Name	B.vadivukarasi
Reg No	820621106022
Department	ECE
Year	3
College name	Arasu engineering college
Group	IBM Group 4
NM ID	au820621106022

Abstract :
Predicting house prices using machine learning is a supervised learning task, where we train a model to predict a numerical output (sale price) based on a set of input features (such as square footage, number of bedrooms, location, etc.).

Here is a general overview of the steps involved in predicting house prices using machine learning:

Predicting house prices using machine learning

Abstract :
Predicting house prices using machine learning is a supervised learning task, where we train a model to predict a numerical output (sale price) based on a set of input features (such as square footage, number of bedrooms, location, etc.).

Here is a general overview of the steps involved in predicting house prices using machine learning:

Module:

1. **Collect data.** The first step is to collect a dataset of historical house sales, including the features that you want to use to predict price. This dataset can be obtained from a variety of sources, such as real estate websites or government databases.
2. **Prepare the data.** Once you have collected your data, you need to prepare it for machine learning. This may involve cleaning the data, removing outliers, and converting categorical features to numerical features.
3. **Choose a machine learning algorithm.** There are many different machine learning algorithms that can be used for house price prediction. Some popular algorithms include linear regression, random forests, and gradient boosting machines.
4. **Train the model.** Once you have chosen an algorithm, you need to train the model on your prepared data. This involves feeding the algorithm the input features and the corresponding output prices. The model will learn to identify the relationships between the features and price, and use this knowledge to make predictions.
5. **Evaluate the model.** Once the model is trained, you need to evaluate its performance on a held-out test set. This will help you to assess how well the model will generalize to new data.
6. **Use the model to make predictions.** Once you are satisfied with the model's performance, you can use it to make predictions for new houses.

Here are some additional tips for predicting house prices using machine learning:

* Use a large and diverse dataset. The larger and more diverse your dataset, the better the model will be able to learn the relationships between the features and price.
* Handle missing data carefully. Missing data is a common problem in real estate datasets. It is important to handle missing data carefully, so that it does not bias your model's predictions.
* Use feature engineering to create new features. Feature engineering is the process of creating new features from existing features. This can be a powerful way to improve the performance of your model.

* Evaluate multiple machine learning algorithms. There is no one-size-fits-all machine learning algorithm for house price prediction. It is important to evaluate multiple algorithms to find the one that works best for your dataset.

Once you have trained and evaluated your model, you can use it to make predictions for new houses. This can be helpful for a variety of purposes, such as buying or selling a house, or assessing the value of a property for tax purposes.

It is important to note that machine learning models are only as good as the data they are trained on. If your data is inaccurate or incomplete, your model's predictions will also be inaccurate. It is also important to remember that house prices are influenced by a variety of factors, some of which may be difficult to predict, such as market conditions and future economic trends..

PREDICTING HOUSE PRICES USING MACHINE LEARNING

ABSTRACT

Real estate is the least transparent industry in our ecosystem. Housing prices keep changing day in and day out and sometimes are hyped rather than being based on valuation. Predicting housing prices with real factors is the main crux of our research project. Here we aim to make our evaluations based on every basic parameter that is considered while determining the price. We use various regression techniques in this pathway, and our results are not sole determination of one technique rather it is the weighted mean of various techniques to give most accurate results.

Predicting House Prices

Now comes the exciting part! By feeding new data into the trained machine learning model, we can predict house prices with remarkable accuracy. This empowers stakeholders to make informed decisions when buying, selling, or investing in real estate, ultimately revolutionizing the industry.

Machine Learning Basics

Machine learning is a subset of *artificial intelligence* that enables computers to learn and make predictions without being explicitly programmed. By analyzing vast amounts of *data*, machine learning algorithms can identify patterns and make accurate predictions. This technology has immense potential in predicting house prices.

Understanding House Prices

Before delving into the power of *machine learning*, let's understand the factors influencing house prices. Factors such as *location*, *size*, *amenities*, and *market demand* play a crucial role. Accurately predicting house prices can empower buyers, sellers, and investors to make informed decisions.

EXISTING SYSTEM

Data is at the heart of technical innovations, achieving any result is now possible using predictive models. Machine learning is extensively used in this approach. Although development of correct and accurate model is needed. Previous algorithms like SVM are not suitable for complex real-world data.

PROPOSED SYSTEM

- Our main focus here is to develop a model which predicts the property cost based on vast datasets using data mining approach.
- The data mining process in a real estate industry provides an advantage to the developers by processing the data, forecasting future trends and thus assisting them to make favourable knowledge-driven decisions.
- Our dataset comprises of various essential parameters and data mining has been at the root of our system.
- We will initially clean up our entire dataset and also truncate the outlier values.
- Finally a system will be made to predict the Real Estate Prices using Deep Learning Approach.

FEASIBILITY STUDY

Basic areas to be covered are –

- Operational Feasibility: The applying can scale back the time consumed to take care of manual records and isn't dull and cumbersome to take care of the records, therefore operational practicableness is assured.
- Technical Feasibility: Minimum hardware requirements: 1.66 GHz Pentium Processor or Intel compatible processor. 1 GB RAM net property, eighty MB disk area.

- Economical Feasibility :Once the hardware and software package needs get consummated, there's no want for the user of our system to pay for any further overhead. For the user, the applying are economically possible within the following aspects: the applying can scale back tons of labor work. therefore the efforts are reduced. Our application can scale back the time that's wasted in manual processes. The storage and handling issues of the registers are resolved .

SYSTEM SPECIFICATIONS

Since this is an AI based Deep Learning approach, we are going to use a dataset extracting the features of houses and algorithmic models to train the dataset for predictions.

- Dataset- We are using an unsupervised dataset,i.e, data without class labels. The objective of this unsupervised technique, is to find patterns in data based on the relationship between data points themselves.

L3

	A	B	C	D	E	F	G	H	I	J
1	-122.23	latitude	housing_median	total_rooms	total_bedrooms	population	households	median_income	median_house_	ocean_proximity
2	-122.22	37.88	41	880	129	322	126	8.3252	452600	NEAR BAY
3	-122.24	37.86	21	7099	1106	2401	1138	8.3014	358500	NEAR BAY
4	-122.25	37.85	52	1467	190	496	177	7.2574	352100	NEAR BAY
5	-122.25	37.85	52	1274	235	558	219	5.6431	341300	NEAR BAY
6	-122.25	37.85	52	1627	280	565	259	3.8462	342200	NEAR BAY
7	-122.25	37.85	52	919	213	413	193	4.0368	269700	NEAR BAY
8	-122.25	37.84	52	2535	489	1094	514	3.6591	299200	NEAR BAY
9	-122.26	37.84	52	3104	687	1157	647	3.12	241400	NEAR BAY
10	-122.25	37.84	42	2555	665	1206	595	2.0804	226700	NEAR BAY
11	-122.26	37.84	52	3549	707	1551	714	3.6912	261100	NEAR BAY
12	-122.26	37.85	52	2202	434	910	402	3.2031	281500	NEAR BAY
13	-122.26	37.85	52	3503	752	1504	734	3.2705	241800	NEAR BAY
14	-122.26	37.85	52	2491	474	1098	468	3.075	213500	NEAR BAY
15	-122.26	37.84	52	696	191	345	174	2.6736	191300	NEAR BAY
16	-122.26	37.85	52	2643	626	1212	620	1.9167	159200	NEAR BAY
17	-122.27	37.85	50	1120	283	697	264	2.125	140000	NEAR BAY
18	-122.27	37.85	52	1966	347	793	331	2.775	152500	NEAR BAY
19	-122.26	37.85	52	1228	293	648	303	2.1202	155500	NEAR BAY
20	-122.27	37.84	50	2239	455	990	419	1.9911	158700	NEAR BAY
21	-122.27	37.84	52	1503	298	690	275	2.6033	162900	NEAR BAY

- The dataset consists of more than 20,000 records.
- Around nine features are provided in the dataset like latitude, total rooms, total bedrooms etc.
- Data mining technique is used for processing of the dataset to extract relevant data to be further used in data models.

- Data model-The basic technique used for the model is Regression Analysis which is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables. It predicts continuous/real values.

5/29/2021

- For our application we are using three types of algorithm train the model, which are-
- **KN-Neighbors**: K nearest neighbors is a simple algorith that stores all available cases and predict the numerical target based on a similarity measure (e.g., distance functions)
- **Support Vector Machine**: The goal of the SVM algorith to create the best line or decision boundary that can segregate n-dimensional space into classes so that we easily put the new data point in the correct category in t future.
- **Artificial Neural Network(ANN)**: It's a computational m in which artificial neurons are nodes, and directed edge weights are connections between neuron outputs and n

HARDWARE AND SOFTWARE REQUIREMENTS

SOFTWARE REQUIREMENTS-

- Python version- 3.8
- Libraries-

✓ Pandas: library for manipulating data in numerical tables.

✓ scikit-learn: machine learning library that features various classification, regression and clustering algorithms.

✓ Matplotlib: Matplotlib is a plotting library for creating static, animated, and interactive visualizations in python.

5/29/2021

- ✓Seaborn: library based on matplotlib. It provides a h
  interface for drawing attractive and informative statis
  graphics
- ✓Tensorflow: The core open source library to help yo
  and train ML models.
- ✓Jupyterlab: JupyterLab is a web-based interactive
  development environment for Jupyter notebooks, co
  data.

HARDWARE REQUIREMENTS-

- ✓1.66 GHz Pentium Processor or Intel compatible processor.
- ✓1 GB RAM
- ✓ 80 MB disk area.

Benefits of Predictive Models

Predictive models powered by machine learning offer numerous benefits. They enable buyers to estimate fair prices, sellers to set competitive prices, and investors to identify lucrative opportunities. By reducing uncertainty and providing data-driven insights, these models enhance decision-making and improve market efficiency.

Conclusion

In conclusion, *machine learning* has the potential to revolutionize the real estate industry by accurately predicting house prices. By leveraging comprehensive datasets, training models, and evaluating their performance, stakeholders can make informed decisions. Embracing this technology will enhance market efficiency and empower buyers, sellers, and investors.

Thank you

Gradient Boosting with scikit-learn:

pythonCopy code

from sklearn.ensemble import GradientBoostingRegressor from sklearn.model_selection import train_test_split from sklearn.metrics import mean_squared_error # Load your dataset and split it 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 a Gradient Boosting Regressor model gb_regressor = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42) # Fit the model to the training data gb_regressor.fit(X_train, y_train) # Make predictions on the test set y_pred = gb_regressor.predict(X_test) # Evaluate the model mse = mean_squared_error(y_test, y_pred) print(f"Mean Squared Error: {mse}")

XGBoost:

You'll need to install the xgboost library if you haven't already:

bashCopy code

pip install xgboost

Here's an example of using XGBoost for regression:

pythonCopy code

import xgboost as xgb from sklearn.metrics import mean_squared_error # Create a DMatrix from your data dtrain = xgb.DMatrix(X_train, label=y_train) dtest = xgb.DMatrix(X_test, label=y_test) # Set hyperparameters params = { 'objective': 'reg:squarederror', 'max_depth': 3, 'learning_rate': 0.1, 'n_estimators': 100, 'seed': 42 } # Train the XGBoost model xgboost_model = xgb.train(params, dtrain) # Make predictions on the test set y_pred = xgboost_model.predict(dtest) # Evaluate the model mse = mean_squared_error(y_test, y_pred) print(f"Mean Squared Error: {mse}")

These code examples demonstrate how to use Gradient Boosting and XGBoost for regression tasks. Make sure to adjust hyperparameters like n_estimators, max_depth, and learning_rate to fine-tune the model for your specific dataset and problem. Additionally, you can explore other gradient boosting libraries like LightGBM and CatBoost, which offer similar functionality and often perform well in different scenarios.

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Load the dataset (replace 'your_dataset.csv' with your dataset file)
data = pd.read_csv('your_dataset.csv')

# Data preprocessing
# 1. Handle missing values if any
# data.dropna(inplace=True)  # Remove rows with missing values
# Alternatively, you can impute missing values

# 2. Feature selection and extraction
# Decide which features (columns) are relevant for the prediction
# X = data[['feature1', 'feature2', ...]]
# y = data['target_column']  # The column containing house prices

# 3. 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)

# 4. Initialize and train the model (in this case, Linear Regression)
model = LinearRegression()
model.fit(X_train, y_train)

# 5. Make predictions
y_pred = model.predict(X_test)

# 6. Evaluate the model
```

```
mse = mean_squared_error(y_test, y_pred)

print(f"Mean Squared Error: {mse}")


# You can further fine-tune your model and try different algorithms for better results.
```

PREDICTING HOUSE PRICES USING MACHINE LEARNING

Introduction

Welcome to the presentation on *Forecasting House Prices with Machine Learning: An Advanced Predictive Modeling Approach*. In this presentation, we will explore how machine learning techniques can be applied to accurately predict house prices. We will discuss the importance of accurate house price predictions for various stakeholders in the real estate industry.

Objective

Our objective is to demonstrate how advanced predictive modeling techniques can be used to forecast house prices with high precision. We will delve into the details of different machine learning algorithms and their application in the real estate domain. By the end of this presentation, you will have a clear understanding of how machine learning can revolutionize house price predictions.

Machine Learning Algorithms for House Price Predictions

We will explore various machine learning algorithms suitable for house price predictions. This includes linear regression, decision trees, random forests, support vector machines, and neural networks. We will discuss the strengths and weaknesses of each algorithm and their applicability in the context of house price predictions. Understanding these algorithms will enable us to choose the best model for our predictive modeling approach.

Importance of House Price Predictions

Accurate house price predictions are crucial for various stakeholders, including homebuyers, sellers, real estate agents, and investors. *Homebuyers* rely on accurate predictions to make informed decisions about their investments. *Sellers* need to price their properties competitively. *Real estate agents* can provide better guidance to their clients. *Investors* can identify profitable opportunities. Machine learning can enhance the accuracy of these predictions, benefiting all stakeholders.

DESIGN AND DEVELOPMENT

Since this project mainly integrates software features rather than hardware ones, the emphasis is primarily on code development as follows-

OVERVIEW OF STEPS

Step 1

- Import required libraries
- Import data from dataset
- Get some information about dataset.

Run Markdown

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import systemcheck
from sklearn.metrics import mean_absolute_percentage_error
```

In [2]:
```
data = pd.read_csv("housing.csv")
```

In [3]:
```
data.head()
```

Out[3]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

In [4]:
```
data[data["total_rooms"]<1000].head()
```

Activate Windows

Step 2-

- Visualizing the data in terms of house prices, affordability, closeness to ocean, number of bedrooms etc.
- First we display a histogram for column 'median_house_value' from the dataset
- Next we display a Heatmap of the dataset. A heatmap() method contains values representing various shades of the same colour for each value to be plotted. Usually the darker shades of the chart represent higher values than the lighter shade.
- Next we display a Distplot() for columns 'median_house_value', 'affordable', and 'ocean_proximity'. A distplot() method lets you show a histogram with a line on it. It plots a univariate distribution of observations.

```
In [10]: data["median_house_value"].hist()
         plt.show()
```

```
In [11]: sns.set()
         sns.heatmap(data.corr(), annot = True)
         plt.figure(figsize = (30,25))
         plt.tight_layout()
         plt.show()
```

```
In [12]: sns.set_color_codes(palette = "bright")
         sns.distplot(data["median_house_value"], color = "r")
         plt.show()
```

```
C:\Users\sreek\AppData\Local\Programs\Python\Python38\lib\site-packages\
` is a deprecated function and will be removed in a future version. Plea
vel function with similar flexibility) or `histplot` (an axes-level func
  warnings.warn(msg, FutureWarning)
```

- Next we display a Countplot() for the column 'ocean_proximity'. A countplot() method is used to show the counts of observations in each categorical bin using bars.
- Next we concatenate the values of dataset with values of 'ocean_proximity' column according to the different levels of distance between the ocean and housing area.
- Next we view the countplot() of 'affordable' column, convert it into string and replace string type "true, false" values by int values 0,1 to represent if the house is affordable or not.

```
In [16]: sns.countplot(x=data["ocean_proximity"])
         plt.show()
```

```
In [17]: data = pd.concat([data,pd.get_dummies(data["ocean_proximity"], prefix="ocean_proximity")],axis=1)
         data.drop(columns=["ocean_proximity"],inplace=True)
         data
```

In [18]:
```
sns.countplot(x=data["affordable"])
plt.show()
#balanced data
```

In [20]:
```
data["affordable"] = data["affordable"].astype(str)
```

In [21]:
```
data["affordable"].replace(["True","False"],[1,0], inplace = True)
```

In [22]:
```
data.head()
```

Out[22]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	affordable	ocean_pro
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	0	

- Next we use the info() function on the dataset which basically shows the datatypes used for different columns.

```
In [23]: data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 20433 entries, 0 to 20639
Data columns (total 15 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   longitude                   20433 non-null  float64
 1   latitude                    20433 non-null  float64
 2   housing_median_age          20433 non-null  float64
 3   total_rooms                 20433 non-null  float64
 4   total_bedrooms              20433 non-null  float64
 5   population                  20433 non-null  float64
 6   households                  20433 non-null  float64
 7   median_income               20433 non-null  float64
 8   median_house_value          20433 non-null  float64
 9   affordable                  20433 non-null  int64
 10  ocean_proximity_<1H OCEAN   20433 non-null  uint8
 11  ocean_proximity_INLAND      20433 non-null  uint8
 12  ocean_proximity_ISLAND      20433 non-null  uint8
 13  ocean_proximity_NEAR BAY    20433 non-null  uint8
 14  ocean_proximity_NEAR OCEAN  20433 non-null  uint8
dtypes: float64(9), int64(1), uint8(5)
memory usage: 2.3 MB
```

Step 3-

- Detecting and removing outliers using 'IsolationForest' algorithm.
- Isolation forest is an unsupervised learning algorithm for anomaly detection that works on the principle of isolating anomalies. It isolates the outliers by randomly selecting a feature from the given set of features and then randomly selecting a split value between the maximum and minimum values of the selected feature.
- In our dataset after running the algorithm we get around 7170 outliers in the form of negative values(-1), which we remove from the data, i.e., we delete 7170 rows from around 20,000 rows in the dataset.

In [24]:
```
from sklearn.ensemble import IsolationForest
clf = IsolationForest(max_samples=100, random_state = 0, contamination= 'auto')
preds = clf.fit_predict(data)
preds
```

Out[24]:
```
array([-1, -1, -1, ...,  1,  1,  1])
```

In [25]:
```
print("number of outliers: ",list(preds).count(-1))
```

```
number of outliers:  7170
```

In [26]:
```
for i,j in list(zip(range(len(preds)),data.index)):
    if preds[i] == -1:
        data.drop([j],inplace=True)
```

In [27]:
```
data
```

Out[27]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	affordable	ocean
10	-122.26	37.85	52.0	2202.0	434.0	910.0	402.0	3.2031	281500.0	0	
21	-122.27	37.85	42.0	1639.0	367.0	929.0	366.0	1.7135	159000.0	1	
90	-122.27	37.80	16.0	994.0	392.0	800.0	362.0	2.0938	162500.0	1	
93	-122.27	37.79	27.0	1055.0	347.0	718.0	302.0	2.6354	187500.0	0	
106	-122.24	37.81	52.0	2026.0	482.0	709.0	456.0	3.2727	268500.0	0	
...	...	...	...	...	...	...	...	...	...	...	
20635	-121.09	39.48	25.0	1665.0	374.0	845.0	330.0	1.5603	78100.0	1	
20636	-121.21	39.49	18.0	697.0	150.0	356.0	114.0	2.5568	77100.0	1	
20637	-121.22	39.43	17.0	2254.0	485.0	1007.0	433.0	1.7000	92300.0	1	
20638	-121.32	39.43	18.0	1860.0	409.0	741.0	349.0	1.8672	84700.0	1	
20639	-121.24	39.37	16.0	2785.0	616.0	1387.0	530.0	2.3886	89400.0	1	

13263 rows × 15 columns

Activate Windows

Step 4-

- Splitting into training and testing data.
- We use the Train set to make the algorithm learn the data's behavior and then check the accuracy of our model on the Test set.
- Divide the data into two values-

✓Features (X): The columns that are inserted into our model will be used to make predictions.

✓Prediction (y): Target variable that will be predicted by the features.

- After defining values of x, y we import train and test functions from scikit learn library.

Case Study: Real Estate Market Analysis

In this section, we will present a real-world case study on real estate market analysis using machine learning. We will showcase how the techniques discussed in this presentation can be applied to a specific market and provide valuable insights. The case study will demonstrate the practical application and effectiveness of our advanced predictive modeling approach.

Limitations and Future Directions

While machine learning techniques offer promising results in house price predictions, there are limitations to consider. We will discuss the challenges associated with data availability, model interpretability, and potential biases. Additionally, we will explore future directions, including the integration of external data sources, incorporation of domain knowledge, and advancements in deep learning for more accurate predictions.

Conclusion

In this presentation, we have explored the application of machine learning in forecasting house prices. We have discussed the importance of accurate predictions for various stakeholders and the challenges associated with house price predictions. By leveraging advanced predictive modeling techniques, we can overcome these challenges and achieve high-precision predictions. Machine learning has the potential to revolutionize the real estate industry by providing valuable insights and improving decision-making processes.

Thank you

Development of predicting House prices using Machine learning

Data collection
Data-Cleaning
Train-Test Splitting
Creating Pipeline
Model Selection
Model Evaluation
Cross Validation
Testing The Pipeline On Test Data
Output

Certainly! To continue building a house price prediction model, you can use Python and popular libraries like scikit-learn and pandas for feature selection, model training, and evaluation. Here's a basic example with comments for each step:

pythonCopy code

Import necessary libraries import pandas as pd from sklearn.model_selection import train_test_split from sklearn.ensemble import RandomForestRegressor from sklearn.metrics import mean_squared_error

Load your dataset (replace 'data.csv' with your dataset file) data = pd.read_csv('data.csv') # Feature selection: Choose relevant features

In this example, we select 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', and 'yr_built' selected_features = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'yr_built'] X = data[selected_features] y = data['price']

Split the dataset 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 and train the model (Random Forest Regressor in this case) model = RandomForestRegressor(n_estimators=100, random_state=42) model.fit(X_train, y_train)

Make predictions on the test set y_pred = model.predict(X_test)

Evaluate the model using Mean Squared Error (MSE) mse = mean_squared_error(y_test, y_pred) print(f"Mean Squared Error: {mse}")

You can further fine-tune your model and evaluate its performance using various metrics and techniques.

This is a basic example, and you can customize it according to your dataset and requirements. You may consider other regression models, hyperparameter tuning, feature engineering, and cross-validation for more robust predictions.

PREDICTING HOUSE PRICES USING MACHINE LEARNING

Introduction

Welcome to the presentation on *Unveiling the Future: Harnessing the Power of Machine Learning to Predict House Prices*. In this session, we will explore how **machine learning** algorithms can accurately predict house prices, enabling better decision-making in the real estate industry.

What is Machine Learning?

Machine Learning is a subset of **artificial intelligence** that enables computers to learn and make predictions without being explicitly programmed. It leverages **statistical models** and **algorithms** to analyze data and identify patterns, allowing us to make accurate predictions.

Importance of House Price Prediction

Accurate house price prediction is crucial for various stakeholders in the real estate industry, including **homebuyers**, **sellers**, and **investors**. It helps in making informed decisions, understanding market trends, and maximizing returns on investments.

TRAINING AND TESTING

To test the accuracy of the model we use a metric mean_absolute_percentage_error which we import from scikit library. The algorithms used for training and testing the model are as follows-

- KNN- First we import the KNeighborsRegressor() function from scikit library and then run the algorithm on training set and then on test set.
- We obtain the mean absolute percentage error of knn as 38.91504096125152 %

Spliting into training and testing data

In [28]:
```
x = data.drop(columns=["longitude","latitude","median_house_value"])
y = data["median_house_value"]
```

In [29]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y, random_state = 1)
```

KN-Neighbors

In [30]:
```
from sklearn.neighbors import KNeighborsRegressor
```

In [31]:
```
knn = KNeighborsRegressor()
knn.fit(x_train, y_train)
```

Out[31]:
```
KNeighborsRegressor()
```

In [32]:
```
b = knn.predict(x_test)
```

In [33]:
```
print("mean_absolute_percentage_error of KNN model:",mean_absolute_percentage_error(b,y_test)*100,"%")
```

```
mean_absolute_percentage_error of KNN model: 38.91504096125152 %
```

- SVM- Similarly we import SVR() from scikit library.
- Then we run the svm algorithm on the training data and then the test data.
- The mean absolute percentage error of svm obtained is 32.246431895552135 %, which is slightly less than that of knn algorithm.

- ANN- We use tensor flow library to build an ANN here for the application.
- We also use the Keras library here which is an open source deep learning framework for python. Keras is one of the most powerful and easy to use python library, which is built on top of popular deep learning libraries like TensorFlow, Theano, etc., for creating deep learning models.

- To build the different layers of ANN we use ‘Flatten’ and ‘Dense’ function of keras framework.
- Dense layer is the regular deeply connected neural network layer. It is most common and frequently used layer.
- Flatten layer is used to flatten the input.
- Then after building the layers of the model we need to compile the model so that the code is converted into machine readable code.
- We can view the summary of the model using model.summary() function.

Support Vector Machine

In [34]:
```
from sklearn.svm import SVR
```

In [35]:
```
svm = SVR(C = 2, kernel = "linear")
svm.fit(x_train, y_train)
```

Out[35]:
```
SVR(C=2, kernel='linear')
```

In [36]:
```
c = svm.predict(x_test)
```

In [37]:
```
print("mean_absolute_percentage_error of SVM model: ",mean_absolute_percentage_error(c,y_test)*100,"%")
```

```
mean_absolute_percentage_error of SVM model:  32.246431895552135 %
```

In [38]:
```
import tensorflow as tf
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=x.shape[1:]),
  tf.keras.layers.Dense(256, activation='relu'),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(32, activation='relu'),
  tf.keras.layers.Dense(16, activation='relu'),
  tf.keras.layers.Dense(8, activation='relu'),
  tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam',
              loss='mean_squared_error')
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 12)                0
_________________________________________________________________
dense (Dense)                (None, 256)               3328
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 16)                528
_________________________________________________________________
dense_5 (Dense)              (None, 8)                 136
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 9
=================================================================
Total params: 47,233
Trainable params: 47,233
Non-trainable params: 0
_________________________________________________________________
```

```
In [39]: model.fit(x_train, y_train, epochs = 100, batch_size = 32)
```

```
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 12)                0
_________________________________________________________________
dense (Dense)                (None, 256)               3328
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 16)                528
_________________________________________________________________
dense_5 (Dense)              (None, 8)                 136
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 9
=================================================================
Total params: 47,233
Trainable params: 47,233
Non-trainable params: 0
_________________________________________________________________
```

```
In [39]: model.fit(x_train, y_train, epochs = 100, batch_size = 32)
```

- As we notice , among the three models we find the ANN model to be the most accurate on the dataset.
- Next we visualize the output of the three models graphically, where the original output is represented in blue colour, the svm output is represented in green colour and the ANN output is represented in red colour.

IMPLEMENTATION AND OUTPUT

- To implement and check the accurate output of the model, we first declare variables for each attribute of the dataset and provide user defined values to the variables.
- These values are fed as input to the model which predicts the final output, i.e, the price of the house based on the given input.
- The final output or prediction is as follows-

In [56]:
```
housing_median_age = 54
total_rooms = 500
total_bedrooms = 200
population = 200
households = 200
median_income = 3.5
affordable = 1
ocean_proximity_1 = 0
ocean_proximity_INLAND = 0
ocean_proximity_ISLAND = 1
ocean_proximity_NEARBAY = 0
ocean_proximity_NEAROCEAN = 0

input = [housing_median_age,total_rooms,total_bedrooms,population,households,median_income,affordable,ocean_proximity_1,ocean_pro

input = np.array(input).reshape(1,-1)

output = model.predict(input)
print("the house price is:",output[0][0])
```

```
the house price is: 216358.81
```

Activate Windows
Go to Settings to activ

CONCLUSION

From the training and testing of dataset on the model, a strong deduction can be made that the model ANN works most effectively on the data producing lower levels of errors, and hence we can conclude that Deep Learning techniques prove useful in the implementation and estimation of HOUSE PRICE PREDICTION.

THANK YOU