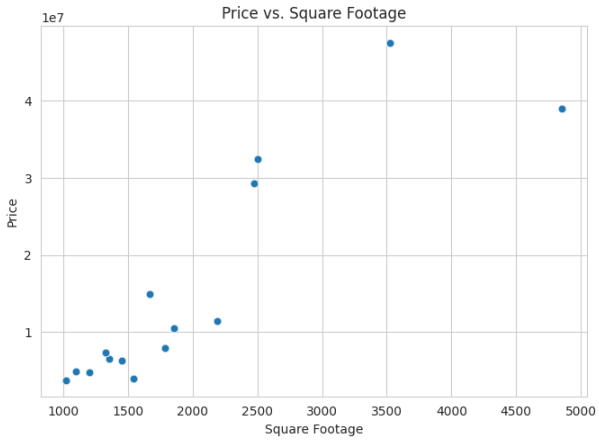
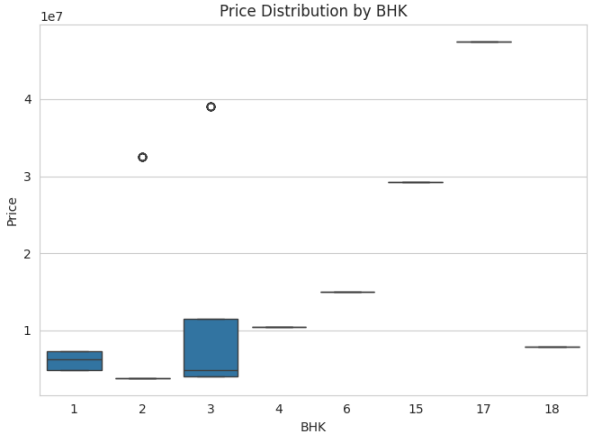
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,
r2_score,mean_absolute_error
import matplotlib.pyplot as plt
import math
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder

path="/content/drive/MyDrive/Dataset/hp_data - hp_data.csv"
df=pd.read_csv(path)

print("Data Preview:")
print(df.head())
print(df.info())
```

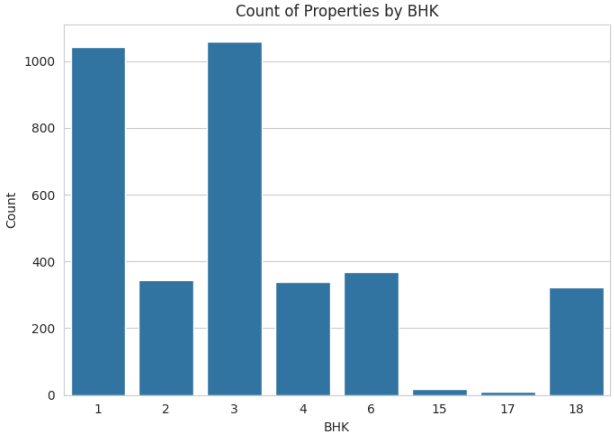
```
Data Preview:
        place                  built  sqft    sale  yearsOld  floor
totalFloor  bhk     price
0  BTM Layout  Super built-up  Area  1450  Resale         5      1
4    1   6300000
1   Yelahanka  Super built-up  Area  2190  Resale         5      3
5    3  11500000
2  Whitefield  Super built-up  Area  1019  Resale         1      2
5    2   3800000
3  Ambalipura  Super built-up  Area  1857  Resale        15      4
5    4  10500000
4   Yelahanka  Super built-up  Area  2190  Resale         5      3
5    3  11500000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3500 entries, 0 to 3499
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   place       3500 non-null   object
 1   built       3500 non-null   object
 2   sqft        3500 non-null   int64
 3   sale        3500 non-null   object
 4   yearsOld    3500 non-null   int64
 5   floor       3500 non-null   int64
 6   totalFloor  3500 non-null   int64
 7   bhk         3500 non-null   int64
 8   price       3500 non-null   int64
dtypes: int64(6), object(3)
memory usage: 246.2+ KB
None
```

```python
sns.set_style("whitegrid")
```

```
# Create subplots
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# 1. Distribution of Property Prices
sns.histplot(df["price"], bins=30, kde=True, ax=axes[0, 0])
axes[0, 0].set_title("Distribution of Property Prices")
axes[0, 0].set_xlabel("Price")
axes[0, 0].set_ylabel("Count")

# 2. Scatter Plot of Price vs. Square Footage
sns.scatterplot(x=df["sqft"], y=df["price"], alpha=0.5, ax=axes[0, 1])
axes[0, 1].set_title("Price vs. Square Footage")
axes[0, 1].set_xlabel("Square Footage")
axes[0, 1].set_ylabel("Price")

# 3. Boxplot of Prices by BHK
sns.boxplot(x=df["bhk"], y=df["price"], ax=axes[1, 0])
axes[1, 0].set_title("Price Distribution by BHK")
axes[1, 0].set_xlabel("BHK")
axes[1, 0].set_ylabel("Price")

# 4. Count of Properties by BHK
sns.countplot(x=df["bhk"], ax=axes[1, 1])
axes[1, 1].set_title("Count of Properties by BHK")
axes[1, 1].set_xlabel("BHK")
axes[1, 1].set_ylabel("Count")

# Adjust layout and show plots
plt.tight_layout()
plt.show()
```








```
print("Missing Values:")
print(df.isnull().sum())
print("Statistical Summary:")
print(df.describe())
```

```
Missing Values:
place         0
built         0
sqft          0
sale          0
yearsOld      0
floor         0
totalFloor    0
bhk           0
price         0
dtype: int64
Statistical Summary:
              sqft     yearsOld        floor   totalFloor          bhk
price
count  3500.000000  3500.000000  3500.000000  3500.000000  3500.000000
3.500000e+03
mean   1538.163143     7.602000     4.197714     6.371429     4.197714
8.067807e+06
```

```
std     416.264178     3.803196     4.782410     4.461808     4.782410
4.984973e+06
min    1019.000000     1.000000     1.000000     4.000000     1.000000
3.800000e+06
25%    1200.000000     5.000000     1.000000     4.000000     1.000000
4.800000e+06
50%    1543.000000     5.000000     3.000000     4.000000     3.000000
6.600000e+06
75%    1784.000000    10.000000     4.000000     5.000000     4.000000
1.050000e+07
max    4856.000000    15.000000    18.000000    29.000000    18.000000
4.750000e+07
```

```
#Handling Missing Values by filling with meadian
df.fillna(df.select_dtypes(include=['number']).median(), inplace=True)

X = df.drop('price',axis=1)  # Features (all except the last column)
y = df['price']  # Target (last column)

# Split 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)
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-2-acfbc6cdd205> in <cell line: 0>()
----> 1 X = df.drop('price',axis=1)  # Features (all except the last
column)
      2 y = df['price']  # Target (last column)
      3
      4 # Split into training and testing sets
      5 x_train, x_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

NameError: name 'df' is not defined
```

```
#Define Model
model=LinearRegression()

#Training the model
model.fit(x_train, y_train)
```

```
---------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-89-fe1c47adc574> in <cell line: 0>()
      1 #Training the model
----> 2 model.fit(x_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/base.py in
wrapper(estimator, *args, **kwargs)
   1387                 )
   1388             ):
-> 1389                 return fit_method(estimator, *args, **kwargs)
   1390
   1391         return wrapper

/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_base.py
in fit(self, X, y, sample_weight)
    599         accept_sparse = False if self.positive else ["csr",
"csc", "coo"]
    600
--> 601         X, y = validate_data(
    602             self,
    603             X,

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py in
validate_data(_estimator, X, y, reset, validate_separately,
skip_check_array, **check_params)
   2959             y = check_array(y, input_name="y",
**check_y_params)
   2960         else:
-> 2961             X, y = check_X_y(X, y, **check_params)
   2962         out = X, y
   2963

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py in
check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order,
copy, force_writeable, force_all_finite, ensure_all_finite, ensure_2d,
allow_nd, multi_output, ensure_min_samples, ensure_min_features,
y_numeric, estimator)
   1368     ensure_all_finite =
_deprecate_force_all_finite(force_all_finite, ensure_all_finite)
   1369
-> 1370     X = check_array(
   1371         X,
   1372         accept_sparse=accept_sparse,

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py in
check_array(array, accept_sparse, accept_large_sparse, dtype, order,
copy, force_writeable, force_all_finite, ensure_all_finite,
ensure_non_negative, ensure_2d, allow_nd, ensure_min_samples,
ensure_min_features, estimator, input_name)
   1053                     array = xp.astype(array, dtype,
copy=False)
   1054                 else:
-> 1055                     array = _asarray_with_order(array,
order=order, dtype=dtype, xp=xp)
```

```
   1056             except ComplexWarning as complex_warning:
   1057                 raise ValueError(

/usr/local/lib/python3.11/dist-packages/sklearn/utils/_array_api.py in
_asarray_with_order(array, dtype, order, copy, xp, device)
    837             array = numpy.array(array, order=order,
dtype=dtype)
    838         else:
--> 839             array = numpy.asarray(array, order=order,
dtype=dtype)
    840
    841         # At this point array is a NumPy ndarray. We convert
it to an array

/usr/local/lib/python3.11/dist-packages/pandas/core/generic.py in
__array__(self, dtype, copy)
   2151     ) -> np.ndarray:
   2152         values = self._values
-> 2153         arr = np.asarray(values, dtype=dtype)
   2154         if (
   2155             astype_is_view(values.dtype, arr.dtype)

ValueError: could not convert string to float: 'Abbaiah Reddy Layout'
```

```
#Get model parameters
c=model.intercept_#y_intersept
m=model.coef_#slope
print(f"intersept:{c}")
print(f"slope:{m}")
```

```
intersept:-6476299.4585976135
slope:[9454.02057915]
```

```
#Evaluation
print("linear model:")
print(f"R2 score:{model.score(x_train, y_train)}")
print(f"MSE:{mean_squared_error(y_test, y_pred)}")
print(f"RMSE:{math.sqrt(mean_squared_error(y_test, y_pred))}")
print(f"MAE:{mean_absolute_error(y_test, y_pred)}")
```

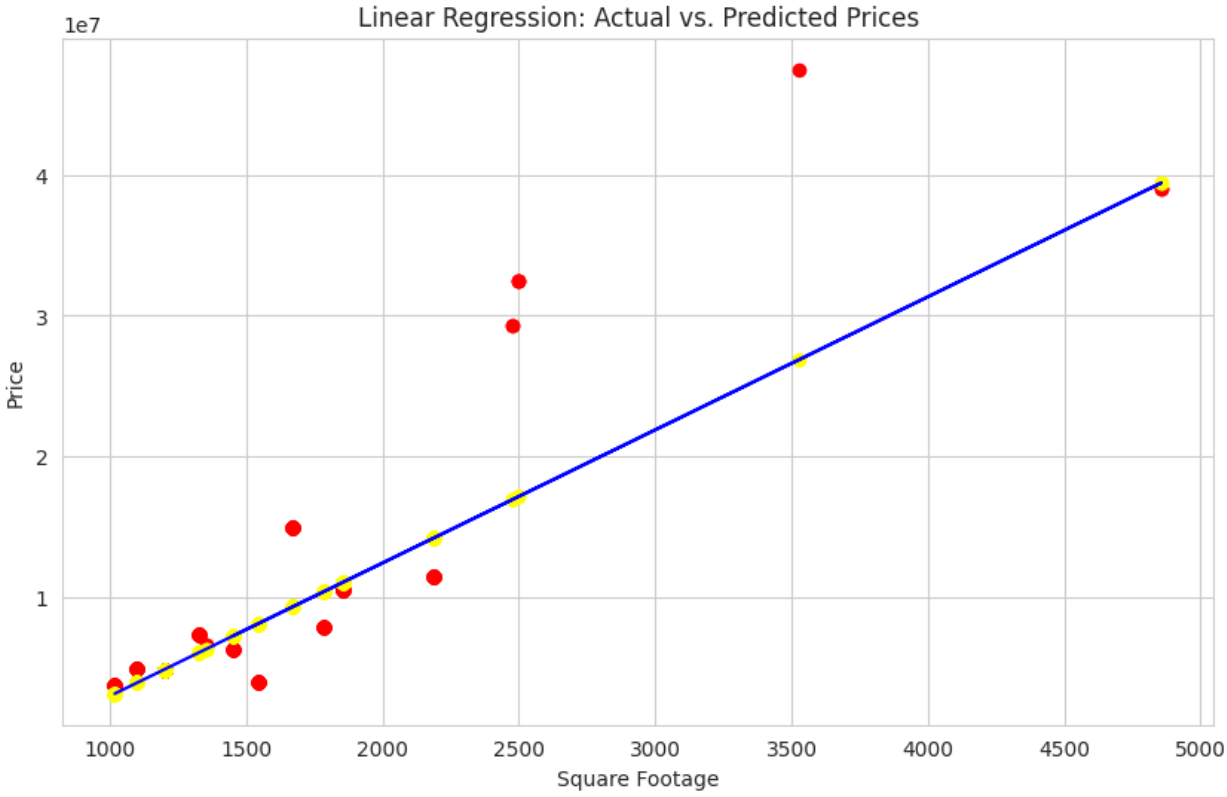
```
linear model:
R2 score:0.6107916345891731
MSE:88336150804000.89
RMSE:9398731.34013314
MAE:8134284.818571429
```

```
# Predict using the trained model
y_pred = model.predict(x_test)
```

```
yhat = m*X+ c  # Best fit Line
X,yhat
```

```
(      sqft
 0     1450
 1     2190
 2     1019
 3     1857
 4     2190
 ...    ...
 3495  1019
 3496  1450
 3497  1330
 3498  1200
 3499  1019

 [3500 rows x 1 columns],
               sqft
 0     7.232030e+06
 1     1.422801e+07
 2     3.157348e+06
 3     1.107982e+07
 4     1.422801e+07
 ...            ...
 3495  3.157348e+06
 3496  7.232030e+06
 3497  6.097548e+06
 3498  4.868525e+06
 3499  3.157348e+06

 [3500 rows x 1 columns])
```

```python
# Visualization
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))  # Adjust figure size as needed

plt.plot(X, yhat, color='blue', label='Best Fit Line')
plt.scatter(x_test['sqft'], y_test, color='red', label='Actual
Values')
plt.scatter(x_test['sqft'], y_pred, color='yellow', label='Predicted
Values')

plt.xlabel('Square Footage')
plt.ylabel('Price')
plt.title('Linear Regression: Actual vs. Predicted Prices')
plt.grid(True)
plt.show()
```


Linear Regression: Actual vs. Predicted Prices
1e7
4
3
2
1
Price
1000
1500
2000
2500
3000
3500
4000
4500
5000
Square Footage