DATA PROCESSING

Dr. Shyama Prasad Mukherjee International Institute of information technology

Khelawan Singh(201020424) DSAI

Abstract

Data processing occurs when data is collected and translated into usable information. Usually performed by a data scientist or team of data scientists, it is important for data processing to be done correctly as not to negatively affect the end product, or data output.

Data processing starts with data in its raw form and converts it into a more readable format (graphs, documents, etc.), giving it the form and context necessaryto be interpreted by computers and utilized by employees throughout an organization.

Six stages of data processing

1. Data collection

Collecting data is the first step in data processing. Data is pulled from available sources, including data lakes and data warehouses. It is important that the data sources available are trustworthy and well-built so the data collected (and later used as information) is of the highest possible quality.

2. Data preparation

Once the data is collected, it then enters the data preparation stage. Data preparation, often referred to as "pre-processing" is the stage at which raw data is cleaned up and organized for the following stage of data processing. During preparation, raw data is diligently checked for any errors. The purpose of this step is to eliminate bad data , incomplete, or incorrect data) and begin to create high-quality data for the best business intelligence.

3. Data input

The clean data is then entered into its destination (perhaps a CRM like Salesforces or a data warehouse like Redshift, and translated into a language that it can understand. Data input is the first stage in which raw data begins to take the form of usable information.

4. Processing

During this stage, the data inputted to the computer in the previous stage is actually processed for interpretation. Processing is done using machine lerning algorithms, though the process itself may vary slightly depending on the source of data being processed (data lakes, social networks, connected devices etc.) and its intended use (examining advertising patterns, medical diagnosis from connected devices, determining customer needs, etc.).

5. Data output/interpretation

The output/interpretation stage is the stage at which data is finally usable to non-data scientists. It is translated, readable, and often in the form of graphs, videos, images, plain text, etc.). Members of the company or institution can now begin to for their own projects.

6. Data storage

The final stage of data processing is Storage After all of the data is processed, it is then stored for future use. While some information may be put to use immediately, much of it will serve a purpose later on. Plus, properly stored data is a necessity for compliance with data protection legislation like GDPR. When data is properly stored, it can be quickly and easily accessed by members of the organization when needed.

DATA IMPUTATION

Data cleaning

Data cleaning means fixing bad data in your data set.
Dirty data could be:
- Empty cells
- Data in wrong format
- Wrong data
- Duplicates

Code for data imputaion

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv('E:\data  processing\housing.csv')
```

```
df.he
ad()
```

Out[4]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500

1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

5 rows × 81 columns

##df.head is used for printing the data set##

subset of data to work with

```
housing = df[['LotFrontage','LotArea','LotShape','SalePrice','YrSold','SaleCondition','BsmtQual','GarageType','GarageArea','FireplaceQu']].copy()
##it is used to make the copy of data set with required coloms##
```

```
housing.head(20)
## printing the first 20 rows of data set##
ihfjhfijh
```

Out[6]:

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu
0	65.0	8450	Reg	208500	2008	Normal	Gd	Attchd	548	NaN
1	80.0	9600	Reg	181500	2007	Normal	Gd	Attchd	460	TA
2	68.0	11250	IR1	223500	2008	Normal	Gd	Attchd	608	TA
3	60.0	9550	IR1	140000	2006	Abnorml	TA	Detchd	642	Gd
4	84.0	14260	IR1	250000	2008	Normal	Gd	Attchd	836	TA
5	85.0	14115	IR1	143000	2009	Normal	Gd	Attchd	480	NaN
6	75.0	10084	Reg	307000	2007	Normal	Ex	Attchd	636	Gd
7	NaN	10382	IR1	200000	2009	Normal	Gd	Attchd	484	TA
8	51.0	6120	Reg	129900	2008	Abnorml	TA	Detchd	468	TA
9	50.0	7420	Reg	118000	2008	Normal	TA	Attchd	205	TA
10	70.0	11200	Reg	129500	2008	Normal	TA	Detchd	384	NaN
11	85.0	11924	IR1	345000	2006	Partial	Ex	BuiltIn	736	Gd

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu
12	**NaN**	**12968**	**IR2**	**144000**	**2008**	**Normal**	**TA**	**Detchd**	**352**	**NaN**
13	**91.0**	**10652**	**IR1**	**279500**	**2007**	**Partial**	**Gd**	**Attchd**	**840**	**Gd**
14	**NaN**	**10920**	**IR1**	**157000**	**2008**	**Normal**	**TA**	**Attchd**	**352**	**Fa**
15	**51.0**	**6120**	**Reg**	**132000**	**2007**	**Normal**	**TA**	**Detchd**	**576**	**NaN**
16	**NaN**	**11241**	**IR1**	**149000**	**2010**	**Normal**	**TA**	**Attchd**	**480**	**TA**
17	**72.0**	**10791**	**Reg**	**90000**	**2006**	**Normal**	**NaN**	**CarPort**	**516**	**NaN**
18	**66.0**	**13695**	**Reg**	**159000**	**2008**	**Normal**	**TA**	**Detchd**	**576**	**NaN**
19	**70.0**	**7560**	**Reg**	**139000**	**2009**	**Abnorml**	**TA**	**Attchd**	**294**	**NaN**

In [7]:

```
housing.isnull().sum()
## is is used to know the missing values##
```

Out[7]:

```
LotFrontage      259
LotArea            0
LotShape           0
SalePrice          0
YrSold             0
SaleCondition      0
BsmtQual          37
GarageType        81
GarageArea         0
FireplaceQu      690
dtype: int64
```

In [8]:

```
housing.isnull().sum()/len(housing)
##to know the percentage of missing values##
```

Out[8]:

```
LotFrontage      0.177397
LotArea          0.000000
LotShape         0.000000
SalePrice        0.000000
YrSold           0.000000
SaleCondition    0.000000
BsmtQual         0.025342
GarageType       0.055479
GarageArea       0.000000
FireplaceQu      0.472603
dtype: float64
```

In [9]:

```
# Finding mean and median for imputation

# finding missing values
```

In [10]:

```
housing.isnull().mean()
## used to find the mean of coloms##
```

Out[10]:

```
LotFrontage      0.177397
LotArea          0.000000
LotShape         0.000000
SalePrice        0.000000
YrSold           0.000000
```

SaleCondition	0.000000
BsmtQual	0.025342

```
GarageType      0.055479
GarageArea      0.000000
FireplaceQu     0.472603
dtype: float64
```

In [11]:

```python
# Step 1 - Seprate the numerical data

housing1 =
  df[['LotFrontage','LotArea','Sal
  ePrice','YrSold','GarageArea']]
  housing1.head(20)
```

Out[11]:

	LotFrontage	LotArea	SalePrice	YrSold	GarageArea
0	65.0	8450	208500	2008	548
1	80.0	9600	181500	2007	460
2	68.0	11250	223500	2008	608
3	60.0	9550	140000	2006	642
4	84.0	14260	250000	2008	836
5	85.0	14115	143000	2009	480
6	75.0	10084	307000	2007	636
7	NaN	10382	200000	2009	484
8	51.0	6120	129900	2008	468
9	50.0	7420	118000	2008	205
10	70.0	11200	129500	2008	384
11	85.0	11924	345000	2006	736
12	NaN	12968	144000	2008	352
13	91.0	10652	279500	2007	840
14	NaN	10920	157000	2008	352
15	51.0	6120	132000	2007	576
16	NaN	11241	149000	2010	480
17	72.0	10791	90000	2006	516
18	66.0	13695	159000	2008	576
19	70.0	7560	139000	2009	294

In [12]:

```python
# step 2 - calculate the mean for the seperated data set and display the coloum having maximum percentage of missing values.

housing1.isnull().mean()
```

Out[12]:

```
LotFrontage     0.177397
LotArea         0.000000
SalePrice       0.000000
YrSold          0.000000
GarageArea      0.000000
```

dtype: float64

In [13]:

Out[13]: 70.04995836802665

In [14]:

Out[14]: 69.0

In [15]:

```python
# step 3 - Replaceing the missing values with mean and median on column LotFrontage.

housing1["LotFrontage_mean"]  =  housing1.LotFrontage.fillna(mean)
housing1["LotFrontage_median"] = housing1.LotFrontage.fillna(median)

housing1.head(20)
```

<ipython-input-15-ac69c26a556d>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.   Try using
.loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy housing1["LotFrontage_mean"] =
housing1.LotFrontage.fillna(mean)
<ipython-input-15-ac69c26a556d>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.   Try using
.loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy housing1["LotFrontage_median"] =
housing1.LotFrontage.fillna(median)

Out[15]:

	LotFrontage	LotArea	SalePrice	YrSold	GarageArea	LotFrontage_mean	LotFrontage_median
0	65.0	8450	208500	2008	548	65.000000	65.0
1	80.0	9600	181500	2007	460	80.000000	80.0
2	68.0	11250	223500	2008	608	68.000000	68.0
3	60.0	9550	140000	2006	642	60.000000	60.0
4	84.0	14260	250000	2008	836	84.000000	84.0
5	85.0	14115	143000	2009	480	85.000000	85.0
6	75.0	10084	307000	2007	636	75.000000	75.0
7	NaN	10382	200000	2009	484	70.049958	69.0
8	51.0	6120	129900	2008	468	51.000000	51.0
9	50.0	7420	118000	2008	205	50.000000	50.0
10	70.0	11200	129500	2008	384	70.000000	70.0
11	85.0	11924	345000	2006	736	85.000000	85.0

12	NaN	12968	144000	2008	352	70.049958	69.0

	LotFrontage	LotArea	SalePrice	YrSold	GarageArea	LotFrontage_mean	LotFrontage_median
13	91.0	10652	279500	2007	840	91.000000	91.0
14	NaN	10920	157000	2008	352	70.049958	69.0
15	51.0	6120	132000	2007	576	51.000000	51.0
16	NaN	11241	149000	2010	480	70.049958	69.0
17	72.0	10791	90000	2006	516	72.000000	72.0
18	66.0	13695	159000	2008	576	66.000000	66.0
19	70.0	7560	139000	2009	294	70.000000	70.0

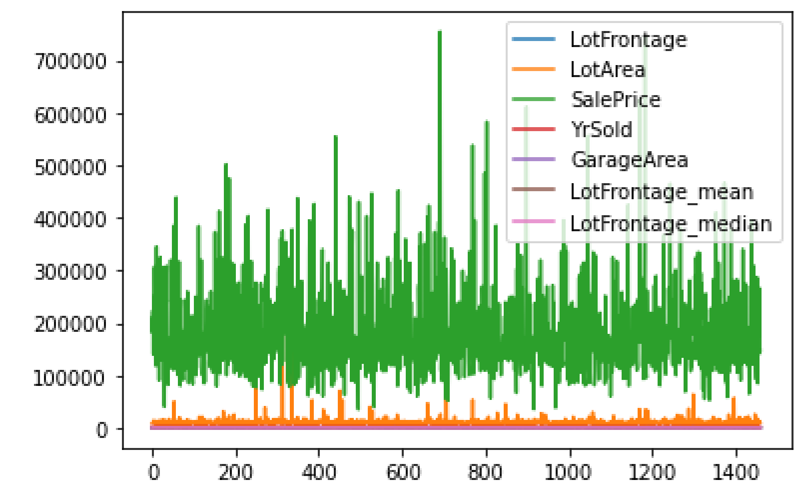
In [16]:

```
# plot graph for the data set to know the resultant data distribution

housing1.plot()
```

Out[16]:

<AxesSubplot:>

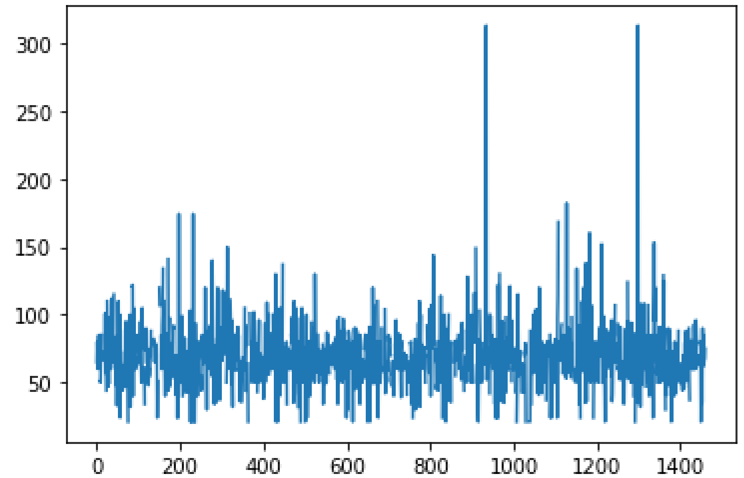


In [17]:

```
# ploting Graph for Lot Frontage column
housing1.LotFrontage.plot()
```

Out[17]:

<AxesSubplot:>

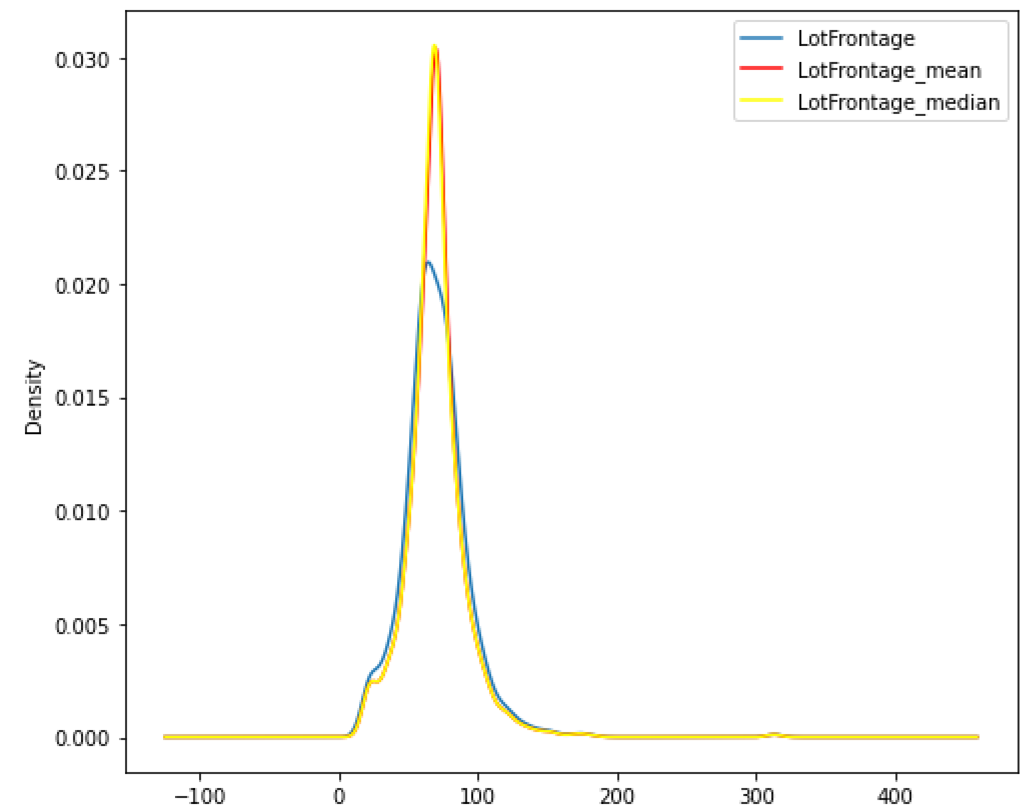

300
250
200
150
100
50
0
200
400
600
800
1000
1200
1400

In [18]:

```
# PLOT THE LINE GRAPH FOR THE LotFrontage , LotFrontage_mean and LotFrontage_median
```

Out[18]:

<matplotlib.legend.Legend at 0x26d76fd7910>



In [19]:

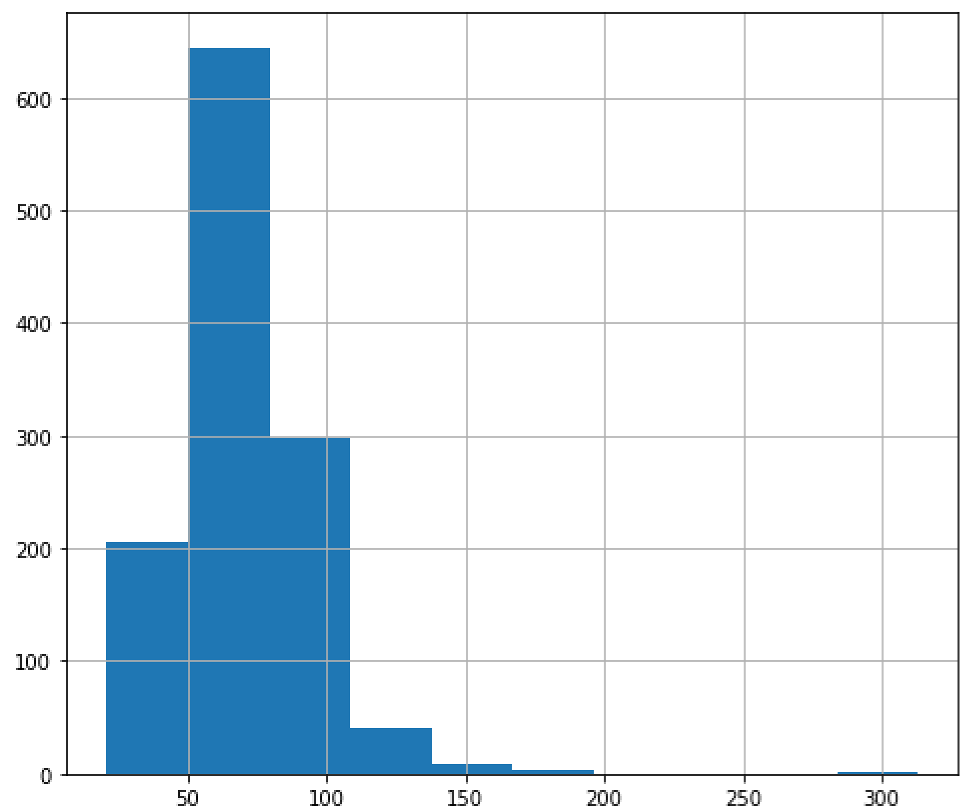
```
# end of imputation

# checking the dta is nornally distributed or not

housing1.LotFrontage.hist()
```

Out[19]:

<AxesSubplot:>

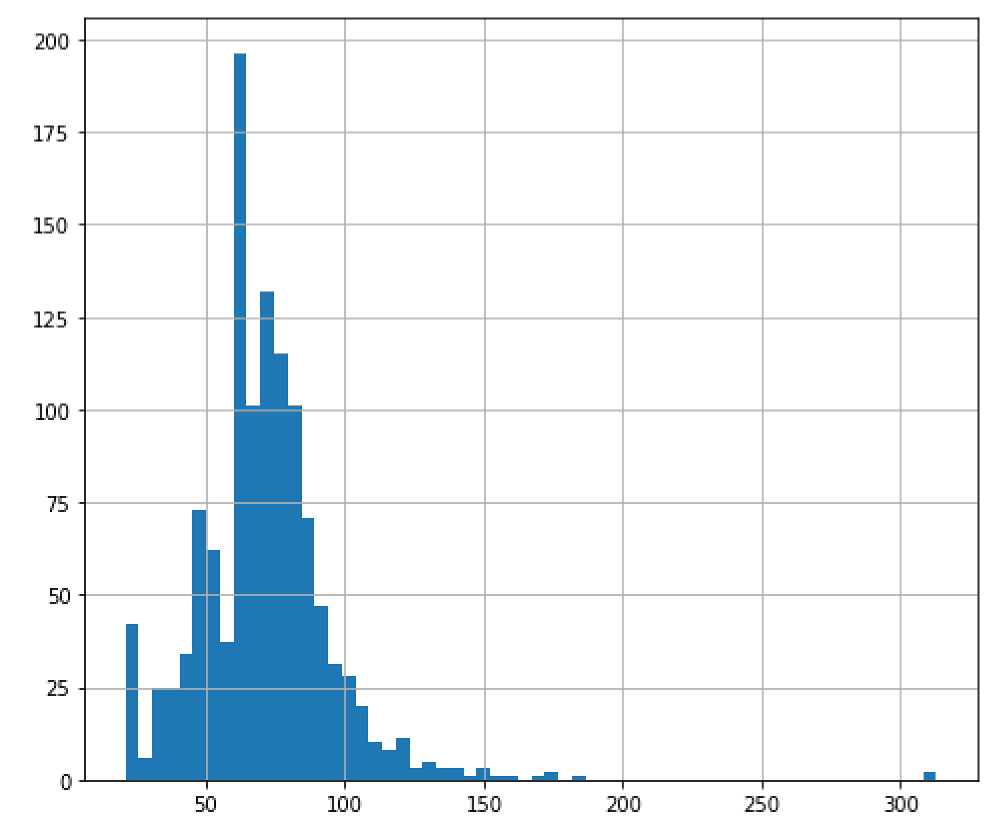


In [20]:

```
housing1.LotFrontage.hist(bins=60)
```

Out[20]:

<AxesSubplot:>



In [21]:

```
# data is normally distributed so compute end of distribution value
```

```
eod_value =
housing1.LotFrontage.mean()
+ 3 *
housing1.LotFrontage.std()
print(eod_value)
```

97.33471014250986

In [22]:

```
# create a new column age_eod and filling missing values with end of distribution value
housing1['LotFrontage_eod'] =
housing1.LotFrontage.fillna(eo
d_value) housing1.head(20)

# Printing first 20 rows after filling with eod_
```

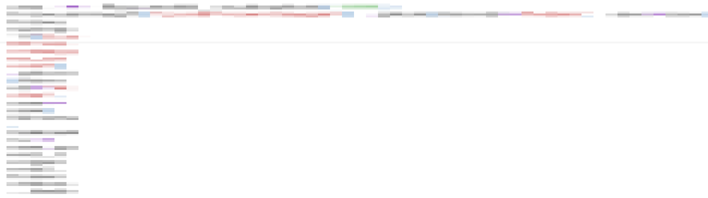
<ipython-input-22-54f91150ed18>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame. Try using
.loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy housing1['LotFrontage_eod'] =
housing1.LotFrontage.fillna(eod_value)

Out[22]:

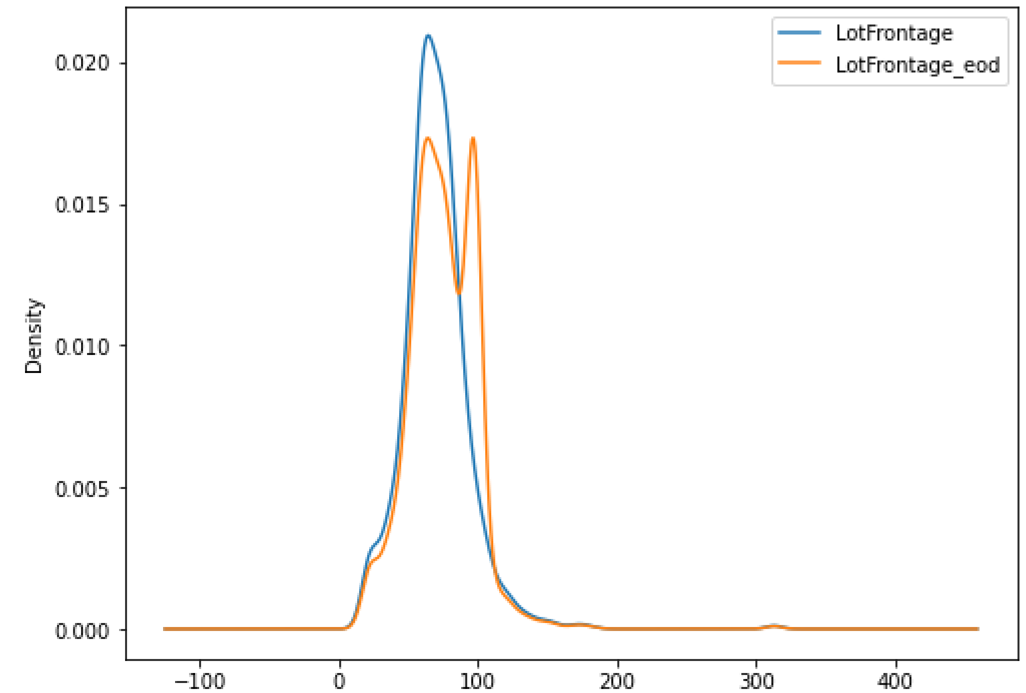
	LotFrontage	LotArea	SalePrice	YrSold	GarageArea	LotFrontage_mean	LotFrontage_median	LotFrontage_eod
0	65.0	8450	208500	2008	548	65.000000	65.0	65.00000
1	80.0	9600	181500	2007	460	80.000000	80.0	80.00000
2	68.0	11250	223500	2008	608	68.000000	68.0	68.00000
3	60.0	9550	140000	2006	642	60.000000	60.0	60.00000
4	84.0	14260	250000	2008	836	84.000000	84.0	84.00000
5	85.0	14115	143000	2009	480	85.000000	85.0	85.00000
6	75.0	10084	307000	2007	636	75.000000	75.0	75.00000
7	NaN	10382	200000	2009	484	70.049958	69.0	97.33471
8	51.0	6120	129900	2008	468	51.000000	51.0	51.00000
9	50.0	7420	118000	2008	205	50.000000	50.0	50.00000
10	70.0	11200	129500	2008	384	70.000000	70.0	70.00000
11	85.0	11924	345000	2006	736	85.000000	85.0	85.00000
12	NaN	12968	144000	2008	352	70.049958	69.0	97.33471
13	91.0	10652	279500	2007	840	91.000000	91.0	91.00000
14	NaN	10920	157000	2008	352	70.049958	69.0	97.33471
15	51.0	6120	132000	2007	576	51.000000	51.0	51.00000
16	NaN	11241	149000	2010	480	70.049958	69.0	97.33471
17	72.0	10791	90000	2006	516	72.000000	72.0	72.00000
18	66.0	13695	159000	2008	576	66.000000	66.0	66.00000
19	70.0	7560	139000	2009	294	70.000000	70.0	70.00000

In [23]:



Out[23]:

<matplotlib.legend.Legend at 0x26d771d6b20>

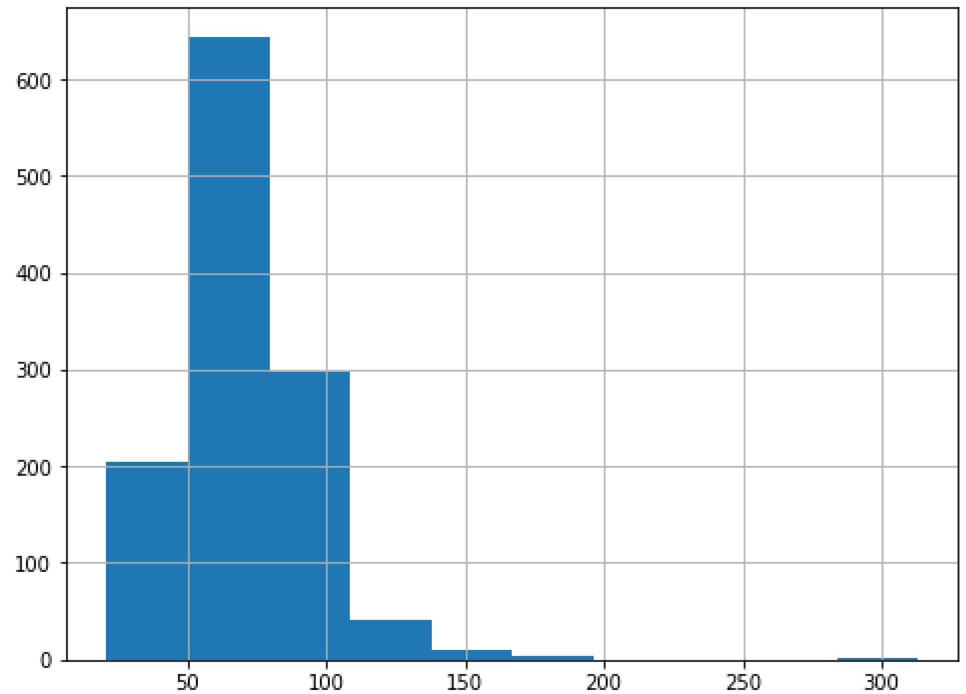


In [24]:

```
# plot histrogram to check the data distribution
housing.LotFrontage.hist()
```

Out[24]:

<AxesSubplot:>



0
100
200
300
400
500
600
50
100
150
200
250
300

In [25]:

```
# Filling with 99 and -1 on Age column

# create new columns age_99 and age_minus1 and filling missing values
housing1['LotFrontage_99'] = housing1.LotFrontage.fillna(99)
housing1['LotFronta
ge_minus1'] =
housing1.LotFronta
ge.fillna(-1)
housing1.head(20)
```

<ipython-input-25-987b17d611a8>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame. Try using
.loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy housing1['LotFrontage_99'] =
housing1.LotFrontage.fillna(99)
<ipython-input-25-987b17d611a8>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame. Try using
.loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy housing1['LotFrontage_minus1'] =
housing1.LotFrontage.fillna(-1)

Out[25]:

	LotFrontage	LotArea	SalePrice	YrSold	GarageArea	LotFrontage_mean	LotFrontage_median	LotFrontage_eod	LotFrontage_99	LotFrontage_minus1
0	65.0	8450	208500	2008	548	65.000000	65.0	65.00000	65.0	65.0
1	80.0	9600	181500	2007	460	80.000000	80.0	80.00000	80.0	80.0
2	68.0	11250	223500	2008	608	68.000000	68.0	68.00000	68.0	68.0
3	60.0	9550	140000	2006	642	60.000000	60.0	60.00000	60.0	60.0
4	84.0	14260	250000	2008	836	84.000000	84.0	84.00000	84.0	84.0
5	85.0	14115	143000	2009	480	85.000000	85.0	85.00000	85.0	85.0
6	75.0	10084	307000	2007	636	75.000000	75.0	75.00000	75.0	75.0
7	NaN	10382	200000	2009	484	70.049958	69.0	97.33471	99.0	-1.0
8	51.0	6120	129900	2008	468	51.000000	51.0	51.00000	51.0	51.0
9	50.0	7420	118000	2008	205	50.000000	50.0	50.00000	50.0	50.0
10	70.0	11200	129500	2008	384	70.000000	70.0	70.00000	70.0	70.0
11	85.0	11924	345000	2006	736	85.000000	85.0	85.00000	85.0	85.0
12	NaN	12968	144000	2008	352	70.049958	69.0	97.33471	99.0	-1.0
13	91.0	10652	279500	2007	840	91.000000	91.0	91.00000	91.0	91.0
14	NaN	10920	157000	2008	352	70.049958	69.0	97.33471	99.0	-1.0
15	51.0	6120	132000	2007	576	51.000000	51.0	51.00000	51.0	51.0
16	NaN	11241	149000	2010	480	70.049958	69.0	97.33471	99.0	-1.0
17	72.0	10791	90000	2006	516	72.000000	72.0	72.00000	72.0	72.0
18	66.0	13695	159000	2008	576	66.000000	66.0	66.00000	66.0	66.0
19	70.0	7560	139000	2009	294	70.000000	70.0	70.00000	70.0	70.0

In [26]:

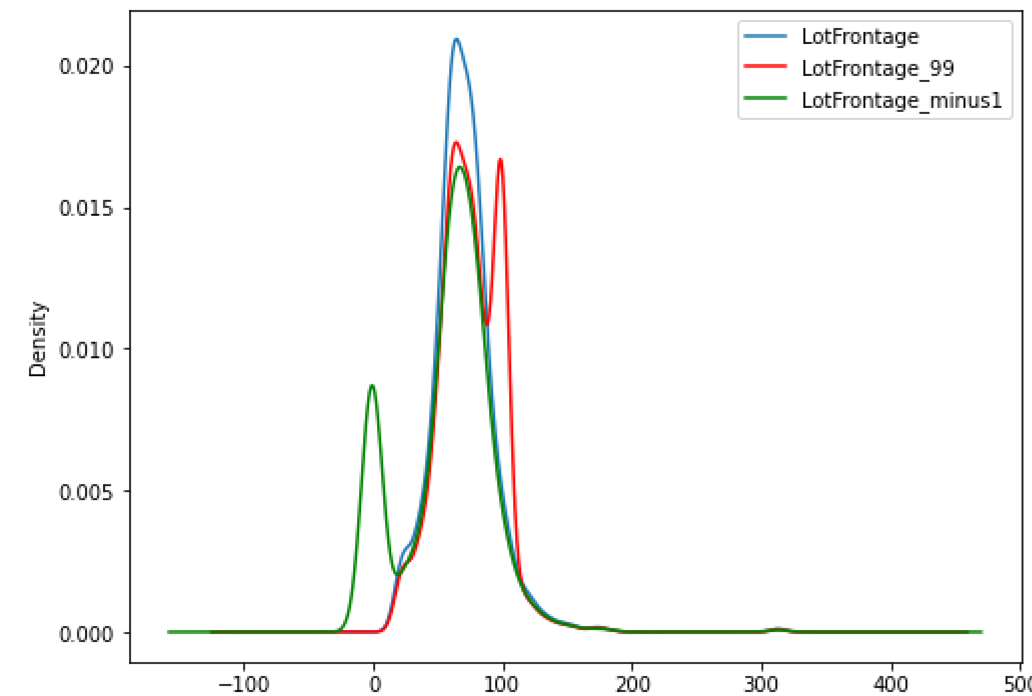
```
# Ploting KDE

plt.rcParams["figure.figsize"] = [8,6]
```

```
fig = plt.figure()
ax  =  fig.add_subplot(111)
housing1['LotFrontage']  .plot(kind='kde',  ax=ax)
housing1['LotFrontage_99']  .plot(kind='kde',  ax=ax,  color='red')
housing1['LotFrontage_min
us1']  .plot(kind='kde',
ax=ax,  color='green') lines,
labels =
ax.get_legend_handles_lab
els()
ax.legend(lines, labels, loc='best')
```

Out[26]:

<matplotlib.legend.Legend at 0x26d772e7820>



In [27]:

```
# Again printing the subset of data to work

housing.head()
```

Out[27]:

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu
0	65.0	8450	Reg	208500	2008	Normal	Gd	Attchd	548	NaN
1	80.0	9600	Reg	181500	2007	Normal	Gd	Attchd	460	TA
2	68.0	11250	IR1	223500	2008	Normal	Gd	Attchd	608	TA
3	60.0	9550	IR1	140000	2006	Abnorml	TA	Detchd	642	Gd
4	84.0	14260	IR1	250000	2008	Normal	Gd	Attchd	836	TA

In [28]:

```
# seperating the categorical data
housing2 = df[['LotFrontage','LotShape','SaleCondition','BsmtQual','GarageType','FireplaceQu']]
```

In [29]:

```
housing2.head(20)
```

Out[29]:

LotFrontage LotShape SaleCondition BsmtQual GarageType FireplaceQu

	LotFrontage	LotShape	SaleCondition	BsmtQual	GarageType	FireplaceQu
0	65.0	Reg	Normal	Gd	Attchd	NaN
1	80.0	Reg	Normal	Gd	Attchd	TA
2	68.0	IR1	Normal	Gd	Attchd	TA
3	60.0	IR1	Abnorml	TA	Detchd	Gd
4	84.0	IR1	Normal	Gd	Attchd	TA
5	85.0	IR1	Normal	Gd	Attchd	NaN
6	75.0	Reg	Normal	Ex	Attchd	Gd
7	NaN	IR1	Normal	Gd	Attchd	TA
8	51.0	Reg	Abnorml	TA	Detchd	TA
9	50.0	Reg	Normal	TA	Attchd	TA
10	70.0	Reg	Normal	TA	Detchd	NaN
11	85.0	IR1	Partial	Ex	BuiltIn	Gd
12	NaN	IR2	Normal	TA	Detchd	NaN
13	91.0	IR1	Partial	Gd	Attchd	Gd
14	NaN	IR1	Normal	TA	Attchd	Fa
15	51.0	Reg	Normal	TA	Detchd	NaN
16	NaN	IR1	Normal	TA	Attchd	TA
17	72.0	Reg	Normal	NaN	CarPort	NaN
18	66.0	Reg	Normal	TA	Detchd	NaN
19	70.0	Reg	Abnorml	TA	Attchd	NaN

In [30]:

```
housing2.isnull().mean()
```

Out[30]:

```
LotFrontage      0.177397
LotShape         0.000000
SaleCondition    0.000000
BsmtQual         0.025342
GarageType       0.055479
FireplaceQu      0.472603
dtype: float64
```

In [31]:

c:\users\asus\appdata\local\programs\python\python39\lib\site-packages\pandas\core\frame.py:4308: SettingWithCopyWarning: A value is trying to be set on a copy of a slice from a DataFrame

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

Out[31]:

In [32]:

```
# now we check the mode of BsmtQual
housing2.BsmtQual.mode()
```

Out[32]:

```
0    TA
dtype: object
```

In [33]:

```
housing2.BsmtQual.fillna('mode')
```

Out[33]:

```
0       Gd
1       Gd
2       Gd
3       TA
4       Gd
.
.
1
4
5
5

G
```

d
1
4
5
6

G

d
1457   TA
1458   TA
1459   TA
Name: BsmtQual, Length: 1460, dtype: object

In [34]:

```
housing2.hea
d(20)
```

Out[34]:

	LotFrontage	LotShape	SaleCondition	BsmtQual	GarageType
0	65.0	Reg	Normal	Gd	Attchd
1	80.0	Reg	Normal	Gd	Attchd
2	68.0	IR1	Normal	Gd	Attchd
3	60.0	IR1	Abnorml	TA	Detchd
4	84.0	IR1	Normal	Gd	Attchd
5	85.0	IR1	Normal	Gd	Attchd
6	75.0	Reg	Normal	Ex	Attchd
7	NaN	IR1	Normal	Gd	Attchd
8	51.0	Reg	Abnorml	TA	Detchd
9	50.0	Reg	Normal	TA	Attchd
10	70.0	Reg	Normal	TA	Detchd
11	85.0	IR1	Partial	Ex	BuiltIn
12	NaN	IR2	Normal	TA	Detchd
13	91.0	IR1	Partial	Gd	Attchd
14	NaN	IR1	Normal	TA	Attchd
15	51.0	Reg	Normal	TA	Detchd
16	NaN	IR1	Normal	TA	Attchd
17	72.0	Reg	Normal	NaN	CarPort
18	66.0	Reg	Normal	TA	Detchd
19	70.0	Reg	Abnorml	TA	Attchd

In [35]:

```
# now we find the mode of LotFrontage

housing2.LotFrontage.mode()
```

Out[35]:

```
0       60.0
dtype: float64
```

In [36]:

```
housing2.LotFrontage.fillna('mode')
```

Out[36]:

```
0       65.0
1       80.0
2       68.0
3       60.0
4       84.0
        ...
```

1455	62.0
1456	85.0
1457	66.0
1458	68.0

```
1459    75.0
Name: LotFrontage, Length: 1460, dtype: object
```

In [37]:

```python
housing2.head(20)
```

Out[37]:

	LotFrontage	LotShape	SaleCondition	BsmtQual	GarageType
0	65.0	Reg	Normal	Gd	Attchd
1	80.0	Reg	Normal	Gd	Attchd
2	68.0	IR1	Normal	Gd	Attchd
3	60.0	IR1	Abnorml	TA	Detchd
4	84.0	IR1	Normal	Gd	Attchd
5	85.0	IR1	Normal	Gd	Attchd
6	75.0	Reg	Normal	Ex	Attchd
7	NaN	IR1	Normal	Gd	Attchd
8	51.0	Reg	Abnorml	TA	Detchd
9	50.0	Reg	Normal	TA	Attchd
10	70.0	Reg	Normal	TA	Detchd
11	85.0	IR1	Partial	Ex	BuiltIn
12	NaN	IR2	Normal	TA	Detchd
13	91.0	IR1	Partial	Gd	Attchd
14	NaN	IR1	Normal	TA	Attchd
15	51.0	Reg	Normal	TA	Detchd
16	NaN	IR1	Normal	TA	Attchd
17	72.0	Reg	Normal	NaN	CarPort
18	66.0	Reg	Normal	TA	Detchd
19	70.0	Reg	Abnorml	TA	Attchd

In [38]:

```python
housing2['LotFrontage_mode'] = housing2.LotFrontage.fillna(60)
housing2.head(20)
```

```
<ipython-input-38-3a7be1a53461>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.    Try using
.loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
  https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
  xing.html#returning-a-view-versus-a-copy
  housing2['LotFrontage_mode'] =
  housing2.LotFrontage.fillna(60)
```

Out[38]:

	LotFrontage	LotShape	SaleCondition	BsmtQual	GarageType	LotFrontage_mode

0	65.0	Reg	Normal	Gd	Attchd	65.0
1	80.0	Reg	Normal	Gd	Attchd	80.0
2	68.0	IR1	Normal	Gd	Attchd	68.0

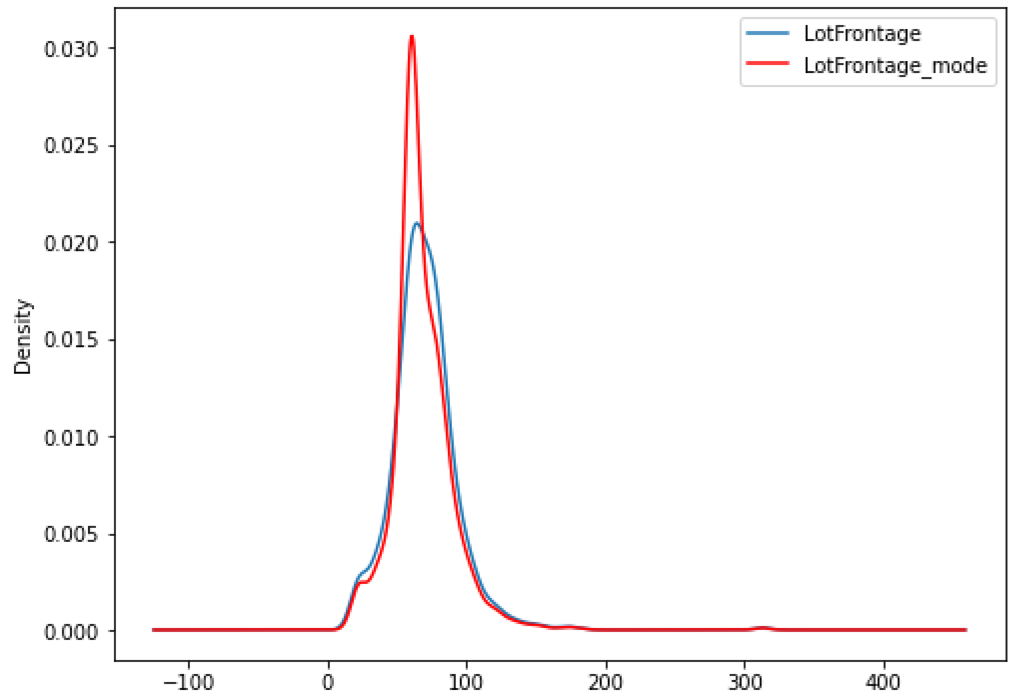
	LotFrontage	LotShape	SaleCondition	BsmtQual	GarageType	LotFrontage_mode
3	60.0	IR1	Abnorml	TA	Detchd	60.0
4	84.0	IR1	Normal	Gd	Attchd	84.0
5	85.0	IR1	Normal	Gd	Attchd	85.0
6	75.0	Reg	Normal	Ex	Attchd	75.0
7	NaN	IR1	Normal	Gd	Attchd	60.0
8	51.0	Reg	Abnorml	TA	Detchd	51.0
9	50.0	Reg	Normal	TA	Attchd	50.0
10	70.0	Reg	Normal	TA	Detchd	70.0
11	85.0	IR1	Partial	Ex	BuiltIn	85.0
12	NaN	IR2	Normal	TA	Detchd	60.0
13	91.0	IR1	Partial	Gd	Attchd	91.0
14	NaN	IR1	Normal	TA	Attchd	60.0
15	51.0	Reg	Normal	TA	Detchd	51.0
16	NaN	IR1	Normal	TA	Attchd	60.0
17	72.0	Reg	Normal	NaN	CarPort	72.0
18	66.0	Reg	Normal	TA	Detchd	66.0
19	70.0	Reg	Abnorml	TA	Attchd	70.0

In [39]:

```
[illegible]
```

Out[39]:

<matplotlib.legend.Legend at 0x26d7740f460>



	LotFrontage	LotShape	SaleCondition	BsmtQual	GarageType
0	65.0	Reg	Normal	Gd	Attchd
1	80.0	Reg	Normal	Gd	Attchd
2	68.0	IR1	Normal	Gd	Attchd
3	60.0	IR1	Abnorml	TA	Detchd
4	84.0	IR1	Normal	Gd	Attchd
5	85.0	IR1	Normal	Gd	Attchd
6	75.0	Reg	Normal	Ex	Attchd
7	NaN	IR1	Normal	Gd	Attchd
8	51.0	Reg	Abnorml	TA	Detchd
9	50.0	Reg	Normal	TA	Attchd
10	70.0	Reg	Normal	TA	Detchd
11	85.0	IR1	Partial	Ex	BuiltIn
12	NaN	IR2	Normal	TA	Detchd
13	91.0	IR1	Partial	Gd	Attchd
14	NaN	IR1	Normal	TA	Attchd
15	51.0	Reg	Normal	TA	Detchd
16	NaN	IR1	Normal	TA	Attchd
17	72.0	Reg	Normal	NaN	CarPort
18	66.0	Reg	Normal	TA	Detchd
19	70.0	Reg	Abnorml	TA	Attchd

In [40]:

```
# Missing Category Imputation
housing2.head(20)
```

Out[40]:

	LotFrontage	LotShape	SaleCondition	BsmtQual	GarageType	LotFrontage_mode
0	65.0	Reg	Normal	Gd	Attchd	65.0
1	80.0	Reg	Normal	Gd	Attchd	80.0
2	68.0	IR1	Normal	Gd	Attchd	68.0
3	60.0	IR1	Abnorml	TA	Detchd	60.0
4	84.0	IR1	Normal	Gd	Attchd	84.0
5	85.0	IR1	Normal	Gd	Attchd	85.0
6	75.0	Reg	Normal	Ex	Attchd	75.0
7	NaN	IR1	Normal	Gd	Attchd	60.0
8	51.0	Reg	Abnorml	TA	Detchd	51.0
9	50.0	Reg	Normal	TA	Attchd	50.0
10	70.0	Reg	Normal	TA	Detchd	70.0
11	85.0	IR1	Partial	Ex	BuiltIn	85.0
12	NaN	IR2	Normal	TA	Detchd	60.0
13	91.0	IR1	Partial	Gd	Attchd	91.0
14	NaN	IR1	Normal	TA	Attchd	60.0
15	51.0	Reg	Normal	TA	Detchd	51.0
16	NaN	IR1	Normal	TA	Attchd	60.0
17	72.0	Reg	Normal	NaN	CarPort	72.0

	LotFrontage	LotShape	SaleCondition	BsmtQual	GarageType	LotFrontage_mode
18	66.0	Reg	Normal	TA	Detchd	66.0
19	70.0	Reg	Abnorml	TA	Attchd	70.0

In [41]:

```
housing2.BsmtQual.fillna('missing')
```

Out[41]:

```
0       Gd
1       Gd
2       Gd
3       TA
4       Gd
.
.
1
4
5
5

G
d
1
4
5
6

G
d
1457    TA
1458    TA
1459    TA
Name: BsmtQual, Length: 1460, dtype: object
```

In [42]:

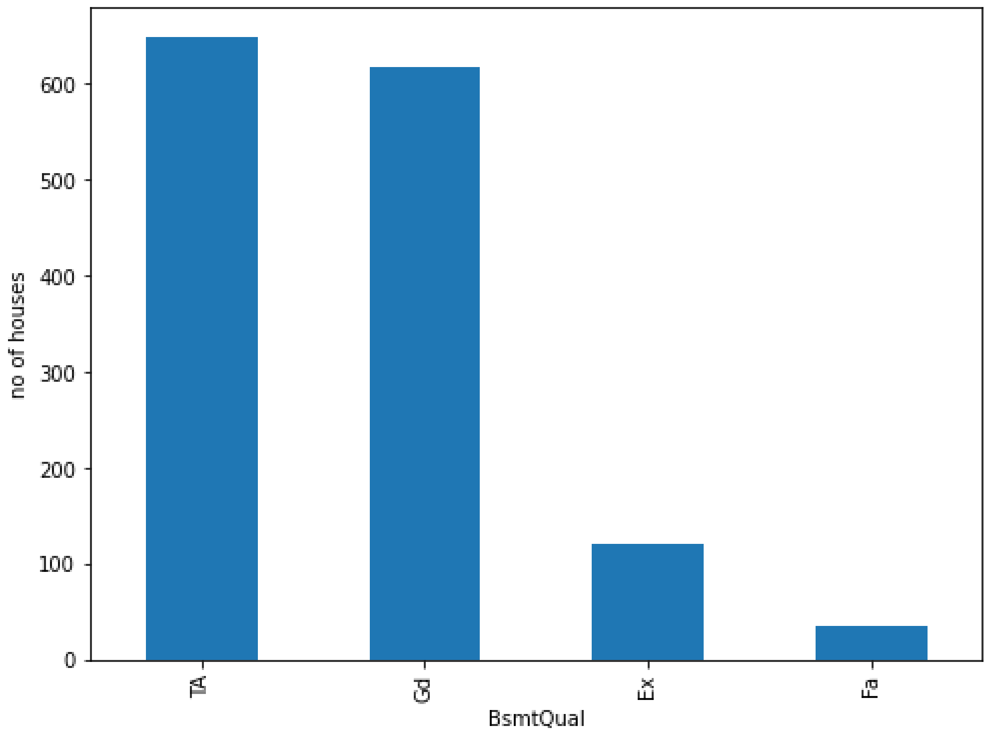
```
# plot bar graph
housing2.BsmtQual.value_count
  s().sort_values(ascending=Fal
  se).plot.bar()
  plt.xlabel("BsmtQual")
plt.ylabel("no of houses")
```

Out[42]:

In [43]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Text(0, 0.5, 'no of houses')



ENCODING CATEGORICAL DATA

Out[43]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

5 rows × 81 columns

In [44]:

```
# extracting CATEGORICAL VARIABLES ( sampling)

housing = df[['HouseStyle','GarageType','SaleType','LotFrontage']]
```

In [45]:

```
housing.head(20)
```

Out[45]:

	HouseStyle	GarageType	SaleType	LotFrontage
0	2Story	Attchd	WD	65.0
1	1Story	Attchd	WD	80.0
2	2Story	Attchd	WD	68.0
3	2Story	Detchd	WD	60.0
4	2Story	Attchd	WD	84.0
5	1.5Fin	Attchd	WD	85.0
6	1Story	Attchd	WD	75.0
7	2Story	Attchd	WD	NaN
8	1.5Fin	Detchd	WD	51.0
9	1.5Unf	Attchd	WD	50.0
10	1Story	Detchd	WD	70.0
11	2Story	BuiltIn	New	85.0
12	1Story	Detchd	WD	NaN
13	1Story	Attchd	New	91.0
14	1Story	Attchd	WD	NaN
15	1.5Unf	Detchd	WD	51.0

	HouseStyle	GarageType	SaleType	LotFrontage
16	1Story	Attchd	WD	NaN
17	1Story	CarPort	WD	72.0
18	1Story	Detchd	WD	66.0
19	1Story	Attchd	COD	70.0

In [46]:

```
# ENCODING CATEGORICAL DATA

# 1. One HOT Encoding
# 2. Frequency Encoding
# 3. Ordinal Encoding

# 1. ONE HOT ENCODING
# 1. Find the "Unique" values on each column
# 2. Find "Dummies" on each column

# Finding unique values on each categorical column
print(housing['HouseStyle'].unique())
print(housing['GarageType'].unique())
print(housing['SaleType'].unique())
```

['2Story', '1Story', '1.5Fin', '1.5Unf', 'S

In [47]:

```
Foyer' 'SLvl' '2.5Unf' '2.5Fin']
['Attchd' 'Detc
```

```
hd' 'BuiltIn' 'CarPort' nan 'Basment' '2Types']
['WD' 'New' 'COD' 'ConLD' 'ConLI' 'CWD' 'ConLw' 'Con' 'Oth']
```

Out[47]:

	1.5Fin	1.5Unf	1Story	2.5Fin	2.5Unf	2Story	SFoyer	SLvl
0	0	0	0	0	0	1	0	0
1	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	0	0
3	0	0	0	0	0	1	0	0
4	0	0	0	0	0	1	0	0

In [48]:

```
# priting actual and encoded values of column 'HouseStyle'
pd.concat([housing['HouseStyle'], pd.get_dummies(housing['HouseStyle'])], axis=1).head()
```

Out[48]:

	HouseStyle	1.5Fin	1.5Unf	1Story	2.5Fin	2.5Unf	2Story	SFoyer	SLvl
0	2Story	0	0	0	0	0	1	0	0
1	1Story	0	0	1	0	0	0	0	0
2	2Story	0	0	0	0	0	1	0	0

HouseStyle	1.5Fin	1.5Unf	1Story	2.5Fin	2.5Unf	2Story	SFoyer	SLvl
3 **2Story**	0	0	0	0	0	1	0	0
4 **2Story**	0	0	0	0	0	1	0	0

In [49]:

Out[49]:

	2Types	Attchd	Basment	BuiltIn	CarPort	Detchd
0	0	1	0	0	0	0
1	0	1	0	0	0	0
2	0	1	0	0	0	0
3	0	0	0	0	0	1
4	0	1	0	0	0	0

In [50]:

```
# priting actual and encoded values of column 'GarageType'
pd.concat([housing['GarageType'], pd.get_dummies(housing['GarageType'])], axis=1).head()
```

Out[50]:

GarageType	2Types	Attchd	Basment	BuiltIn	CarPort	Detchd
0 **Attchd**	0	1	0	0	0	0
1 **Attchd**	0	1	0	0	0	0
2 **Attchd**	0	1	0	0	0	0
3 **Detchd**	0	0	0	0	0	1
4 **Attchd**	0	1	0	0	0	0

In [51]:

```
# need N-1 columns in the one hot encoded dataset for a column that originally contained N unique labels.

temp =
  pd.get_dummies(h
  ousing['GarageTyp
  e'], drop_first =
  True) temp.head()
```

Out[51]:

	Attchd	Basment	BuiltIn	CarPort	Detchd
0	1	0	0	0	0
1	1	0	0	0	0
2	1	0	0	0	0
3	0	0	0	0	1
4	1	0	0	0	0

In [52]:

```
# Also, we can create one hot encoded column for null values in
```

```
# the actual
column by
passing True as
a value for the
dummy_na #
parameter.
temp =
pd.get_dummies(housing['GarageType']
, dummy_na = True ,drop_first =
True) temp.head()
```

Out[52]:

	Attchd	Basment	BuiltIn	CarPort	Detchd	NaN
0	1	0	0	0	0	0
1	1	0	0	0	0	0
2	1	0	0	0	0	0
3	0	0	0	0	1	0
4	1	0	0	0	0	0

In [53]:

```
[illegible]
```

Out[53]:

	COD	CWD	Con	ConLD	ConLI	ConLw	New	Oth	WD
0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	1
4	0	0	0	0	0	0	0	0	1

In [54]:

```
# priting actual and encoded values of column 'SaleType'

pd.concat([housing['GarageType'], pd.get_dummies(housing['SaleType'])], axis=1).head()
```

Out[54]:

	GarageType	COD	CWD	Con	ConLD	ConLI	ConLw	New	Oth	WD
0	Attchd	0	0	0	0	0	0	0	0	1
1	Attchd	0	0	0	0	0	0	0	0	1
2	Attchd	0	0	0	0	0	0	0	0	1
3	Detchd	0	0	0	0	0	0	0	0	1
4	Attchd	0	0	0	0	0	0	0	0	1

In [55]:

```
# Advantage
# No assumption about the dataset and all the categorical
values can be successfully encoded. #Disadvantage
# Feature space can become very large since a categorical column can have a lot of unique values.
```

```
# 2. FREQUENCY ENCODING
# Step1: Remove NULL/NA values on the categorical column.
# Step2: Call the value_counts() method on the categorical column, and then chain it with the to_dict() column to obtain the count for each
unique label in th # Step3 : Finally, call the map() method and pass it the dictionarycontaining the labels and count

housing.dropna(inplace = True)
```

<ipython-input-55-7f3dd51c37ed>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
housing.dropna(inplace = True)

In [56]:

```
value_counts =
  housing['GarageType'].value_
  counts(). to_dict()
  print(value_counts)
```

{'Attchd': 694, 'Detchd': 340, 'BuiltIn': 65, 'Basment': 15, 'CarPort': 8, '2Types': 5}

In [57]:

```
  # call the map()

housing['GarageType'] =
  housing['GarageType'].
  map(value_counts)
  housing.head()
```

<ipython-input-57-2cf6afff38c1>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame. Try using
.loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
housing['GarageType'] = housing['GarageType'].
map(value_counts)

Out[57]:

	HouseStyle	GarageType	SaleType	LotFrontage
0	2Story	694	WD	65.0
1	1Story	694	WD	80.0
2	2Story	694	WD	68.0
3	2Story	340	WD	60.0
4	2Story	694	WD	84.0

In [58]:

```
  # we can also add percentage frequency by dividing the label count by the total number of rows as follows:

  import pandas
  import numpy
frequency_count =
  (housing['GarageType'].value_counts() /
  len('GarageType') ).to_dict()
  print(frequency_count)

housing['GarageType'] =
  housing['GarageType'].
  map(frequency_count)
  housing.head()
```

{694: 69.4, 340: 34.0, 65: 6.5, 15: 1.5, 8: 0.8, 5: 0.5}

<ipython-input-58-45aff1cd509b>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame. Try using
.loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
housing['GarageType'] = housing['GarageType'].map(frequency_count)

Out[58]:

	HouseStyle	GarageType	SaleType	LotFrontage
0	2Story	69.4	WD	65.0
1	1Story	69.4	WD	80.0
2	2Story	69.4	WD	68.0
3	2Story	34.0	WD	60.0
4	2Story	69.4	WD	84.0

In [59]:

```
# ORDINAL ENCODING


housing=
  housing[["HouseStyle",
  "GarageType",
  "SaleType","LotFrontage"]]
  housing.groupby(['GarageTyp
  e'])['LotFrontage'].mean().sort
  _values()
```

Out[59]:

```
GarageType
34.0      59.897059
0.5       68.600000
69.4      74.923631
0.8       76.750000
6.5       79.076923
1.5       80.066667
Name:  LotFrontage, dtype: float64
```

In [60]:

```
  ordered_cats  =
  housing.groupby(['GarageType'])['LotFrontage'].me
  an().sort_values().index cat_map= {k: i for i, k in
  enumerate(ordered_cats, 0)} # Dictionary creation:
housing['GarageType_or
  dered'] =
  housing['GarageType']
  .map(cat_map)
  housing.head()
```

Out[60]:

	HouseStyle	GarageType	SaleType	LotFrontage	GarageType_ordered
0	2Story	69.4	WD	65.0	2
1	1Story	69.4	WD	80.0	2
2	2Story	69.4	WD	68.0	2
3	2Story	34.0	WD	60.0	0
4	2Story	69.4	WD	84.0	2

In [61]:

```
  # Mean Encoding

  # Step1: Identify
  the column which
  you want apply
  mean encoding #
  Step2: Calculate
  the mean on
  categorical
  column.
  # Step3: Use to_dict() method to convert the dataframe into dictionary
  # Step4: Use map() method to transform and add the new mean column to the original dataset.


housing.groupby(['GarageType'])['LotFrontage'].mean()
```

Out[61]:

GarageType

```
0.5     68.600000
0.8     76.750000
1.5     80.066667
6.5     79.076923
34.0    59.897059
69.4    74.923631
Name:  LotFrontage, dtype: float64
```

In [62]:

```
mean_labels =
  housing.groupby(['GarageType'])['Lot
  Frontage']. mean().to_dict()
  housing['GarageType_mean'] =
  housing['GarageType'].map(mean_lab
  els)
housing.head()
```

Out[62]:

	HouseStyle	GarageType	SaleType	LotFrontage	GarageType_ordered	GarageType_mean
0	2Story	69.4	WD	65.0	2	74.923631
1	1Story	69.4	WD	80.0	2	74.923631
2	2Story	69.4	WD	68.0	2	74.923631
3	2Story	34.0	WD	60.0	0	59.897059
4	2Story	69.4	WD	84.0	2	74.923631

Data Discretization

In [63]:

```
  # How  to convert continuous numeric values into discrete intervals. # The  process of converting continuous numeric values into discrete intervals is called
  discretization or binning.          # Examples: price, age, weight, etc. # Advantages:
  # Helpful to handle Outliers. # With discretization, the outliers can be placed into tail intervals along with the remaining inlier values that occur at tails. # Discretization is particularly helpful

  # 1) Equal width Discretiation
  # Step1: Select a column to apply discretization. # Step2: Plot a histogram for the column. Histogram shows the distribution of data (normal / skewed) # Step3: Find the total value of the
  column by sub         # The value will be rounded off to the nearest integer value.

  import matplotlib.pyplot as plt
  import seaborn as sns
  import pandas as pd
  import numpy as np

df
  =
  pd.read_csv('E:\data preprocessing
```

Out[63]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

5 rows × 81 columns

In [64]:

```python
# selection subdata set to work with

housing =
df[['LotFrontage','LotArea','LotShape','SalePrice','YrSold','SaleCondition','BsmtQual','GarageType','GarageArea','FireplaceQu','OverallQual']].
copy() housing.head(20)
```

Out[64]:

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu	OverallQual
0	65.0	8450	Reg	208500	2008	Normal	Gd	Attchd	548	NaN	7
1	80.0	9600	Reg	181500	2007	Normal	Gd	Attchd	460	TA	6
2	68.0	11250	IR1	223500	2008	Normal	Gd	Attchd	608	TA	7
3	60.0	9550	IR1	140000	2006	Abnorml	TA	Detchd	642	Gd	7
4	84.0	14260	IR1	250000	2008	Normal	Gd	Attchd	836	TA	8
5	85.0	14115	IR1	143000	2009	Normal	Gd	Attchd	480	NaN	5
6	75.0	10084	Reg	307000	2007	Normal	Ex	Attchd	636	Gd	8
7	NaN	10382	IR1	200000	2009	Normal	Gd	Attchd	484	TA	7
8	51.0	6120	Reg	129900	2008	Abnorml	TA	Detchd	468	TA	7
9	50.0	7420	Reg	118000	2008	Normal	TA	Attchd	205	TA	5
10	70.0	11200	Reg	129500	2008	Normal	TA	Detchd	384	NaN	5
11	85.0	11924	IR1	345000	2006	Partial	Ex	BuiltIn	736	Gd	9
12	NaN	12968	IR2	144000	2008	Normal	TA	Detchd	352	NaN	5
13	91.0	10652	IR1	279500	2007	Partial	Gd	Attchd	840	Gd	7
14	NaN	10920	IR1	157000	2008	Normal	TA	Attchd	352	Fa	6
15	51.0	6120	Reg	132000	2007	Normal	TA	Detchd	576	NaN	7
16	NaN	11241	IR1	149000	2010	Normal	TA	Attchd	480	TA	6
17	72.0	10791	Reg	90000	2006	Normal	NaN	CarPort	516	NaN	4
18	66.0	13695	Reg	159000	2008	Normal	TA	Detchd	576	NaN	5
19	70.0	7560	Reg	139000	2009	Abnorml	TA	Attchd	294	NaN	5

In [65]:

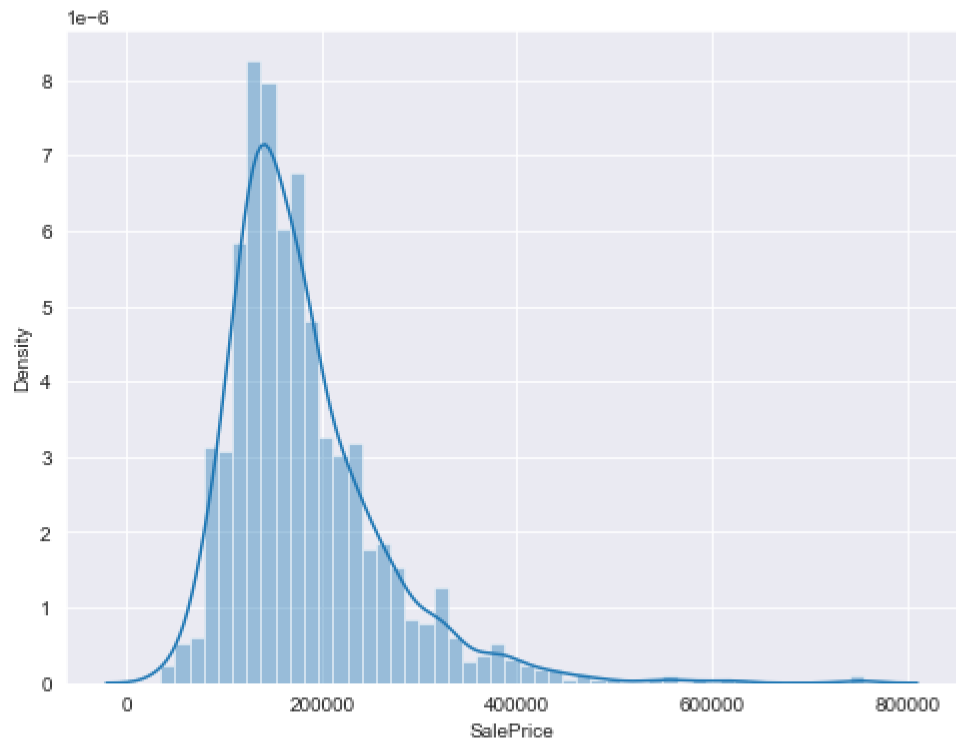
```python
# STEP1: Plot a histogram for the price column

sns.distplot(housing['SalePrice'])
```

c:\users\asus\appdata\local\programs\python\python39\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please a dapt 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)

Out[65]:

<AxesSubplot:xlabel='SalePrice', ylabel='Density'>



In [66]:

```
# STEP2: find the total price range by subtracting the minimum price from the maximum price.
price_range =
  housing['SalePrice'].max()
  -
  housing['SalePrice'].min()
  print(price_range )
```

720100

In [67]:

```
# STEP3: find the length or width of each interval, we simply need to divide the price by the number of intervals.
price_range / 10
```

Out[67]:

72010.0

In [68]:

```
[illegible]
```

34900
755000
72010

In [69]:

```
# Data Discretization
# STEP 5: Next, Let's create the 10 bins for our dataset. To create bins,
# we will start with the minimum value, and add the bin interval or # Length to it. To get the se

total_bins = [i for i in range(lower_interval,
  upper_interval+interval_length, interval_length)]
  print(total_bins)
```

[34900, 106910, 178920, 250930, 322940, 394950, 466960, 538970, 610980, 682990, 755000]

In [70]:

```python
# STEP 6: Next, we will create string Labels for each bin. You can give any name to the bin Labels.

bin_labels = ["Bin_no_"
+ str(i) for i in
range(1,
len(total_bins))]
print(bin_labels)
```

['Bin_no_1', 'Bin_no_2', 'Bin_no_3', 'Bin_no_4', 'Bin_no_5', 'Bin_no_6', 'Bin_no_7', 'Bin_no_8', 'Bin_no_9', 'Bin_no_10']

In [71]:

```python
# STEP 7: You can create
the Pandas Libraries "cut()"
method to convert # the
continuous column values
to numeric bin values.
# You need     to pass the data column that you want to be discretized, along # with the bin intervals and the bin Labels.

housing['SalePrice_bins'] = pd.cut(x=housing['SalePrice'],bins=total_bins,
 labels=bin_labels, include_lowest=True) housing.head(20)
```

Out[71]:

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu	OverallQual	SalePrice_bins
0	65.0	8450	Reg	208500	2008	Normal	Gd	Attchd	548	NaN	7	Bin_no_3
1	80.0	9600	Reg	181500	2007	Normal	Gd	Attchd	460	TA	6	Bin_no_3
2	68.0	11250	IR1	223500	2008	Normal	Gd	Attchd	608	TA	7	Bin_no_3
3	60.0	9550	IR1	140000	2006	Abnorml	TA	Detchd	642	Gd	7	Bin_no_2
4	84.0	14260	IR1	250000	2008	Normal	Gd	Attchd	836	TA	8	Bin_no_3
5	85.0	14115	IR1	143000	2009	Normal	Gd	Attchd	480	NaN	5	Bin_no_2
6	75.0	10084	Reg	307000	2007	Normal	Ex	Attchd	636	Gd	8	Bin_no_4
7	NaN	10382	IR1	200000	2009	Normal	Gd	Attchd	484	TA	7	Bin_no_3
8	51.0	6120	Reg	129900	2008	Abnorml	TA	Detchd	468	TA	7	Bin_no_2
9	50.0	7420	Reg	118000	2008	Normal	TA	Attchd	205	TA	5	Bin_no_2
10	70.0	11200	Reg	129500	2008	Normal	TA	Detchd	384	NaN	5	Bin_no_2
11	85.0	11924	IR1	345000	2006	Partial	Ex	BuiltIn	736	Gd	9	Bin_no_5
12	NaN	12968	IR2	144000	2008	Normal	TA	Detchd	352	NaN	5	Bin_no_2
13	91.0	10652	IR1	279500	2007	Partial	Gd	Attchd	840	Gd	7	Bin_no_4
14	NaN	10920	IR1	157000	2008	Normal	TA	Attchd	352	Fa	6	Bin_no_2
15	51.0	6120	Reg	132000	2007	Normal	TA	Detchd	576	NaN	7	Bin_no_2
16	NaN	11241	IR1	149000	2010	Normal	TA	Attchd	480	TA	6	Bin_no_2
17	72.0	10791	Reg	90000	2006	Normal	NaN	CarPort	516	NaN	4	Bin_no_1
18	66.0	13695	Reg	159000	2008	Normal	TA	Detchd	576	NaN	5	Bin_no_2
19	70.0	7560	Reg	139000	2009	Abnorml	TA	Attchd	294	NaN	5	Bin_no_2

In [72]:

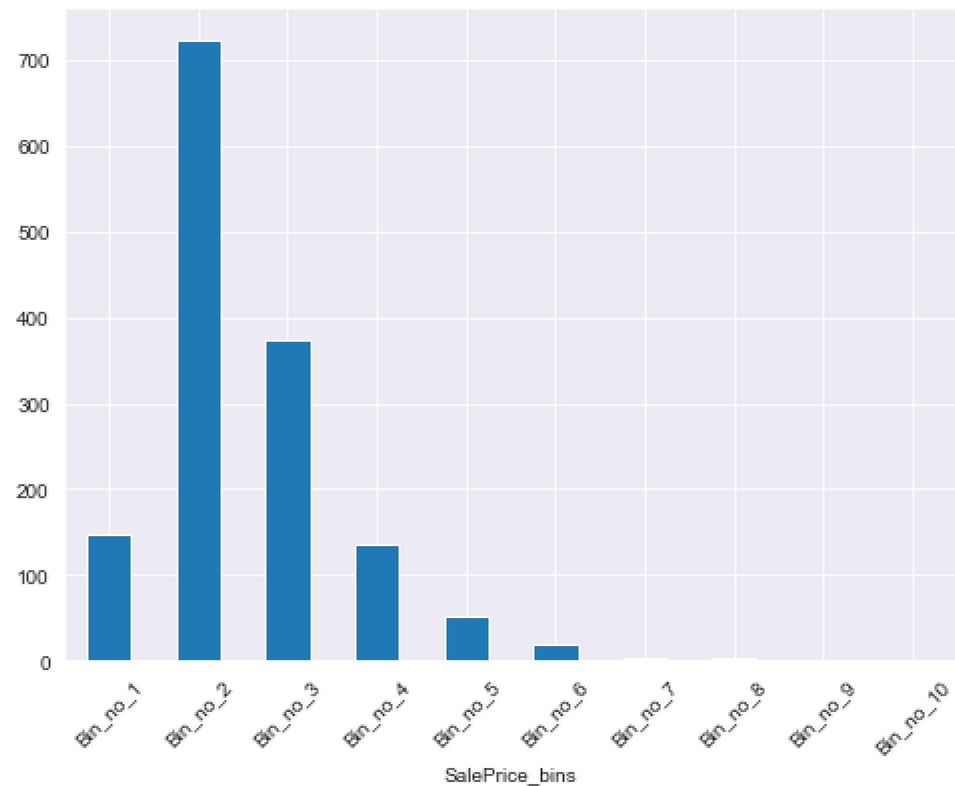
```python
# STEP 8: Next, let's plot a bar plot that shows the frequency of prices in each bin.

housing.groupby('SalePrice_bin
 s')['SalePrice'].count().plot.bar
 () plt.xticks(rotation=45)
```

Out[72]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
```

[Text(0, 0, 'Bin_no_1'),
 Text(1, 0, 'Bin_no_2'),
 Text(2, 0, 'Bin_no_3'),
 Text(3, 0, 'Bin_no_4'),
 Text(4, 0, 'Bin_no_5'),
 Text(5, 0, 'Bin_no_6'),
 Text(6, 0, 'Bin_no_7'),
 Text(7, 0, 'Bin_no_8'),
 Text(8, 0, 'Bin_no_9'),
 Text(9, 0, 'Bin_no_10')])



In [73]:

```
# 2. EQUAL FREQUENCY DISCRETIZATION

#  In equal frequency discretization, the bin width is
adjusted automatically in such  a  way  that     # each bin
contains exactly the same number of records or has the
same frequency.
#  In equal frequency
discretization, the bin interval
may  not be  the same.   # Is a
unsupervised discretization
technique.
# Let's apply equal frequency discretization on the price column of the Diamonds dataset

# STEP1: Load the dataset. # STEP2: Select a column.
# STEP3: Use qcut() method to convert a continuous column into equal frequency discretized bins.
# The qcut() function returns quartiles, equal to the number of specified intervals along with the bins.
# You have to pass the dataset column, the number of intervals, the Labels as mandatory
parameters for the "qcut()" function.          # Step4: Create a dataframe that shows the
actual values and quartile information.

# STEP5: print bins
# STEP6: Next, find the number of records per bin.
# Step7: Create a Pandas dataframe containing the bins

# STEP4: Print bins and quartile
discretised_price, bins = pd.qcut(housing['SalePrice'], 10, labels=None, retbins=True,
  precision=3, duplicates='raise') pd.concat([discretised_price, housing['SalePrice']],
  axis=1).head(10)
```

Out[73]:

SalePrice SalePrice

In [74]:

```
[ 34900. 106475. 124000. 135500. 147000. 163000. 179280. 198620. 230000.
  278000. 755000.]
<class 'numpy.ndarray'>
```

In [75]:

```
# STEP6: find the number of records per bin.
discretised_price.value_counts()
```

Out[75]:

```
 (135500.0,   147000.0]      150
 (106475.0,   124000.0]      149
 (198620.0,   230000.0]      149
(34899.999, 106475.0]        146
(179280.0,    198620.0]      146
(278000.0,    755000.0]      145
(124000.0,    135500.0]      144
(163000.0,    179280.0]      144
(230000.0,    278000.0]      144
(147000.0,    163000.0]      143
Name: SalePrice, dtype: int64
```

In [76]:

```
[illegible]
```

```
['Bin_no_1', 'Bin_no_2', 'Bin_no_3', 'Bin_no_4', 'Bin_no_5', 'Bin_no_6', 'Bin_no_7', 'Bin_no_8', 'Bin_no_9', 'Bin_no_10']
```

In [77]:

```
# print dataset
housing['SalePrice_bins'] = pd.cut(x=housing['SalePrice'],bins=bins, labels=bin_labels,
  include_lowest=True) housing.head(20)
```

Out[77]:

LotFrontage LotArea LotShape SalePrice YrSold SaleCondition BsmtQual GarageType GarageArea FireplaceQu OverallQual SalePrice_bins

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu	OverallQual	SalePrice_bins
0	65.0	8450	Reg	208500	2008	Normal	Gd	Attchd	548	NaN	7	Bin_no_8
1	80.0	9600	Reg	181500	2007	Normal	Gd	Attchd	460	TA	6	Bin_no_7
2	68.0	11250	IR1	223500	2008	Normal	Gd	Attchd	608	TA	7	Bin_no_8
3	60.0	9550	IR1	140000	2006	Abnorml	TA	Detchd	642	Gd	7	Bin_no_4
4	84.0	14260	IR1	250000	2008	Normal	Gd	Attchd	836	TA	8	Bin_no_9
5	85.0	14115	IR1	143000	2009	Normal	Gd	Attchd	480	NaN	5	Bin_no_4
6	75.0	10084	Reg	307000	2007	Normal	Ex	Attchd	636	Gd	8	Bin_no_10
7	NaN	10382	IR1	200000	2009	Normal	Gd	Attchd	484	TA	7	Bin_no_8
8	51.0	6120	Reg	129900	2008	Abnorml	TA	Detchd	468	TA	7	Bin_no_3
9	50.0	7420	Reg	118000	2008	Normal	TA	Attchd	205	TA	5	Bin_no_2
10	70.0	11200	Reg	129500	2008	Normal	TA	Detchd	384	NaN	5	Bin_no_3
11	85.0	11924	IR1	345000	2006	Partial	Ex	BuiltIn	736	Gd	9	Bin_no_10
12	NaN	12968	IR2	144000	2008	Normal	TA	Detchd	352	NaN	5	Bin_no_4
13	91.0	10652	IR1	279500	2007	Partial	Gd	Attchd	840	Gd	7	Bin_no_10
14	NaN	10920	IR1	157000	2008	Normal	TA	Attchd	352	Fa	6	Bin_no_5
15	51.0	6120	Reg	132000	2007	Normal	TA	Detchd	576	NaN	7	Bin_no_3
16	NaN	11241	IR1	149000	2010	Normal	TA	Attchd	480	TA	6	Bin_no_5
17	72.0	10791	Reg	90000	2006	Normal	NaN	CarPort	516	NaN	4	Bin_no_1
18	66.0	13695	Reg	159000	2008	Normal	TA	Detchd	576	NaN	5	Bin_no_5
19	70.0	7560	Reg	139000	2009	Abnorml	TA	Attchd	294	NaN	5	Bin_no_4

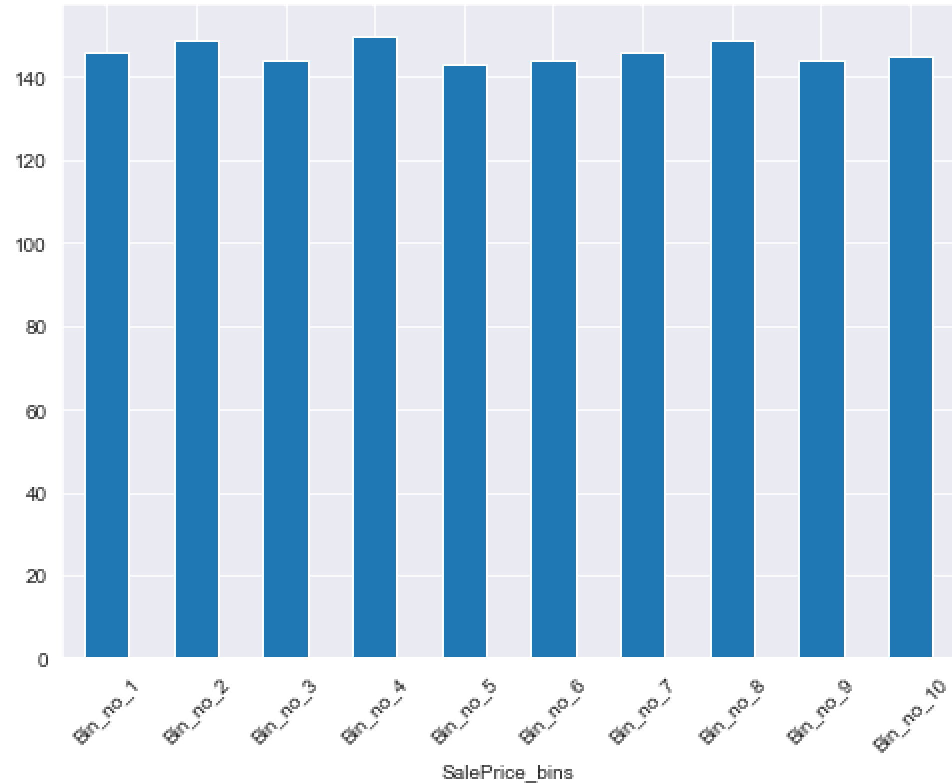
In [78]:

```
# Bar plot
housing.groupby('SalePrice_b
ins')['SalePrice'].count().plot.b
ar() plt.xticks (rotation=45)
```

Out[78]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),

[Text(0,  0,  'Bin_no_1'),
 Text(1,  0,  'Bin_no_2'),
 Text(2,  0,  'Bin_no_3'),
 Text(3,  0,  'Bin_no_4'),
 Text(4,  0,  'Bin_no_5'),
 Text(5,  0,  'Bin_no_6'),
 Text(6,  0,  'Bin_no_7'),
 Text(7,  0,  'Bin_no_8'),
 Text(8,  0,  'Bin_no_9'),
 Text(9,  0,  'Bin_no_10')])
```



	SalePrice	SalePrice
0	(198620.0, 230000.0]	208500
1	(179280.0, 198620.0]	181500
2	(198620.0, 230000.0]	223500
3	(135500.0, 147000.0]	140000
4	(230000.0, 278000.0]	250000
5	(135500.0, 147000.0]	143000
6	(278000.0, 755000.0]	307000
7	(198620.0, 230000.0]	200000
8	(124000.0, 135500.0]	129900
9	(106475.0, 124000.0]	118000

```
# STEP5: Print bins

print(bins)
print(type (bins))
```

In [79]:

3. K-Means Discretization

 # K-means discretization is another unsupervised discretization technique based on the K-means algorithm. # A brief description of the K-Means algorithm is given below:
 # 1. In the beginning, K random clusters of data points are created, where K is the number of bins or intervals. # 2. Each data point is linked to the

Out[79]:

```
KBinsDiscretizer(encode='ordinal', n_bins=10, strategy='kmeans')
```

In [80]:

```
# Use "bin_edges" attribute to access the bins created via K-means clustering
```

In [81]:

```
[array([ 34900.        , 117054.22907167, 160410.274379  , 207515.23665734,
        262815.94007217, 329335.04116968, 403199.20516304, 489155.78125     ,
        572192.25      , 678265.        , 755000.        ])]
```

In [82]:

['Bin_no_1', 'Bin_no_2', 'Bin_no_3', 'Bin_no_4', 'Bin_no_5', 'Bin_no_6', 'Bin_no_7', 'Bin_no_8', 'Bin_no_9', 'Bin_no_10']

In [83]:

```
cut_bins =[ 34900., 117054.22907167, 160410.274379 , 207515.23665734,
262815.94007217, 329335.04116968, 403199.20516304, 489155.78125 ,
572192.25 , 678265. , 755000. ]
```

In [84]:

```
housing['SalePrice_bins'] = pd.cut(x=housing['SalePrice'], bins=cut_bins,
 labels=bin_labels, include_lowest=True) housing.head(20)
```

Out[84]:

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu	OverallQual	SalePrice_bins
0	65.0	8450	Reg	208500	2008	Normal	Gd	Attchd	548	NaN	7	Bin_no_4
1	80.0	9600	Reg	181500	2007	Normal	Gd	Attchd	460	TA	6	Bin_no_3
2	68.0	11250	IR1	223500	2008	Normal	Gd	Attchd	608	TA	7	Bin_no_4
3	60.0	9550	IR1	140000	2006	Abnorml	TA	Detchd	642	Gd	7	Bin_no_2
4	84.0	14260	IR1	250000	2008	Normal	Gd	Attchd	836	TA	8	Bin_no_4
5	85.0	14115	IR1	143000	2009	Normal	Gd	Attchd	480	NaN	5	Bin_no_2
6	75.0	10084	Reg	307000	2007	Normal	Ex	Attchd	636	Gd	8	Bin_no_5
7	NaN	10382	IR1	200000	2009	Normal	Gd	Attchd	484	TA	7	Bin_no_3
8	51.0	6120	Reg	129900	2008	Abnorml	TA	Detchd	468	TA	7	Bin_no_2
9	50.0	7420	Reg	118000	2008	Normal	TA	Attchd	205	TA	5	Bin_no_2
10	70.0	11200	Reg	129500	2008	Normal	TA	Detchd	384	NaN	5	Bin_no_2
11	85.0	11924	IR1	345000	2006	Partial	Ex	BuiltIn	736	Gd	9	Bin_no_6
12	NaN	12968	IR2	144000	2008	Normal	TA	Detchd	352	NaN	5	Bin_no_2
13	91.0	10652	IR1	279500	2007	Partial	Gd	Attchd	840	Gd	7	Bin_no_5
14	NaN	10920	IR1	157000	2008	Normal	TA	Attchd	352	Fa	6	Bin_no_2
15	51.0	6120	Reg	132000	2007	Normal	TA	Detchd	576	NaN	7	Bin_no_2
16	NaN	11241	IR1	149000	2010	Normal	TA	Attchd	480	TA	6	Bin_no_2
17	72.0	10791	Reg	90000	2006	Normal	NaN	CarPort	516	NaN	4	Bin_no_1
18	66.0	13695	Reg	159000	2008	Normal	TA	Detchd	576	NaN	5	Bin_no_2
19	70.0	7560	Reg	139000	2009	Abnorml	TA	Attchd	294	NaN	5	Bin_no_2

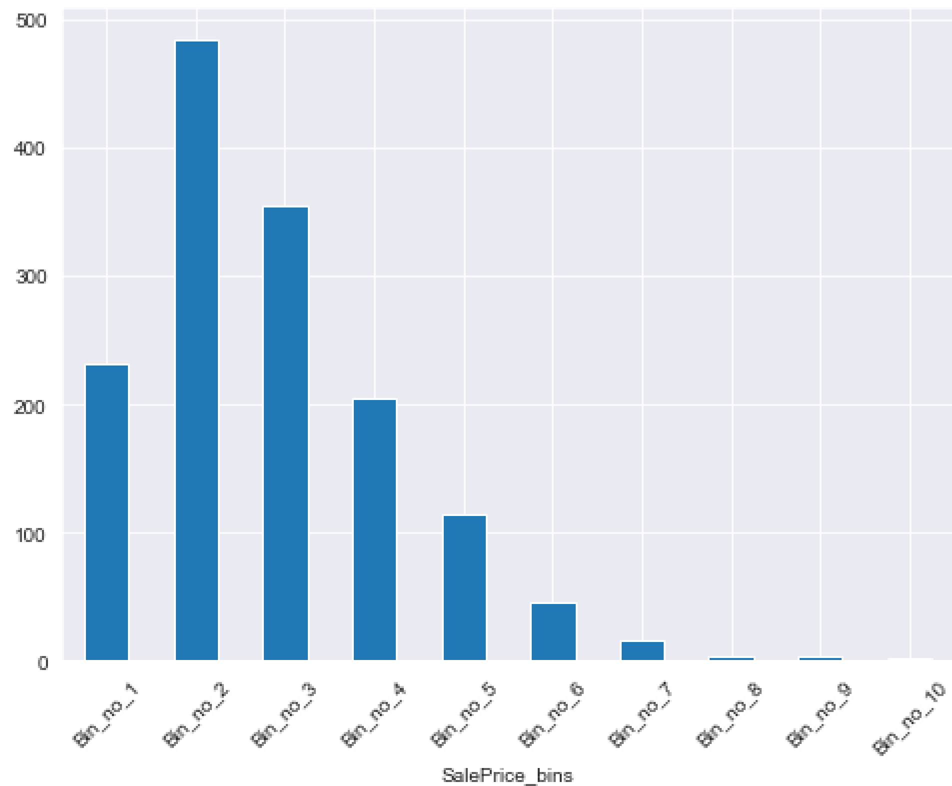
In [85]:

```
housing.groupby('SalePrice_bin
 s')['SalePrice'].count().plot.bar
 () plt.xticks(rotation=45)
```

Out[85]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 [Text(0, 0, 'Bin_no_1'),
```

```
Text(1,    0,  'Bin_no_2'),
Text(2,    0,  'Bin_no_3'),
Text(3,    0,  'Bin_no_4'),
Text(4,    0,  'Bin_no_5'),
Text(5,    0,  'Bin_no_6'),
Text(6,    0,  'Bin_no_7'),
Text(7,    0,  'Bin_no_8'),
Text(8,    0,  'Bin_no_9'),
Text(9,    0,  'Bin_no_10')])
```



In [86]:

```
# 4. Decision Tree Discretization # Decision tree discretization is a type of supervised discretization algorithm.

#  In decision tree discretization, bins are created
based  on  the values in some  other columns.   # In
decision tree discretization, NO  NEED to specify the
number  of bins or intervals.
# The decision tree identifies the optimal number of bins

# IMPORT DECISION TREE CLASSIFIER from SKLEARN
# To implement decision tree discretization, you can use the "DecisionTreeClassifier class from the "sklearn.tree" module.

from sklearn.tree import DecisionTreeClassifier

# Now  call the "fit()" method on the class and pass the continuous
column name  and the column on the basis of        # which you want to
create your bins.

tree_model = DecisionTreeClassifier(max_depth=3)
tree_model.fit(housing['SalePrice'].to_frame(), housing['OverallQual'])
housing['SalePrice_tree']=
  tree_model.predict_proba(housing['SalePrice'].to_fr
  ame())[:,1] housing.head()
```

Out[86]:

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu	OverallQual	SalePrice_bins	SalePrice_tree
0	65.0	8450	Reg	208500	2008	Normal	Gd	Attchd	548	NaN	7	Bin_no_4	0.0
1	80.0	9600	Reg	181500	2007	Normal	Gd	Attchd	460	TA			

6	Bin_no_3	0.0

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu	OverallQual	SalePrice_bins	SalePrice_tree
2	68.0	11250	IR1	223500	2008	Normal	Gd	Attchd	608	TA	7	Bin_no_4	0.0
3	60.0	9550	IR1	140000	2006	Abnorml	TA	Detchd	642	Gd	7	Bin_no_2	0.0
4	84.0	14260	IR1	250000	2008	Normal	Gd	Attchd	836	TA	8	Bin_no_4	0.0

In [87]:

```
# Now find the unique probability values in the price_tree column

housing['SalePrice_tree'].unique()
```

Out[87]:

```
array([0.        , 0.01818182])
```

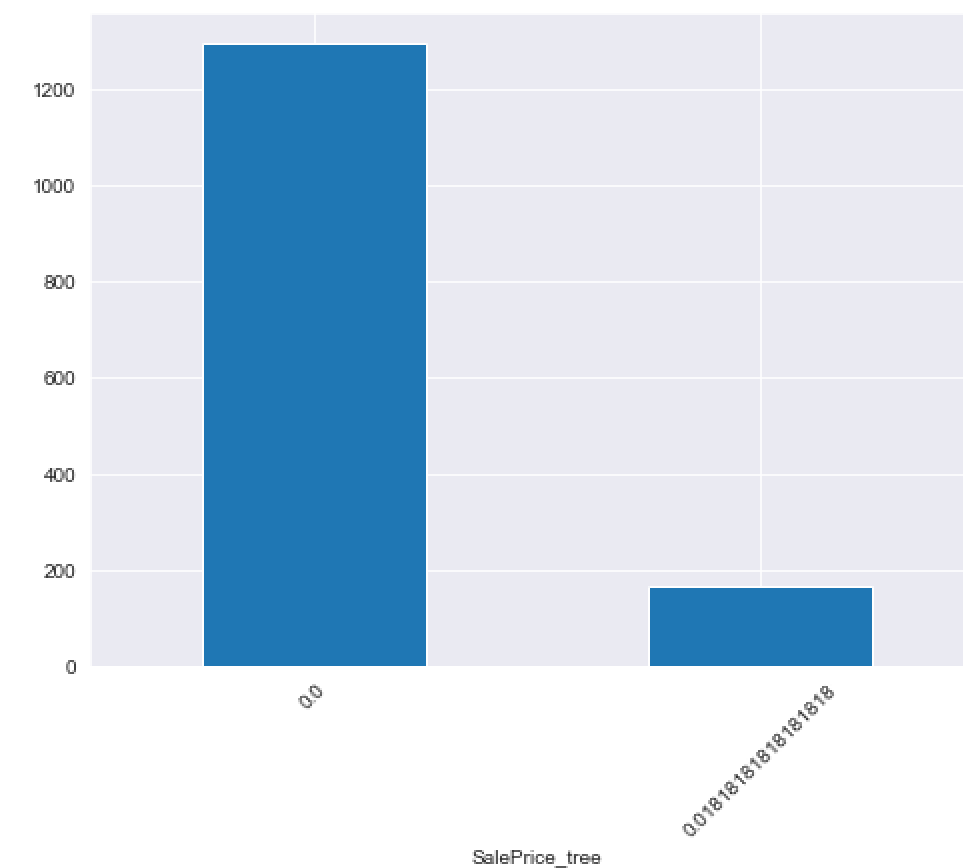
In [88]:

```
# Plot the frequency of records per bin

housing.groupby(['SalePrice_tre
  e'])['SalePrice'].count().plot.ba
  r() plt.xticks (rotation=45)
```

Out[88]:

```
(array([0, 1]), [Text(0, 0, '0.0'), Text(1, 0, '0.018181818181818')])
```



Outlier Handling

In [ ]:

```
# Housing dataset Loaded with Seaborn package importing
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
import pandas as pd
import numpy as np
```

In [129...

Out[129...

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

5 rows × 81 columns

In [131...

```python
housing=df[['LotFrontage','LotArea','LotShape','SalePrice','YrSold','SaleCondition','BsmtQual','GarageType','GarageArea','FireplaceQu']].copy()
  housing.head(20)
```

Out[131...

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu
0	65.0	8450	Reg	208500	2008	Normal	Gd	Attchd	548	NaN
1	80.0	9600	Reg	181500	2007	Normal	Gd	Attchd	460	TA
2	68.0	11250	IR1	223500	2008	Normal	Gd	Attchd	608	TA
3	60.0	9550	IR1	140000	2006	Abnorml	TA	Detchd	642	Gd
4	84.0	14260	IR1	250000	2008	Normal	Gd	Attchd	836	TA
5	85.0	14115	IR1	143000	2009	Normal	Gd	Attchd	480	NaN
6	75.0	10084	Reg	307000	2007	Normal	Ex	Attchd	636	Gd
7	NaN	10382	IR1	200000	2009	Normal	Gd	Attchd	484	TA
8	51.0	6120	Reg	129900	2008	Abnorml	TA	Detchd	468	TA
9	50.0	7420	Reg	118000	2008	Normal	TA	Attchd	205	TA
10	70.0	11200	Reg	129500	2008	Normal	TA	Detchd	384	NaN
11	85.0	11924	IR1	345000	2006	Partial	Ex	BuiltIn	736	Gd
12	NaN	12968	IR2	144000	2008	Normal	TA	Detchd	352	NaN
13	91.0	10652	IR1	279500	2007	Partial	Gd	Attchd	840	Gd
14	NaN	10920	IR1	157000	2008	Normal	TA	Attchd	352	Fa
15	51.0	6120	Reg	132000	2007	Normal	TA	Detchd	576	NaN
16	NaN	11241	IR1	149000	2010	Normal	TA	Attchd	480	TA
17	72.0	10791	Reg	90000	2006	Normal	NaN	CarPort	516	NaN

18	66.0	13695	Reg	159000	2008	Normal	TA	Detchd	576	NaN

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu
19	70.0	7560	Reg	139000	2009	Abnorml	TA	Attchd	294	NaN

In [132…

```
housing[30:40]
```

Out[132…

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu
30	50.0	8500	Reg	40000	2008	Normal	TA	Detchd	250	NaN
31	NaN	8544	IR1	149350	2008	Normal	TA	Attchd	271	NaN
32	85.0	11049	Reg	179900	2008	Normal	Ex	Attchd	484	NaN
33	70.0	10552	IR1	165500	2010	Normal	TA	Attchd	447	Gd
34	60.0	7313	Reg	277500	2007	Normal	Ex	Attchd	556	Gd
35	108.0	13418	Reg	309000	2006	Normal	Ex	BuiltIn	691	Gd
36	112.0	10859	Reg	145000	2009	Normal	Gd	Attchd	672	NaN
37	74.0	8532	Reg	153000	2009	Normal	TA	Attchd	498	TA
38	68.0	7922	Reg	109000	2010	Abnorml	TA	Detchd	246	NaN
39	65.0	6040	Reg	82000	2008	AdjLand	NaN	NaN	0	NaN

In [ ]:

```
# Plot box plot for LotFrontage after removing outliers
```

In [133…

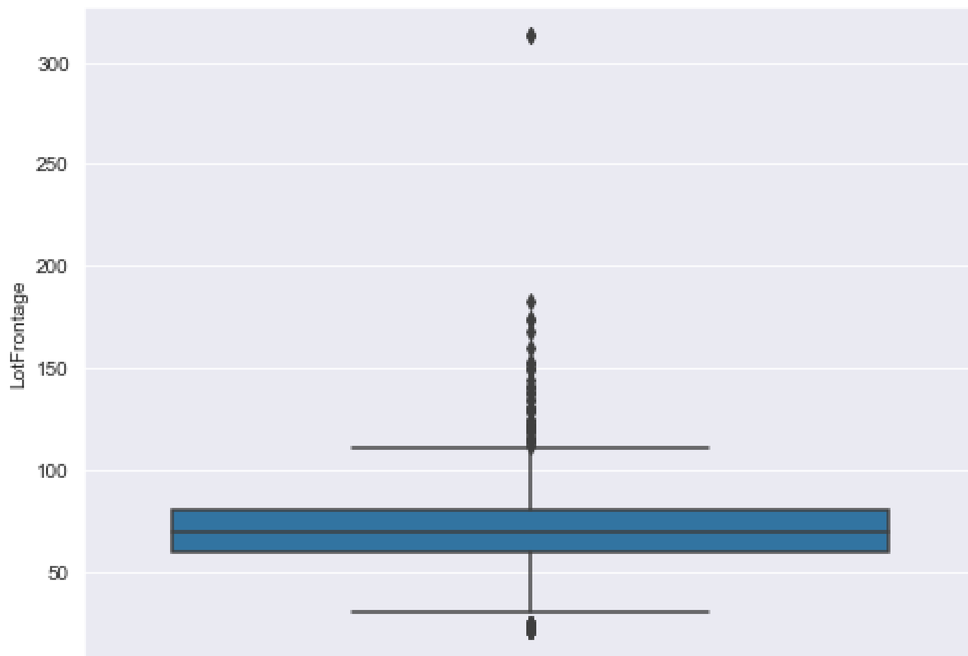
```
sns.boxplot(y="LotFrontage",data=housing)
```

Out[133…

In [134…

```
# use the IQR method to find the upper and lower Limits
```

<AxesSubplot:ylabel='LotFrontage'>



```
# to find the outliers in the LotFrontage column.

IQR = housing["LotFrontage"].quantile(0.75) -
housing["LotFrontage"].quantile(0.25) lower_LotFrontage_limit =
housing["LotFrontage"].quantile(0.25) - (IQR * 1.5)
upper_LotFrontage_limit =
housing["LotFrontage"].quantile(0.75) +
(IQR * 1.5)
print(lower_LotFrontage_limit)
print(upper_LotFrontage_limit)
```

```
27.5
111.5
```

In [137…

```
# Finding outlier values in SalePrice column

LotFrontage_outliers = np.where(housing["LotFrontage"] > upper_LotFrontage_limit, True,
```

In [138…

```
LotFrontage_outliers[1:40]
```

Out[138…

```
array([False, False, False, False, False, False,
```

In [139…

```
# 1. OUTERLIER TRIMMING (Removing the outlier values in LotFrontage column)

housing_without_LotFrontage_outliers
  =
  housing.loc[~(LotFrontage_outliers),
  ] housing.shape,
  housing_without_LotFrontage_outlie
  rs.shape
```

False,

False,False,False,False,False,False,

False,False,True,False,False,False])

Out[139… In [140…

((1460, 10), (1372, 10))

```
housing_without_LotFrontage_outliers[30:40]
```

Out[140…

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu
30	50.0	8500	Reg	40000	2008	Normal	TA	Detchd	250	NaN
31	NaN	8544	IR1	149350	2008	Normal	TA	Attchd	271	NaN
32	85.0	11049	Reg	179900	2008	Normal	Ex	Attchd	484	NaN
33	70.0	10552	IR1	165500	2010	Normal	TA	Attchd	447	Gd
34	60.0	7313	Reg	277500	2007	Normal	Ex	Attchd	556	Gd
35	108.0	13418	Reg	309000	2006	Normal	Ex	BuiltIn	691	Gd
37	74.0	8532	Reg	153000	2009	Normal	TA	Attchd	498	TA

38	68.0	7922	Reg	109000	2010	Abnorml	TA	Detchd	246	NaN
39	65.0	6040	Reg	82000	2008	AdjLand	NaN	NaN	0	NaN
40	84.0	8658	Reg	160000	2006	Abnorml	TA	Attchd	440	TA

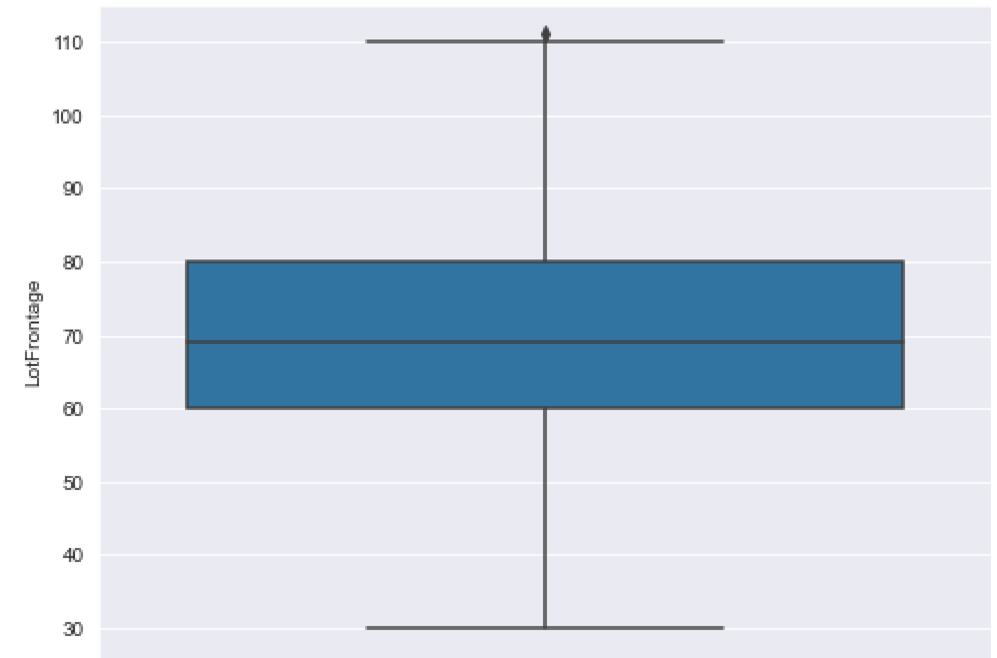
In [161...

```
# Plot box plot for LotFrontage column after removing outliers

sns.boxplot( y='LotFrontage', data = housing_without_LotFrontage_outliers)
```

Out[161...

<AxesSubplot:ylabel='LotFrontage'>



In [ ]:

In [ ]:

```
# 2. OUTLIER CAPPING
# The outliers are capped at certain minimum and maximum values.
# The rows containing the outliers are not removed from the dataset.
#  We  will again use  the Inter Quartile Range
technique to find the lower and  upper  Limit   #
for the outliers in the fare column of the Housing
dataset.
```

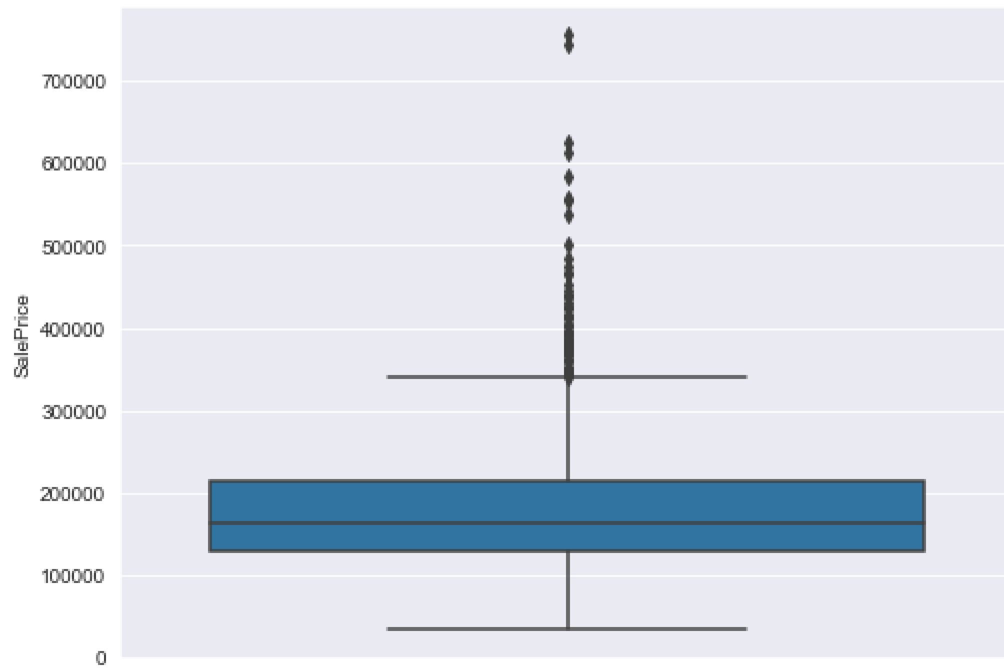
In [141...

```
# Plot box plot for the SalePrice column in Housing dataset

sns.boxplot(  y='SalePrice',  data=housing)
```

Out[141...

<AxesSubplot:ylabel='SalePrice'>



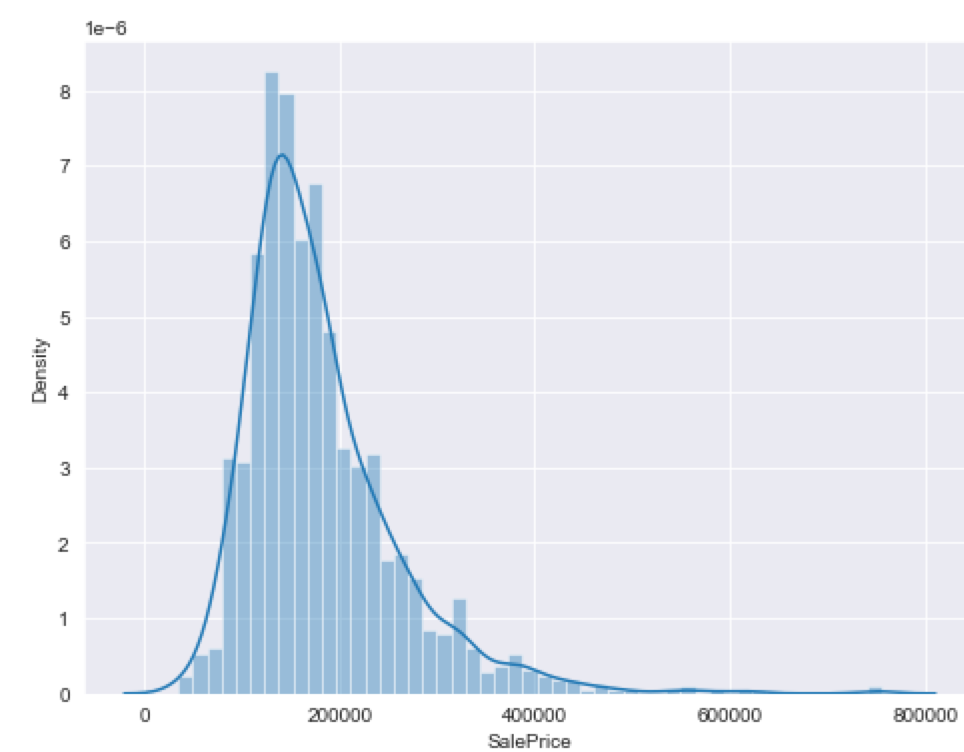
In [142…

```
# Now Let
us see how
the data is
distributed
in the
column  #
Plot the
histogram
sns.distplot(housing["SalePrice"])
```

c:\users\asus\appdata\local\programs\python\python39\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please a dapt 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)

Out[142…

<AxesSubplot:xlabel='SalePrice', ylabel='Density'>



In [143…

```
# use the IQR method to find the upper and lower Limits to find the outliers in the SalePrice column.
IQR =
housing["SalePrice"].quantile(0.75) -
housing["SalePrice"].quantile(0.25)
lower_SalePrice_limit =
housing["SalePrice"].quantile(0.25) -
(IQR * 1.5)
```

```
upper_SalePrice_limit =
housing["SalePrice"].quantile(
0.75) + (IQR * 1.5)
print(lower_SalePrice_limit)
print(upper_SalePrice_limit)
```

```
3937.5
340037.5
```

In [144…

```
# Replace the outlier values that are there in SalePrice column:
# The outliers (SalePrice values)
greater than the upper limit with
upper Limit        # The outliers
(SalePrice values) less than the
lower limit with Lower Limit

housing["SalePrice"]= np.where(housing["SalePrice"] >
  upper_SalePrice_limit, upper_SalePrice_limit,
  np.where(housing["SalePrice"] < lower_SalePrice_limit,
  lower_SalePrice_limit, housing["SalePrice"]))
```

In [145…

```
hou
sin
g
```

Out[145…

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu
0	65.0	8450	Reg	208500.0	2008	Normal	Gd	Attchd	548	NaN
1	80.0	9600	Reg	181500.0	2007	Normal	Gd	Attchd	460	TA
2	68.0	11250	IR1	223500.0	2008	Normal	Gd	Attchd	608	TA
3	60.0	9550	IR1	140000.0	2006	Abnorml	TA	Detchd	642	Gd
4	84.0	14260	IR1	250000.0	2008	Normal	Gd	Attchd	836	TA
...	...	...	...	...	...	...	...	...	...	...
1455	62.0	7917	Reg	175000.0	2007	Normal	Gd	Attchd	460	TA
1456	85.0	13175	Reg	210000.0	2010	Normal	Gd	Attchd	500	TA
1457	66.0	9042	Reg	266500.0	2010	Normal	TA	Attchd	252	Gd
1458	68.0	9717	Reg	142125.0	2010	Normal	TA	Attchd	240	NaN
1459	75.0	9937	Reg	147500.0	2008	Normal	TA	Attchd	276	NaN

1460 rows × 10 columns

In [146…

```
housing[3
0:40]
```

Out[146…

	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu
30	50.0	8500	Reg	40000.0	2008	Normal	TA	Detchd	250	NaN
31	NaN	8544	IR1	149350.0	2008	Normal	TA	Attchd	271	NaN
32	85.0	11049	Reg	179900.0	2008	Normal	Ex	Attchd	484	NaN
33	70.0	10552	IR1	165500.0	2010	Normal	TA	Attchd	447	Gd

34	60.0	7313	Reg	277500.0	2007	Normal	Ex	Attchd	556	Gd
35	108.0	13418	Reg	309000.0	2006	Normal	Ex	BuiltIn	691	Gd
36	112.0	10859	Reg	145000.0	2009	Normal	Gd	Attchd	672	NaN

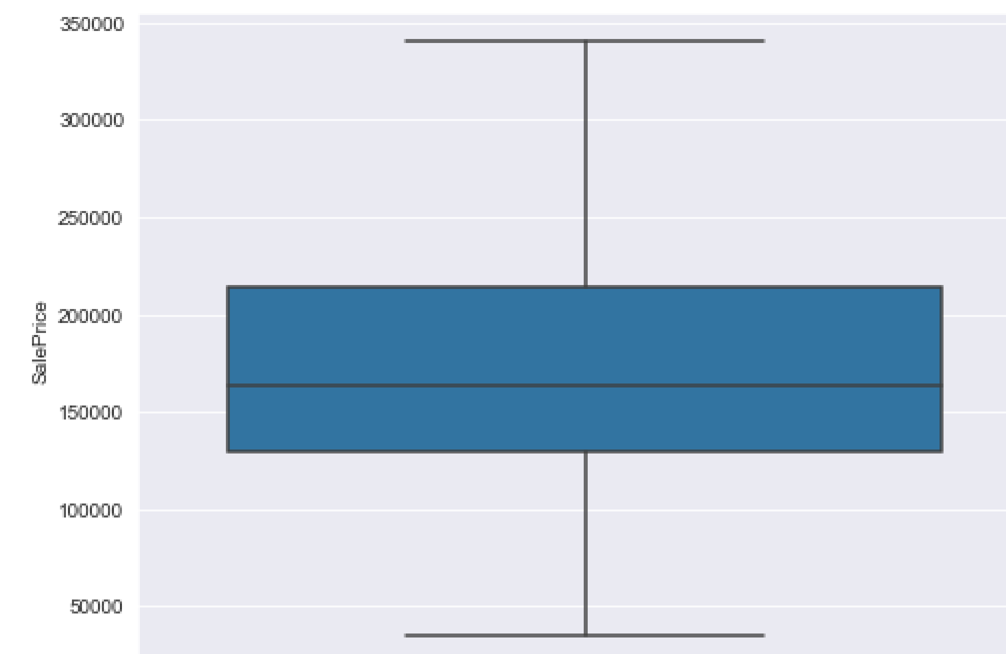
	LotFrontage	LotArea	LotShape	SalePrice	YrSold	SaleCondition	BsmtQual	GarageType	GarageArea	FireplaceQu
37	**74.0**	**8532**	**Reg**	**153000.0**	**2009**	**Normal**	**TA**	**Attchd**	**498**	**TA**
38	**68.0**	**7922**	**Reg**	**109000.0**	**2010**	**Abnorml**	**TA**	**Detchd**	**246**	**NaN**
39	**65.0**	**6040**	**Reg**	**82000.0**	**2008**	**AdjLand**	**NaN**	**NaN**	**0**	**NaN**

In [147…

```
# Plot the box plot to check any outliers in the SalePrice column

sns.boxplot( y='SalePrice', data=housing)
```

Out[147… <AxesSubplot:ylabel='SalePrice'>

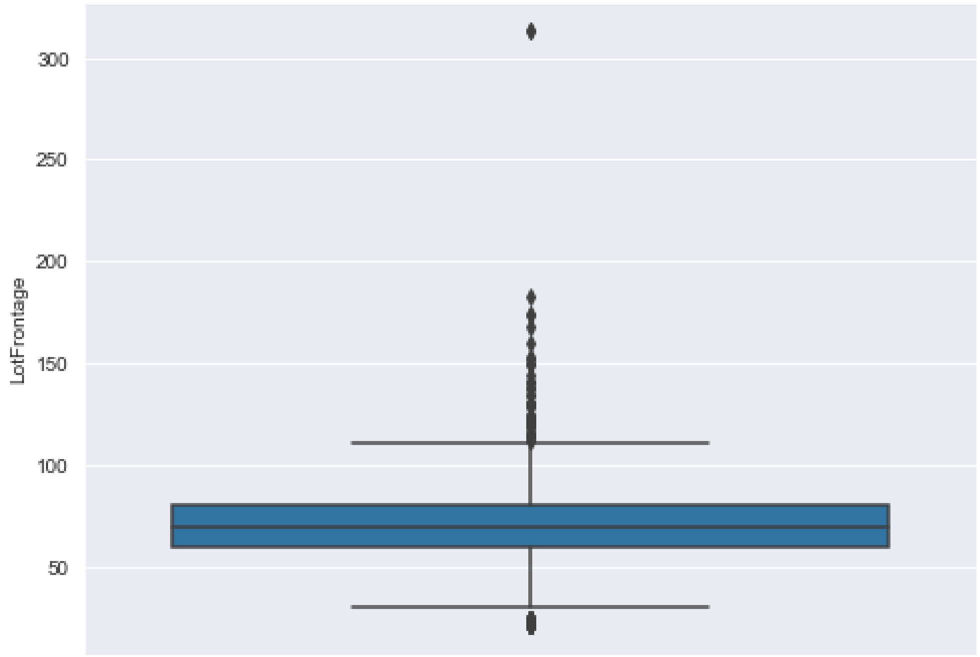


In [ ]:

In [148…

```
# Plot a box plot that displays the distribution of data in the LotFrontage column of the housing dataset.
sns.boxplot( y='LotFrontage', data=housing)
```

Out[148… <AxesSubplot:ylabel='LotFrontage'>



In [149…

```
[illegible]
```

-2.80429695542297
142.9042136914763

In [150…

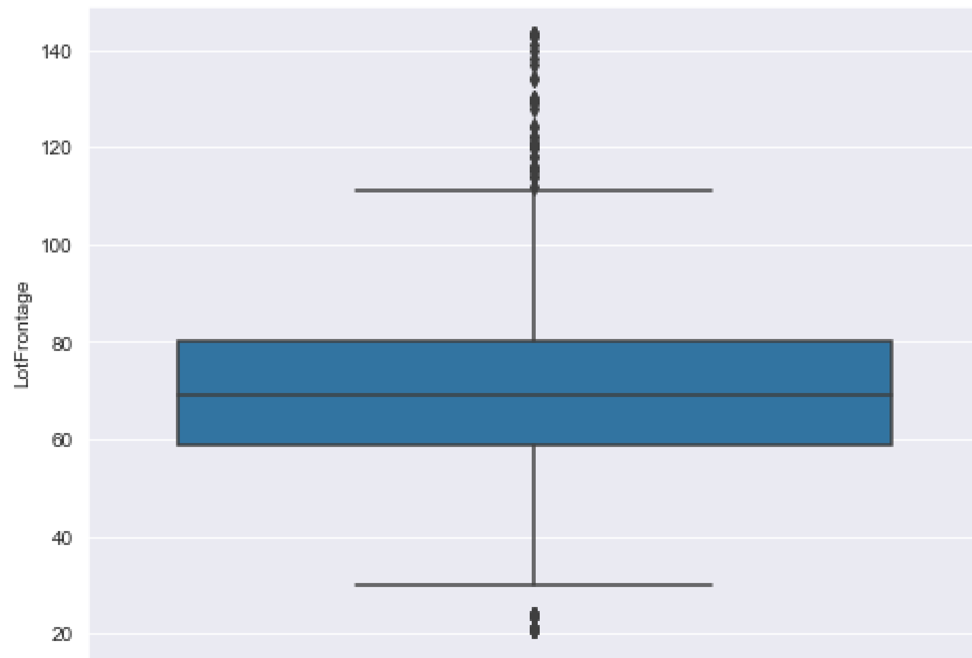
```
# Now replace the outlier values by the upper and lower limits.
housing["LotFrontage"]= np.where(housing["LotFrontage"] > upper_LotFrontage_limit, upper
np.where(housing["LotFrontage"] < lower_LotFrontage_limit, lower_LotFrontage_limit, housi
```

In [151…

```
sns.boxplot( y='LotFrontage', data=housing)
```

Out[151…

<AxesSubplot:ylabel='LotFrontage'>



In [ ]:

```
# 4. OUTLIER CAPPING USING QUANTILES
# You can also use quantile information to set the lower and upper Limits to find outliers.
# For instance, we can set 0.05 as the lower limit and 0.95 as the upper Limit to find the outliers, which means that
# if the data point is within the first 5 percent Lower values or 5 percent highest values, we consider it as an outlier.
```

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
```

In [ ]:

```
# plot a box plot for the SalePrice column of the housing dataset.
sns.boxplot(y='SalePrice', data=housing)
```

In [ ]:

```
#        setting 0.05 as the lower limit and 0.95 as the upper limit for the quantiles to find the outliers
```

In [152...

```
88000.0
326099.9999999999
```

In [153...

```
# replace all the values greater than 50 in the LotFrontage column of the
housing dataset by 50. Similarly, values        # Less than 10 have been
arbitrarily replaced by 20.
housing["SalePrice"]= np.where(housing["SalePrice"] > 50, 50,
np.where(housing["SalePrice"] < 10, 10, housing["SalePrice"]))
```

In [154...

```
# The output shows that anything beyond 112.07 is an outlier, and similarly, the fare value below 7.22 is also an outlier.
```

```
# Now replace the outlier values by the upper and lower limit.
housing["SalePrice"]= np.where(housing["SalePrice"] > upper_SalePrice_limit, upper_SalePrice_limit,
                      np.where(housing["SalePrice"] < lower_SalePrice_limit, lower_SalePrice_limit, housing["SalePrice"]))
```

In [155…

```
housing.SalePrice[1:40]
```

Out[155…

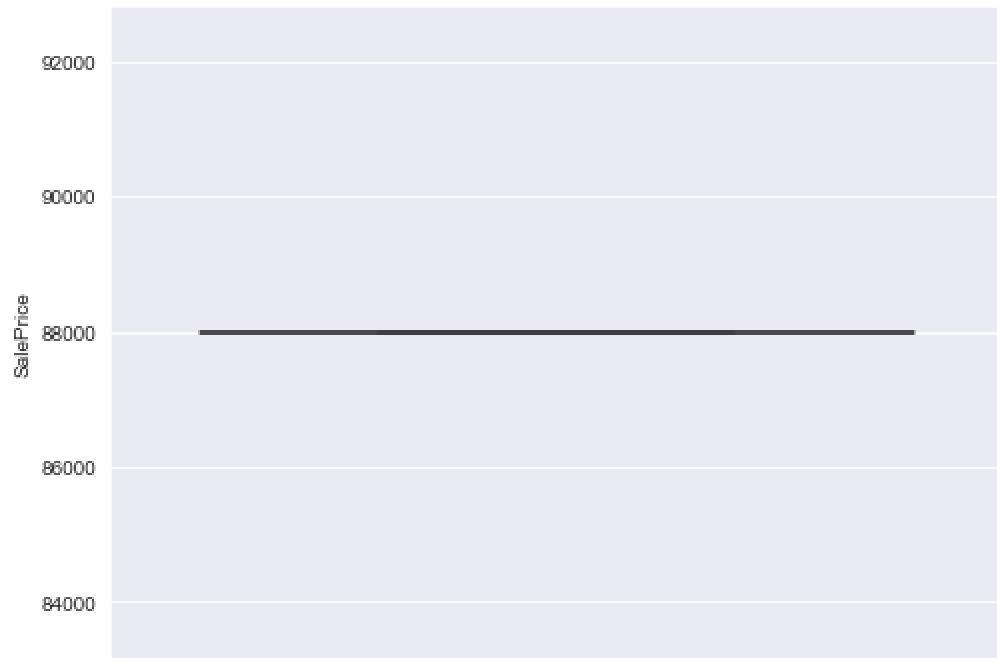
```
1     88000.0
2     88000.0
3     88000.0
4     88000.0
5     88000.0
6     88000.0
7     88000.0
8     88000.0
9     88000.0
10    88000.0
11    88000.0
12    88000.0
13    88000.0
14    88000.0
15    88000.0
16    88000.0
17    88000.0
18    88000.0
19    88000.0
20    88000.0
21    88000.0
22    88000.0
23    88000.0
24    88000.0
25    88000.0
26    88000.0
27    88000.0
28    88000.0
29    88000.0
30    88000.0
31    88000.0
32    88000.0
33    88000.0
34    88000.0
35    88000.0
36    88000.0
37    88000.0
38    88000.0
39    88000.0
Name: SalePrice,    dtype: float64
```

In [156…

```
# plot a box plot for the fare column of the housing dataset after removing outliers using the quantile method.
sns.boxplot( y='SalePrice', data=housing)
```

Out[156…

```
<AxesSubplot:ylabel='SalePrice'>
```



In [157…

88000.0
88000.0

In [158…

```
# replace all the values greater than 50 in the LotFrontage column of the
LotFrontage dataset by 50. Similarly, values      # Less than 10 have been
arbitrarily replaced by 20.
housing["LotFrontage"]= np.where(housing["LotFrontage"] > 50, 50,
np.where(housing["LotFrontage"] < 10, 10, housing["LotFrontage"]))
```

In [159…

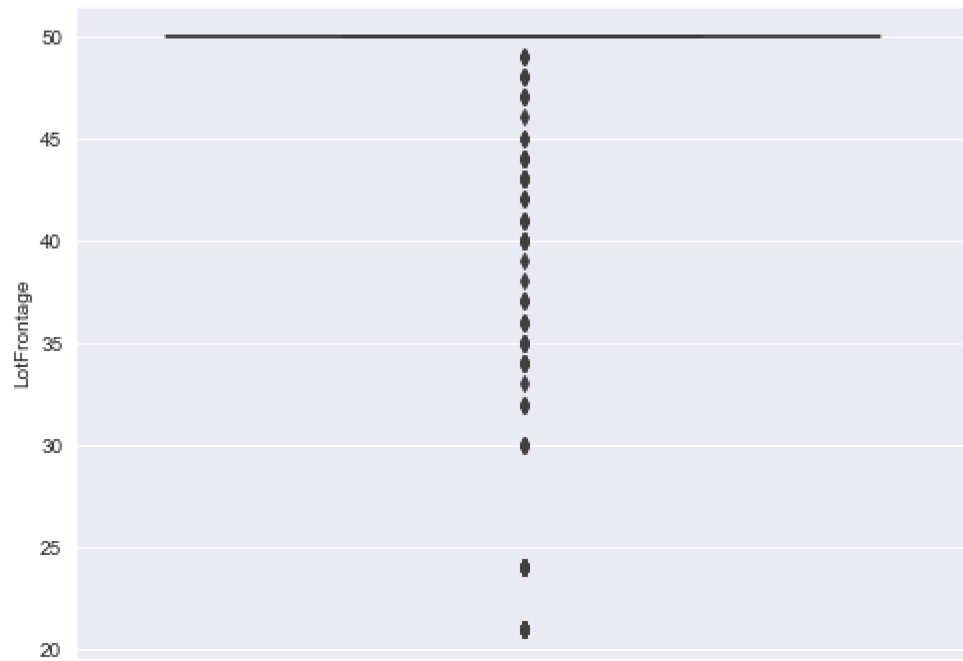
50.0
21.0

In [160…

```
# plot the box plot.
sns.boxplot(y='LotFrontage', data=housing)
```

Out[160…

<AxesSubplot:ylabel='LotFrontage'>



In [ ]:

In [ ]:

In [ ]:

TRANSFORMATIONS

In [89]:

In [96]:

Out[96]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

5 rows × 81 columns

In [102…

Out[102…

	YrSold	LotArea	OverallQual	SalePrice
0	2008	8450	7	208500
1	2007	9600	6	181500
2	2008	11250	7	223500
3	2006	9550	7	140000
4	2008	14260	8	250000
5	2009	14115	5	143000
6	2007	10084	8	307000
7	2009	10382	7	200000
8	2008	6120	7	129900
9	2008	7420	5	118000
10	2008	11200	5	129500
11	2006	11924	9	345000
12	2008	12968	5	144000
13	2007	10652	7	279500
14	2008	10920	6	157000
15	2007	6120	7	132000
16	2010	11241	6	149000
17	2006	10791	4	90000
18	2008	13695	5	159000
19	2009	7560	5	139000

In [104…

```
# Statistical measures
hu.describe()  # You can see that the mean, min, and max values for the three columns are very different.
```

Out[104…

	YrSold	LotArea	OverallQual	SalePrice
count	1460.000000	1460.000000	1460.000000	1460.000000

mean	2007.815753	10516.828082	6.099315	180921.195890

	YrSold	LotArea	OverallQual	SalePrice
std	1.328095	9981.264932	1.382997	79442.502883
min	2006.000000	1300.000000	1.000000	34900.000000
25%	2007.000000	7553.500000	5.000000	129975.000000
50%	2008.000000	9478.500000	6.000000	163000.000000
75%	2009.000000	11601.500000	7.000000	214000.000000
max	2010.000000	215245.000000	10.000000	755000.000000

In [109…

In [110…

Out[110…

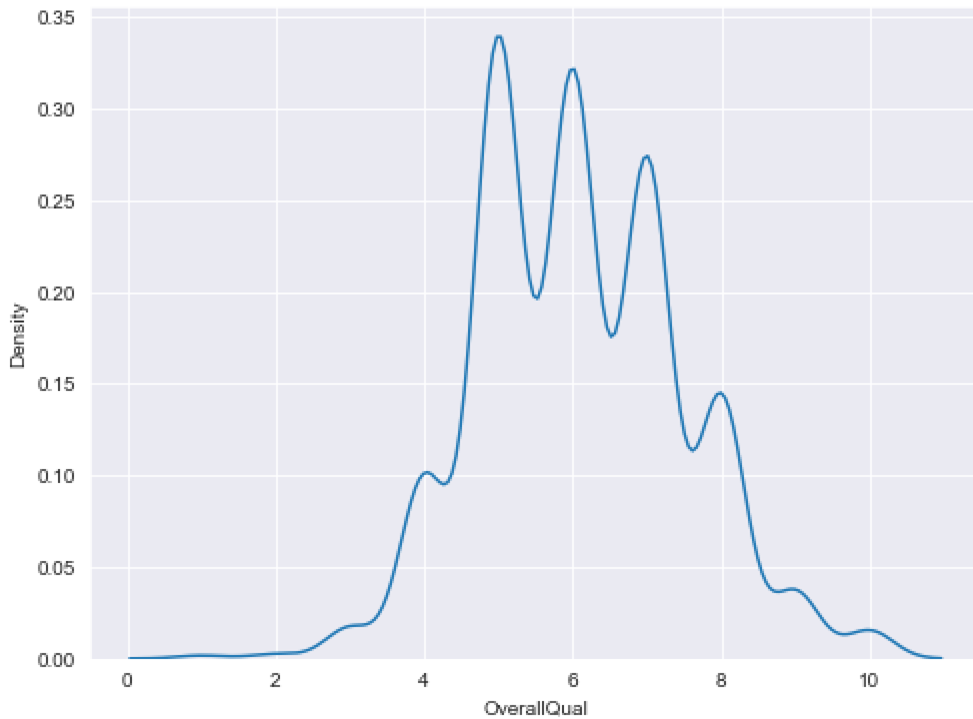
	YrSold	LotArea	OverallQual	SalePrice
0	0.138777	-0.207142	0.651479	0.347273
1	-0.614439	-0.091886	-0.071836	0.007288
2	0.138777	0.073480	0.651479	0.536154
3	-1.367655	-0.096897	0.651479	-0.515281
4	0.138777	0.375148	1.374795	0.869843

In [111…

```
sns.kdeplot(hu['OverallQual'])
```

Out[111…

<AxesSubplot:xlabel='OverallQual', ylabel='Density'>

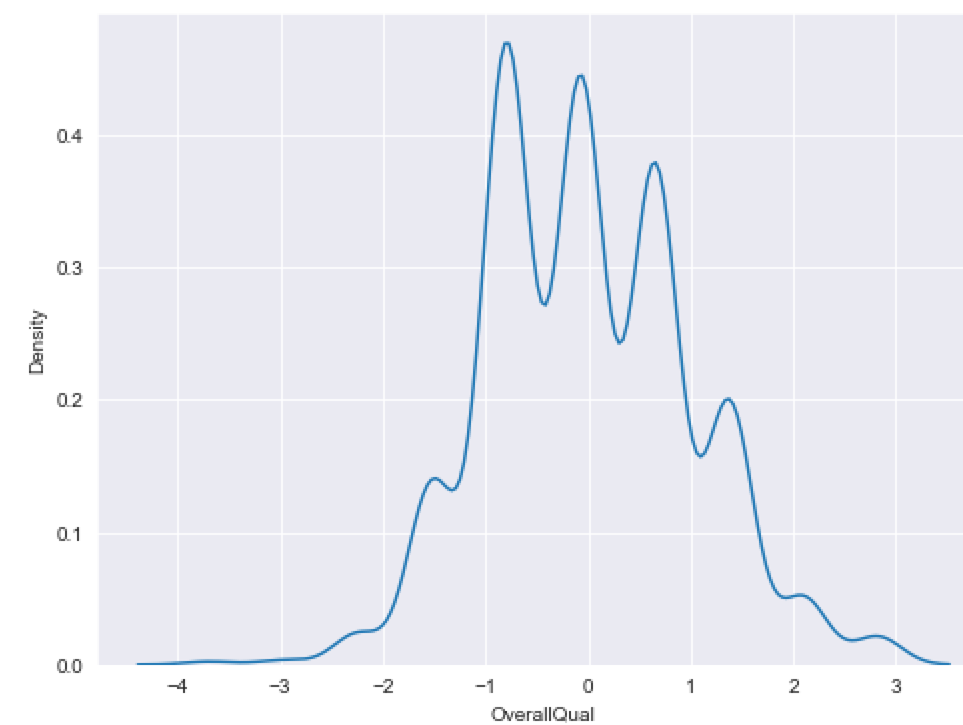


In [113...

```
sns.kdeplot(hu_scaled['OverallQual'])
```

Out[113...

<AxesSubplot:xlabel='OverallQual', ylabel='Density'>

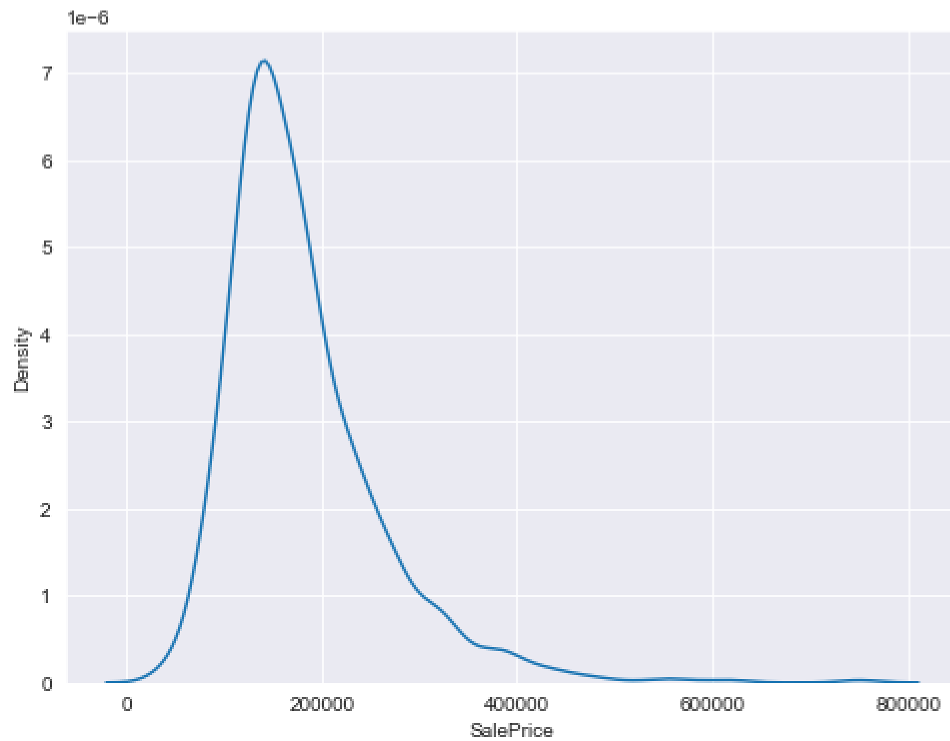


In [115...

```
# FARE COLUMN
sns.kdeplot(hu['SalePrice'])
```

Out[115...

<AxesSubplot:xlabel='SalePrice', ylabel='Density'>

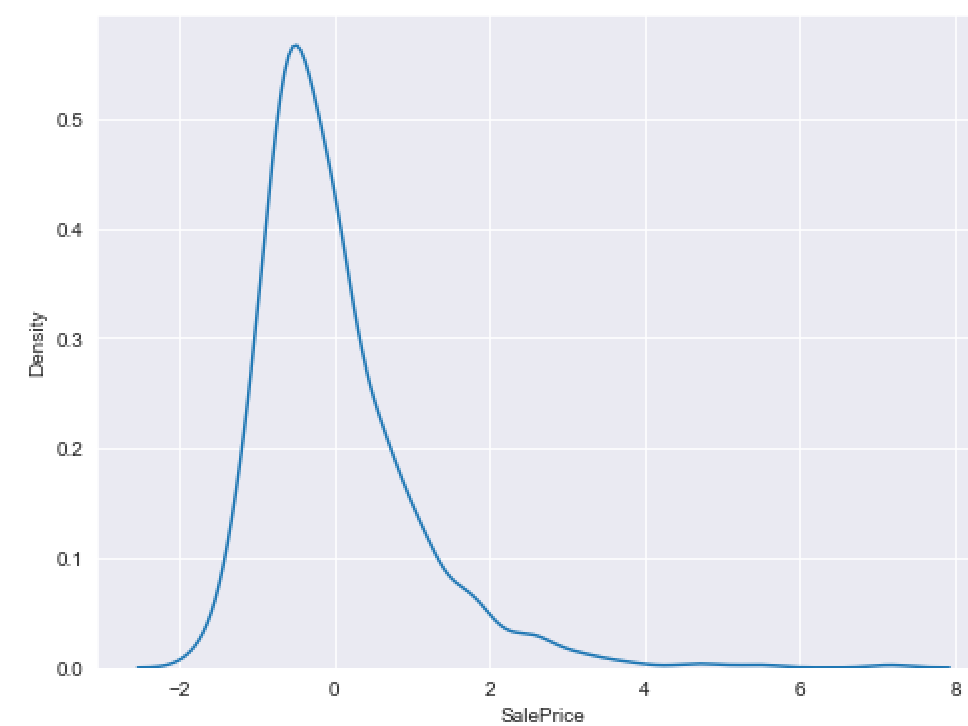


In [116…

```
sns.kdeplot(hu_scaled['SalePrice'])
```

Out[116…

<AxesSubplot:xlabel='SalePrice', ylabel='Density'>

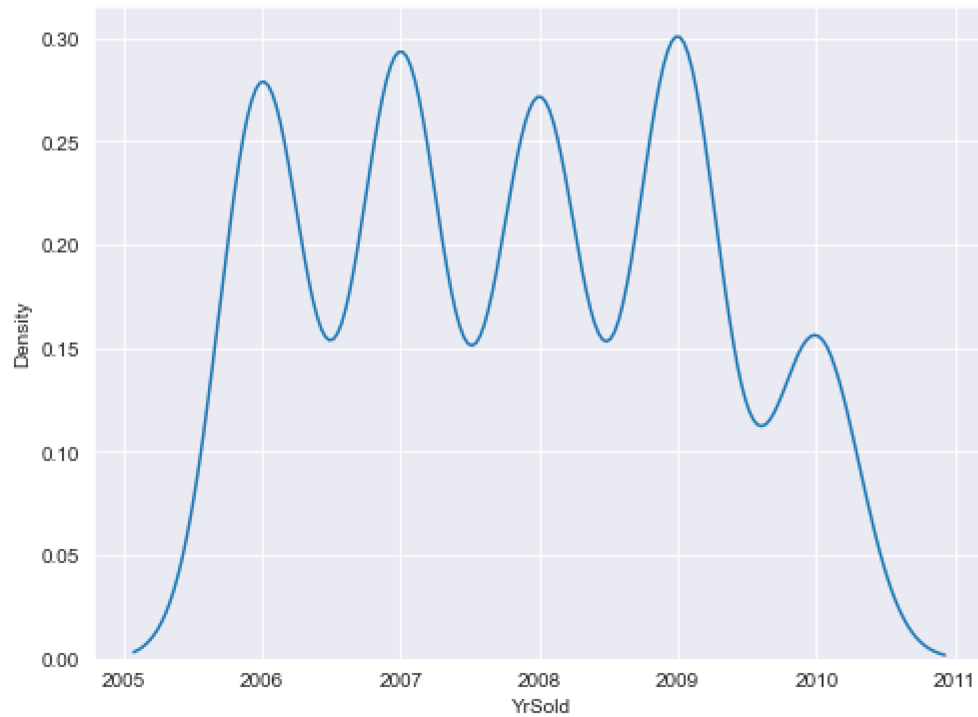


In [118…

```
sns.kdeplot(hu['YrSold'])
```

Out[118…

<AxesSubplot:xlabel='YrSold', ylabel='Density'>

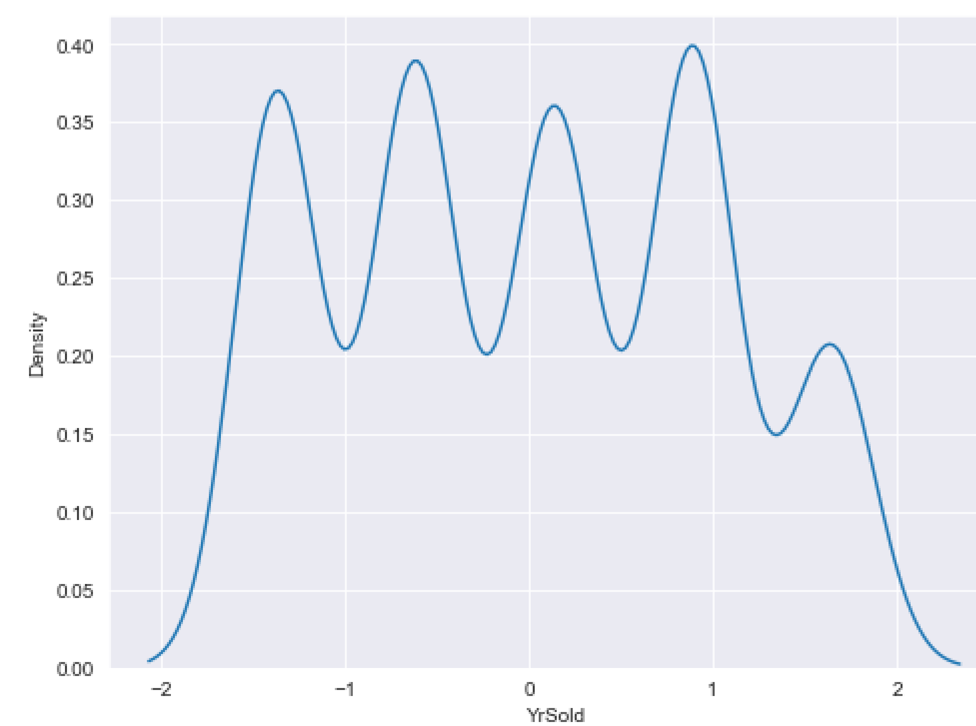


In [119…

```
sns.kdeplot(hu_scaled['YrSold'])
```

Out[119…

<AxesSubplot:xlabel='YrSold', ylabel='Density'>

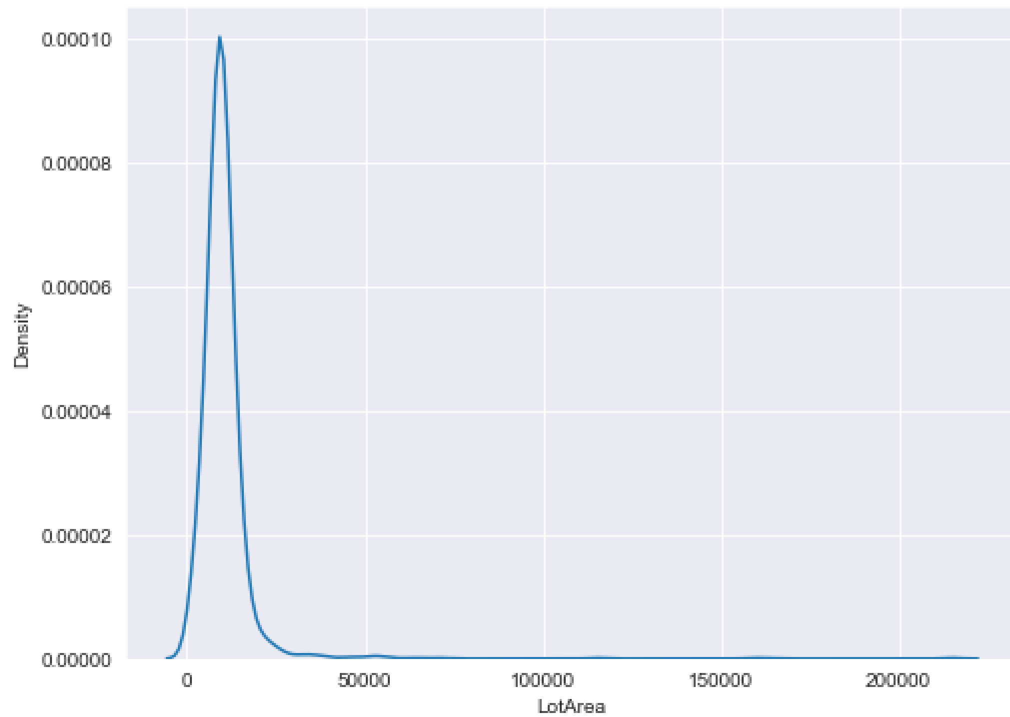


In [120…

```
sns.kdeplot(hu['LotArea'])
```

Out[120…

<AxesSubplot:xlabel='LotArea', ylabel='Density'>

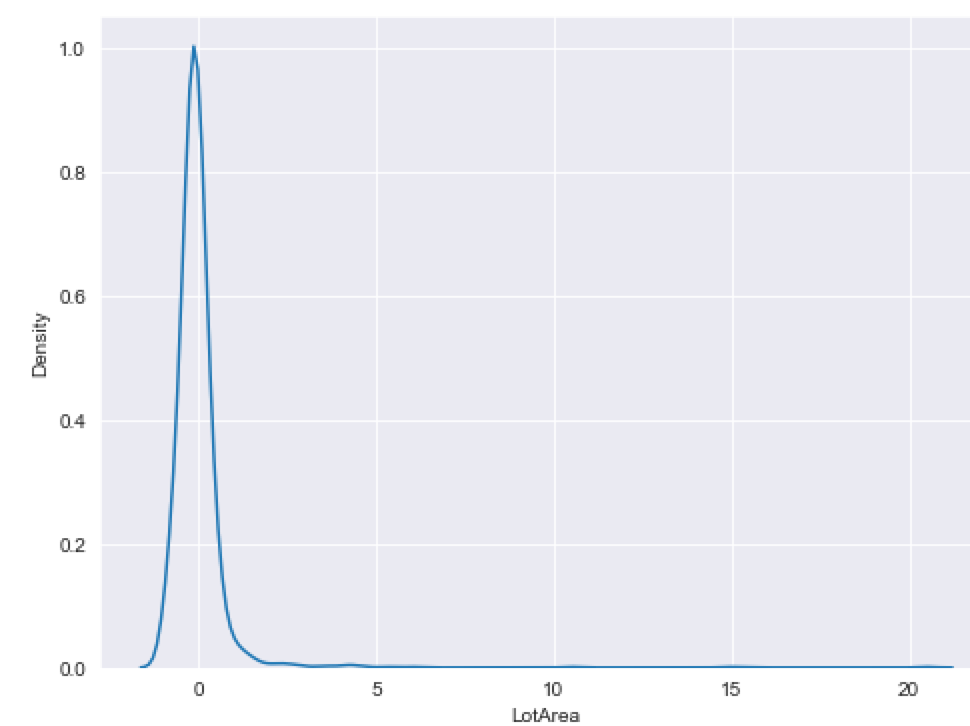


In [121…

```
sns.kdeplot(hu_scaled['LotArea'])
```

Out[121…

<AxesSubplot:xlabel='LotArea', ylabel='Density'>



In [122…

In [123...

```
hu_MMscaled = pd.DataFrame (hu_MMscaled, columns = hu.columns)
```

```
hu_MMscaled.
head()
```

Out[123…

	YrSold	LotArea	OverallQual	SalePrice
0	0.50	0.033420	0.666667	0.241078
1	0.25	0.038795	0.555556	0.203583
2	0.50	0.046507	0.666667	0.261908
3	0.00	0.038561	0.666667	0.145952
4	0.50	0.060576	0.777778	0.298709

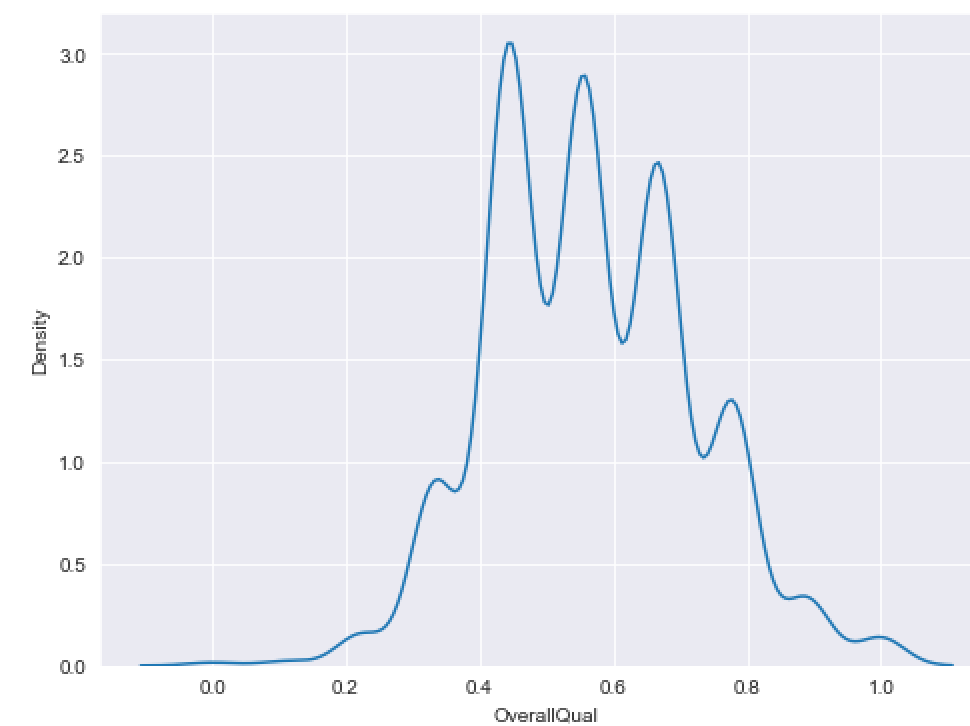
In [125…

```
# KERNEL DENSITY PLOT

sns.kdeplot(hu_MMscaled['OverallQual'])
```

Out[125…

<AxesSubplot:xlabel='OverallQual', ylabel='Density'>



In [126…

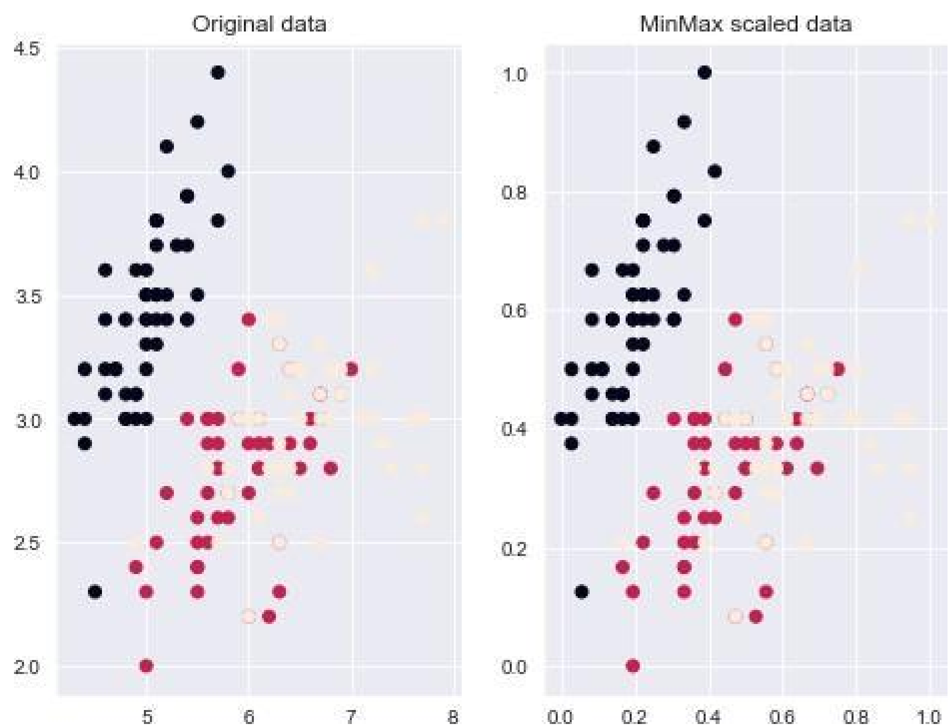
```
#IRIS DATA SET

from sklearn.datasets import load_iris
from sklearn.preprocessing import MinMaxScaler
import numpy as np
# use the iris dataset
X
,
y
=
l
o
a
d
_
i
r
i
s
(
r
e
t
u
r
n
_
X
_
y
```

```python
# Verify minimum value of all features
X_scaled.min(axis=0)
# array([0., O., 0., 0.])
# Verify maximum value of all features
X_scaled.max(axis=0)
# array([1., 1., 1., 1.])
# Manually normalise without using scikit-Learn
X_manual_scaled = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
# Verify manually VS scikit-Learn estimation
print(np.allclose(X_scaled, X_manual_scaled)) #True
```

(150, 4)
True

In [127...



In [ ]: