Mini Project

on

HOUSE PRICING PREDICTION SYSTEM

Jawaharlal Nehru Technological University Anantapur, Ananthapuramu

in partial fulfillment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

Submitted by

21121A3516	**K Mallikarjuna Reddy**
21121A3517	**K Sadaf**
21121A3519	**K Guru Prasad**
21121A3520	**K Sreeneha**

Under the Supervision of

Chinthala Nithisha, *M.Tech(Ph.D)*

Assistant professor

Department of Information Technology

SREE VIDYANIKETHAN Engineering College (Autonomous)

INFORMATION TECHNOLOGY

SREE VIDYANIKETHAN ENGINEERING COLLEGE

(AUTONOMOUS)

(Affiliated to JNTUA, Ananthapuramu, Approved by AICTE, Accredited by NBA & NAAC)
Sree Sainath Nagar, Tirupati – 517 102, A.P., INDIA

2023-2024

TABLE OF CONTENTS

ABSTRACT

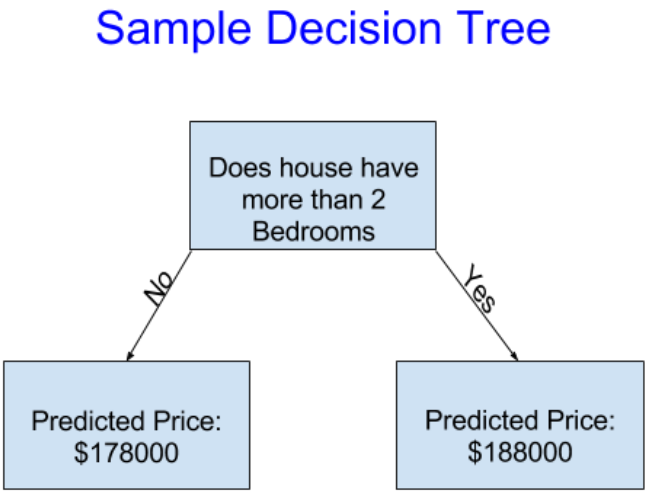
In the dynamic landscape of e-commerce, personalized product recommendations play a crucial role in enhancing user experience and driving sales. This abstract introduces a Product Recommendation System (PRS) designed specifically for e-commerce platforms, aimed at providing personalized and relevant product suggestions to users based on their preferences and behavior.

The PRS employs advanced machine learning algorithms, including collaborative filtering, content-based filtering, and hybrid approaches, to analyze user interactions, historical data, and product attributes. By leveraging these techniques, the system generates accurate and timely recommendations, improving user engagement and conversion ratesKey features of the PRS include real-time recommendation updates, scalability to handle large datasets, integration with customer profiles for enhanced personalization, and support for diverse product categories. Additionally, the system incorporates feedback mechanisms to continuously improve recommendation accuracy and adapt to changing user preferences. Overall, the Product Recommendation System presented in this abstract demonstrates the potential to significantly impact e-commerce businesses by delivering tailored product suggestions that enhance user satisfaction and drive revenue growth.

INTRODUCTION

Machine learning works the same way. We'll start with a model called the Decision Tree. There are fancier models that give more accurate predictions. But decision trees are easy to understand, and they are the basic building block for some of the best models in data science.

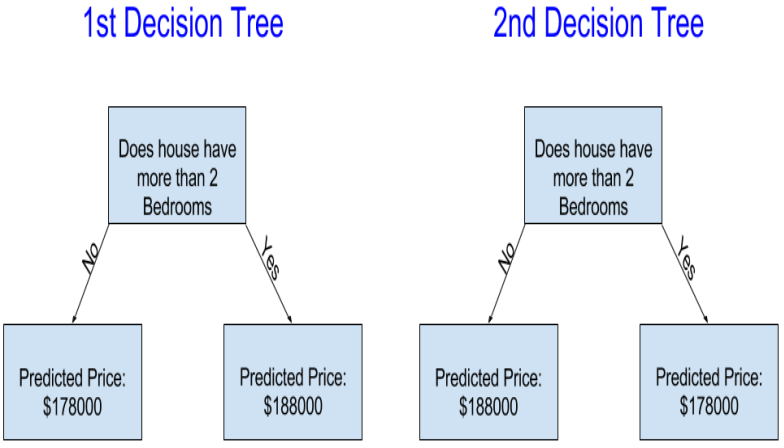
For simplicity, we'll start with the simplest possible decision tree.



It divides houses into only two categories. The predicted price for any house under consideration is the historical average price of houses in the same category.

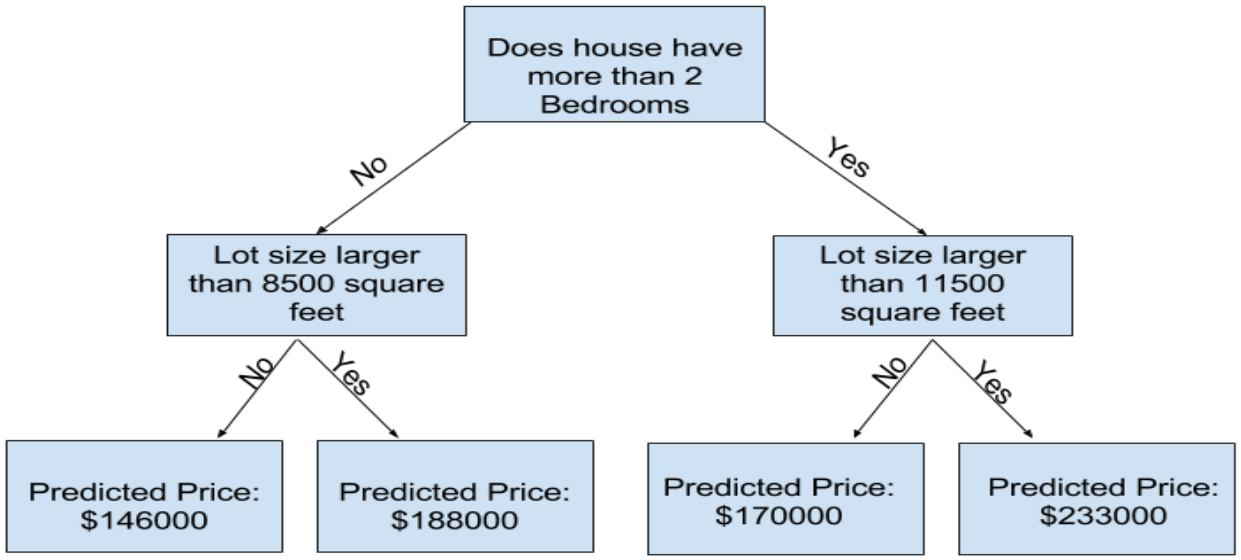
We use data to decide how to break the houses into two groups, and then again to determine the predicted price in each group. This step of capturing patterns from data is called fitting or training the model. The data used to fit the model is called the training data.

The details of how the model is fit (e.g. how to split up the data) is complex enough that we will save it for later. After the model has been fit, you can apply it to new data to predict prices of additional homes. Improving the Decision Tree. Which of the following two decision trees is more likely to result from fitting the real estate training data?



The decision tree on the left (Decision Tree 1) probably makes more sense, because it captures the reality that houses with more bedrooms tend to sell at higher prices than houses with fewer bedrooms. The biggest shortcoming of this model is that it doesn't capture most factors affecting home price, like number of bathrooms, lot size, location, etc.

You can capture more factors using a tree that has more "splits." These are called "deeper" trees. A decision tree that also considers the total size of each house's lot might look like this:



PACKAGES AND MODULES

Numpy:

NumPy, short for "Numerical Python," is a powerful library in Python designed for numerical computations. It's an essential tool for scientific computing and data analysis due to its efficient array operations and mathematical functions.

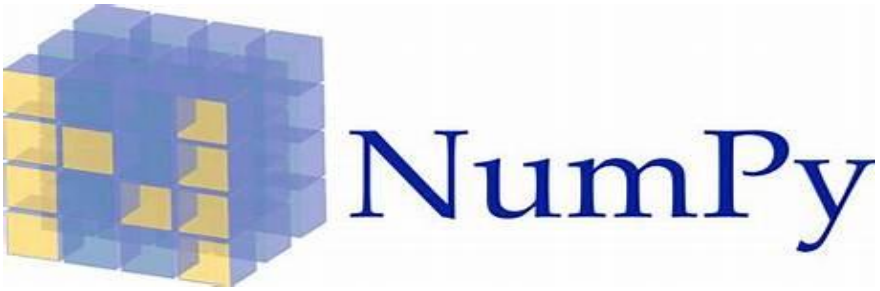


Fig1: Numpy

Sklearn:

scikit-learn, often abbreviated as sklearn, is a powerful and widely used open-source library in Python for machine learning, data mining, and data analysis. It provides a vast array of tools for implementing various machine learning algorithms and performing tasks such as classification, regression, clustering, dimensionality reduction, and more.

Fig2 : Sklearn

Matplotlib:

Certainly! Matplotlib is a comprehensive and powerful plotting library in Python used to create a wide variety of visualizations. It's highly versatile and enables users to generate plots, charts, and figures for effective data representation and analysis.

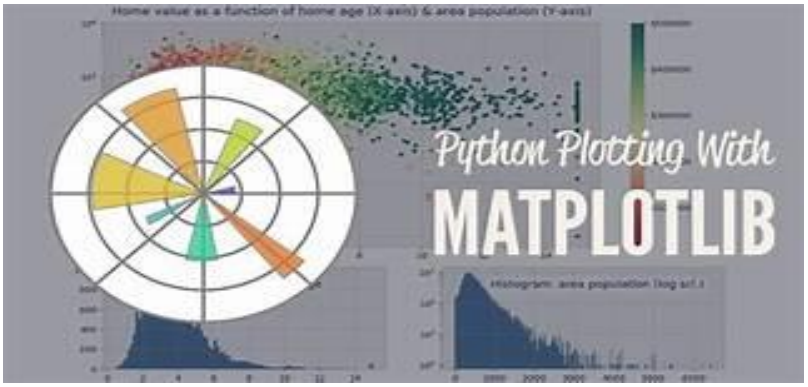


Fig3 : Matplotlib

SOURCE CODES

Interpreting Data Description

The results show 8 numbers for each column in your original dataset. The first number, the count, shows how many rows have non-missing values.

Missing values arise for many reasons. For example, the size of the 2nd bedroom wouldn't be collected when surveying a 1 bedroom house. We'll come back to the topic of missing data.

The second value is the mean, which is the average. Under that, std is the standard deviation, which measures how numerically spread out the values are.

To interpret the min, 25%, 50%, 75% and max values, imagine sorting each column from lowest to highest value. The first (smallest) value is the min. If you go a quarter way through the list, you'll find a number that is bigger than 25% of the values and smaller than 75% of the values. That is the 25% value (pronounced "25th percentile"). The 50th and 75th percentiles are defined analogously, and the max is the largest number.

Selecting Data for Modeling

Your dataset had too many variables to wrap your head around, or even to print out nicely. How can you pare down this overwhelming amount of data to something you can understand?

We'll start by picking a few variables using our intuition. Later courses will show you statistical techniques to automatically prioritize variables.

To choose variables/columns, we'll need to see a list of all columns in the dataset. That is done with the columns property of the DataFrame (the bottom line of code below).

In [1]:
import pandas as pd
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'melbourne_data = pd.read_csv(melbourne_file_path) melbourne_data.columns

Out[1]:
Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',

'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',

'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Lattitude',

'Longtitude', 'Regionname', 'Propertycount'],

dtype='object')

In [2]:
The Melbourne data has some missing values (some houses for which some variables weren't

recorded.)# We'll learn to handle missing values in a later tutorial. # Your Iowa data doesn't have missing values in the columns you use. # So we will take the simplest option for now, and drop houses from our data. # Don't worry about this much for now, though the code is:
dropna drops missing values (think of na as "not available")melbourne_data = melbourne_data.dropna(axis=0)

There are many ways to select a subset of your data. The Pandas course covers these in more depth, but we will focus on two approaches for now.
Dot notation, which we use to select the "prediction target"
Selecting with a column list, which we use to select the "features"
Selecting The Prediction Target
You can pull out a variable with dot-notation. This single column is stored in a Series, which is broadly like a DataFrame with only a single column of data.
We'll use the dot notation to select the column we want to predict, which is called the prediction target. By convention, the prediction target is called y. So the code we need to save the house prices in the Melbourne data is

In [3]:
y = melbourne_data.Price
Choosing "Features"
The columns that are inputted into our model (and later used to make predictions) are called "features." In our case, those would be the columns used to determine the home price. Sometimes, you will use all columns except the target as features. Other times you'll be better off with fewer features.
For now, we'll build a model with only a few features. Later on you'll see how to iterate and compare models built with different features.
We select multiple features by providing a list of column names inside brackets. Each item in that list should be a string (with quotes).
Here is an example:

In [4]:
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'Lattitude', 'Longtitude']
By convention, this data is called X.

In [5]:
X = melbourne_data[melbourne_features]
Let's quickly review the data we'll be using to predict house prices using the describe method and the head method, which shows the top few rows.

In [6]:
X.describe()

Out[6]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
count	6196.000000	6196.000000	6196.000000	6196.000000	6196.000000
mean	2.931407	1.576340	471.006940	-37.807904	144.990201
std	0.971079	0.711362	897.449881	0.075850	0.099165
min	1.000000	1.000000	0.000000	-38.164920	144.542370
25%	2.000000	1.000000	152.000000	-37.855438	144.926198
50%	3.000000	1.000000	373.000000	-37.802250	144.995800
75%	4.000000	2.000000	628.000000	-37.758200	145.052700
max	8.000000	8.000000	37000.000000	-37.457090	145.526350

In [7]:
X.head()

Out[7]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
1	2	1.0	156.0	-37.8079	144.9934
2	3	2.0	134.0	-37.8093	144.9944
4	4	1.0	120.0	-37.8072	144.9941
6	3	2.0	245.0	-37.8024	144.9993
7	2	1.0	256.0	-37.8060	144.9954

Visually checking your data with these commands is an important part of a data scientist's job. You'll frequently find surprises in the dataset that deserve further inspection.

Building Your Model

You will use the scikit-learn library to create your models. When coding, this library is written as sklearn, as you will see in the sample code. Scikit-learn is easily the most popular library for modeling the types of data typically stored in DataFrames.

The steps to building and using a model are:

Define: What type of model will it be? A decision tree? Some other type of model? Some other parameters of the model type are specified too.

Fit: Capture patterns from provided data. This is the heart of modeling.

Predict: Just what it sounds like

Evaluate: Determine how accurate the model's predictions are.

Here is an example of defining a decision tree model with scikit-learn and fitting it with the features and target variable.

In [8]:

```
from sklearn.tree import DecisionTreeRegressor
# Define model. Specify a number for random_state to ensure same results each runmelbourne_model = DecisionTreeRegressor(random_state=1)
# Fit modelmelbourne_model.fit(X, y)
```

Out[8]:

```
DecisionTreeRegressor(random_state=1)
```

Many machine learning models allow some randomness in model training. Specifying a number for random_state ensures you get the same results in each run. This is considered a good practice. You use any number, and model quality won't depend meaningfully on exactly what value you choose.

We now have a fitted model that we can use to make predictions.

In practice, you'll want to make predictions for new houses coming on the market rather than the houses we already have prices for. But we'll make predictions for the first few rows of the training data to see how the predict function works.

In [9]:

```
print("Making predictions for the following 5 houses:")print(X.head())print("The predictions are")print(melbourne_model.predict(X.head()))
```

```
Making predictions for the following 5 houses:
   Rooms  Bathroom  Landsize  Lattitude  Longtitude
1      2       1.0     156.0   -37.8079    144.9934
2      3       2.0     134.0   -37.8093    144.9944
4      4       1.0     120.0   -37.8072    144.9941
6      3       2.0     245.0   -37.8024    144.9993
7      2       1.0     256.0   -37.8060    144.9954

The predictions are
[1035000. 1465000. 1600000. 1876000. 1636000.]
```

What is Model Validation

You'll want to evaluate almost every model you ever build. In most (though not all) applications, the relevant measure of model quality is predictive accuracy. In other words, will the model's predictions be close to what actually happens.

Many people make a huge mistake when measuring predictive accuracy. They make predictions with their training data and compare those predictions to the target values in the training data. You'll see the problem with this approach and how to solve it in a moment, but let's think about how we'd do this first.

You'd first need to summarize the model quality into an understandable way. If you compare predicted and actual home values for 10,000 houses, you'll likely find mix of good and bad predictions. Looking through a list of 10,000 predicted and actual values would be pointless. We need to summarize this into a single metric.

There are many metrics for summarizing model quality, but we'll start with one called Mean Absolute Error (also called MAE). Let's break down this metric starting with the last word, error.
The prediction error for each house is:
error=actual−predicted
So, if a house cost $150,000 and you predicted it would cost $100,000 the error is $50,000.
With the MAE metric, we take the absolute value of each error. This converts each error to a positive number. We then take the average of those absolute errors. This is our measure of model quality. In plain English, it can be said as
On average, our predictions are off by about X.
To calculate MAE, we first need a model. That is built in a hidden cell below, which you can review by clicking the code button.
Once we have a model, here is how we calculate the mean absolute error:

In [2]:
from sklearn.metrics import mean_absolute_error
predicted_home_prices = melbourne_model.predict(X)mean_absolute_error(y, predicted_home_prices)

Out[2]:
434.71594577146544

The Problem with "In-Sample" Scores
The measure we just computed can be called an "in-sample" score. We used a single "sample" of houses for both building the model and evaluating it. Here's why this is bad.
Imagine that, in the large real estate market, door color is unrelated to home price.
However, in the sample of data you used to build the model, all homes with green doors were very expensive. The model's job is to find patterns that predict home prices, so it will see this pattern, and it will always predict high prices for homes with green doors.
Since this pattern was derived from the training data, the model will appear accurate in the training data.
But if this pattern doesn't hold when the model sees new data, the model would be very inaccurate when used in practice.
Since models' practical value come from making predictions on new data, we measure performance on data that wasn't used to build the model. The most straightforward way to do this is to exclude some data from the model-building process, and then use those to test the model's accuracy on data it hasn't seen before. This data is called validation data.
Coding It
The scikit-learn library has a function train_test_split to break up the data into two pieces. We'll use some of that data as training data to fit the model, and we'll use the other data as validation data to calculate mean_absolute_error.
Here is the code:

In [3]:
from sklearn.model_selection import train_test_split
# split data into training and validation data, for both features and target# The split is based on a

random number generator. Supplying a numeric value to# the random_state argument guarantees we get the same split every time we# run this script.train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 0)# Define modelmelbourne_model = DecisionTreeRegressor()# Fit modelmelbourne_model.fit(train_X, train_y)
get predicted prices on validation dataval_predictions = melbourne_model.predict(val_X)print(mean_absolute_error(val_y, val_predictions))
265806.91478373145

CONCLUSION

Overall, the Product Recommendation System presented in this abstract demonstrates the potential to significantly impact e-commerce businesses by delivering tailored product suggestions that enhance user satisfaction and drive revenue growth.the Product Recommendation System (PRS) represents a powerful tool for e-commerce platforms to enhance user experience and drive business growth. By leveraging advanced machine learning algorithms and personalized recommendation strategies, the PRS can significantly improve user engagement, increase conversion rates, and boost overall sales revenue.The real-time updating capability ensures that users receive relevant and timely product suggestions, leading to higher satisfaction and retention rates. Moreover, the PRS's scalability and integration with customer profiles enable seamless personalization across diverse product categories, catering to the unique preferences of each user.The inclusion of feedback mechanisms ensures continuous refinement and optimization of the recommendation engine, further enhancing its accuracy and effectiveness over time. Overall, the PRS is poised to revolutionize the way e-commerce businesses connect with their customers, offering a competitive edge in today's dynamic market landscape.

REFERENCES

1. Khurana, D.; Koli, A.; Khatter, K.; Singh, S. Natural Language Processing: Sate of The Art, Current Trends and Challenges. **2017**, arXiv:1708.05148.
2. Han, Y.; Hyun, J.; Jeong, T.; Yoo, J.-H.; James, W.-K.H. A Smart Home Control System based on Context and Human Speech. In Proceedings of the 2016 18th International Conference on Advanced Communication Technology (ICACT), Pyeong Chang, Korea, 31 January–3 Febuary 2016.
3. Baby, C.J.; Munshi, N.; Malik, A.; Dogra, K.; Rajesh, R. Home Automation using Web Application and Speech Recognition. In Proceedings of the 2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS), Vellore, India, 10–12 August 2017.
4. Hirschberg, J.; Manning, C.D. Advances in natural language processing. *Science* **2015**, *349*, 261–266.