CSE 4/587 Spring 2023
Project Phase 1
Team
Name: Pavana Lakshmi Venugopal
UBIT: pavanala
Name: Vaidurya Malathesha
UBIT: vaidurya

Deliverables [50 marks total]

1. Problem Statement: Form a title and problem statement that clearly state the problem and questions you are trying to answer. Additionally:
a. [5 marks] Discuss the background of the problem leading to your objectives. Why is it a significant problem?
b. [5 marks] Explain the potential of your project to contribute to your problem domain. Discuss why this contribution is crucial?

Title: Analysis of property for taxes in West Roxbury.

Problem Statement: We'll do the analysis, search for the main reasons for the Total value to increase, and calculate the property taxes the owner must pay each year.

a. Background: Sending auditors to each property to access the property value is a cumbersome task, quite time consuming and expensive for the government. If we can save some amount for the government, they can be used for other essentials citizens might need. We will go through all the data points or values for West Roxbury, what features lead to the value of the property increasing or decreasing, based on that we would find out if we could predict the Total value and hence calculate the tax. If a website is created where homeowners add the details or update the details of the property regularly then this goal can be achieved. We will be analyzing the predictors to calculate the property value.

b. Potential: It would no longer be necessary for assessors to devote the considerable time inspecting each home, saving governments hundreds of thousands of dollars. Machine Learning models can be used to predict the total value of the homes.

The following are the questions that will be addressed:

Which data can be retained for Total value?

Does any data need modification for our models to interpret?

Which data does not contribute to the predictions?

Which are some of the main reasons for change in total value?

We would basically analyze the data, find which features or predictors are needed, to gain an understanding predict the response variable. Here it is Total value.

2. Data Sources [5 marks]: Collect your data. Your data can come from multiple sources. For example, Medical, Bank, sports, health, Kaggle, Amazon reviews, Twitter, Youtube, Reddit, etc. This data has to be large enough for the data analysis to yield significance. At least 2000 rows/records.

Solution:

Link to dataset: https://github.com/reisanar/datasets/blob/master/WestRoxbury.csv

3. Data Cleaning/Processing [10 Marks]: Your dataset has to be cleaned and properly processed. Please submit a report where you explain each processing/cleaning step properly. We expect to see comments and markup for this step. In order to get full marks you must clearly document 7 (10 for 587 students) distinct processing/cleaning operations.

1. **Renaming column names:**

   The column names had spaces in between them and after them. The spaces in between words are replaced with hyphen. This helps to easily understand and work with more clarity with the renamed columns.

```
In [197]: ds = dataset
          print(ds.columns)

          Index(['TOTAL VALUE ', 'TAX', 'LOT SQFT ', 'YR BUILT', 'GROSS AREA ',
                 'LIVING AREA', 'FLOORS ', 'ROOMS', 'BEDROOMS ', 'FULL BATH',
                 'HALF BATH', 'KITCHEN', 'FIREPLACE', 'REMODEL'],
                dtype='object')

In [198]: #1 renaming column names
          ds = ds.rename(columns={'TOTAL VALUE ': 'TOTAL_VALUE', 'LOT SQFT ': 'LOT_SQFT', 'YR BUILT': 'YR_BUILT', 'GROSS AREA ': 'GROSS_ARE
```

2. **Handling null/missing values:**

   Null/missing values adds bias to the data analysis and can cause errors in the calculations. In this step, we check and drop the rows having null or missing values.

We observed that the YR_BUILT column has a value of 0. And the same is removed from the dataset.

```
In [249]: ds['YR_BUILT'].value_counts()
Out[249]: 1920    566
          1950    478
          1930    430
          1960    356
          1925    302
                 ...
          1883      1
          1976      1
          0         1
          1874      1
          1800      1
          Name: YR_BUILT, Length: 149, dtype: int64

In [250]: ds = ds[ds.YR_BUILT != 0]

In [251]: ds['YR_BUILT'].value_counts()
Out[251]: 1920    566
          1950    478
          1930    430
          1960    356
          1925    302
                 ...
          1798      1
          1883      1
          1881      1
          1874      1
          1872      1
          Name: YR_BUILT, Length: 148, dtype: int64
```

3. **Identify and remove any duplicated rows in the dataset.**

Removing duplicates is another step in data cleaning to avoid skew statistical analysis and improve the integrity of the dataset.
We found that three rows in the dataset were duplicated when checked against the TAX column. And they are dropped from the dataset.

```
In [215]: #3 Identify and remove any duplicated rows in the dataset
          dup = ds[ds.duplicated()]
          print("duplicated rows:{}".format(dup))
          ds.drop_duplicates(inplace=True)
          print("After dropping, rows=", len(ds.TAX))

          duplicated rows:       TOTAL_VALUE   TAX  LOT_SQFT  YR_BUILT  GROSS_AREA  LIVING_AREA  FLOORS  \
          1178         564.8  7105      6000      2005        4398         2543     2.0
          3894         582.8  7331      6009      2004        3826         2341     2.0
          5227         620.4  7804      5000      2004        4149         2516     2.0

                ROOMS  BEDROOMS  FULL_BATH  HALF_BATH  KITCHEN  FIREPLACE REMODEL
          1178      8         4          2          2        1          1    None
          3894      7         4          2          1        1          1    None
          5227      7         4          3          1        1          2    None
          After dropping, rows= 5799
```

4. Examine feature variance

By examining the feature variance, we can make sure that the dataset is not of poor quality, dimensionality is maintained, and model performance can be improved by removing features having low variance, thus improving the interpretability of the data. With our chosen dataset, we didn't need to eliminate any features because TOTAL_VALUE, LOT_SQFT variance are justifiable. And their variance won't affect analysis rather is part of the analysis.

In [216]:
```
#4 Examine feature varience
pd.set_option('display.max_columns', None)
ds.describe()
ds.round(2)
```

Out[216]:

	TOTAL_VALUE	TAX	LOT_SQFT	YR_BUILT	GROSS_AREA	LIVING_AREA	FLOORS	ROOMS	BEDROOMS	FULL_BATH	HALF_BATH	KITCHEN	FIREPL.
0	344.2	4330	9965	1880	2436	1352	2.0	6	3	1	1	1	
1	412.6	5190	6590	1945	3108	1976	2.0	10	4	2	1	1	
2	330.1	4152	7500	1890	2294	1371	2.0	8	4	1	1	1	
3	498.6	6272	13773	1957	5032	2608	1.0	9	5	1	1	1	
4	331.5	4170	5000	1910	2370	1438	2.0	7	3	2	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
5797	404.8	5092	6762	1938	2594	1714	2.0	9	3	2	1	1	
5798	407.9	5131	9408	1950	2414	1333	2.0	6	3	1	1	1	
5799	406.5	5113	7198	1987	2480	1674	2.0	7	3	1	1	1	
5800	308.7	3883	6890	1946	2000	1000	1.0	5	2	1	0	1	
5801	447.6	5630	7406	1950	2510	1600	2.0	7	3	1	1	1	

5799 rows × 14 columns

5. Drop irrelevant columns

Having irrelevant data can add noise and further decrease efficiency and accuracy of the predictions.
In our dataset, GROSS_AREA wouldn't add any significance to decision making process. And there is LOT_SQFT which would add more weightage than GROSS_AREA in correlating with other features. And the column TAX also doesn't add any value to the predictions. So, in this step we are dropping GROSS_AREA and TAX columns

```
In [305]: #5 Drop irrelevant column - adds no value to the analysis
          ds = ds.drop('GROSS_AREA', axis=1)
```

```
In [306]: ds = ds.drop('TAX', axis=1)
```

```
In [307]: ds
```

Out[307]:

	TOTAL_VALUE	LOT_SQFT	YR_BUILT	LIVING_AREA	FLOORS	ROOMS	BEDROOMS	FULL_BATH	HALF_BATH	KITCHEN	FIREPLACE	REMODEL
0	344.2	9965	1880	1352	2.0	6	3	1	1	1	0	None
1	412.6	6590	1945	1976	2.0	10	4	2	1	1	0	Recent
2	330.1	7500	1890	1371	2.0	8	4	1	1	1	0	None
3	498.6	13773	1957	2608	1.0	9	5	1	1	1	1	None
4	331.5	5000	1910	1438	2.0	7	3	2	0	1	0	None
...	...	...	...	...	...	...	...	...	...	...	...	...
5797	404.8	6762	1938	1714	2.0	9	3	2	1	1	1	Recent
5798	407.9	9408	1950	1333	2.0	6	3	1	1	1	1	None
5799	406.5	7198	1987	1674	2.0	7	3	1	1	1	1	None
5800	308.7	6890	1946	1000	1.0	5	2	1	0	1	0	None
5801	447.6	7406	1950	1600	2.0	7	3	1	1	1	1	None

5798 rows × 12 columns

6. Encode categorical data

This is another important data preprocessing step as it simplifies data analysis, reduces memory usage and improves accuracy.
The column REMODEL has three values – Old, Recent, None. We encoded this column into categorical data. Converting it to values that is integer the machine can understand.

```
In [219]: #6 Encode categorical data
          ds = pd.get_dummies(ds,columns=['REMODEL'])
```

```
In [220]: ds = pd.get_dummies(ds, drop_first = True)
```

```
In [221]: ds
```

LOT_SQFT	YR_BUILT	LIVING_AREA	FLOORS	ROOMS	BEDROOMS	FULL_BATH	HALF_BATH	KITCHEN	FIREPLACE	REMODEL_None	REMODEL_Old	REMODEL_Recent
9965	1880	1352	2.0	6	3	1	1	1	0	1	0	0
6590	1945	1976	2.0	10	4	2	1	1	0	0	0	1
7500	1890	1371	2.0	8	4	1	1	1	0	1	0	0
13773	1957	2608	1.0	9	5	1	1	1	1	1	0	0
5000	1910	1438	2.0	7	3	2	0	1	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
6762	1938	1714	2.0	9	3	2	1	1	1	0	0	1
9408	1950	1333	2.0	6	3	1	1	1	1	1	0	0
7198	1987	1674	2.0	7	3	1	1	1	1	1	0	0
6890	1946	1000	1.0	5	2	1	0	1	0	1	0	0
7406	1950	1600	2.0	7	3	1	1	1	1	1	0	0

7. Removing outliers

Outliers impact the visualizing of data and make it difficult to identify patterns and trends. It captures the measurement errors. So, it's important to remove outliers to improve accuracy of the analysis.

The TOTAL_VALUE isn't proportionate in the way the LOT_SQFT changes for the house with 13 and 14 rooms for the same YR_BUILT. Due to this inconsistency, in this step we will be dropping the rows having 14 rooms.

1	TOTAL VALUE	TAX	LOT SQFT	YR BUILT	GROSS AREA	LIVING AREA	FLOORS	ROOMS	BEDROOM	FULL BATH	HALF BATH	KITCHEN	FIREPLACE	REMODEL
2457	597.3	7514	6900	1919	5243	2926	2	14	6	3	1	1	3	Old
2496	660	8302	13650	1924	5192	2640	2	13	7	2	1	1	2	Recent
2608	658.7	8286	18288	1888	5698	3613	2.5	13	5	1	1	1	1	None
2996	721.6	9077	5505	1900	5806	3626	2	14	7	3	1	1	2	Recent
3060	740.6	9316	16576	1920	7187	4288	2	14	8	3	0	1	2	None
3068	462.2	5814	6625	1880	4074	2349	2	13	6	1	2	1	0	None
3234	668.7	8412	20897	1910	7175	3977	2	13	4	1	1	1	1	Old
3566	453.6	5706	5000	1910	5401	2951	2	13	4	2	0	1	0	None
3637	599.2	7537	7496	1920	4408	2677	2.5	13	4	2	2	1	1	None
3840	562.8	7080	9018	1900	4686	2718	2	14	5	3	0	1	1	Recent
3852	410.8	5167	5293	1900	3948	2071	2	13	6	1	1	1	1	None
4279	538	6768	8348	1964	3218	1914	2	14	4	2	2	1	1	None
5658	555.8	6991	7235	1949	4221	2484	1	13	4	2	1	2	1	Recent
5804														

```
In [222]: #7 removing outliers

          ds = ds[ds.ROOMS != 14]

In [223]: ds
Out[223]:
```

	TOTAL_VALUE	TAX	LOT_SQFT	YR_BUILT	LIVING_AREA	FLOORS	ROOMS	BEDROOMS	FULL_BATH	HALF_BATH	KITCHEN	FIREPLACE	REMODEL
0	344.2	4330	9965	1880	1352	2.0	6	3	1	1	1	0	
1	412.6	5190	6590	1945	1976	2.0	10	4	2	1	1	0	
2	330.1	4152	7500	1890	1371	2.0	8	4	1	1	1	0	
3	498.6	6272	13773	1957	2608	1.0	9	5	1	1	1	1	
4	331.5	4170	5000	1910	1438	2.0	7	3	2	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
5797	404.8	5092	6762	1938	1714	2.0	9	3	2	1	1	1	
5798	407.9	5131	9408	1950	1333	2.0	6	3	1	1	1	1	
5799	406.5	5113	7198	1987	1674	2.0	7	3	1	1	1	1	
5800	308.7	3883	6890	1946	1000	1.0	5	2	1	0	1	0	
5801	447.6	5630	7406	1950	1600	2.0	7	3	1	1	1	1	

5794 rows × 15 columns

Till this step, we are confident that our data is cleaned enough to proceed further with EDA. The following steps are tried to check if they will make any more attempts to get the dataset cleaned and processed.

8. Data Normalization – MinMaxScaling()

Min-Max scaling is one of the common techniques used for data normalization. It equalizes the importance of all features by scaling the features to same range, and thus improving the accuracy of model predictions.
For this step, we scaled on the features TOTAL_VALUE and LOT_SQFT.
We will compare how the model performs with and without this normalization in the upcoming project phase.

```
In [170]: #8 normalization - min-max scaling

          from sklearn.preprocessing import MinMaxScaler

          # Creating an object for MinMaxScaler
          mms = MinMaxScaler()
          df = ds
          # Scaling secific columns
          df[['TOTAL_VALUE','LOT_SQFT']] = mms.fit_transform(df[['TOTAL_VALUE','LOT_SQFT']])

          # Display Modified DataFrame
          print("Modified DataFrame:\n",df)
```

```
Modified DataFrame:
       TOTAL_VALUE  LOT_SQFT  YR_BUILT  LIVING_AREA  FLOORS  ROOMS  BEDROOMS  \
0        0.214953  0.197472      1880         1352     2.0      6         3
1        0.276420  0.123156      1945         1976     2.0     10         4
2        0.202283  0.143194      1890         1371     2.0      8         4
3        0.353702  0.281323      1957         2608     1.0      9         5
4        0.203541  0.088145      1910         1438     2.0      7         3
...           ...       ...       ...          ...     ...    ...       ...
5797     0.269410  0.126943      1938         1714     2.0      9         3
5798     0.272196  0.185207      1950         1333     2.0      6         3
5799     0.270938  0.136544      1987         1674     2.0      7         3
5800     0.183052  0.129762      1946         1000     1.0      5         2
5801     0.307872  0.141124      1950         1600     2.0      7         3

      FULL_BATH  HALF_BATH  KITCHEN  FIREPLACE  REMODEL_None  REMODEL_Old  \
0             1          1        1          0             1            0
1             2          1        1          0             0            0
2             1          1        1          0             1            0
3             1          1        1          1             1            0
4             2          0        1          0             1            0
...         ...        ...      ...        ...           ...          ...
5797          2          1        1          1             0            0
5798          1          1        1          1             1            0
5799          1          1        1          1             1            0
5800          1          0        1          0             1            0
5801          1          1        1          1             1            0
```

9. Data Standardization - Z-score

Z-score data normalization is used to transform data so that it has a mean of zero and a standard deviation of one. The purpose of this transform is to get data to a standard scale that is easier to work with in statistical analysis and machine learning models.
In this step, after applying the Z-score transformation, we see no outliers or extreme values in the dataset.

```
In [171]: #9 standardization - z-score
          from sklearn.preprocessing import StandardScaler

          # create a scaler object
          std_scaler = StandardScaler()

          df = ds
          # Scaling secific columns
          df[['TOTAL_VALUE','LOT_SQFT']] = std_scaler.fit_transform(df[['TOTAL_VALUE','LOT_SQFT']])

          # Display Modified DataFrame
          print("Modified DataFrame:\n",df)

          Modified DataFrame:
                TOTAL_VALUE  LOT_SQFT  YR_BUILT  LIVING_AREA  FLOORS  ROOMS  BEDROOMS  \
          0       -0.487139  1.382989      1880         1352     2.0      6         3
          1        0.204918  0.117823      1945         1976     2.0     10         4
          2       -0.629800  0.458949      1890         1371     2.0      8         4
          3        1.075048  2.810471      1957         2608     1.0      9         5
          4       -0.615635 -0.478211      1910         1438     2.0      7         3
          ...          ...       ...       ...          ...     ...    ...       ...
          5797     0.125999  0.182299      1938         1714     2.0      9         3
          5798     0.157364  1.174190      1950         1333     2.0      6         3
          5799     0.143199  0.345740      1987         1674     2.0      7         3
          5800    -0.846321  0.230282      1946         1000     1.0      5         2
          5801     0.559041  0.423712      1950         1600     2.0      7         3

                FULL_BATH  HALF_BATH  KITCHEN  FIREPLACE  REMODEL_None  REMODEL_Old  \
          0             1          1        1          0             1            0
          1             2          1        1          0             0            0
          2             1          1        1          0             1            0
          3             1          1        1          1             1            0
          4             2          0        1          0             1            0
          ...         ...        ...      ...        ...           ...          ...
          5797          2          1        1          1             0            0
          5798          1          1        1          1             1            0
          5799          1          1        1          1             1            0
          5800          1          0        1          0             1            0
          5801          1          1        1          1             1            0
```

10. Handling inconsistencies

To identify and handle inconsistency in the data set is important.
As our data is numerical based, for this step, we checked if all column's values adhere to the same datatypes like float,int.

```
In [361]: #10 handling inconsistencies

          ds['TOTAL_VALUE'].dtypes

Out[361]: dtype('float64')

In [362]: ds['LOT_SQFT'].dtypes

Out[362]: dtype('float64')

In [363]: ds['YR_BUILT'].dtypes

Out[363]: dtype('int64')

In [366]: ds['FLOORS'].dtypes

Out[366]: dtype('float64')

In [367]: ds['LIVING_AREA'].dtypes

Out[367]: dtype('int64')

In [368]: ds['BEDROOMS'].dtypes

Out[368]: dtype('int64')

In [369]: ds['FULL_BATH'].dtypes

Out[369]: dtype('int64')
```

11. Data Discretization – Binning qcut

Data discretization is the process of transforming continuous variables into discrete variables by dividing the range of data into bins.
Quantile based binning or qcut is used to create bins with an equal number of observations in each bin.

YR_BUILT is years used as measure of time, thus can be considered as continuous variables. And binning it gives the following output.

```
In [373]: #11 binning - qcut

          ds['YR_BUILT'].describe()

Out[373]: count    5793.000000
          mean     1937.058174
          std        25.426573
          min      1798.000000
          25%      1920.000000
          50%      1935.000000
          75%      1955.000000
          max      2011.000000
          Name: YR_BUILT, dtype: float64
```

We can see that the column contains 5793 observations, with a mean value of 1937.058174 and a standard deviation of 25.426573. The minimum value in the column is 1798, while the maximum value is 2011. The quartile values show that 25% of the observations were built before 1920, 50% were built before 1935, and 75% were built before 1955. This information can be useful for further analysis and interpretation of the data.

12. Data Transformation

Splitting data into training and testing data is a form of data transformation. The process of splitting the data typically involves randomly selecting a portion of the data to be used as the testing dataset, while the remaining data is used as the training dataset. The proportion of data used for testing and training may vary depending on the size of the dataset and the complexity of the problem being solved.

For our selected dataset, we split the 95% as training data and the rest 5% as test data.

In [376]:
```python
#12 Data Transformation - split data as train and test

from sklearn.model_selection import train_test_split

df = ds
# get the locations
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# split the dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.05, random_state=0)
```

In [380]:
```python
X_train
```

Out[380]:

	TOTAL_VALUE	LOT_SQFT	YR_BUILT	LIVING_AREA	FLOORS	ROOMS	BEDROOMS	FULL_BATH	HALF_BATH	KITCHEN	FIREPLACE	REMODEL_None
898	0.219626	0.089224	1956	1506	1.5	6	3	1	0	1	1	1
1190	0.161125	0.040428	1935	1440	2.0	7	3	1	0	1	0	1
5314	0.235262	0.143194	1945	1583	1.0	6	3	2	0	1	1	1
2533	0.242631	0.115669	1920	1861	2.0	7	4	1	1	1	0	1
4473	0.285047	0.080680	1931	1536	2.0	7	3	1	1	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...
4939	0.266175	0.093650	1929	1732	2.0	7	3	1	1	1	1	1
3269	0.313354	0.116836	1910	2906	2.0	9	4	1	1	1	1	1
1655	0.229062	0.149712	1860	1808	2.0	8	3	2	0	1	0	0
2610	0.252336	0.218611	1950	1127	1.0	5	2	2	0	1	1	1
2735	0.339594	0.196415	1922	2304	2.0	8	4	1	1	1	2	1

5503 rows × 18 columns

In [378]:
```python
X_test
```

Out[378]:

	TOTAL_VALUE	LOT_SQFT	YR_BUILT	LIVING_AREA	FLOORS	ROOMS	BEDROOMS	FULL_BATH	HALF_BATH	KITCHEN	FIREPLACE	REMODEL_None
1842	0.387132	0.100850	1924	2988	2.0	10	4	2	1	1	2	0
1828	0.243081	0.166601	1928	1846	2.0	8	4	1	0	1	0	1
5777	0.228433	0.086471	1930	1376	2.0	6	3	1	0	1	1	1
2032	0.109792	0.079293	1910	1470	2.0	7	3	1	0	1	1	1

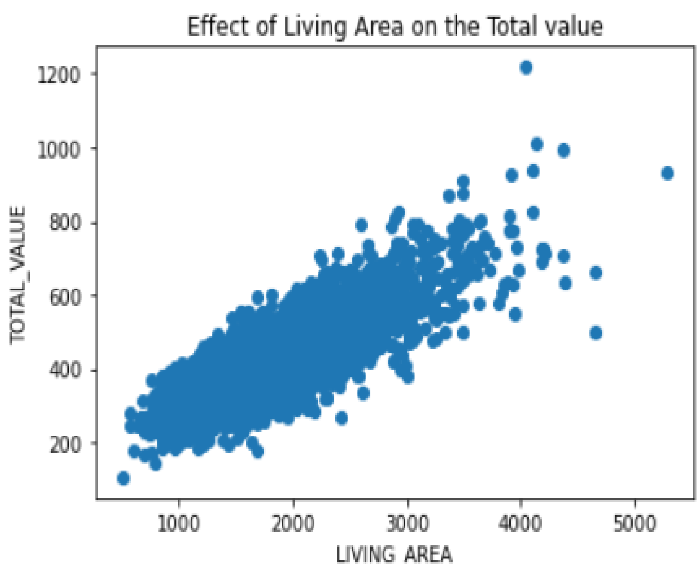
4. Exploratory Data Analysis (EDA) [25 Marks]: Perform exploratory data analysis as defined in the NIST publication [2] and as originally described by John Tukey [3]. Record the outcomes and what you learned and how you will use this information. For example, in choosing features (columns) and dropping columns, and in short feature engineering. You need to demonstrate 7 (10 for 587 students) different, significant and relevant EDA operations and describe how you used these to process the data sets further to provision them for downstream modeling and analytics. Figures and tables should be included where relevant.

Exploratory Data Analysis

1. Analysis of Living Area

- Here we observe how Living area effects the total value of the property.
- For most of the data we see as the Square feet is increasing the total value also increases with few outliers.
- This might still be used for prediction.
- The below scatter plot shows the relation.

```
plt.scatter(ds['LIVING_AREA'],ds['TOTAL_VALUE'])
plt.xlabel("LIVING_AREA")
plt.ylabel("TOTAL_VALUE")
plt.title("Effect of Living Area on the Total value")
plt.show()
```



2. Correlation between the features

- Using the correlation matrix to analyze the significance of attributes.
- Below image shows Correlation between the features/attributes.
- Generally, we would consider a correlation with value 0.9 or higher. We do not see any in this after analyzing.

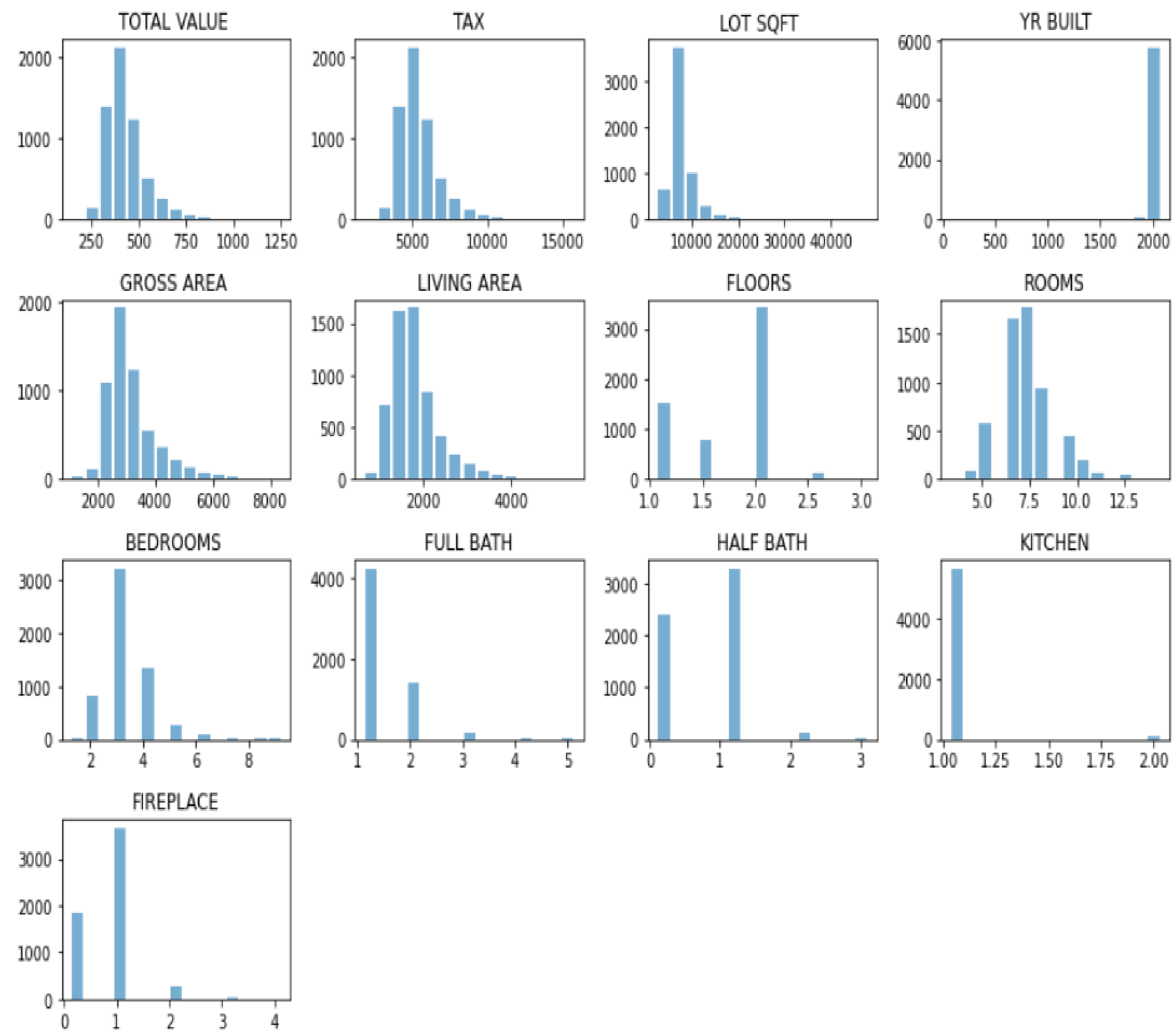
```
Co = ds.corr()
Co.style.background_gradient(cmap='PuBuGn')
```

	TOTAL VALUE	TAX	LOT SQFT	YR BUILT	GROSS AREA	LIVING AREA	FLOORS	ROOMS	BEDROOMS	FULL BATH	HALF BATH	KITCHEN	FIREPLACE
TOTAL VALUE	1.000000	1.000000	0.546123	-0.100917	0.800519	0.837120	0.481523	0.638539	0.561871	0.432807	0.348167	0.018265	0.358567
TAX	1.000000	1.000000	0.546120	-0.100918	0.800518	0.837122	0.481524	0.638542	0.561872	0.432806	0.348165	0.018261	0.358566
LOT SQFT	0.546123	0.546120	1.000000	-0.068908	0.448880	0.426045	0.073662	0.308395	0.254106	0.201317	0.134996	0.044525	0.181879
YR BUILT	-0.100917	-0.100918	-0.068908	1.000000	-0.167928	-0.131274	-0.190453	-0.144686	-0.130411	0.073706	0.060685	0.052091	0.087234
GROSS AREA	0.800519	0.800518	0.448880	-0.167928	1.000000	0.899775	0.300666	0.651501	0.571791	0.419734	0.226683	0.030501	0.270080
LIVING AREA	0.837120	0.837122	0.426045	-0.131274	0.899775	1.000000	0.475824	0.720671	0.641041	0.437987	0.301098	0.082825	0.262159
FLOORS	0.481523	0.481524	0.073662	-0.190453	0.300666	0.475824	1.000000	0.432856	0.431242	0.112166	0.316142	-0.114602	0.120506
ROOMS	0.638539	0.638542	0.308395	-0.144686	0.651501	0.720671	0.432856	1.000000	0.710693	0.378274	0.282655	0.129223	0.205223
BEDROOMS	0.561871	0.561872	0.254106	-0.130411	0.571791	0.641041	0.431242	0.710693	1.000000	0.332620	0.256852	0.085353	0.164380
FULL BATH	0.432807	0.432806	0.201317	0.073706	0.419734	0.437987	0.112166	0.378274	0.332620	1.000000	-0.130628	0.146650	0.140160
HALF BATH	0.348167	0.348165	0.134996	0.060685	0.226683	0.301098	0.316142	0.282655	0.256852	-0.130628	1.000000	-0.020071	0.176234
KITCHEN	0.018265	0.018261	0.044525	0.052091	0.030501	0.082825	-0.114602	0.129223	0.085353	0.146650	-0.020071	1.000000	-0.009562
FIREPLACE	0.358567	0.358566	0.181879	0.087234	0.270080	0.262159	0.120506	0.205223	0.164380	0.140160	0.176234	-0.009562	1.000000

3. Data Distribution

- Analyzing distribution of data
- Below histogram is to analyze distribution of data.
- Here we can see that Living area has a major impact on Total value compared to other columns.

```
hist = ds.hist(bins=15,figsize=(12, 8),grid = False,rwidth = 0.8,align='right',histtype= 'barstacked',alpha=0.6,)
plt.title('Distribution of all the columns')
plt.tight_layout()
```



4. Summary statistics of the dataset.

- This provides a descriptive statistic for the data set being used.
- Count: Returns the number of non-null values in the dataset.
- Mean: The average value of the dataset.
- Standard Deviation: Measure of the spread.
- Minimum: Represents the minimum value in the dataset.
- 25th Percentile: The value at which 25% of the data is below.
- 50th Percentile (Median): Value at which 50% of the data is below.
- 75th Percentile: The value at which 75% of the data is below.
- Maximum: Represents the maximum value in the dataset.

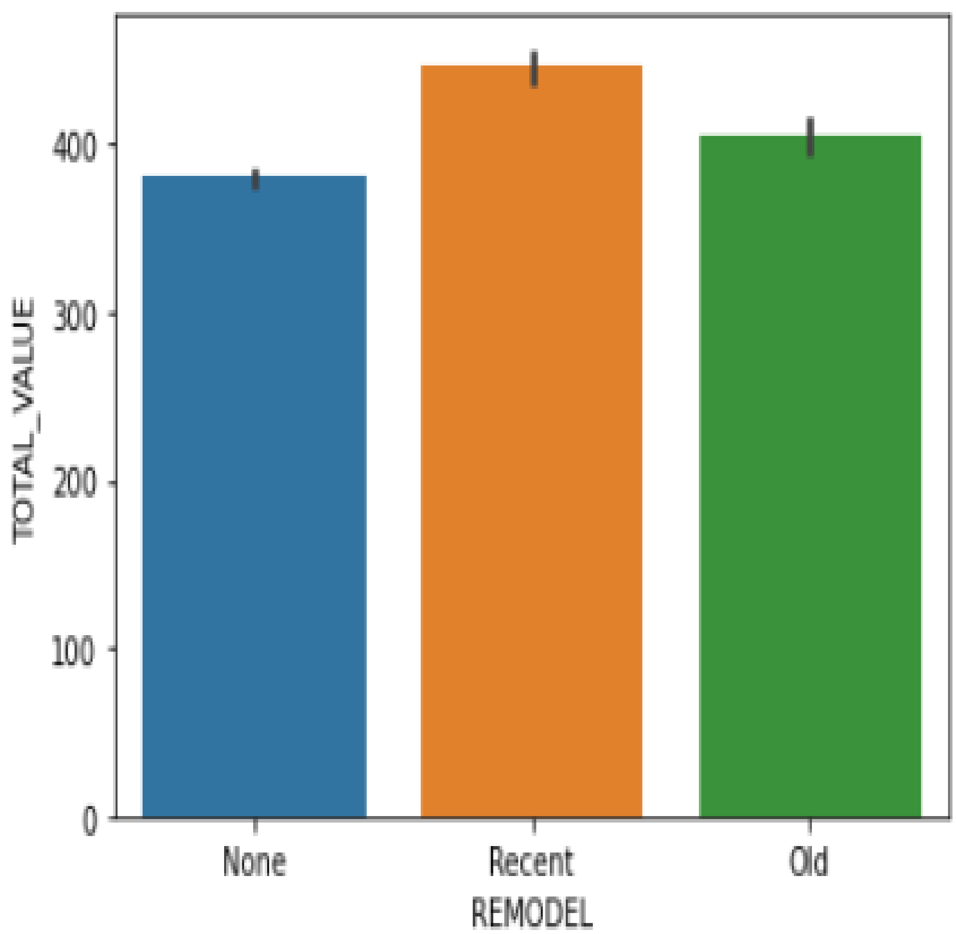
```
print(ds.describe())
```

```
       TOTAL_VALUE          TAX      LOT_SQFT     YR_BUILT  LIVING_AREA  \
count  5793.000000  5793.000000   5793.000000  5793.000000  5793.000000
mean    392.346801     0.258249   6275.692387  1937.058174  1655.235629
std      98.844294     0.088825   2667.863747    25.426573   538.224005
min     105.000000     0.000000    997.000000  1798.000000   504.000000
25%     325.000000     0.197728   4770.000000  1920.000000  1306.000000
50%     375.800000     0.243375   5682.000000  1935.000000  1546.000000
75%     438.400000     0.299664   7020.000000  1955.000000  1872.000000
max    1217.800000     1.000000  46411.000000  2011.000000  5289.000000

            FLOORS        ROOMS     BEDROOMS    FULL_BATH    HALF_BATH  \
count  5793.000000  5793.000000  5793.000000  5793.000000  5793.000000
mean      1.683238     6.988434     3.227171     1.294839     0.613326
std       0.445055     1.423839     0.842074     0.519711     0.533390
min       1.000000     3.000000     1.000000     1.000000     0.000000
25%       1.000000     6.000000     3.000000     1.000000     0.000000
50%       2.000000     7.000000     3.000000     1.000000     1.000000
75%       2.000000     8.000000     4.000000     2.000000     1.000000
max       3.000000    13.000000     9.000000     5.000000     3.000000

           KITCHEN    FIREPLACE
count  5793.000000  5793.000000
mean      1.015363     0.738650
std       0.123004     0.563985
min       1.000000     0.000000
25%       1.000000     0.000000
50%       1.000000     1.000000
75%       1.000000     1.000000
max       2.000000     4.000000
```

5. Remodel Analysis.

- From the below bar plot, we can infer that the recently remodeled houses cost the most.
- Around 470 thousand dollars.
- If a house did not have any remodeling or modification done it cost the least. Approximately 475 thousand dollars.

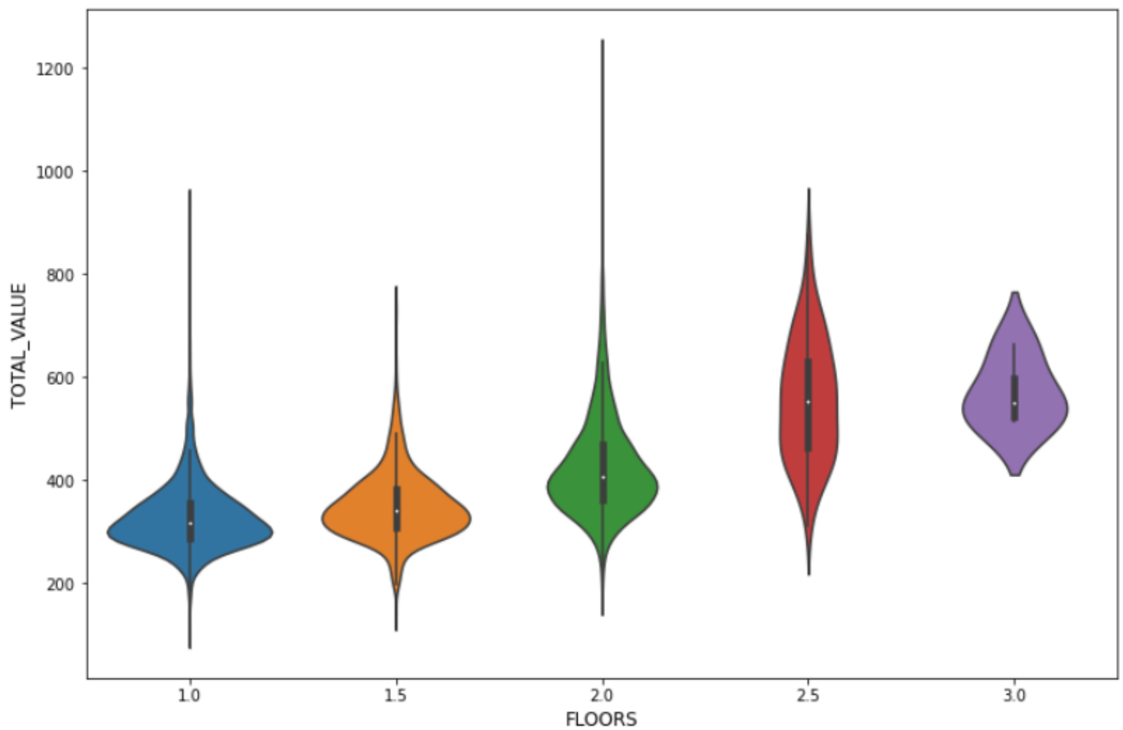
```
sns.barplot(x="REMODEL", y="TOTAL_VALUE", data=ds)
plt.show()
```



6.Floor Analysis.

- The characteristics of a box plot and a kernel density plot are combined to create a violin plot, a type of data visualization.
- The distribution of a numerical variable over various categories is depicted in this fashion.
- Here the distribution of floors data is showed in violin shape.

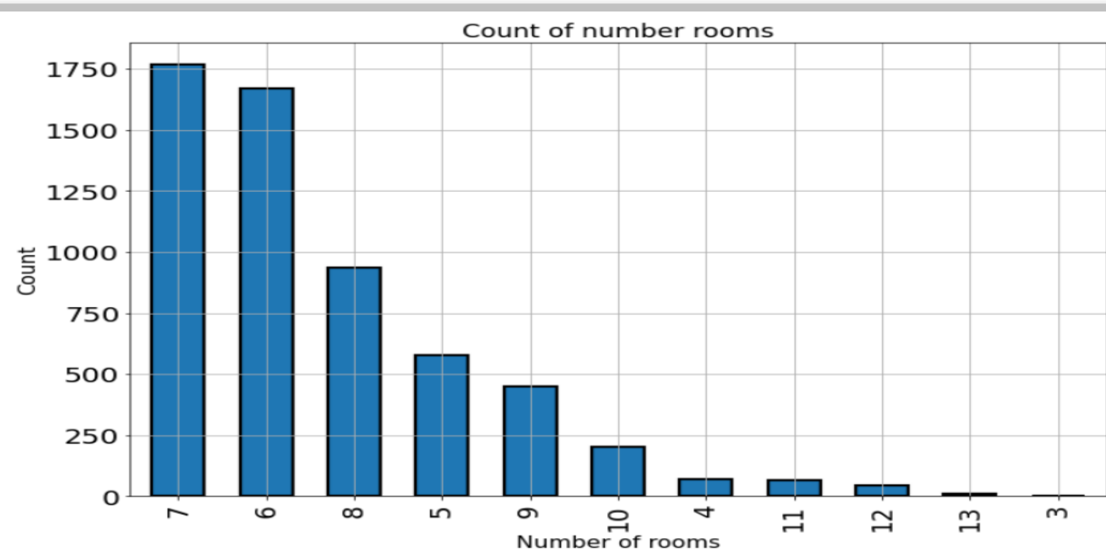
```
#voilin plot to study the distribution of the data
plt.figure(figsize=(12,8))
sns.violinplot(x='FLOORS', y='TOTAL_VALUE', data=ds)
plt.xlabel('FLOORS', fontsize=12)
plt.ylabel('TOTAL_VALUE', fontsize=12)
plt.show()
```



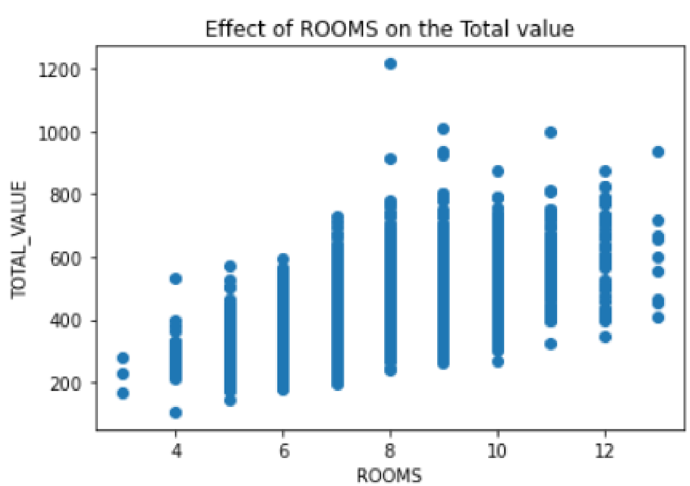
7. Analyzing rooms

- Analyzing how rooms effect the total value.
- Approximately 31% of total data comprises of 7 rooms.
- When it comes to effect on total value homes 8 to 10 rooms has the highest price.

```
fig, ax=plt.subplots(figsize=(12,8))
ds['ROOMS'].value_counts().sort_values(ascending=False).head(12).plot.bar(width=0.6,edgecolor='black',align='center',linewidth=2.
plt.xlabel('Number of rooms',fontsize=16)
plt.ylabel('Count',fontsize=16)
ax.tick_params(labelsize=20)
plt.title('Count of number rooms',fontsize=18)
plt.grid()
plt.ioff()
```



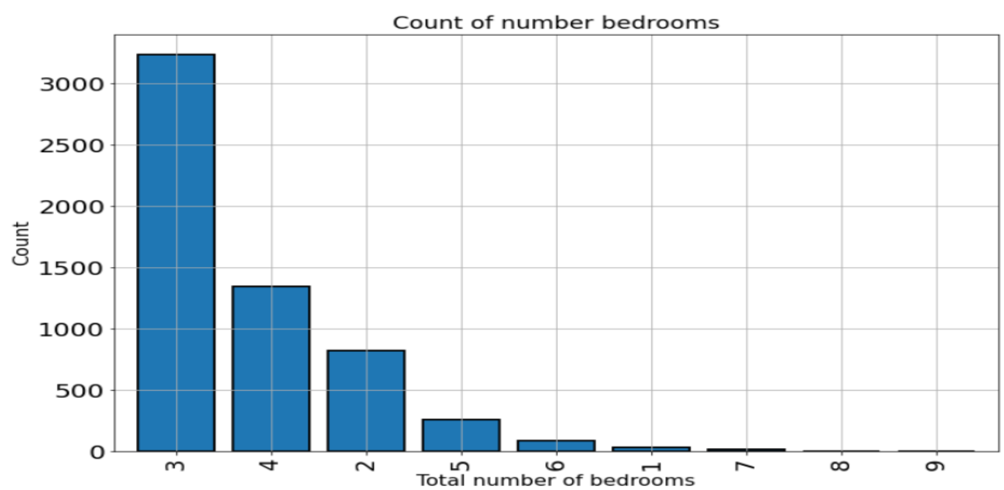

```
plt.scatter(ds['ROOMS'],ds['TOTAL_VALUE'])
plt.xlabel("ROOMS")
plt.ylabel("TOTAL_VALUE")
plt.title("Effect of ROOMS on the Total value")
plt.show()
```



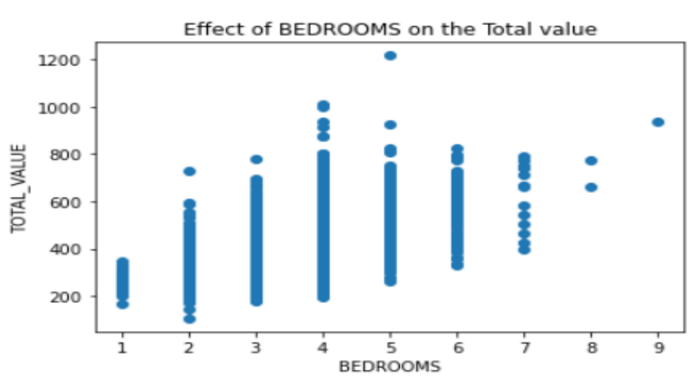
8.Analysis of Bedrooms.

- Analyzing how bedrooms effect the total value.
- Approximately 61% of total data comprises of 3 bedrooms.
- When it comes to effect on total value homes 4 to 5 bedrooms has the highest price.

```
fig, ax=plt.subplots(figsize=(12,8))
ds['BEDROOMS'].value_counts().sort_values(ascending=False).head(9).plot.bar(width=0.8,edgecolor='black',align='center',linewidth=
plt.xlabel('Total number of bedrooms',fontsize=16)
plt.ylabel('Count',fontsize=16)
ax.tick_params(labelsize=20)
plt.title('Count of number bedrooms',fontsize=18)
ax.grid()
plt.ioff()
```



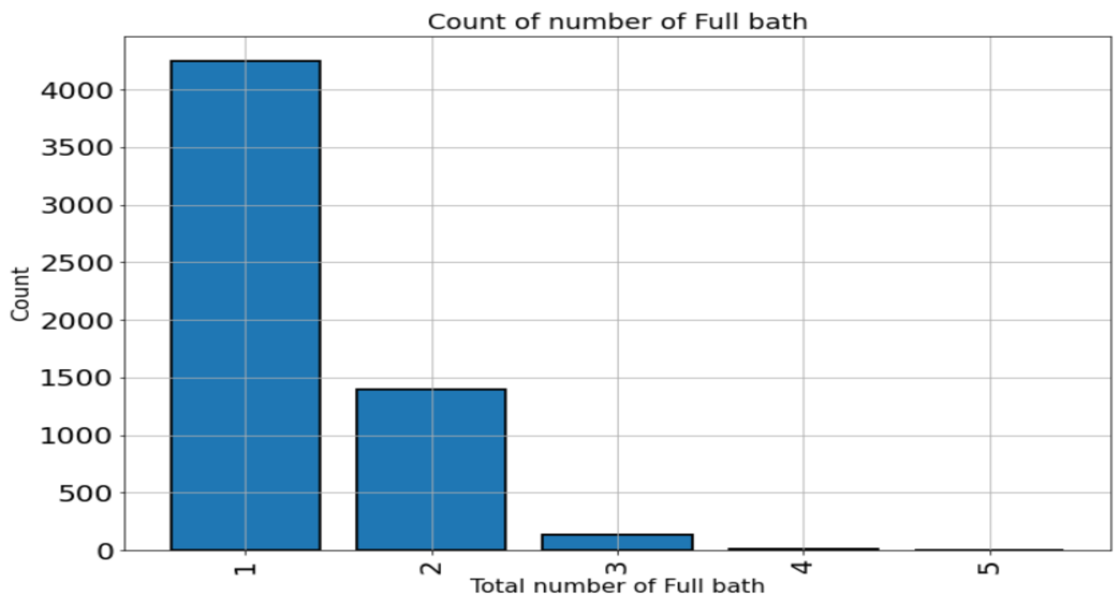

```
plt.scatter(ds['BEDROOMS'],ds['TOTAL_VALUE'])
plt.xlabel("BEDROOMS")
plt.ylabel("TOTAL_VALUE")
plt.title("Effect of BEDROOMS on the Total value")
plt.show()
```



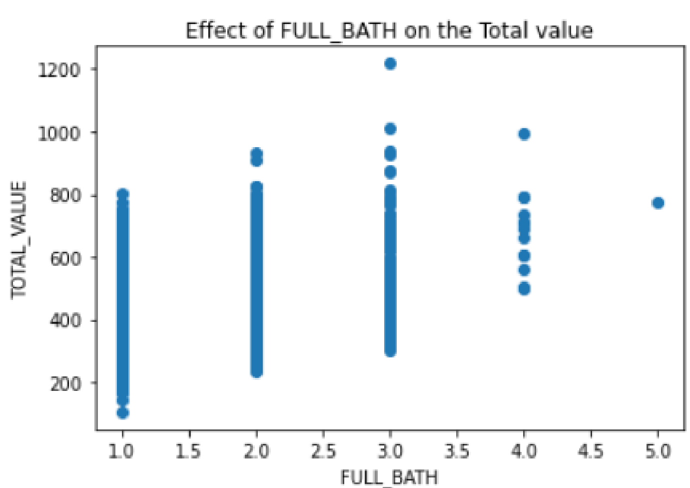
9. Analysis of Full bath.

- Analyzing how full bath effect the total value.
- Approximately 73% of total data comprises of 1 full bath.
- When it comes to effect on total value homes 3 full bath has the highest price.

```
fig, ax=plt.subplots(figsize=(12,8))
ds['FULL_BATH'].value_counts().sort_values(ascending=False).head(5).plot.bar(width=0.8,edgecolor='black',align='center',linewidt
plt.xlabel('Total number of Full bath',fontsize=16)
plt.ylabel('Count',fontsize=16)
ax.tick_params(labelsize=20)
plt.title('Count of number of Full bath',fontsize=18)
ax.grid()
plt.show()
```




```
plt.scatter(ds['FULL_BATH'],ds['TOTAL_VALUE'])
plt.xlabel("FULL_BATH")
plt.ylabel("TOTAL_VALUE")
plt.title("Effect of FULL_BATH on the Total value")
plt.show()
```

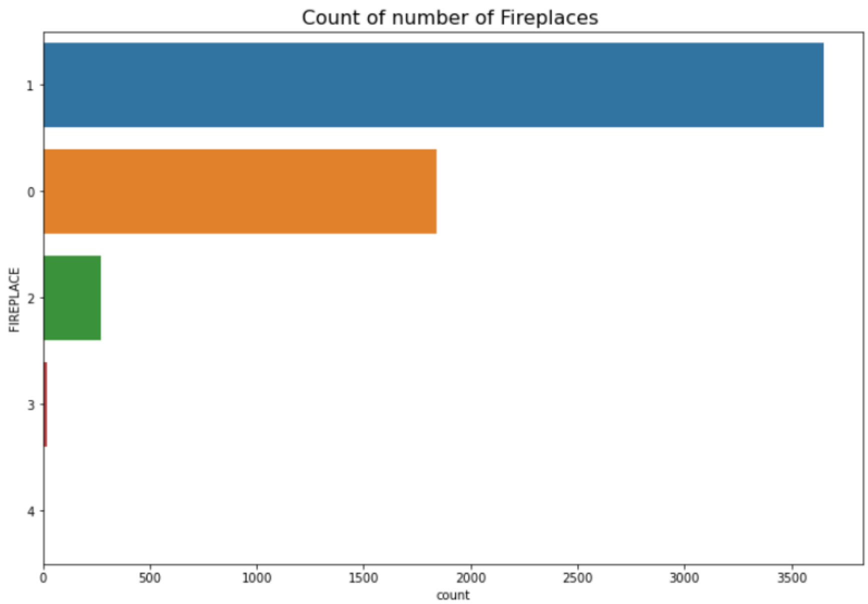


10. Analysis of Fireplaces.

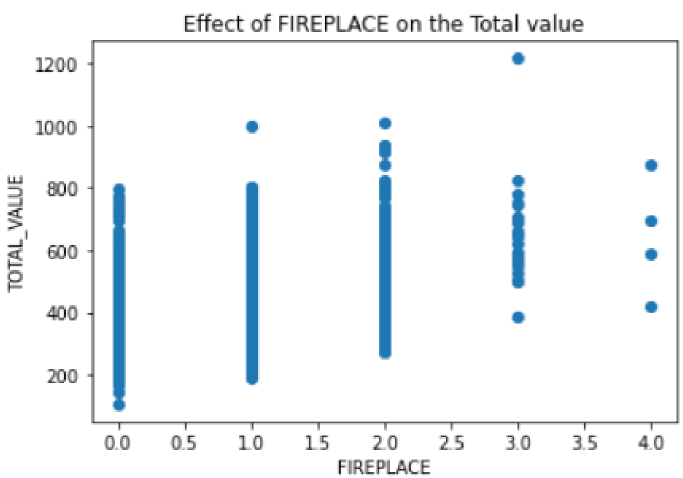
- From the plot below most houses have 1 fireplace.
- When it comes to effect on total value home with 3 fireplaces has highest price, but maximum distribution is seen for 2 fireplaces.

```
fig = plt.figure(figsize = (12, 8))
sns.countplot(y='FIREPLACE', data=ds, order=ds['FIREPLACE'].value_counts()[0:15].index).set_title("Count of number of Fireplaces
plt.ioff()
```

```
<matplotlib.pyplot._IoffContext at 0x201d198f7f0>
```




```
plt.scatter(ds['FIREPLACE'],ds['TOTAL_VALUE'])
plt.xlabel("FIREPLACE")
plt.ylabel("TOTAL_VALUE")
plt.title("Effect of FIREPLACE on the Total value")
plt.show()
```

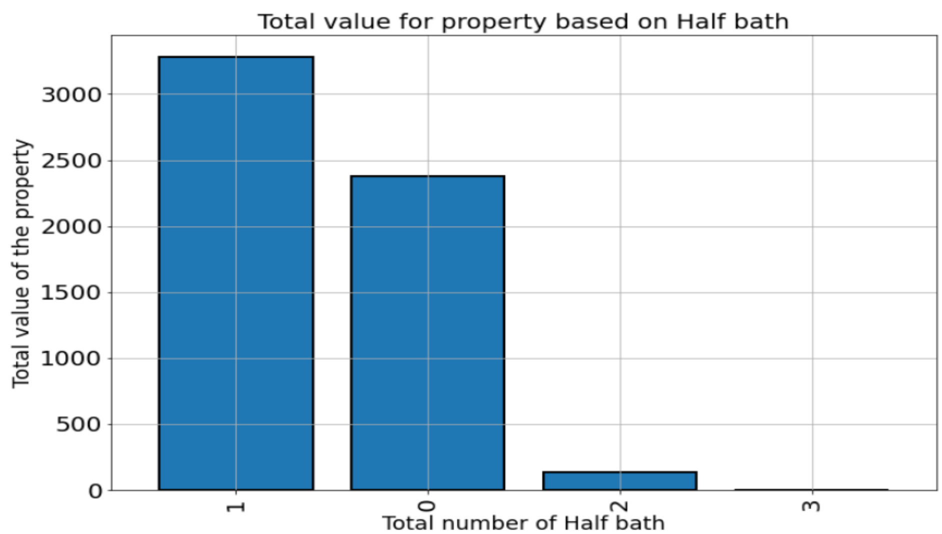


11. Analysis of Half bath

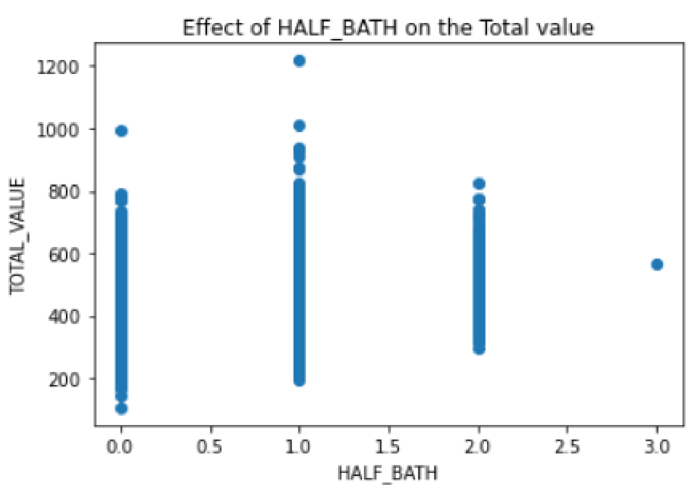
- From the plot below most houses have 1 half bath.
- When it comes to effect on total value homes with 1 half bath has highest price.

```
fig, ax=plt.subplots(figsize=(12,8))
ds['HALF_BATH'].value_counts().sort_values(ascending=False).head(12).plot.bar(width=0.8,edgecolor='black',align='center',linewidt
plt.xlabel('Total number of Half bath',fontsize=18)
plt.ylabel('Count',fontsize=18)
plt.title('Count of number of Half bath',fontsize=20)
ax.tick_params(labelsize=20)
ax.grid()
plt.ioff()
```

```
<matplotlib.pyplot._IoffContext at 0x201d224a3a0>
```




```
plt.scatter(ds['HALF_BATH'],ds['TOTAL_VALUE'])
plt.xlabel("HALF_BATH")
plt.ylabel("TOTAL_VALUE")
plt.title("Effect of HALF_BATH on the Total value")
plt.show()
```

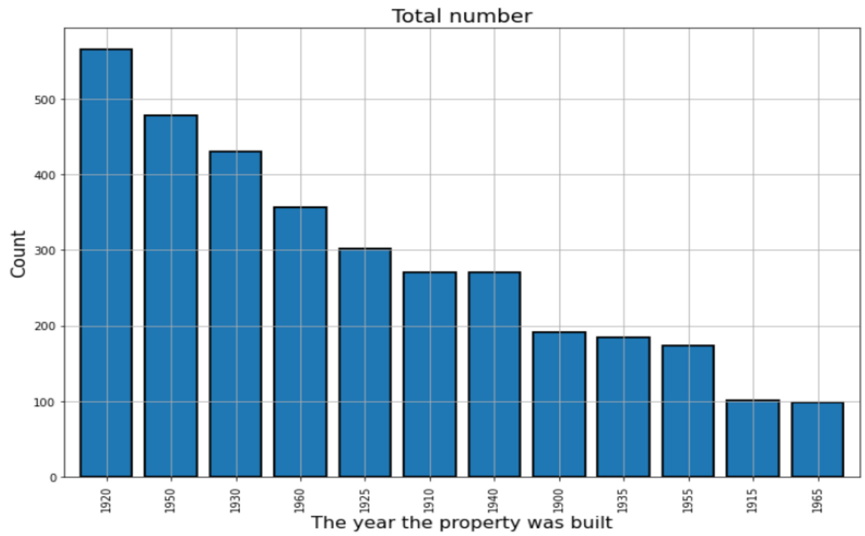


12. Analysis of Year built

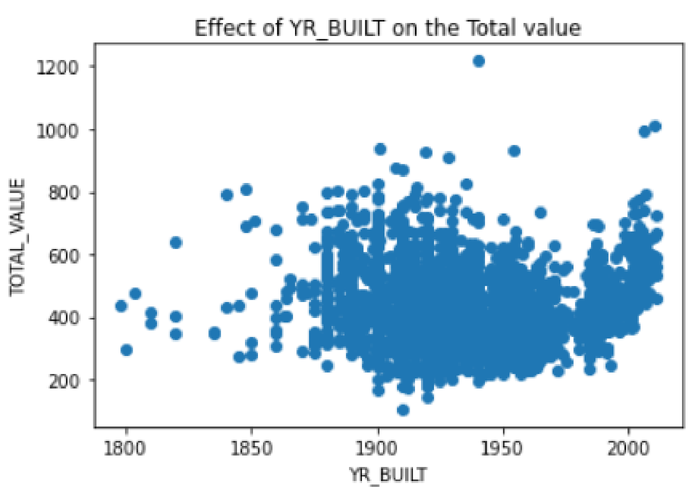
- From the plot below most houses are built around 1920.
- When it comes to effect on total value homes with only 1 house built around 1940 has the highest value whereas around 2000's homes have a higher rate too.

```
fig, ax=plt.subplots(figsize=(12,8))
ds['YR_BUILT'].value_counts().sort_values(ascending=False).head(12).plot.bar(width=0.8,edgecolor='black',align='center',linewidth
plt.xlabel('The year the property was built',fontsize=16)
plt.ylabel('Count',fontsize=16)
plt.title('Count of year built',fontsize=18)
ax.grid()
plt.ioff()
```

```
]: <matplotlib.pyplot._IoffContext at 0x201d22b5550>
```



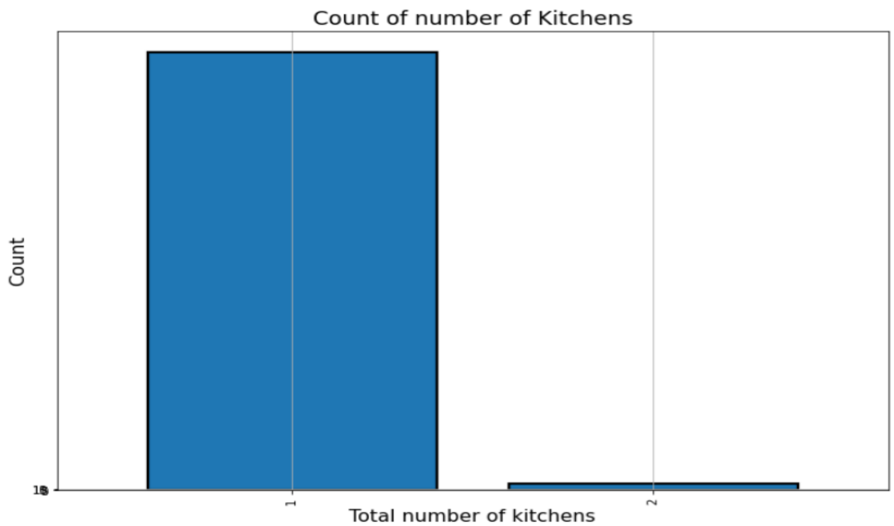

```
plt.scatter(ds['YR_BUILT'],ds['TOTAL_VALUE'])
plt.xlabel("YR_BUILT")
plt.ylabel("TOTAL_VALUE")
plt.title("Effect of YR_BUILT on the Total value")
plt.show()
```



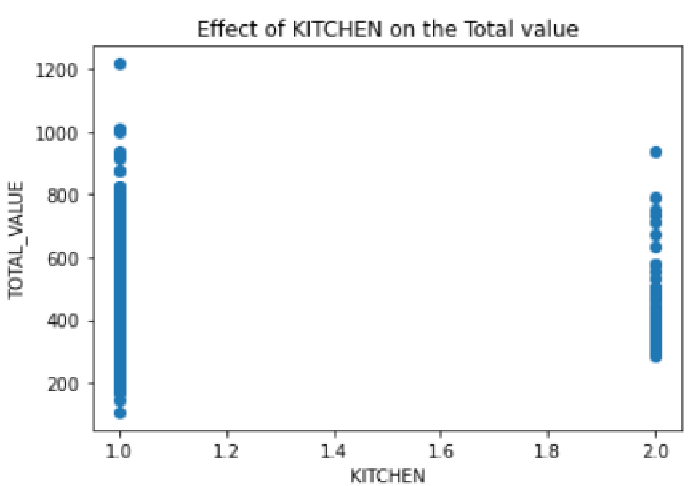
13. Analysis of Kitchen

- From the plot below most houses have 1 kitchen.
- When it comes to effect on total value homes with 1 Kitchen is expensive and effects the total value.
- Highest value of house has only 1 Kitchen.

```
fig, ax=plt.subplots(figsize=(12,8))
ds['KITCHEN'].value_counts().sort_values(ascending=False).head(2).plot.bar(width=0.8,edgecolor='black',align='center',linewidth=
plt.xlabel('Total number of kitchens',fontsize=16)
plt.ylabel('Count',fontsize=16)
ax.set_yticks([0, 5, 10, 15])
plt.title('Count of number of Kitchens',fontsize=18)
ax.grid()
plt.show()
```




```
plt.scatter(ds['KITCHEN'],ds['TOTAL_VALUE'])
plt.xlabel("KITCHEN")
plt.ylabel("TOTAL_VALUE")
plt.title("Effect of KITCHEN on the Total value")
plt.show()
```

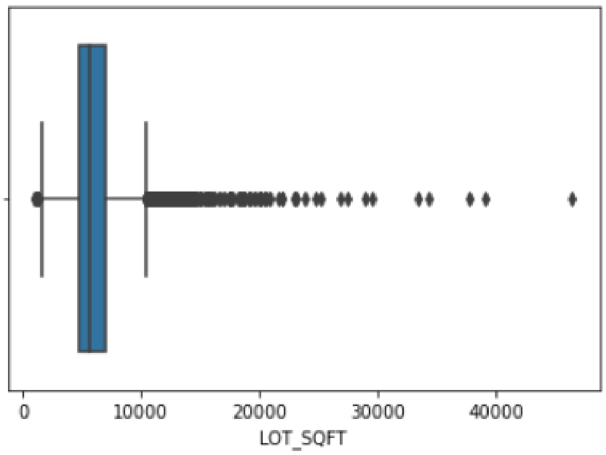


14.Analysis of lot size.

- Many houses have lot size between 5000 and 10000 square feet.

```
#26
sns.boxplot(x=ds['LOT_SQFT'])
```

<AxesSubplot:xlabel='LOT_SQFT'>

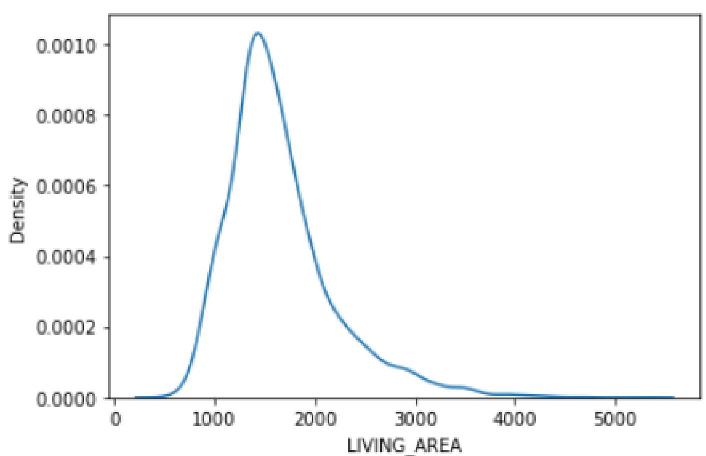


15.Analysis of Living Area

- Many houses have a mean distribution around 1500 square feet.

```
sns.kdeplot(ds['LIVING_AREA'])
```

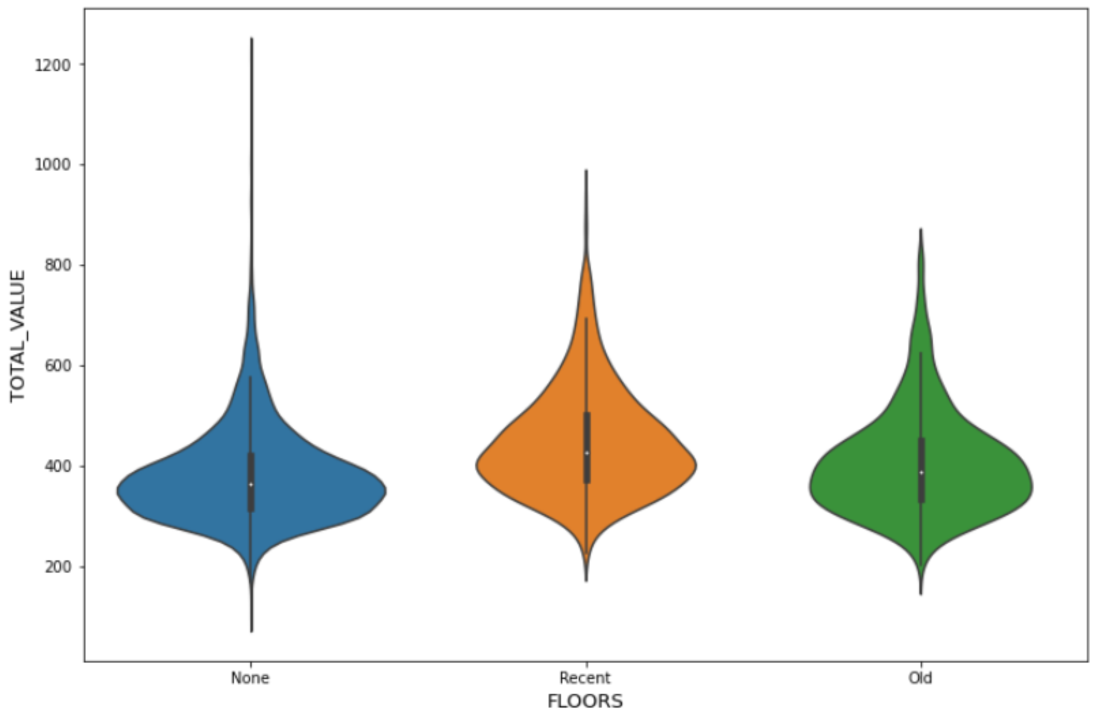
<AxesSubplot:xlabel='LIVING_AREA', ylabel='Density'>



16. Analysis of REMODEL before and after data cleaning.

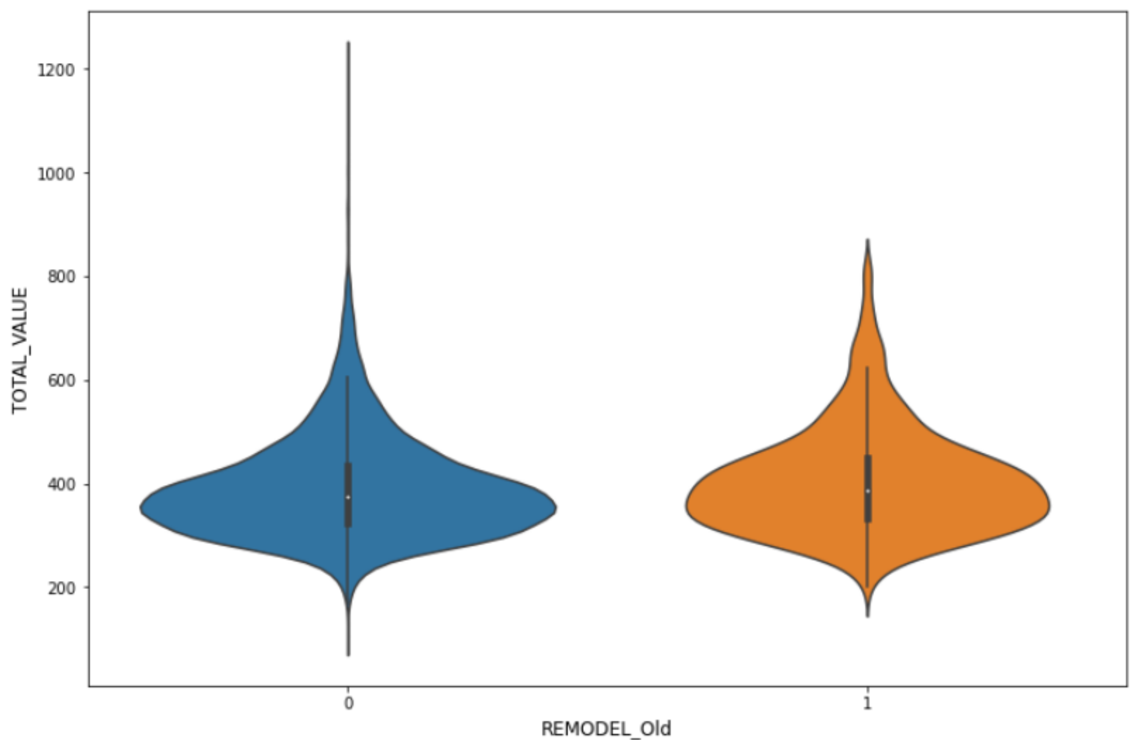
Before Data cleaning

```
#voilin plot to study the distribution of the data
plt.figure(figsize=(12,8))
sns.violinplot(x='REMODEL', y='TOTAL_VALUE', data=ds)
plt.xlabel('FLOORS', fontsize=13)
plt.ylabel('TOTAL_VALUE', fontsize=13)
plt.show()
```

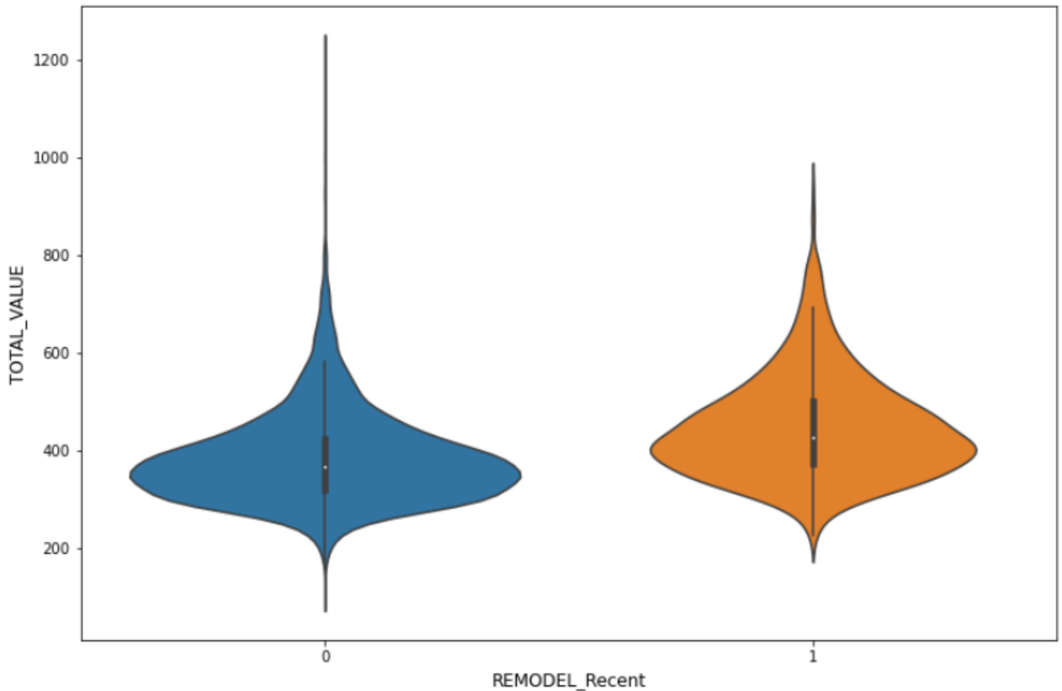


After Data cleaning since the variable had to be flagged.

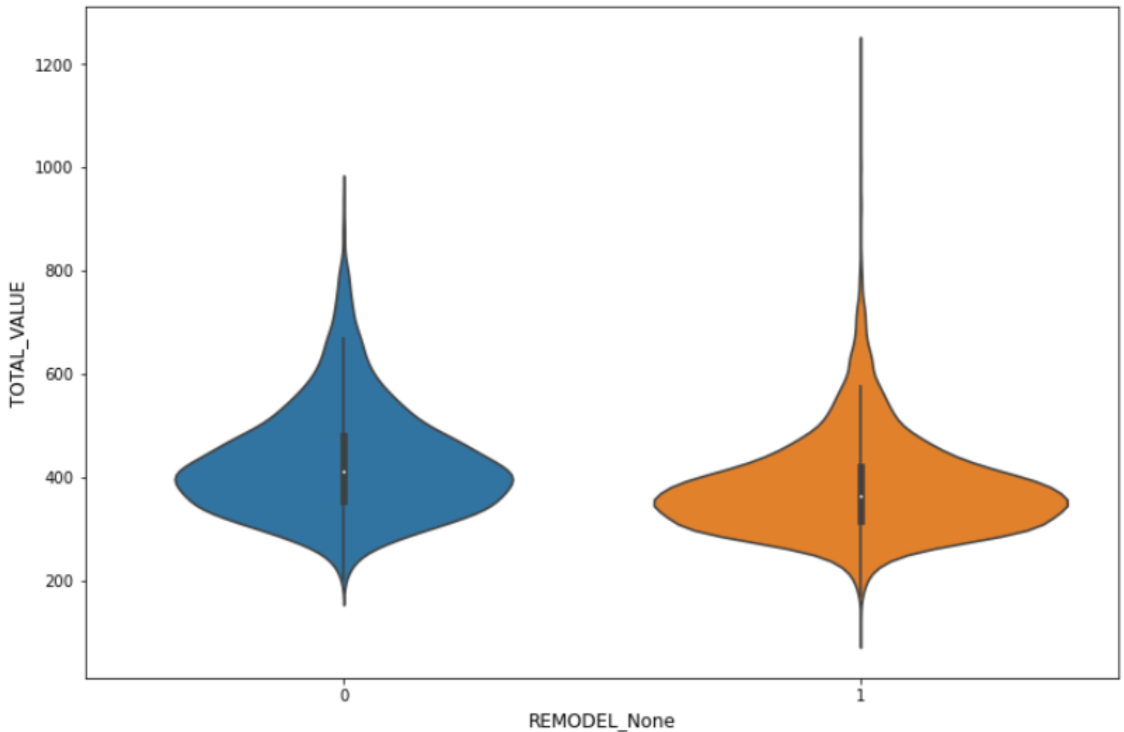
```
#29
plt.figure(figsize=(12,8))
sns.violinplot(x='REMODEL_Old',y='TOTAL_VALUE', data=ds)
plt.xlabel('REMODEL_Old', fontsize=12)
plt.ylabel('TOTAL_VALUE', fontsize=12)
plt.show()
```



```python
plt.figure(figsize=(12,8))
sns.violinplot(x='REMODEL_Recent',y='TOTAL_VALUE', data=ds)
plt.xlabel('REMODEL_Recent', fontsize=12)
plt.ylabel('TOTAL_VALUE', fontsize=12)
plt.show()
```



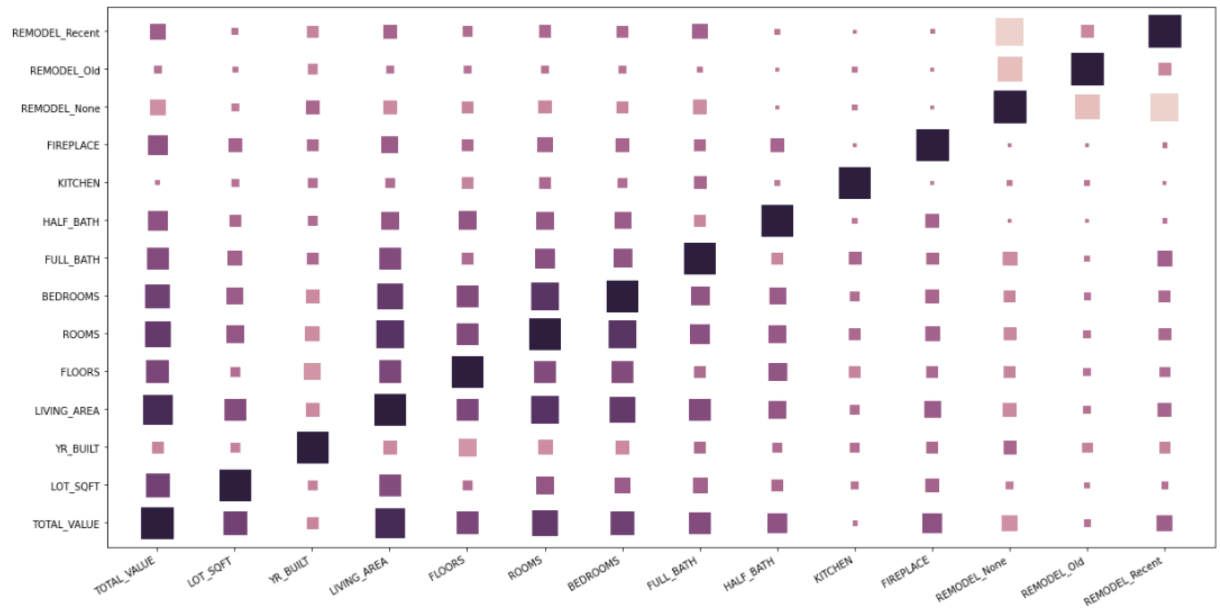
```
plt.figure(figsize=(12,8))
sns.violinplot(x='REMODEL_None',y='TOTAL_VALUE', data=ds)
plt.xlabel('REMODEL_None', fontsize=12)
plt.ylabel('TOTAL_VALUE', fontsize=12)
plt.show()
```



17.Heatmap

We will be able to identify the dependency between the attributes and the factor of change of one attribute with respect to another attribute.

```
def value_to_color(val):
        ind = int((float(val - corr['value'].min()) / (corr['value'].max() - corr['value'].min()))*255)
        return palette[ind]
corr = ds.corr()
corr = pd.melt(corr.reset_index(), id_vars='index')
fig, ax = plt.subplots(figsize=(20,10))
n_colors = 256
palette = sns.cubehelix_palette(n_colors)
color_min, color_max = [-1, 1]

ax.scatter(
    x = corr['index'].map({p[1]:p[0] for p in enumerate(ds.columns)}),
    y = corr['variable'].map({p[1]:p[0] for p in enumerate(ds.columns)}),
    s = corr['value'].abs() * 1000,
    c = corr['value'].apply(value_to_color),
    marker='s')
ax.set_xticks([x for x in range(len(ds.columns))])
ax.set_xticklabels(ds.columns, rotation=30, horizontalalignment='right')
ax.set_yticks([x for x in range(len(ds.columns))])
ax.set_yticklabels(ds.columns)
plt.show()
```



References:

1. "Data Cleaning in Python: the Ultimate Guide" by Will Koehrsen on Towards Data Science https://towardsdatascience.com/data-cleaning-in-python-the-ultimate-guide-2020-c63b88bf0a0d

2. "8 Best Data Transformation in Pandas" https://ai.plainenglish.io/data-transformation-in-pandas-29b2b3c61b34

3. MinMaxScalar https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

4. http://theta.edu.pl/wp-content/uploads/2012/10/exploratorydataanalysis_tukey.pdf