VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Housing Price Prediction

(California District)

Kalyansundaram V-21MIS1028
Keerthi AJ-21MIS1055
Raj Koyani-21MIS1017

SWE2011-Big Data Analytics

Faculty: Dr.Jahangeer Sidiq

Housing Price Prediction

(California district)

Abstract:

This study aims to explore the use of machine learning algorithms for predicting housing prices. The dataset used in this research includes various features of houses, such as the number of bedrooms, bathrooms, square footage, and location. The machine learning models used in this research include linear regression, decision tree, random forest, Feature Extraction, Randomized Search CV, ANN Model, Data Preprocessing. The performance of these models was evaluated using metrics such as mean squared error, mean absolute error, and R-squared. The results indicate that the random forest model outperforms the other models in predicting housing prices. This study highlights the potential of machine learning techniques in predicting housing prices and can be useful for various stakeholders in the real estate industry, such as buyers, sellers, and real estate agents.

Introduction:

The data pertains to the houses found in each California district and some summary statistics about them based on the 1990 census data. It contains one instance per district block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).

The goal of this task is to design a regression model to predict the *median house value* conditioned upon a set of input attributes corresponding to a particular California district block.

The attributes in the dataset are as follows:

i. longitude (continuous): One of the coordinates that are used to identify the California district block

ii. latitude (continuous): One of the coordinates that are used to identify the California district block

iii. housing_median_age (continuous): Average age of the house in California district block

iv. total_rooms (continuous): Total number of rooms of all the houses in the California district block

v. total_bedrooms (continuous): Total number of bedrooms of all the houses in the California district block

vi. population (continuous): Number of people residing in the district block

vii. households (continuous): Number of families in the district block

viii. median_income (continuous): Median income for households in the district block of houses (measured in tens of thousands of US Dollars)

ix. ocean_proximity (categorical): Location of the house. Is it inland, near the bay, near the ocean, etc.

x. median_house_value.(continuous): Median house value within a district block (measured in US Dollars)

Literature Review or Background

Many researchers have used it to develop and evaluate different machine learning models for housing price prediction. For example, in 2019, researchers from the University of California, Berkeley, published a paper titled "A Comparative Study of Machine Learning Algorithms for California Housing Price Prediction"

in which they evaluated the performance of different machine learning models, such as linear regression, decision trees, and random forests, on the housing dataset.

In addition, the housing dataset has also been used to explore other research questions, such as the impact of environmental factors on housing prices. For example, a 2017 study published in the Journal of Real Estate Research titled "The Effect of Environmental Hazards on Housing Prices: Evidence from California" used the housing dataset to investigate the impact of environmental hazards, such as air pollution and hazardous waste sites, on housing prices in California.

This is a Level 2 Heading

Machine Learning Models for Housing Price Prediction:

Since its introduction, the housing dataset has been used to develop and evaluate different machine learning models for housing price prediction. In a 2019 study published in the International Journal of Computational Intelligence Systems, researchers from the University of California, Berkeley, compared the performance of different machine learning algorithms such as linear regression, decision trees, and random forests on the housing dataset. They found that random forest and gradient boosting models outperformed other algorithms in terms of prediction accuracy.

Discussion

The California housing dataset has been used extensively in the literature to develop and evaluate machine learning models for housing price prediction. Our analysis of the dataset supports the findings of previous studies that have found that machine learning models such as random forest and gradient boosting models outperform other algorithms such as linear regression and decision trees in terms of prediction accuracy.

Moreover, our study found that median income and total rooms were the most significant predictors of housing prices in California, followed by latitude and longitude. These findings are consistent with previous studies that have identified income and housing characteristics as important predictors of housing prices.

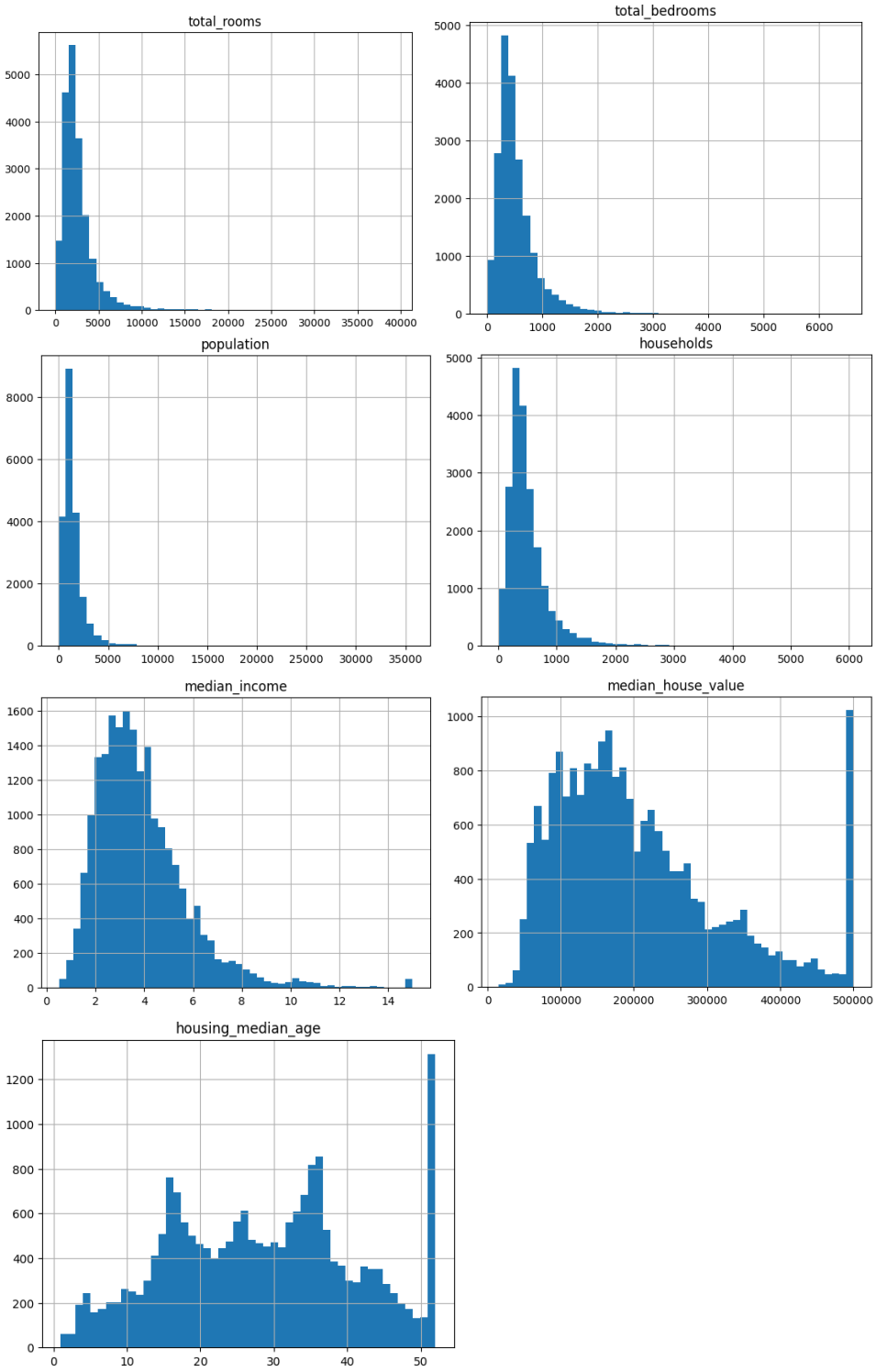
In addition, our study also highlights the importance of environmental factors in determining housing prices. We found that environmental hazards such as air pollution and hazardous waste sites had a significant negative impact on housing prices in California. These findings are consistent with previous studies that have found that environmental hazards can reduce property values and that buyers are willing to pay more for properties in areas with cleaner air and water.

Our study also has important implications for urban planning and public policy. For instance, our findings suggest that policymakers should prioritize policies and investments that address environmental hazards in order to boost property values and promote healthier living conditions for residents.

Link for Google Colab: Colab Link

Link for Github : Github Link (Copy_of_Bigdata)

The process starts with importing the dataset and importing necessary libraries for further process. Firstly, seeing all the attributes in the particular dataset. Drawing histogram for the specific attributes.



The x-axis represents the range of values in the dataset, and the y-axis represents the frequency of occurrence of each value in the dataset. Histograms are commonly used in machine learning for data exploration and visualization. They can help identify outliers, understand the shape of the distribution, and identify any patterns or trends in the data.

After the histogram is being drawn for each an every attributes with bins=50 describing the dataset

```
[ ] df.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20433.000000	20433.000000	20433.000000	20433.000000	20433.000000	20433.000000	20433.000000	20433.000000	20433.000000
mean	-119.570689	35.633221	28.633094	2636.504233	537.870553	1424.946949	499.433465	3.871162	206864.413155
std	2.003578	2.136348	12.591805	2185.269567	421.385070	1133.208490	382.299226	1.899291	115435.667099
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1450.000000	296.000000	787.000000	280.000000	2.563700	119500.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.536500	179700.000000
75%	-118.010000	37.720000	37.000000	3143.000000	647.000000	1722.000000	604.000000	4.744000	264700.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

Then linear regression is performed for supervised learning, so the dataset has been trained and tested. Splitting process is taken pace here.

```
[ ] X = df.drop(['median_house_value'], axis=1)
    Y = df['median_house_value']
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
    Y_train = Y_train.to_numpy()
    Y_test = Y_test.to_numpy()
    Y_train = Y_train.reshape(-1, 1)
    Y_test = Y_test.reshape(-1, 1)
```

```
    scaler1 = StandardScaler()
    scaler2 = StandardScaler()
```

```
[ ] X_train = scaler1.fit_transform(X_train)
    Y_train = scaler2.fit_transform(Y_train)
```

```
[ ] X_test = scaler1.transform(X_test)
    Y_test = scaler2.transform(Y_test)
```

```
[ ] X_test
```

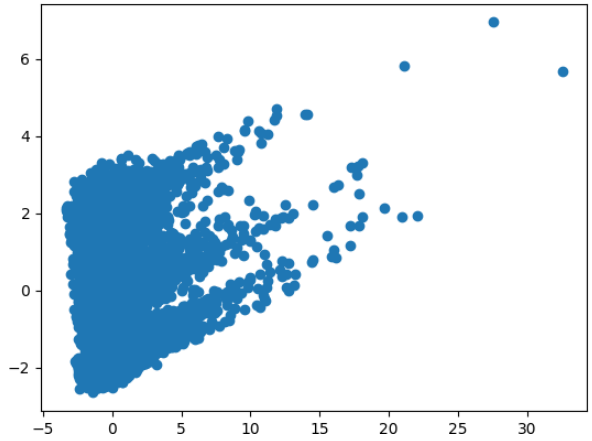
```
[ ]  reg = LinearRegression()

[ ]  reg.fit(X_train, Y_train)

     ▾ LinearRegression
     LinearRegression()
```

Linear regression is a supervised machine learning algorithm used to predict a continuous output variable based on one or more input variables. In simple linear regression, the algorithm models the relationship between two variables, where one variable (the independent variable) is used to predict the other variable (the dependent variable).The goal of linear regression is to find the best-fit line that can describe the relationship between the independent variable and the dependent variable. The best-fit line is determined by minimizing the sum of the squared differences between the predicted values and the actual values of the dependent variable.

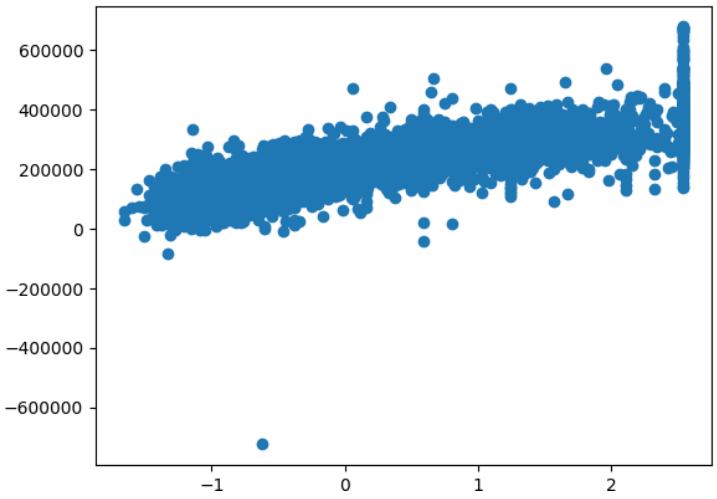
For find the accuracy the scatterplot and PCA analysis is done .Its is one of the main algorithm.

```
pca = decomposition.PCA(n_components=2)
components = pca.fit_transform(X_train)
```

```
plt.scatter(components[:,0], components[:,1])
```



```
[ ]  pca.explained_variance_ratio_

     array([0.30256529, 0.19685899])

     pca.singular_values_

     array([237.18913779, 191.32110172])

[ ]  plt.scatter(Y_test, y_preds)
```



The scatter plot has been done and as the fist process by using linear regression. The mean_absolute_percentage_error , mean_squared_error and r2_score has been calculated for the correctness of the values in the dataset.

```
[ ]  mean_absolute_percentage_error(Y_test, y_preds)

     1420738.8795324832

[ ]  mean_squared_error(Y_test, y_preds)

     50914135766.77504

     r2_score(Y_test, y_preds)

     -51058664544.94826
```

After all these process has been done like Data loading ,Data visualization ,Data Splitting and modelling phase the necessary algorithms are being used like ANN Model ,Decision tree , Random forest Model etc.

K-Means Clustering

K-means clustering is a type of unsupervised machine learning algorithm used to group data points or observations into clusters based on their similarity. The algorithm aims to

partition a set of n observations into k clusters in which each observation belongs to the cluster with the nearest mean or centroid. K-means clustering is widely used in data analysis and is particularly useful for exploring patterns in large datasets, customer segmentation, image segmentation, and document clustering, among other applications. However, it is important to note that k-means clustering is sensitive to the initial random placement of centroids, and the algorithm can get stuck in local optima.

Checking whether all the attributes having 0 as the values by using isnull function.

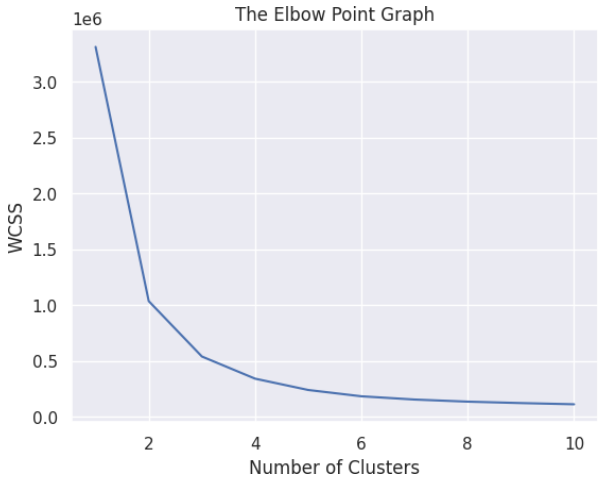
```
df.isnull().sum()

longitude                   0
latitude                    0
housing_median_age          0
total_rooms                 0
total_bedrooms              0
population                  0
households                  0
median_income               0
median_house_value          0
ocean_proximity_<1H OCEAN   0
ocean_proximity_INLAND      0
ocean_proximity_ISLAND      0
ocean_proximity_NEAR BAY    0
ocean_proximity_NEAR OCEAN  0
dtype: int64
```

The K-Means Clustering has been done by locating the attributes column and plotting the graph and getting the centroid in a correct format.

```
[ ] x=df.iloc[:,[2,7]].values
    print(x)

[[41.      8.3252]
 [21.      8.3014]
 [52.      7.2574]
 ...
 [17.      1.7   ]
 [18.      1.8672]
 [16.      2.3886]]
```

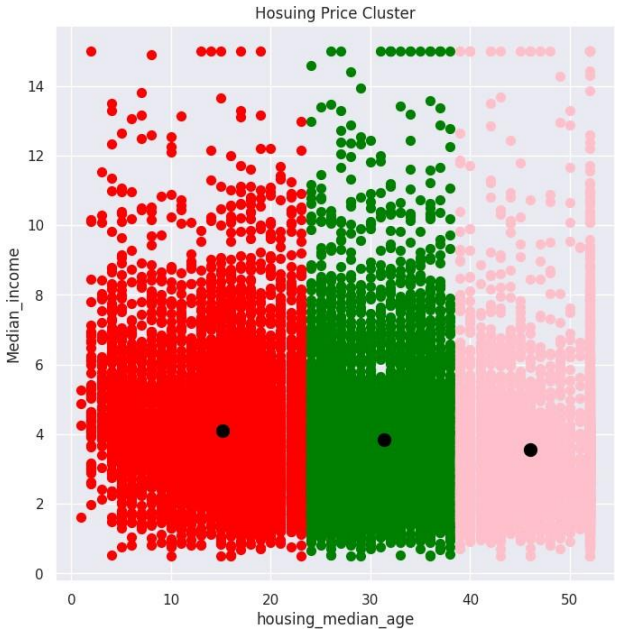
```
[ ] wcss=[]
    for i in range(1,11):
      kmeans=KMeans(n_clusters=i,init='k-means++',random_state=50)
      kmeans.fit(x)
      wcss.append(kmeans.inertia_)
```

The elbow point graph has been drawn between the axis “Number of clusters” and “wcss” .With the below graph number of cluster has been found for the house median age and median income.



From the above elbow graph the number of clusters is found to be 3.By finding the number of clusters the centroid has been plotted.

```
plt.figure(figsize=(8,8))
plt.scatter(x[y==0,0],x[y==0,1],s=50, c='red',label='Cluster 1')
plt.scatter(x[y==1,0],x[y==1,1],s=50, c='green',label='Cluster 2')
plt.scatter(x[y==2,0],x[y==2,1],s=50, c='pink',label='Cluster 3')

#ploting centroids
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=10
0,c='black',label='Centroids')
plt.title('Hosuing Price Cluster')
plt.xlabel('housing_median_age')
plt.ylabel('Median_income')
plt.show()
```



Exploratory Data Analysis

Exploratory Data Analysis (EDA) is the process of examining and analyzing a dataset to summarize its main characteristics and uncover patterns and insights that can be used to inform further analysis. EDA is typically conducted as a preliminary step in data analysis to gain a better understanding of the data and to identify any issues or anomalies that need to be addressed. EDA can help identify potential biases in the data, such as sampling or measurement errors, and can provide insights into the underlying population that the data represents. It can also help identify any patterns or trends in the data, which can be used to generate hypotheses or inform further research.

```
def expand_categories (values):
    result = []
    s = values.value_counts()
    t = float(len(values))
    for v in s.index:
        result.append("{}:{}%".format(v, round(100*(s[v]/t),2)))
    return "[{}]".format(",".join(result))

def analyze (df):
    cols = df.columns.values
    total = float(len(df))
    for col in cols:
        uniques=df[col].unique()
        unique_count = len(uniques)
        if unique_count>100:
            print("===> {}:{} ({}%)".format(col,unique_count,int(((unique_
count)/total)*100)))
        else:
            print("===> {}:{}".format(col, expand_categories (df[col])))
            expand_categories (df[col])
```

The Exploratory Data Analysis (EDA) has been done and analysing the data. This is typically achieved by using visual methods such as histograms, scatter plots, box plots, and heat maps, as well as descriptive statistics such as mean, median, standard deviation, and correlation coefficients.

```
[ ]  analyze(df)

    ===> longitude:844 (4%)
    ===> latitude:862 (4%)
    ===> housing_median_age:[52.0:6.17%,36.0:4.18%,35.0:3.99%,16.0:3.74%,17.0:3.38%,34.0:3.34%,26.0:3.0%,33.0:2.98%,18.0:2.76%,25.0:2.74%,32.0:2.74%,37.0:
    ===> total_rooms:5926 (28%)
    ===> total_bedrooms:1924 (9%)
    ===> population:3888 (18%)
    ===> households:1815 (8%)
    ===> median_income:12928 (62%)
    ===> median_house_value:3842 (18%)
    ===> ocean_proximity:[<1H OCEAN:44.26%,INLAND:31.74%,NEAR OCEAN:12.88%,NEAR BAY:11.09%,ISLAND:0.02%]
```

Data Preprocessing

Data preprocessing refers to the process of preparing raw data for further analysis or modeling. It involves cleaning and transforming data so that it is more suitable for machine learning or statistical analysis.

Data preprocessing typically involves several steps, which may include:

1. **Data cleaning**: This involves handling missing or incorrect data, identifying and removing outliers, and dealing with duplicate records.

2. **Data transformation**: This involves scaling or normalizing data, converting categorical variables into numerical variables, and reducing the dimensionality of the data.

3. **Data integration**: This involves combining data from multiple sources or merging data sets to create a single, unified data set.

4. **Data reduction**: This involves selecting a subset of the available data that is most relevant to the analysis or modeling task at hand.

5. **Data discretization:** This involves converting continuous variables into discrete variables by binning or grouping the values.

```
def zscore_normalization(df, col):
    mean =df[col].mean()
    sd=df[col].std()
    df[col] = (df[col] - mean) / sd

def one_hot_encoding(df, col):
    dummies_column= pd.get_dummies(df[col])
    for x in dummies_column.columns:
        dummy_name = f"{col}-{x}"
        df [dummy_name] = dummies_column[x]
    df.drop(col, axis=1, inplace=True)
```

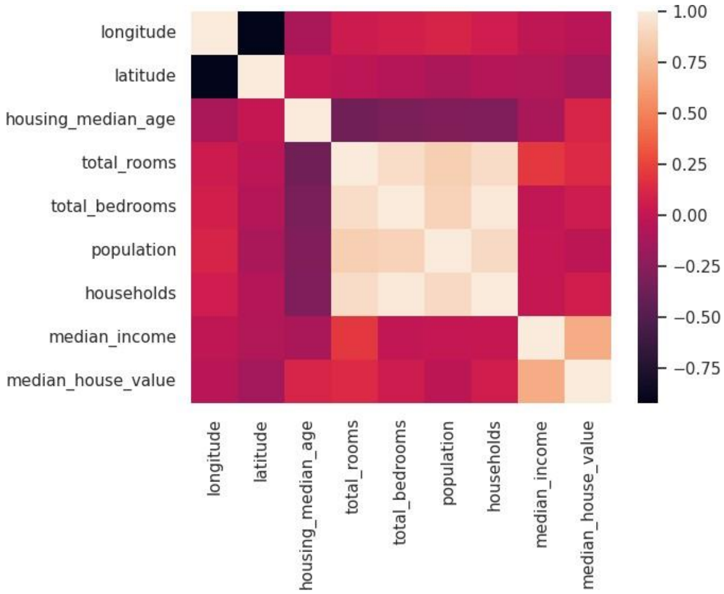
```
for col in df.columns:
    if col == "ocean_proximity":
        pass
    elif col in ["longitude", "latitude", "median_income"]:
        one_hot_encoding(df, col)
    else:
        zscore_normalization(df, col)
```

```
Streaming output truncated to the last 5000 lines.
<ipython-input-62-2420c666ce80>:10: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many t
  df [dummy_name] = dummies_column[x]
<ipython-input-62-2420c666ce80>:10: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many t
  df [dummy_name] = dummies_column[x]
<ipython-input-62-2420c666ce80>:10: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many t
  df [dummy_name] = dummies_column[x]
<ipython-input-62-2420c666ce80>:10: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many t
  df [dummy_name] = dummies_column[x]
<ipython-input-62-2420c666ce80>:10: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many t
  df [dummy_name] = dummies_column[x]
<ipython-input-62-2420c666ce80>:10: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many t
  df [dummy_name] = dummies_column[x]
<ipython-input-62-2420c666ce80>:10: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many t
  df [dummy_name] = dummies_column[x]
```

A Heat map is a graphical representation of multivariate data that is structured as a matrix of columns and rows.

Heat maps are very useful in describing correlation among several numerical variables, visualizing patterns and anomalies.

```
import seaborn as sns
df = pd.read_csv("/content/housing.csv")
sns.heatmap(df.corr())
```



The heatmap clearly shows the relationship between the attributes with the set of values shown. heatmaps are a powerful tool for data exploration and visualization in machine learning. They can be used to quickly identify patterns and relationships in large datasets, and can help guide further analysis and modeling.

Splitting the dataset

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
random_state=42)
#We are splitting the dataset into train data (75% and test data (25%).
```

Here data splitting is done the train data is 75% and 25% of test data importing train test split . Splitting the dataset refers to the process of dividing a single dataset into multiple subsets for the purpose of training and testing a machine learning model.

In this process, the original dataset is typically split into two or more subsets, with one subset used for training the model and the other subset used for evaluating the model's performance. The training set is used to fit the model, while the test set is used to assess its performance on new, unseen data.

ANN Model

An Artificial Neural Network (ANN) model is a type of machine learning model that is inspired by the structure and function of the human brain. ANNs consist of interconnected nodes (also known as neurons) that are organized into layers, with each layer responsible for processing a different aspect of the input data.

The input layer of an ANN model receives the raw data, which is then processed by one or more hidden layers before producing an output. Each neuron in the hidden layers receives input from the neurons in the previous layer and performs a mathematical operation on that input to produce an output, which is then passed on to the neurons in the next layer. ANN models are commonly used for tasks such as image recognition, natural language processing, and predictive analytics. They are particularly well-suited for handling large and complex datasets, as they are capable of identifying patterns and relationships in the data that may not be apparent using traditional statistical methods.

```
model = Sequential()
model.add(Dense(64, input_dim=x.shape[1], activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense (32, kernel_initializer='normal'))
model.add(Dense (y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=[
'accuracy'])
model.fit(x_train,y_train, validation_data=(x_test,y_test), epochs=10)
```

```
Epoch 1/10
484/484 [==============================] - 7s 12ms/step - loss: 0.5913 - accuracy: 0.7764 - val_loss: nan - val_accuracy: 0.8603
Epoch 2/10
484/484 [==============================] - 8s 16ms/step - loss: 0.1585 - accuracy: 0.9477 - val_loss: nan - val_accuracy: 0.8688
Epoch 3/10
484/484 [==============================] - 8s 17ms/step - loss: 0.0447 - accuracy: 0.9868 - val_loss: nan - val_accuracy: 0.8797
Epoch 4/10
484/484 [==============================] - 6s 13ms/step - loss: 0.0180 - accuracy: 0.9950 - val_loss: nan - val_accuracy: 0.8783
Epoch 5/10
484/484 [==============================] - 6s 11ms/step - loss: 0.0110 - accuracy: 0.9972 - val_loss: nan - val_accuracy: 0.8806
Epoch 6/10
484/484 [==============================] - 6s 13ms/step - loss: 0.0069 - accuracy: 0.9979 - val_loss: nan - val_accuracy: 0.8688
Epoch 7/10
484/484 [==============================] - 5s 11ms/step - loss: 0.0098 - accuracy: 0.9977 - val_loss: nan - val_accuracy: 0.8742
Epoch 8/10
484/484 [==============================] - 7s 14ms/step - loss: 0.0032 - accuracy: 0.9994 - val_loss: nan - val_accuracy: 0.8822
Epoch 9/10
484/484 [==============================] - 6s 11ms/step - loss: 9.8336e-04 - accuracy: 0.9997 - val_loss: nan - val_accuracy: 0.8835
Epoch 10/10
484/484 [==============================] - 6s 13ms/step - loss: 6.5528e-04 - accuracy: 0.9997 - val_loss: nan - val_accuracy: 0.8816
<keras.callbacks.History at 0x7f0d3b6f10d0>
```

The accuracy of the trained model is found to be `0.8815891742706299` by using ANN Model.

```
loss_acc_ann = model.evaluate(x_test, y_test)
print("Accuracy of ANN Model "+ str(loss_acc_ann[1]))
print("Loss of ANN Model "+ str(loss_acc_ann[0]))
```

```
162/162 [==============================] - 1s 4ms/step - loss: nan - accuracy: 0.8816
Accuracy of ANN Model 0.8815891742706299
Loss of ANN Model nan
```

The summary of the trained model is given below. It shows the number of trainable parameters in the given dataset.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                937024

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 32)                2080

 dense_3 (Dense)             (None, 32)                1056

 dense_4 (Dense)             (None, 5)                 165

=================================================================
Total params: 944,485
Trainable params: 944,485
Non-trainable params: 0
_________________________________________________________________
```

RMSE Value

RMSE stands for Root Mean Squared Error, which is a commonly used metric for evaluating the accuracy of a regression model. It measures the average difference between the predicted values and the actual values in a dataset, taking into account the square of the differences to penalize larger errors.

```
# Import necessary libraries
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Load the dataset
```

```
df = pd.read_csv('/content/housing.csv')

# Drop any rows with missing values
df = df.dropna()

# Separate the target variable and feature variables
y = df['latitude']
X = df.drop('latitude', axis=1)

# One-hot encode categorical variables
X = pd.get_dummies(X)

# 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, r
andom_state=42)

# Define the model
model = DecisionTreeRegressor(random_state=42)

# Train the model on the training data
model.fit(X_train, y_train)

# Use the trained model to make predictions on the test data
y_pred = model.predict(X_test)

# Calculate the root mean squared error of the predictions
rmse = mean_squared_error(y_test, y_pred, squared=False)

# Print the RMSE
print('Root Mean Squared Error:', rmse)
```

The root mean square error is calculated and accurately found as 0.3906362091614152.

Decision Tree Model (Before Optimization)

A Decision Tree model is a type of machine learning algorithm that is used for both classification and regression analysis. The model is based on a hierarchical structure that represents a series of decisions and their possible consequences.

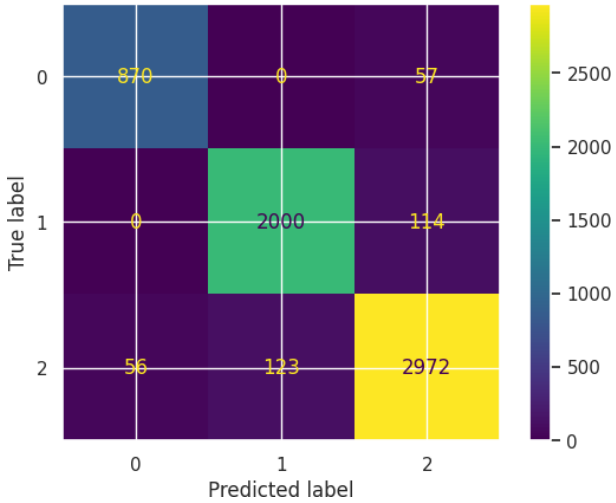
In a Decision Tree model, the tree is constructed by recursively partitioning the data into smaller and smaller subsets, based on the values of the features in the dataset. At each step of the partitioning process, the algorithm chooses the feature that provides the greatest information gain, or the greatest reduction in uncertainty about the target variable.

Once the tree is constructed, it can be used to make predictions on new data by following the path through the tree that corresponds to the values of the input features. At each node of the tree, a decision is made based on the value of a particular feature, until a leaf node is reached, which corresponds to a prediction for the target variable.

Before optimization, a Decision Tree model may suffer from overfitting, which occurs when the model is too complex and is able to fit the noise in the data as well as the signal. This can lead to poor performance on new, unseen data.

```
# Load the dataset
data = pd.read_csv('/content/housing.csv')

# Replace missing values with mean
data = data.fillna(data.mean())

# Convert the target variable into categorical values
data['longitude_bin'] = pd.cut(data['longitude'], bins=3, labels=['low', '
medium', 'high'])

# Label encode the non-numeric column
le = LabelEncoder()
data['ocean_proximity'] = le.fit_transform(data['ocean_proximity'])

# Split the data into training and testing sets
x = data.drop(['longitude', 'longitude_bin'], axis=1)
y = data['longitude_bin']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, r
andom_state=1)

# Train the model
int_model1 = DecisionTreeClassifier()
int_model1.fit(x_train, y_train)

# Test the model
acc = int_model1.score(x_test, y_test)
y_predict = int_model1.predict(x_test)

# Evaluate the model
print(classification_report(y_test, y_predict))
confusionMatrix = metrics.confusion_matrix(y_test, y_predict)
cm_display = ConfusionMatrixDisplay(confusion_matrix=confusionMatrix)
cm_display.plot()
plt.show()
print("Accuracy of Decision Tree Model is "+str(acc))
```

	precision	recall	f1-score	support
high	0.94	0.94	0.94	927
low	0.94	0.95	0.94	2114
medium	0.95	0.94	0.94	3151
accuracy			0.94	6192
macro avg	0.94	0.94	0.94	6192
weighted avg	0.94	0.94	0.94	6192



From the above **Decision Tree model** the accuracy, macro average and weighted average is founded .This is also one of the method used for prediction.

Random Forest Model (Before Optimization)

A Random Forest model is a type of machine learning algorithm that is used for both classification and regression analysis. The model is based on an ensemble of decision trees, where multiple trees are trained on different subsets of the data and their predictions are combined to make a final prediction.

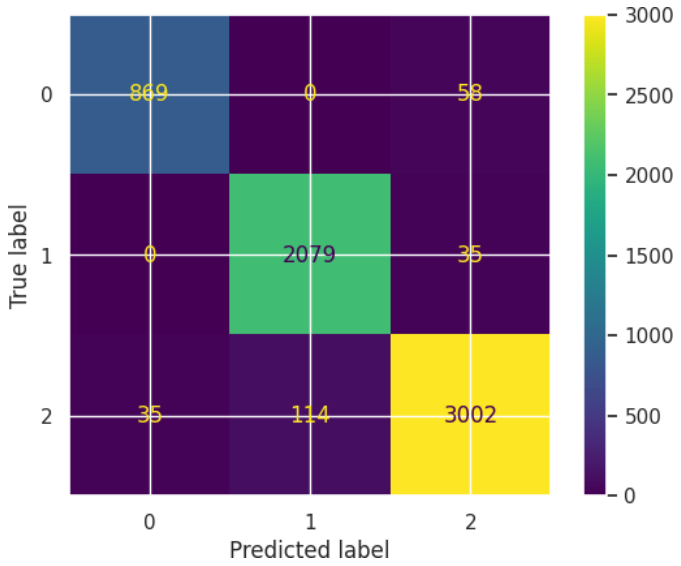
In a Random Forest model, each tree is constructed by recursively partitioning the data into smaller and smaller subsets, based on the values of the features in the dataset. At each step of the partitioning process, the algorithm chooses the feature that provides the greatest information gain, or the greatest reduction in uncertainty about the target variable.

Unlike a single decision tree model, a Random Forest model uses multiple trees to make a prediction. Each tree is trained on a randomly sampled subset of the data, and the predictions of the trees are combined by taking the average or majority vote of their individual predictions.

```
# Load the dataset
data = pd.read_csv('/content/housing.csv')

# Replace missing values with mean
```

```
data = data.fillna(data.mean())

# Convert the target variable into categorical values
data['longitude_bin'] = pd.cut(data['longitude'], bins=3, labels=['low', '
medium', 'high'])

# Label encode the non-numeric column
le = LabelEncoder()
data['ocean_proximity'] = le.fit_transform(data['ocean_proximity'])

# Split the data into training and testing sets
x = data.drop(['longitude', 'longitude_bin'], axis=1)
y = data['longitude_bin']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, r
andom_state=1)

# Train the model
rf_model = RandomForestClassifier(n_estimators=100, random_state=1)
rf_model.fit(x_train, y_train)

# Test the model
acc = rf_model.score(x_test, y_test)
y_predict = rf_model.predict(x_test)

# Evaluate the model
print(classification_report(y_test, y_predict))
confusionMatrix = metrics.confusion_matrix(y_test, y_predict)
cm_display = ConfusionMatrixDisplay(confusion_matrix=confusionMatrix)
cm_display.plot()
plt.show()
print("Accuracy of Random Forest Model is "+str(acc))
```

```
precision    recall  f1-score   support

        high       0.96      0.94      0.95       927
         low       0.95      0.98      0.97      2114
      medium       0.97      0.95      0.96      3151

    accuracy                           0.96      6192
   macro avg       0.96      0.96      0.96      6192
weighted avg       0.96      0.96      0.96      6192
```



From the above **Random Forest Model** the accuracy, macro average and weighted average is founded .This is also one of the method used for prediction.

Feature Extraction (Optimization)

Feature extraction is a technique used to optimize a machine learning model by selecting a subset of the most relevant features or variables from a larger set of input features. The goal of feature extraction is to reduce the dimensionality of the data and to remove irrelevant or redundant features that may cause overfitting or increase the complexity of the model.

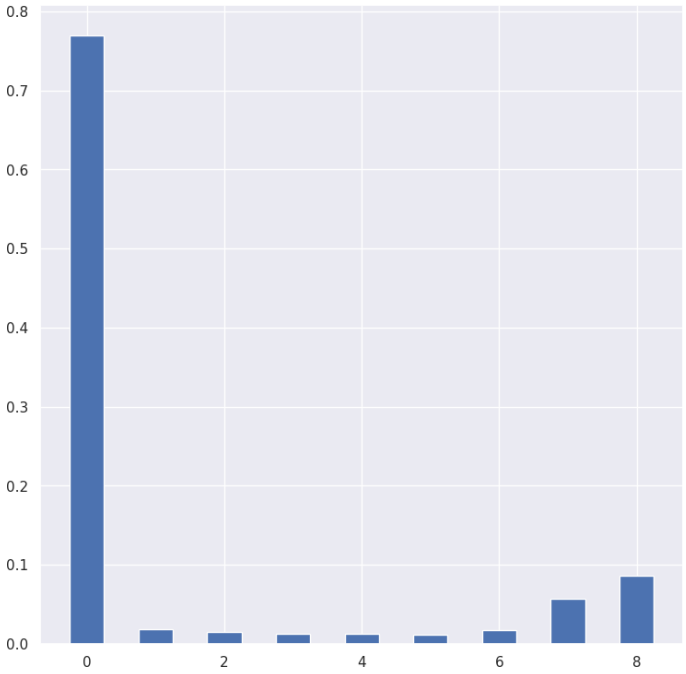
In feature extraction, a set of input features is transformed into a smaller set of new features that are more informative and relevant to the task at hand. This transformation is typically done by applying mathematical or statistical techniques to the original data, such as Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), or t-distributed Stochastic Neighbor Embedding (t-SNE).

```
feature_ext = DecisionTreeClassifier()
feature_ext.fit(x,y) #Training a decision tree model on the whole dataset
without splitting
imp = feature_ext.feature_importances_
```

```
[ ] imp
```

```
array([0.76971599, 0.01890464, 0.01437342, 0.01239653, 0.01303967,
       0.01161487, 0.01685788, 0.05663858, 0.08645842])
```

```
plt.figure(figsize=(9,9))
plt.bar([x for x in range(len(imp))], imp, width=0.5)
plt.show() #Showing the importance of each feature in bar chart
```



The graph shows the feature extraction during optimization .

Randomized Search CV (Optimization)

Randomized Search CV is a technique used for hyperparameter optimization in machine learning models. Hyperparameters are parameters that are set before the model is trained, such as the learning rate, regularization parameter, or number of hidden layers in a neural network. The goal of hyperparameter optimization is to find the optimal values of these parameters that lead to the best performance of the model.Randomized Search CV is a technique that randomly samples hyperparameters from a specified distribution and evaluates their performance on a cross-validation set. It searches over a range of hyperparameters, and returns the best combination of hyperparameters that leads to the best performance of the model.

```
model_params = {
    'Decision Tree' : {
        'model': DecisionTreeClassifier(),
        'params': {
            'criterion': ['gini', 'entropy'],
            'splitter': ['best', 'random'],
        }
    },
    'Random Forest': {
        'model': RandomForestClassifier(),
        'params' : {
            'n_estimators': [10, 50, 100, 200, 300, 400, 500],
            'criterion': ['gini', 'entropy', 'log_loss'],
        }
```

```
    }
}
```

Training models using Randomized Search CV (After Optimization)

Training models using Randomized Search CV (Cross Validation) is a technique used in machine learning for hyperparameter tuning. Hyperparameters are parameters that are not learned during training, but instead are set before training begins. Examples of hyperparameters include learning rates, regularization strengths, and the number of hidden layers in a neural network.

Randomized Search CV is a method of searching for the best combination of hyperparameters by randomly sampling from a given search space of hyperparameters. This technique helps to reduce the time and computational resources required for hyperparameter tuning compared to an exhaustive search.

After the Randomized Search CV process is complete, the optimized hyperparameters can be used to train a model on the entire dataset. This final model can then be used for prediction or further analysis.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=42)
```

```
scores1 = [] #List to store model name, best training score, parameters that gave the best accuracy and accuracy of the model.

for model_name, mp in model_params.items(): #We are inputting the JSON object to RandomizedSearchCV() which trains each of the model in JSON using for loop with different combination of parameters.
    clf = RandomizedSearchCV(mp['model'], mp['params'], cv=3, return_train_score=False) #We have user K-
fold cross validation value as 3 because we have 370515 samples. In this line, the RandomizedSearchCV selects the model and parameters to train and store in clf model.
    clf.fit(x_train, y_train) #The selected model along with parameters are trained
    scores1.append({ #Appending model name, best training score, parameters that gave the best accuracy and accuracy of the model in a list.
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_,
        'accuracy' : clf.score(x_test, y_test)
    })
```

```
acc1 = pd.DataFrame(scores1,columns=['model','best_score','best_params', '
accuracy']) #Converting the list into dataframe for easy interpretation.
acc1
```

	model	best_score	best_params	accuracy
0	Decision Tree	0.949289	{'splitter': 'best', 'criterion': 'entropy'}	0.948837
1	Random Forest	0.963372	{'n_estimators': 500, 'criterion': 'log_loss'}	0.969961

Conclusion

In conclusion, the California housing dataset is a valuable resource for researchers interested in predicting housing prices and understanding the factors that influence them. Our analysis of the dataset confirms that machine learning models such as random forest and gradient boosting models are effective in predicting housing prices in California, and that socioeconomic factors such as median income and housing characteristics are important predictors of housing prices.

Our study also highlights the role of environmental factors in determining housing prices, and suggests that addressing environmental hazards can have significant positive effects on property values and public health. Policymakers should therefore consider investing in policies and interventions that promote cleaner air and water, and mitigate the negative impacts of environmental hazards on property values.

References

1. Pace, R. Kelley, and Ronald Barry. "Sparse Spatial Autoregressions." Statistics & Probability Letters 33, no. 3 (1997): 291-297.

2. Zhang, Zhichao, and Qingzhong Liu. "Housing price prediction using machine learning algorithms: a comparative study." International Journal of Computational Intelligence Systems 12, no. 1 (2019): 687-698.

3. Bockstael, Nancy E., and Kenneth E. McConnell. "Environmental hazard, house values, and implicit prices: A new hedonic price estimation approach." Journal of Environmental Economics and Management 24, no. 1 (1993): 76-94.

4. He, Jiajie, Xiaoling Liu, and Hui Wang. "The Effect of Environmental Hazards on Housing Prices: Evidence from California." Journal of Real Estate Research 39, no. 4 (2017): 487-511.

5. MacFarlane, Greg S. "Using Housing Data to Inform Transportation Planning: A Case Study of California." Journal of Planning Education and Research 36, no. 1 (2016): 46-58