```
# %%
#Impoting libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import os


# %%
#reading_inputs
files= [file for file in os.listdir(r'C:\Users\amrah\Downloads\house_price_prediction')]
print(files)
os.chdir(r'C:\Users\amrah\Downloads\house_price_prediction')
df=pd.read_csv('house_price_2nd_data.csv')


# %%
#check there is no null value in the data
print(df.isnull().sum())


# %%
#name of the columns
print(df.columns)
df.head()


# %%
#chaning columns header if there is blanks, and convrting the header names to lowercase
```

```
df.columns=df.columns.str.strip().str.lower().str.replace(" ","_")


# %%
#changing numeric_columns to float
numeric_cols = ['price', 'sqft_living', 'sqft_lot', 'sqft_above',
            'sqft_basement']
for col in numeric_cols:
  df[col]=df[col].astype(float)


# %%
#based on year calculating age of house
df['age_of_house']=2025-df['yr_built']


# %%
#if house was renovated then what is the age age after renovation , otherwise taking the age of  the house as final age of the house
df['final_age']=0
for i in range(len('age_of_house')):
  if (df['yr_renovated'][i]==0):
    df['final_age'][i]=df['age_of_house'][i]
  else:
    df['final_age'][i]=2025-df['yr_renovated'][i]
df['is_renovated']=0
df['is_renovated']=df['yr_renovated'].apply(lambda x:1 if x!=0 else 0)


# %%
#checking Response variable(#Y variable i.e price)
import matplotlib.pyplot as plt
```

```
import seaborn as sns

plt.figure(figsize=(8, 4))
sns.histplot(df["price"], bins=50, kde=True)
plt.title("Price Distribution")
plt.xlabel("Price")
plt.ylabel("Frequency")
plt.show()

# %%
#Checking for outliers in Response variable by boxplot
import matplotlib.pyplot as plt
import seaborn as sns

# Set plot style
sns.set(style="whitegrid")

# Create the boxplot
plt.figure(figsize=(6, 4))
sns.boxplot(y=df["price"])
plt.title("Boxplot of House Prices")
plt.ylabel("Price")
plt.tight_layout()
plt.show()

# %%
#outlier detection from ressponse (y)
Q1=df['price'].quantile(0.25)
```

```
Q3=df['price'].quantile(0.75)

IQR=Q3-Q1

lower_bound=Q1-1.5*IQR

upper_bound=Q3+1.5*IQR


# %%

#Outlier removal

df_clean = df[(df["price"] >= lower_bound) & (df["price"] <=
upper_bound)].reset_index(drop=True)


print(f"Original rows: {len(df)}")

print(f"After removing Price outliers: {len(df_clean)}")


# %%

df=df_clean


# %%

#There are 44 cities, Dummy variables can be too much in mumber if we create 43
dummies, So we are taking the top 10 most frequent(repeated cities)

city_df=df[['city']].groupby(['city']).agg({'city':'count'})


# %%

city_df.rename(columns={'city': 'city_count'}, inplace=True)


# %%

city_df=city_df.reset_index()


# %%

city_df = city_df.sort_values('city_count', ascending=False)
```

```
# %%
city_df=city_df.head(10)

# %%
city_list=city_df['city'].to_list()

# %%
df['city']=df['city'].apply(lambda x: x if x in city_list else 'other')

# %%
df.reset_index(inplace=True)

# %%
#Making Regression inputs ready

x=df.drop(['price','date','yr_built','yr_renovated','street','statezip','country','city'],axis=1)

# %%
#assigning dummy variable for city
df_encoded=pd.get_dummies(df['city'],drop_first=True).astype(int)

# %%
x=pd.concat([x,df_encoded],axis=1)

# %%
x.drop(['index'],axis=1,inplace=True)
```

```
# %%
#making response variable ready for regression
y=df['price']

# %%
#starting calculations for PCA, startig with
x_numeric=x
#  standardize data
x_mean=np.mean(x_numeric,axis=0)
x_std=np.std(x_numeric,axis=0)
x_numeric=(x_numeric-x_mean)/x_std

# %%
#calculating COV matrix on standardized data
cov_matrix=np.cov(x_numeric.T)

# %%
# Compute eigenvalues and eigenvectors
eigen_values, eigen_vectors = np.linalg.eigh(cov_matrix)

# Sort them in descending order of eigenvalue
sorted_indices = np.argsort(eigen_values)[::-1]
eigen_values = eigen_values[sorted_indices]
eigen_vectors = eigen_vectors[:, sorted_indices]

# %%
# Choose number of components (e.g., enough to explain 95% variance)
total_variance = np.sum(eigen_values)
```

```
explained_variance_ratio = eigen_values / total_variance

cumulative_variance = np.cumsum(explained_variance_ratio)


# Pick number of components that explain >= 95% variance

k = np.argmax(cumulative_variance >= 0.95) + 1

print(f"Number of components explaining 95% variance: {k}")


# %%

# Select top-k eigenvectors

principal_components = eigen_vectors[:, :k]


# Project data

x_pca = np.dot(x_numeric, principal_components)


#PCA is done till here .


# %%

#Running linear regression on this transformed data

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score


# Train/test split on reduced data

X_train, X_test, y_train, y_test = train_test_split(x_pca, y, test_size=0.3,
random_state=42)


model = LinearRegression()

model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)


print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
print("R² Score:", r2_score(y_test, y_pred))


# %%
adjusted_r2 = 1 - (1 - r2_score(y_test, y_pred)) * (len(y_test) - 1) / (len(y_test) - X_test.shape[1] - 1)


# %%
adjusted_r2
```