Data_Cleaning_Lab

May 3, 2022

1 Data Cleaning

Estimated time needed: **45** minutes

Most of the real-world data, that the data scientist work with, are raw data, meaning that it can contain repeated, missing, and irrelevant entries of information. Hence, if this data is used in any machine learning analysis, it will result in low accuracy or incorrect prediction. For this reason, data cleaning, also known as data cleansing, is an important technique that comes prior to any model building.

In this notebook, we will take a look at some of the common data cleaning techniques that data scientists may use to prepare their data for analysis.

1.1 Objectives

After completing this lab you will be able to:

- Use Log function to transform the data
- Handle the duplicates
- Handle the missing values
- Standardize and normalize the data
- Handle the outliers

1.2 Setup

For this lab, we will be using the following libraries:

- `pandas` for managing the data.
- `numpy` for mathematical operations.
- `seaborn` for visualizing the data.
- `matplotlib` for visualizing the data.
- `sklearn` for machine learning and machine-learning-pipeline related functions.
- `scipy` for statistical computations.

1.3 Import the required libraries

The following required modules are pre-installed in the Skills Network Labs environment. However if you run this notebook commands in a different Jupyter environment (e.g. Watson Studio or Ananconda) you will need to install these libraries by removing the `#` sign before `!mamba` in the code cell below.

[ ]:
```
# All Libraries required for this lab are listed below. The libraries␣
↪pre-installed on Skills Network Labs are commented.
# !mamba install -qy pandas==1.3.4 numpy==1.21.4 seaborn==0.9.0 matplotlib==3.5.
↪0 scikit-learn==0.20.1
# Note: If your environment doesn't support "!mamba install", use "!pip install"
```

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pylab as plt
%matplotlib inline

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from scipy.stats import norm
from scipy import stats
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/utils/validation.py:37: DeprecationWarning: distutils Version
classes are deprecated. Use packaging.version instead.
  LARGE_SPARSE_SUPPORTED = LooseVersion(scipy_version) >= '0.14.0'
```

1.4 Reading and understanding our data

For this lab, we will be using the Ames_Housing_Data.tsv file, hosted on IBM Cloud object storage. The Ames housing dataset examines features of houses sold in Ames (a small city in the state of Iowa in the United States) during the 2006–2010 timeframe.

Let's read the data into *pandas* data frame and look at the first 5 rows using the `head()` method.

[3]:
```
housing = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.
↪appdomain.cloud/IBM-ML0232EN-SkillsNetwork/asset/Ames_Housing_Data1.tsv",␣
↪sep='\t')
housing.head(10)
```

[3]:
```
   Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  \
0      1  526301100           20        RL         141.0     31770   Pave
1      1  526301100           20        RL         141.0     31770   Pave
2      2  526350040           20        RH          80.0     11622   Pave
3      3  526351010           20        RL          81.0     14267   Pave
4      4  526353030           20        RL          93.0     11160   Pave
5      5  527105010           60        RL          74.0     13830   Pave
6      6  527105030           60        RL          78.0      9978   Pave
```

```
7       7  527127150          120         RL           41.0      4920   Pave
8       8  527145080          120         RL           43.0      5005   Pave
9       9  527146030          120         RL           39.0      5389   Pave

  Alley Lot Shape Land Contour  … Pool Area Pool QC  Fence Misc Feature  \
0   NaN       IR1          Lvl  …         0     NaN    NaN          NaN
1   NaN       IR1          Lvl  …         0     NaN    NaN          NaN
2   NaN       Reg          Lvl  …         0     NaN  MnPrv          NaN
3   NaN       IR1          Lvl  …         0     NaN    NaN         Gar2
4   NaN       Reg          Lvl  …         0     NaN    NaN          NaN
5   NaN       IR1          Lvl  …         0     NaN  MnPrv          NaN
6   NaN       IR1          Lvl  …         0     NaN    NaN          NaN
7   NaN       Reg          Lvl  …         0     NaN    NaN          NaN
8   NaN       IR1          HLS  …         0     NaN    NaN          NaN
9   NaN       IR1          Lvl  …         0     NaN    NaN          NaN

  Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  SalePrice
0        0       5    2010       WD           Normal     215000
1        0       5    2010       WD           Normal     215000
2        0       6    2010       WD           Normal     105000
3    12500       6    2010       WD           Normal     172000
4        0       4    2010       WD           Normal     244000
5        0       3    2010       WD           Normal     189900
6        0       6    2010       WD           Normal     195500
7        0       4    2010       WD           Normal     213500
8        0       1    2010       WD           Normal     191500
9        0       3    2010       WD           Normal     236500

[10 rows x 82 columns]
```

We can find more information about the features and types using the `info()` method.

[4]:
```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2931 entries, 0 to 2930
Data columns (total 82 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Order            2931 non-null   int64
 1   PID              2931 non-null   int64
 2   MS SubClass      2931 non-null   int64
 3   MS Zoning        2931 non-null   object
 4   Lot Frontage     2441 non-null   float64
 5   Lot Area         2931 non-null   int64
 6   Street           2931 non-null   object
 7   Alley            198 non-null    object
 8   Lot Shape        2931 non-null   object
```

```
9   Land Contour     2931 non-null   object
10  Utilities        2931 non-null   object
11  Lot Config       2931 non-null   object
12  Land Slope       2931 non-null   object
13  Neighborhood     2931 non-null   object
14  Condition 1      2931 non-null   object
15  Condition 2      2931 non-null   object
16  Bldg Type        2931 non-null   object
17  House Style      2931 non-null   object
18  Overall Qual     2931 non-null   int64
19  Overall Cond     2931 non-null   int64
20  Year Built       2931 non-null   int64
21  Year Remod/Add   2931 non-null   int64
22  Roof Style       2931 non-null   object
23  Roof Matl        2931 non-null   object
24  Exterior 1st     2931 non-null   object
25  Exterior 2nd     2931 non-null   object
26  Mas Vnr Type     2908 non-null   object
27  Mas Vnr Area     2908 non-null   float64
28  Exter Qual       2931 non-null   object
29  Exter Cond       2931 non-null   object
30  Foundation       2931 non-null   object
31  Bsmt Qual        2851 non-null   object
32  Bsmt Cond        2851 non-null   object
33  Bsmt Exposure    2848 non-null   object
34  BsmtFin Type 1   2851 non-null   object
35  BsmtFin SF 1     2930 non-null   float64
36  BsmtFin Type 2   2850 non-null   object
37  BsmtFin SF 2     2930 non-null   float64
38  Bsmt Unf SF      2930 non-null   float64
39  Total Bsmt SF    2930 non-null   float64
40  Heating          2931 non-null   object
41  Heating QC       2931 non-null   object
42  Central Air      2931 non-null   object
43  Electrical       2930 non-null   object
44  1st Flr SF       2931 non-null   int64
45  2nd Flr SF       2931 non-null   int64
46  Low Qual Fin SF  2931 non-null   int64
47  Gr Liv Area      2931 non-null   int64
48  Bsmt Full Bath   2929 non-null   float64
49  Bsmt Half Bath   2929 non-null   float64
50  Full Bath        2931 non-null   int64
51  Half Bath        2931 non-null   int64
52  Bedroom AbvGr    2931 non-null   int64
53  Kitchen AbvGr    2931 non-null   int64
54  Kitchen Qual     2931 non-null   object
55  TotRms AbvGrd    2931 non-null   int64
56  Functional       2931 non-null   object
```

```
 57  Fireplaces       2931 non-null   int64
 58  Fireplace Qu     1509 non-null   object
 59  Garage Type      2774 non-null   object
 60  Garage Yr Blt    2772 non-null   float64
 61  Garage Finish    2772 non-null   object
 62  Garage Cars      2930 non-null   float64
 63  Garage Area      2930 non-null   float64
 64  Garage Qual      2772 non-null   object
 65  Garage Cond      2772 non-null   object
 66  Paved Drive      2931 non-null   object
 67  Wood Deck SF     2931 non-null   int64
 68  Open Porch SF    2931 non-null   int64
 69  Enclosed Porch   2931 non-null   int64
 70  3Ssn Porch       2931 non-null   int64
 71  Screen Porch     2931 non-null   int64
 72  Pool Area        2931 non-null   int64
 73  Pool QC          13 non-null     object
 74  Fence            572 non-null    object
 75  Misc Feature     106 non-null    object
 76  Misc Val         2931 non-null   int64
 77  Mo Sold          2931 non-null   int64
 78  Yr Sold          2931 non-null   int64
 79  Sale Type        2931 non-null   object
 80  Sale Condition   2931 non-null   object
 81  SalePrice        2931 non-null   int64
dtypes: float64(11), int64(28), object(43)
memory usage: 1.8+ MB
```

According to the output above, we have 2930 entries, 0 to 2929, as well as 81 features. The "Non-Null Count" column shows the number of non-null entries. If the count is 2930 then there is no missing values for that particular feature. 'SalePrice' is our target or response variable and the rest of the features are our predictor variables.

We also have a mix of numerical (28 int64 and 11 float64) and object data types.

Next, let's use the `describe()` function to show the count, mean, min, max of the sale price attribute.

[5]:
```python
housing["SalePrice"].describe()
```

[5]:
```
count      2931.000000
mean     180807.729785
std       79875.557267
min       12789.000000
25%      129500.000000
50%      160000.000000
75%      213500.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

From the above analysis, it is important to note that the minimum value is greater than 0. Also, there is a big difference between the minimum value and the 25th percentile. It is bigger than the 75th percentile and the maximum value. This means that our data might not be normally distributed (an important assumption for linear regression analysis), so will check for normality in the Log Transform section.

1.5 Exercise 1

The `describe()` function reveals the statistical information about the numeric attributes. To reveal some information about our categorical (object) attributes, we can use `value_counts()` function. In this exercise, describe all categories of the 'Sale Condition' attribute.

[6]:
```
# Enter your code and run the cell
housing['Sale Condition'].value_counts()
```

[6]:
```
Normal     2414
Partial     245
Abnorml     190
Family       46
Alloca       24
AdjLand      12
Name: Sale Condition, dtype: int64
```

Solution (Click Here)

`    <code>`

housing["Sale Condition"].value_counts()

1.6 Looking for Correlations

Before proceeding with the data cleaning, it is useful to establish a correlation between the response variable (in our case the sale price) and other predictor variables, as some of them might not have any major impact in determining the price of the house and will not be used in the analysis. There are many ways to discover correlation between the target variable and the rest of the features. Building pair plots, scatter plots, heat maps, and a correlation matrixes are the most common ones. Below, we will use the `corr()` function to list the top features based on the pearson correlation coefficient (measures how closely two sequences of numbers are correlated). Correlation coefficient can only be calculated on the numerical attributes (floats and integers), therefore, only the numeric attributes will be selected.

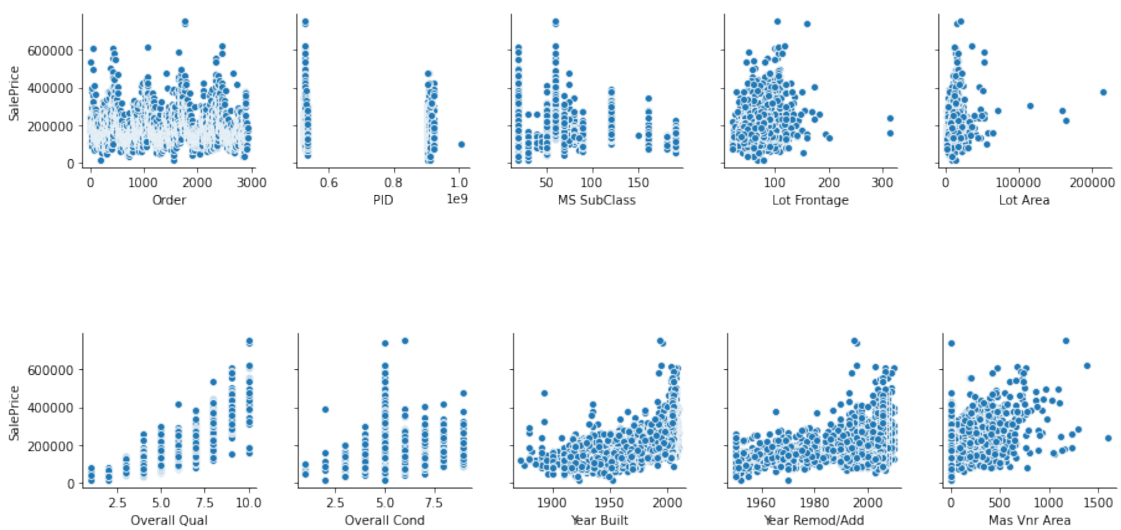
[7]:
```
hous_num = housing.select_dtypes(include = ['float64', 'int64'])
hous_num_corr = hous_num.corr()['SalePrice'][:-1] # -1 means that the latest␣
↪row is SalePrice
top_features = hous_num_corr[abs(hous_num_corr) > 0.5].
↪sort_values(ascending=False) #displays pearsons correlation coefficient␣
↪greater than 0.5
print("There is {} strongly correlated values with SalePrice:\n{}".
↪format(len(top_features), top_features))
```

```
There is 11 strongly correlated values with SalePrice:
Overall Qual      0.799226
Gr Liv Area       0.706791
Garage Cars       0.647891
Garage Area       0.640411
Total Bsmt SF     0.632270
1st Flr SF        0.621672
Year Built        0.558340
Full Bath         0.545339
Year Remod/Add    0.532664
Garage Yr Blt     0.526808
Mas Vnr Area      0.508277
Name: SalePrice, dtype: float64
```

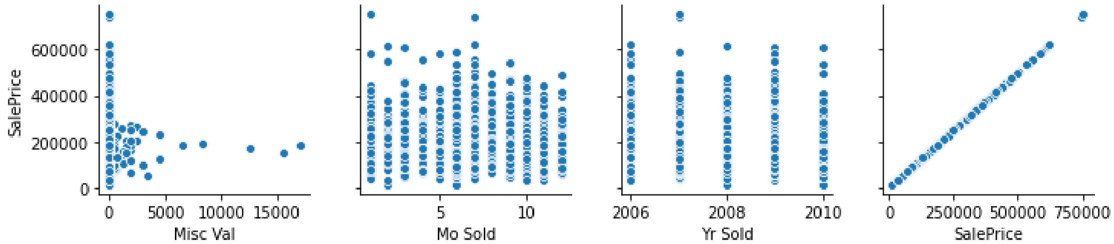
Above, there are 11 features, with coefficients greater than 0.5, that are strongly correlated with the sale price.

Next, let's generate some par plots to visually inspect the correlation between some of these features and the target variable. We will use seaborns **sns.pairplot()** function for this analysis. Also, building pair plots is one of the possible ways to spot the outliers that might be present in the data.

[8]:
```python
for i in range(0, len(hous_num.columns), 5):
    sns.pairplot(data=hous_num,
                 x_vars=hous_num.columns[i:i+5],
                 y_vars=['SalePrice'])
```




SalePrice
BsmtFin SF 1
BsmtFin SF 2
Bsmt Unf SF
Total Bsmt SF
1st Flr SF
2nd Flr SF
Low Qual Fin SF
Gr Liv Area
Bsmt Full Bath
Bsmt Half Bath
Full Bath
Half Bath
Bedroom AbvGr
Kitchen AbvGr
TotRms AbvGrd
Fireplaces
Garage Yr Blt
Garage Cars
Garage Area
Wood Deck SF
Open Porch SF
Enclosed Porch
3Ssn Porch
Screen Porch
Pool Area

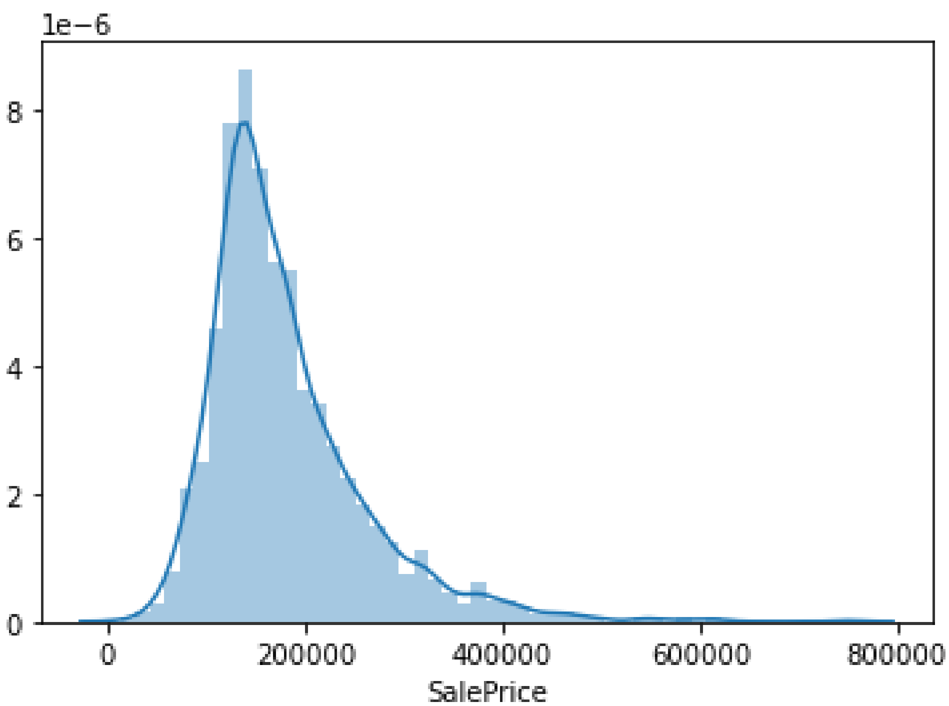


From Pearsons Correlation Coefficients and pair plots, we can draw some conclusions about the features that are most strongly correlated to the 'SalePrice'. They are: 'Overall Qual', 'Gr Liv Area', 'Garage Cars', 'Garage Area', and others.

1.7 Log Transformation

In this section, we are going to inspect whether our 'SalePrice' data are normally distributed. The assumption of the normal distribution must be met in order to perform any type of regression analysis. There are several ways to check for this assumption, however here, we will use the visual method, by plotting the 'SalePrice' distribution using the `distplot()` function from the `seaborn` library.

[9]:
```
sp_untransformed = sns.distplot(housing['SalePrice'])
```



As the plot shows, our 'SalePrice' deviates from the normal distribution. It has a longer tail to the right, so we call it a positive skew. In statistics *skewness* is a measure of asymmetry of the distribution. In addition to skewness, there is also a kurtosis, parameter which refers to the pointedness of a peak in the distribution curve. Both skewness and kurtosis are frequently used together to characterize the distribution of data.

Here, we can simply use the `skew()` function to calculate our skewness level of the `SalePrice`.

[10]:
```
print("Skewness: %f" % housing['SalePrice'].skew())
```

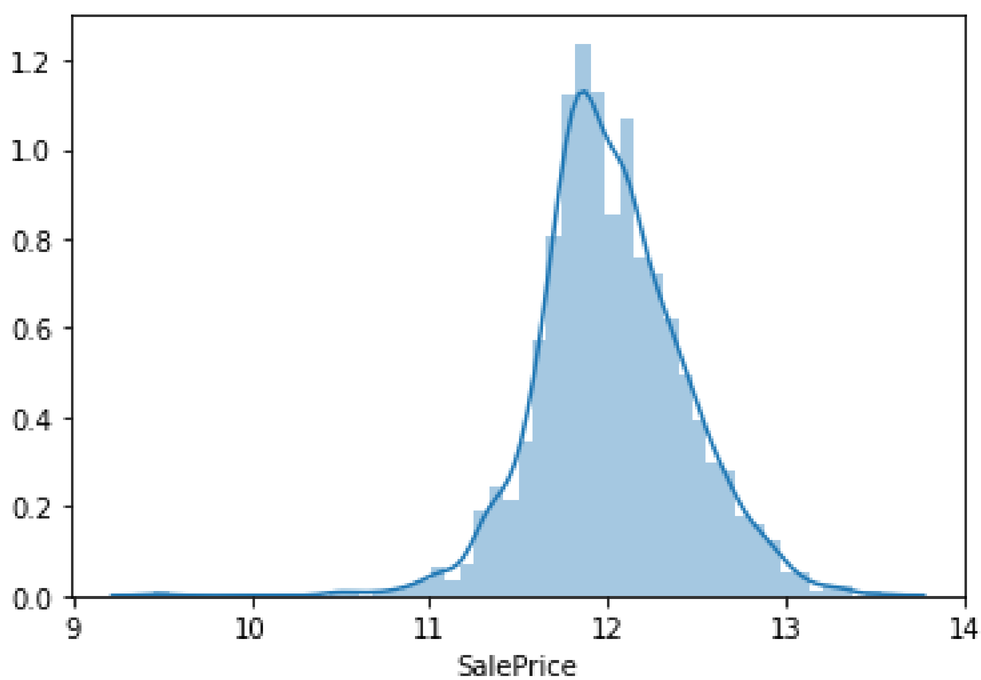
```
Skewness: 1.743222
```

The range of skewness for a fairly symmetrical bell curve distribution is between -0.5 and 0.5; moderate skewness is -0.5 to -1.0 and 0.5 to 1.0; and highly skewed distribution is < -1.0 and > 1.0. In our case, we have ~1.7, so it is considered highly skewed data.

Now, we can try to transform our data, so it looks more normally distributed. We can use the `np.log()` function from the `numpy` library to perform log transform. This documentation contains more information about the numpy log transform.

[11]:
```
log_transformed = np.log(housing['SalePrice'])
```

[12]:
```
sp_transformed = sns.distplot(log_transformed)
```



[13]:
```
print("Skewness: %f" % (log_transformed).skew())
```

```
Skewness: -0.015354
```

As we can see, the log method transformed the 'SalePrice' distribution into a more symmetrical bell curve and the skewness level now is -0.01, well within the range.

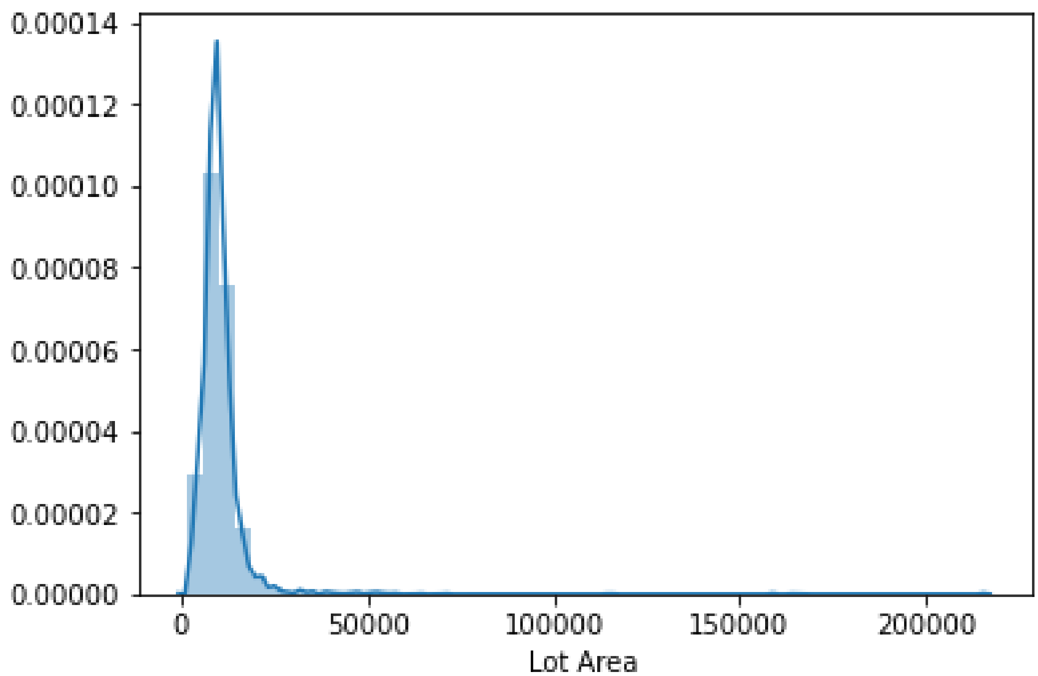
There are other ways to correct for skewness of the data. For example, Square Root Transform (`np.sqrt`) and the Box-Cox Transform (`stats.boxcox` from the `scipy stats` library). To learn more about these two methods, please check out this article.

1.8 Exercise 2

In this exercise, visually inspect the 'Lot Area' feature. If there is any skewness present, apply log transform to make it more normally distributed.

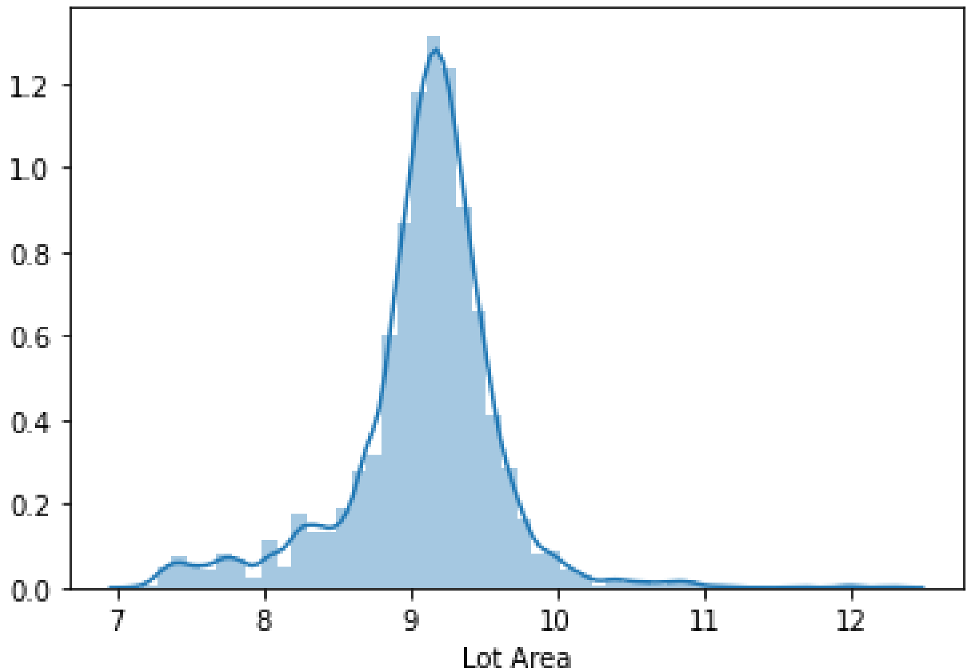
[16]:
```
# Enter your code and run the cell
la_untransformed = sns.distplot(housing['Lot Area'])
print("Skewness: %f" % housing['Lot Area'].skew())
```

```
Skewness: 12.778041
```



[18]:
```
la_log_transformed = np.log(housing['Lot Area'])
la_transformed = sns.distplot(la_log_transformed)
print("Skewness: %f" % (la_log_transformed).skew())
```

```
Skewness: -0.494639
```



Solution (Click Here)

`    <code>`

la_plot = sns.distplot(housing['Lot Area']) print("Skewness: %f" % housing['Lot Area'].skew()) la_log = np.log(housing['Lot Area']) print("Skewness: %f" % la_log.skew())

1.9 Handling the Duplicates

As mentioned in the video, having duplicate values can effect our analysis, so it is good to check whether there are any duplicates in our data. We will use pandas `duplicated()` function and search by the 'PID' column, which contains a unique index number for each entry.

[19]:
```
duplicate = housing[housing.duplicated(['PID'])]
duplicate
```

[19]:
```
   Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  \
1      1  526301100           20        RL         141.0     31770   Pave

  Alley Lot Shape Land Contour  … Pool Area Pool QC Fence Misc Feature  \
1   NaN       IR1          Lvl  …         0     NaN   NaN          NaN

  Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  SalePrice
1        0       5    2010       WD           Normal     215000

[1 rows x 82 columns]
```

As we can see, there is one duplicate row in this dataset. To remove it, we can use pandas `drop_duplicates()` function. By default, it removes all duplicate rows based on all the columns.

[20]:
```python
dup_removed = housing.drop_duplicates()
dup_removed
```

[20]:
```
      Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  \
0         1  526301100           20        RL         141.0     31770   Pave
2         2  526350040           20        RH          80.0     11622   Pave
3         3  526351010           20        RL          81.0     14267   Pave
4         4  526353030           20        RL          93.0     11160   Pave
5         5  527105010           60        RL          74.0     13830   Pave
…       …        …          …       …           …        …    …
2926   2926  923275080           80        RL          37.0      7937   Pave
2927   2927  923276100           20        RL           NaN      8885   Pave
2928   2928  923400125           85        RL          62.0     10441   Pave
2929   2929  924100070           20        RL          77.0     10010   Pave
2930   2930  924151050           60        RL          74.0      9627   Pave

     Alley Lot Shape Land Contour  … Pool Area Pool QC  Fence Misc Feature  \
0      NaN       IR1          Lvl  …         0     NaN    NaN          NaN
2      NaN       Reg          Lvl  …         0     NaN  MnPrv          NaN
3      NaN       IR1          Lvl  …         0     NaN    NaN         Gar2
4      NaN       Reg          Lvl  …         0     NaN    NaN          NaN
5      NaN       IR1          Lvl  …         0     NaN  MnPrv          NaN
…     …       …          …  …         …     …    …          …
2926   NaN       IR1          Lvl  …         0     NaN  GdPrv          NaN
2927   NaN       IR1          Low  …         0     NaN  MnPrv          NaN
2928   NaN       Reg          Lvl  …         0     NaN  MnPrv         Shed
2929   NaN       Reg          Lvl  …         0     NaN    NaN          NaN
2930   NaN       Reg          Lvl  …         0     NaN    NaN          NaN

     Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  SalePrice
0           0       5    2010       WD           Normal     215000
2           0       6    2010       WD           Normal     105000
3       12500       6    2010       WD           Normal     172000
4           0       4    2010       WD           Normal     244000
5           0       3    2010       WD           Normal     189900
…         …     …     …       …              …       …
2926        0       3    2006       WD           Normal     142500
2927        0       6    2006       WD           Normal     131000
2928      700       7    2006       WD           Normal     132000
2929        0       4    2006       WD           Normal     170000
2930        0      11    2006       WD           Normal     188000

[2930 rows x 82 columns]
```

An alternative way to check if there are any duplicated Indexes in our dataset is using `index.is_unique` function.

[21]:
```
housing.index.is_unique
```

[21]:
```
True
```

1.10 Exercise 3

In this exercise try to remove duplicates on a specific column by setting the subset equal to the column that contains the duplicate, such as 'Order'.

[24]:
```
# Enter your code and run the cell
removed_sub = housing.drop_duplicates(subset=['Order'])
```

Solution (Click Here)

`    <code>`

removed_sub = housing.drop_duplicates(subset=['Order'])

1.11 Handling the Missing Values

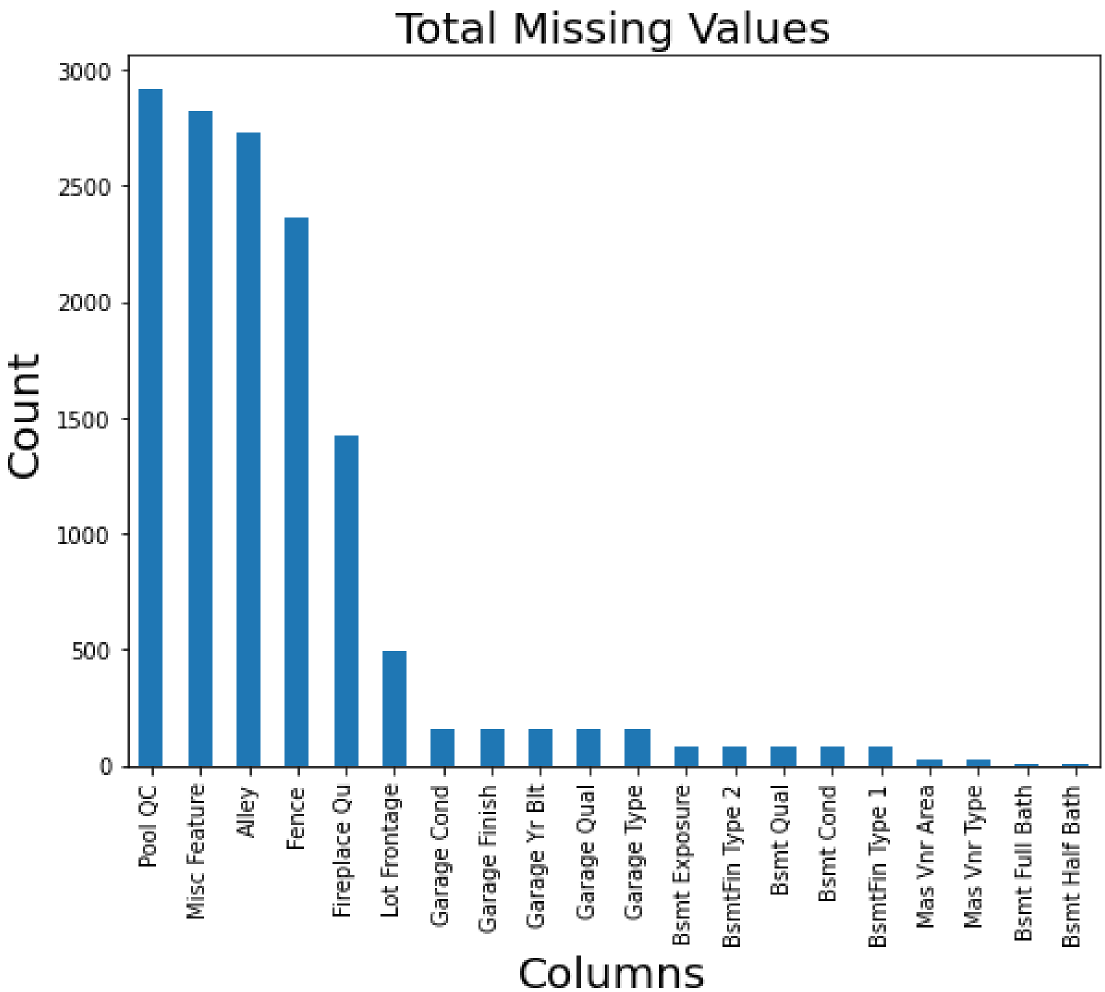
1.11.1 Finding the Missing Values

For easier detection of missing values, pandas provides the `isna()`, `isnull()`, and `notna()` functions. For more information on pandas missing values please check out this documentation.

To summarize all the missing values in our dataset, we will use `isnull()` function. Then, we will add them all up, by using `sum()` function, sort them with `sort_values()` function, and plot the first 20 columns (as the majority of our missing values fall within first 20 columns), using the `bar plot` function from the `matplotlib` library.

[25]:
```
total = housing.isnull().sum().sort_values(ascending=False)
total_select = total.head(20)
total_select.plot(kind="bar", figsize = (8,6), fontsize = 10)

plt.xlabel("Columns", fontsize = 20)
plt.ylabel("Count", fontsize = 20)
plt.title("Total Missing Values", fontsize = 20)
```

[25]:
```
Text(0.5, 1.0, 'Total Missing Values')
```



There are several options for dealing with missing values. We will use 'Lot Frontage' feature to analyze for missing values.

1. We can drop the missing values, using `dropna()` method.

[26]:
```
housing.dropna(subset=["Lot Frontage"])
```

[26]:

	Order	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	\
0	1	526301100	20	RL	141.0	31770	Pave	
1	1	526301100	20	RL	141.0	31770	Pave	
2	2	526350040	20	RH	80.0	11622	Pave	
3	3	526351010	20	RL	81.0	14267	Pave	
4	4	526353030	20	RL	93.0	11160	Pave	
…	…	…	…	…	…	…	…	
2925	2925	923251180	20	RL	160.0	20000	Pave	
2926	2926	923275080	80	RL	37.0	7937	Pave	
2928	2928	923400125	85	RL	62.0	10441	Pave	

```
2929    2929  924100070           20          RL           77.0      10010    Pave
2930    2930  924151050           60          RL           74.0       9627    Pave

     Alley Lot Shape Land Contour  … Pool Area Pool QC  Fence Misc Feature  \
0      NaN       IR1          Lvl  …         0     NaN    NaN          NaN
1      NaN       IR1          Lvl  …         0     NaN    NaN          NaN
2      NaN       Reg          Lvl  …         0     NaN  MnPrv          NaN
3      NaN       IR1          Lvl  …         0     NaN    NaN         Gar2
4      NaN       Reg          Lvl  …         0     NaN    NaN          NaN
…      …         …          …   …  …      …      …          …
2925   NaN       Reg          Lvl  …         0     NaN    NaN          NaN
2926   NaN       IR1          Lvl  …         0     NaN  GdPrv          NaN
2928   NaN       Reg          Lvl  …         0     NaN  MnPrv         Shed
2929   NaN       Reg          Lvl  …         0     NaN    NaN          NaN
2930   NaN       Reg          Lvl  …         0     NaN    NaN          NaN

     Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  SalePrice
0           0       5    2010       WD           Normal     215000
1           0       5    2010       WD           Normal     215000
2           0       6    2010       WD           Normal     105000
3       12500       6    2010       WD           Normal     172000
4           0       4    2010       WD           Normal     244000
…         …     …     …       …             …       …
2925        0       9    2006       WD          Abnorml     131000
2926        0       3    2006       WD           Normal     142500
2928      700       7    2006       WD           Normal     132000
2929        0       4    2006       WD           Normal     170000
2930        0      11    2006       WD           Normal     188000

[2441 rows x 82 columns]
```

Using this method, all the rows, containing null values in 'Lot Frontage' feature, for example, will be dropped.

2. We can drop the whole attribute (column), that contains missing values, using the `drop()` method.

[27]:
```python
housing.drop("Lot Frontage", axis=1)
```

[27]:
```
       Order        PID  MS SubClass MS Zoning  Lot Area Street Alley  \
0          1  526301100           20        RL     31770   Pave   NaN
1          1  526301100           20        RL     31770   Pave   NaN
2          2  526350040           20        RH     11622   Pave   NaN
3          3  526351010           20        RL     14267   Pave   NaN
4          4  526353030           20        RL     11160   Pave   NaN
…        …        …          …      …        …     …    …
2926    2926  923275080           80        RL      7937   Pave   NaN
2927    2927  923276100           20        RL      8885   Pave   NaN
```

```
2928   2928  923400125           85        RL     10441   Pave   NaN
2929   2929  924100070           20        RL     10010   Pave   NaN
2930   2930  924151050           60        RL      9627   Pave   NaN

     Lot Shape Land Contour Utilities  … Pool Area Pool QC  Fence  \
0          IR1          Lvl    AllPub  …         0     NaN    NaN
1          IR1          Lvl    AllPub  …         0     NaN    NaN
2          Reg          Lvl    AllPub  …         0     NaN  MnPrv
3          IR1          Lvl    AllPub  …         0     NaN    NaN
4          Reg          Lvl    AllPub  …         0     NaN    NaN
…        …          …        …  …         …     …    …
2926       IR1          Lvl    AllPub  …         0     NaN  GdPrv
2927       IR1          Low    AllPub  …         0     NaN  MnPrv
2928       Reg          Lvl    AllPub  …         0     NaN  MnPrv
2929       Reg          Lvl    AllPub  …         0     NaN    NaN
2930       Reg          Lvl    AllPub  …         0     NaN    NaN

     Misc Feature Misc Val Mo Sold Yr Sold  Sale Type  Sale Condition  \
0             NaN        0       5    2010        WD          Normal
1             NaN        0       5    2010        WD          Normal
2             NaN        0       6    2010        WD          Normal
3            Gar2    12500       6    2010        WD          Normal
4             NaN        0       4    2010        WD          Normal
…            …      …      …      …         …            …
2926          NaN        0       3    2006        WD          Normal
2927          NaN        0       6    2006        WD          Normal
2928         Shed      700       7    2006        WD          Normal
2929          NaN        0       4    2006        WD          Normal
2930          NaN        0      11    2006        WD          Normal

      SalePrice
0        215000
1        215000
2        105000
3        172000
4        244000
…          …
2926     142500
2927     131000
2928     132000
2929     170000
2930     188000

[2931 rows x 81 columns]
```

Using this method, the entire column containing the null values will be dropped.

3. We can replace the missing values (zero, the mean, the median, etc.), using `fillna()` method.

[28]:
```
median = housing["Lot Frontage"].median()
median
```

[28]:
```
68.0
```

[29]:
```
housing["Lot Frontage"].fillna(median, inplace = True)
```

[30]:
```
housing.tail()
```

[30]:
```
      Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  \
2926   2926  923275080           80        RL          37.0      7937   Pave
2927   2927  923276100           20        RL          68.0      8885   Pave
2928   2928  923400125           85        RL          62.0     10441   Pave
2929   2929  924100070           20        RL          77.0     10010   Pave
2930   2930  924151050           60        RL          74.0      9627   Pave

     Alley Lot Shape Land Contour  … Pool Area Pool QC  Fence Misc Feature  \
2926   NaN       IR1          Lvl  …         0     NaN  GdPrv          NaN
2927   NaN       IR1          Low  …         0     NaN  MnPrv          NaN
2928   NaN       Reg          Lvl  …         0     NaN  MnPrv         Shed
2929   NaN       Reg          Lvl  …         0     NaN    NaN          NaN
2930   NaN       Reg          Lvl  …         0     NaN    NaN          NaN

     Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  SalePrice
2926        0       3    2006       WD           Normal     142500
2927        0       6    2006       WD           Normal     131000
2928      700       7    2006       WD           Normal     132000
2929        0       4    2006       WD           Normal     170000
2930        0      11    2006       WD           Normal     188000

[5 rows x 82 columns]
```

Index# 2927, containing a missing value in the “Lot Frontage”, now has been replaced with the median value.

1.12 Exercise 4

In this exercise, let’s look at ‘Mas Vnr Area’ feature and replace the missing values with the mean value of that column.

[32]:
```
# Enter your code and run the cell
mean = housing['Mas Vnr Area'].mean()
housing['Mas Vnr Area'].fillna(mean, inplace= True)
housing.tail()
```

[32]:
```
      Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  \
2926   2926  923275080           80        RL          37.0      7937   Pave
2927   2927  923276100           20        RL          68.0      8885   Pave
```

```
2928  2928  923400125          85        RL          62.0     10441   Pave
2929  2929  924100070          20        RL          77.0     10010   Pave
2930  2930  924151050          60        RL          74.0      9627   Pave

     Alley Lot Shape Land Contour  … Pool Area Pool QC  Fence Misc Feature  \
2926   NaN       IR1          Lvl  …         0     NaN  GdPrv          NaN
2927   NaN       IR1          Low  …         0     NaN  MnPrv          NaN
2928   NaN       Reg          Lvl  …         0     NaN  MnPrv         Shed
2929   NaN       Reg          Lvl  …         0     NaN    NaN          NaN
2930   NaN       Reg          Lvl  …         0     NaN    NaN          NaN

     Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  SalePrice
2926        0       3    2006       WD          Normal     142500
2927        0       6    2006       WD          Normal     131000
2928      700       7    2006       WD          Normal     132000
2929        0       4    2006       WD          Normal     170000
2930        0      11    2006       WD          Normal     188000

[5 rows x 82 columns]
```

Solution (Click Here)

    <code>

mean = housing["Mas Vnr Area"].mean() housing["Mas Vnr Area"].fillna(mean, inplace = True)

1.13 Feature Scaling

One of the most important transformations we need to apply to our data is feature scaling. There are two common ways to get all attributes to have the same scale: min-max scaling and standardization.

Min-max scaling (or normalization) is the simplest: values are shifted and rescaled so they end up ranging from 0 to 1. This is done by subtracting the min value and dividing by the max minus min.

Standardization is different: first it subtracts the mean value (so standardized values always have a zero mean), and then it divides by the standard deviation, so that the resulting distribution has unit variance.

Scikit-learn library provides `MinMaxScaler` for normalization and `StandardScaler` for standardization needs. For more information on `scikit-learn` `MinMaxScaler` and `StandardScaler` please visit their respective documentation websites.

First, we will normalize our data.

[33]:
```
norm_data = MinMaxScaler().fit_transform(hous_num)
norm_data
```

[33]:
```
array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, …,
        3.63636364e-01, 1.00000000e+00, 2.72444089e-01],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, …,
        3.63636364e-01, 1.00000000e+00, 2.72444089e-01],
       [3.41413452e-04, 1.01788895e-04, 0.00000000e+00, …,
        4.54545455e-01, 1.00000000e+00, 1.24238256e-01],
       …,
       [9.99317173e-01, 8.25914814e-01, 3.82352941e-01, …,
        5.45454545e-01, 0.00000000e+00, 1.60616051e-01],
       [9.99658587e-01, 8.27370610e-01, 0.00000000e+00, …,
        2.72727273e-01, 0.00000000e+00, 2.11814430e-01],
       [1.00000000e+00, 8.27476641e-01, 2.35294118e-01, …,
        9.09090909e-01, 0.00000000e+00, 2.36066294e-01]])
```

Note the data is now a `ndarray`

we can also standardize our data.

[34]:
```
scaled_data = StandardScaler().fit_transform(hous_num)
scaled_data
```

[34]:
```
array([[-1.73027969, -0.99682434, -0.87674019, …, -0.44796566,
         1.67740664,  0.4281423 ],
       [-1.73027969, -0.99682434, -0.87674019, …, -0.44796566,
         1.67740664,  0.4281423 ],
       [-1.72909781, -0.99656498, -0.87674019, …, -0.07945953,
         1.67740664, -0.94923488],
       …,
       [ 1.729097  ,  1.10758639,  0.64804102, …,  0.2890466 ,
        -1.36026952, -0.61115139],
       [ 1.73027889,  1.11129572, -0.87674019, …, -0.81647179,
        -1.36026952, -0.13533019],
       [ 1.73146077,  1.11156589,  0.06158671, …,  1.76307112,
        -1.36026952,  0.09005881]])
```

1.14 Exercise 5

In this exercise, use `StandardScaler()` and `fit_transform()` functions to standardize the 'SalePrice' feature only.

[42]:
```
# Enter your code and run the cell
scale_test = StandardScaler().fit_transform(housing[['SalePrice']])
scale_test
```

[42]:
```
array([[ 0.4281423 ],
       [ 0.4281423 ],
       [-0.94923488],
       …,
```

```
       [-0.61115139],
       [-0.13533019],
       [ 0.09005881]])
```

Solution (Click Here)

`    <code>`

scaled_sprice = StandardScaler().fit_transform(housing['SalePrice'][:,np.newaxis]) scaled_sprice

1.15 Handling the Outliers

1.15.1 Finding the Outliers

In statistics, an outlier is an observation point that is distant from other observations. An outlier can be due to some mistakes in data collection or recording, or due to natural high variability of data points. How to treat an outlier highly depends on our data or the type of analysis to be performed. Outliers can markedly affect our models and can be a valuable source of information, providing us insights about specific behaviours.

There are many ways to discover outliers in our data. We can do Uni-variate analysis (using one variable analysis) or Multi-variate analysis (using two or more variables). One of the simplest ways to detect an outlier is to inspect the data visually, by making box plots or scatter plots.

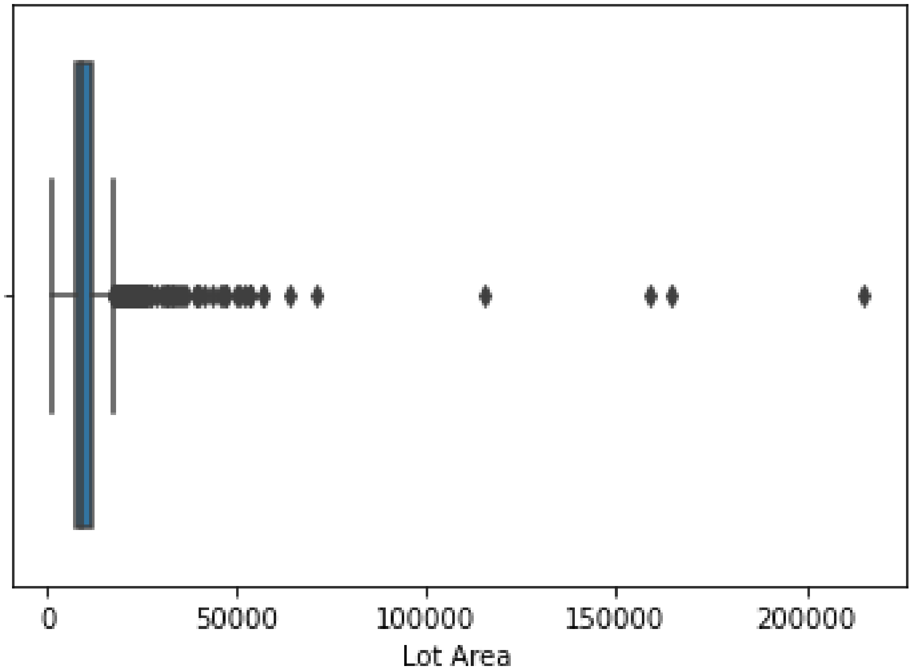
1.15.2 Uni-variate Analysis

A box plot is a method for graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending vertically from the boxes (whiskers) indicating variability outside the upper and lower quartiles. Outliers may be plotted as individual points. To learn more about box plots please click here.

Here, we will use a box plot for the 'Lot Area' and the 'SalePrice' features.

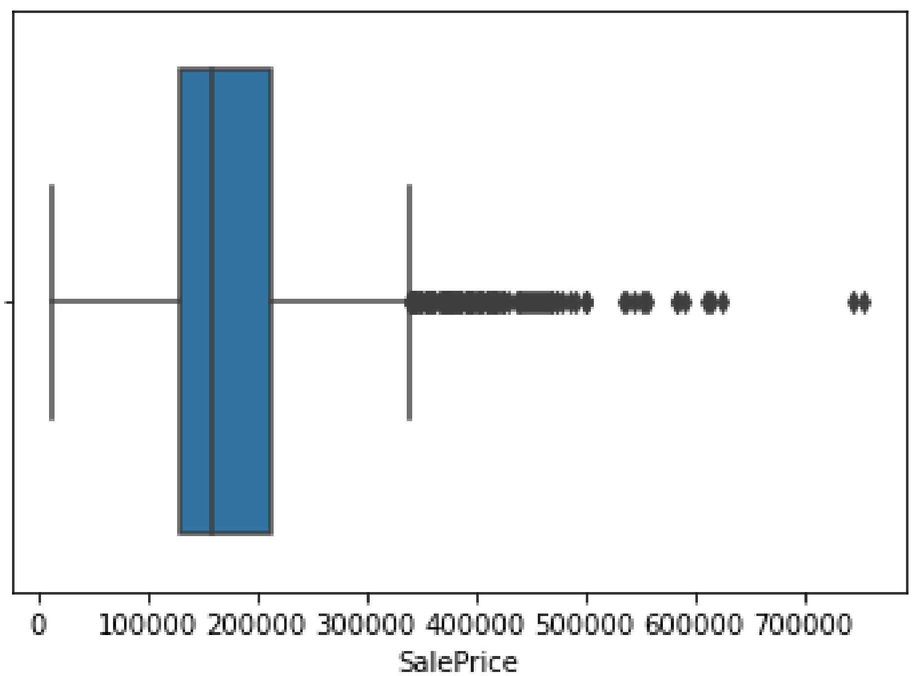
[43]:
```
sns.boxplot(x=housing['Lot Area'])
```

[43]:
```
<AxesSubplot:xlabel='Lot Area'>
```



[44]:
```
sns.boxplot(x=housing['SalePrice'])
```

[44]: <AxesSubplot:xlabel='SalePrice'>

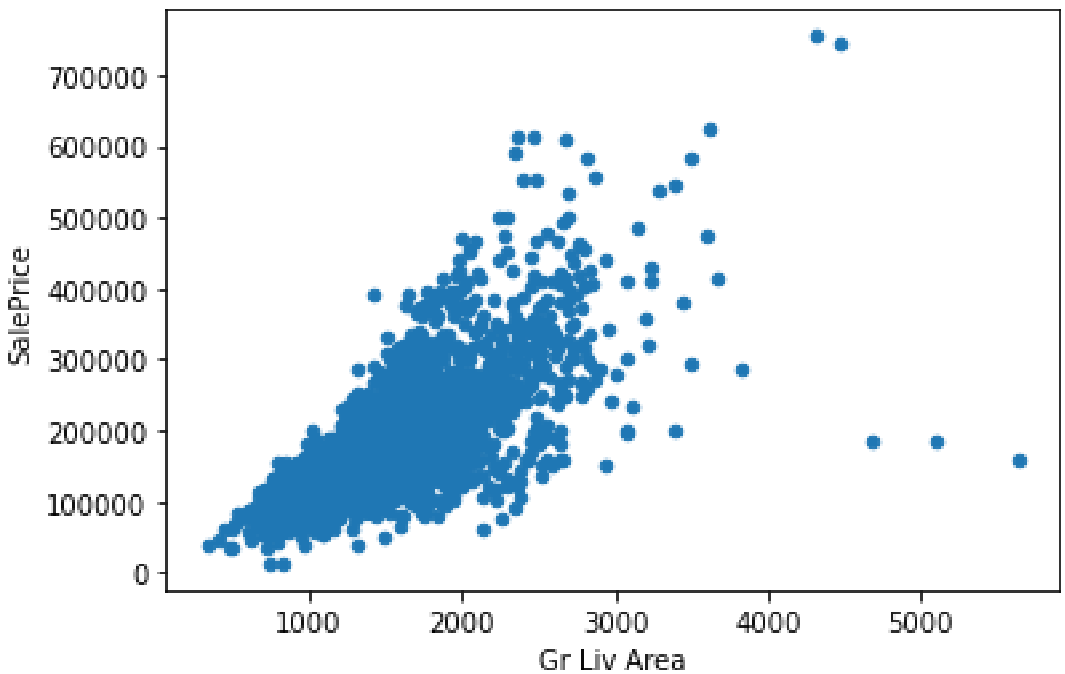


As we can see from these two plots, we have some points that are plotted outside the box plot area and that greatly deviate from the rest of the population. Whether to remove or keep them will greatly depend on the understanding of our data and the type of analysis to be performed. In this case, the points that are outside of our box plots in the 'Lot Area' and the 'Sale Price' might be the actual true data points and do not need to be removed.

1.15.3 Bi-variate Analysis

Next, we will look at the bi-variate analysis of the two features, the sale price, 'SalePrice', and the ground living area, 'GrLivArea', and plot the scatter plot of the relationship between these two parameters.

[45]:
```
price_area = housing.plot.scatter(x='Gr Liv Area',
                                  y='SalePrice')
```



From the above graph, there are two values above 5000 sq. ft. living area that deviate from the rest of the population and do not seem to follow the trend. It can be speculated why this is happening but for the purpose of this lab we can delete them.

The other two observations on the top are also deviating from the rest of the points but they also seem to be following the trend, so, perhaps, they can be kept.

1.15.4 Deleting the Outliers

First, we will sort all of our 'Gr Liv Area' values and select only the last two.

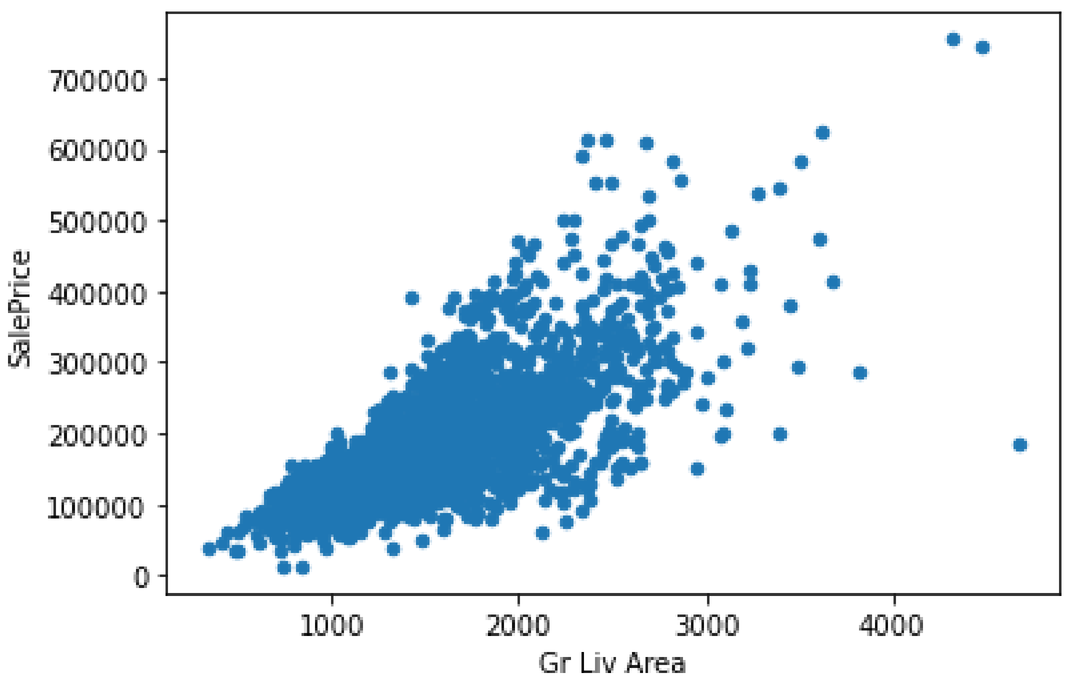
[46]:
```
housing.sort_values(by = 'Gr Liv Area', ascending = False)[:2]
```

[46]:
```
      Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  \
1499   1499  908154235           60        RL         313.0     63887   Pave
2181   2181  908154195           20        RL         128.0     39290   Pave

     Alley Lot Shape Land Contour  … Pool Area Pool QC Fence Misc Feature  \
1499   NaN       IR3          Bnk  …       480      Gd   NaN          NaN
2181   NaN       IR1          Bnk  …         0     NaN   NaN         Elev

     Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  SalePrice
1499        0       1    2008       New         Partial     160000
2181    17000      10    2007       New         Partial     183850

[2 rows x 82 columns]
```

Now we will use the pandas `drop()` function to remove these two rows.

[47]:
```
outliers_dropped = housing.drop(housing.index[[1499,2181]])
```

[48]:
```
new_plot = outliers_dropped.plot.scatter(x='Gr Liv Area',
                                         y='SalePrice')
```



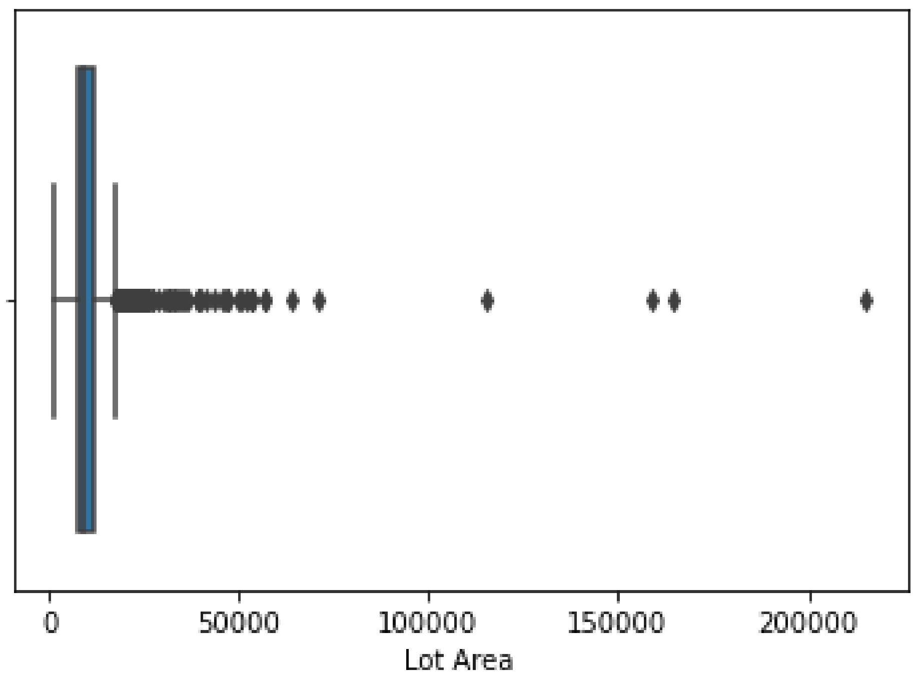
As you can see, we do not have the last two points of the 'Gr Liv Area' anymore.

1.16 Exercise 6

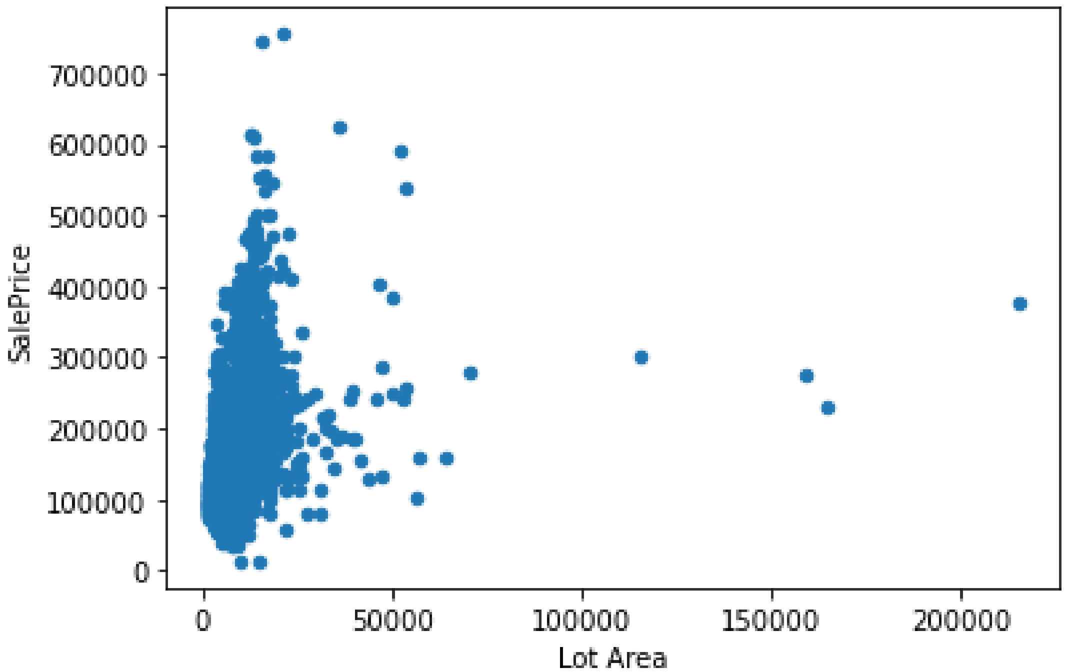
In this exercise, determine whether there are any outliers in the 'Lot Area' feature. You can either plot the box plot for the 'Lot Area', perform a bi-variate analysis by making a scatter plot between the 'SalePrice' and the 'Lot Area', or use the Z-score analysis. If there are any outliers, remove them from the dataset.

[51]:
```
# Enter your code and run the cell
sns.boxplot(x=housing['Lot Area'])
```

[51]:
```
<AxesSubplot:xlabel='Lot Area'>
```



[52]:
```
# bi-variate analysis
price_lot_area = housing.plot.scatter(x='Lot Area',
                                      y='SalePrice')
```



Solution (Click Here)

    <code>

sns.boxplot(x=housing[‘Lot Area’]) price_lot = housing.plot.scatter(x=‘Lot Area’, y=‘SalePrice’) housing[‘Lot_Area_Stats’] = stats.zscore(housing[‘Lot Area’]) housing[[‘Lot Area’,‘Lot_Area_Stats’]].describe().round(3) housing.sort_values(by = ‘Lot Area’, ascending = False)[:1] lot_area_rem = housing.drop(housing.index[[957]])

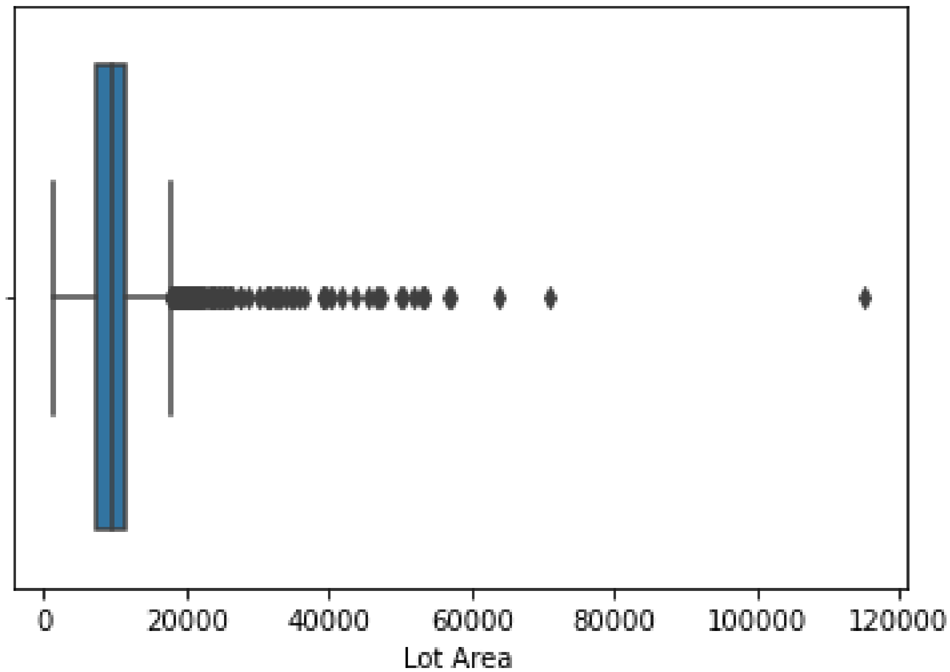
[53]:
```python
housing.sort_values(by = 'Lot Area', ascending = False)[:3]
```

[53]:
```
      Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  \
957     957  916176125           20        RL         150.0    215245   Pave
1571   1571  916125425          190        RL          68.0    164660   Grvl
2116   2116  906426060           50        RL          68.0    159000   Pave

     Alley Lot Shape Land Contour  … Pool Area Pool QC Fence Misc Feature  \
957    NaN       IR3          Low  …         0     NaN   NaN          NaN
1571   NaN       IR1          HLS  …         0     NaN   NaN         Shed
2116   NaN       IR2          Low  …         0     NaN   NaN         Shed

     Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  SalePrice
957         0       6    2009        WD          Normal     375000
1571      700       8    2008        WD          Normal     228950
2116      500       6    2007        WD          Normal     277000
```

```
[3 rows x 82 columns]
```

[54]:
```
outliers_dropped_2 = housing.drop(housing.index[[957,1571,2116]])
sns.boxplot(x=outliers_dropped_2['Lot Area'])
```

[54]:
```
<AxesSubplot:xlabel='Lot Area'>
```



Answer (Click Here)

`    <code>`

There seems to be one outlier, the very last point in the 'Lot Area' is too far from the rest of the group. Also, according to the Z-score, the standard deviation of that point exceeds the threshhold of 3.

1.17 Z-score Analysis

Z-score is another way to identify outliers mathematically. Z-score is the signed number of standard deviations by which the value of an observation or data point is above the mean value of what is being observed or measured. In another words, Z-score is the value that quantifies relationship between a data point and a standard deviation and mean values of a group of points. Data points which are too far from zero will be treated as the outliers. In most of the cases, a threshold of 3 or -3 is used. For example, if the Z-score value is greater than or less than 3 or -3 standard deviations respectively, that data point will be identified as a outlier.

To learn more about Z-score, please visit this Wikipedia site.

Below, we are using Z-score function from `scipy` library to detect the outliers in our 'Low Qual Fin SF' parameter. To learn more about `scipy.stats`, please visit this link.

[55]:
```
housing['LQFSF_Stats'] = stats.zscore(housing['Low Qual Fin SF'])
housing[['Low Qual Fin SF','LQFSF_Stats']].describe().round(3)
```

[55]:

	Low Qual Fin SF	LQFSF_Stats
count	2931.000	2931.000
mean	4.675	-0.000
std	46.303	1.000
min	0.000	-0.101
25%	0.000	-0.101
50%	0.000	-0.101
75%	0.000	-0.101
max	1064.000	22.882

The scaled results show a mean of 0.000 and a standard deviation of 1.000, indicating that the transformed values fit the z-scale model. The max value of 22.882 is further proof of the presence of outliers, as it falls well above the z-score limit of +3.

2 Congratulations! - You have completed the lab

2.1 Author

Svitlana Kramar

2.2 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-11-30	0.1	Svitlana	Added the Log Transformation section
2022-01-18	0.2	Svitlana	Added the Introduction