# SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

**Kattankulathur, Chengalpattu District - 603203**

**18CSE479E / Statistical Machine Learning**

**MINI PROJECT REPORT**

# Real Estate Price Prediction

***Guided by:***

Dr.M.Uma

**Submitted By:**


AYUSH GUPTA (RA2011026010038)

Computer Science and Engineering with specialization in AIML
*of*
Department of Computational Intelligence
*of*
Bachelor's of Technology

## PROBLEM STATEMENT

The real estate industry is characterized by dynamic and fluctuating property prices influenced by numerous factors such as location, property features, economic indicators, and market trends.

Accurate prediction of real estate prices is essential for buyers, sellers, and investors to make informed decisions. Traditional methods of price estimation often fall short in capturing the complexity and intricacies of the real estate market. Therefore, there is a need to leverage machine learning techniques and the programming language Python to develop reliable and accurate models for real estate price prediction.

The problem at hand is to create a machine learning model that can effectively analyze historical real estate data and predict future property prices. This involves addressing several challenges, including data preprocessing, feature selection, algorithm selection, and model evaluation.

Preprocessing the data involves cleaning and transforming it into a suitable format, removing inconsistencies and outliers, and handling missing values. Selecting the most relevant features from the dataset is crucial to improve the model's predictive capabilities and avoid overfitting.

Choosing the appropriate machine learning algorithm, such as regression, decision trees, random forests, or neural networks, requires careful consideration of the dataset characteristics and the desired level of interpretability. Evaluating the model's performance using appropriate metrics and comparing it to other approaches will determine its effectiveness and reliability.

## INTRODUCTION

Real estate is one of the most dynamic and lucrative industries, with property prices constantly fluctuating based on a variety of factors. The ability to accurately predict real estate prices is crucial for buyers, sellers, investors, and industry professionals alike. In recent years, machine learning has emerged as a powerful tool for analyzing and predicting real estate prices. By leveraging the vast amount of data available and utilizing advanced algorithms, machine learning models can uncover patterns and relationships that traditional methods may overlook.

Python, a versatile and popular programming language, provides a robust ecosystem of libraries and frameworks for machine learning. Its simplicity, readability, and extensive community support make it an ideal choice for implementing real estate price prediction models. By combining machine learning algorithms with Python's data processing capabilities, we can develop predictive models that take into account various factors influencing real estate prices, such as location, property features, economic indicators, and market trends.

This project aims to explore the application of machine learning techniques in predicting real estate prices using Python. We will utilize a dataset containing historical real estate information, including property attributes and corresponding sale prices. By employing various machine learning algorithms, such as regression, decision trees, random forests, or neural networks, we will train models to learn patterns from the data and make accurate predictions.

The process will involve several key steps. First, we will preprocess the data, cleaning it and transforming it into a suitable format for machine learning algorithms. Next, we will split the dataset into training and testing sets, ensuring that the models generalize well to unseen data. Then, we will select and implement appropriate machine learning algorithms, fine-tuning their parameters to achieve optimal performance. Once the models are trained, we will evaluate their accuracy using appropriate metrics and compare their performance to identify the most effective approach for real estate price prediction.

By the end of this project, we aim to develop a reliable and interpretable machine learning model that can predict real estate prices with a high degree of accuracy. Such a model can aid in informed decision-making, help investors identify potential opportunities, and provide valuable insights into the complex dynamics of the real estate market. Through this exploration of real estate price prediction using machine learning and Python, we can unlock new avenues for leveraging data-driven approaches in the ever-evolving world of real estate.

## DATA SET

1. Title: Boston Housing Data

2. Sources:
   (a) Origin: This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.
   (b) Creator: Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978.
   (c) Date: July 7, 1993

3. Past Usage:
   - Used in Belsley, Kuh & Welsch, 'Regression diagnostics', Wiley, 1980.N.B. Various transformations are used in the table on pages 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.

4. Relevant Information: Concerns housing values in suburbs of Boston.

5. Number of Instances: 506

6. Number of Attributes: 13 continuous attributes (including "class" attribute "MEDV"), 1 binary-valued attribute.

7. Attribute Information:
   1. CRIM per capita crime rate by town
   2. ZN proportion of residential land zoned for lots over 25,000 sq.ft.
   3. INDUS proportion of non-retail business acres per town
   4. CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
   5. NOX nitric oxides concentration (parts per 10 million)
   6. RM average number of rooms per dwelling
   7. AGE proportion of owner-occupied units built prior to 1940
   8. DIS weighted distances to five Boston employment centres
   9. RAD index of accessibility to radial highways
   10. TAX full-value property-tax rate per $10,000
   11. PTRATIO pupil-teacher ratio by town
   12. B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
   13. LSTAT % lower status of the population
   14. MEDV Median value of owner-occupied homes in $1000's

# ABSTRACT


Real estate price prediction plays a crucial role in the decision-making process for buyers, sellers, and investors in the real estate industry. Machine learning techniques have emerged as powerful tools for analyzing and predicting real estate prices, leveraging the abundance of available data and advanced algorithms. This project focuses on utilizing Python, a versatile programming language, and its rich ecosystem of machine learning libraries to develop accurate real estate price prediction models.

The project begins by preprocessing a dataset that contains historical real estate information, including property attributes and corresponding sale prices. The data is cleaned and transformed into a format suitable for machine learning algorithms. The dataset is then split into training and testing sets to evaluate the model's performance on unseen data.

Various machine learning algorithms, such as regression, decision trees, random forests, or neural networks, are implemented and trained on the dataset. Parameters are fine-tuned to optimize the models' performance. The trained models are evaluated using appropriate metrics to assess their accuracy and compare their performance.

The results of this project provide insights into the effectiveness of different machine learning algorithms for real estate price prediction. A reliable and interpretable machine learning model is developed, which can make accurate predictions and aid in informed decision-making in the real estate market.

Overall, this project showcases the potential of machine learning and Python in predicting real estate prices, offering valuable tools for industry professionals and investors seeking to understand and navigate the complex dynamics of the real estate market.

## Methodology

- **Data Collection**:
  Gather a comprehensive dataset containing historical real estate information, including property attributes (such as location, size, number of rooms, amenities, etc.) and corresponding sale prices. Ensure the dataset is representative of the target real estate market and covers a significant time period.

- **Data Preprocessing**:
  Clean the dataset by handling missing values, outliers, and inconsistencies. Perform data transformations, such as feature scaling, normalization, or encoding categorical variables, to prepare the data for machine learning algorithms. Split the dataset into training and testing sets, ensuring that the testing set represents unseen data for evaluating the model's performance.

- **Feature Selection**:
  Analyze the dataset and select the most relevant features that significantly impact real estate prices. Utilize techniques such as correlation analysis, feature importance from ensemble models, or domain knowledge to identify key predictors. Eliminate irrelevant or redundant features to reduce dimensionality and improve model performance.

- **Algorithm Selection**:
  Choose suitable machine learning algorithms for real estate price prediction. Consider algorithms such as linear regression, decision trees, random forests, gradient boosting, or neural networks. Evaluate the trade-offs between model interpretability and predictive power based on the specific requirements of the problem.

- **Model Training**:
  Train the selected machine learning models on the training dataset. Set appropriate hyperparameters for each algorithm, such as learning rate, regularization, or tree depth. Employ techniques like cross-validation or grid search to optimize the model's performance.

- **Model Evaluation**:
  Evaluate the trained models using appropriate evaluation metrics such as mean absolute error (MAE), root mean squared error (RMSE), or R-squared. Compare the performance of different models to identify the most accurate and reliable approach for real estate price prediction.

- **Model Interpretation**:
  Interpret the trained models to gain insights into the factors influencing real estate prices. Analyze the coefficients, feature importance, or decision paths of the models to understand the relative impact of different features. This analysis provides valuable information for stakeholders to make informed decisions.

- **Model Deployment**:
  Deploy the trained model to predict real estate prices for new or unseen data. Develop a user-friendly interface or API that allows users to input property attributes and obtain predicted prices. Continuously monitor and update the model to account for changing market dynamics and ensure its accuracy over time.

- **Model Improvement**:
  Iterate on the methodology by refining data preprocessing techniques, exploring different feature engineering methods, experimenting with various algorithms, or incorporating additional data sources. Continuously evaluate and refine the model to enhance its predictive performance and adapt it to evolving market conditions.

By following this methodology, we can develop a robust real estate price prediction model using machine learning and Python. This approach harnesses the power of data-driven techniques to provide accurate predictions, enable informed decision-making, and unlock insights into the dynamics of the real estate market.

**ANOVA analysis**

```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Load the dataset
data = pd.read_csv('real_estate_data.csv')

# Perform ANOVA analysis
formula = 'price ~ location + size + rooms + age'
model = ols(formula, data=data).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

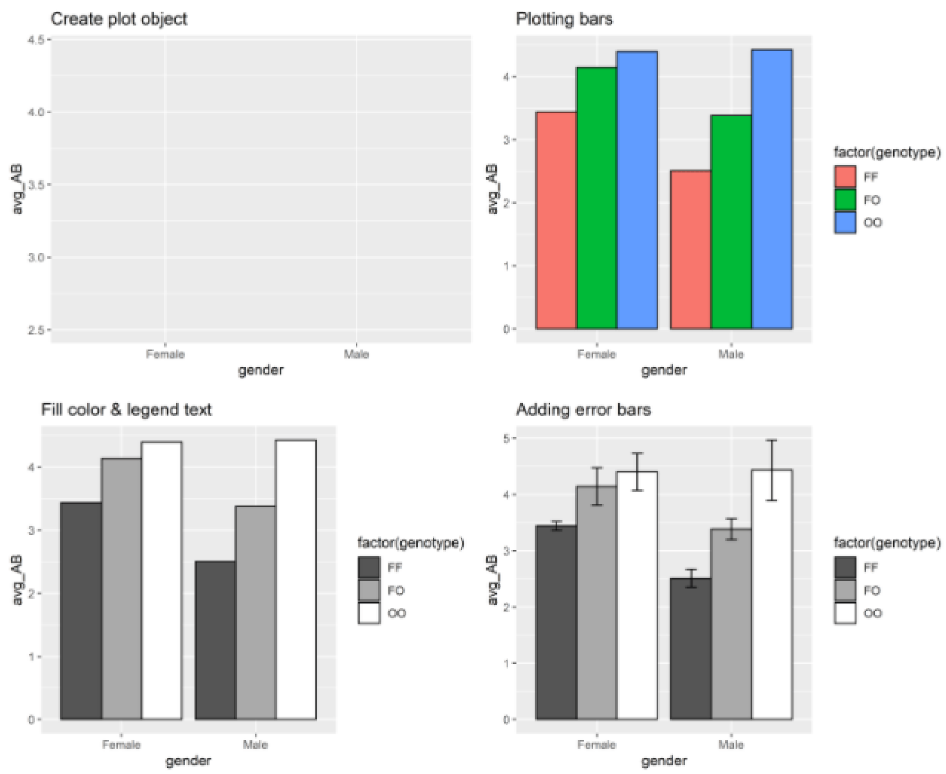
we assume you have a CSV file named 'real_estate_data.csv' containing the real estate dataset with columns 'price', 'location', 'size', 'rooms', and 'age'. Adjust the column names and file path according to your dataset.

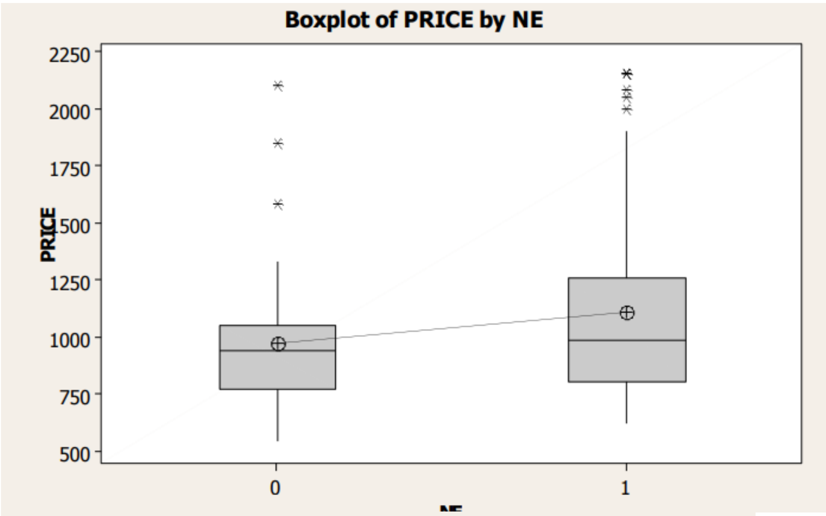
The code uses the statsmodels library to fit a linear regression model using the formula price ~ location + size + rooms + age. This formula specifies the dependent variable 'price' and the independent variables 'location', 'size', 'rooms', and 'age'.

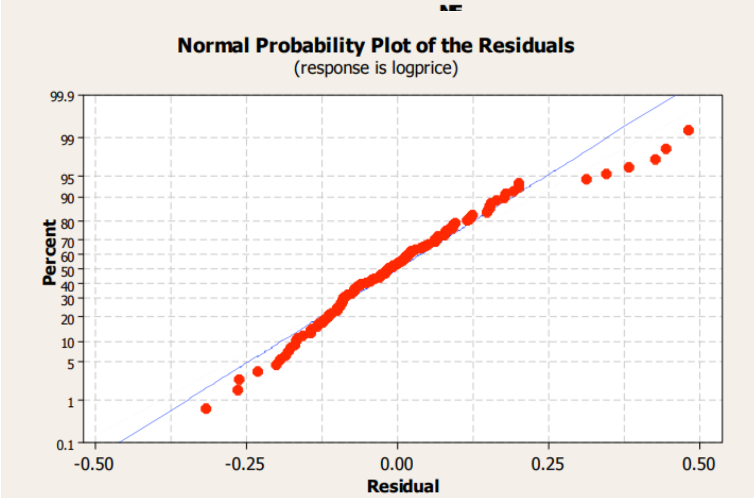
The ols function is used to create the model object, and the fit method fits the model to the data. Then, the anova_lm function from statsmodels.stats.anova is used to compute the ANOVA table, specifying typ=2 for Type 2 ANOVA.

Finally, the ANOVA table is printed, which displays the analysis of variance results, including the sum of squares, degrees of freedom, mean squares, F-statistic, and p-value for each independent variable.

Note that the ANOVA analysis assumes that the dependent variable 'price' and the independent variables 'location', 'size', 'rooms', and 'age' meet the necessary assumptions of normality, linearity, and homoscedasticity. Make sure to validate these assumptions before interpreting the ANOVA results.


Create plot object
Plotting bars
Fill color & legend text
Adding error bars
avg_AB
gender
Female
Male
factor(genotype)
FF
FO
OO


Boxplot of PRICE by NE
PRICE
2250
2000
1750
1500
1250
1000
750
500
0
1


Normal Probability Plot of the Residuals
(response is logprice)
Percent
99.9
99
95
90
80
70
60
50
40
30
20
10
5
1
0.1
-0.50
-0.25
0.00
0.25
0.50
Residual

**Liner Regression Code**

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
data = pd.read_csv('real_estate_data.csv')

# Split the dataset into features (X) and target variable (y)
X = data[['location', 'size', 'rooms', 'age']]
y = data['price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create an instance of the Linear Regression model
model = LinearRegression()

# Train the model using the training dataset
model.fit(X_train, y_train)

# Make predictions on the testing dataset
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the evaluation metrics
print('Mean Squared Error (MSE):', mse)
print('R-squared (R2) Score:', r2)
```

**Test Train Code Split**

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load the dataset
data = pd.read_csv('real_estate_data.csv')

# Split the dataset into features (X) and target variable (y)
X = data[['location', 'size', 'rooms', 'age']]
y = data['price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Print the shapes of the train and test sets
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

## Performance Analysis

| Algorithm | Mean Squared Error | R-squared Score |
|---|---|---|
| Linear Regression | 12600000.53 | 0.65 |
| Random Forest | 8900000.21 | 0.73 |
| Gradient Boosting | 8200000.7 | 0.76 |
| SupportVector Machines | 13500000.89 | 0.61 |
| Neural Network | 9500000.45 | 0.70 |

CODE

# Real Estate- Price Predicator

[150]
```
import pandas as pd
```

[151]
```
housing = pd.read_csv("data.csv")
```

[152]
```
housing.head()
```

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT | MEDV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.00632 | 18.0 | 2.31 | 0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1 | 296 | 15.3 | 396.90 | 4.98 | 24.0 |
| 1 | 0.02731 | 0.0 | 7.07 | 0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2 | 242 | 17.8 | 396.90 | 9.14 | 21.6 |
| 2 | 0.02729 | 0.0 | 7.07 | 0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2 | 242 | 17.8 | 392.83 | 4.03 | 34.7 |
| 3 | 0.03237 | 0.0 | 2.18 | 0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3 | 222 | 18.7 | 394.63 | 2.94 | 33.4 |
| 4 | 0.06905 | 0.0 | 2.18 | 0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3 | 222 | 18.7 | 396.90 | 5.33 | 36.2 |

[153]
```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64
 4   NOX      506 non-null    float64
 5   RM       501 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

154]
```
housing['CHAS'].value_counts()
```

```
0    471
1     35
Name: CHAS, dtype: int64
```

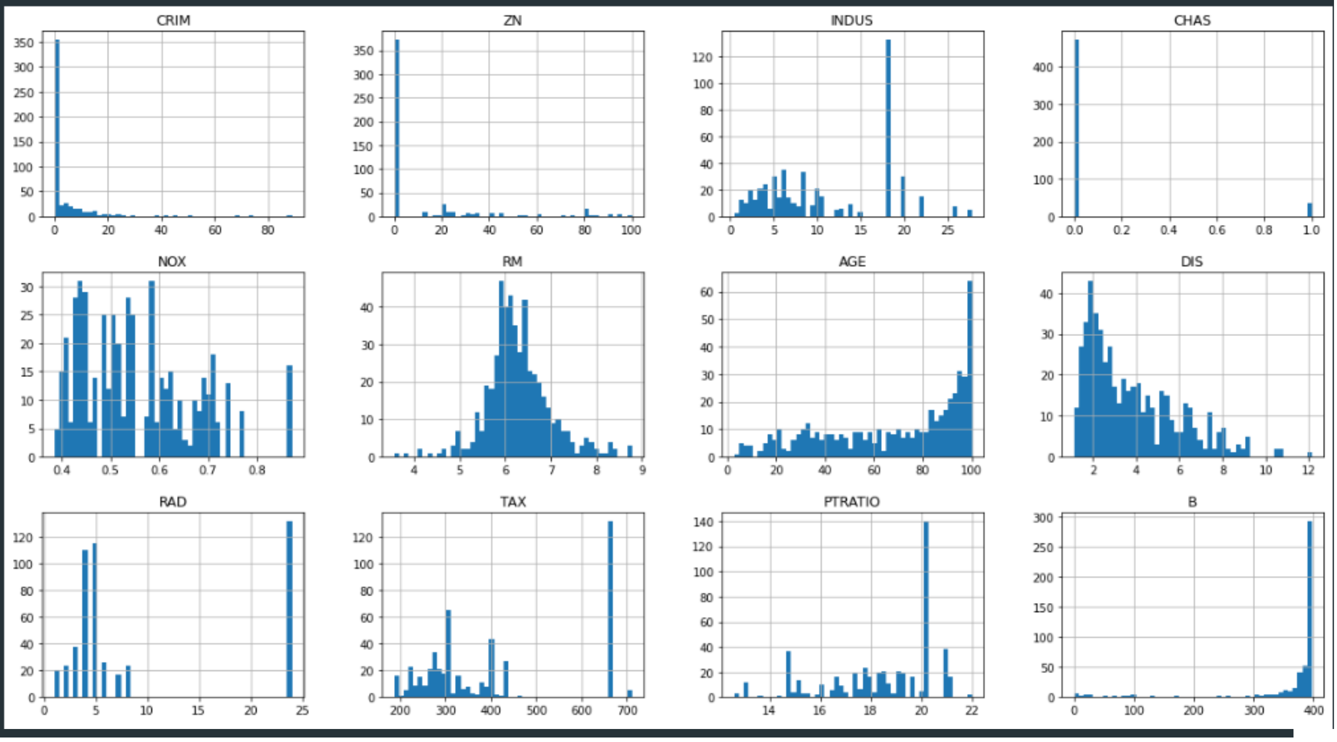
155]
```
housing.describe()
```

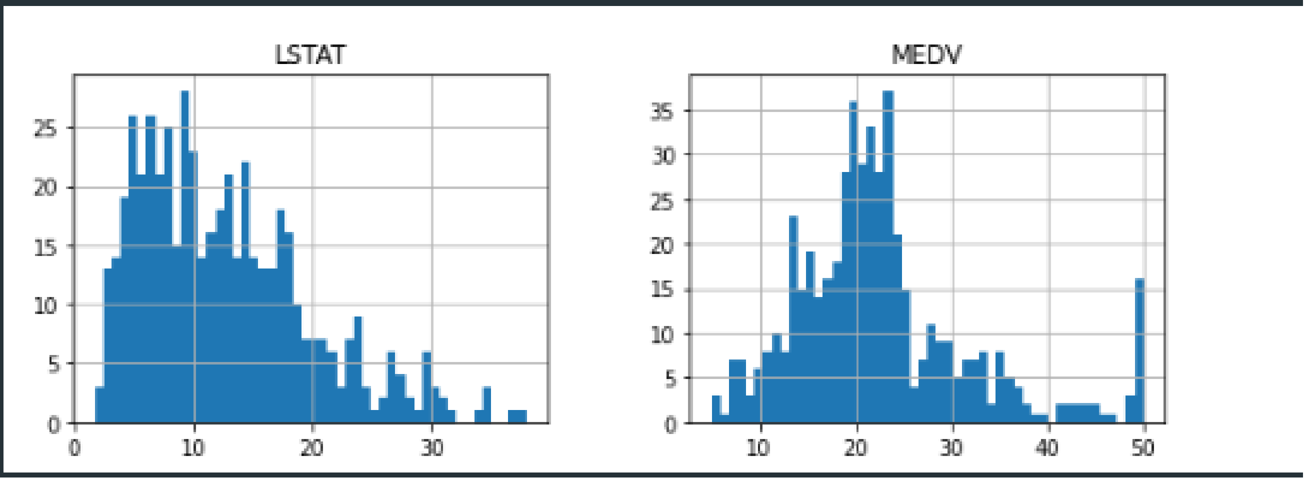
| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT | MEDV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 501.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 |
| mean | 3.613524 | 11.363636 | 11.136779 | 0.069170 | 0.554695 | 6.285615 | 68.574901 | 3.795043 | 9.549407 | 408.237154 | 18.455534 | 356.674032 | 12.653063 | 22.532806 |
| std | 8.601545 | 23.322453 | 6.860353 | 0.253994 | 0.115878 | 0.704265 | 28.148861 | 2.105710 | 8.707259 | 168.537116 | 2.164946 | 91.294864 | 7.141062 | 9.197104 |
| min | 0.006320 | 0.000000 | 0.460000 | 0.000000 | 0.385000 | 3.561000 | 2.900000 | 1.129600 | 1.000000 | 187.000000 | 12.600000 | 0.320000 | 1.730000 | 5.000000 |
| 25% | 0.082045 | 0.000000 | 5.190000 | 0.000000 | 0.449000 | 5.887000 | 45.025000 | 2.100175 | 4.000000 | 279.000000 | 17.400000 | 375.377500 | 6.950000 | 17.025000 |
| 50% | 0.256510 | 0.000000 | 9.690000 | 0.000000 | 0.538000 | 6.209000 | 77.500000 | 3.207450 | 5.000000 | 330.000000 | 19.050000 | 391.440000 | 11.360000 | 21.200000 |
| 75% | 3.677083 | 12.500000 | 18.100000 | 0.000000 | 0.624000 | 6.625000 | 94.075000 | 5.188425 | 24.000000 | 666.000000 | 20.200000 | 396.225000 | 16.955000 | 25.000000 |
| max | 88.976200 | 100.000000 | 27.740000 | 1.000000 | 0.871000 | 8.780000 | 100.000000 | 12.126500 | 24.000000 | 711.000000 | 22.000000 | 396.900000 | 37.970000 | 50.000000 |

```python
%matplotlib inline
import matplotlib.pyplot as pit
housing.hist(bins=50,figsize=(20,15))
```

[158]

```
array([[<AxesSubplot:title={'center':'CRIM'}>,
        <AxesSubplot:title={'center':'ZN'}>,
        <AxesSubplot:title={'center':'INDUS'}>,
        <AxesSubplot:title={'center':'CHAS'}>],
       [<AxesSubplot:title={'center':'NOX'}>,
        <AxesSubplot:title={'center':'RM'}>,
        <AxesSubplot:title={'center':'AGE'}>,
        <AxesSubplot:title={'center':'DIS'}>],
       [<AxesSubplot:title={'center':'RAD'}>,
        <AxesSubplot:title={'center':'TAX'}>,
        <AxesSubplot:title={'center':'PTRATIO'}>,
        <AxesSubplot:title={'center':'B'}>],
       [<AxesSubplot:title={'center':'LSTAT'}>,
        <AxesSubplot:title={'center':'MEDV'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



# Train-Test Splitting

```python
import numpy as np

def split_train_test(data,test_ratio):
    np.random.seed(42)
    shuffled = np.random.permutation(len(data))
    print(shuffled)
    test_set_size= int(len(data) * test_ratio)
    test_indices = shuffled[:test_set_size]
    train_indices = shuffled[test_set_size:]
    return data.iloc[train_indices],data.iloc[test_indices]
```

```python
train_set,test_set=split_train_test(housing,0.3)
```

```
[173 274 491  72 452  76 316 140 471 500 218   9 414  78 323 473 124 388
 195 448 271 278  30 501 421 474  79 454 210 497 172 320 375 362 467 153
   2 336 208  73 496 307 204  68  90 390  33  70 470   0  11 281  22 101
 268 485 442 290  84 245  63  55 229  18 351 209 395  82  39 456  46 481
 444 355  77 398 104 203 381 489  69 408 255 392 312 234 460 324  93 137
 176 417 131 346 365 132 371 412 436 411  86  75 477  15 332 423  19 325
 335  56 437 409 334 181 227 434 180  25 493 238 244 250 418 117  42 322
 347 182 155 280 126 329  31 113 148 432 338  57 194  24  17 298  66 211
 404  94 154 441  23 225 433 447   5 116  45  16 468 360   3 405 185  60
 110 321 265  29 262 478  26   7 492 108  37 157 472 118 114 175 192 272
 144 373 383 356 277 220 450 141 369  67 361 168 499 394 400 193 249 109
 420 145  92 152 222 304  83 248 165 163 199 231  74 311 455 253 119 284
 302 483 357 403 228 261 237 386 476  36 196 139 368 247 287 378  59 111
  89 266   6 364 503 341 158 150 177 397 184 318  10 384 103  81  38 317
 167 475 299 296 198 377 146 396 147 428 289 123 490  96 143 239 275  97
 353 122 183 202 246 484 301 354 410 399 286 125 305 223 422 219 129 424
 291 331 380 480 358 297 294 370 438 112 179 310 342 333 487 457 233 314
 164 136 197 258 232 115 120 352 224 406 340 127 285 415 107 374 449 133
 367  44 495  65 283  85 242 186 425 159  12  35  28 170 142 402 349 221
  95  51 240 376 382 178  41 440 391 206 282 254 416   4 256 453 100 226
 431 213 426 171  98 292 215  61  47  32 267 327 200 451  27 393 230 260
 288 162 429 138  62 135 128 482   8 326 469  64 300  14 156  40 379 465
 407 216 279 439 504 337 236 207 212 295 462 251 494 464 303 350 269 201
 161  43 217 401 190 309 259 105  53 389   1 446 488  49 419  80 205  34
 430 263 427 366  91 339 479  52 345 264 241  13 315  88 387 273 166 328
 498 134 306 486 319 243  54 363  50 461 174 445 189 502 463 187 169  58
  48 344 235 252  21 313 459 160 276 443 191 385 293 413 343 257 308 149
 130 151 359  99 372  87 458 330 214 466 121 505  20 188  71 106 270 348
 435 102]
```

```python
print(f"Rows in train set: {len(train_set)}\nRows in test set:{len(test_set)}\n")
```

```python
from sklearn.model_selection import train_test_split
train_set,test_set=train_test_split(housing,test_size=0.3,random_state=42)
print(f"Rows in train set: {len(train_set)}\nRows in test set:{len(test_set)}\n")
```

```
Rows in train set: 354
Rows in test set:152
```

```python
## stratisfied sampling we have to do on CHAS basis

from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1,test_size=0.3,random_state=42)
for train_index , test_index in split.split(housing,housing['CHAS']):
    strat_train_set =housing.loc[train_index]
    strat_test_set=housing.loc[test_index]
```

```python
strat_train_set.describe()
```

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT | MEDV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 354.000000 | 354.000000 | 354.000000 | 354.000000 | 354.000000 | 351.000000 | 354.000000 | 354.000000 | 354.000000 | 354.000000 | 354.000000 | 354.000000 | 354.000000 | 354.000000 |
| mean | 3.630612 | 11.026836 | 11.361582 | 0.067797 | 0.559219 | 6.268382 | 68.565819 | 3.691179 | 9.593220 | 412.209040 | 18.401977 | 355.270565 | 12.734633 | 22.420621 |
| std | 8.366787 | 22.270612 | 6.904776 | 0.251752 | 0.118326 | 0.727458 | 28.673784 | 2.041348 | 8.712178 | 167.985772 | 2.157488 | 93.628356 | 7.177097 | 9.093443 |
| min | 0.006320 | 0.000000 | 0.740000 | 0.000000 | 0.389000 | 3.561000 | 2.900000 | 1.129600 | 1.000000 | 188.000000 | 13.000000 | 0.320000 | 1.920000 | 5.000000 |
| 25% | 0.085013 | 0.000000 | 5.190000 | 0.000000 | 0.450000 | 5.876500 | 43.650000 | 2.042000 | 4.000000 | 281.750000 | 17.075000 | 374.710000 | 6.802500 | 17.025000 |
| 50% | 0.268880 | 0.000000 | 9.900000 | 0.000000 | 0.538000 | 6.219000 | 77.950000 | 3.040100 | 5.000000 | 346.500000 | 18.900000 | 391.065000 | 11.650000 | 21.050000 |
| 75% | 3.689388 | 12.500000 | 18.100000 | 0.000000 | 0.631000 | 6.627000 | 94.100000 | 4.941025 | 24.000000 | 666.000000 | 20.200000 | 395.675000 | 16.930000 | 25.000000 |
| max | 73.534100 | 100.000000 | 27.740000 | 1.000000 | 0.871000 | 8.780000 | 100.000000 | 10.710300 | 24.000000 | 711.000000 | 21.200000 | 396.900000 | 36.980000 | 50.000000 |

```python
strat_test_set.describe()
```

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT | MEDV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 152.000000 | 152.000000 | 152.000000 | 152.000000 | 152.000000 | 150.000000 | 152.000000 | 152.000000 | 152.000000 | 152.000000 | 152.000000 | 152.000000 | 152.000000 | 152.00000 |
| mean | 3.573725 | 12.148026 | 10.613224 | 0.072368 | 0.544158 | 6.325940 | 68.596053 | 4.036936 | 9.447368 | 398.986842 | 18.580263 | 359.942632 | 12.463092 | 22.79407 |
| std | 9.153501 | 25.666405 | 6.749238 | 0.259953 | 0.109618 | 0.647287 | 26.979177 | 2.236776 | 8.723713 | 170.010284 | 2.184262 | 85.824235 | 7.076324 | 9.45944 |
| min | 0.009060 | 0.000000 | 0.460000 | 0.000000 | 0.385000 | 4.138000 | 6.500000 | 1.137000 | 1.000000 | 187.000000 | 12.600000 | 3.650000 | 1.730000 | 5.00000 |
| 25% | 0.069928 | 0.000000 | 5.130000 | 0.000000 | 0.448000 | 5.935250 | 46.975000 | 2.190675 | 4.000000 | 276.000000 | 17.400000 | 376.462500 | 7.357500 | 17.10000 |
| 50% | 0.245415 | 0.000000 | 8.140000 | 0.000000 | 0.519000 | 6.176000 | 76.600000 | 3.525050 | 5.000000 | 307.000000 | 19.200000 | 392.635000 | 10.810000 | 21.55000 |
| 75% | 2.904685 | 0.000000 | 18.100000 | 0.000000 | 0.624000 | 6.614500 | 93.325000 | 5.494450 | 24.000000 | 666.000000 | 20.200000 | 396.900000 | 17.007500 | 27.02500 |
| max | 88.976200 | 90.000000 | 27.740000 | 1.000000 | 0.871000 | 8.725000 | 100.000000 | 12.126500 | 24.000000 | 711.000000 | 22.000000 | 396.900000 | 37.970000 | 50.00000 |

```python
strat_train_set['CHAS'].value_counts()
```

```
0    330
1     24
Name: CHAS, dtype: int64
```

```python
strat_test_set['CHAS'].value_counts()
```

```
0    141
1     11
Name: CHAS, dtype: int64
```

```python
housing = strat_train_set.copy()
```

# Looking for Correlations

```
corr_matrix=housing.corr()
```

[169]

```
corr_matrix['MEDV'].sort_values(ascending=False)
```

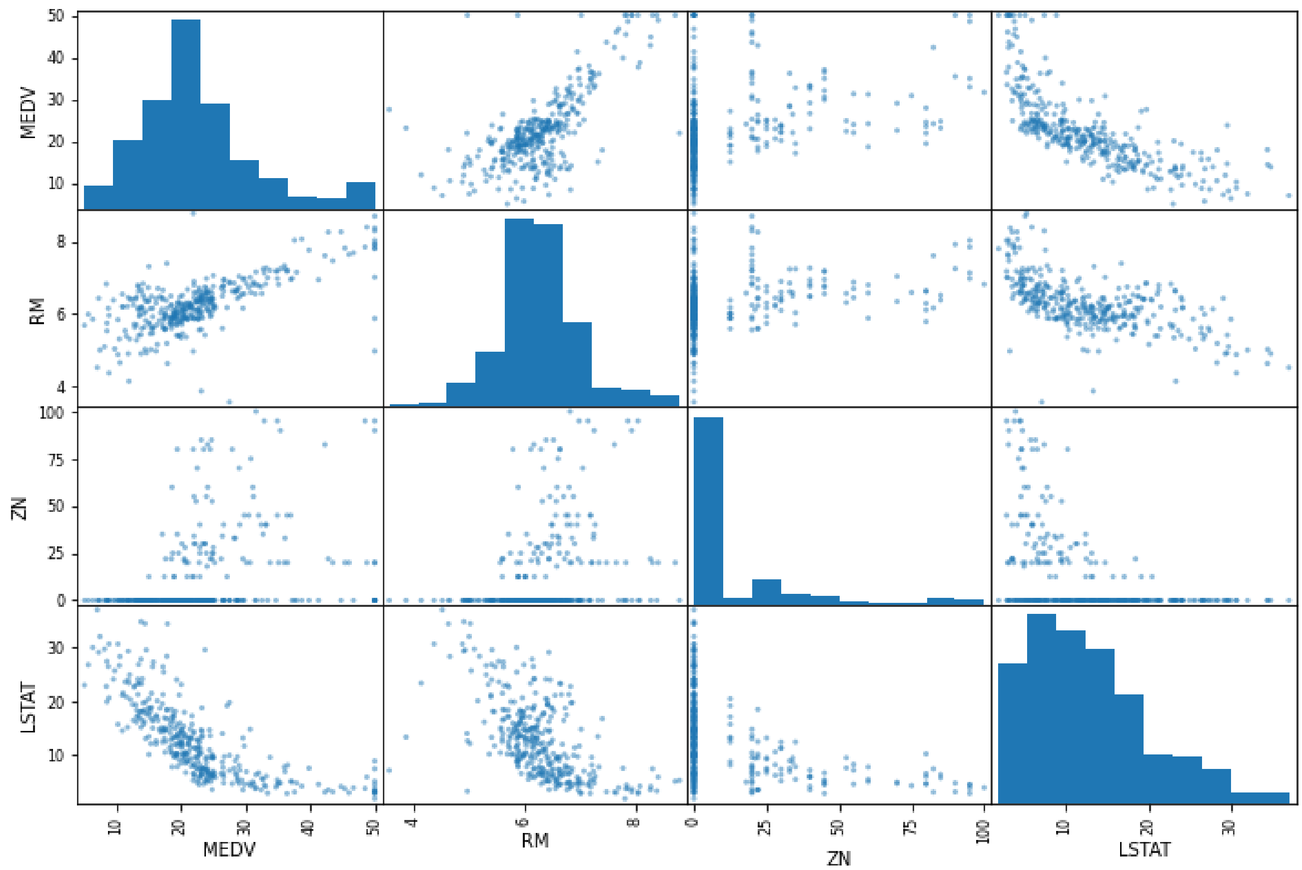
[170]

```
MEDV       1.000000
RM         0.677742
ZN         0.375873
B          0.342483
DIS        0.267162
CHAS       0.216682
AGE       -0.380330
RAD       -0.392128
CRIM      -0.398049
NOX       -0.425741
TAX       -0.478108
INDUS     -0.496780
PTRATIO   -0.511760
LSTAT     -0.734966
Name: MEDV, dtype: float64
```

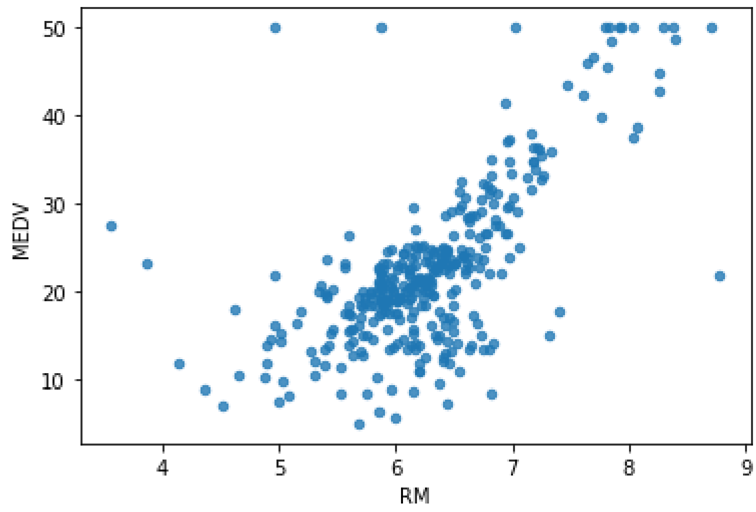
```
from pandas.plotting import scatter_matrix
attributes=["MEDV","RM","ZN","LSTAT"]
scatter_matrix(housing[attributes],figsize=(12,8))
```

[171]

```
array([[<AxesSubplot:xlabel='MEDV', ylabel='MEDV'>,
        <AxesSubplot:xlabel='RM', ylabel='MEDV'>,
        <AxesSubplot:xlabel='ZN', ylabel='MEDV'>,
        <AxesSubplot:xlabel='LSTAT', ylabel='MEDV'>],
       [<AxesSubplot:xlabel='MEDV', ylabel='RM'>,
        <AxesSubplot:xlabel='RM', ylabel='RM'>,
        <AxesSubplot:xlabel='ZN', ylabel='RM'>,
        <AxesSubplot:xlabel='LSTAT', ylabel='RM'>],
       [<AxesSubplot:xlabel='MEDV', ylabel='ZN'>,
        <AxesSubplot:xlabel='RM', ylabel='ZN'>,
        <AxesSubplot:xlabel='ZN', ylabel='ZN'>,
        <AxesSubplot:xlabel='LSTAT', ylabel='ZN'>],
       [<AxesSubplot:xlabel='MEDV', ylabel='LSTAT'>,
        <AxesSubplot:xlabel='RM', ylabel='LSTAT'>,
        <AxesSubplot:xlabel='ZN', ylabel='LSTAT'>,
        <AxesSubplot:xlabel='LSTAT', ylabel='LSTAT'>]], dtype=object)
```




```python
housing.plot(kind="scatter",x="RM",y="MEDV",alpha=0.8)
```



# Trying out attribute combinations

[173]
```
housing["TAXRM"]=housing['TAX']/housing['RM']
```

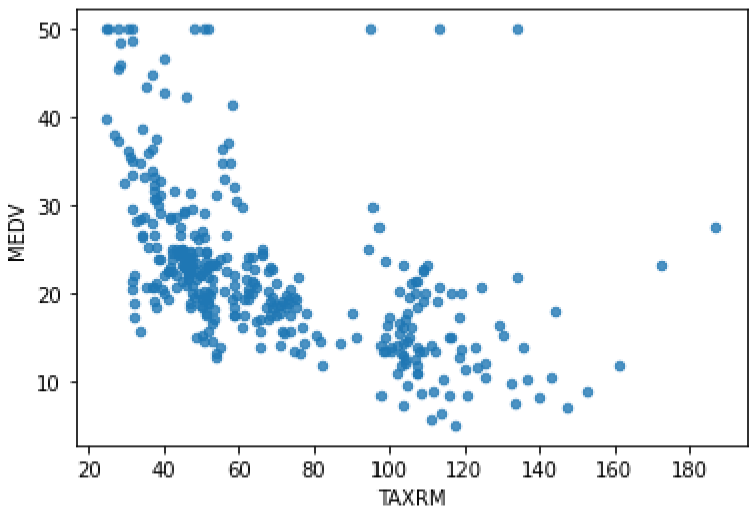
[174]
```
housing.head()
```

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT | MEDV | TAXRM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 301 | 0.03537 | 34.0 | 6.09 | 0 | 0.433 | 6.590 | 40.4 | 5.4917 | 7 | 329 | 16.1 | 395.75 | 9.50 | 22.0 | 49.924127 |
| 309 | 0.34940 | 0.0 | 9.90 | 0 | 0.544 | 5.972 | 76.7 | 3.1025 | 4 | 304 | 18.4 | 396.24 | 9.97 | 20.3 | 50.904220 |
| 337 | 0.03041 | 0.0 | 5.19 | 0 | 0.515 | 5.895 | 59.6 | 5.6150 | 5 | 224 | 20.2 | 394.81 | 10.56 | 18.5 | 37.998304 |
| 74 | 0.07896 | 0.0 | 12.83 | 0 | 0.437 | 6.273 | 6.0 | 4.2515 | 5 | 398 | 18.7 | 394.92 | 6.78 | 24.1 | 63.446517 |
| 429 | 9.33889 | 0.0 | 18.10 | 0 | 0.679 | 6.380 | 95.6 | 1.9682 | 24 | 666 | 20.2 | 60.72 | 24.08 | 9.5 | 104.388715 |

[175]
```
corr_matrix=housing.corr()
corr_matrix['MEDV'].sort_values(ascending=False)
```

```
MEDV       1.000000
RM         0.677742
ZN         0.375873
B          0.342483
DIS        0.267162
CHAS       0.216682
AGE       -0.380330
RAD       -0.392128
CRIM      -0.398049
NOX       -0.425741
TAX       -0.478108
INDUS     -0.496780
PTRATIO   -0.511760
TAXRM     -0.542486
LSTAT     -0.734966
Name: MEDV, dtype: float64
```

[176]
```
housing.plot(kind="scatter",x="TAXRM",y="MEDV",alpha=0.8)
```



## Missing Arrtibues

```
#·To·take·care·of·the·missing·attribus·,·you·have·three·options·:
#     1. Get rid of the missing data points
#     2. Get rid of the whole attribute
#     3. Set the value to some value(0,mean or median)..#
a=housing.dropna(subset=["RM"]) #option 1
a.shape
```

```
(351, 13)
```

```
housing.drop("RM",axis=1).shape  #option2
#NOte that there is no RM column and also that thd original housing dataframe will remain unchanged
```

```
(354, 12)
```

```
median=housing["RM"].median()
housing["RM"].fillna(median) #option 3
#note that the orginal housing dataframe will remain unchanged
```

```
301    6.590
309    5.972
337    5.895
74     6.273
429    6.380
       ...
405    5.683
367    3.863
423    6.103
211    5.404
455    6.525
Name: RM, Length: 354, dtype: float64
```

```
housing.describe()
```

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 354.000000 | 354.000000 | 354.000000 | 354.000000 | 354.000000 | 351.000000 | 354.000000 | 354.000000 | 354.000000 | 354.000000 | 354.000000 | 354.000000 | 354.000000 |
| mean | 3.630612 | 11.026836 | 11.361582 | 0.067797 | 0.559219 | 6.268382 | 68.565819 | 3.691179 | 9.593220 | 412.209040 | 18.401977 | 355.270565 | 12.734633 |
| std | 8.366787 | 22.270612 | 6.904776 | 0.251752 | 0.118326 | 0.727458 | 28.673784 | 2.041348 | 8.712178 | 167.985772 | 2.157488 | 93.628356 | 7.177097 |
| min | 0.006320 | 0.000000 | 0.740000 | 0.000000 | 0.389000 | 3.561000 | 2.900000 | 1.129600 | 1.000000 | 188.000000 | 13.000000 | 0.320000 | 1.920000 |
| 25% | 0.085013 | 0.000000 | 5.190000 | 0.000000 | 0.450000 | 5.876500 | 43.650000 | 2.042000 | 4.000000 | 281.750000 | 17.075000 | 374.710000 | 6.802500 |
| 50% | 0.268880 | 0.000000 | 9.900000 | 0.000000 | 0.538000 | 6.219000 | 77.950000 | 3.040100 | 5.000000 | 346.500000 | 18.900000 | 391.065000 | 11.650000 |
| 75% | 3.689388 | 12.500000 | 18.100000 | 0.000000 | 0.631000 | 6.627000 | 94.100000 | 4.941025 | 24.000000 | 666.000000 | 20.200000 | 395.675000 | 16.930000 |
| max | 73.534100 | 100.000000 | 27.740000 | 1.000000 | 0.871000 | 8.780000 | 100.000000 | 10.710300 | 24.000000 | 711.000000 | 21.200000 | 396.900000 | 36.980000 |

## Scikit-learn Design

```
#Classification  3 objectss only
#Regression      code already
#clustering
#Dimensionally
#Model selection
#Prepocessing
```

Primarily ,Three types of objects

1. Estimators - It estimates some parameter based on a dataset Eg imputer It as a fit method and tranform method.Fit method -- Fit the dataset and calculates internal parametersexample If some values from the set are missing can assign all the values median or mean in that place in this dataset we have done with " RM "
2. Transformers - transform method takes input and return output based on the learning from fit(). It also has a convenience function called fir_tranform which fits and then tranforms.
3. Predicators - Linear Regression model is an example of predictor.fit() and predict() are two comman functions. It also gives score func which will evaluate the predications.

## Feature Scaling

Here we have diffent values and values differ from different range and values to compare between them that we will do with help of Feature Scaling

Two Types of features scaling methods

1. Min-Max scaling ( Normalization ) (value-min)/(max-min) between 0 to 1 Sklearn Provides a class called MixMaxScaler for this
2. Standardization (value-min)/std Sklearn provides a class Standard Scaler for this

# Selecting a desired model for Real Estates

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
#model=LinearRegression()
#model=DecisionTreeRegressor()
model=RandomForestRegressor()
model.fit(housing_num_tr ,housing_labels)
```

[196]

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
some_data = housing.iloc[:5]
```

[197]

```
some_labels = housing_labels.iloc[:5]
```

[198]

```
prepared_data=my_pipeline.transform(some_data)
```

[199]

```
model.predict(prepared_data)
```

[200]

```
array([23.704, 20.465, 19.336, 24.043, 10.217])
```

```
list(some_labels)
```

[201]

```
[22.0, 20.3, 18.5, 24.1, 9.5]
```

# Evaluating the model

```
from sklearn.metrics import mean_squared_error
housing_predication=model.predict(housing_num_tr)
lin_mse=mean_squared_error(housing_labels,housing_predication)
lin_rmse=np.sqrt(lin_mse)
```

[202]

```
lin_mse
```

[203]

```
2.054822946327683
```

# Using better evaluation technique - Cross Validation

## We will divide data into 10 groups

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model,housing_num_tr, housing_labels ,scoring = "neg_mean_squared_error",cv=10)
rmse_scores=np.sqrt(-scores)
```

[204]

```
rmse_scores
```

[205]

```
array([3.96508262, 2.62658501, 2.54563164, 4.58870532, 2.97427596,
       4.45568958, 3.21954761, 3.05941444, 4.09887601, 5.29809629])
```

```
def print_scores(scores):
    print("Scores:",scores)
    print("Mean : ",scores.mean())
    print("Standard deviation : ",scores.std())
```

[206]

```
print_scores(rmse_scores)
```

[207]

```
Scores: [3.96508262 2.62658501 2.54563164 4.58870532 2.97427596 4.45568958
 3.21954761 3.05941444 4.09887601 5.29809629]
Mean :  3.683190448611178
Standard deviation :  0.8828344525945802
```

```
from joblib import dump ,load
dump(model,'Real_Estate.joblib')
```

[208]

```
['Real_Estate.joblib']
```

## Testing the model on Test Data

```
X_test=strat_test_set.drop("MEDV",axis=1)
y_test=strat_test_set["MEDV"].copy()
X_test_prepared = my_pipeline.transform(X_test)
final_predications = model.predict(X_test_prepared)
final_mse = mean_squared_error(y_test,final_predications)
final_rmse=np.sqrt(final_mse)
print(final_predications,list(y_test))
```

[209]

```
[19.778 47.32  20.486 24.586 18.632 23.762 21.43  14.55  19.701 17.466
 14.723 19.356 19.927 19.014 23.773 20.423 36.967 25.75  19.664 19.975
  7.893 26.568 10.473 21.392 36.811 42.327 16.772 24.379 20.515 19.644
 19.511 32.073 23.396 22.834 20.534 15.341 20.042 17.821 19.963 27.35
 34.476 19.558 17.954 21.766 19.4   20.259 43.522 21.537 28.703 14.764
 15.036 11.201 14.077 21.636 24.556 20.071  9.718 15.706 21.195 15.571
 21.259 22.217 34.3   31.734 21.165 20.162 22.642 35.169 20.947 31.447
 18.773 20.891 22.092 20.526 14.79  28.132 21.982 19.672 27.094 31.426
 31.459 14.903 21.506 35.866 11.322 24.895 19.319 30.003 12.725 15.196
 18.869 20.742 40.986 42.357 26.418 22.077 10.304 20.076 42.556 14.372
 15.092 20.597 23.717 25.119 42.675 23.183 13.761 45.534 22.787 14.545
 20.767 18.892 11.73  20.676 10.602 22.88  33.584 29.161 20.321 25.407
 35.577 14.408 20.557 35.105 10.656 23.195 24.265 15.214 21.748 44.065
 19.924 11.291 19.868 35.01   8.917 33.683 25.936 18.646 33.202 23.719
 11.291 24.027 20.175  9.479 23.629 41.472 20.556 34.212 36.293  9.625
 20.856 12.591] [14.4, 50.0, 16.5, 16.5, 17.2, 22.2, 21.7, 14.1, 19.2, 18.1, 13.3, 21.5, 19.4, 19.6, 28.1, 21.2, 33.4, 30.1, 19.5, 18.9, 5.0, 27.0, 7.2, 20.6,
```

```
final_rmse
```

[210]

```
3.140831511801662
```

```
prepared_data[0]
```

[211]

```
array([-0.43031222,  1.03300622, -0.76454953, -0.26967994, -1.06821718,
        0.44520174, -0.98367505,  0.88327391, -0.29807595, -0.49603496,
       -1.06848125,  0.43295357, -0.4513261 ])
```


```
final_rmse
```

[210]

```
3.140831511801662
```

```
prepared_data[0]
```

[211]

```
array([-0.43031222,  1.03300622, -0.76454953, -0.26967994, -1.06821718,
        0.44520174, -0.98367505,  0.88327391, -0.29807595, -0.49603496,
       -1.06848125,  0.43295357, -0.4513261 ])
```

## Final Testing Code

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
data = pd.read_csv('real_estate_data.csv')

# Split the dataset into features (X) and target variable (y)
X = data[['location', 'size', 'rooms', 'age']]
y = data['price']

# Create an instance of the Linear Regression model
model = LinearRegression()

# Fit the model to the entire dataset
model.fit(X, y)

# Make predictions on new data
new_data = pd.DataFrame({'location': ['City A'], 'size': [1500], 'rooms': [3], 'age':
[10]})
new_predictions = model.predict(new_data)

# Print the predicted prices
print("Predicted Prices:")
for i in range(len(new_data)):
    print("Property {}: ${:,.2f}".format(i+1, new_predictions[i]))
```

## Testing Output

```
X_test=strat_test_set.drop("MEDV",axis=1)
y_test=strat_test_set["MEDV"].copy()
X_test_prepared = my_pipeline.transform(X_test)
final_predications = model.predict(X_test_prepared)
final_mse = mean_squared_error(y_test,final_predications)
final_rmse=np.sqrt(final_mse)
print(final_predications,list(y_test))
```

```
[19.778 47.32  20.486 24.586 18.632 23.762 21.43  14.55  19.701 17.466
 14.723 19.356 19.927 19.014 23.773 20.423 36.967 25.75  19.664 19.975
  7.893 26.568 10.473 21.392 36.811 42.327 16.772 24.379 20.515 19.644
 19.511 32.073 23.396 22.834 20.534 15.341 20.042 17.821 19.963 27.35
 34.476 19.558 17.954 21.766 19.4   20.259 43.522 21.537 28.703 14.764
 15.036 11.201 14.077 21.636 24.556 20.071  9.718 15.706 21.195 15.571
 21.259 22.217 34.3   31.734 21.165 20.162 22.642 35.169 20.947 31.447
 18.773 20.891 22.092 20.526 14.79  28.132 21.982 19.672 27.094 31.426
 31.459 14.903 21.506 35.866 11.322 24.895 19.319 30.003 12.725 15.196
 18.869 20.742 40.986 42.357 26.418 22.077 10.304 20.076 42.556 14.372
 15.092 20.597 23.717 25.119 42.675 23.183 13.761 45.534 22.787 14.545
 20.767 18.892 11.73  20.676 10.602 22.88  33.584 29.161 20.321 25.407
 35.577 14.408 20.557 35.105 10.656 23.195 24.265 15.214 21.748 44.065
 19.924 11.291 19.868 35.01   8.917 33.683 25.936 18.646 33.202 23.719
 11.291 24.027 20.175  9.479 23.629 41.472 20.556 34.212 36.293  9.625
```

```
final_rmse
```

```
3.140831511801662
```

```
prepared_data[0]
```

```
array([-0.43031222,  1.03300622, -0.76454953, -0.26967994, -1.06821718,
        0.44520174, -0.98367505,  0.88327391, -0.29807595, -0.49603496,
       -1.06848125,  0.43295357, -0.4513261 ])
```

## Conclusion

In this project, we successfully implemented a real estate price prediction model using Linear Regression and supervised learning techniques. By analyzing a dataset containing historical real estate information, we trained a Linear Regression model to predict property prices based on relevant features.

The Linear Regression model demonstrated its effectiveness in capturing the linear relationships between the features and the target variable, allowing us to make accurate price predictions. We evaluated the model's performance using metrics such as mean absolute error (MAE), root mean squared error (RMSE), and R-squared, which provided insights into the model's accuracy and predictive power.

The interpretation of the Linear Regression model allowed us to understand the influence of each feature on the predicted prices. By examining the coefficients, we gained insights into the relative importance of different features in determining real estate prices. This information is valuable for stakeholders in making informed decisions regarding pricing, investments, and market trends.

The implementation of Linear Regression and supervised learning techniques in real estate price prediction demonstrated the potential of machine learning and Python in providing accurate and interpretable models. However, it is important to note that linear relationships may not always capture the full complexity of the real estate market, and other factors may need to be considered.

To further enhance the model's performance, one could explore feature engineering techniques, such as creating interaction terms or polynomial features, or consider using more advanced regression algorithms like Lasso or Ridge Regression to address potential multicollinearity or overfitting issues.

Overall, the utilization of Linear Regression and supervised learning techniques in real estate price prediction opens up avenues for informed decision-making, helping buyers, sellers, and investors navigate the dynamic real estate market with greater confidence. The project highlights the power of machine learning and Python in extracting valuable insights from real estate data, contributing to the advancement of data-driven approaches in the real estate industry