COMP-5011

Assignment 1
Linear Regression
Report

Amirhossein Zeinali Dehaghani
azeinali@lakeheadu.ca
Student ID: 1225496

Fall 2024

Problem Statement

There is a data set that includes different features of houses and their prices. We are asked for training our model to predict prices for new houses using Linear Regression. The target variable is the price, and the given features are as follows:

- Area (in square feet)
- Number of Bedrooms
- Hot Water Heating
- Air Conditioning
- Number of Bathrooms

Linear Regression

Linear Regression aims to model the linear relationship between two variables (an independent variable and a dependent variable) in order to predict new inputs.

In this case: Price is the dependent variable and given features are independent variable.

Variable Type	Variable Name	Description
Dependent	Price	The price of the house (we want to predict)
Independent	Area	The size of the house measured in square feet
Independent	Number of Bedrooms	The total number of bedrooms in the house
Independent	Hot Water Heating	Indicator variable for hot water heating (Yes/No)
Independent	Air Conditioning	Indicator variable for air conditioning (Yes/No)
Independent	Number of Bathrooms	The total number of bathrooms in the house

Table 1: Variables in the House Price Prediction Model

Phase No.1: (Data Preprocessing)

1- Load the data set and extract features that we have been asked for:

```
# Load the dataset:
df = pd.read_csv("/content/drive/MyDrive/COMP-5011-Assignment1-
   LinearRegression/Housing.csv")

# Extract features that should be Consider according to Assignment Tasks:
required_features = ["area", "bedrooms", "hotwaterheating", "airconditioning",
    "bathrooms","price"]
df = df[required_features]
df.head(4)
```

output:

```
    area   bedrooms hotwaterheating     airconditioning     bathrooms        price
0   7420      4            no                 yes               2          13300000
1   8960      4            no                 yes               4          12250000
2   9960      3            no                  no               2          12250000
3   7500      4            no                 yes               2          12215000
```

2- Perform basic exploration and describe the data:

- df.shape

```
    # Size of Data set
    df.shape
```

output:

```
    (545, 6)
```

It means the total entries and columns are 545 and 6 respectively.

- df.info

```
    # Summarize the structure of loaded data set
    df.info()
```

output:

```
    <class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   area             545 non-null    int64
 1   bedrooms         545 non-null    int64
 2   hotwaterheating  545 non-null    object
 3   airconditioning  545 non-null    object
 4   bathrooms        545 non-null    int64
 5   price            545 non-null    int64
dtypes: int64(4), object(2)
memory usage: 25.7+ KB
```

Highlighted information:

- Total Entries: 545
- Columns: 6
- Data Types:

Variable	Data Type
area	int64
bedrooms	int64
Hot Water Heating	Object
air Conditioning	Object
bathrooms	int64
price	int64

Table 2: Data Types of Variables

- Describe() : Quantitative summary of the numerical columns in data set.

```
    # quantitative summary of the numerical columns in data set
    df.describe()
```

output:

```
              area         bedrooms      bathrooms       price
count    545.000000      545.000000     545.000000    5.450000e+02
mean     5150.541284     2.965138       1.286239      4.766729e+06
std      2170.141023      0.738064      0.502470      1.870440e+06
min      1650.000000      1.000000      1.000000      1.750000e+06
25%      3600.000000      2.000000      1.000000      3.430000e+06
50%      4600.000000      3.000000      1.000000      4.340000e+06
75%      6360.000000      3.000000      2.000000      5.740000e+06
max      16200.000000     6.000000      4.000000      1.330000e+07
```

- isnull() : Check for missing values

```
    #Check for missing values
    print("Missing values in each column:", df.isnull().sum())
```

output:

```
Missing values in each column: area                 0
bedrooms             0
hotwaterheating      0
airconditioning      0
bathrooms            0
price                0
dtype: int64
```

According to the result, there is not any missing values in data set

3- Pre-processing:

- As it was noticed from df.info(), there are 2 categorical features (hot water heating and air conditioning) that we should convert them to binary. (0= No, 1= Yes).

  Why:
  Algorithms such as linear regression cannot operate on categorical data directly. They need numerical representations to calculate distances, slopes, and other mathematical operations

```
  # Convert categorical features (hotwaterheating and airconditioning) to
     binary. (0= No, 1= Yes)

  df["hotwaterheating"] = [0 if i=="no" else 1 for i in df["hotwaterheating"]]
  df["airconditioning"] = [0 if i=="no" else 1 for i in df["airconditioning"]]
  df.head()
```

output:

```
    area     bedrooms  hotwaterheating   airconditioning   bathrooms     price
0   7420        4            0                  1              2        13300000
1   8960        4            0                  1              4        12250000
2   9960        3            0                  0              2        12250000
3   7500        4            0                  1              2        12215000
4   7420        4            0                  1              1        11410000
```

4- Scale:

In preparing the data set for linear regression, StandardScaler is applied to standardize feature values.

- Why:
  - Standardizing features prevents variables with larger scales from dominating the model, ensuring that all features contribute equally to the outcome.
  - Scaling enhances the performance of linear regression, leading to more efficient model training.
- By using the StandardScaler: Features are transformed such that each has a mean of 0 and a standard deviation of 1.

  Meaning of current features Values:

  - Positive Values: value is above the mean for that feature.
  - Negative Values: value is below the mean for that feature.
  - Around: value is near the mean for that feature.

```
# Select features to scale (excluding the target variable 'price')
features_to_scale = df[['area', 'bedrooms', 'hotwaterheating', 'airconditioning', 'bathrooms']]

# Fit and transform the selected features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features_to_scale)

# Convert scaled features back to a DataFrame and Add the target variable back (price) to the scaled DataFrame
scaled_df = pd.DataFrame(scaled_features, columns=['area', 'bedrooms', 'hotwaterheating', 'airconditioning', 'bathrooms'])
scaled_df['price'] = df['price'].values

scaled_df.head()
```

output:

```
    area     bedrooms hotwaterheating  airconditioning  bathrooms   price
0   1.046726 1.403419  -0.219265          1.472618       1.421812   13300000
1   1.757010 1.403419  -0.219265          1.472618       5.405809   12250000
2   2.218232 0.047278  -0.219265         -0.679063       1.421812   12250000
3   1.083624 1.403419  -0.219265          1.472618       1.421812   12215000
4   1.046726 1.403419  -0.219265          1.472618      -0.570187   11410000
```

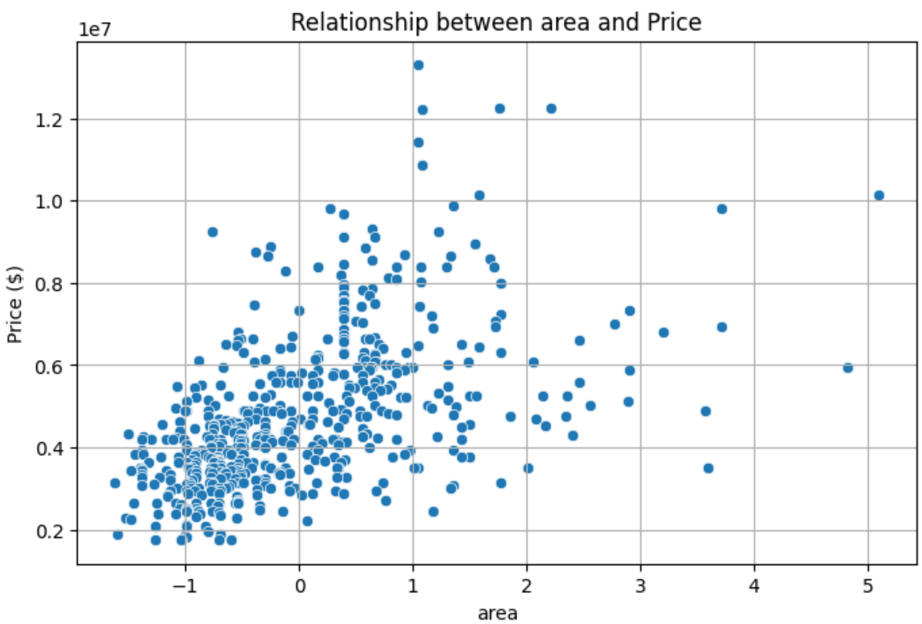
Phase No.2: (Exploratory Data Analysis)

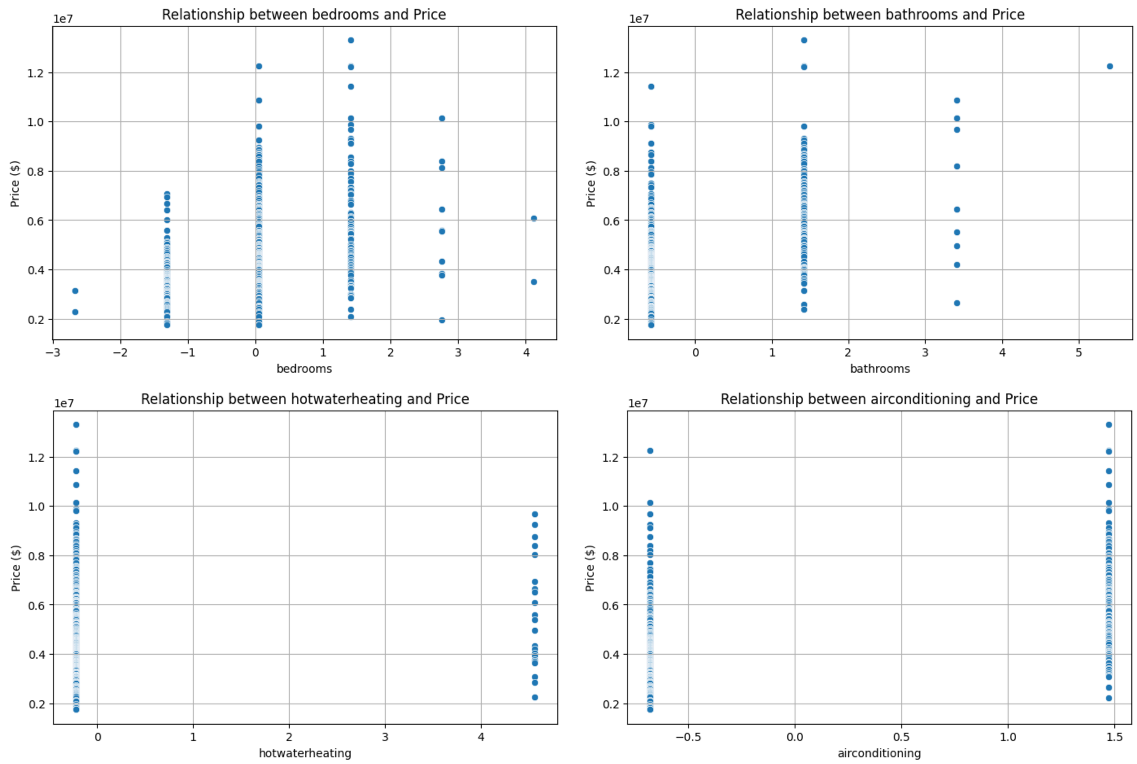
Visualization: Show the relationships between each feature and the target variable(price). We use both scatter plot and heat map.

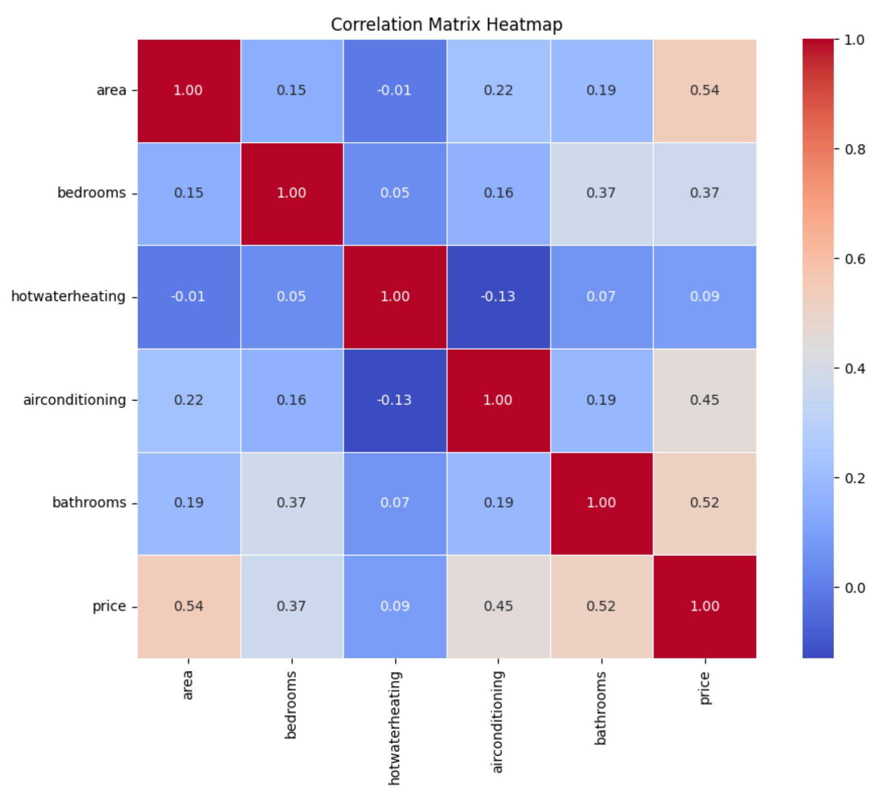
- Why scatter plot:
  It can help visually show relationships between dependent and independent variables.
- Why heatmap:
  It condenses the complexity of data through color representation.

```
# Extract features and the target variable
    features = scaled_df.drop('price', axis=1)
    target = scaled_df['price']

# Loop through each feature and create a scatter plot
    for feature in features.columns:
        plt.figure(figsize=(8, 5))
        sns.scatterplot(x=features[feature], y=target)
        plt.title(f'Relationship between {feature} and Price')
        plt.xlabel(feature)
        plt.ylabel('Price ($)')
        plt.grid(True)
        plt.show()


# Compute the correlation matrix
    correlation_matrix = scaled_df.corr()
    plt.figure(figsize=(12, 8))
    sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', square
        =True, linewidths=0.5)
    plt.title('Correlation Matrix Heatmap')
    plt.show()
```

output:




Relationship between bedrooms and Price
Relationship between bathrooms and Price
Relationship between hotwaterheating and Price
Relationship between airconditioning and Price
1e7
Price ($)
bedrooms
bathrooms
hotwaterheating
airconditioning


Correlation Matrix Heatmap
area
bedrooms
hotwaterheating
airconditioning
bathrooms
price
1.00 0.15 -0.01 0.22 0.19 0.54
0.15 1.00 0.05 0.16 0.37 0.37
-0.01 0.05 1.00 -0.13 0.07 0.09
0.22 0.16 -0.13 1.00 0.19 0.45
0.19 0.37 0.07 0.19 1.00 0.52
0.54 0.37 0.09 0.45 0.52 1.00

Phase No.3: (Implement Linear Regression)

1- Split the data set into training and test sets (80% train, 20% test).

```
# Features (independent variables)
    X = scaled_df.drop('price', axis=1)
# Target variable (dependent variable (price))
    y = scaled_df['price']

# Split the data into training and test sets (80% train, 20% test)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
        random_state=42)

    #Shapes of the Train and Test datasets
    print(f'Train set size: {X_train.shape}')
    print(f'Test set size: {X_test.shape}')
```

output:

```
Train set size: (436, 5)
Test set size: (109, 5)
```

2- Implement Linear Regression from scratch using NumPy, calculating the parameters via the Normal Equation and fit the model on the training data and predict the house prices on the test data.

```
# Add intercept
    X_b_train = np.c_[np.ones((X_train.shape[0], 1)), X_train]
# Add intercept for test set
    X_b_test = np.c_[np.ones((X_test.shape[0], 1)), X_test]

#Calculate parameters using Normal Equation
    theta_best = np.linalg.inv(X_b_train.T.dot(X_b_train)).dot(X_b_train.T).dot(
        y_train)

#Extract intercept (b) and slopes (m)
    b = theta_best[0]  # Intercept
    m = theta_best[1:]  # Slopes for each feature

#Function to predict values
    def predict(X, b, m):
          X_b = np.c_[np.ones((X.shape[0], 1)), X]  # Add intercept
         return X_b.dot(np.r_[b, m])  # Concatenate b with m for predictions

#Make predictions on the test dataset
    predictions = predict(X_test, b, m)

#Showing predicted value (price), Actual value (price) and difference of them
    for i in range(len(predictions)):
        print(f"
        Predicted Price: {int(predictions[i])}    |
        Actual Price: {y_test.values[i]}    |
        difference:{int(predictions[i])-y_test.values[i]}
        ")
```

Few rows of output:

```
Predicted Price: 5741068    |    Actual Price: 4060000    |     difference:1681068
Predicted Price: 6819096    |    Actual Price: 6650000    |     difference:169096
Predicted Price: 3378466    |    Actual Price: 3710000    |     difference:-331534
Predicted Price: 5100092    |    Actual Price: 6440000    |     difference:-1339908
Predicted Price: 3620897    |    Actual Price: 2800000    |     difference:820897
Predicted Price: 4463414    |    Actual Price: 4900000    |     difference:-436586
Predicted Price: 6174606    |    Actual Price: 5250000    |     difference:924606
Predicted Price: 5463282    |    Actual Price: 4543000    |     difference:920282
Predicted Price: 3134258    |    Actual Price: 2450000    |     difference:684258
Predicted Price: 3236270    |    Actual Price: 3353000    |     difference:-116730
Predicted Price: 8982004    |    Actual Price: 10150000    |     difference:-1167996
Predicted Price: 3346108    |    Actual Price: 2660000    |     difference:686108
Predicted Price: 4442508    |    Actual Price: 3360000    |     difference:1082508
Predicted Price: 3358374    |    Actual Price: 3360000    |     difference:-1626
Predicted Price: 3623950    |    Actual Price: 2275000    |     difference:1348950
Predicted Price: 6042391    |    Actual Price: 2660000    |     difference:3382391
Predicted Price: 2755736    |    Actual Price: 2660000    |     difference:95736
Predicted Price: 5405352    |    Actual Price: 7350000    |     difference:-1944648
Predicted Price: 4197838    |    Actual Price: 2940000    |     difference:1257838
Predicted Price: 3976775    |    Actual Price: 2870000    |     difference:1106775
Predicted Price: 5106197    |    Actual Price: 6720000    |     difference:-1613803
Predicted Price: 5657191    |    Actual Price: 5425000    |     difference:232191
Predicted Price: 3259109    |    Actual Price: 1890000    |     difference:1369109
Predicted Price: 3572056    |    Actual Price: 5250000    |     difference:-1677944
Predicted Price: 4930461    |    Actual Price: 4193000    |     difference:737461
Predicted Price: 6713570    |    Actual Price: 12250000    |     difference:-5536430
Predicted Price: 3088469    |    Actual Price: 3080000    |     difference:8469
Predicted Price: 4588963    |    Actual Price: 5110000    |     difference:-521037
Predicted Price: 7603221    |    Actual Price: 9800000    |     difference:-2196779
Predicted Price: 3060996    |    Actual Price: 2520000    |     difference:540996
Predicted Price: 6055947    |    Actual Price: 6790000    |     difference:-734053
Predicted Price: 3364479    |    Actual Price: 3500000    |     difference:-135521
Predicted Price: 6666467    |    Actual Price: 6650000    |     difference:16467
Predicted Price: 4431362    |    Actual Price: 2940000    |     difference:1491362
Predicted Price: 4136536    |    Actual Price: 3325000    |     difference:811536
Predicted Price: 6239103    |    Actual Price: 4200000    |     difference:2039103
Predicted Price: 3830305    |    Actual Price: 4900000    |     difference:-1069695
Predicted Price: 4811372    |    Actual Price: 3290000    |     difference:1521372
Predicted Price: 4636098    |    Actual Price: 3500000    |     difference:1136098
Predicted Price: 4790795    |    Actual Price: 2380000    |     difference:2410795
Predicted Price: 4848041    |    Actual Price: 5495000    |     difference:-646959
Predicted Price: 4415034    |    Actual Price: 3675000    |     difference:740034
Predicted Price: 6794676    |    Actual Price: 6650000    |     difference:144676
Predicted Price: 3694160    |    Actual Price: 4907000    |     difference:-1212840
Predicted Price: 4205719    |    Actual Price: 3150000    |     difference:1055719
Predicted Price: 5103395    |    Actual Price: 4480000    |     difference:623395
Predicted Price: 6666467    |    Actual Price: 6580000    |     difference:86467
Predicted Price: 4095326    |    Actual Price: 5740000    |     difference:-1644674
Predicted Price: 4602557    |    Actual Price: 3003000    |     difference:1599557
Predicted Price: 3060996    |    Actual Price: 1820000    |     difference:1240996
Predicted Price: 7116725    |    Actual Price: 8400000    |     difference:-1283275
Predicted Price: 3060996    |    Actual Price: 2450000    |     difference:610996
Predicted Price: 4517084    |    Actual Price: 4270000    |     difference:247084
Predicted Price: 4947462    |    Actual Price: 4007500    |     difference:939962
Predicted Price: 3790067    |    Actual Price: 3234000    |     difference:556067
Predicted Price: 3300375    |    Actual Price: 1750000    |     difference:1550375
Predicted Price: 6590152    |    Actual Price: 9800000    |     difference:-3209848
```

Phase No.4: (Evaluate Model Performance)

1- Evaluate the performance of the model using Mean Squared Error(MSE), Mean Absolute Error(MAE), and R2 Score.

```

# Evaluate the model using \MSE, \MAE, and \R2 Score
    \mse = mean_squared_error(y_test, predictions)
    mae = mean_absolute_error(y_test, predictions)
    r2 = r2_score(y_test, predictions)

# Print the evaluation metrics
    print("\MSE:", \mse.__round__(2))
    print("\MAE:", mae.__round__(2))
    print("R2 Score:", r2.__round__(2))
```

output:

```
MSE: 2346728742911.69
MAE: 1156665.64
R2 Score: 0.54
```

2- Compare the predicted values with actual values using a scatter plot (predicted vs. actual).

```

#Compare the predicted values with actual values using a scatter plot (predicted
    vs.actual)
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, predictions, color='blue', alpha=0.6)  # Actual vs
        Predicted
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='
        red', linewidth=2)  # Diagonal line
    plt.title('Actual Prices vs Predicted Prices')
    plt.xlabel('Actual Prices')
    plt.ylabel('Predicted Prices')
    plt.grid(True)
    plt.xlim([0, max(y_test.max(), predictions.max())])  # Set x and y limits
    plt.ylim([0, max(y_test.max(), predictions.max())])
    plt.gca().set_aspect('equal', adjustable='box')  # Equal aspect ratio
    plt.show()
```

output:

3- Discuss how well the model performs based on these evaluation metrics.

Given the evaluation results of linear regression model:

- MSE:
  - The MSE is extremely large, which is typical when dealing with large numeric scales (like house prices) since the error is squared.
  - A very high MSE indicates that there are large discrepancies between the predicted values and the actual values.
- MAE:

  The MAE indicates that, on average, the model's predictions are off by about $1156665. This is quite a substantial error if the target is house prices, implying that model's predictions are not very close to actual values.

- R^2 Score:

  An R2 score of 0.54 means that approximately 54% of the variance in house prices is explained by the model. This suggests that the model has captured some relationships between the features and the target variable, but a significant portion of variance is still unexplained.

******************** END OF ASSIGNMENT 1 ********************