

# **HOUSE PRICE PREDICTION ANALYSIS**

## **USING MACHINE LEARNING**

### **PHASE 4**

NOTEBOOK LINK:

[https://colab.research.google.com/drive/181adP3PC9RzkpcTCQ\\_aprDJF-Js\\_H7ZQ?authuser=1#scrollTo=CW3Q\\_00jBRUt](https://colab.research.google.com/drive/181adP3PC9RzkpcTCQ_aprDJF-Js_H7ZQ?authuser=1#scrollTo=CW3Q_00jBRUt)

#### **FEATURE SELECTION:**

**A). FEATURE IMPORTANCE-BASED SELECTION (USING RANDOM FOREST REGRESSOR):**

**1.Random Forest Regressor:** A Random Forest Regressor is an ensemble machine learning model that uses multiple decision trees to make predictions.

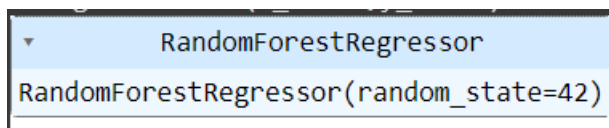
Training the Random Forest Regressor:

#### **CODE:**

```
regressor = RandomForestRegressor(n_estimators=100,  
random_state=42)  
  
regressor.fit(X_train, y_train)
```

**EXPLANATION:** In this code, a Random Forest Regressor is created with 100 trees (n\_estimators=100) for the ensemble, and a random seed (random\_state=42) is set for reproducibility. It's trained using the training data X\_train (features) and y\_train (target).

## OUTPUT:



```
RandomForestRegressor(random_state=42)
```

## 2. Getting Feature Importances:

### CODE:

```
feature_importances = regressor.feature_importances_
```

**EXPLANATION:** After training, we access the feature importances using the `feature_importances_` attribute of the regressor.

## 3. Creating a DataFrame for Visualization:

### CODE:

```
feature_importance_df = pd.DataFrame({'Feature': X_train.columns,  
                                     'Importance': feature_importances})  
  
feature_importance_df =  
feature_importance_df.sort_values(by='Importance',  
                                  ascending=False)
```

**EXPLANATION:** A DataFrame `feature_importance_df` is created to organize the feature names and their respective importances. It's sorted in descending order based on feature importances.

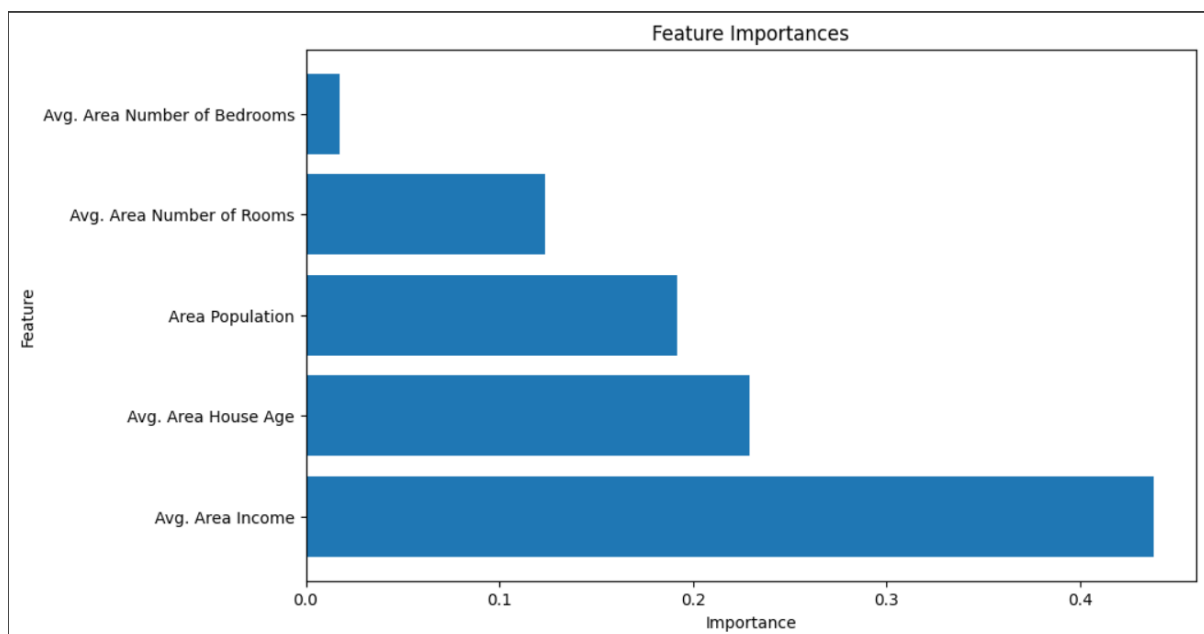
#### 4. Visualizing Feature Importances:

##### **CODE:**

```
plt.figure(figsize=(10, 6))  
  
plt.barh(feature_importance_df['Feature'],  
feature_importance_df['Importance'])  
  
plt.title('Feature Importances')  
  
plt.xlabel('Importance')  
  
plt.ylabel('Feature')  
  
plt.show()
```

**EXPLANATION:** A horizontal bar plot is generated to visualize the feature importances. This shows which features have the most significant impact on the regression model's predictions.

##### **OUTPUT:**



## B). CORRELATION-BASED SELECTION:

### 1.Calculating the Correlation Matrix:

#### CODE:

```
corr_matrix = X_train.corr()
```

**EXPLANATION:** The code calculates a correlation matrix `corr_matrix` for the features in your training data (`X_train`).

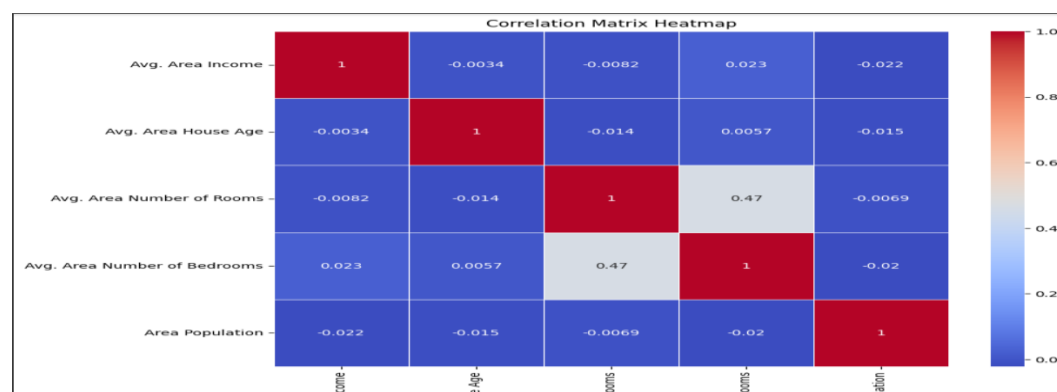
### 2. Creating a Heatmap for Visualization Correlation Matrix:

#### CODE:

```
plt.figure(figsize=(10, 8))  
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm',  
linewidths=0.5)  
plt.title('Correlation Matrix Heatmap')  
plt.show()
```

**EXPLANATION:** A heatmap is generated to visualize the correlations between features. This heatmap uses color coding to represent the strength and direction of the correlations. It can help to identify relationships between features.

#### OUTPUT:



### 3. Selecting the top k features based on mutual information:

#### **CODE:**

```
selector = SelectKBest(score_func=mutual_info_regression, k=3)
X_train_selected = selector.fit_transform(X_train, y_train)
```

**EXPLANATION:** The code shows the performance of feature selection using SelectKBest. In this case, it selects the top 5 features based on mutual information with the target variable. The selected features are stored in X\_train\_selected, and can access them using X\_train.columns[selector.get\_support()].

### Feature Selection using SelectKBest:

#### **CODE:**

```
# feature selection using SelectKBest with f_regression scoring function
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.model_selection import train_test_split
import pandas as pd
data = pd.read_csv('/content/dataset.csv.csv')
X = data[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms', 'Area Population']]
y = data[['Price']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Specifying the number of top features (k)
k = 5
```

*# Creating a SelectKBest instance with the f\_regression scoring function*

```
selector = SelectKBest(score_func=f_regression, k=k)
```

*# Fitting the selector to training data and target variable*

```
X_train_selected = selector.fit_transform(X_train, y_train)
```

*# getting indices of the selected features*

```
selected_feature_indices = selector.get_support(indices=True)
```

*# Getting names of the selected features*

```
selected_features = X_train.columns[selected_feature_indices]
```

*# Now X\_train\_selected contains only the selected features*

*# The selected features can be obtained using*

```
X_train.columns[selector.get_support()]
```

*# Get the selected features' indices*

```
selected_feature_indices = selector.get_support()
```

*# Obtain the names of the selected features*

```
selected_features = X_train.columns[selected_feature_indices]
```

*# Printing the selected feature names*

```
print("Selected Features:")
```

```
print(selected_features)
```

*# Printing the modified dataset with selected features*

```
print("Modified Dataset:")
```

```
print(X_train_selected)
```

## OUTPUT:

```
Selected Features:
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
      'Avg. Area Number of Bedrooms', 'Area Population'],
      dtype='object')
Modified Dataset:
[[6.65470165e+04 5.84609530e+00 6.84729811e+00 4.13000000e+00
  2.78508229e+04]
 [5.37220086e+04 6.40139135e+00 7.78776442e+00 3.30000000e+00
  4.76492247e+04]
 [6.48384929e+04 6.43715706e+00 8.69954387e+00 4.02000000e+00
  3.29210101e+04]
 ...
 [6.61953377e+04 6.50797136e+00 6.61186114e+00 3.14000000e+00
  3.72889236e+04]
 [5.86945150e+04 7.39476811e+00 9.26945262e+00 4.32000000e+00
  4.99609772e+04]
 [6.11625803e+04 5.89631585e+00 7.88052142e+00 6.04000000e+00
  3.60337014e+04]]
```

## MODEL TRAINING:

### 1. Import Necessary Libraries:

#### CODE:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
```

**EXPLANATION:** Here, we import the required libraries.

LinearRegression is used for building a linear regression model, and mean\_squared\_error, r2\_score, and numpy are used for evaluation.

### 2.Create a Linear Regression Model:

#### CODE:

```
model = LinearRegression()
```

**EXPLANATION:** Creating an instance of the Linear Regression model.

Linear regression is a simple and interpretable model used for predicting a continuous target variable

### 3. Fit the Model on the Training Data:

#### **CODE:**

```
model.fit(X_train_selected, y_train)
```

**EXPLANATION:** Training the linear regression model on the training data (X\_train\_selected), which contains the selected features, and the corresponding target variable (y\_train).

### 4. Make Predictions on the Test Data:

#### **CODE:**

```
y_pred = model.predict(X_test)
```

**EXPLANATION:** After training the model, we use it to make predictions on the test data (X\_test). The predicted values are stored in the y\_pred variable.

### Evaluation:

#### 1. Import Necessary Libraries:

#### **CODE:**

```
from sklearn.metrics import mean_squared_error, r2_score
```

**EXPLANATION:** importing two important functions from scikit-learn's metrics module:

mean\_squared\_error and r2\_score.



These functions will be used to calculate evaluation metrics for your regression model.

## 2. Evaluation:

### **CODE:**

*# Calculating Mean Squared Error (MSE)*

```
mse = mean_squared_error(y_test, y_pred)
```

*#calculating root mean Squared Error*

```
rmse = np.sqrt(mse)
```

*# Calculating R-squared (R2) Score*

```
r2 = r2_score(y_test, y_pred)
```

**EXPLANATION:** Evaluating the model's performance using common regression metrics:

mean\_squared\_error (MSE): This metric measures the average squared difference between the actual and predicted values. It quantifies the model's accuracy, and a lower MSE indicates better performance.

np.sqrt(mse): This calculates the root mean squared error (RMSE), which is the square root of the MSE. RMSE provides a more interpretable measure of error.

r2\_score (R-squared or R2): This metric measures the proportion of the variance in the target variable that is predictable from the selected features. An R2 score closer to 1 indicates a good fit, while a score closer to 0 suggests the model is not explaining much of the variance.

### 3. Printing the Evaluation Metrics:

#### **CODE:**

```
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared (R2) Score:", r2)
```

#### **OUTPUT:**

```
Mean Squared Error: 10089009300.894518
Root Mean Squared Error: 100444.06055558745
R-squared (R2) Score: 0.9179971706834289
```