Predicting House Prices Using Machine Learning

Development Part - 5

Double Tap

to Edit

NAME : M.RAGULAN

REG NO : 513521106028

COLLEGE: AMCET

PROJECT : PHASE 5

Email ID : ragulandevil@gmail.com

- **Data cleaning** can be applied to filling in missing values, remove noise, resolving inconsistencies, identifying and removing outliers in the data.
- **Data integration** merges data from multiple sources into a coherent data store, such as a data warehouse.
- **Data transformations**, such as normalization, may be applied. For example, normalization may improve the accuracy and efficiency of mining algorithms involving distance measurements.
- **Data reduction** can reduce the data size by eliminating redundant features, or clustering, for instance.

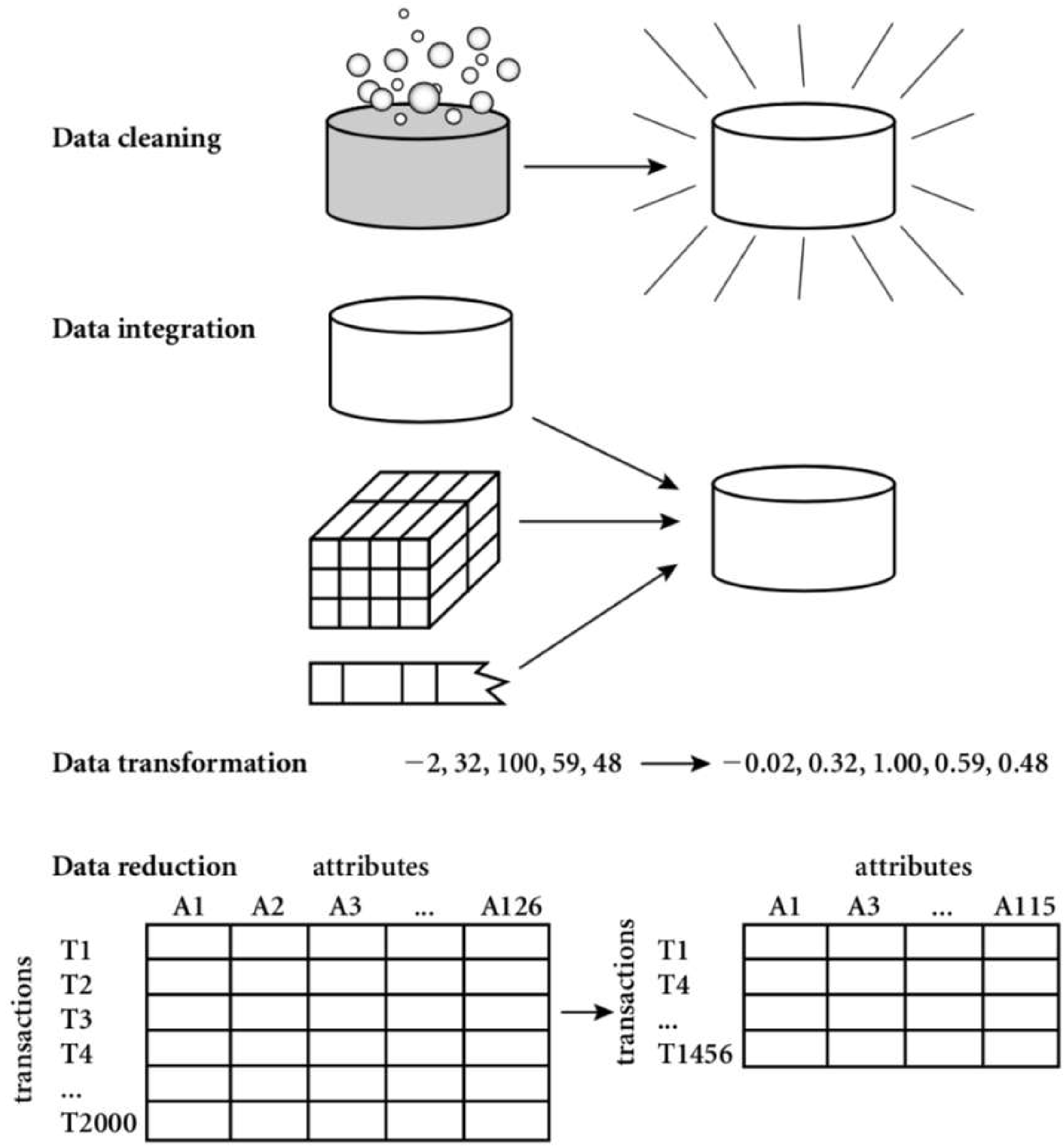
Reference: Data Mining:Concepts and Techniques Second Edition, Jiawei Han, Micheline Kamber.

PS: This is my first kaggle notebook contribution. Hope you like it!!

```python
def find_missing_percent(data):
    """
    Returns dataframe containing the total missing values and percentage of total
    missing values of a column.
    """
    miss_df = pd.DataFrame({'ColumnName':[],'TotalMissingVals':[],'PercentMissing':[]})
    for col in data.columns:
        sum_miss_val = data[col].isnull().sum()
        percent_miss_val = round((sum_miss_val/data.shape[0])*100,2)
        miss_df = miss_df.append(dict(zip(miss_df.columns,[col,sum_miss_val,percent_miss_val])),ignore_index=True)
    return miss_df
```

There are a number of data preprocessing techniques available such as,

1. **Data Cleaning**
2. **Data Integration**
3. **Data Transformation**
4. **Data Reduction**



```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from operator import itemgetter
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import OrdinalEncoder
from category_encoders.target_encoder import TargetEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import (GradientBoostingRegressor, GradientBoostingClassifier)
import xgboost
```

	ColumnName	TotalMissingVals	PercentMissing
3	LotFrontage	259.0	17.74
6	Alley	1369.0	93.77
25	MasVnrType	8.0	0.55
26	MasVnrArea	8.0	0.55
30	BsmtQual	37.0	2.53
31	BsmtCond	37.0	2.53
32	BsmtExposure	38.0	2.60
33	BsmtFinType1	37.0	2.53
35	BsmtFinType2	38.0	2.60
42	Electrical	1.0	0.07
57	FireplaceQu	690.0	47.26
58	GarageType	81.0	5.55
59	GarageYrBlt	81.0	5.55
60	GarageFinish	81.0	5.55
63	GarageQual	81.0	5.55
64	GarageCond	81.0	5.55
72	PoolQC	1453.0	99.52
73	Fence	1179.0	80.75
74	MiscFeature	1406.0	96.30

```
miss_df = find_missing_percent(train)
'''Displays columns with missing values'''
display(miss_df[miss_df['PercentMissing']>0.0])
print("\n")
print(f"Number of columns with missing values:{str(miss_df[miss_df['PercentMissing']>0.0].shape[0])}")
```

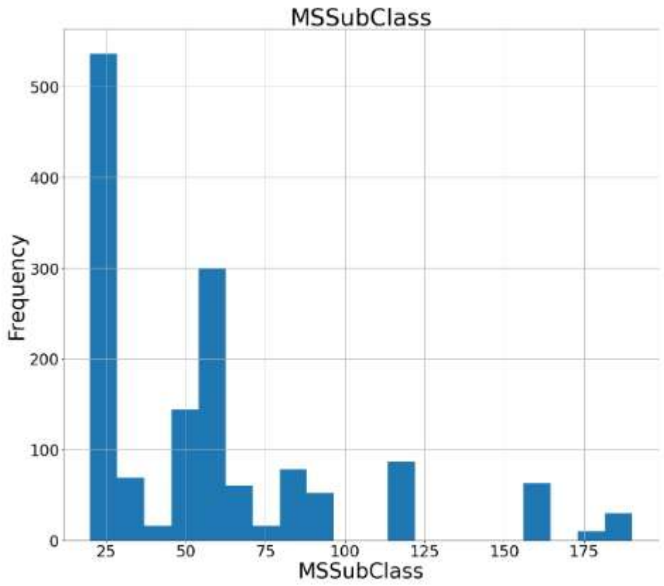
1.2 Drop the columns which have more than 70% of missing values

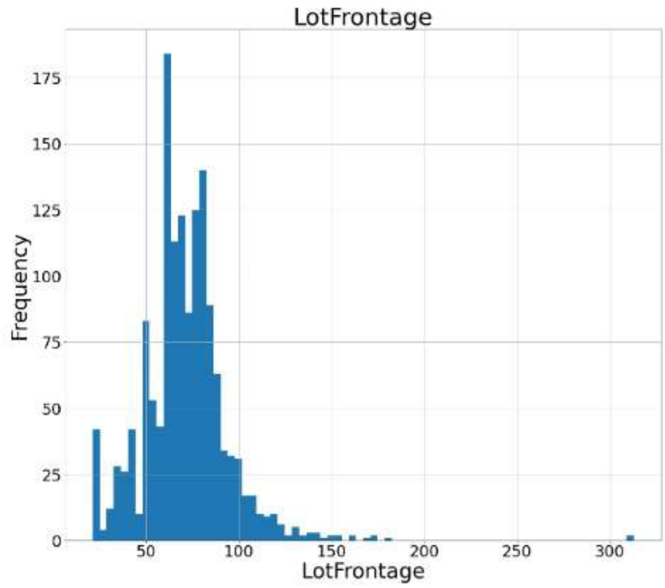
In [5]:

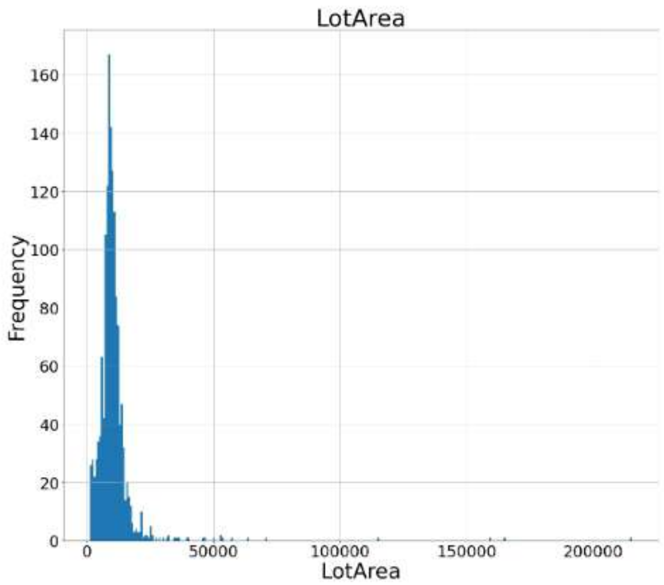
```
drop_cols = miss_df[miss_df['PercentMissing'] >70.0].ColumnName.tolist()
print(f"Number of columns with more than 70%: {len(drop_cols)}")
train = train.drop(drop_cols,axis=1)
test = test.drop(drop_cols,axis =1)

miss_df = miss_df[miss_df['ColumnName'].isin(train.columns)]
'''Columns to Impute'''
impute_cols = miss_df[miss_df['TotalMissingVals']>0.0].ColumnName.tolist()
miss_df[miss_df['TotalMissingVals']>0.0]
```

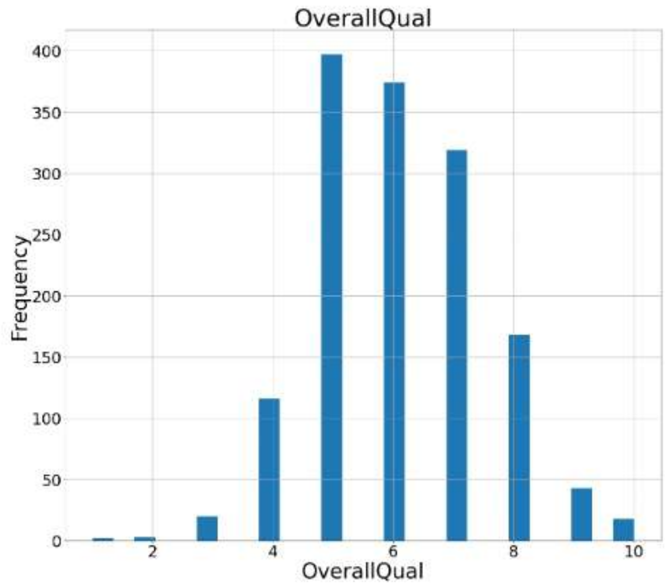
```
Number of columns with more than 70%: 4
```

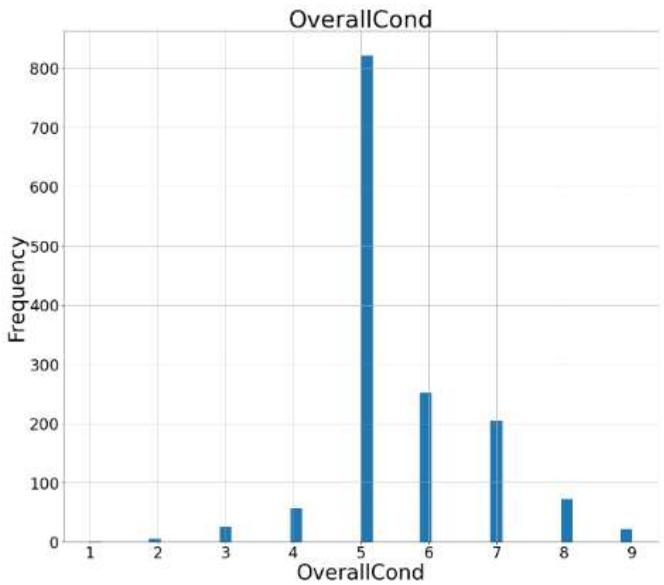
	ColumnName	TotalMissingVals	PercentMissing
3	LotFrontage	259.0	17.74
25	MasVnrType	8.0	0.55
26	MasVnrArea	8.0	0.55
30	BsmtQual	37.0	2.53
31	BsmtCond	37.0	2.53
32	BsmtExposure	38.0	2.60
33	BsmtFinType1	37.0	2.53
35	BsmtFinType2	38.0	2.60
42	Electrical	1.0	0.07
57	FireplaceQu	690.0	47.26
58	GarageType	81.0	5.55
59	GarageYrBlt	81.0	5.55
60	GarageFinish	81.0	5.55
63	GarageQual	81.0	5.55
64	GarageCond	81.0	5.55

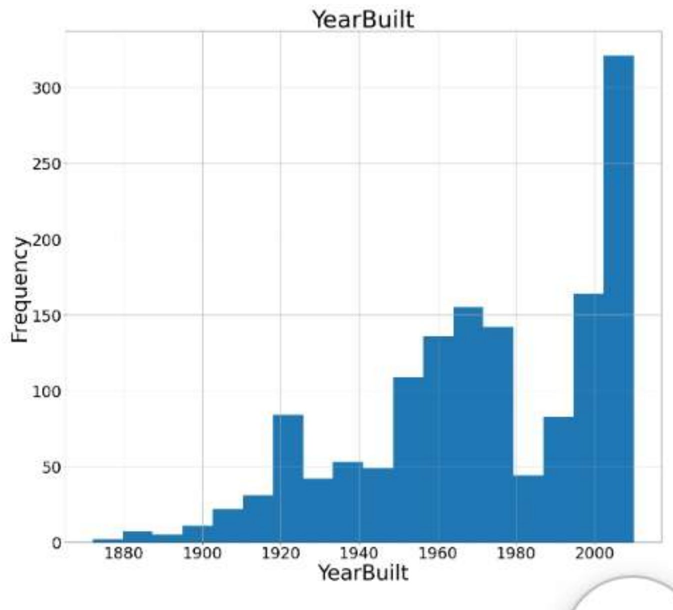

MSSubClass
Frequency
0
100
200
300
400
500
25
50
75
100
125
150
175
MSSubClass

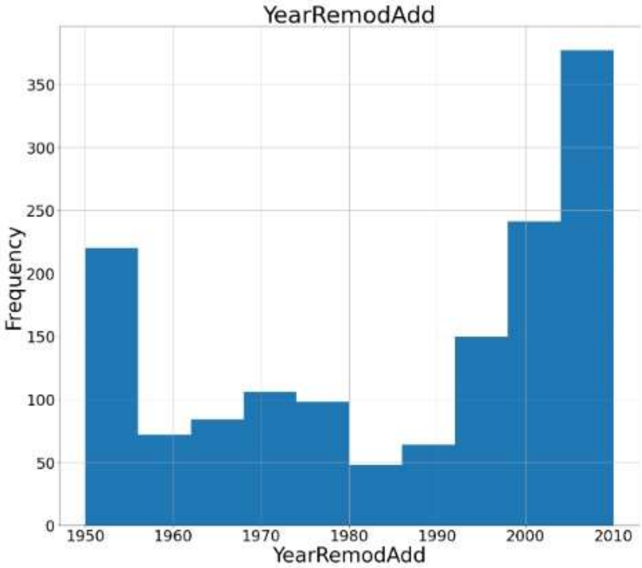

LotFrontage
Frequency
0
25
50
75
100
125
150
175
50
100
150
200
250
300
LotFrontage

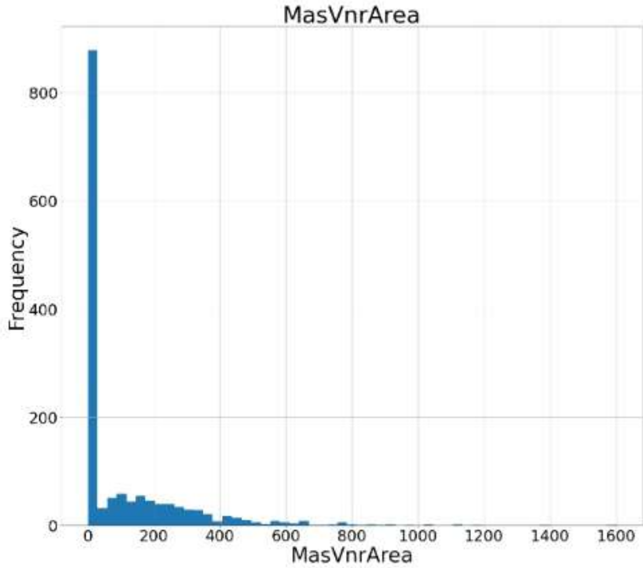

LotArea
Frequency
0
20
40
60
80
100
120
140
160
0
50000
100000
150000
200000
LotArea

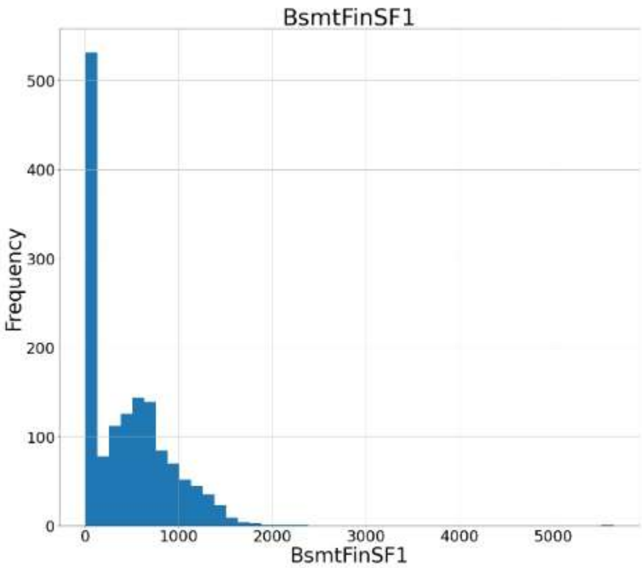

OverallQual
Frequency
0
50
100
150
200
250
300
350
400
2
4
6
8
10
OverallQual

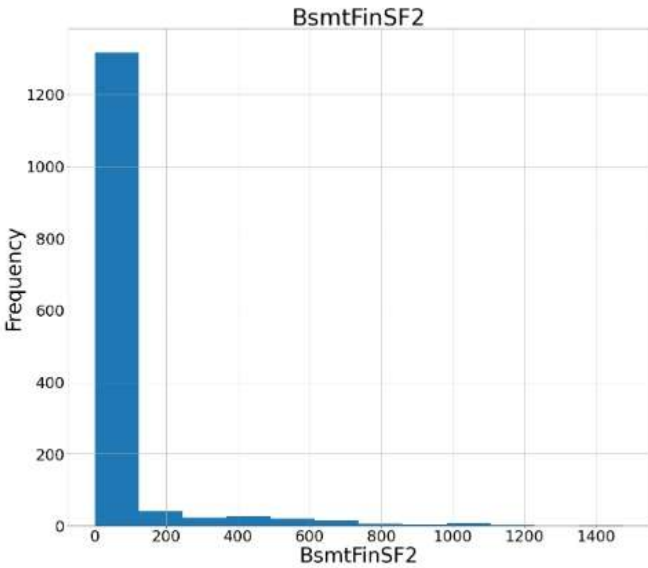

OverallCond
Frequency
0
100
200
300
400
500
600
700
800
1
2
3
4
5
6
7
8
9
OverallCond

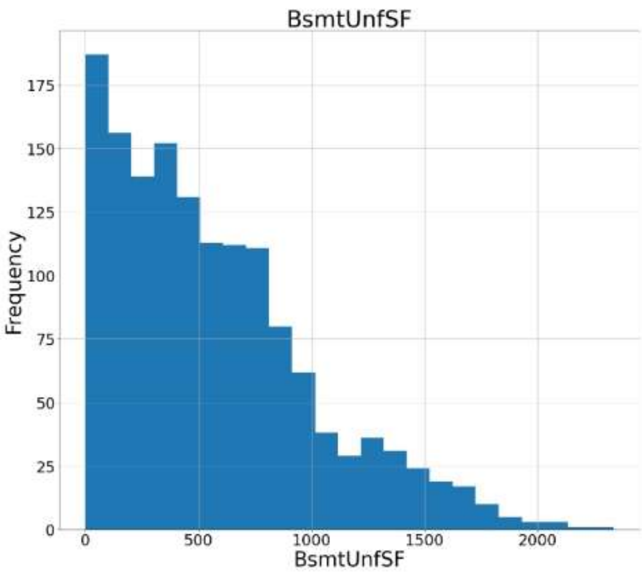

YearBuilt
Frequency
0
50
100
150
200
250
300
1880
1900
1920
1940
1960
1980
2000
YearBuilt

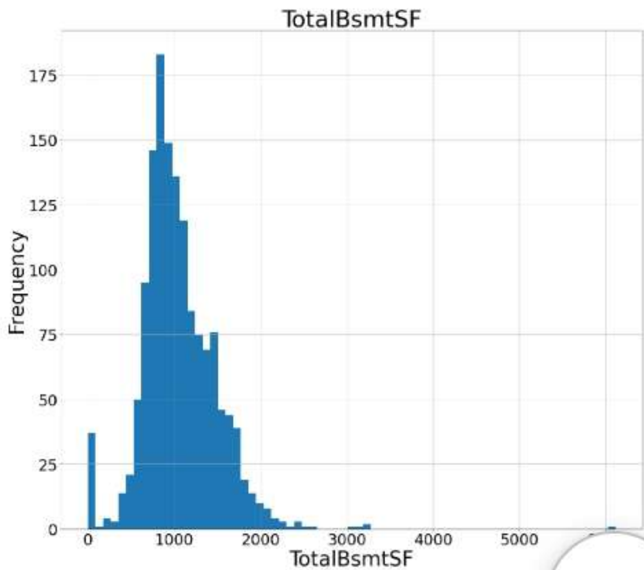

YearRemodAdd
Frequency
YearRemodAdd


MasVnrArea
Frequency
MasVnrArea


BsmtFinSF1
Frequency
BsmtFinSF1


BsmtFinSF2
Frequency
BsmtFinSF2


BsmtUnfSF
Frequency
BsmtUnfSF


TotalBsmtSF
Frequency
TotalBsmtSF

DATASET

Here we have web scrapped the Data from 99acres.com website which is one of the leading real estate websites operating in INDIA.

Our Data contains Bombay Houses only.

Dataset looks as follows-

	Price	PricePerSqft	Area_Sqm	Location	Bedrooms	Latitude	Longitude	PricePerSqM
0	13300000	16625	74.32	Kandivali (East)	2	19.210200	72.864891	178885.00
1	9000000	15666	55.74	Ramgad Nagar	1	19.167700	72.949300	168566.16
2	9000000	19148	43.66	Mahakali Caves	1	19.130609	72.873816	206032.48
3	9000000	10588	78.97	Louis Wadi	2	19.126005	72.825052	113926.88
4	100000000	20000	464.51	Barrister Nath Pai Nagar	5	19.075014	72.907571	215200.00

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 840 entries, 0 to 839
Data columns (total 6 columns):
Price          840 non-null int64
Area_Sqm       840 non-null float64
Bedrooms       840 non-null int64
Latitude       840 non-null float64
Longitude      840 non-null float64
PricePerSqM    840 non-null float64
dtypes: float64(4), int64(2)
memory usage: 39.5 KB
```

3. Multivariable Analysis

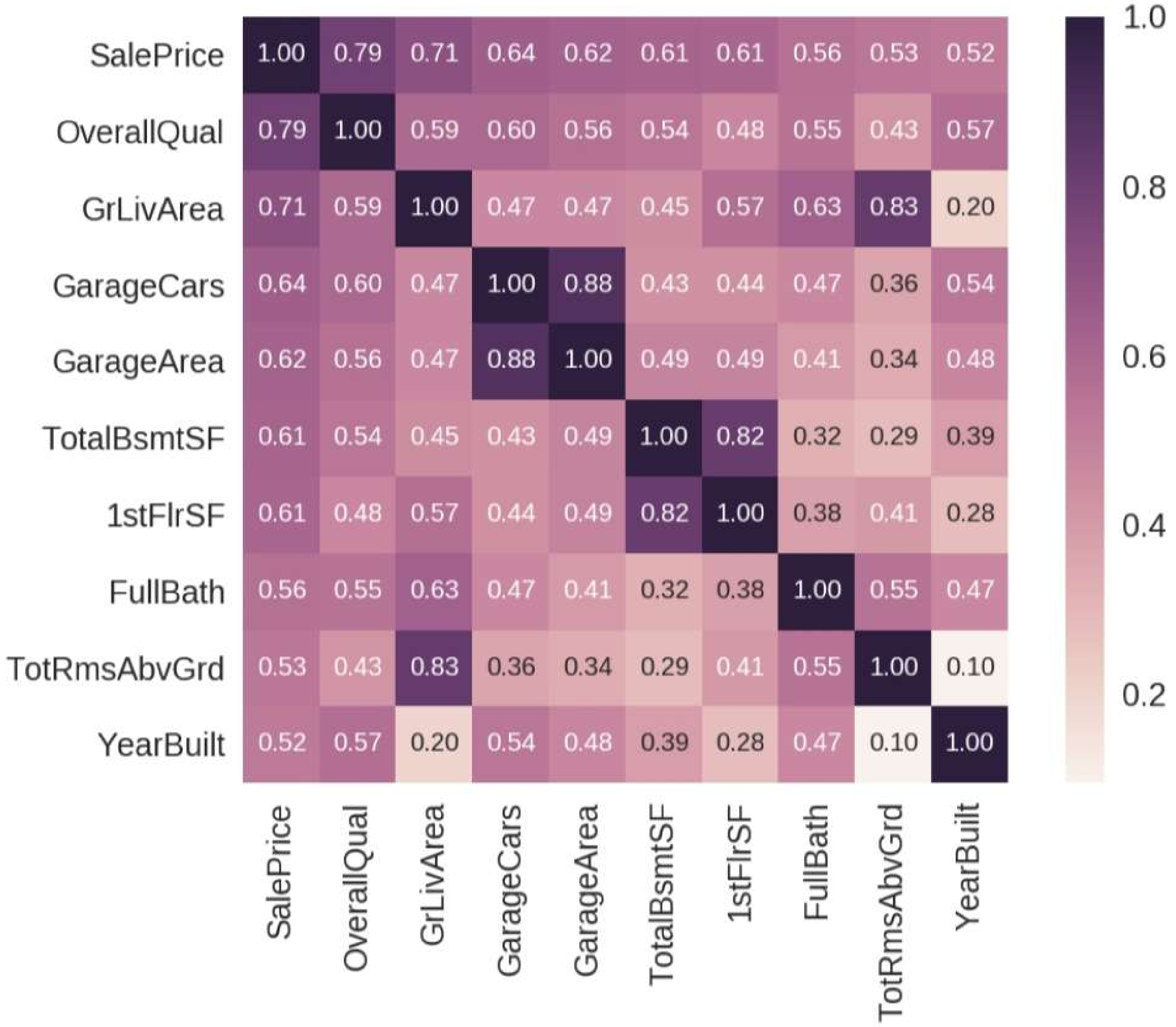
Let's check out all the variables! There are two types of features in housing data, categorical and numerical.

Categorical data is just like it sounds. It is in categories. It isn't necessarily linear, but it follows some kind of pattern. For example, take a feature of "Downtown". The response is either "Near", "Far", "Yes", and "No". Back then, living in downtown usually meant that you couldn't afford to live in uptown. Thus, it could be implied that downtown establishments cost less to live in. However, today, that is not the case. (Thank you, hipsters!) So we can't really establish any particular order of response to be "better" or "worse" than the other.

Numerical data is data in number form. (Who could have thought!) These features are in a linear relationship with each other. For example, a 2,000 square foot place is 2 times "bigger" than a 1,000 square foot place. Plain and simple. Simple and

```
Index(['MSZoning', 'Street', 'Alley',
'LotShape', 'LandContour', 'Utilitie
s',
       'LotConfig', 'LandSlope', 'Neig
hborhood', 'Condition1', 'Condition2',
       'BldgType', 'HouseStyle', 'Roof
Style', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'E
xterQual', 'ExterCond', 'Foundation',
       'BsmtQual', 'BsmtCond', 'BsmtEx
posure', 'BsmtFinType1', 'BsmtFinType
2',
       'Heating', 'HeatingQC', 'Centra
lAir', 'Electrical', 'KitchenQual',
       'Functional', 'FireplaceQu', 'G
arageType', 'GarageFinish', 'GarageQua
l',
       'GarageCond', 'PavedDrive', 'Po
olQC', 'Fence', 'MiscFeature',
       'SaleType', 'SaleCondition'],
      dtype='object')
```

```
def fit_model(x_train,y_train, model):
    """
    Fits x_train to y_train for the given
    model.
    """
    model.fit(x_train,y_train)
    return model

'''Xtreme Gradient Boosting Regressor'''
model = xgboost.XGBRegressor(objective="reg:squarederror", random_state=42)
model = fit_model(x_train,y_train, model)
'''Predict the outcomes'''
predictions = model.predict(test)
```


SalePrice
OverallQual
GrLivArea
GarageCars
GarageArea
TotalBsmtSF
1stFlrSF
FullBath
TotRmsAbvGrd
YearBuilt
1.00 0.79 0.71 0.64 0.62 0.61 0.61 0.56 0.53 0.52
0.79 1.00 0.59 0.60 0.56 0.54 0.48 0.55 0.43 0.57
0.71 0.59 1.00 0.47 0.47 0.45 0.57 0.63 0.83 0.20
0.64 0.60 0.47 1.00 0.88 0.43 0.44 0.47 0.36 0.54
0.62 0.56 0.47 0.88 1.00 0.49 0.49 0.41 0.34 0.48
0.61 0.54 0.45 0.43 0.49 1.00 0.82 0.32 0.29 0.39
0.61 0.48 0.57 0.44 0.49 0.82 1.00 0.38 0.41 0.28
0.56 0.55 0.63 0.47 0.41 0.32 0.38 1.00 0.55 0.47
0.53 0.43 0.83 0.36 0.34 0.29 0.41 0.55 1.00 0.10
0.52 0.57 0.20 0.54 0.48 0.39 0.28 0.47 0.10 1.00
1.0
0.8
0.6
0.4
0.2

	Most Correlated Features
0	SalePrice
1	OverallQual
2	GrLivArea
3	GarageCars
4	GarageArea
5	TotalBsmtSF
6	1stFlrSF
7	FullBath
8	TotRmsAbvGrd
9	YearBuilt