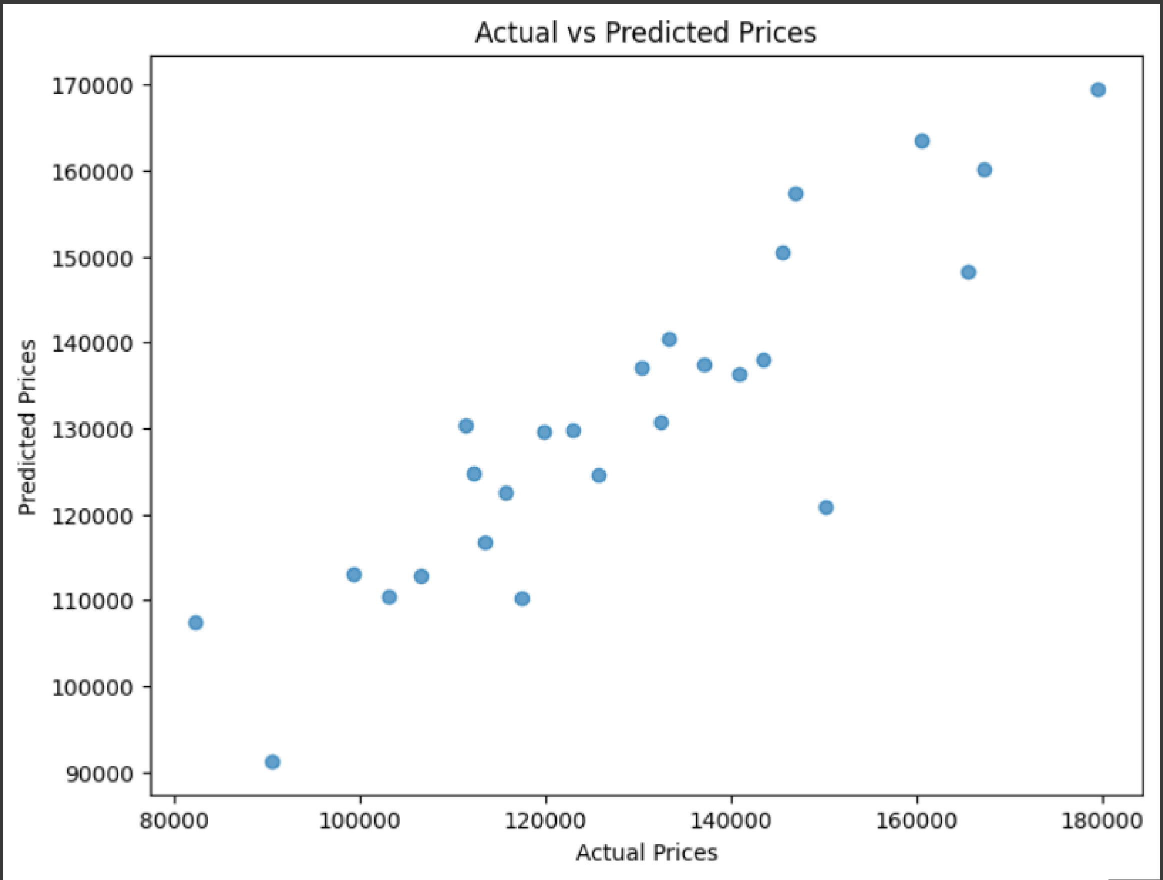
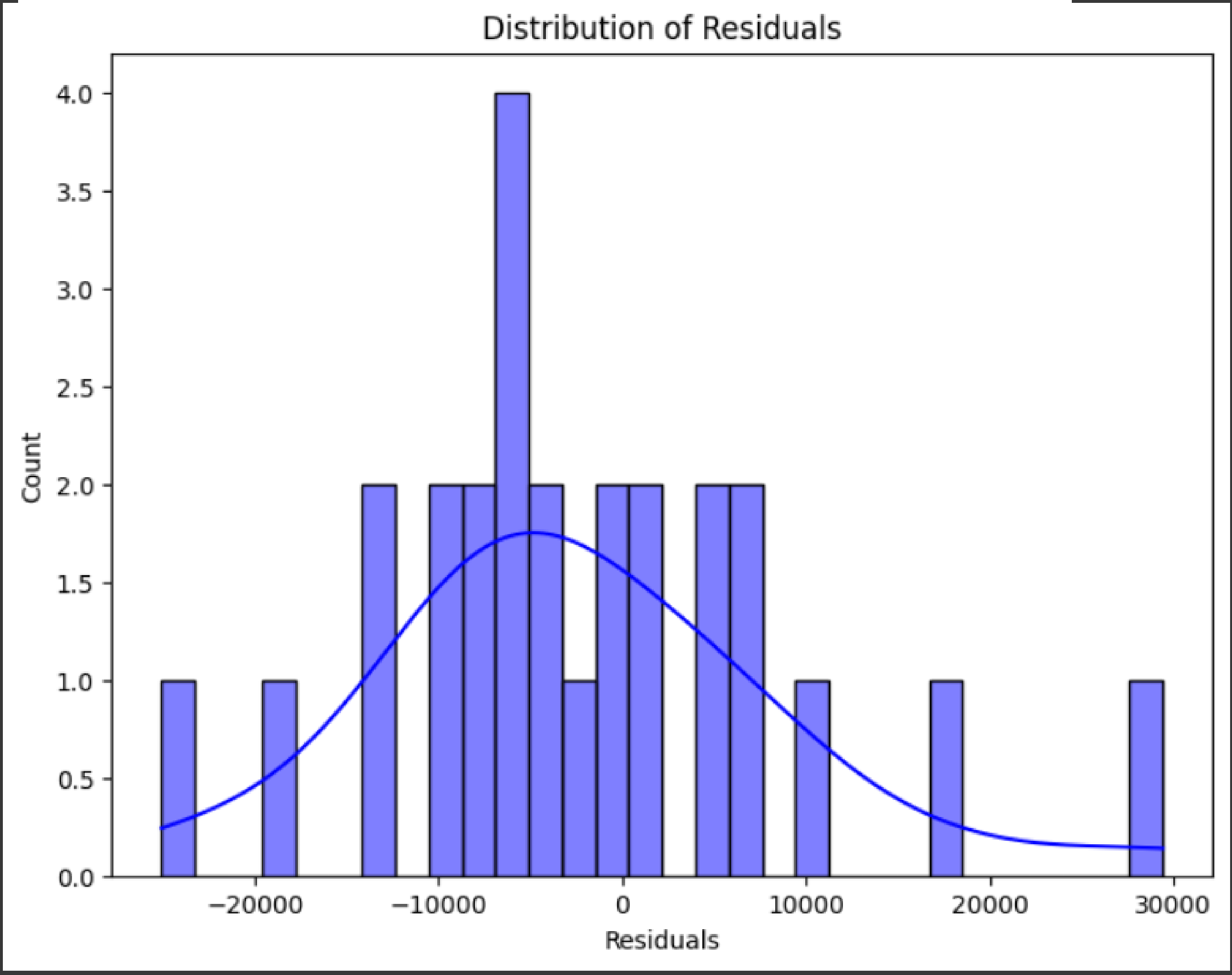
```
# Importing necessary Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
# Assuming 'house_prices.csv' is in the current directory
data = pd.read_csv('/content/house-prices MF ori.csv')
print("First 5 rows of the dataset:")
print(data.head())

# Dataset info
print("\nDataset Information:")
print(data.info())

# Check for missing values
print("\nMissing Values: ")
print(data.isnull().sum())

# Fill missing values (example: median for numerical columns)
data['SqFt'].fillna(data['SqFt'].median(), inplace=True)
data['Bedrooms'].fillna (data['Bedrooms'].median(), inplace=True)

# Handle outliers (example: capping)
upper_limit = data['Price'].quantile (0.95)
data['Price'] = np.where(data['Price'] > upper_limit, upper_limit, data['Price'])


# Encoding the 'Location' column using one-hot encoding
data = pd.get_dummies(data, columns=['Neighborhood', 'Home'], drop_first=True)

from sklearn.preprocessing import MinMaxScaler

# Normalize numerical columns
scaler=MinMaxScaler()
data[ [ 'SqFt', 'Bedrooms']]= scaler.fit_transform(data[['SqFt', 'Bedrooms']])

# Define features and target variable
X = data.drop('Price', axis=1)
y = data['Price']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) # Changed 'x' to 'X'
print (f"Training set size: {X_train.shape}")
print (f"Testing set size: {X_test.shape}")

# Initialize and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Display coefficients
print("Model Coefficients:", model.coef_)
print("Intercept: ", model.intercept_)


y_pred = model.predict(X_test)

#Evaluate the model
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score (y_test, y_pred)

print(f"RMSE: {rmse:.2f}")
print(f"R2: {r2:.2f}")


# Scatter plot of actual vs predicted prices
plt.figure(figsize=(8, 6))
plt.scatter (y_test, y_pred, alpha=0.7)
plt.title("Actual vs Predicted Prices")
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.show()

# Residual plot
residuals = y_test - y_pred
plt.figure(figsize=(8, 6))
```

```
    sns.histplot(residuals, kde=True, bins=30, color='blue')
    plt.title("Distribution of Residuals")
    plt.xlabel("Residuals")
    plt.show()
```

```
First 5 rows of the dataset:
   Home   Price  SqFt  Bedrooms  Bathrooms  Offers  Brick  Neighborhood
0     1  114300  1790         2          2       2      2           101
1     2  114200  2030         4          2       3      2           101
2     3  114800  1740         3          2       1      2           101
3     4   94700  1980         3          2       3      2           101
4     5  119800  2130         3          3       3      2           101

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 128 entries, 0 to 127
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Home          128 non-null    int64
 1   Price         128 non-null    int64
 2   SqFt          128 non-null    int64
 3   Bedrooms      128 non-null    int64
 4   Bathrooms     128 non-null    int64
 5   Offers        128 non-null    int64
 6   Brick         128 non-null    int64
 7   Neighborhood  128 non-null    int64
dtypes: int64(8)
memory usage: 8.1 KB
None

Missing Values:
Home            0
Price           0
SqFt            0
Bedrooms        0
Bathrooms       0
Offers          0
Brick           0
Neighborhood    0
dtype: int64
Training set size: (102, 134)
Testing set size: (26, 134)
Model Coefficients: [ 3.96012842e+04  5.45124252e+03  8.13017724e+03 -6.45453118e+03
 -1.27413706e+04  1.85749763e+04 -7.19045383e+03 -5.61674296e+03
 -6.03471359e+03 -2.15627637e+04 -3.04023473e-12 -1.97446774e+04
  1.10140751e+03  1.33781364e+03 -3.39578045e+03 -1.17084488e+04
  0.00000000e+00  2.27373675e-12  2.41387912e+03  3.35975003e+03
  1.72793723e+04  3.18371053e+03  9.12524628e+03 -2.74953356e+04
 -3.63797881e-12 -3.63797881e-12  6.53745034e+03  1.04874619e+04
 -7.29829897e+03 -1.01556054e+04  3.00434371e+03  1.95116618e+04
  3.63797881e-12  9.09494702e-12 -2.49548010e+04  6.94529090e+03
  1.25593798e+04  1.27329258e-11 -1.15135630e+04 -5.13533257e+03
  9.85001160e+03 -3.57628397e+03 -9.09494702e-12 -6.61517308e+03
 -7.41493051e+03  1.01027142e+03  5.45696821e-12 -1.27762136e+04
 -9.19306254e+03  1.41710815e+04  5.72569978e+03 -9.32232069e-12
  7.61268286e+03 -1.84037413e+03 -5.00021404e+03 -4.58704921e+03
  1.79273095e+04 -1.19866728e+04  2.56367025e+03  1.75591339e+04
 -1.63087802e+04 -3.63797881e-12 -8.52651283e-12 -1.51686991e+03
 -1.29136640e+04  2.01494026e+03  1.59456673e+04  4.32881768e+01
  1.72335828e+03 -6.09873348e+03 -9.09494702e-12 -3.39453858e+03
  7.46231129e+03  2.36057001e+04 -1.45758537e+04 -1.81898940e-12
  1.20328513e+04  1.78328246e+03  9.69121977e+03 -7.23175024e+03
 -4.32599221e+03 -4.03072272e+03 -9.75537274e+03  8.55224079e+03
  7.01699484e+03  9.83710532e+02 -1.81898940e-12 -3.18323146e-12
  3.69725013e+03  1.91520138e+03 -1.81898940e-12  6.10484867e+03
  1.04417796e+03  7.35544671e+03  5.77988014e+03 -4.04283690e+03
  2.07492611e+03 -1.21227495e+04 -3.20820709e+03  2.27787310e+04
  0.00000000e+00 -5.82371238e+03  0.00000000e+00 -1.39908751e+04
  0.00000000e+00  5.36238417e+03  5.99461344e+03  1.88949021e+03
  3.92426374e+03  5.95916581e+03  0.00000000e+00  0.00000000e+00
  6.71526023e+02 -4.78677943e+03 -2.04816912e+04 -1.20786996e+04
  8.14960135e+02  9.52735226e+03 -8.65206584e+03  0.00000000e+00
 -9.20669202e+03 -1.19234496e+04 -1.16864593e+04  4.35743707e+03
  0.00000000e+00  1.09835803e+04 -3.24850784e+03 -1.70560425e+04
  1.92484674e+03 -1.46552740e+04  2.69577332e+03  0.00000000e+00
 -7.36649834e+03  4.47279254e+03]
Intercept:  124620.53965201211
RMSE: 11240.03
R2: 0.78
<ipython-input-30-4aa918083afd>:25: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained a
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  data['SqFt'].fillna(data['SqFt'].median(), inplace=True)
<ipython-input-30-4aa918083afd>:26: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained a
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  data['Bedrooms'].fillna (data['Bedrooms'].median(), inplace=True)
```





Actual vs Predicted Prices