California Housing Price Prediction

DESCRIPTION

Background of Problem Statement :

The US Census Bureau has published California Census Data which has 10 types of metrics such as the number of houses, population, income, median house prices for each block group in California. The dataset also serves as an ideal for project

Learning and how to specify the functional and nonfunctional requirements for it.

Problem Objective :

Our goal is to build a model of housing prices to predict median house values in California using the provided dataset.

provided dataset. This model should learn from the data and be able to predict the median housing price in any district,

given the other metrics, given all the other metrics.

Distinct or block groups are the smallest geographical units for which the US Census Bureau

publishes sample data (a block group typically has a population of 600 to 3,000 people). There are

district in the project dataset.

Objective: Finance and Housing

Analysis Tasks to be performed:

1. Build a model of housing prices to predict median house values in California using the provided dataset.

2. Train the model to learn from the data to predict the median housing price in any district, given all other metrics.

3. Predict housing prices based on median_income and plot the difference for test set.

1. Load the data

Read the "housing.csv" file from the folder into the program.

Print first few rows of this data.

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

```
# Import numpy and pandas
import numpy as np
import pandas as pd

# Read the housing.csv file into a dataset variable
housing = pd.read_csv("housing.csv")

# Print first few rows of this data.
housing.head()
```

```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  population  households  median_income  \
0    -122.23     37.88                41.0        880           129.0       322       126         8.3252
1    -122.22     37.86                21.0       7099          1106.0      2401      1138         8.3014
2    -122.24     37.85                52.0       1467           190.0       496       177         7.2574
3    -122.25     37.85                52.0       1274           235.0       558       219         5.6431
4    -122.25     37.85                52.0       1627           280.0       565       259         3.8462

   median_house_value ocean_proximity
0            452600        NEAR BAY
1            358500        NEAR BAY
2            352100        NEAR BAY
3            341300        NEAR BAY
4            342200        NEAR BAY
```

```
housing.shape
```

```
(20640, 10)
```

2. Handle Missing Values

```
# Count of null values
housing.isnull().sum()
```

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms      207
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64
```

```
housing.describe(include="object").T.sort_values(by="unique",ascending=False)
```

```
                count unique top freq
ocean_proximity 20640      5 <1H OK 9136
```

```
housing.describe(include="object").T.sort_values(by="freq",ascending=False)
```

```
                count unique top freq
ocean_proximity 20640      5 <1H OK 9136
```

```
# Fill the missing values with the mean of the respective column.
housing['total_bedrooms'].fillna(housing['total_bedrooms'].mean(), inplace=True)
```

```
# Count of null values
housing.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       0
dtype: int64
```

3. Encode Categorical data

Convert categorical column in the dataset to numerical data.

```
housing.groupby('ocean_proximity').sum()[['median_house_value']]
```

```
                 median_house_value
ocean_proximity
<1H OK                    9.136565e
INLAND                    5.551162e
ISLAND                    3.822400e
NEAR BAY                  1.660201e
NEAR OCEAN                1.735659e
```

```
housing.groupby('ocean_proximity').sum()[['median_house_value']].sort_values(by='median_house_value')
```

```
                 median_house_value
ocean_proximity
ISLAND                    3.822400e
INLAND                    5.551162e
NEAR BAY                  1.660201e
NEAR OCEAN                1.735659e
<1H OK                    9.136565e
```

```
housing.head()
```

```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  population  households  median_income  \
0    -122.23     37.88                41.0        880           129.0       322       126         8.3252
1    -122.22     37.86                21.0       7099          1106.0      2401      1138         8.3014
2    -122.24     37.85                52.0       1467           190.0       496       177         7.2574
3    -122.25     37.85                52.0       1274           235.0       558       219         5.6431
4    -122.25     37.85                52.0       1627           280.0       565       259         3.8462

   median_house_value ocean_proximity
0            452600        NEAR BAY
1            358500        NEAR BAY
2            352100        NEAR BAY
3            341300        NEAR BAY
4            342200        NEAR BAY
```

```
housing.corr()
```

```
                    longitude  latitude  housing_median_age  total_rooms  total_bedrooms  population  households  median_income  \
longitude            1.000000 -0.924664           -0.108197     0.044568        0.069608    0.099773    0.055848      -0.015176
latitude            -0.924664  1.000000            0.011173    -0.036100       -0.066983    -0.108785    -0.071035      -0.079809
housing_median_age  -0.108197  0.011173            1.000000    -0.361762       -0.320451    -0.296244    -0.302916      -0.119034
total_rooms          0.044568 -0.036100           -0.361762     1.000000        0.930380     0.857126     0.918484       0.198050
total_bedrooms       0.069608 -0.066983           -0.320451     0.930380        1.000000     0.877747     0.979728       -0.007723
population           0.099773 -0.108785           -0.296244     0.857126        0.877747     1.000000     0.907222       0.004834
households           0.055848 -0.071035           -0.302916     0.918484        0.979728     0.907222     1.000000       0.013033
median_income       -0.015176 -0.079809           -0.119034     0.198050       -0.007723     0.004834     0.013033       1.000000
median_house_value  -0.045967 -0.144160            0.105623     0.134153        0.049686    -0.024650     0.065843       0.688075

                    median_house_value
longitude                    -0.045967
latitude                     -0.144160
housing_median_age           0.105623
total_rooms                  0.134153
total_bedrooms               0.049686
population                   -0.024650
households                   0.065843
median_income                0.688075
median_house_value           1.000000
```

```
# Import OneHotEncoder from sklearn.preprocessing
from sklearn.preprocessing import OneHotEncoder

# Create a OneHotEncoder object
ohe = OneHotEncoder(sparse=False)

# Fit and transform the 'ocean_proximity' column
ohe_ocean_proximity = ohe.fit_transform(housing[['ocean_proximity']])

# Print the shape of the encoded data
print("Shape of ohe_ocean_proximity:", ohe_ocean_proximity.shape)

# Print the first few rows of the encoded data
print("First few rows of ohe_ocean_proximity:")
ohe_ocean_proximity[:5]
```

```
Shape of ohe_ocean_proximity: (20640, 5)
First few rows of ohe_ocean_proximity:
```

```
array([[0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.]])
```

```
# Create DataFrame from the encoded data
ohe_df = pd.DataFrame(ohe_ocean_proximity, columns=ohe.get_feature_names())

# Display the first few rows of the DataFrame
ohe_df.head()
```

```
   x0_<1H OK  x0_INLAND  x0_ISLAND  x0_NEAR BAY  x0_NEAR OCEAN
0        0.0       0.0        0.0          1.0            0.0
1        0.0       0.0        0.0          1.0            0.0
2        0.0       0.0        0.0          1.0            0.0
3        0.0       0.0        0.0          1.0            0.0
4        0.0       0.0        0.0          1.0            0.0
```

```
# Concatenate the original DataFrame and the encoded DataFrame
housing_encoded = pd.concat([housing, ohe_df], axis=1)

# Display the first few rows of the combined DataFrame
housing_encoded.head()
```

```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  population  households  median_income  \
0    -122.23     37.88                41.0        880           129.0       322       126         8.3252
1    -122.22     37.86                21.0       7099          1106.0      2401      1138         8.3014
2    -122.24     37.85                52.0       1467           190.0       496       177         7.2574
3    -122.25     37.85                52.0       1274           235.0       558       219         5.6431
4    -122.25     37.85                52.0       1627           280.0       565       259         3.8462

   median_house_value ocean_proximity  x0_<1H OK  x0_INLAND  x0_ISLAND  x0_NEAR BAY  x0_NEAR OCEAN
0            452600        NEAR BAY        0.0       0.0        0.0          1.0            0.0
1            358500        NEAR BAY        0.0       0.0        0.0          1.0            0.0
2            352100        NEAR BAY        0.0       0.0        0.0          1.0            0.0
3            341300        NEAR BAY        0.0       0.0        0.0          1.0            0.0
4            342200        NEAR BAY        0.0       0.0        0.0          1.0            0.0
```

```
# Drop the original 'ocean_proximity' column
housing_encoded = housing_encoded.drop('ocean_proximity', axis=1)

# Display the first few rows of the updated DataFrame
housing_encoded.head()
```

```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  population  households  median_income  \
0    -122.23     37.88                41.0        880           129.0       322       126         8.3252
1    -122.22     37.86                21.0       7099          1106.0      2401      1138         8.3014
2    -122.24     37.85                52.0       1467           190.0       496       177         7.2574
3    -122.25     37.85                52.0       1274           235.0       558       219         5.6431
4    -122.25     37.85                52.0       1627           280.0       565       259         3.8462

   median_house_value  x0_<1H OK  x0_INLAND  x0_ISLAND  x0_NEAR BAY  x0_NEAR OCEAN
0            452600        0.0       0.0        0.0          1.0            0.0
1            358500        0.0       0.0        0.0          1.0            0.0
2            352100        0.0       0.0        0.0          1.0            0.0
3            341300        0.0       0.0        0.0          1.0            0.0
4            342200        0.0       0.0        0.0          1.0            0.0
```

Extract input (X) and output (Y) data from the dataset

```
# Extracting input (X) and output (Y) data
X = housing[['median_income']]
Y = housing['median_house_value']
```

```
# Print the shapes of X and Y
print("Shape of X:", X.shape)
print("Shape of Y:", Y.shape)
```

```
Shape of X: (20640, 1)
Shape of Y: (20640,)
```

4. Split the dataset :

```
# Import the train_test_split function from sklearn.model_selection
from sklearn.model_selection import train_test_split

# Split the data into 70% training dataset and 30% test dataset.
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=42)
```

5. Standardize data :

Standardize training and test datasets.

```
# Import the StandardScaler function from sklearn.preprocessing
from sklearn.preprocessing import StandardScaler

# Create a StandardScaler object
scaler = StandardScaler()

# Fit and transform the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the test data using the same scaler
X_test_scaled = scaler.transform(X_test)

# Print the first few rows of the scaled training data
print("First few rows of X_train_scaled:")
X_train_scaled[:5]
```

```
First few rows of X_train_scaled:
```

```
array([[-0.02430414],
       [ 1.348669  ],
       [ 0.049319  ],
       [ 0.728435  ],
       [-0.704705  ]])
```

```
# Print the first few rows of the scaled test data
print("First few rows of X_test_scaled:")
X_test_scaled[:5]
```

```
First few rows of X_test_scaled:
```

```
array([[-0.02430414],
       [ 1.348669  ],
       [ 0.049319  ],
       [ 0.728435  ],
       [-0.704705  ]])
```

6. Perform Linear Regression :

Perform Linear Regression on training data.

```
# Import the LinearRegression class from sklearn.linear_model
from sklearn.linear_model import LinearRegression

# Create a Linear Regression model object
model = Linear_regression()

# Fit the model to the training data
model.fit(X_train_scaled, Y_train)
```

```
LinearRegression()
```

Predict output for test dataset using the fitted model.

```
# Predict the target variable for the test data
Y_pred = model.predict(X_test_scaled)

# Print the predicted values
print("Predicted values:")
Y_pred[:5]
```

```
Predicted values:
```

```
array([[-0.02430414],
       [ 1.348669  ],
       [ 0.049319  ],
       [ 0.728435  ],
       [-0.704705  ]])
```

Print mean squared error (MSE) from Linear Regression. (MSE? Import mean_squared_error from sklearn.metrics)

```
# Import the mean_squared_error function from sklearn.metrics
from sklearn.metrics import mean_squared_error

# Calculate the mean squared error
mse = mean_squared_error(Y_test, Y_pred)

# Print the mean squared error
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.5558915986952442
```

Extract the median_income column from the independent variables (Base X) train and X_test.

```
# Extract the median_income column from X_train
X_train_median_income = X_train[['median_income']]

# Extract the median_income column from X_test
X_test_median_income = X_test[['median_income']]

# Print the first few rows of the extracted data
print("First few rows of X_train_median_income:")
X_train_median_income[:5]
```

```
First few rows of X_train_median_income:
```

```
array([[-0.02430414],
       [ 1.348669  ],
       [ 0.049319  ],
       [ 0.728435  ],
       [-0.704705  ]])
```

7. Bonus exercise: Perform Linear Regression with one independent variable

Extract the median_income column from the independent variables (Base X) train and X_test.

```
# Create a LinearRegression model object
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train_median_income, Y_train)
```

```
LinearRegression()
```

Perform Linear Regression to predict housing values based on median_income.

```
# Predict the target variable for the test data
Y_pred = model.predict(X_test_median_income)

# Print the predicted values
print("Predicted values:")
Y_pred[:5]
```

```
Predicted values:
```

```
array([[-0.02430414],
       [ 1.348669  ],
       [ 0.049319  ],
       [ 0.728435  ],
       [-0.704705  ]])
```

Predict output for test dataset using the fitted model.

```
# Calculate the mean squared error
mse = mean_squared_error(Y_test, Y_pred)

# Print the mean squared error
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.5558915986952442
```

Print the fitted model for training data as well as for test data to check if the fitted model satisfies the test data.

```
# Predicted values for training data
Y_train_pred = model.predict(X_train_median_income)

# Plot the scatter plot for training data
plt.scatter(X_train_median_income, Y_train, alpha=0.5)
plt.plot(X_train_median_income, Y_train_pred, color='red')
plt.xlabel('Median Income')
plt.ylabel('Median House Value')
plt.title('Linear Regression - Training Data')
plt.show()
```

Perform Linear Regression to predict housing values based on median_income.

```
# Predicted values for test data
Y_test_pred = model.predict(X_test_median_income)

# Plot the scatter plot for test data
plt.scatter(X_test_median_income, Y_test, alpha=0.5)
plt.plot(X_test_median_income, Y_test_pred, color='red')
plt.xlabel('Median Income')
plt.ylabel('Median House Value')
plt.title('Linear Regression - Test Data')
plt.show
```