Linear Regression Model To Predict Housing Prices

```
import pandas as pd

#Pandas is a Python library used for data manipulation, analysis,and preprocessing.
#Key Benefits:
#1.Easy Data Handling: Read/write CSV, Excel, SQL, JSON.
#2.Efficient Processing: Fast operations on large datasets.
#3.Data Cleaning: Handle missing values, duplicates.
#4.Flexible Transformation: Filtering, grouping, merging, and reshaping.
#5. Integration: Works well with NumPy, Matplotlib, and Scikit-learn for ML.
```

```
import numpy as np
#NumPy(Numerical Python) is a fundamental library for Python numerical computing.
#Features of Numpy:
#1. N-Dimensional Arrays
#2. Arrays with High Performance: Arrays are stored in contiguous memory locations, enabling faster computations than Python lists
```

```
# Let's consider one example for Basic Numpy operation
import numpy as np

x = np.array([1, 2, 3])
y = np.array([4, 5, 6])

# Addition
add = x + y
print("Addition:",add)

# Subtraction
subtract = x - y
print("substration:",subtract)

# Multiplication
multiply = x * y
print("multiplication:",multiply)

# Division
divide = x / y
print("division:", divide)
```

```
Addition: [5 7 9]
substration: [-3 -3 -3]
multiplication: [ 4 10 18]
division: [0.25 0.4  0.5 ]
```

```
#Pandas provides powerful data structures like DataFrame (tabular data) and Series (1D array).

#pandas Series
# --- Pandas Series can be created from lists, dictionaries, scalar values, etc.
import pandas as pd
import numpy as np
# Creating empty series
ser = pd.Series()
print("Pandas Series: ", ser)

# simple array
data = np.array(['g', 'e', 'e', 'k', 's'])

ser = pd.Series(data)
print("Pandas Series:\n", ser)
```

```
#Dataframe
# Creating a DataFrame Using the Pandas Library
import pandas as pd
# Calling DataFrame constructor
df = pd.DataFrame()
print(df)
# list of strings
lst = ['Geeks', 'For', 'Geeks', 'is', 'portal', 'for', 'Geeks']
# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
print(df)
```

```
Pandas Series:  Series([], dtype: object)
Pandas Series:
 0    g
1    e
2    e
3    k
4    s
dtype: object
Empty DataFrame
Columns: []
Index: []
        0
0   Geeks
1     For
2   Geeks
3      is
4  portal
5     for
6   Geeks
```

```
import statsmodels.api as sm  # Used for performing linear regression (OLS- ordinary least square)
```

```
from sklearn.preprocessing import LabelBinarizer
```

```
# Converts binary categorical values (Yes/No, Male/Female) into 0 and 1.
```

```
from sklearn.model_selection import train_test_split

# Splits dataset into training and testing sets to avoid overfitting.
```

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

#Evaluation Metrics
#R² Score → Measures how well the model explains variance (1 = perfect, 0 = random).
#Mean Absolute Error → Measures average absolute difference between predicted & actual values.
#Mean Squared Error → Penalizes large errors more than MAE (squared differences).
```

Double-click (or enter) to edit

```
#step-1 upload dataset
```

```
# we read the dataset (Housing_Modified.csv) into a pandas DataFrame.

df = pd.read_csv("/content/Housing_Modified.xls")  # Read CSV file into a DataFrame

print(df.info())  # Shows dataset structure: column names, data types, missing values
df.head()  # Displays the first 5 rows

#Understanding df.info()
#This function provides:
#1. Column names
#2. Data types (int, float, object)
#3. Missing values (if any)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 546 entries, 0 to 545
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   price     546 non-null    float64
 1   lotsize   546 non-null    int64
 2   bedrooms  546 non-null    int64
 3   bathrms   546 non-null    int64
 4   stories   546 non-null    object
 5   driveway  546 non-null    object
 6   recroom   546 non-null    object
 7   fullbase  546 non-null    object
 8   gashw     546 non-null    object
 9   airco     546 non-null    object
 10  garagepl  546 non-null    int64
 11  prefarea  546 non-null    object
dtypes: float64(1), int64(4), object(7)
memory usage: 51.3+ KB
None
```

	price	lotsize	bedrooms	bathrms	stories	driveway	recroom	fullbase	gashw	airco	garagepl	prefarea
0	42000.0	5850	3	1	two	yes	no	yes	no	no	1	no
1	38500.0	4000	2	1	one	yes	no	no	no	no	0	no
2	49500.0	3060	3	1	one	yes	no	no	no	no	0	no
3	60500.0	6650	3	1	two	yes	yes	no	no	no	0	no
4	61000.0	6360	2	1	one	yes	no	no	no	no	0	no

```
df.shape
```

```
(546, 12)
```

```
# Step 2: Data Cleaning and Conversion
# Convert categorical columns ('yes'/'no') into binary values (1/0)
binary_cols = ['driveway', 'recroom', 'fullbase', 'gashw', 'airco', 'prefarea']
for col in binary_cols:
    lb = LabelBinarizer()
    df[col] = lb.fit_transform(df[col])  # 'Yes' -> 1, 'No' -> 0

# Convert 'stories' into dummy variables (one-hot encoding)
df = pd.get_dummies(df, columns=['stories'], drop_first=True)
# drop_first=True avoids dummy variable trap
```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
# Step 3: Feature Selection - Checking Multicollinearity using VIF
# Multicollinearity occurs when independent variables are highly correlated.
# High multicollinearity can make it difficult to interpret the model.
# Variance Inflation Factor (VIF) is used to identify multicollinear variables.
# A VIF value greater than 10 suggests a high correlation with other independent variables.
# We remove features with high VIF to improve model performance.

X = df.drop(columns=['price'])
# Explicitly convert all columns to numeric, replacing non-numeric with NaN
X = X.apply(pd.to_numeric, errors='coerce')

vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns

# Drop missing values
X = X.fillna(X.mean())  # Remove rows with NaN
# OR replace NaN with the column's mean
# X = X.fillna(X.mean())
# Convert all columns to float type before calculating VIF
X = X.astype(float)

# Compute VIF after cleaning
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Display VIF values
print("Variance Inflation Factor (VIF) Table:\n", vif_data)
```

```
Variance Inflation Factor (VIF) Table:
           Feature        VIF
0          lotsize   8.958098
```

```
1        bedrooms  18.469879
2         bathrms   8.984672
3        driveway   7.088579
4         recroom   1.477015
5        fullbase   2.013320
6           gashw   1.103488
7           airco   1.756746
8        garagepl   1.982649
9        prefarea   1.533295
10    stories_one   3.965753
11  stories_three   1.770040
12    stories_two   5.511702
```

```
# Step 4: Remove features with high multicollinearity (VIF > 10)
if vif_data.loc[vif_data["VIF"] > 10, "Feature"].any():
    df = df.drop(columns=['bedrooms'])  # Dropping 'bedrooms' due to high multicollinearity


# Step 5: Splitting the Data into Training and Testing sets
X = df.drop(columns=['price'])  # Independent variables
y = df['price']  # Target variable (house price)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# 80% training, 20% testing


# Step 6: Building the Regression Model using OLS (Ordinary Least Squares)
X_train = sm.add_constant(X_train)  # Adding a constant term for the regression model
X_test = sm.add_constant(X_test)  # Adding constant term to test data

# Convert all columns in X_train and X_test to numeric, replacing non-numeric with NaN
X_train = X_train.apply(pd.to_numeric, errors='coerce')
X_test = X_test.apply(pd.to_numeric, errors='coerce')

# Replace NaN values with the column mean
X_train = X_train.fillna(X_train.mean())
X_test = X_test.fillna(X_test.mean())

# ----> Explicitly convert all columns to float
for col in X_train.columns:
    X_train[col] = X_train[col].astype(float)
for col in X_test.columns:
    X_test[col] = X_test[col].astype(float)

# Fit the model after data type conversion
lm = sm.OLS(y_train, X_train).fit()  # Train the model


# Step 7: Model Evaluation
y_pred = lm.predict(X_test)  # Predict house prices for test data
r2 = r2_score(y_test, y_pred)  # R² Score (Goodness of Fit)
mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error

# Display model performance
print("\nModel Performance:")
print(f"R² Score: {r2:.4f}")
print(f"Mean Absolute Error: {mae:.2f}")
print(f"Mean Squared Error: {mse:.2f}")
```

```
Model Performance:
R² Score: 0.6062
Mean Absolute Error: 11548.17
Mean Squared Error: 263075151.21
```

```
# Step 8: Predicting House Price using User Input
def predict_price():
    # Take user input for house features
    lotsize = int(input("Enter Lot Size: "))
    bathrms = int(input("Enter Bathrooms: "))
    driveway = int(input("Enter Driveway (1 for Yes, 0 for No): "))
    recroom = int(input("Enter Rec Room (1 for Yes, 0 for No): "))
    fullbase = int(input("Enter Full Basement (1 for Yes, 0 for No): "))
    gashw = int(input("Enter Gas Hot Water (1 for Yes, 0 for No): "))
    airco = int(input("Enter Air Conditioning (1 for Yes, 0 for No): "))
    garagepl = int(input("Enter Garage Places: "))
    prefarea = int(input("Enter Preferred Area (1 for Yes, 0 for No): "))
    story_one = int(input("Story One (1 for Yes, 0 for No): "))
    story_three = int(input("Story Three (1 for Yes, 0 for No): "))
    story_two = int(input("Story Two (1 for Yes, 0 for No): "))

    # Create a DataFrame for user input
    input_data = pd.DataFrame([[1, lotsize, bathrms, driveway, recroom, fullbase, gashw, airco, garagepl,
```

```
                              prefarea, story_one, story_three, story_two]],
                            columns=X_train.columns)

    # Predict house price using the trained model
    predicted_price = lm.predict(input_data)[0]
    print(f"\nPredicted House Price: ${predicted_price:.2f}")

# Call function to predict house price
predict_price()
```

```
Enter Lot Size: 100
Enter Bathrooms: 4
Enter Driveway (1 for Yes, 0 for No): 1
Enter Rec Room (1 for Yes, 0 for No): 1
Enter Full Basement (1 for Yes, 0 for No): 1
Enter Gas Hot Water (1 for Yes, 0 for No): 1
Enter Air Conditioning (1 for Yes, 0 for No): 1
Enter Garage Places: 1
Enter Preferred Area (1 for Yes, 0 for No): 1
Story One (1 for Yes, 0 for No): 1
Story Three (1 for Yes, 0 for No): 1
Story Two (1 for Yes, 0 for No): 1

Predicted House Price: $94180.93
```