

A

PROJECT REPORT ON

“Real Estate Price Prediction Using Machine Learning”

SUBMITTED BY
Singh Abhishek Amar
Roll No – MC22064
Seat No - 2783

UNDER THE GUIDANCE OF
“Prof. Yugandhara Patil”

IN PARTIAL FULFILMENT OF
Award of the Degree of
MASTER OF COMPUTER APPLICATION
(Sem II)

SUBMITTED TO



SAVITRIBAI PHULE PUNE UNIVERSITY

THROUGH



YASHASWI EDUCATION SOCIETY'S
INTERNATIONAL INSTITUTE OF MANAGEMENT SCIENCE
CHINCHWAD, PUNE
ACADEMIC YEAR 2022-2023

DECLARATION

I, **Singh Abhishek Amar**, student of International Institute of Management Science, Chinchwad, Pune, hereby declare that this Mini Project report entitled “**Real Estate Price Prediction Using Machine Learning**”.

Is a bonafide record of work done by me for the partial fulfilment of the requirement for the degree of **Master of Computer Application** (M.C.A) through Savitribai Phule Pune University.

I, hereby, declare that I have adequately referenced the original sources and this is my original work. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

Singh Abhishek Amar
IIMS, Chinchwad, Pune.

ACKNOWLEDGEMENT

I am deeply indebted towards my project guide “**Prof. Yugandhara Patil**” who gave me the opportunity and was instrumental in providing me all the knowledge and insight to do the research. It is their inspiration that has kept me motivated all along my project and the discipline and integrity they had expected from a mini project that made me to learn the real live projects.

I would like to express my earnest gratitude and thanks to **Dr. Shivaji Mundhe** (Director of IIMS) and my project guide **Prof. Yugandhara Patil** for providing me all the knowledge and skills, resources, technical support, guidance as required to achieve this Endeavour.

I thank my all-faculty members and friends for their support and blessings. The report is the result of contribution of numerous people to mention individually.

I also thank all respondent who have given their value time, views and authentic information for this mini project. I thank each and everybody who has contributed directly or indirectly to the successful completion of this project.

(Signature)

Singh Abhishek Amar
IIMS, Chinchwad, Pune.



Yashaswi Education Society's

Reg No. Maha. : 417/2007/Pune

INTERNATIONAL INSTITUTE OF MANAGEMENT SCIENCE

An ISO 9001 Certified Institute

(Approved by AICTE Ministry of HRD Govt. of India, Recognised by Govt. of Maharashtra
and Affiliated to Savitribai Phule Pune University)

Campus. : IIMS Bldg, S. No. 169/1/A, Opp. Elpro International, Chinchwad, Pune - 411033. Ph.: (020) 27353730/32/33/34, Fax : (020) 27354731
Website. : www.iims.ac.in E-mail : info@iims.ac.in

CERTIFICATE

*This is to Certify that **Abhishek Amar Singh** is a Bonafide student of International Institute of Management Science, Chinchwad, Pune, worked on Online Blogging System and has successfully completed project work in partial fulfillment for award of degree **Master of Computer Application (MCA) Sem III** of Savitribai Phule Pune University.*

This report is the record of Student's own efforts under our supervision and guidelines.

Date: 21.12.2023

Internal Guide

HOD-MCA

*Dr. Shivaji Mundhe Director,
IIMS, Chinchwad*

Internal Examiner

External Examiner

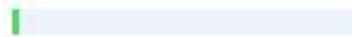
PLAGIARISM REPORT



Plagiarism Checker X - Report

Originality Assessment

2%



Overall Similarity

Date: Dec 19, 2023

Matches: 100 / 4452 words

Sources: 3

Remarks: Low similarity detected, check with your supervisor if changes are required.

Verify Report:

Scan this QR Code



TABLE OF CONTENTS

	Page No.
1. INTRODUCTION	
1.1. Project description	1
2. SYSTEM DETAILS	
2.1. Proposed System	2
2.2. Tools and Technologies used	2
2.3. System requirements	3
3. SOFTWARE REQUIREMENT SPECIFICATION	
3.1. Functional requirements	3
3.2. Non-Functional requirements	3
4. SYSTEM DESIGN	
4.1. System Architecture	4
5. DETAILED DESIGN	
5.1. System development methodology	5
5.2. Approach	6
5.3. Product Requirements Platform independency	8
6. IMPLEMENTATION	
6.1. Programming and Naming convention	8-11
6.2. Snippet code	12-20
6.3. Screenshots	21-31
7. SOFTWARE TESTING	32
8. CONCLUSION	33
9. FUTURE ENHANCEMENT	33
10. BIBLIOGRAPHY	33

1. INTRODUCTION

- Welcome to our **Real Estate Price Prediction Using Machine Learning**! We are excited to bring you a seamless and convenient way to Price Prediction Using ML. The purpose of **Real Estate Price Prediction Using Machine Learning** is to automate the existing manual system by the help of computerized and full-fledged computer software, fulfilling their requirements, so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same.
- This Project Provides us an overview on how to Real Estate Price Prediction Using Machine Learning various ML models with the help of different python libraries.
- This proposed model considers as the most accurate model used for calculating the house price and provides a most accurate prediction. This project consists of what and how the Real Estate Price Prediction model works with the assistance of machine learning technique using scikit-learn and which datasets we will be using in our proposed model.
- By using real world data entities, we are going to predict the price of the house in that area. For better results we require data pre-processing units to improve the efficiency of the model. for this project we are using supervised learning, which is a part of machine learning. We have to go through different attributes of the dataset.
- By using real world data entities, we are going to predict the price of the house in that area. For better results we require data pre-processing units to improve the efficiency of the model. for this project we are using supervised learning, which is a part of machine learning. We have to go through different attributes of the dataset.
- Keywords: - Python, Machine learning, scikit-learn, python libraries, data pre-processing, Linear Regression, Decision Tree, RandomForestRegressor algorithm, Supervised learning.

2. SYSTEM DETAIL

2.1 Proposed System

One of the basic requirements of livelihood in the recent world is to buy a house of your own. The price of the house may depend on various factors. Real estate agents and many who are involved in selling the house want a price tag on the house which would be the real worth of buying the Real Estate Or House.

Thousands of houses are sold every day. There are some questions every buyer asks himself like: What is the actual price that this house deserves? Am I paying a fair price? In this paper, a machine learning model is proposed to predict a house price based on data related to the house and This will facilitate the reproducibility of our work. In this study, Python programming language with a number of Python packages will be used.

2.2 Tools and Technologies used

Tools

- | | |
|--------------------|---|
| ➤ Jupyter Notebook | : An interactive python environment for data analysis and scientific computing. |
|--------------------|---|

Technologies

- | | |
|--------------|---|
| ➤ Numpy | : Library for numerical computing. |
| ➤ Matplotlib | : Library which provides collection of functions for creating variety of static, animated and interactive visualizations in python. |
| ➤ Pandas | : A data manipulation library for data analysis |
| ➤ Sklearn | : A machine learning library for python |

2.3 System requirements

Hardware Requirements (Optimum)

- Processor : intel i3 and above
- RAM : 4 GB and above
- Operating System : Windows 7 and above
- Hard Disk : 512 GB and above

Software requirements

- Tools : MS Excel, Python.
- Platform : Vs Code
- IDE : Jupyter

3. SOFTWARE REQUIREMENT SPECIFICATION

3.1 Functional requirements

Functional Requirement defines a function of a software system and how the system must behave when presented with specific inputs or conditions. These may include calculations, data manipulation and processing and other specific functionality. Following are the functional requirements on the system: The whole process can be handled at minimal human interaction with android and web both. The application automatically receives the captured data from server.

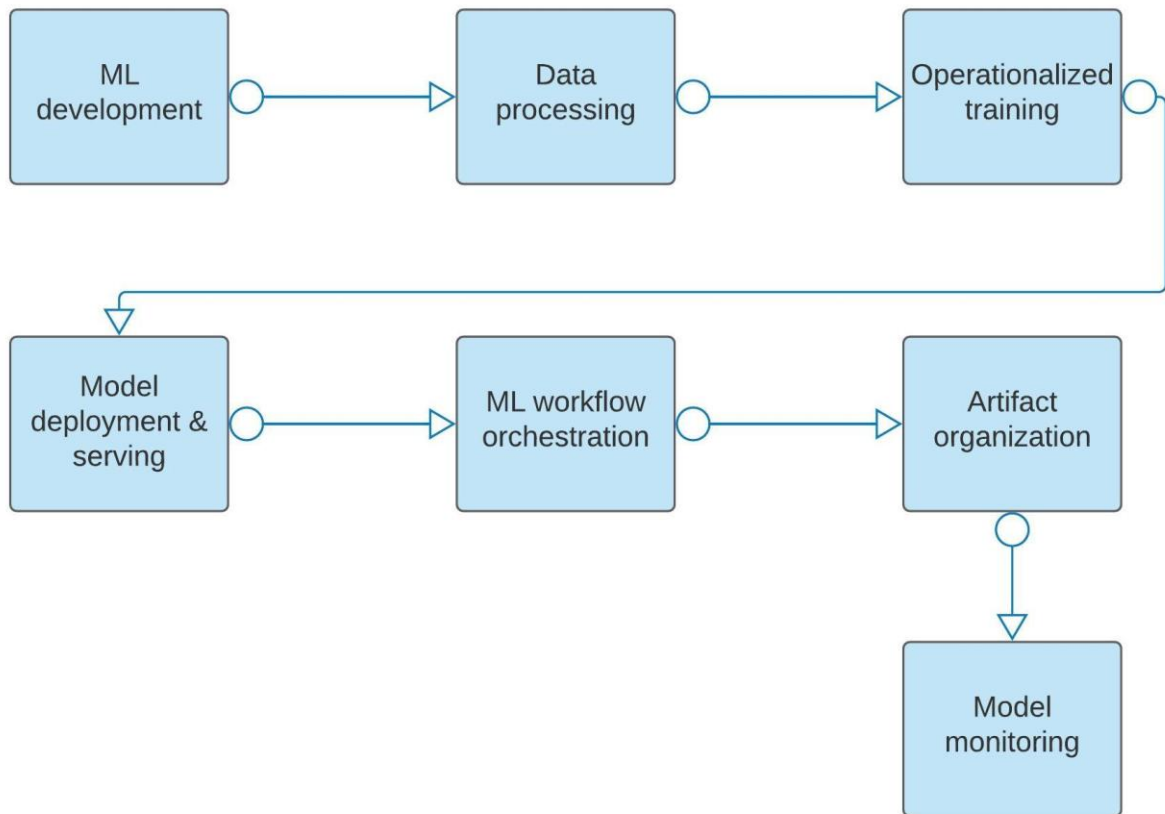
3.2 Non-Functional Requirements

Non-functional requirements are the requirements which are not directly concerned with the specific function delivered by the system. They specify the criteria that can be used to judge the operation of a system rather than specific behaviors. They may relate to emergent system properties such as reliability, response time and store occupancy. Non-functional requirements arise through the user needs, because of budget constraints, organizational policies, the need for interoperability with other software and hardware systems or because of external factors such as

- ✓ Performance Requirements
- ✓ Design Requirements
- ✓ Security Constraints
- ✓ Basic Operational Requirements

4. SYSTEM DESIGN

4.1 System Architecture



5. DETAILED DESIGN

Design is a meaningful engineering representation of something that is to be built. It is the most crucial phase in the developments of a system. Software design is a process through which the requirements are translated into a representation of software. Design is a place where design is fostered in software Engineering. Based on the user requirements and the detailed analysis of the existing system, the new system must be designed.

This is the phase of system designing. Design is the perfect way to accurately translate a customer's requirement in the finished software product. Design creates a representation or model, provides details about software data structure, architecture, interfaces and components that are necessary to implement

a system. The logical system design arrived at as a result of systems analysis is converted into physical system design.

5.1 System Development Methodology

System development method is a process through which a product will get completed or a product gets rid from any problem. Software development process is described as a number of phases, procedures and steps that gives the complete software. It follows series of steps which is used for product progress. The development method followed in this project is waterfall model.

The waterfall model is a successive programming improvement process, in which advance is seen as streaming relentlessly downwards (like a waterfall) through the periods of Requirement start, Analysis, Design, Implementation, Testing and upkeep.

5.1.1 Prerequisite Analysis

This stage is worried about gathering of necessity of the framework. This procedure includes producing record and necessity survey. Framework Design: Keeping the prerequisites at the top of the priority list the framework details are made an interpretation of into a product representation. In this stage the fashioner underlines on calculation, information structure, programming design and so on.

5.1.2 Coding

In this stage developer begins his coding with a specific end goal to give a full portray of item. At the end of the day framework particulars are just changed over into machine coherent register code.

5.1.3 Usage

The execution stage includes the genuine coding or programming of the product. The yield of this stage is regularly the library, executables, client manuals and extra programming documentation.

5.1.4 Testing

In this stage all projects (models) are coordinated and tried to guarantee that the complete framework meets the product prerequisites. The testing is worried with check and approval.

5.1.5 Support

The upkeep stage is the longest stage in which the product is upgraded to satisfy the changing client need, adjust to suit change in the outside environment, right mistakes and oversights beforehand undetected in the testing stage, improve the proficiency of the product.

5.2 Approach

The below figure explains the approach we have taken into building the predictive model using machine learning algorithms.

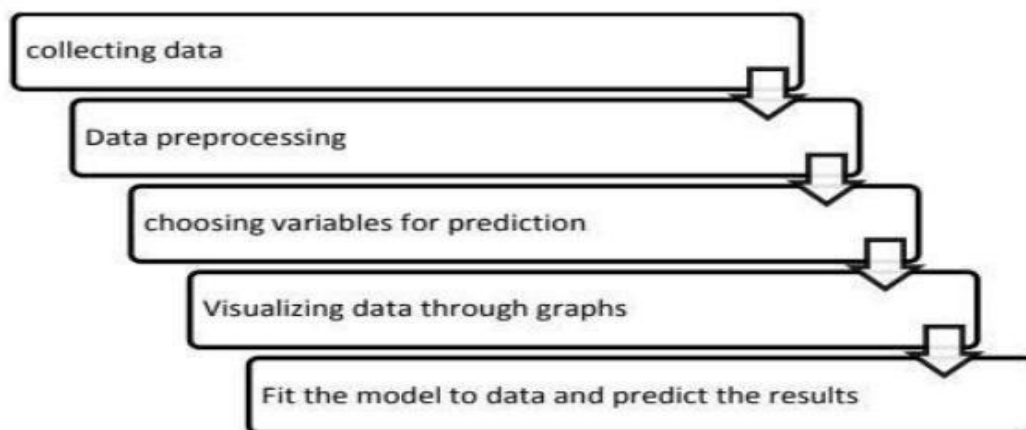


Fig 1: Process of Predicting Real Estate Price

5.2.1 Data Collection

Data collection is the process of gathering and measuring information from countless different sources. In order to use the data, we collect to develop practical machine learning solutions. Collecting data allows you to capture a record of past events so that we can use data analysis to find Predict Cost. From those Predict Cost, you build predictive models using machine learning algorithms that look for trends and predict future changes.

The Real Estate Price Prediction Using Machine Learning is the principal basis of data for this project. The data was web scrapped from the website and kept in the appropriate format using a python library called beautiful soup.

5.2.2 Data Preprocessing

5.2.2.1 Data cleaning

There are some null values in the dataset in the columns. Due to the presence of these null values, the classification cannot be done accurately. So, we tried to replace the null values in different columns with dummy values.

5.2.2.2 Choosing Required Attributes

This step is the main part where we can eliminate some columns of the dataset that are not useful for the estimation of match winning team. This is estimated using feature importance. The considered attributes have the following feature importance.

5.3 Product Requirements Platform independency

A progressive web app will be developed and deployed so that users with a smartphone or a computer can access the voting site to cast their vote. Ease of use: The progressive web app provides an interface which is easy to use and eliminates the need for the voter to go to a voting booth. Modularity: The complete product is broken up into modules and well-defined interfaces are developed to explore the benefit of flexibility of the product. Robustness: This software is being developed in such a way that the overall performance is optimized, and the user can expect the results within a limited time with utmost relevancy and correctness.

6. IMPLEMENTATION

6.1 Programming and Naming Conventions

6.1.1 Program Name

- ✓ Comparison Analysis using Data Preprocessing and Linear Regression, Decision Tree, RandomForestRegressor
- ✓ Prediction analysis using Linear Regression.

6.1.2 Module Names

- ✓ `import pandas as pd`
- ✓ `import numpy as np`
- ✓ `import seaborn as sns`
- ✓ `import matplotlib.pyplot as plt`
- ✓ `from sklearn.model_selection import train_test_split`
- ✓ `from sklearn.model_selection import StratifiedShuffleSplit`

- ✓ `from pandas.plotting import scatter_matrix`
- ✓ `from sklearn.impute import SimpleImputer`
- ✓ `from sklearn.pipeline import Pipeline`
- ✓ `from sklearn.preprocessing import StandardScaler`
- ✓ `from sklearn.metrics import mean_squared_error`
- ✓ `from sklearn.model_selection import cross_val_score`
- ✓ `from joblib import dump, load`
- ✓ `from sklearn.linear_model import LinearRegression`
- ✓ `from sklearn.tree import DecisionTreeRegressor`
- ✓ `from sklearn.ensemble import RandomForestRegressor`

6.1.3 Function Names

- ✓ `pd.read_csv()`
- ✓ `np.array()`
- ✓ `housing.head()`
- ✓ `housing.info()`
- ✓ `housing.describe()`
- ✓ `housing.hist()`
- ✓ `sns.pairplot()`
- ✓ `np.random.seed()`
- ✓ `np.random.permutation()`
- ✓ `housing.corr()`
- ✓ `housing.plot()`
- ✓ `housing.dropna()`
- ✓ `housing.drop()`
- ✓ `imputer.fit()`
- ✓ `imputer.transform()`

- ✓ `pd.DataFrame()`
- ✓ `model.fit()`
- ✓ `model.predict()`
- ✓ `np.sqrt()`
- ✓ `np.array()`

6.1.4 Variable Mames

- ✓ `housing`
- ✓ `object_cols`
- ✓ `num_cols`
- ✓ `fl_cols`
- ✓ `def`
- ✓ `shuffled`
- ✓ `train_set`
- ✓ `test_set`
- ✓ `split`
- ✓ `corr_matrix`
- ✓ `housing_labels`
- ✓ `X`
- ✓ `housing_tr`
- ✓ `my_pipeline`
- ✓ `housing_num_tr`
- ✓ `model`
- ✓ `some_data`
- ✓ `some_label`
- ✓ `prepared_data`
- ✓ `housing_predictions`
- ✓ `final_predictions`

- ✓ X_test
- ✓ Y_test
- ✓ X_test_prepared
- ✓ final_mse
- ✓ final_rmse
- ✓ features

6.1.5 Comments

- ✓ #Importing Lab.
- ✓ #Reading Data.
- ✓ # We will create some simple plot for visualizing the data.
- ✓ #Data Preprocessing.
- ✓ #Splitting Dataset into Training and Testing.
- ✓ #For Correlations.
- ✓ #Plotting Size Attr.
- ✓ # To take care of missing attributes, you have three options.
- ✓ #For Imputer.
- ✓ #For Creating Pipeline from Sklearn.
- ✓ #Model and Accuracy.
 - Linear Regressor
 - DecisionTree Regressor
 - RandomForest Regresso
- ✓ #Evaluating the Model.
- ✓ #Cross Validaton Technique
- ✓ #Testing The Model * Test Data
- ✓ #Used Model ---

6.2 Snippet Code

Real Estate - Price Prediction Using Machine Learning

#Importing Lib

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

#Reading Data

```
housing = pd.read_csv("data.csv")
```

```
housing.head(9)
```

```
housing.info()
```

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

```
housing.describe()
```

```
housing.isna().sum()
```

#For Plotting Histogram

```
import matplotlib.pyplot as plt
```

```
housing.hist(bins=50, figsize=(20,15))
```

We will create some simple plot for visualizing the data.

```
sns.pairplot(housing)
```

#Data Preprocessing

#we categorize the features depending on their datatype (int, float, object) and then calculate the number of them!!

```
obj = (housing.dtypes == 'object')
```

```
object_cols = list(obj[obj].index)
print("Categorical variables:",len(object_cols))
```

```
int_ = (housing.dtypes == 'int')
num_cols = list(int_[int_].index)
print("Integer variables:",len(num_cols))
```

```
fl = (housing.dtypes == 'float')
fl_cols = list(fl[fl].index)
print("Float variables:",len(fl_cols))
```

Splitting Dataset into Training and Testing

#For T-T

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

```
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
print(f"Rows in train set: {len(train_set)}\nRows in test set: {len(test_set)}\n")
```

```

from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, 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]

strat_test_set['CHAS'].value_counts()
strat_train_set['CHAS'].value_counts()
95/7
376/28
housing = strat_train_set.copy()

```

Looking For Correlations

#For Correlations

```

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

```

#Plotting Size Attr

```

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

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

```

Trying out Attribute Combintions

```

housing["TAXRM"] = housing['TAX']/housing['RM']
housing.head()

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

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

```

```
housing = strat_train_set.drop("MEDV", axis=1)
housing_labels = strat_train_set["MEDV"].copy()
```

Missing Attributes

To take care of missing attributes, 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
```

Note that the original housing dataframe will remain unchanged

```
housing.drop("RM", axis=1).shape # Option 2
```

Note that there is no RM column and also note that the original housing dataframe will remain unchanged.

```
median = housing["RM"].median() # Compute median for Option 3
```

```
housing["RM"].fillna(median) # Option 3
```

Note that the original housing dataframe will remain unchanged

```
housing.shape
```

```
housing.describe() # before we started filling missing attributes
```

```
housing.head()
```

#For Imputer

```
from sklearn.impute import SimpleImputer
```

```
imputer = SimpleImputer(strategy="median")
```

```
imputer.fit(housing)
```

SimpleImputer

```
SimpleImputer(strategy='median')
```

```
imputer.statistics_  
imputer.statistics_.shape  
X = imputer.transform(housing)  
housing_tr = pd.DataFrame(X, columns=housing.columns)  
housing_tr.describe()
```

Scikit-Learn Design

Primarily, three types of objects

1. Estimators - It estimates some parameter based on a dataset. Eg. imputer. It has a fit method and transform method. Fit method - Fits the dataset and calculates internal parameters
2. Transformers - transform method takes input and returns output based on the learnings from fit(). It also has a convenience function called fit_transform() which fits and then transforms.
3. Predictors - LinearRegression model is an example of predictor. fit() and predict() are two common functions. It also gives score() function which will evaluate the predictions.

Feature Scaling

Primarily, two types of feature scaling methods:

1. Min-max scaling (Normalization) $(\text{value} - \text{min}) / (\text{max} - \text{min})$ Sklearn provides a class called MinMaxScaler for this
2. Standardization $(\text{value} - \text{mean}) / \text{std}$ Sklearn provides a class called StandardScaler for this

Creating a Pipeline

```
#For Creating Pipeline from Sklearn
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
my_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    # ..... add as many as you want in your pipeline
    ('std_scaler', StandardScaler()),
])

housing_num_tr = my_pipeline.fit_transform(housing)

housing_num_tr.shape
```

Selecting a desired model for Real Estate Price Prediction Using Machine Learning

#Model and Accuracy As we have to train the model to determine the continuous values, so we will be using these regression models.

Linear Regressor - Linear Regression predicts the final output-dependent value based on the given independent features. Like, here we have to predict SalePrice depending on features like MEDV, RM, TAX Etc

1. DecisionTree Regressor - The decision trees is used to fit a sine curve with addition noisy observation. As a result, it learns local linear regressions approximating the sine curve.
2. RandomForest Regressor - Random Forest is an ensemble technique that uses multiple of decision trees and can be used for both regression and classification tasks.

#Regression is a method for understanding the relationship between independent variables or features and a dependent variable or outcome. Outcomes can then be predicted once the relationship between independent and dependent variables has been estimated.

```

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)

```

RandomForestRegressor
RandomForestRegressor()

```

some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
prepared_data = my_pipeline.transform(some_data)
model.predict(prepared_data)
list(some_labels)

```

Evaluating the Model

```

#Evaluating the Model
from sklearn.metrics import mean_squared_error
housing_predictions = model.predict(housing_num_tr)
mse = mean_squared_error(housing_labels, housing_predictions)
rmse = np.sqrt(mse)

rmse

```

Using better Evaluation Technique - Cross Validation

```

#Cross Validaton Technique
# 1 2 3 4 5 6 7 8 9 10
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)

```



```
rmse_scores
```

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

Saving the Model

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

Testing the Model on test data

```
#Testing The Model * 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_predictions = model.predict(X_test_prepared)  
final_mse = mean_squared_error(Y_test, final_predictions)  
final_rmse = np.sqrt(final_mse)  
# print(final_predictions, list(Y_test))  
  
final_rmse  
  
prepared_data[0]
```

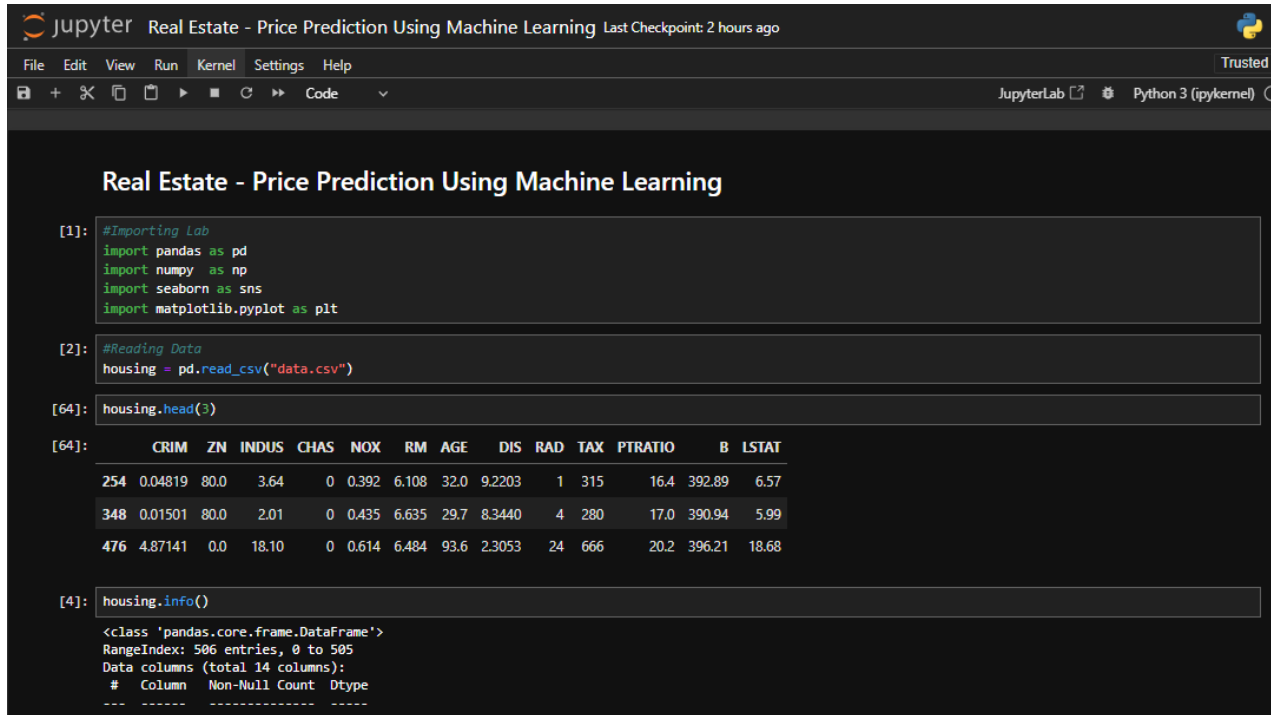
Using the Model

```
#Used Model ---
from joblib import dump
import numpy as np
model = load('Real Estate.joblib')
features = np.array([[ -0.43942006, 11.12628155, -1.12165014, -0.27288841, -
1.42262747,
    -0.23979304, -1.31238772, 2.61111401, -1.0016859 , -0.5778192 ,
    -0.97491834, 0.41164221, -0.86091034],
    [-0.44352175, 3.12628155, -1.35893781, -0.27288841, -1.0542567 ,
    0.5009123 , -1.3938808 , 2.19312325, -0.65766683, -0.78557904,
    -0.69277865, 0.39131918, -0.94116739],
    [ 0.15682292, -0.4898311 , 0.98336806, -0.27288841, 0.47919371,
    0.28867984, 0.87020968, -0.68730678, 1.63579367, 1.50571521,
    0.81196637, 0.44624347, 0.81480158],
    [-0.42292925, -0.4898311 , -0.57719868, -0.27288841, -0.5573845 ,
    0.13688444, -0.52225911, 0.37882487, -0.5429938 , -0.74402708,
    0.52982668, 0.45343469, -0.81939807],
    [-5.40786253, -2.4898311 , -0.57719868, -0.27288841, -0.5573845 ,
    0.04693161, -1.42222622, 5.79643404, -0.5429938 , -0.74402708,
    0.52982668, 0.45343469, -0.91902752]])

model.predict(features)
```

6.2 Screenshots

#Reading data from data.csv



The screenshot shows a JupyterLab environment with the title "Real Estate - Price Prediction Using Machine Learning". The interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar with icons for file operations and code execution. The Python 3 (ipykernel) environment is selected.

The code cells are as follows:

```
[1]: #Importing Lab
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

[2]: #Reading Data
housing = pd.read_csv("data.csv")

[64]: housing.head(3)

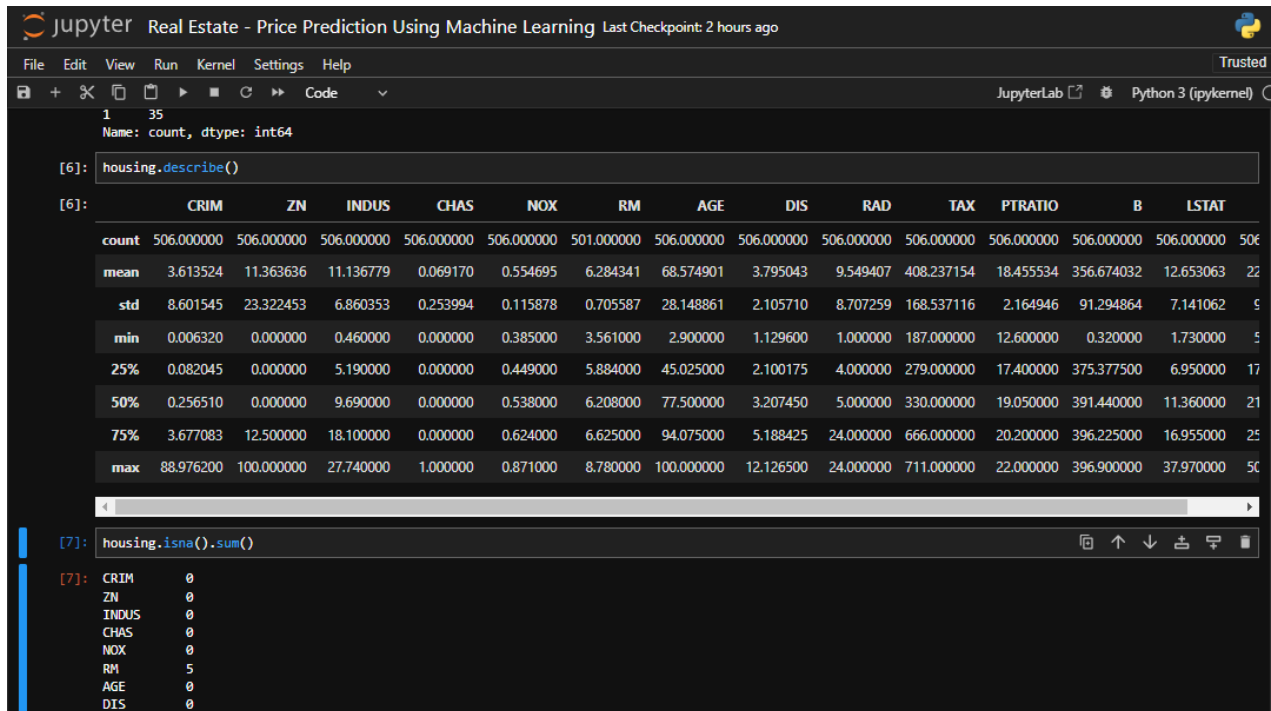
[64]:
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
254	0.04819	80.0	3.64	0	0.392	6.108	32.0	9.2203	1	315	16.4	392.89	6.57
348	0.01501	80.0	2.01	0	0.435	6.635	29.7	8.3440	4	280	17.0	390.94	5.99
476	4.87141	0.0	18.10	0	0.614	6.484	93.6	2.3053	24	666	20.2	396.21	18.68

```
[4]: 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    float64
4  NOX         506 non-null    float64
5  RM          506 non-null    float64
6  AGE         506 non-null    float64
7  DIS         506 non-null    float64
8  RAD         506 non-null    float64
9  TAX         506 non-null    float64
10 PTRATIO    506 non-null    float64
11 B          506 non-null    float64
12 LSTAT      506 non-null    float64
dtypes: float64(14)
memory usage: 30.1 KB
```

#Description Of Real Estate Attributes



The screenshot shows the same JupyterLab environment. The code cells are as follows:

```
[6]: housing.describe()

[6]:
```

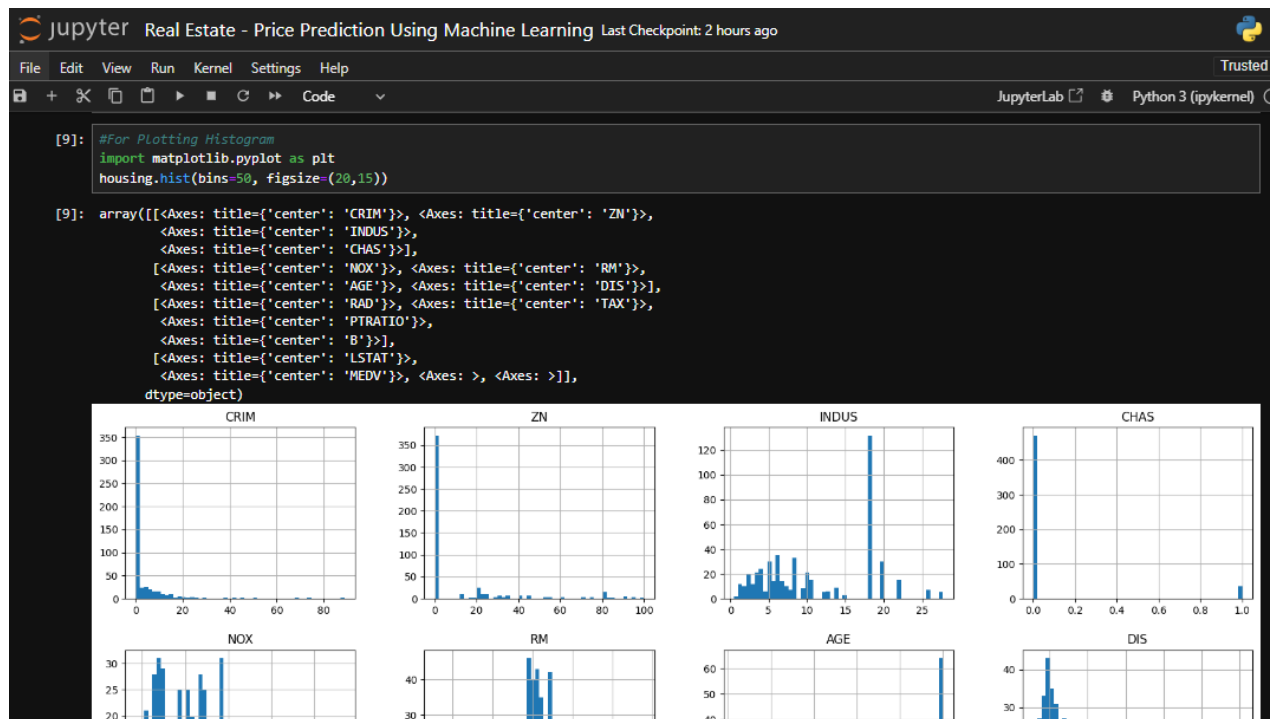
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
count	506.000000	506.000000	506.000000	506.000000	506.000000	501.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284341	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.653063
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.705587	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.141062
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
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.884000	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.950000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208000	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.360000
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
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

```
[7]: housing.isna().sum()

[7]:
```

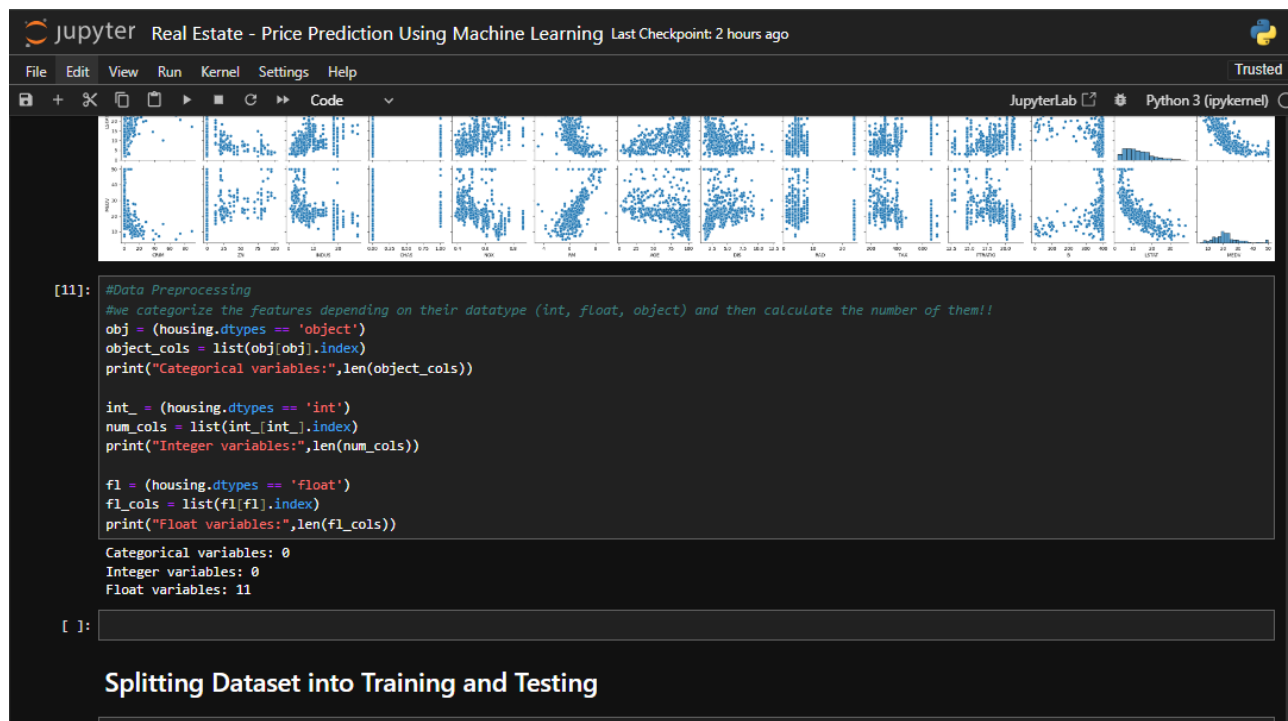
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
CRIM	0							
ZN	0							
INDUS	0							
CHAS	0							
NOX	0							
RM	5							
AGE	0							
DIS	0							

#For Plotting Histogram For Model



#Data Preprocessing

#we categorize the features depending on their datatype (int, float, object) and then calculate the number of them!!



#Splitting Dataset into Train-Test

```
Jupyter Real Estate - Price Prediction Using Machine Learning Last Checkpoint: 2 hours ago
File Edit View Run Kernel Settings Help Trusted
+ - X Copy Paste Run Cell Code

Splitting Dataset into Training and Testing

[12]: #For T-T
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]

[13]: train_set, test_set = split_train_test(housing, 0.2)

[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 60 408 255 392 312 234 460 324 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]
```

#. **StratifiedShuffleSplit** It is used for splitting a dataset into train and test sets while maintaining the distribution of the target variable across the splits

```
Jupyter Real Estate - Price Prediction Using Machine Learning Last Checkpoint: 2 hours ago
File Edit View Run Kernel Settings Help Trusted
+ - X Copy Paste Run Cell Code

[16]: from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, 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]

[17]: strat_test_set['CHAS'].value_counts()

[17]: CHAS
0    95
1     7
Name: count, dtype: int64

[18]: strat_train_set['CHAS'].value_counts()

[18]: CHAS
0    376
1     28
Name: count, dtype: int64

[19]: 95/7

[19]: 13.571428571428571

[20]: 376/28

[20]: 13.428571428571429

[21]: housing = strat_train_set.copy()

[ ]:
```

#Correlations – The `corr()` method finds the correlation of each column in a DataFrame.

```
Jupyter Real Estate - Price Prediction Using Machine Learning Last Checkpoint: 2 hours ago
File Edit View Run Kernel Settings Help Trusted
+ - * / < > Code

Looking For Correlations

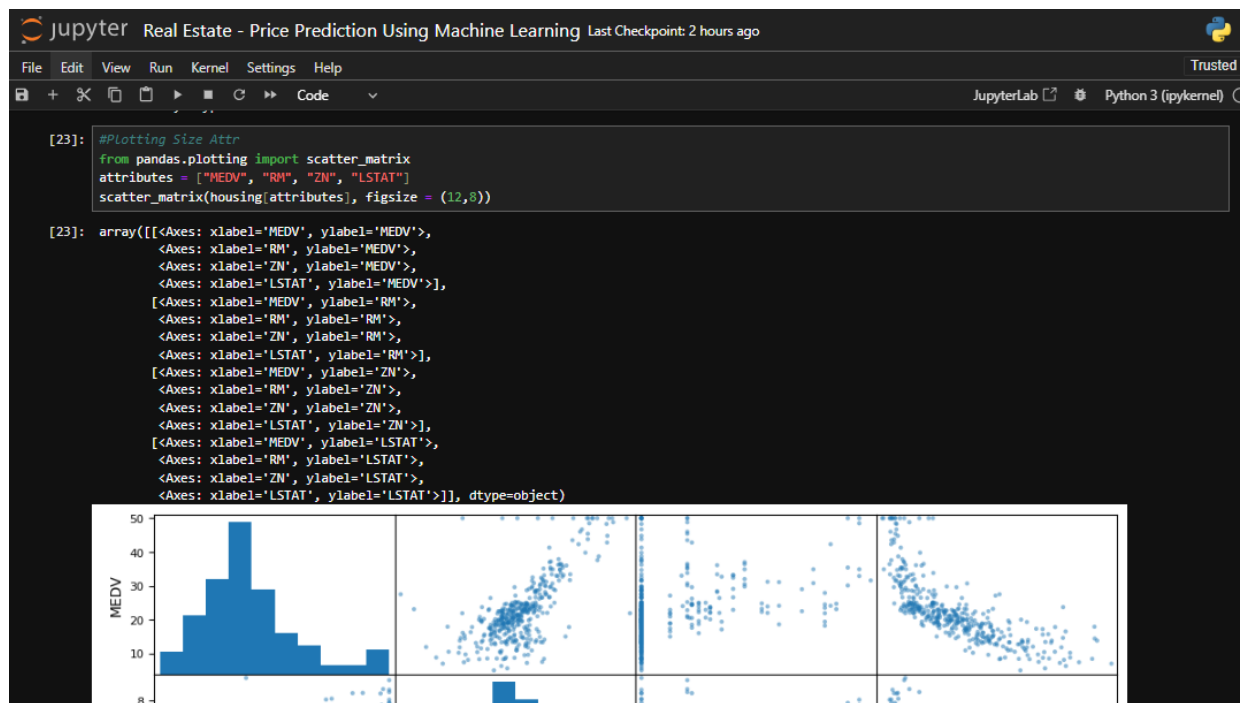
[22]: #For Correlations
      corr_matrix = housing.corr()
      corr_matrix['MEDV'].sort_values(ascending=False)

[22]: MEDV    1.000000
      RM     0.680857
      B      0.361761
      ZN     0.339741
      DIS    0.240451
      CHAS    0.205066
      AGE   -0.364596
      RAD   -0.374693
      CRIM   -0.393715
      NOX    -0.422873
      TAX   -0.456657
      INDUS -0.473516
      PTRATIO -0.493534
      LSTAT  -0.740494
      Name: MEDV, dtype: float64

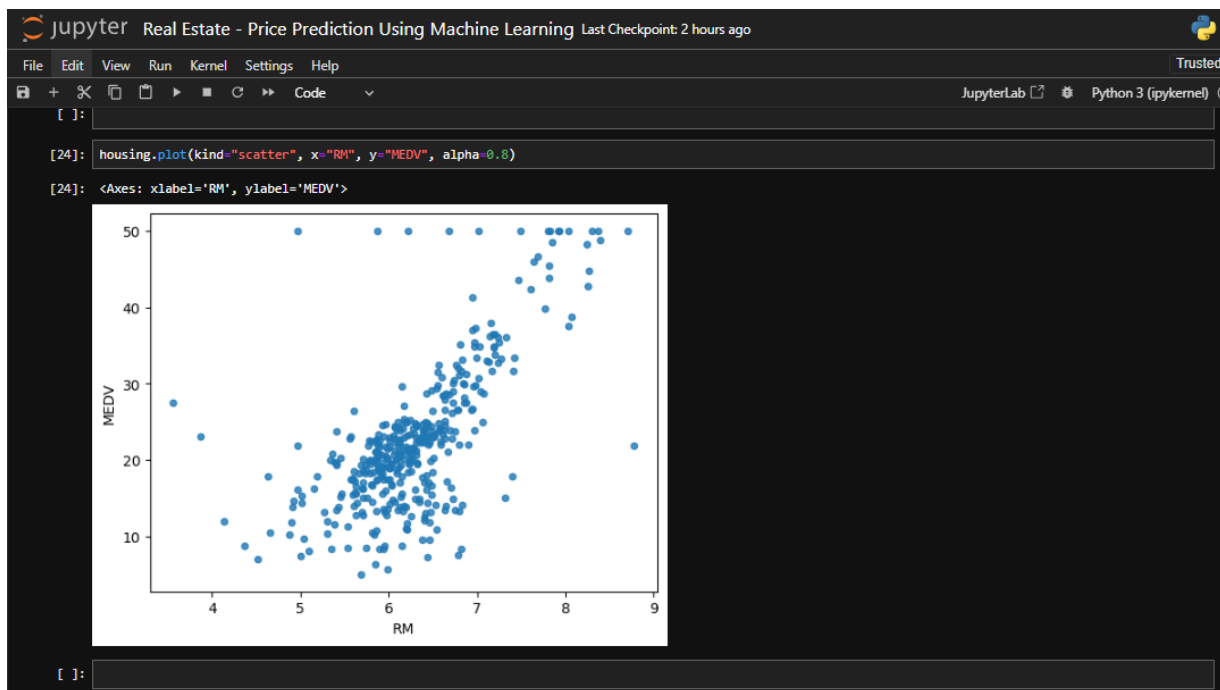
[23]: #Plotting Size Attr
      from pandas.plotting import scatter_matrix
      attributes = ["MEDV", "RM", "ZN", "LSTAT"]
      scatter_matrix(housing[attributes], figsize = (12,8))

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

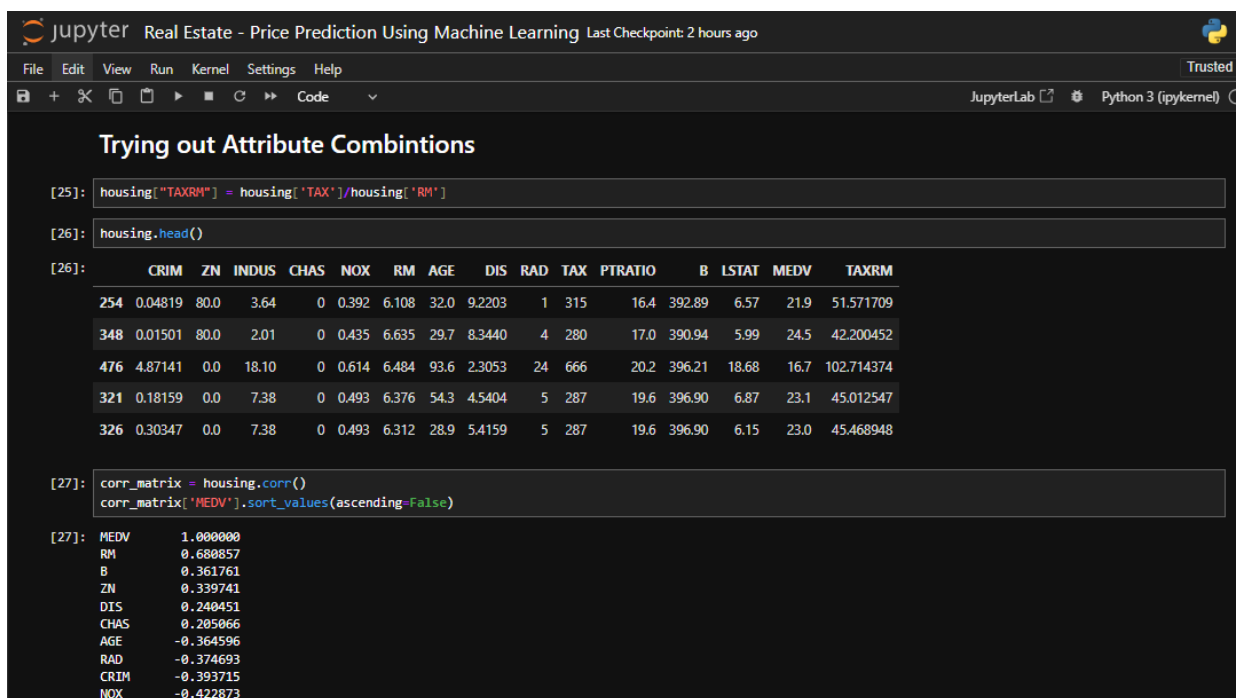
The `scatter_matrix` function is part of the pandas plotting tools and is used to create a matrix of scatterplots.



#Pandas DataFrames, you can use the **scatter** method directly on the DataFrame for quick plotting.



Trying out Attribute Combintions



To take care of missing attributes, 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)

```
jupyter Real Estate - Price Prediction Using Machine Learning Last Checkpoint: 2 hours ago
File Edit View Run Kernel Settings Help Trusted
+ - X Copy Paste Run Cell Code Python 3 (ipykernel)

Missing Attributes

[30]: # To take care of missing attributes, 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)

[31]: a = housing.dropna(subset=["RM"]) #Option 1
a.shape
# Note that the original housing dataframe will remain unchanged

[31]: (399, 13)

[32]: housing.drop("RM", axis=1).shape # Option 2
# Note that there is no RM column and also note that the original housing dataframe will remain unchanged

[32]: (404, 12)

[33]: median = housing["RM"].median() # Compute median for Option 3

[34]: housing["RM"].fillna(median) # Option 3
# Note that the original housing dataframe will remain unchanged

[34]: 254    6.108
      348    6.635
      476    6.484
      321    6.376
      326    6.312
      ...
      155    6.152
      423    6.103
      98     7.820
```

Hosing Describe and Count Attributes

```
jupyter Real Estate - Price Prediction Using Machine Learning Last Checkpoint: 2 hours ago
File Edit View Run Kernel Settings Help Trusted
+ - X Copy Paste Run Cell Code Python 3 (ipykernel)

[36]: housing.describe() # before we started filling missing attributes

[36]:
```

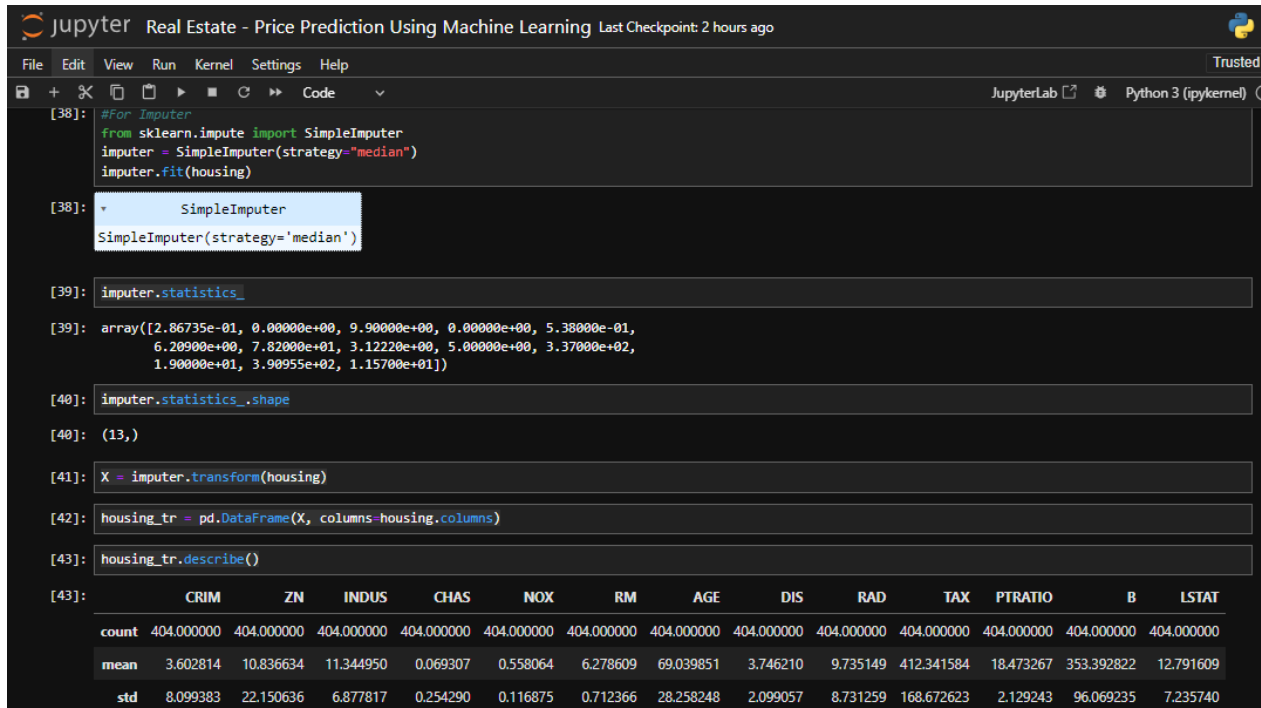
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
count	404.000000	404.000000	404.000000	404.000000	404.000000	399.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000
mean	3.602814	10.836634	11.344950	0.069307	0.558064	6.279481	69.039851	3.746210	9.735149	412.341584	18.473267	353.392822	12.791609
std	8.099383	22.150636	6.877817	0.254290	0.116875	0.716784	28.258248	2.099057	8.731259	168.672623	2.129243	96.069235	7.235740
min	0.006320	0.000000	0.740000	0.000000	0.389000	3.561000	2.900000	1.129600	1.000000	187.000000	13.000000	0.320000	1.730000
25%	0.086962	0.000000	5.190000	0.000000	0.453000	5.876500	44.850000	2.035975	4.000000	284.000000	17.400000	374.617500	6.847500
50%	0.286735	0.000000	9.900000	0.000000	0.538000	6.209000	78.200000	3.122200	5.000000	337.000000	19.000000	390.955000	11.570000
75%	3.731923	12.500000	18.100000	0.000000	0.631000	6.630500	94.100000	5.100400	24.000000	666.000000	20.200000	395.630000	17.102500
max	73.534100	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	36.980000

```
[37]: housing.head()

[37]:
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
254	0.04819	80.0	3.64	0	0.392	6.108	32.0	9.2203	1	315	16.4	392.89	6.57
348	0.01501	80.0	2.01	0	0.435	6.635	29.7	8.3440	4	280	17.0	390.94	5.99
476	4.87141	0.0	18.10	0	0.614	6.484	93.6	2.3053	24	666	20.2	396.21	18.68
321	0.18159	0.0	7.38	0	0.493	6.376	54.3	4.5404	5	287	19.6	396.90	6.87
326	0.30347	0.0	7.38	0	0.493	6.312	28.9	5.4159	5	287	19.6	396.90	6.15

SimpleImputer class provides a simple strategy to handle missing values by replacing them with a constant value or the mean, median, or most frequent value along each column.



```
[38]: #For Imputer
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
imputer.fit(housing)

[38]: SimpleImputer
SimpleImputer(strategy='median')

[39]: imputer.statistics_

[39]: array([2.86735e-01, 0.00000e+00, 9.90000e+00, 0.00000e+00, 5.38000e-01,
        6.20900e+00, 7.82000e+01, 3.12220e+00, 5.00000e+00, 3.37000e+02,
        1.90000e+01, 3.90955e+02, 1.15700e+01])

[40]: imputer.statistics_.shape

[40]: (13,)

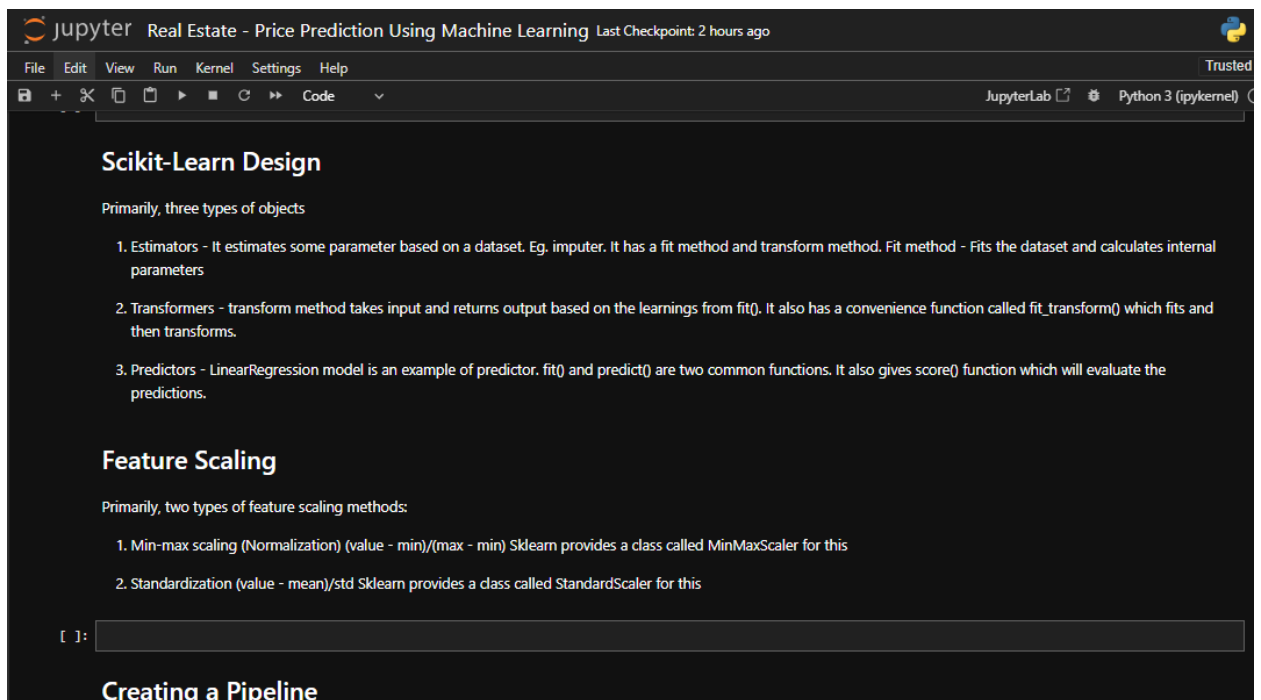
[41]: X = imputer.transform(housing)

[42]: housing_tr = pd.DataFrame(X, columns=housing.columns)

[43]: housing_tr.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
count	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000
mean	3.602814	10.836634	11.344950	0.069307	0.558064	6.278609	69.039851	3.746210	9.735149	412.341584	18.473267	353.392822	12.791609
std	8.099383	22.150636	6.877817	0.254290	0.116875	0.712366	28.258248	2.099057	8.731259	168.672623	2.129243	96.069235	7.235740

Scikit-learn provides simple and efficient tools for data analysis and modeling, including various machine learning algorithms for classification, regression, clustering, dimensionality reduction, and more.



Scikit-Learn Design

Primarily, three types of objects

1. Estimators - It estimates some parameter based on a dataset. Eg. imputer. It has a fit method and transform method. Fit method - Fits the dataset and calculates internal parameters
2. Transformers - transform method takes input and returns output based on the learnings from fit(). It also has a convenience function called fit_transform() which fits and then transforms.
3. Predictors - LinearRegression model is an example of predictor. fit() and predict() are two common functions. It also gives score() function which will evaluate the predictions.

Feature Scaling

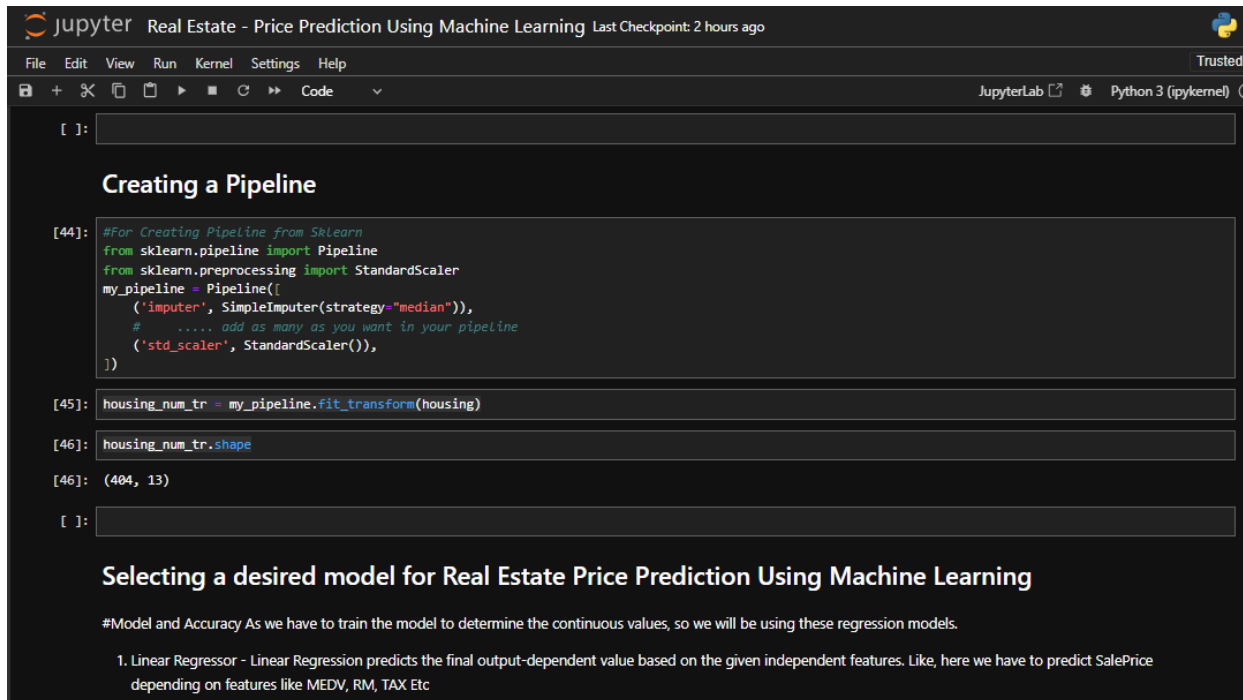
Primarily, two types of feature scaling methods:

1. Min-max scaling (Normalization) $(\text{value} - \text{min}) / (\text{max} - \text{min})$ Sklearn provides a class called MinMaxScaler for this
2. Standardization $(\text{value} - \text{mean}) / \text{std}$ Sklearn provides a class called StandardScaler for this

```
[ ]:
```

Creating a Pipeline

The **Pipeline** class in scikit-learn is a tool to simplify the construction, training, and evaluation of complex machine learning workflows.



The image shows a JupyterLab window titled "Real Estate - Price Prediction Using Machine Learning". The interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar with icons for file operations and execution. The main area displays a code cell with the following content:

```
[44]: #For Creating Pipeline from Sklearn
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
my_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    # ..... add as many as you want in your pipeline
    ('std_scaler', StandardScaler()),
])

[45]: housing_num_tr = my_pipeline.fit_transform(housing)

[46]: housing_num_tr.shape

[46]: (404, 13)

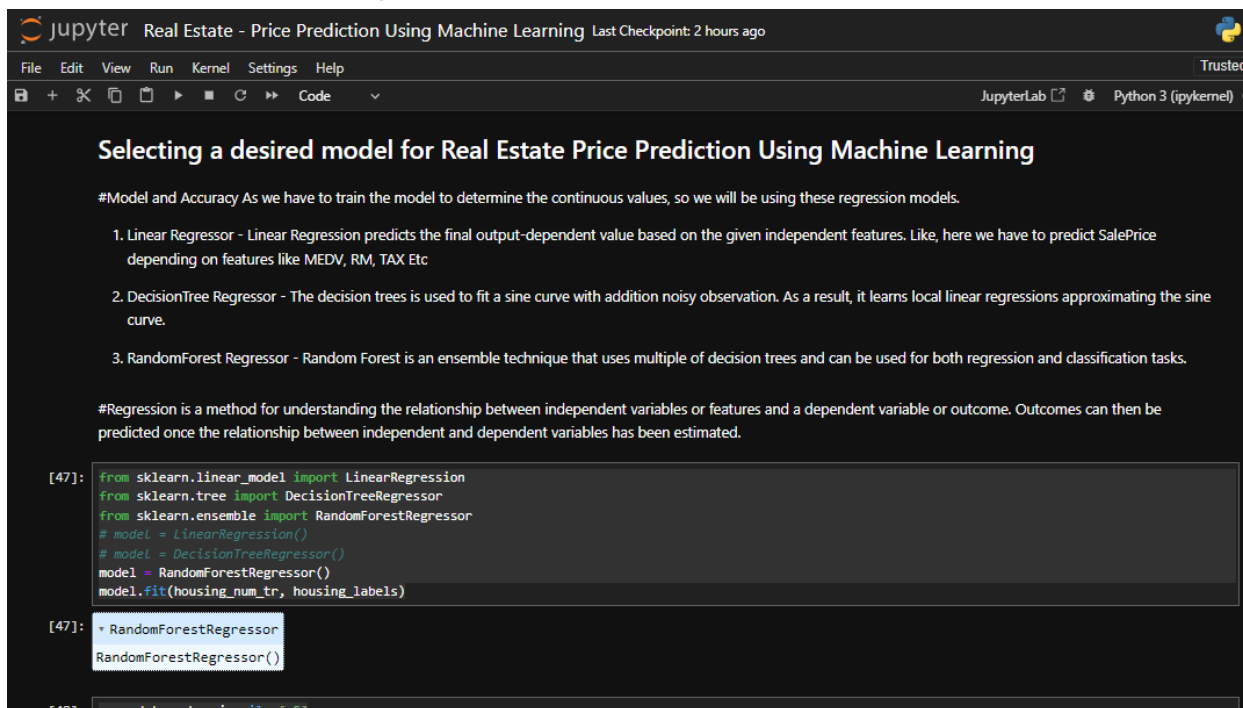
[ ]:
```

Below the code cell, there is a section titled "Selecting a desired model for Real Estate Price Prediction Using Machine Learning". It contains a paragraph: "#Model and Accuracy As we have to train the model to determine the continuous values, so we will be using these regression models." followed by a list:

1. Linear Regressor - Linear Regression predicts the final output-dependent value based on the given independent features. Like, here we have to predict SalePrice depending on features like MEDV, RM, TAX Etc

#Model For Predictions ----

- Liner Regression
- DecisionTress Regression
- RandomForest Regression



The image shows a JupyterLab window titled "Real Estate - Price Prediction Using Machine Learning". The interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar with icons for file operations and execution. The main area displays a code cell with the following content:

```
[47]: 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)

[47]: Random Forest Regressor
RandomForestRegressor()
```

Below the code cell, there is a section titled "Selecting a desired model for Real Estate Price Prediction Using Machine Learning". It contains a paragraph: "#Model and Accuracy As we have to train the model to determine the continuous values, so we will be using these regression models." followed by a list:

1. Linear Regressor - Linear Regression predicts the final output-dependent value based on the given independent features. Like, here we have to predict SalePrice depending on features like MEDV, RM, TAX Etc
2. DecisionTree Regressor - The decision trees is used to fit a sine curve with addition noisy observation. As a result, it learns local linear regressions approximating the sine curve.
3. RandomForest Regressor - Random Forest is an ensemble technique that uses multiple of decision trees and can be used for both regression and classification tasks.

Below the list, there is a paragraph: "#Regression is a method for understanding the relationship between independent variables or features and a dependent variable or outcome. Outcomes can then be predicted once the relationship between independent and dependent variables has been estimated."

#Random Forest is an ensemble learning method that constructs a multitude of decision trees during training and outputs the mean prediction of the individual trees for regression tasks And Predict Price

```
Jupyter Real Estate - Price Prediction Using Machine Learning Last Checkpoint: 2 hours ago
File Edit View Run Kernel Settings Help Trusted
+ - X Copy Paste Run Code Python 3 (ipykernel)

Regression is a machine learning technique for understanding the relationship between independent variables or features and a dependent variable or outcome. Outcomes can either be predicted once the relationship between independent and dependent variables has been estimated.

[47]: 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)

[47]: RandomForestRegressor
      RandomForestRegressor()

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

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

[50]: prepared_data = my_pipeline.transform(some_data)

[51]: model.predict(prepared_data)

[51]: array([22.463, 25.471, 16.331, 23.488, 23.459])

[52]: list(some_labels)

[52]: [21.9, 24.5, 16.7, 23.1, 23.0]

[ ]:
```

Evaluating a Machine learning Model--

```
Jupyter Real Estate - Price Prediction Using Machine Learning Last Checkpoint: 2 hours ago
File Edit View Run Kernel Settings Help Trusted
+ - X Copy Paste Run Code Python 3 (ipykernel)

[51]: array([22.463, 25.471, 16.331, 23.488, 23.459])

[52]: list(some_labels)

[52]: [21.9, 24.5, 16.7, 23.1, 23.0]

[ ]:

Evaluating the Model

[53]: #Evaluating the Model
      from sklearn.metrics import mean_squared_error
      housing_predictions = model.predict(housing_num_tr)
      mse = mean_squared_error(housing_labels, housing_predictions)
      rmse = np.sqrt(mse)

[54]: rmse

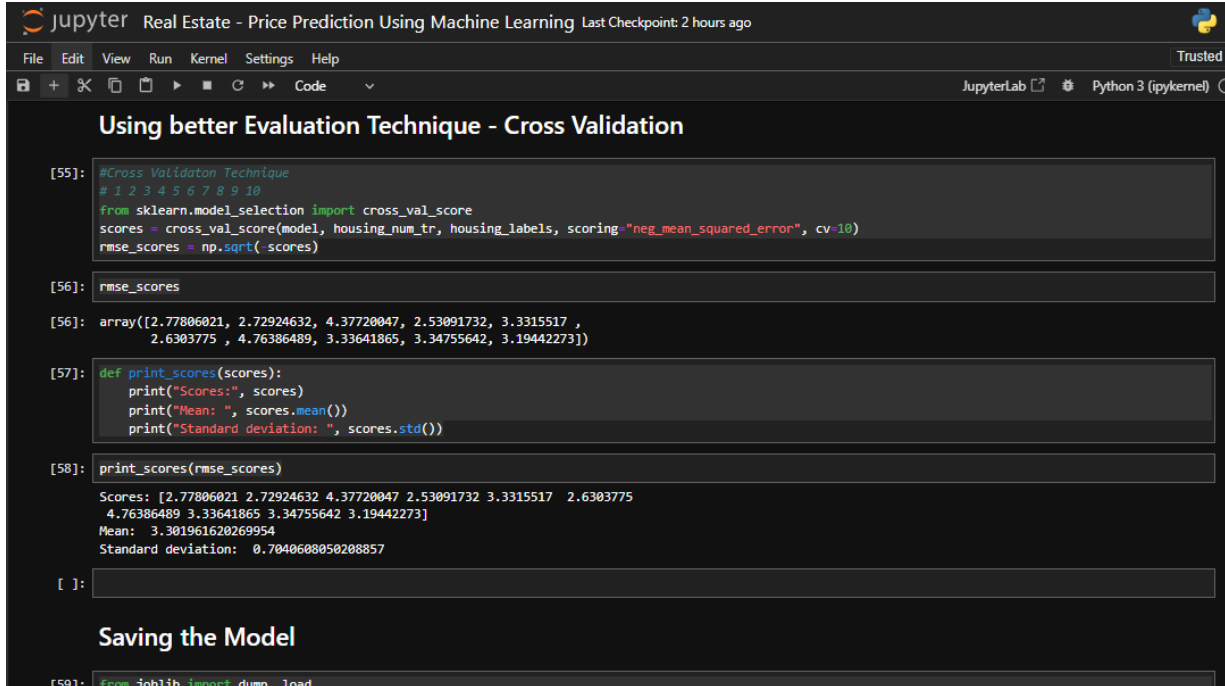
[54]: 1.1634295517057218

[ ]:

Using better Evaluation Technique - Cross Validation

[55]: #Cross Validation Technique
      # 1 2 3 4 5 6 7 8 9 10
      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)
```

Using better Evaluation Technique - Cross Validation is used in machine learning to assess the performance of a model and to reduce the risk of overfitting.



The screenshot shows a JupyterLab window titled "Real Estate - Price Prediction Using Machine Learning". The code in the first cell implements cross-validation using `cross_val_score` with `neg_mean_squared_error` as the scoring function and `cv=10`. The output shows an array of 10 scores. The second cell prints these scores, the mean, and the standard deviation. The third cell shows the RMSE scores.

```
[55]: #Cross Validation Technique
# 1 2 3 4 5 6 7 8 9 10
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)

[56]: rmse_scores

[56]: array([2.77806021, 2.72924632, 4.37720047, 2.53091732, 3.3315517 ,
        2.6303775 , 4.76386489, 3.33641865, 3.34755642, 3.19442273])

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

[58]: print_scores(rmse_scores)

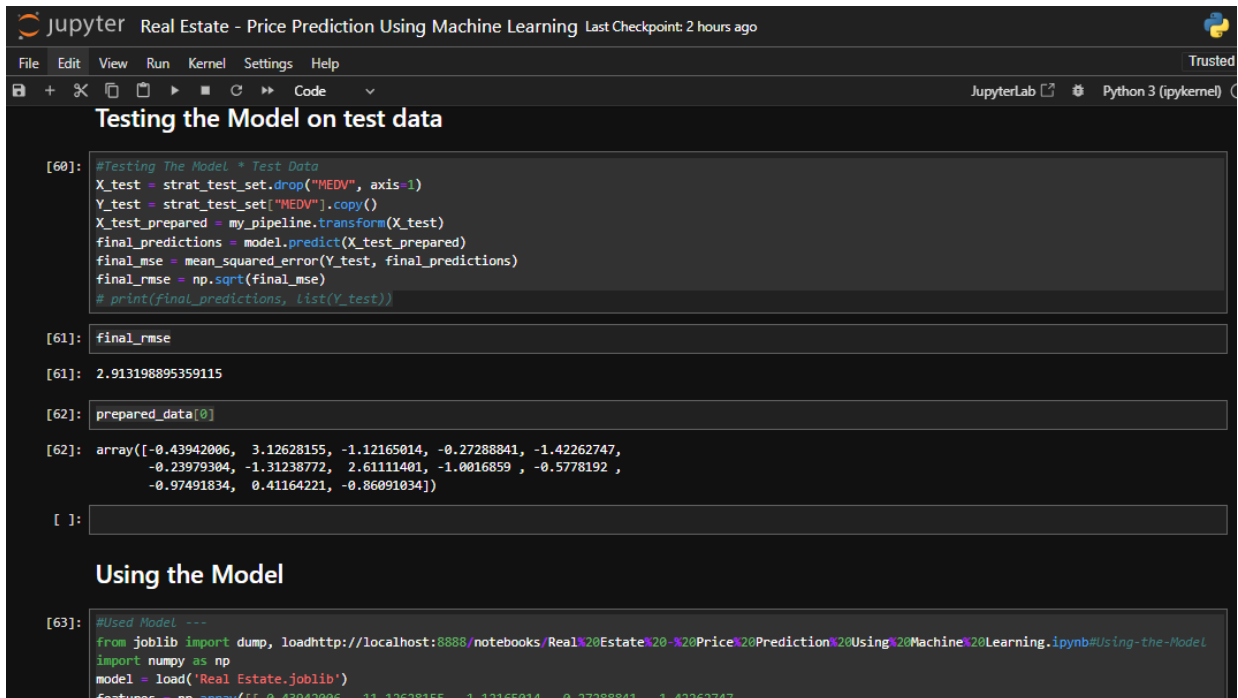
Scores: [2.77806021 2.72924632 4.37720047 2.53091732 3.3315517  2.6303775
 4.76386489 3.33641865 3.34755642 3.19442273]
Mean:  3.381961620269954
Standard deviation:  0.704060805028857

[ ]:

Saving the Model

[59]: from joblib import dump, load
```

Testing The Model on The Test Data and predicts Accurate Price and Cost



The screenshot shows a JupyterLab window titled "Real Estate - Price Prediction Using Machine Learning". The code in the first cell tests the model on test data by dropping the 'MEDV' column from the test set and using the trained pipeline to make predictions. The second cell prints the final RMSE. The third cell prints the first row of the prepared data. The fourth cell prints the first row of the test data. The fifth cell saves the model using `joblib`.

```
[60]: #Testing The Model * 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_predictions = model.predict(X_test_prepared)
final_mse = mean_squared_error(Y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
# print(final_predictions, list(Y_test))

[61]: final_rmse

[61]: 2.913198895359115

[62]: prepared_data[0]

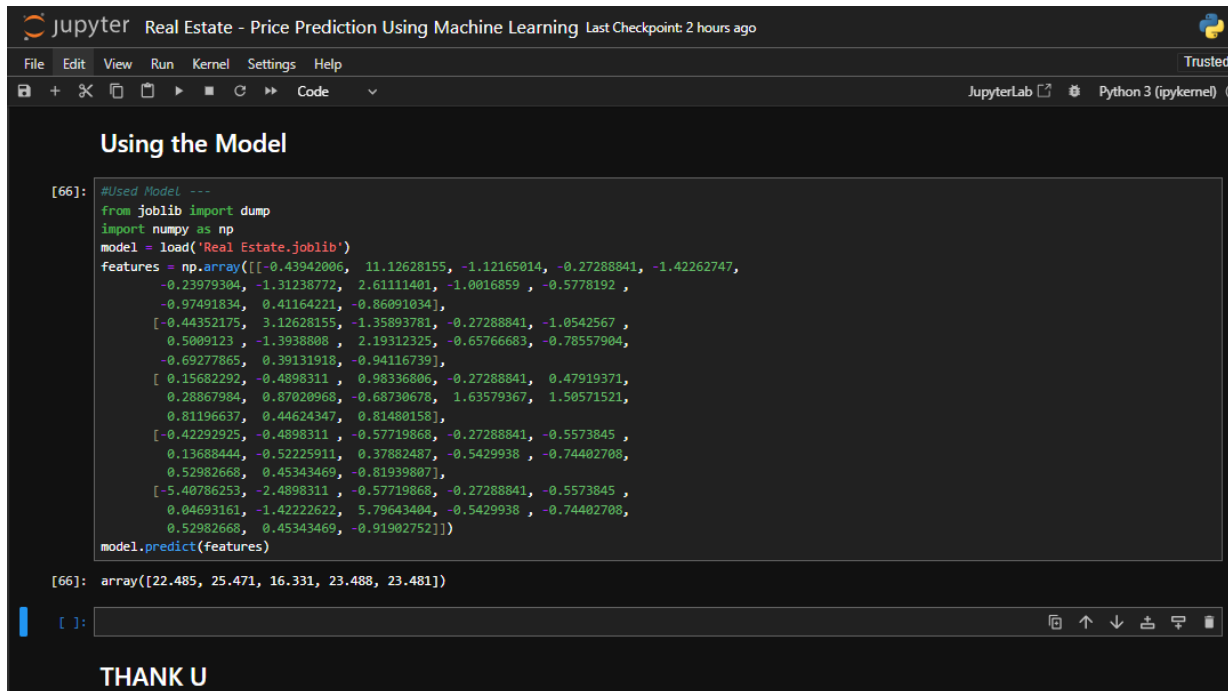
[62]: array([-0.43942006,  3.12628155, -1.12165014, -0.27288841, -1.42262747,
        -0.23979304, -1.31238772,  2.61111401, -1.0016859 , -0.5778192 ,
        -0.97491834,  0.41164221, -0.86091034])

[ ]:

Using the Model

[63]: #Used Model ---
from joblib import dump, load
http://localhost:8888/notebooks/Real%20Estate%20-%20Price%20Prediction%20Using%20Machine%20Learning.ipynb#Using-the-Model
import numpy as np
model = load('Real Estate.joblib')
features = np.array([[ -0.43942006,  3.12628155, -1.12165014, -0.27288841, -1.42262747,
```

Uses Of The Model



The image shows a JupyterLab interface with a dark theme. The title bar reads "jupyter Real Estate - Price Prediction Using Machine Learning Last Checkpoint: 2 hours ago". The menu bar includes File, Edit, View, Run, Kernel, Settings, and Help. The toolbar shows various icons for file operations and execution. The main area displays a code cell with the following content:

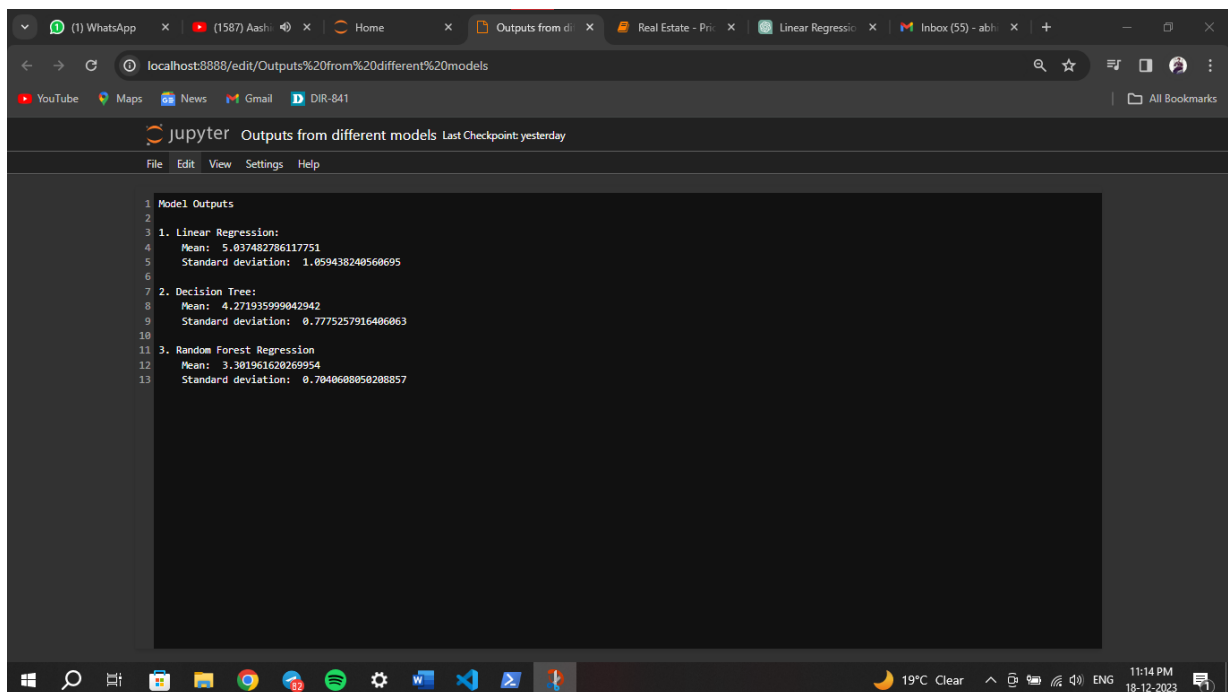
```
[66]: #Used Model ---
from joblib import dump
import numpy as np
model = load('Real Estate.joblib')
features = np.array([[ -0.43942006, 11.12628155, -1.12165014, -0.27288841, -1.42262747,
-0.23979304, -1.31238772, 2.61111401, -1.0016859, -0.5778192,
-0.97491834, 0.41164221, -0.86091034],
[-0.44352175, 3.12628155, -1.35893781, -0.27288841, -1.0542567,
0.5009123, -1.3938808, 2.19312325, -0.65766683, -0.78557904,
-0.69277865, 0.39131918, -0.94116739],
[ 0.15682292, -0.4898311, 0.98336806, -0.27288841, 0.47919371,
0.28867984, 0.87020968, -0.68730678, 1.63579367, 1.50571521,
0.81196637, 0.44624347, 0.81480158],
[-0.42292925, -0.4898311, -0.57719868, -0.27288841, -0.5573845,
0.13688444, -0.52225911, 0.37882487, -0.5429938, -0.74402708,
0.52982668, 0.45343469, -0.81939807],
[-5.40786253, -2.4898311, -0.57719868, -0.27288841, -0.5573845,
0.04693161, -1.4222622, 5.79643404, -0.5429938, -0.74402708,
0.52982668, 0.45343469, -0.91902752]])
model.predict(features)
```

The output of the cell is:

```
[66]: array([22.485, 25.471, 16.331, 23.488, 23.481])
```

At the bottom of the interface, the text "THANK U" is visible.

Output From Different – Different Models (“Price Prediction”)- ML



The image shows a JupyterLab interface with a dark theme. The title bar reads "jupyter Outputs from different models Last Checkpoint: yesterday". The menu bar includes File, Edit, View, Settings, and Help. The main area displays a code cell with the following content:

```
1 Model Outputs
2
3 1. Linear Regression:
4   Mean: 5.037482786117751
5   Standard deviation: 1.059438240560695
6
7 2. Decision Tree:
8   Mean: 4.271935999042942
9   Standard deviation: 0.7775257916406063
10
11 3. Random Forest Regression
12   Mean: 3.301961628269954
13   Standard deviation: 0.7040608050208857
```

7. SOFTWARE TESTING

7.1 UNIT TESTING

- for a particular Testing the accuracy of the calculation of the Real Estate Price Prediction period.
- Testing the accuracy of the Random Forest Regression algorithm in predicting the future values of the indices.

7.2 INTEGRATION TESTING

- Testing the accuracy of the data input and output between the Linear regression, Random Forest Regression algorithm and the Real Estate Price Prediction datasets.
- Testing the accuracy of the user interface that displays the analysis and prediction results.

7.3 SYSTEM TESTING

- Testing the overall accuracy of the analysis and prediction results over the Dataset Last Few Years.
- Testing the user interface, functionality, and performance of the system under different conditions and scenarios.

7.4 BLACK BOX TESTING

- Testing the user interface and functionality of the system without any knowledge of the underlying code and algorithms.
- Testing the accuracy of the analysis and prediction results without any knowledge of the data sources and statistical models used.

7.5 WHITE BOX TESTING

- Testing the accuracy of the linear regression algorithm by examining the source code and performing tests on its individual components.
- Testing the accuracy of the data input and output between the datasets and the DecisionTree regression algorithm by examining the code that processes the data.

8. CONCLUSION

This project provides a strong foundation for real estate price prediction, and further refinement and exploration of features could yield even more accurate predictions. The insights gained from this analysis contribute to a better understanding of the factors influencing real estate prices in the given context.

We also used different python packages like NumPy, pandas, matplotlib etc. For importing the dataset, and also for doing data pre-processing we used pandas. For doing exploratory data analysis we used matplotlib package in python.

This model further helps people understand whether this place is more suited for them based on heatmap correlation. It also helps people looking to sell a house at best time for greater profit.

9. FUTURE ENHANCEMENTS

Future enhancements for a real estate price prediction project can involve improving the model, expanding the dataset, incorporating additional features, and making the prediction system more user-friendly.

- ✓ Fine-Tuning Hyperparameters.
- ✓ Feature Engineering.
- ✓ Interactive Visualization.
- ✓ Cloud Deployment.
- ✓ Model Monitoring and Maintenance.

10. BIBLIOGRAPHY

- *The Hundred-Page Machine Learning Book* by Andriy Burkov
- *Machine Learning for Hackers* by Drew Conway and John Myles White
- <https://chat.openai.com/chat>.
- <https://youtube.com/>
- <https://google.com/>
- Hands-On Machine Learning for Algorithmic Trading” by Stefan Jansen