000
HouseStyle        0.000000
OverallQual       0.000000
OverallCond       0.000000
YearBuilt         0.000000
YearRemodAdd      0.000000
RoofStyle         0.000000
GarageArea        0.000000
SaleCondition     0.000000
SalePrice         0.000000
dtype: float64
```

LotFrontage column have 17. something % missing value which we can replace by mean or median but Alley 93% missing values so we will remove that column becasue it is less importent feature or variable or column

In [108…

```
# remove Alley becasue more than 90% data is null
house.drop(["Alley"],axis=1,inplace=True)
```

In [109…

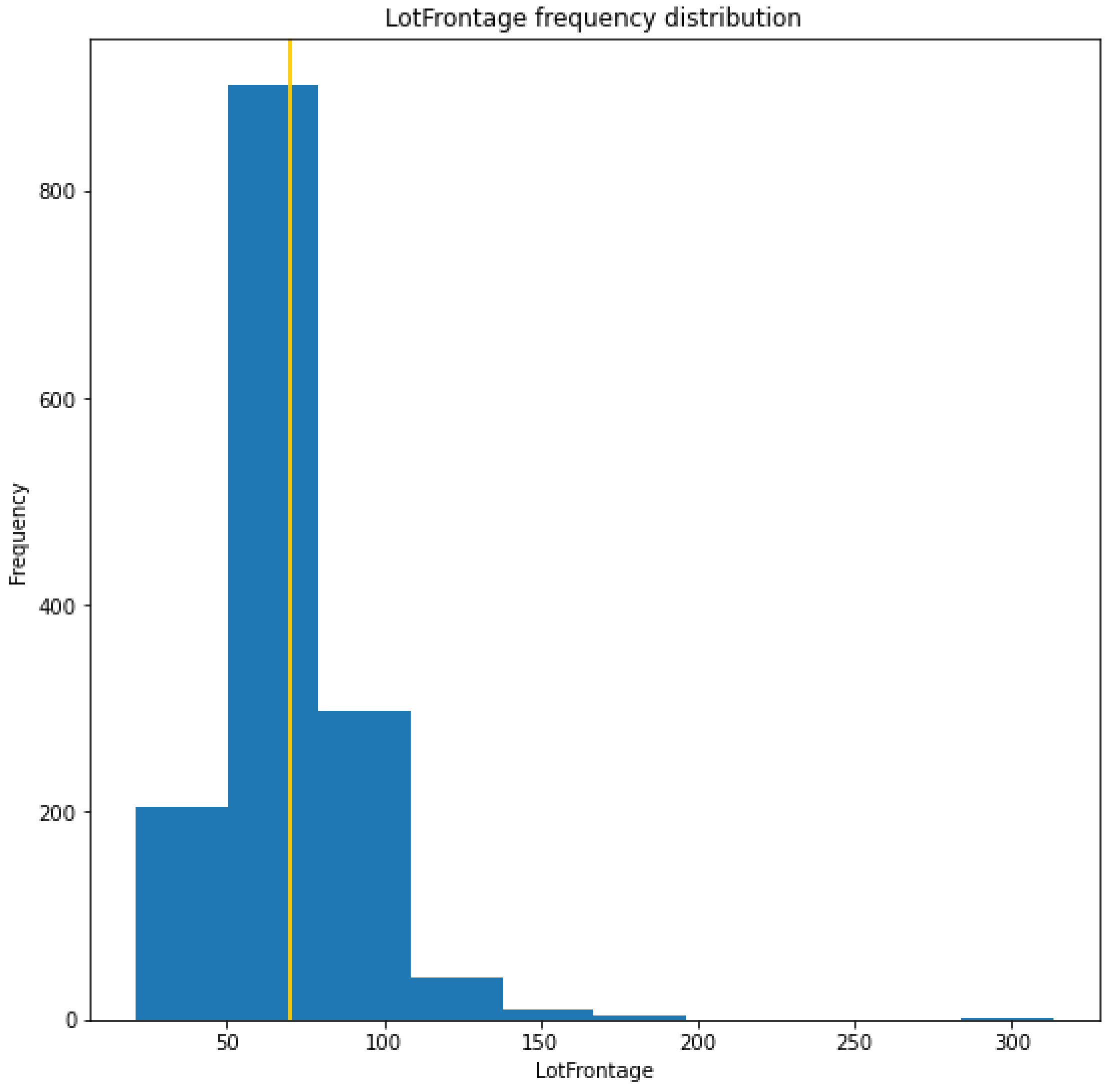
```
house.columns
```

Out[109…

```
Index(['MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope',
       'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle',
       'GarageArea', 'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [159…

```
plt.figure(figsize=(9,9))
plt.hist(house['LotFrontage'],  bins=10)
plt.title("LotFrontage frequency distribution")
plt.axvline(house["LotFrontage"].mean(),color="red")
plt.axvline(house["LotFrontage"].median(),color="yellow")
plt.ylabel("Frequency")
plt.xlabel("LotFrontage")
plt.show()
```



In [111…

```
print("mean for LotFrontage ",house["LotFrontage"].mean())
print("median for LotFrontage ",house["LotFrontage"].median())
```

```
mean for LotFrontage  70.04995836802665
median for LotFrontage  69.0
```

In [112…

```
# missing value treatment for LotFrontage
house["LotFrontage"].fillna(house["LotFrontage"].mean(),inplace=True)
```

In [113…

```
nullper=(house.isnull().sum()/len(house))*100
print(nullper)
```

```
MSSubClass       0.0
MSZoning         0.0
LotFrontage      0.0
LotArea          0.0
Street           0.0
LotShape         0.0
LandContour      0.0
Utilities        0.0
LotConfig        0.0
LandSlope        0.0
Neighborhood     0.0
Condition1       0.0
Condition2       0.0
BldgType         0.0
HouseStyle       0.0
OverallQual      0.0
OverallCond      0.0
YearBuilt        0.0
YearRemodAdd     0.0
RoofStyle        0.0
GarageArea       0.0
SaleCondition    0.0
SalePrice        0.0
dtype: float64
```

In [114…

```
# now the question is for numeric datatypes we need diff analysis and for categorical datatype
# we need diff analysis to make analysis process faster during EDA process we will seperate
# our entier dataframe into two part 1) housenum 2) housecat
# housenum = int,float
# housecat = object,string
```

In [115…

```
house_num=house.select_dtypes(['int64','float64'])
```

In [116…

```
house_num.columns
```

Out[116…

```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'GarageArea', 'SalePrice'],
      dtype='object')
```

In [117…

```
house.columns
```

Out[117…

```
Index(['MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope',
       'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle',
       'GarageArea', 'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [118…

```
house_cat=house.select_dtypes(['object'])
```

In [119…

```
house_cat.columns
```

Out[119…

```
Index(['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'SaleCondition'],
      dtype='object')
```

1) data cleaning 2) seperate numeric and categorical data

In [120…

```
x=[5,10,12,12.5,10.1,11,13,11.2,50]
print("minimum value => ",min(x))
print("maximum value => ",max(x))
```

```
minimum value =>  5
maximum value =>  50
```

In [121…

```
meanx=sum(x)/len(x)
```

In [122…

```
meanx # here mean is influanced by lower and higher extream values
# becasue in data we have outliers
```

Out[122... 14.977777777777778

In [123...
```
x.sort()
```

In [124...
```
x
```

Out[124... [5, 10, 10.1, 11, 11.2, 12, 12.5, 13, 50]

medianx=11.2 q1= 10 + 10.1 /2=10.05 # 25% q2= 11.2 # 50% q3= 12.5+13/2 =12.75 # 75

IQR = 12.75 - 10.5 =2.25 upperwhisker = q3+1.5*iqr = 12.75 + (1.5*2.25)=16.125 if values which is higher than 16.125 then it is oulier we will not accept those values higher extream values / positive extream values

lowerwhisker =q1-1.5*iqr = 10.5-(1.5*2.25)=7.125 below lowerwhisker is lower extream values which is outlier

now whenever we make boxplot using seaborn at that time boxplot will perform all above task and show us the outlier after that

In [125...
```
'''
IQR we will divide our data into quantile
q1 = 25%
q2 = 50%
q3 = 75%

IQR=q3-q1 range which provide me collection data which is free from outlier

iqr will completly ignore all outliers
being a data analyst i need to keep optimal acceptance to my data
for that we have upperwhisker and lowewhisker and to understan outlier we have box plot

'''
```

Out[125... '\nIQR we will divide our data into quantile\nq1 = 25% \nq2 = 50%\nq3 = 75% \n\nIQR=q3-q1 range which provide me collection data which is free from outlier \n\niqr will completly ignore all outliers \nbeing a data analyst i need to keep optimal acceptance to my data \nfor that we have upperwhisker and lowewhisker and to understan outlier we have box plot \n\n'

In [126…

```
'''
1) univariate
2) bivariate
3) distribution normal distributed or skewd data
4) outlier treatments
5) if data is skewd we will try make normal using minmax scaller or standered scaller
'''
```

Out[126…

```
'\n1) univariate \n2) bivariate \n3) distribution normal distributed or skewd data \n4) outlier treatments \n5) if da
ta is skewd we will try make normal using minmax scaller or standered scaller \n'
```

In [127…

```
house_num
```

Out[127…

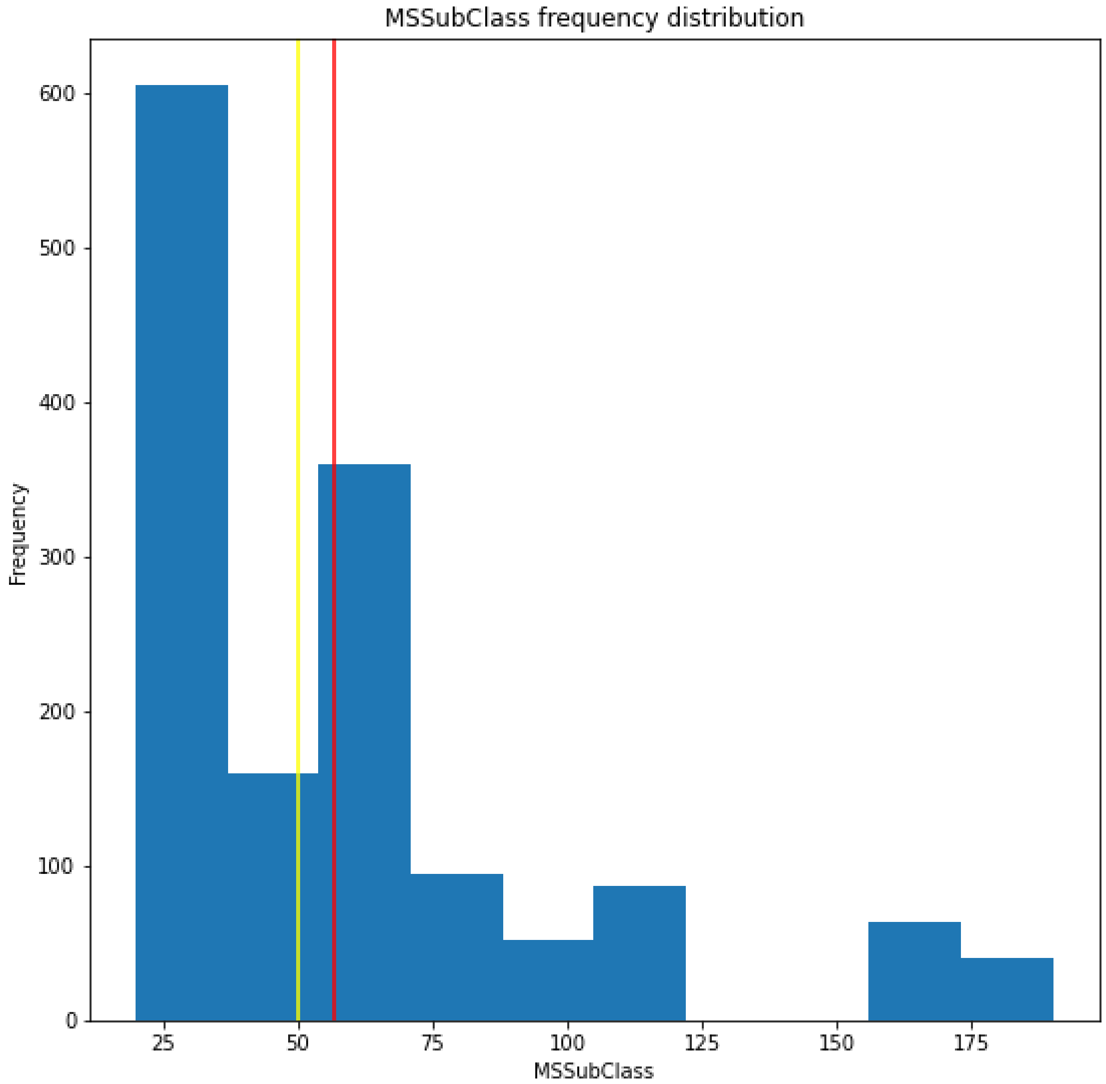
	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	GarageArea	SalePrice
Id									
1	60	65.0	8450	7	5	2003	2003	548	208500
2	20	80.0	9600	6	8	1976	1976	460	181500
3	60	68.0	11250	7	5	2001	2002	608	223500
4	70	60.0	9550	7	5	1915	1970	642	140000
5	60	84.0	14260	8	5	2000	2000	836	250000
...	...	...	...	...	...	...	...	...	...
1456	60	62.0	7917	6	5	1999	2000	460	175000
1457	20	85.0	13175	6	6	1978	1988	500	210000
1458	70	66.0	9042	7	9	1941	2006	252	266500
1459	20	68.0	9717	5	6	1950	1996	240	142125
1460	20	75.0	9937	5	6	1965	1965	276	147500

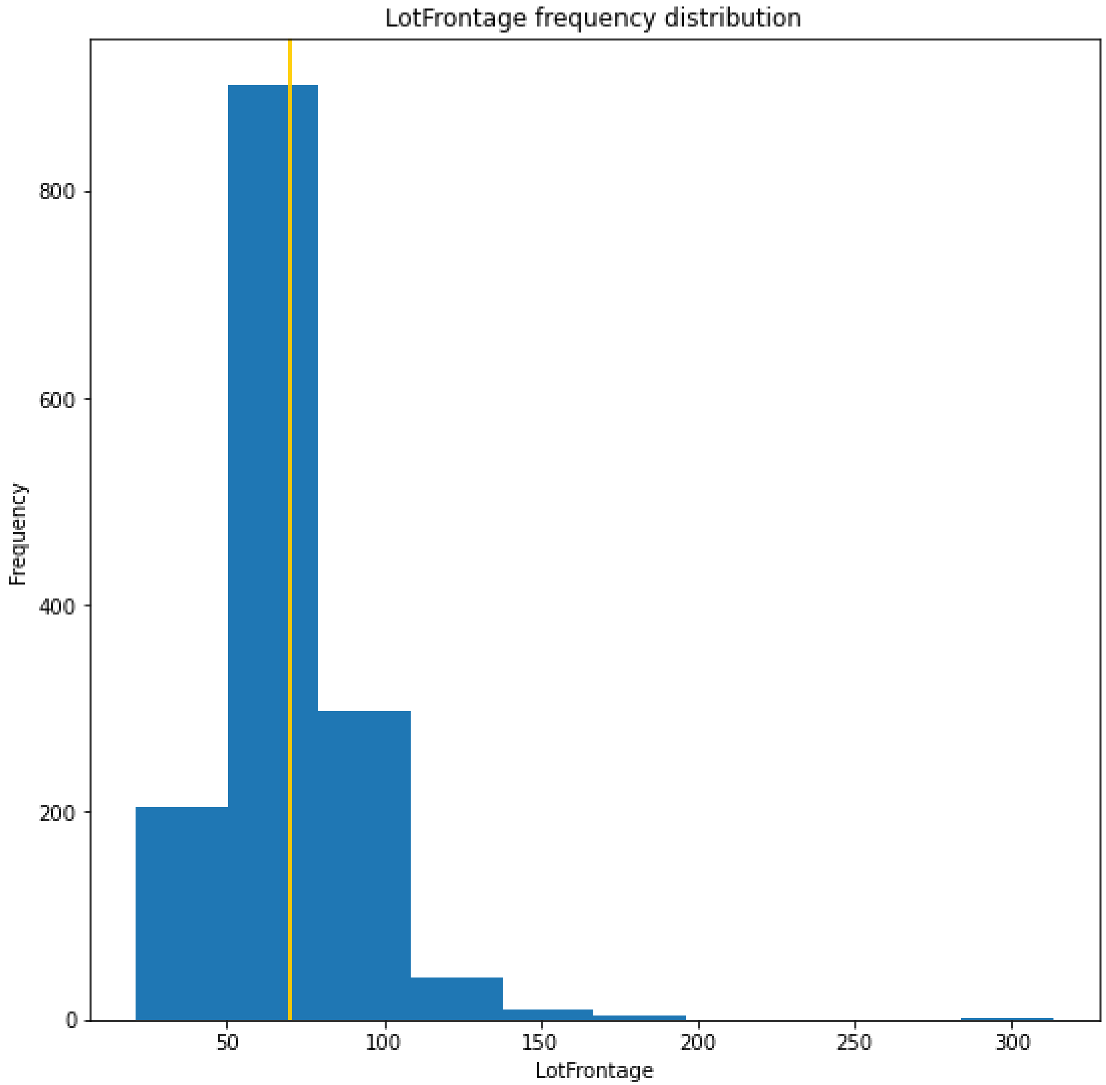
1460 rows × 9 columns

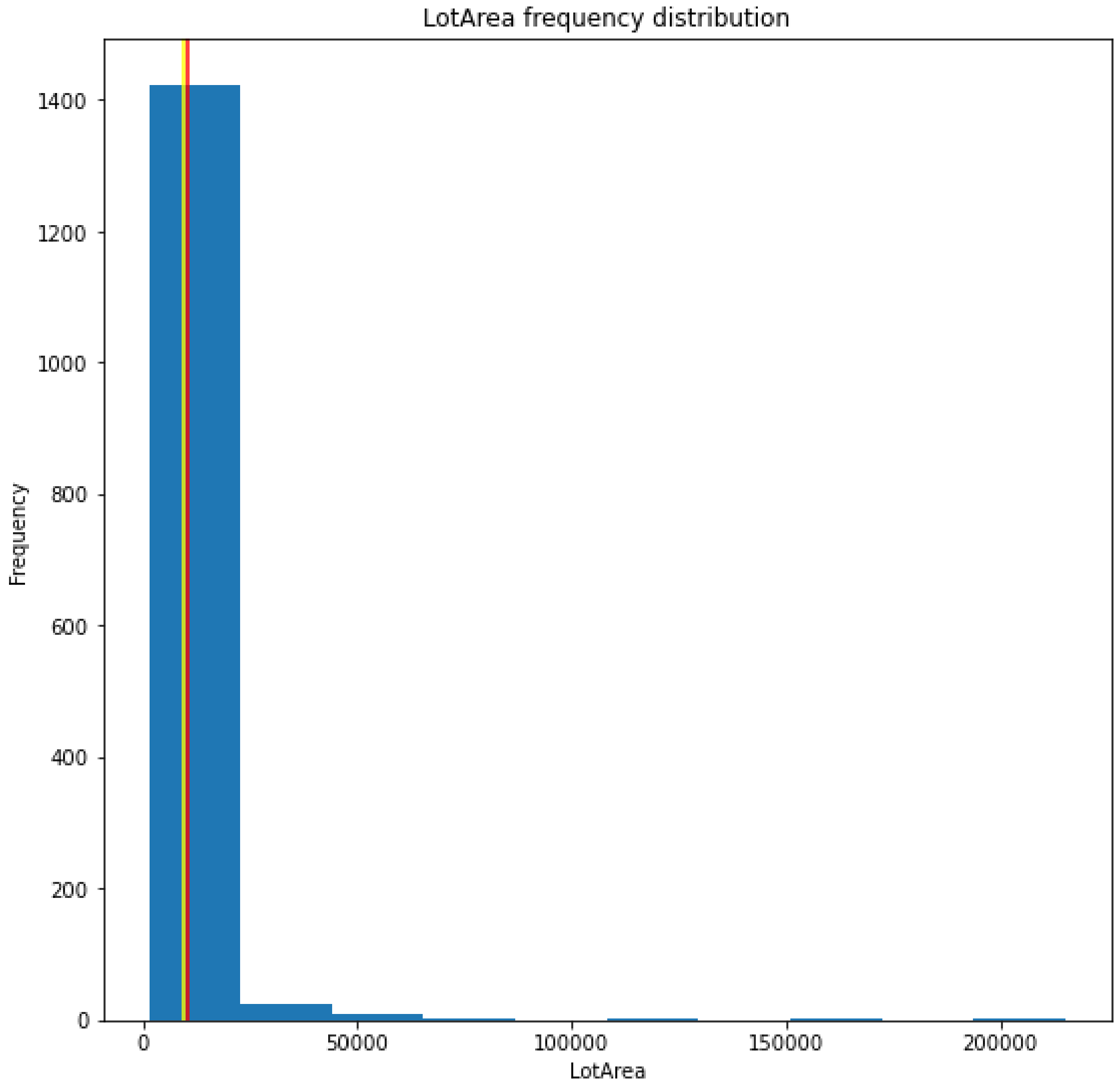
In [128…

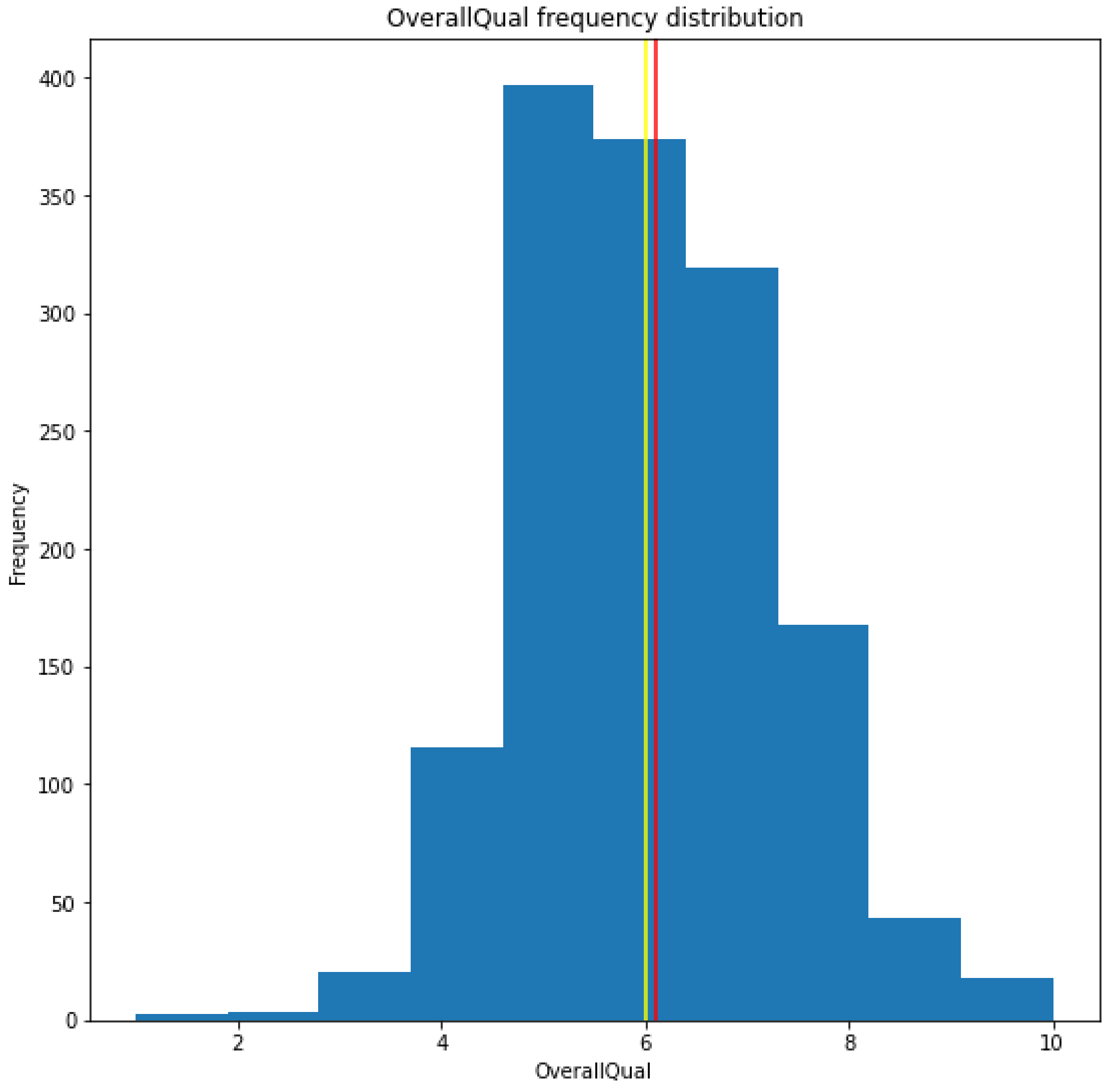
```
# univariate normal distribution data analysis
```

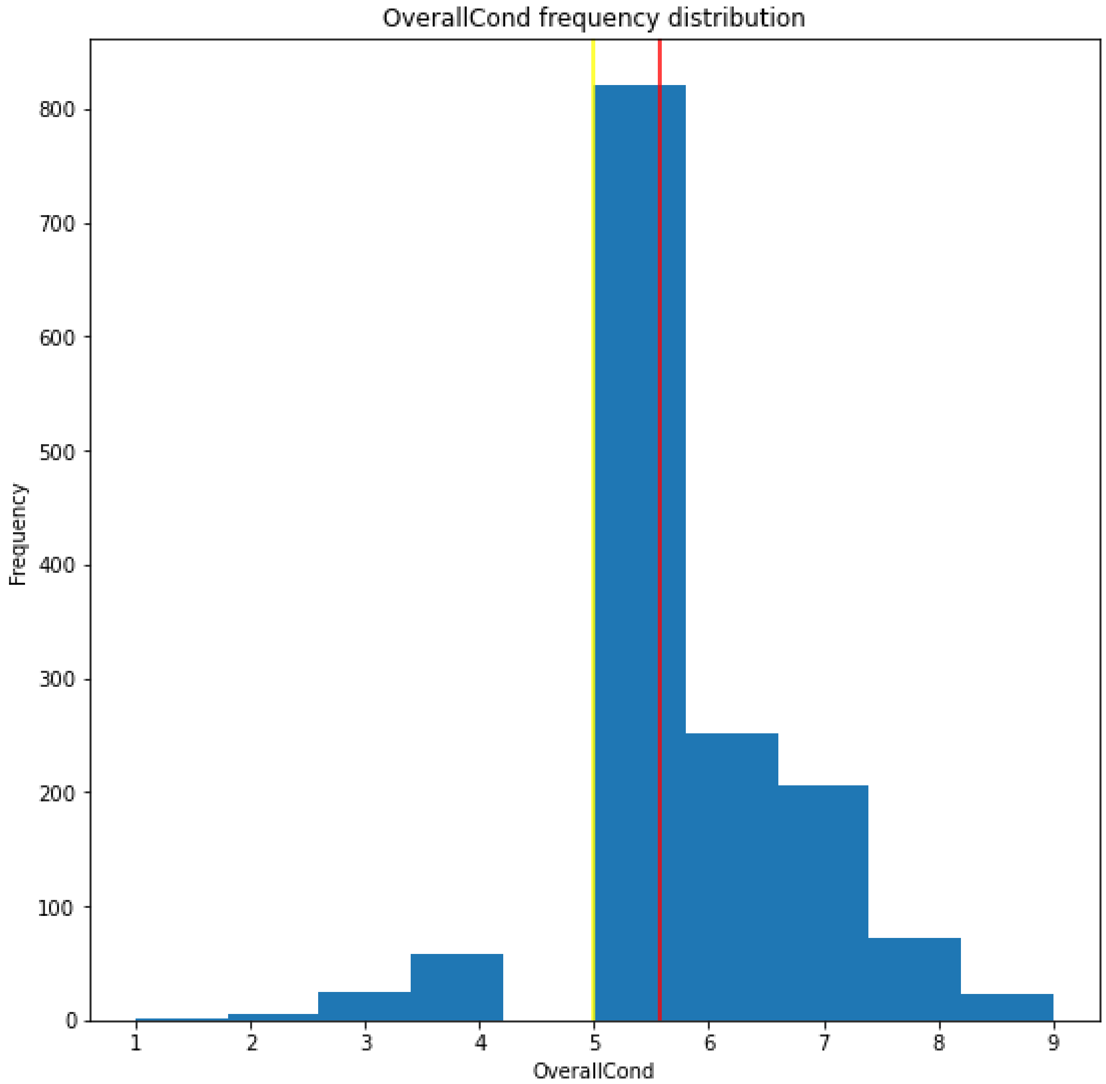
```
for i in house_num:
    plt.figure(figsize=(9,9))
    plt.hist(house[i],bins=10)
    plt.title("{} frequency distribution".format(i))
    plt.axvline(house[i].mean(),color="red")
    plt.axvline(house[i].median(),color="yellow")
    plt.ylabel("Frequency")
    plt.xlabel(i)
    plt.show()
```

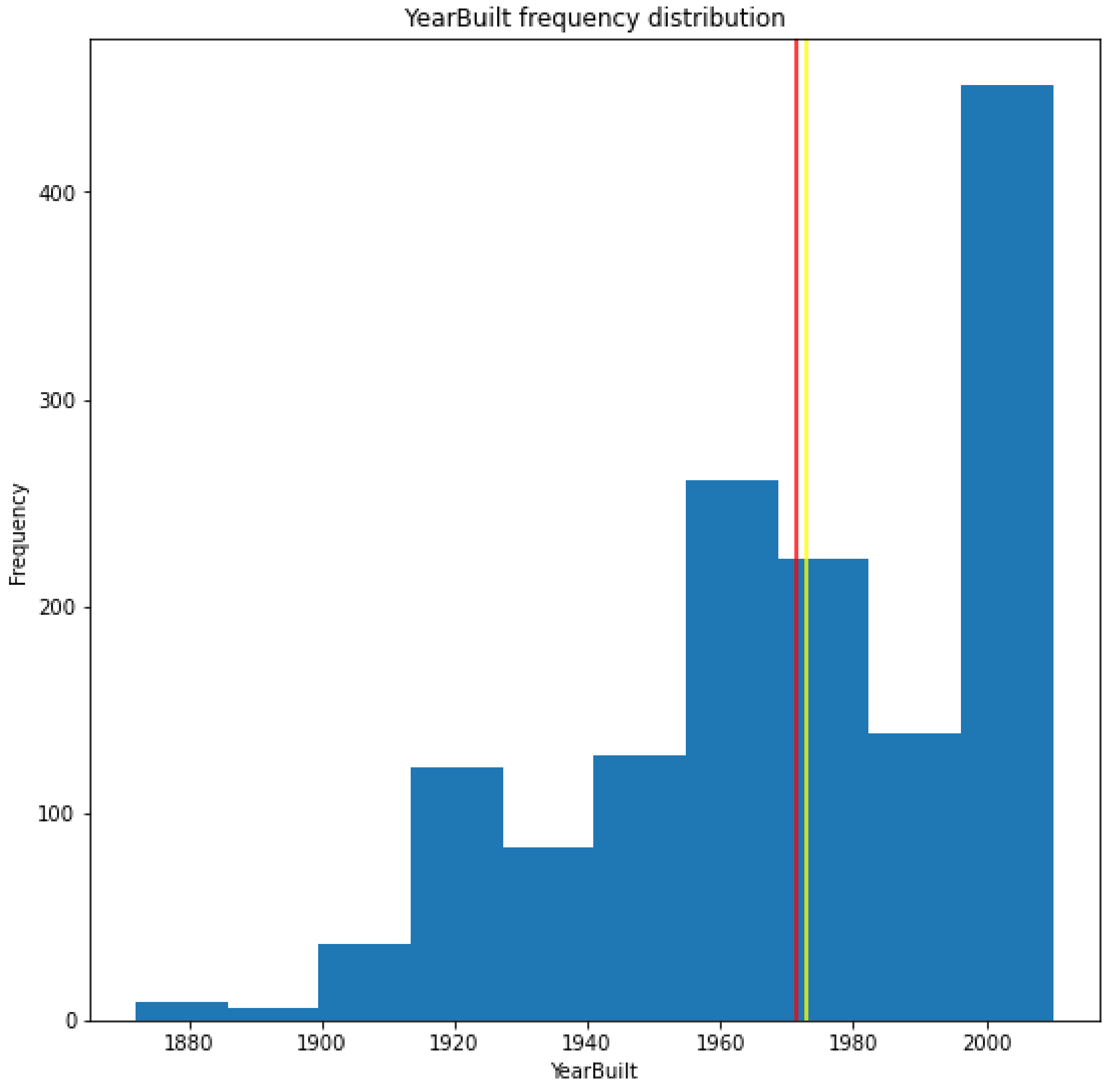

MSSubClass frequency distribution
Frequency
MSSubClass
0
100
200
300
400
500
600
25
50
75
100
125
150
175

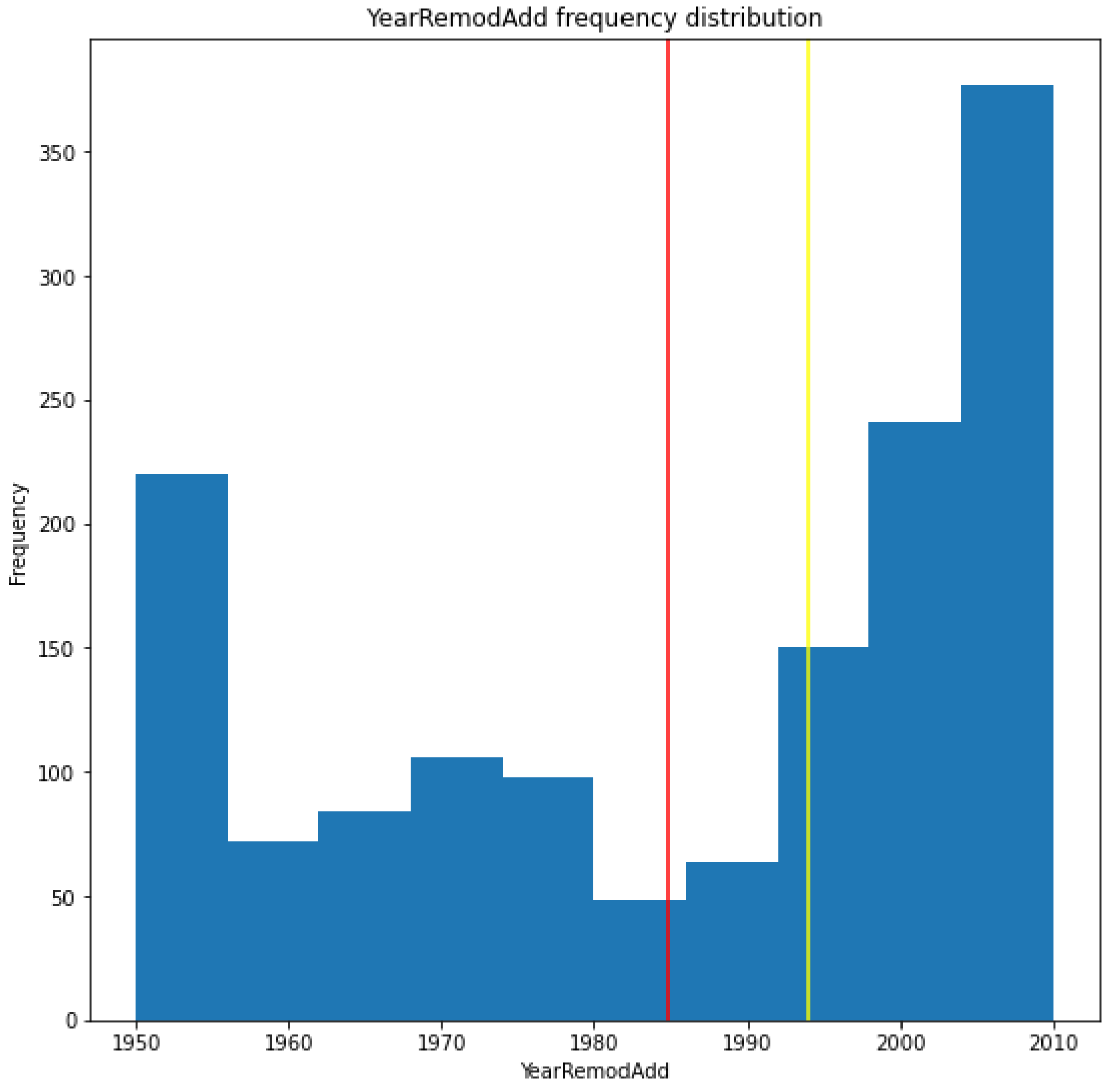

LotFrontage frequency distribution
Frequency
0
200
400
600
800
50
100
150
200
250
300
LotFrontage

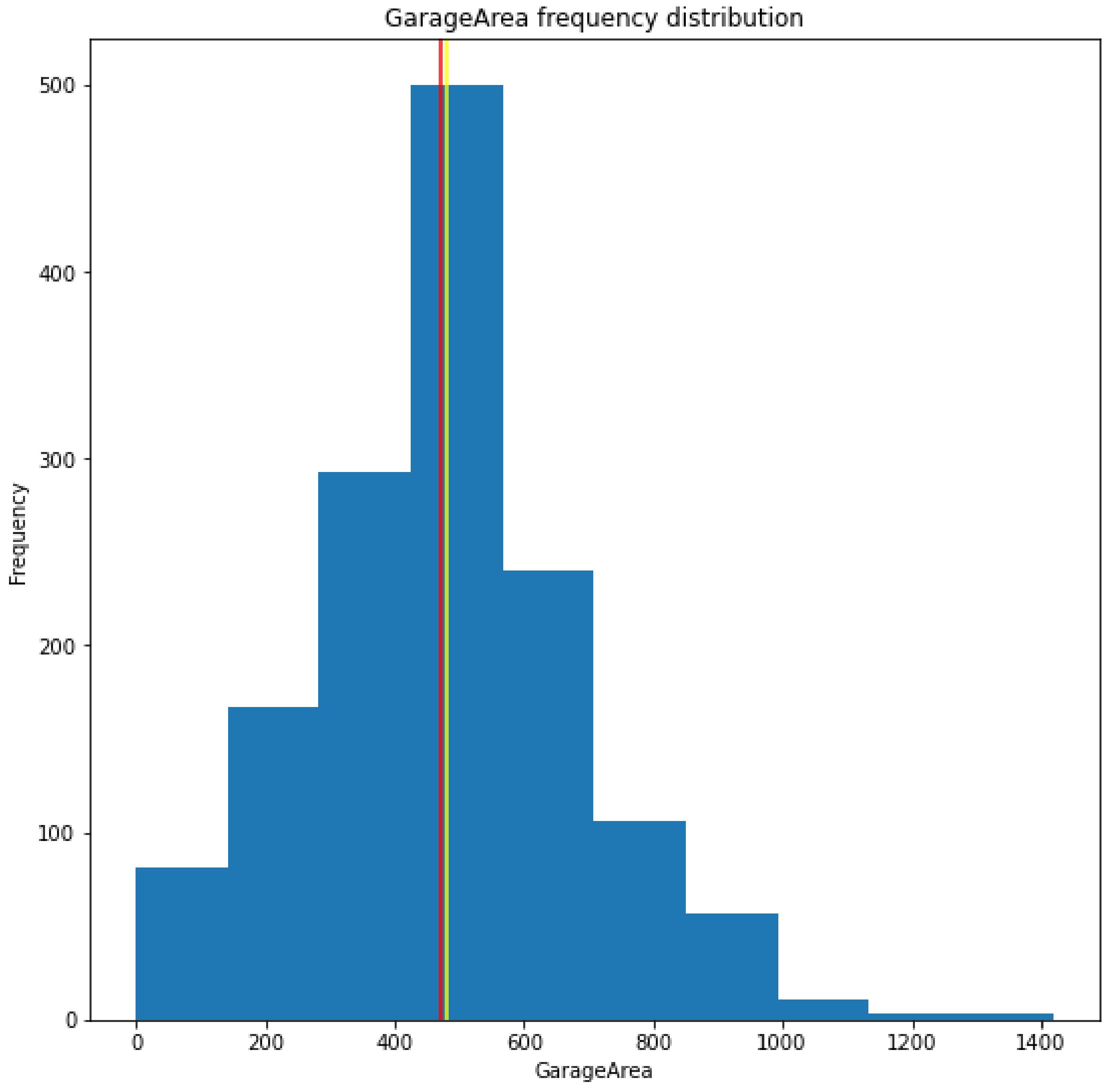

LotArea frequency distribution
Frequency
LotArea
0
200
400
600
800
1000
1200
1400
0
50000
100000
150000
200000

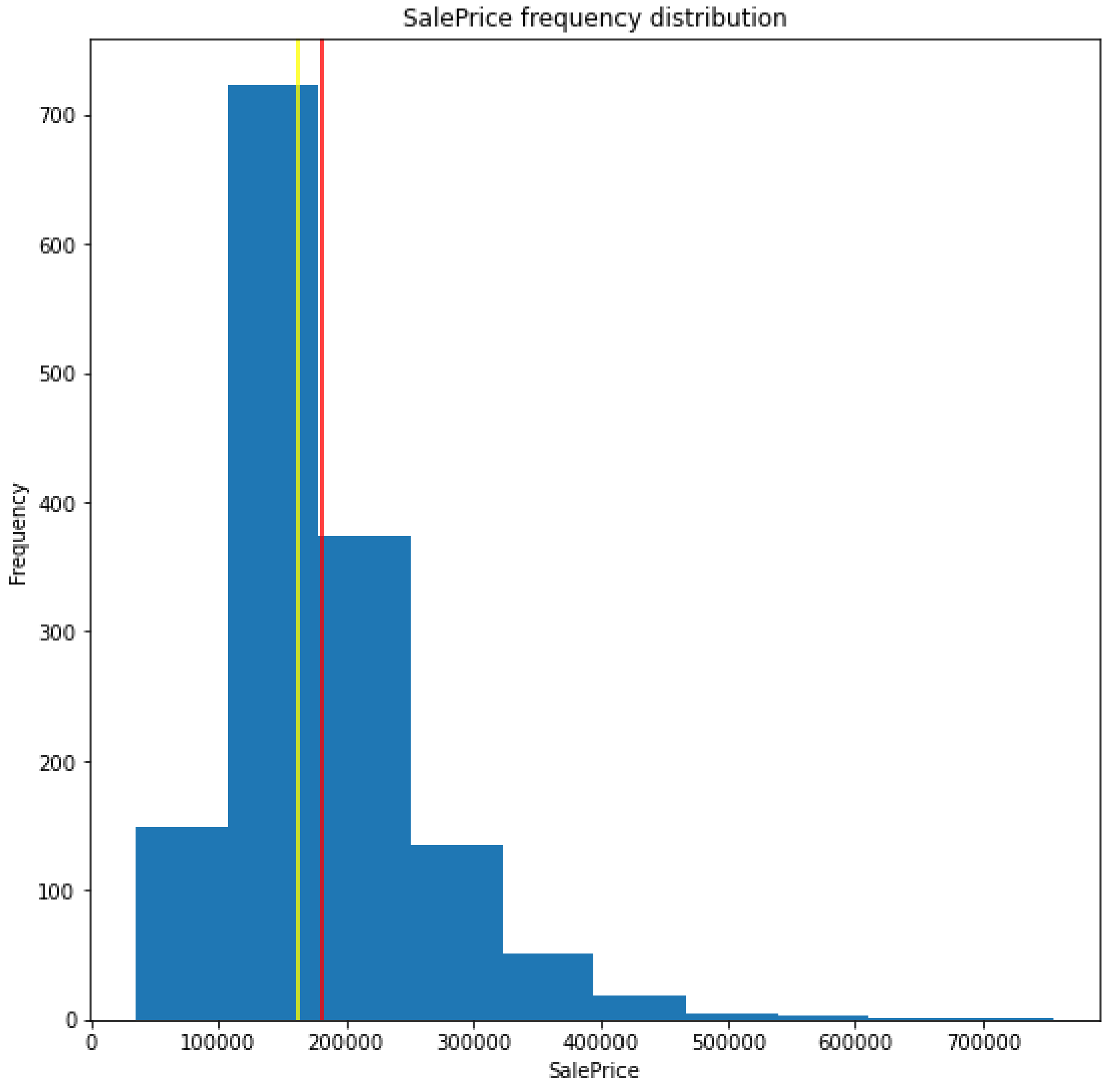

OverallQual frequency distribution
400
350
300
250
200
150
100
50
0
Frequency
2
4
6
8
10
OverallQual


OverallCond frequency distribution
800
700
600
500
400
300
200
100
0
Frequency
1
2
3
4
5
6
7
8
9
OverallCond


YearBuilt frequency distribution
Frequency
0
100
200
300
400
1880
1900
1920
1940
1960
1980
2000
YearBuilt


YearRemodAdd frequency distribution
Frequency
0
50
100
150
200
250
300
350
1950
1960
1970
1980
1990
2000
2010
YearRemodAdd


GarageArea frequency distribution
Frequency
GarageArea
0
100
200
300
400
500
0
200
400
600
800
1000
1200
1400



spain = 200 family mean,median income 1300$meanandmedian100family4500$ 200 + 100 merge and then i calculated average mean 3700 which so far from reality

Outlier Analysis on numeric values

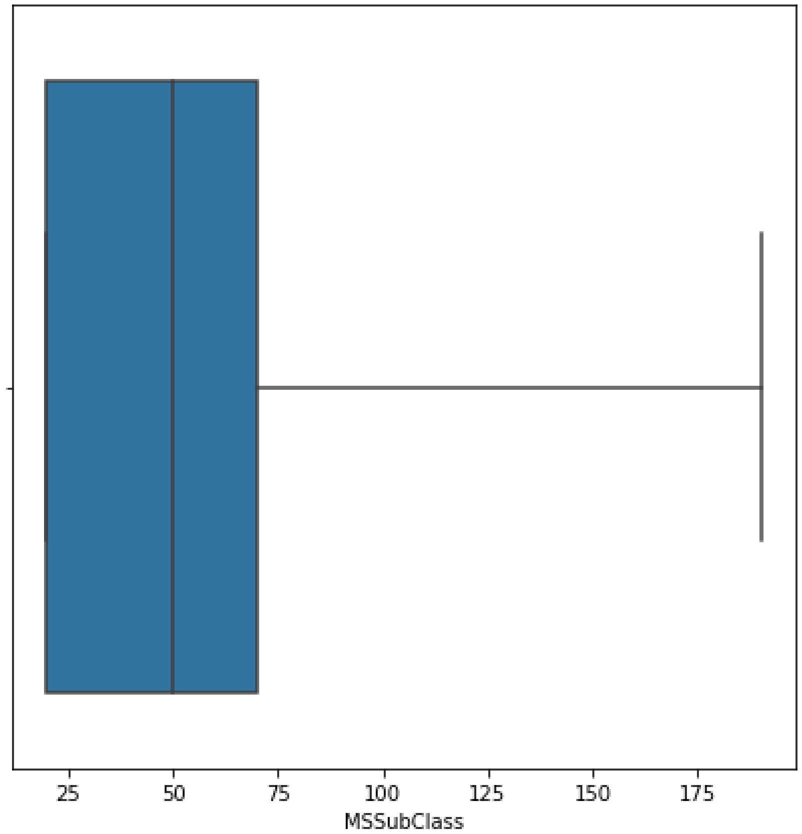
In [129…

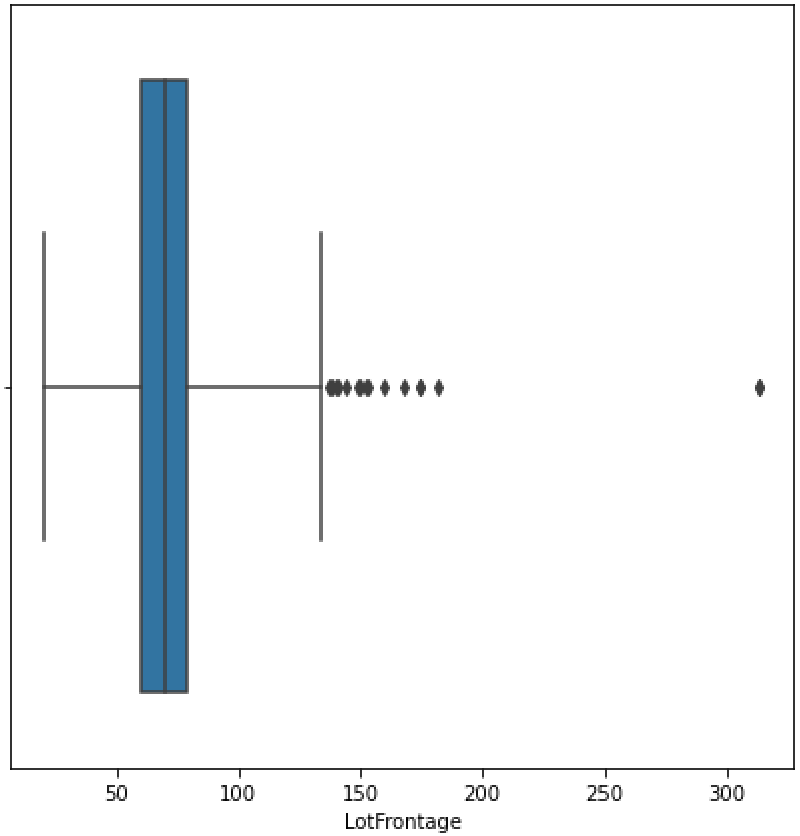
```
house_num.info()
```

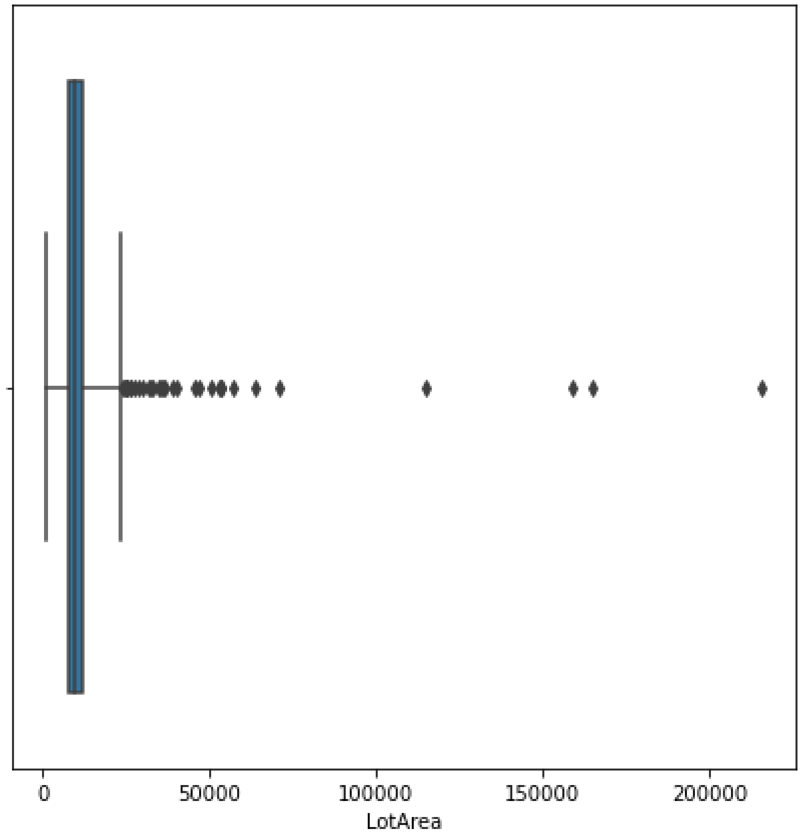
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 1 to 1460
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   MSSubClass    1460 non-null   int64  
 1   LotFrontage   1460 non-null   float64
 2   LotArea       1460 non-null   int64  
 3   OverallQual   1460 non-null   int64  
 4   OverallCond   1460 non-null   int64  
 5   YearBuilt     1460 non-null   int64  
 6   YearRemodAdd  1460 non-null   int64  
 7   GarageArea    1460 non-null   int64  
 8   SalePrice     1460 non-null   int64  
dtypes: float64(1), int64(8)
memory usage: 114.1 KB
```

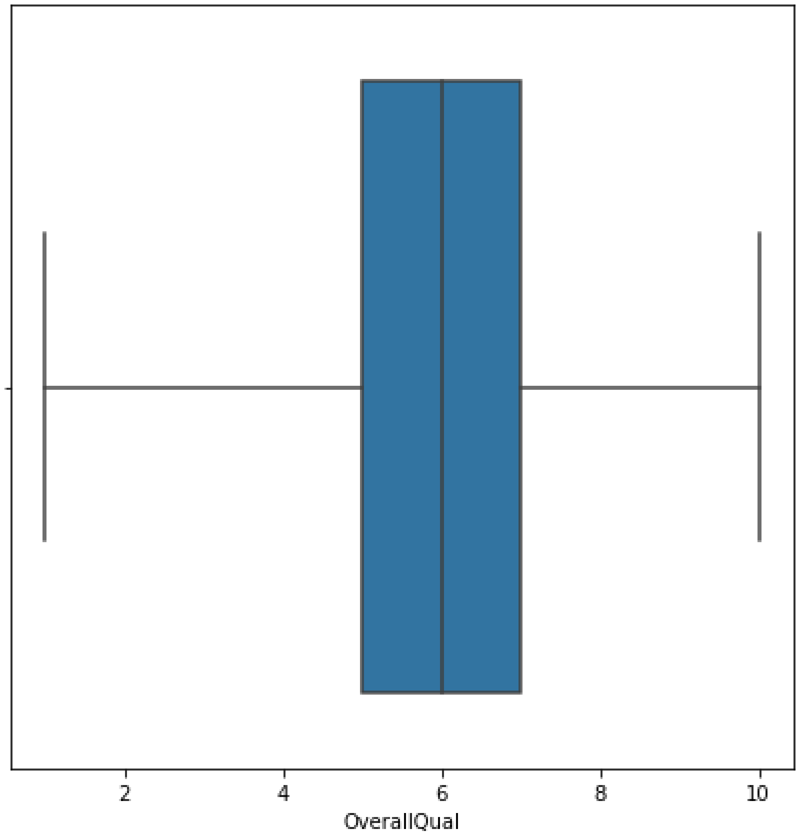
In [130…

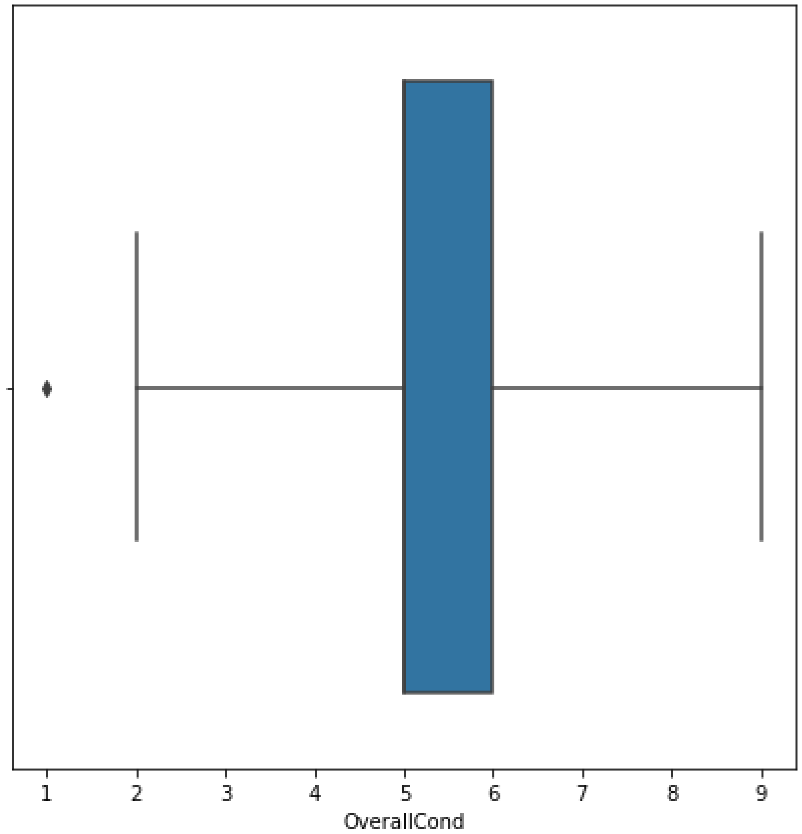
```
for i in house_num:
    plt.figure(figsize=(7,7))
    sns.boxplot(data=house_num,x=i,whis=3)
    # upper whisker = q3+1.5*IQR
    # lower whisker = q1 - 1.5*IQR
    # boxplot will calculate upper whisker and lower whisker by it's own and the nit will plot the box
    plt.show()
```

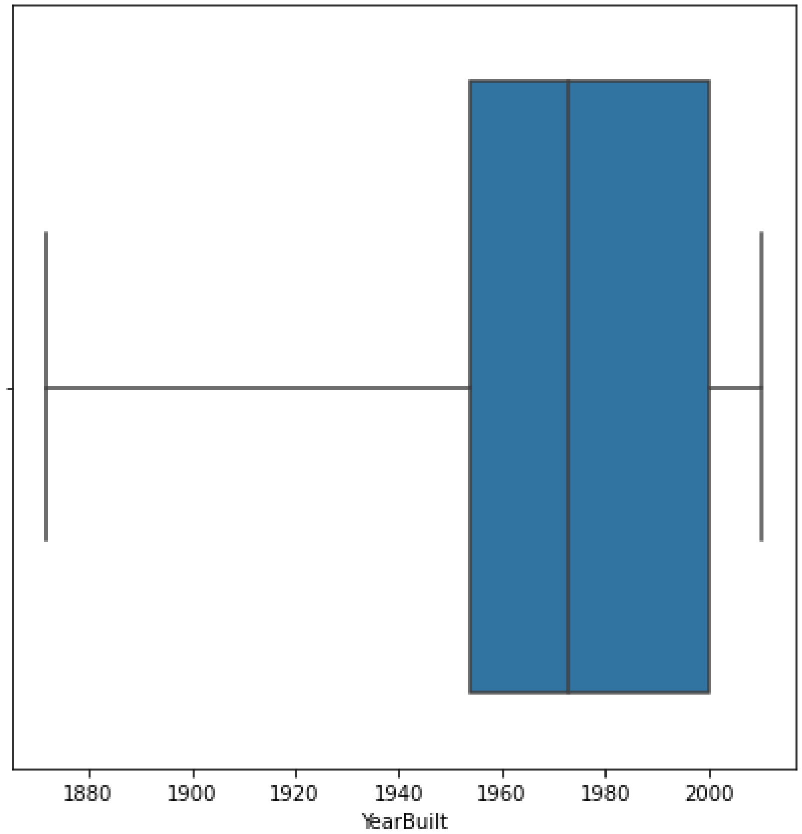

25
50
75
100
125
150
175
MSSubClass

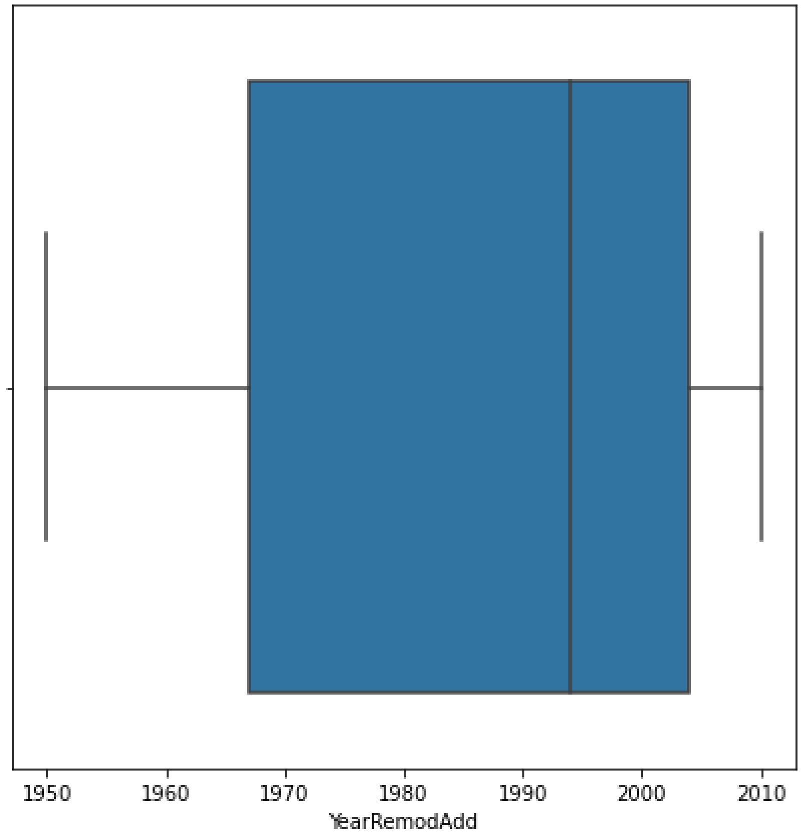

50
100
150
200
250
300
LotFrontage

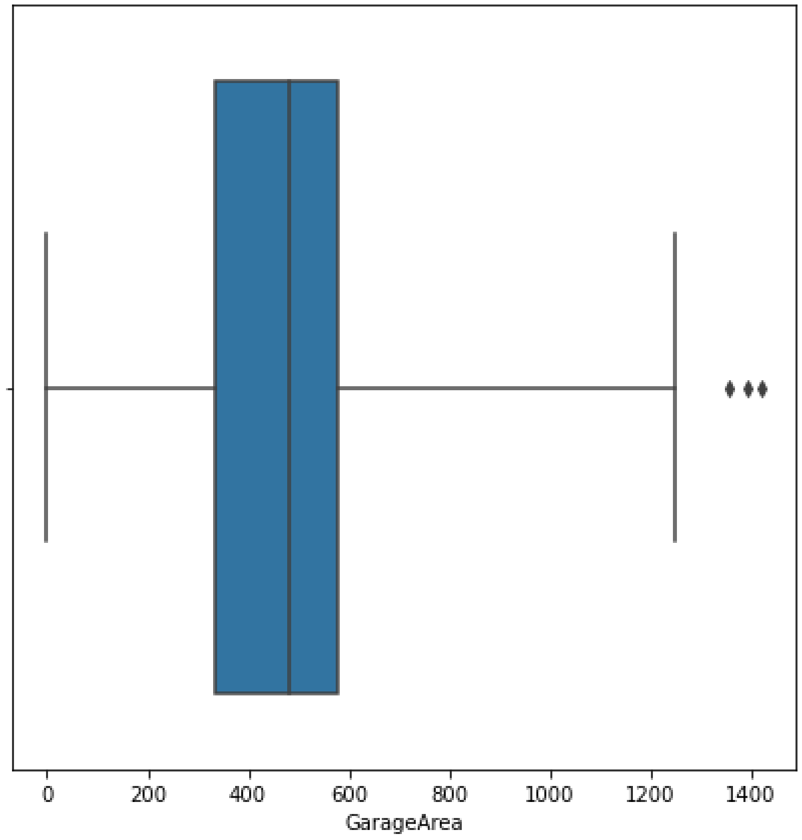

0
50000
100000
150000
200000
LotArea

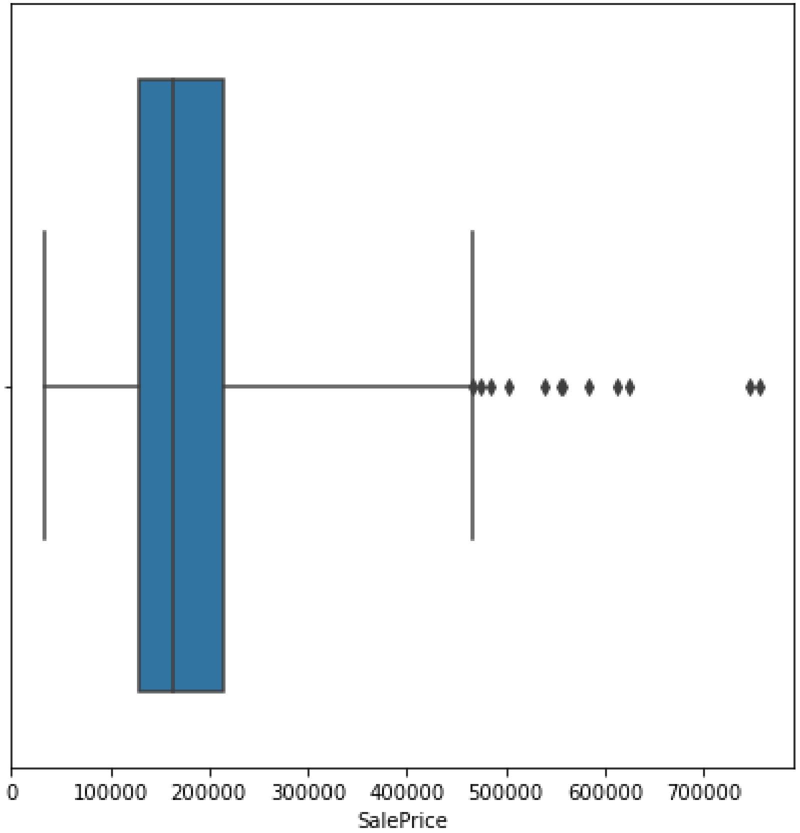

2
4
6
8
10
OverallQual


1
2
3
4
5
6
7
8
9
OverallCond


1880
1900
1920
1940
1960
1980
2000
YearBuilt


1950
1960
1970
1980
1990
2000
2010
YearRemodAdd


0
200
400
600
800
1000
1200
1400
GarageArea



In [131…

```
# if you found data is skewed then 99% there is outlier presence
```

Outlier Treatments

In [132…

```
# SalePrice Column Outlier Treatment
q1=np.quantile(house["SalePrice"],0.25)
q3=np.quantile(house["SalePrice"],0.75)
iqr=q3-q1
```

In [133…

```
print("Quantile1 for sale price is => ",q1)
print("Quantile3 for sale price is => ",q3)
print("IQR for SalePrice column is => ",iqr)
```

```
Quantile1 for sale price is =>  129975.0
Quantile3 for sale price is =>  214000.0
IQR for SalePrice column is =>  84025.0
```

In [134…

```
#as we know we have higher extream values so no need to calculate lower whisker will only go for upper whisker
up_whs=q3+3*iqr
print("upper whisker with 3 penalty is => ",up_whs)
```

```
upper whisker with 3 penalty is =>  466075.0
```

In [135…

```
house_num.shape
```

Out[135… (1460, 9)

In [136…

```
# accept all those records which come below given whisker values
house_num=house_num[house_num["SalePrice"]<up_whs]
```

In [137…

```
house_num.shape
```

Out[137… (1448, 9)

In [138…

```
house_num
```

Out[138…

Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	GarageArea	SalePrice
1	60	65.0	8450	7	5	2003	2003	548	208500
2	20	80.0	9600	6	8	1976	1976	460	181500
3	60	68.0	11250	7	5	2001	2002	608	223500
4	70	60.0	9550	7	5	1915	1970	642	140000

Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	GarageArea	SalePrice
5	60	84.0	14260	8	5	2000	2000	836	250000
...	...	...	...	...	...	...	...	...	...
1456	60	62.0	7917	6	5	1999	2000	460	175000
1457	20	85.0	13175	6	6	1978	1988	500	210000
1458	70	66.0	9042	7	9	1941	2006	252	266500
1459	20	68.0	9717	5	6	1950	1996	240	142125
1460	20	75.0	9937	5	6	1965	1965	276	147500

1448 rows × 9 columns

Skewness Treatment make data normalised and get rid of outlier

as we saw our histplot for all numeric columns we found maximum columns have skewed data that means here we need to normalised our data rule : if the skew value is above less than -1 and greater than 1 then it consider as highly skewd data skew < -1 means negativly skewd skew > 1 means positivly skewd

when our mean is greater than median positive skew when our mean is less than median negative skew if our data is more deviated from given median than it is consider as skewed data to check deviation we can use standered deviation which also known as z score

In [139…

```
from scipy.stats import skew
```

In [140…

```
# Skew = 3 * (Mean – Median) / Standard Deviation.
for i in house_num:
    print(i,skew(house_num[i]))

#data skewness > -1 and < 1 that means data is normally distributed
```

```
MSSubClass 1.4016374700451684
LotFrontage 2.4241983272815286
```

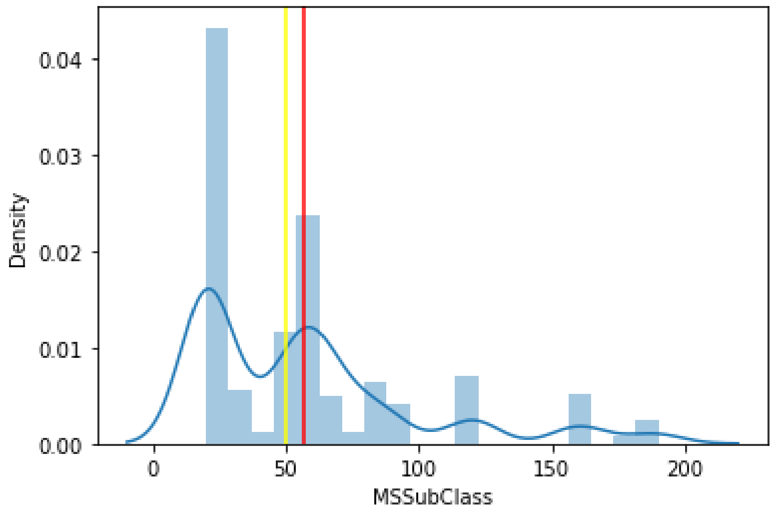
```
LotArea 12.476597041303394
OverallQual 0.15874882949146252
OverallCond 0.6755243853201703
YearBuilt -0.6013498391819941
YearRemodAdd -0.4920110147731343
GarageArea 0.16792353190885764
SalePrice 1.1441959932511412
```

In [141…

```
for col in house_num:
    print(col,skew(house_num[col]))
    plt.figure()
    sns.distplot(house_num[col])
    plt.axvline(house_num[col].mean(),color="red")
    plt.axvline(house_num[col].median(),color="yellow")
    plt.show()
```

```
MSSubClass 1.4016374700451684
```

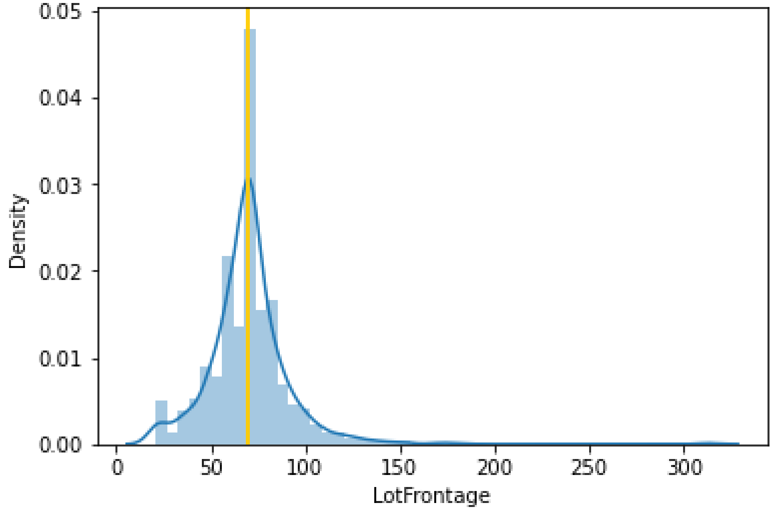
```
C:\Users\Student 12\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a depreca
ted function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
LotFrontage 2.4241983272815286
```

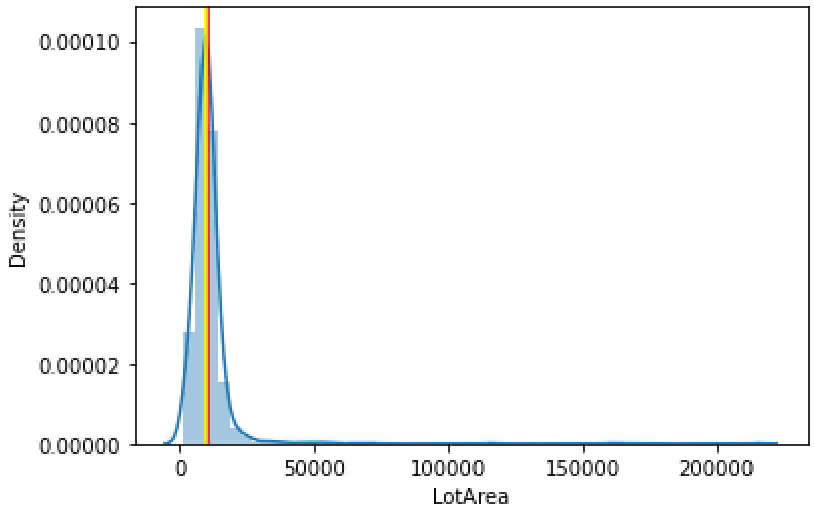
```
C:\Users\Student 12\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a depreca
ted function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level
```

```
function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
LotArea 12.476597041303394
```

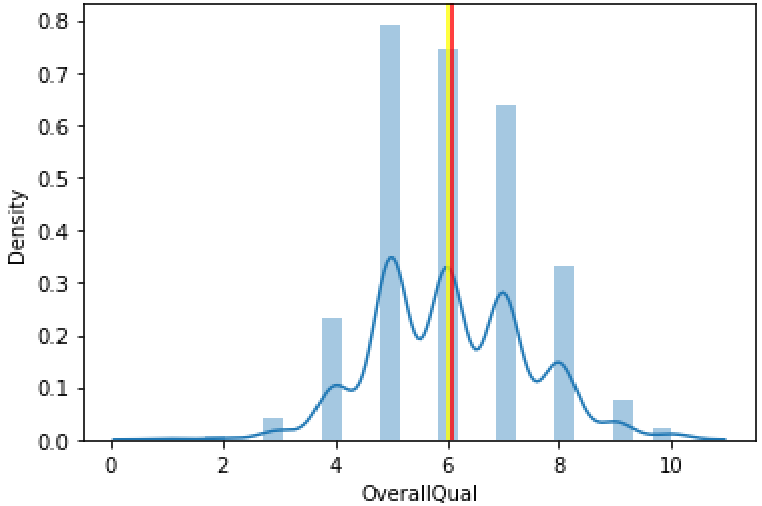
```
C:\Users\Student 12\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a depreca
ted function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
OverallQual 0.15874882949146252
```

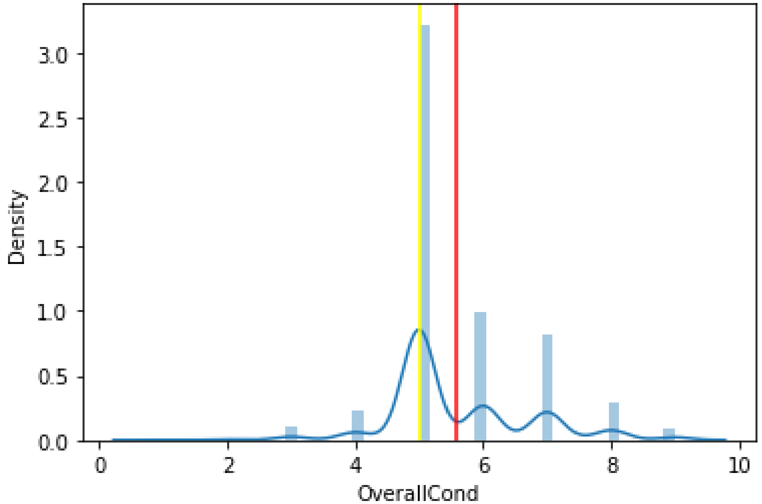
```
C:\Users\Student 12\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a depreca
```

```
ted function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



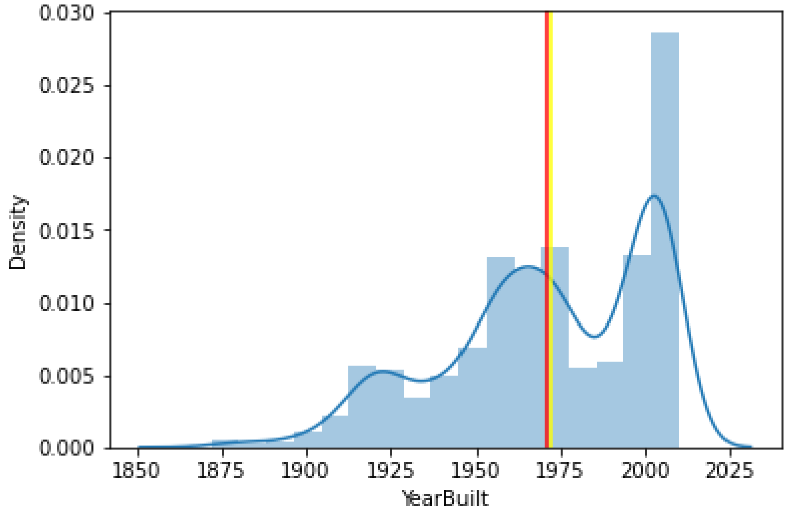

```
OverallCond 0.6755243853201703
```

```
C:\Users\Student 12\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a depreca
ted function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



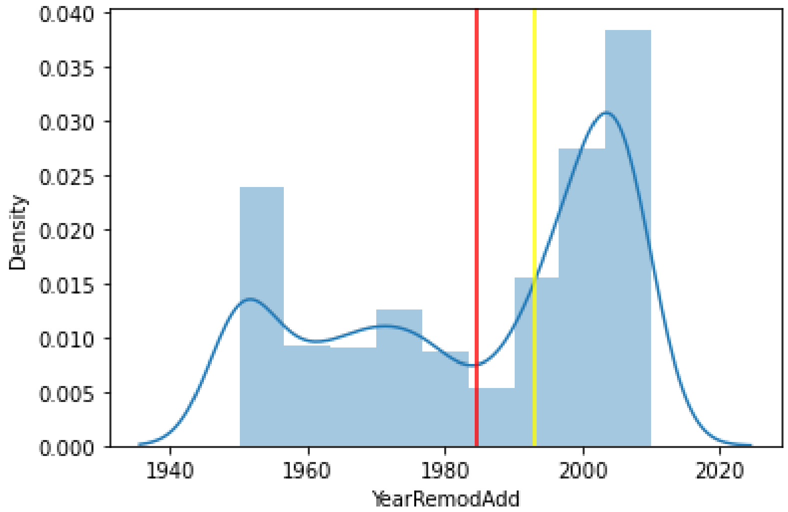

```
YearBuilt -0.6013498391819941
```

```
C:\Users\Student 12\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



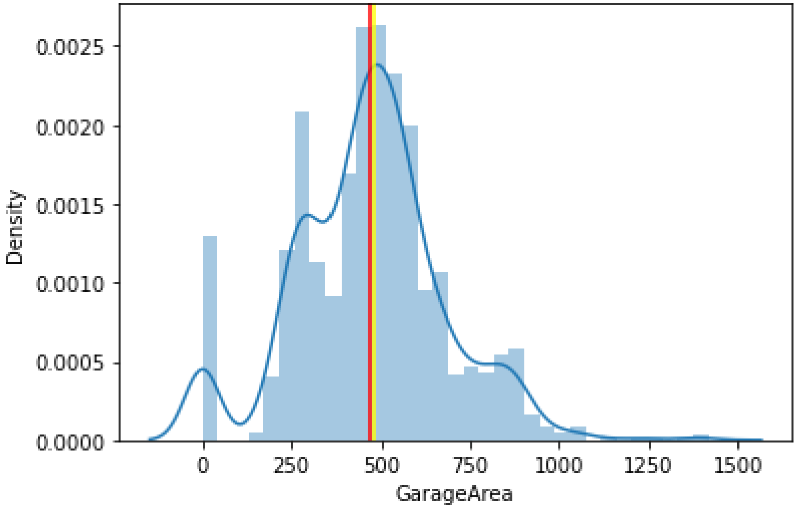

```
YearRemodAdd -0.4920110147731343
```

```
C:\Users\Student 12\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



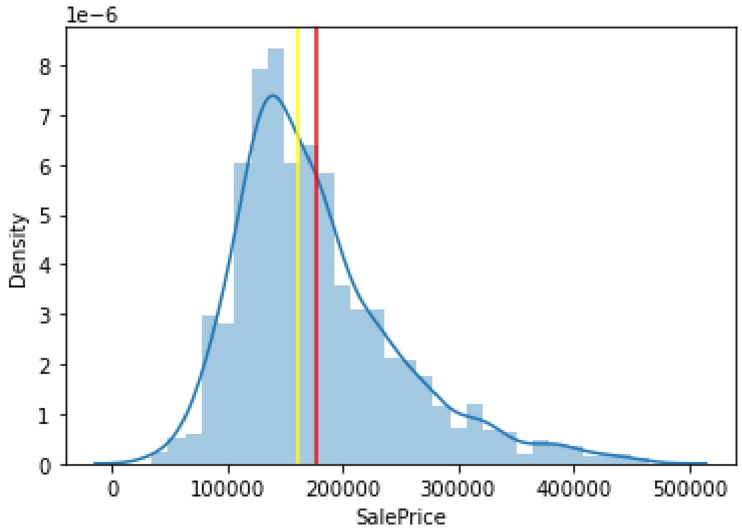
```
GarageArea 0.16792353190885764
```

```
C:\Users\Student 12\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
SalePrice 1.1441959932511412
```

```
C:\Users\Student 12\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [142…

```
# sometimes version conflict or memory overflow happen in jupyter notebook so to avoid warnings we can
# use give command
import warnings
warnings.filterwarnings('ignore')
```

''' there are 4 ways to normalize our data and to reduce skewness of our data 1) squar root of all features 2) log10 for all features 3) minmax scaller 4) standered scaller '''

In [143…

```
# reduce skewness using squar root
house_num
```

Out[143…

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	GarageArea	SalePrice
Id									
1	60	65.0	8450	7	5	2003	2003	548	208500
2	20	80.0	9600	6	8	1976	1976	460	181500
3	60	68.0	11250	7	5	2001	2002	608	223500
4	70	60.0	9550	7	5	1915	1970	642	140000

Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	GarageArea	SalePrice
5	60	84.0	14260	8	5	2000	2000	836	250000
...	...	...	...	...	...	...	...	...	...
1456	60	62.0	7917	6	5	1999	2000	460	175000
1457	20	85.0	13175	6	6	1978	1988	500	210000
1458	70	66.0	9042	7	9	1941	2006	252	266500
1459	20	68.0	9717	5	6	1950	1996	240	142125
1460	20	75.0	9937	5	6	1965	1965	276	147500

1448 rows × 9 columns

skewness reducing using squarroot

In [144…

```
# the columns which skewness values is >=1 and <= -1 those columns skewness we will reduce
house_num_sqrt=house_num.copy()
for col in house_num_sqrt:
    if skew(house_num_sqrt[col]) >=1 or skew(house_num_sqrt[col])<=-1:
        house_num_sqrt[col]=np.sqrt(house_num_sqrt[col])
```

In [145…

```
house_num_sqrt
# price = stockmarket / comodity continuous
# money = bankaccount /paytm wallet discrete
```

Out[145…

Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	GarageArea	SalePrice
1	7.745967	8.062258	91.923882	7	5	2003	2003	548	456.618002
2	4.472136	8.944272	97.979590	6	8	1976	1976	460	426.028168

Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	GarageArea	SalePrice
3	7.745967	8.246211	106.066017	7	5	2001	2002	608	472.757866
4	8.366600	7.745967	97.724101	7	5	1915	1970	642	374.165739
5	7.745967	9.165151	119.415242	8	5	2000	2000	836	500.000000
...	...	...	...	...	...	...	...	...	...
1456	7.745967	7.874008	88.977525	6	5	1999	2000	460	418.330013
1457	4.472136	9.219544	114.782403	6	6	1978	1988	500	458.257569
1458	8.366600	8.124038	95.089432	7	9	1941	2006	252	516.236380
1459	4.472136	8.246211	98.574845	5	6	1950	1996	240	376.994695
1460	4.472136	8.660254	99.684502	5	6	1965	1965	276	384.057287

1448 rows × 9 columns

In [146…

```
house_num['YearBuilt'].value_counts()
```

Out[146…

```
2006    67
2005    63
2004    54
2007    49
2003    44
        ..
1906     1
1911     1
1913     1
1917     1
1872     1
Name: YearBuilt, Length: 112, dtype: int64
```

In [147…

```
house_num
```

Out[147…

MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	GarageArea	SalePrice

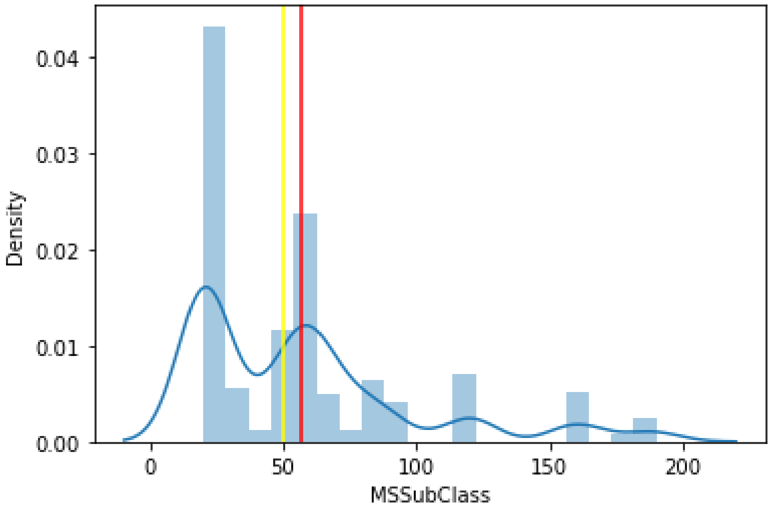
	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	GarageArea	SalePrice
Id										
1	60	65.0	8450	7	5	2003	2003	548	208500
2	20	80.0	9600	6	8	1976	1976	460	181500
3	60	68.0	11250	7	5	2001	2002	608	223500
4	70	60.0	9550	7	5	1915	1970	642	140000
5	60	84.0	14260	8	5	2000	2000	836	250000
...	...	...	...	...	...	...	...	...	...
1456	60	62.0	7917	6	5	1999	2000	460	175000
1457	20	85.0	13175	6	6	1978	1988	500	210000
1458	70	66.0	9042	7	9	1941	2006	252	266500
1459	20	68.0	9717	5	6	1950	1996	240	142125
1460	20	75.0	9937	5	6	1965	1965	276	147500

1448 rows × 9 columns

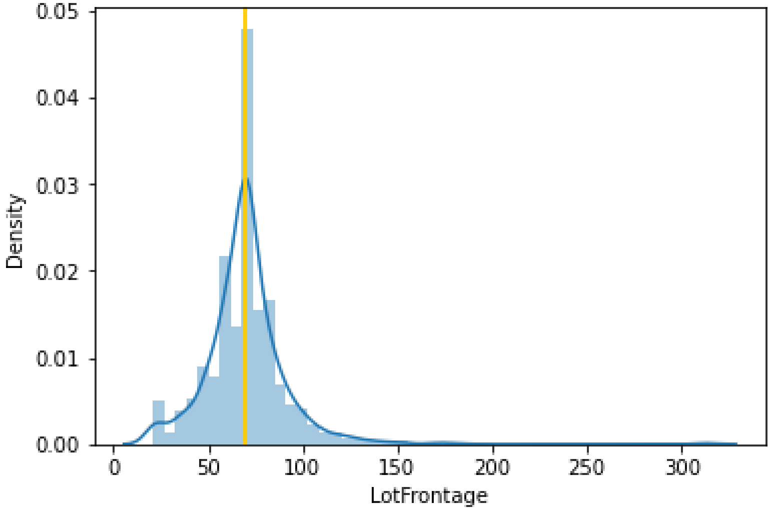
In [148…

```
for col in house_num:
    print(col,skew(house_num[col]))
    plt.figure()
    sns.distplot(house_num[col])
    plt.axvline(house_num[col].mean(),color="red")
    plt.axvline(house_num[col].median(),color="yellow")
    plt.show()
```

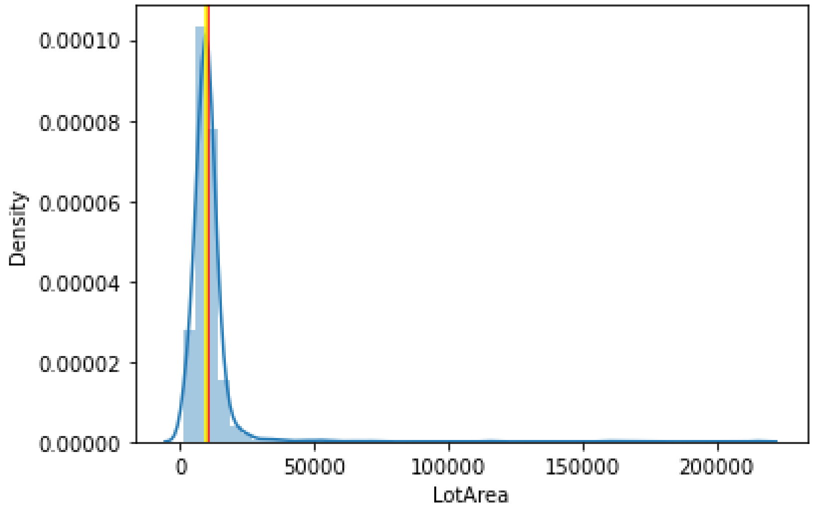
```
MSSubClass 1.4016374700451684
```



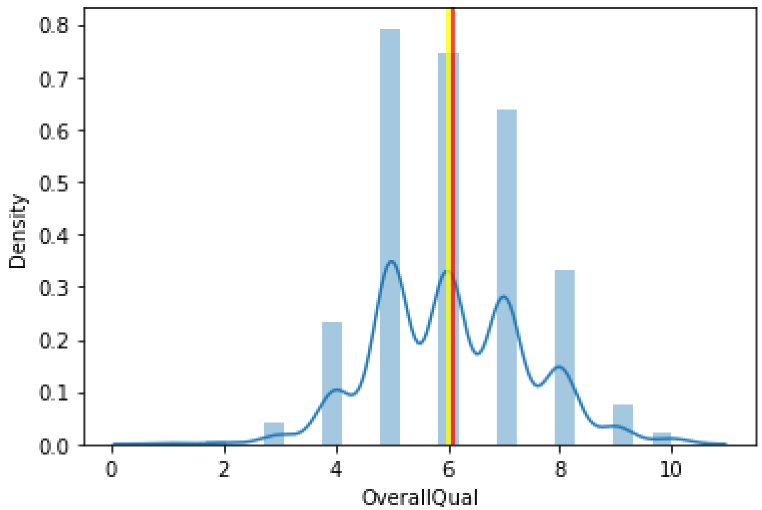
LotFrontage 2.4241983272815286



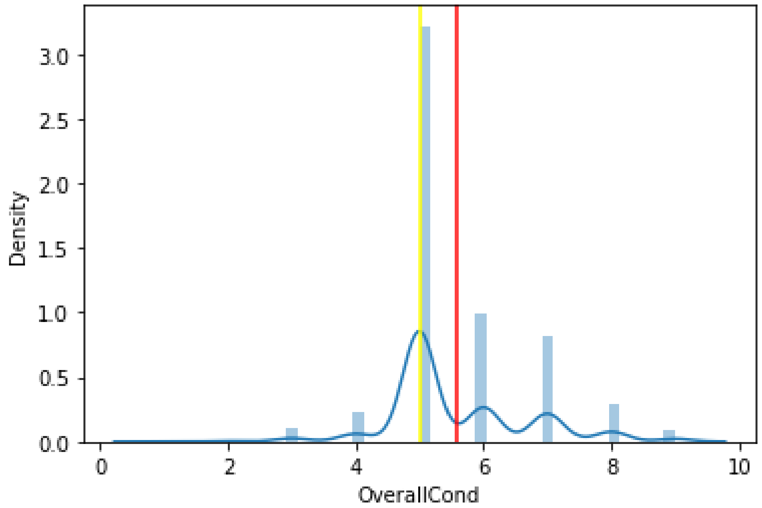
LotArea 12.476597041303394



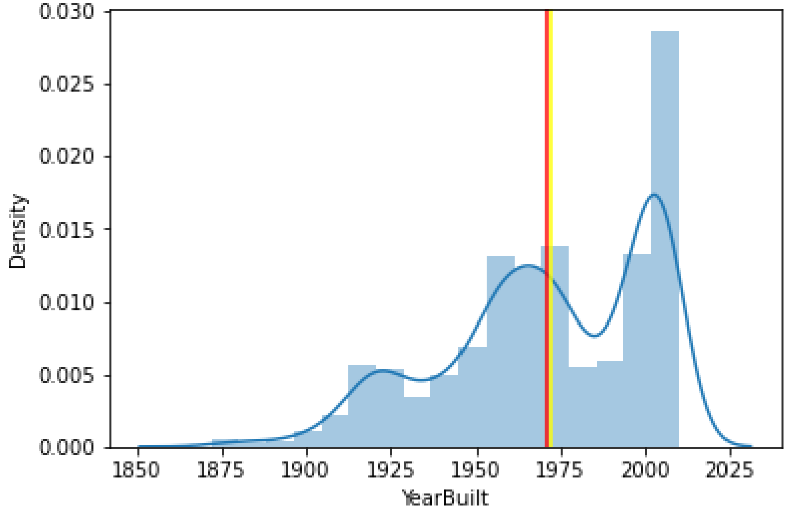

```
OverallQual 0.15874882949146252
```



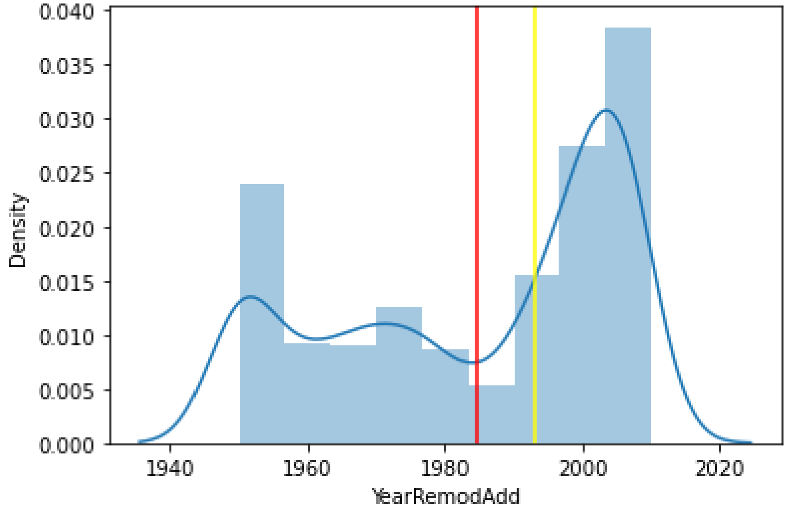

```
OverallCond 0.6755243853201703
```



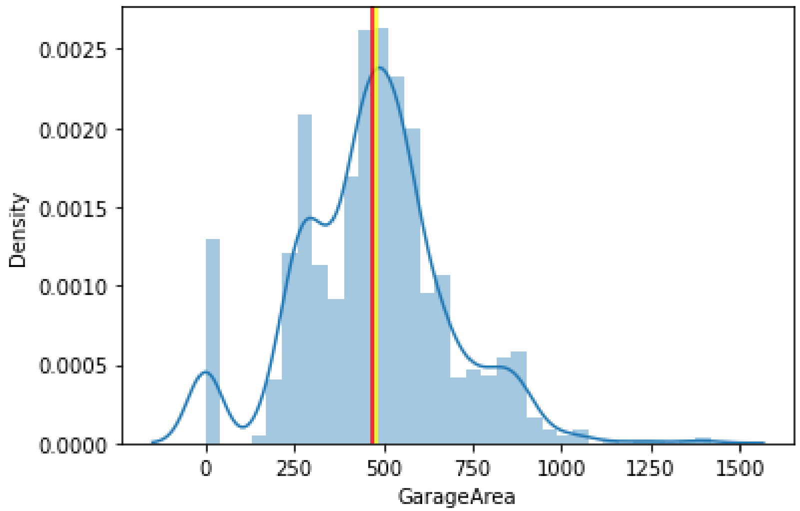

```
YearBuilt -0.6013498391819941
```



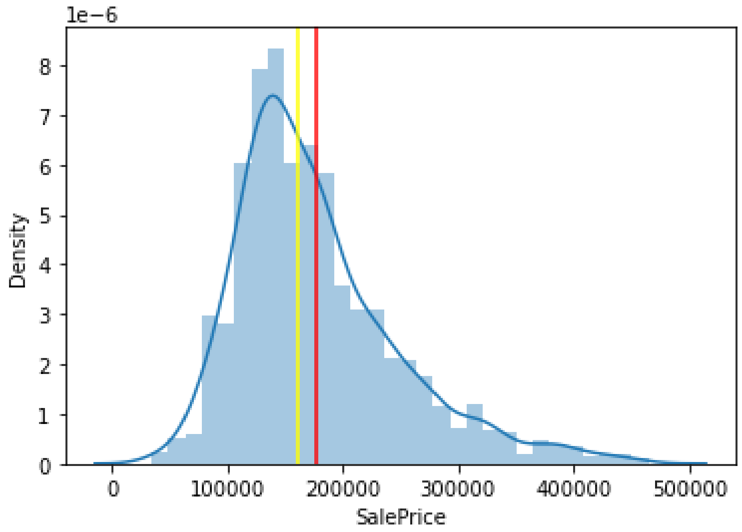

```
YearRemodAdd -0.4920110147731343
```




```
GarageArea 0.16792353190885764
```




```
SalePrice 1.1441959932511412
```



In [149…

```
house_num_minmaxscale=house_num.copy()
```

In [150…

```
house_num_minmaxscale
```

Out[150…

Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	GarageArea	SalePrice
1	60	65.0	8450	7	5	2003	2003	548	208500
2	20	80.0	9600	6	8	1976	1976	460	181500
3	60	68.0	11250	7	5	2001	2002	608	223500
4	70	60.0	9550	7	5	1915	1970	642	140000
5	60	84.0	14260	8	5	2000	2000	836	250000
...	...	...	...	...	...	...	...	...	...
1456	60	62.0	7917	6	5	1999	2000	460	175000
1457	20	85.0	13175	6	6	1978	1988	500	210000
1458	70	66.0	9042	7	9	1941	2006	252	266500

Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	GarageArea	SalePrice
1459	20	68.0	9717	5	6	1950	1996	240	142125
1460	20	75.0	9937	5	6	1965	1965	276	147500

1448 rows × 9 columns

In [151…

```
from sklearn.preprocessing  import MinMaxScaler
scaler=MinMaxScaler()

# it always return series but i need dataframe
# house_num_minmaxscale =scaler.fit_transform(house_num_minmaxscale)
# dataframe
house_num_minmaxscale = pd.DataFrame(scaler.fit_transform(house_num_minmaxscale.values),
                                     columns=house_num_minmaxscale.columns,
                                     index=house_num_minmaxscale.index)
```

In [152…

```
hofrom sklearn.preprocessing  import MinMaxScaler
scaler=MinMaxScaler()

# it always return series but i need dataframe
# house_num_minmaxscale =scaler.fit_transform(house_num_minmaxscale)
# dataframe
house_num_minmaxscale = pd.DataFrame(scaler.fit_transform(house_num_minmaxscale.values),
                                     columns=house_num_minmaxscale.columns,
                                     index=house_num_minmaxscale.index)use_num_minmaxscale
```

```
  File "<ipython-input-152-65a1b593f187>", line 1
    hofrom sklearn.preprocessing  import MinMaxScaler
           ^
SyntaxError: invalid syntax
```

In [ ]:

```
house_num_sqrt
```

In [ ]:

```
for col in house_num_minmaxscale:
```

```python
    print(col,skew(house_num_minmaxscale[col]))
    plt.figure()
    sns.distplot(house_num_minmaxscale[col])
    plt.axvline(house_num_minmaxscale[col].mean(),color="red")
    plt.axvline(house_num_minmaxscale[col].median(),color="yellow")
    plt.show()
```

In [ ]:

```python
for i in house_num_minmaxscale:
    print(i,skew(house_num_minmaxscale[i]))
```

In [ ]:

```python
# to reduce skewness we will apply all those 4 techniques and then we will derrived final conclusion
```

In [ ]:

```python
x=[-12,-1,11,11.5,11.2,12,13,15,20,60,120,500]
newx=[i-min(x)/max(x)-min(x) for i in x] # it always scale or our values from 0 to positive infinitty
```

In [ ]:

```python
newx
```

In [ ]:

```python
newx=np.array(newx)
import math
plt.figure()
sns.distplot(newx)
plt.axvline(np.mean(newx),color="red")
plt.axvline(np.median(newx),color="yellow")
plt.show()
```

In [ ]:

```python
# 1) if we want to apply standered scaler than first remove outliers
# if there is high outliers or low outliers we will not get standered or standered normal distribution data
# Standered scaler
# newx=(x-mean)/sd
house_num_standered_scale=house_num.copy()
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
house_num_standered_scale = pd.DataFrame(scaler.fit_transform(house_num_standered_scale.values),
```

```
                                       columns=house_num_standered_scale.columns,
                                       index=house_num_standered_scale.index)
```

In [ ]:

```
house_num_standered_scale
```

In [ ]:

```
for i in house_num_standered_scale:
    print(i,skew(house_num_standered_scale[i]))
```

spain 200 family average income 1300$weekly100family4500$ near by give data earning happening 200 and 100 skewness means how much my data is deviated from mean log10 # this is highly skewness and outlier treatment

In [ ]:

```
x=60
print("log10 => ",np.log10(60))
print("log => ",np.log(60))
```

Home work 1) minmaxscaller application example 2) standered scaller application example 3) squareroot application example 4) log10,log application example all above use for skewness and normalization of data in data science

In [ ]:

```

```