Import Libraries

```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

Load, Clean and Visualize Dataset

```
df = pd.read_csv("/content/kc_house_data.csv")
df.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_t
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	

5 rows × 21 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

```
# Check for missing values
print(df.isnull().sum())
```

```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

```
# Check for duplicates
print(df.duplicated(subset='id').sum())
```

What can I help you build?

```
177
```

```
df = df.drop_duplicates(subset='id', keep='first')
```

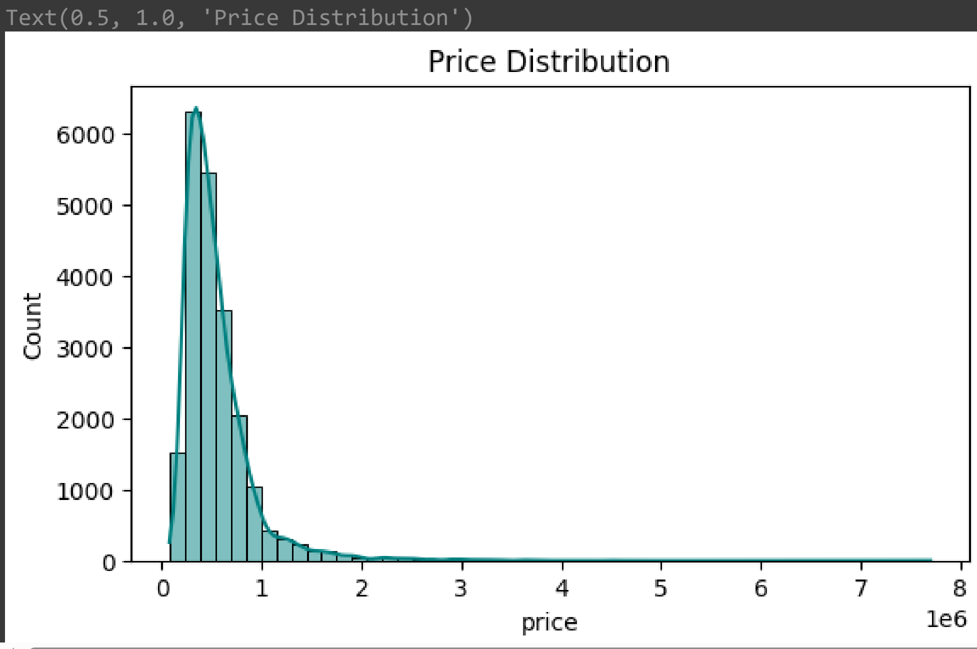
```
 # Check for impossible values
print(df[df['bedrooms'] == 0])
print(df[df['price'] <= 0])
```

```
9854   7849202190  20141223T000000   235000.0         0       0.00
12653  7849202299  20150218T000000   320000.0         0       2.50
14423  9543000205  20150413T000000   139950.0         0       0.00
18379  1222029077  20141029T000000   265000.0         0       0.75
19452  3980300371  20140926T000000   142000.0         0       0.00

       sqft_living  sqft_lot  floors  waterfront  view  ...  grade  \
875           3064      4764     3.5           0     2  ...      7
3119          1470       979     3.0           0     2  ...      8
3467          1430      1650     3.0           0     0  ...      7
4868           390      5900     1.0           0     0  ...      4
6994          4810     28008     2.0           0     0  ...     12
8477          2290      8319     2.0           0     0  ...      8
8484          1810      5669     2.0           0     0  ...      7
9773          2460      8049     2.0           0     0  ...      8
9854          1470      4800     2.0           0     0  ...      7
12653         1490      7111     2.0           0     0  ...      7
14423          844      4269     1.0           0     0  ...      7
18379          384    213444     1.0           0     0  ...      4
19452          290     20875     1.0           0     0  ...      1

       sqft_above  sqft_basement  yr_built  yr_renovated  zipcode      lat  \
875          3064              0      1990             0    98102  47.6362
3119         1470              0      2006             0    98133  47.7145
3467         1430              0      1999             0    98125  47.7222
4868          390              0      1953             0    98118  47.5260
6994         4810              0      1990             0    98053  47.6642
8477         2290              0      1985             0    98042  47.3473
8484         1810              0      2003             0    98038  47.3493
9773         2460              0      1990             0    98031  47.4095
9854         1470              0      1996             0    98065  47.5265
12653        1490              0      1999             0    98065  47.5261
14423         844              0      1913             0    98001  47.2781
18379         384              0      2003             0    98070  47.4177
19452         290              0      1963             0    98024  47.5308

          long  sqft_living15  sqft_lot15
875   -122.322           2360        4000
3119  -122.356           1470        1399
3467  -122.290           1430        1650
4868  -122.261           2170        6000
6994  -122.069           4740       35061
8477  -122.151           2500        8751
8484  -122.053           1810        5685
9773  -122.168           2520        8050
9854  -121.828           1060        7200
12653 -121.826           1500        4675
14423 -122.250           1380        9600
18379 -122.491           1920      224341
19452 -121.888           1620       22850

[13 rows x 21 columns]
Empty DataFrame
Columns: [id, date, price, bedrooms, bathrooms, sqft_living, sqft_lot, floors, waterfront, view, condition, grade, sqft_above, sqft_basement, yr_built,
Index: []

[0 rows x 21 columns]
```

```
# 2. Remove houses with 0 bedrooms
df = df[df['bedrooms'] > 0]
```

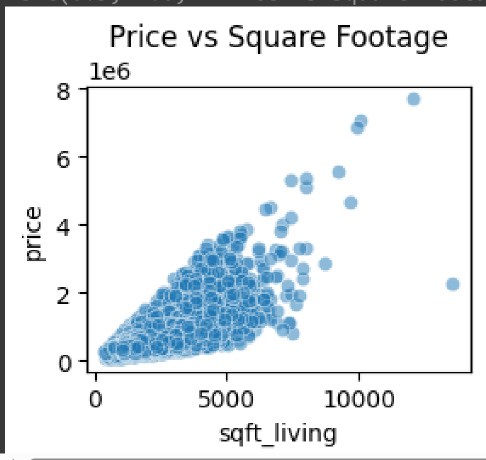
```
plt.figure(figsize=(14, 8))
plt.subplot(2, 2, 1)
sns.histplot(df['price'], bins=50, kde=True, color='teal')
plt.title('Price Distribution')
```

```
Text(0.5, 1.0, 'Price Distribution')
```



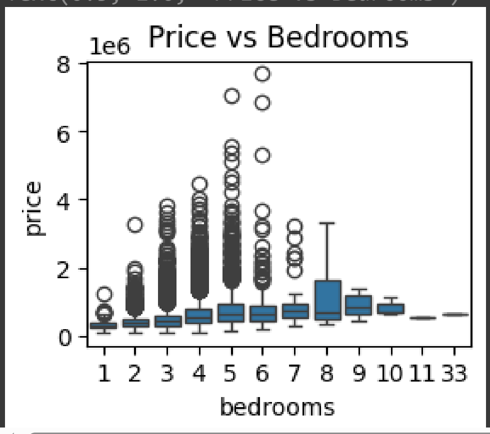

```
plt.subplot(2, 2, 2)
sns.scatterplot(data=df, x='sqft_living', y='price', alpha=0.5)
plt.title('Price vs Square Footage')
```

Text(0.5, 1.0, 'Price vs Square Footage')

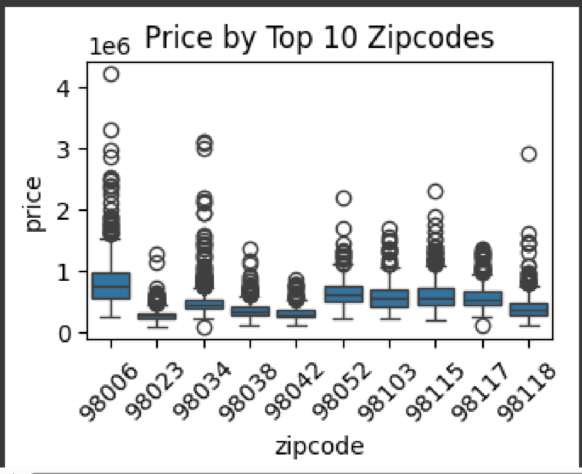



```
plt.subplot(2, 2, 3)
sns.boxplot(x='bedrooms', y='price', data=df)
plt.title('Price vs Bedrooms')
```

Text(0.5, 1.0, 'Price vs Bedrooms')




```
plt.subplot(2, 2, 4)
top_zipcodes = df['zipcode'].value_counts().nlargest(10).index
sns.boxplot(x='zipcode', y='price', data=df[df['zipcode'].isin(top_zipcodes)])
plt.title('Price by Top 10 Zipcodes')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```




```
# Remove top 1% outliers in price and sqft_living
df = df[df['price'] < df['price'].quantile(0.99)]
df = df[df['sqft_living'] < df['sqft_living'].quantile(0.99)]
```

```
# Apply log transformation to stabilize skewness
df['log_price'] = np.log1p(df['price'])
```

KMeans Clustering for Fine-Grained Location

```
# Use latitude and longitude for clustering into location groups
coords = df[['lat', 'long']]
kmeans = KMeans(n_clusters=5, random_state=42, n_init=10)
df['location_cluster'] = kmeans.fit_predict(coords)
```

Feature Selection and Encoding

```
# Select important features
features = ['sqft_living', 'bedrooms', 'bathrooms', 'floors', 'zipcode', 'location_cluster']
X = df[features]
y = df['log_price']

# One-hot encode categorical features
X = pd.get_dummies(X, columns=['zipcode', 'location_cluster'], drop_first=True)
```

Train-Test Split and Scaling

```
# 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)
```

```
# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Train Model

```
# Gradient Boosting Regressor for prediction
model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=5, random_state=42)
model.fit(X_train_scaled, y_train)
```

```
GradientBoostingRegressor
GradientBoostingRegressor(max_depth=5, random_state=42)
```

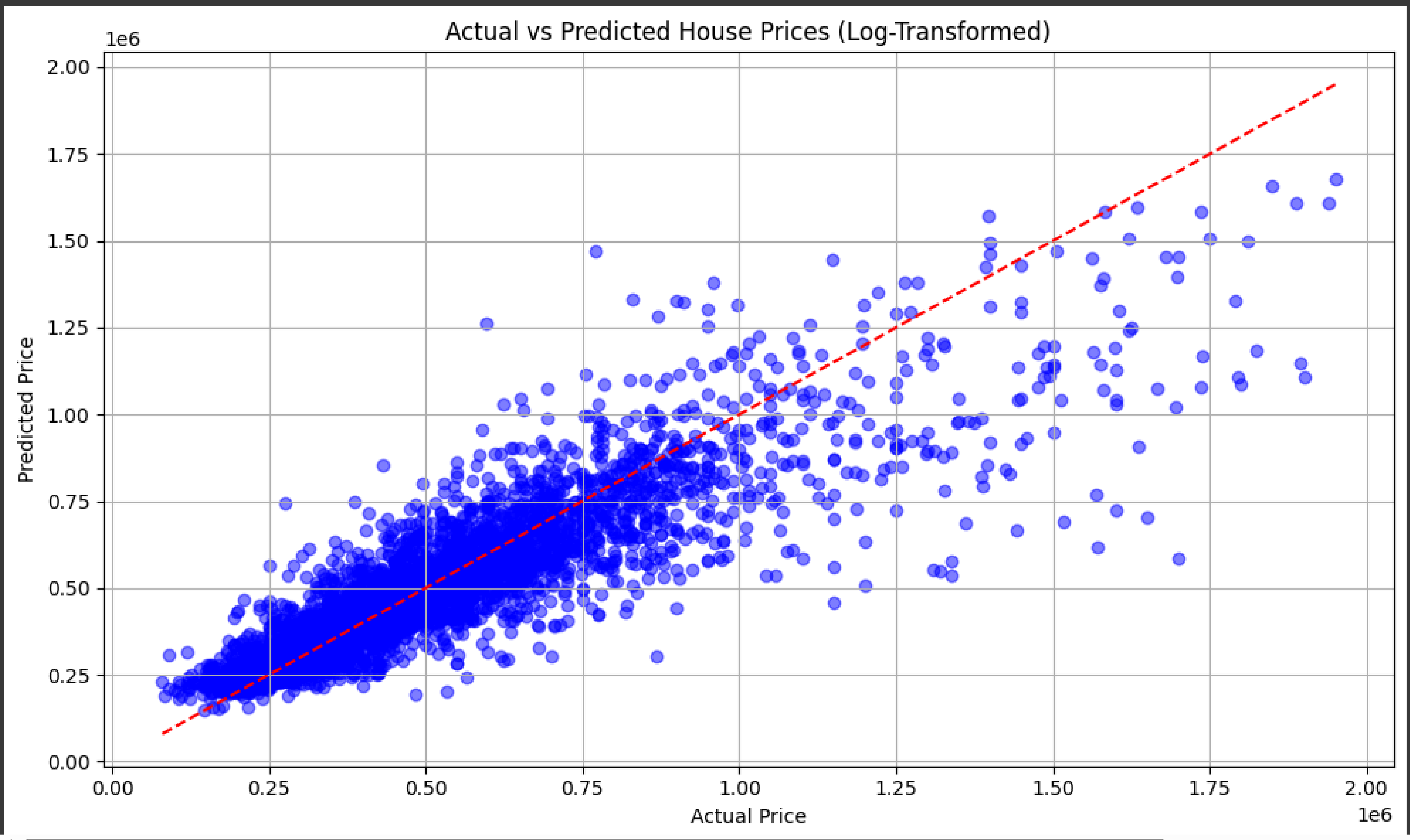
Prediction and Evaluation

```
# Predict and inverse log-transform the predictions
y_pred_log = model.predict(X_test_scaled)
y_pred = np.expm1(y_pred_log)
y_test_actual = np.expm1(y_test)

# Evaluate the model
mae = mean_absolute_error(y_test_actual, y_pred)
rmse = np.sqrt(mean_squared_error(y_test_actual, y_pred))
print(f"MAE: {mae:.2f}")
print(f"RMSE: {rmse:.2f}")
```

```
MAE: 82696.07
RMSE: 129762.05
```

Visualize Actual vs Predicted Prices

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test_actual, y_pred, alpha=0.5, color='blue')
plt.plot([y_test_actual.min(), y_test_actual.max()],
         [y_test_actual.min(), y_test_actual.max()],
         '--', color='red')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual vs Predicted House Prices (Log-Transformed)')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Feature Importance Visualization

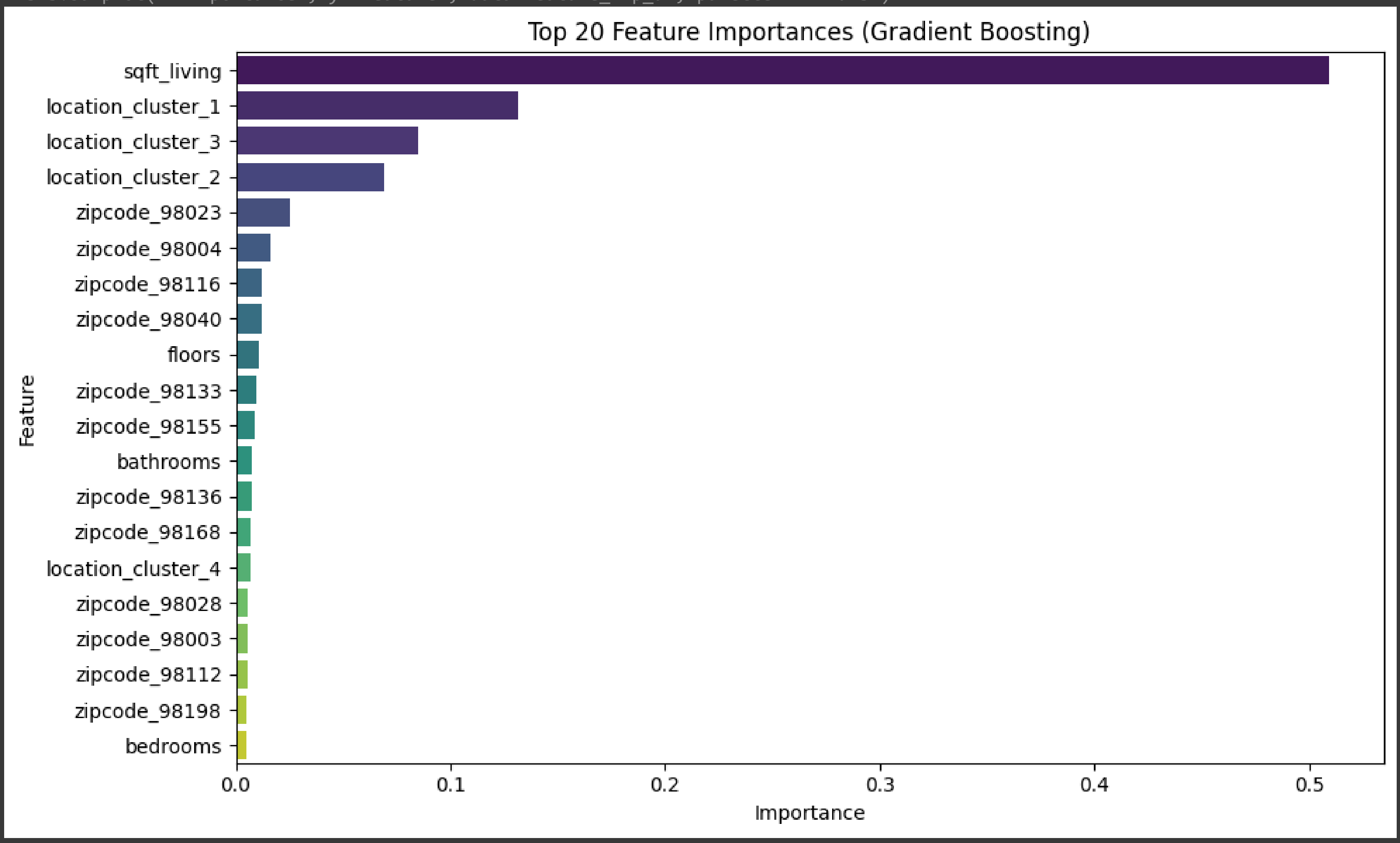
```
# Get feature importances from trained Gradient Boosting model
importances = model.feature_importances_
feature_names = X.columns
```

```
# Create DataFrame of feature importance
feature_imp_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
})
```

```
# Sort and select top 20 most important features
feature_imp_df = feature_imp_df.sort_values(by='Importance', ascending=False).head(20)
```

```
# Plot feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_imp_df, palette='viridis')
plt.title('Top 20 Feature Importances (Gradient Boosting)')
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-40-1887797173.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the s

  sns.barplot(x='Importance', y='Feature', data=feature_imp_df, palette='viridis')
```



Model Tuning using GridSearchCV

```
from sklearn.model_selection import GridSearchCV
```

```
# Define parameter grid to search
param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 5, 7]
}
```

```
# Create GridSearchCV object
grid_search = GridSearchCV(
    estimator=GradientBoostingRegressor(random_state=42),
    param_grid=param_grid,
    cv=3,
    scoring='neg_mean_squared_error',
    verbose=2,
    n_jobs=-1
)
```

```
# Fit on training data
grid_search.fit(X_train_scaled, y_train)
```

```
Fitting 3 folds for each of 18 candidates, totalling 54 fits
```

GridSearchCV

best_estimator_:
GradientBoostingRegressor

GradientBoostingRegressor

```
# Best parameters and model
best_params = grid_search.best_params_
```

```
best_model = grid_search.best_estimator_
print("Best Parameters:", best_params)
```

```
Best Parameters: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 200}
```

Evaluate Tuned Model

```
# Predict and inverse transform log price
y_pred_log_best = best_model.predict(X_test_scaled)
y_pred_best = np.expm1(y_pred_log_best)
y_test_actual = np.expm1(y_test)
```

```
# Evaluate tuned model
mae_best = mean_absolute_error(y_test_actual, y_pred_best)
rmse_best = np.sqrt(mean_squared_error(y_test_actual, y_pred_best))
print(f"Tuned Model MAE: {mae_best:.2f}")
print(f"Tuned Model RMSE: {rmse_best:.2f}")
```

```
Tuned Model MAE: 79774.56
Tuned Model RMSE: 127210.37
```

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test_actual, y_pred, alpha=0.5, color='blue')
plt.plot([y_test_actual.min(), y_test_actual.max()],
         [y_test_actual.min(), y_test_actual.max()],
         '--', color='red')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual vs Predicted House Prices (Log-Transformed)')
plt.grid(True)
plt.tight_layout()
plt.show()
```

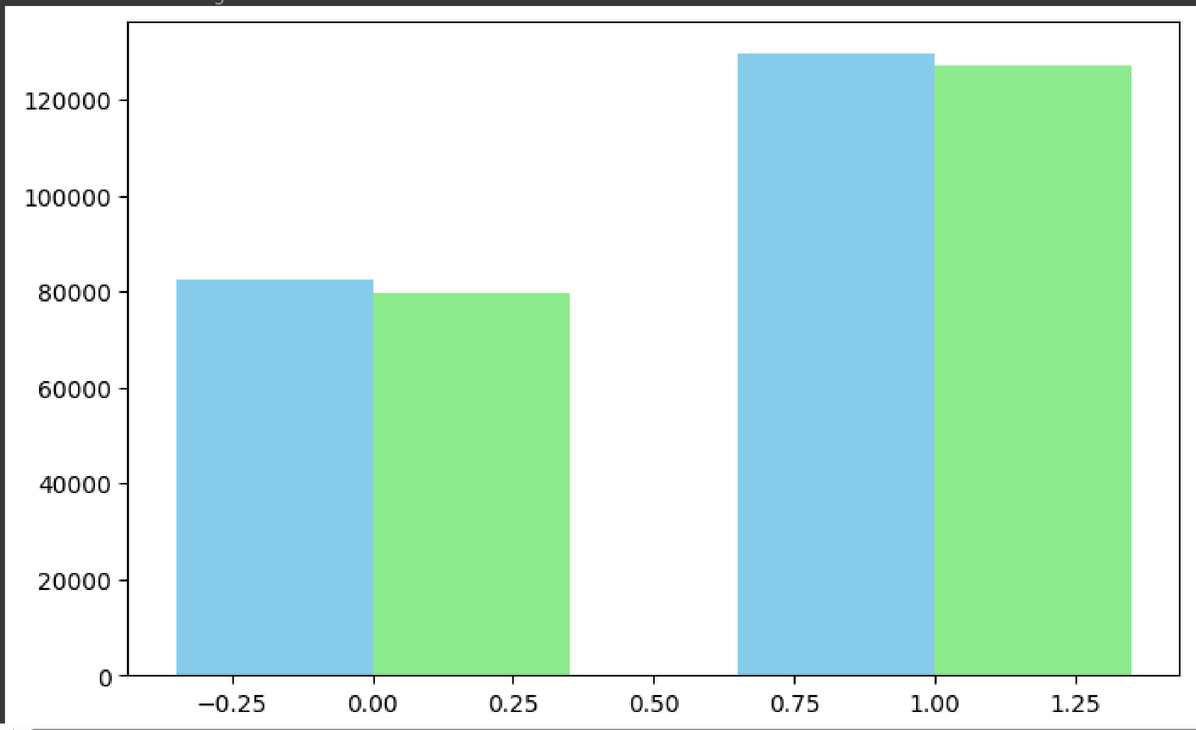
Visualize Evaluation Metrics Comparison (MAE & RMSE)

```
# Define metrics and values
metrics = ['MAE', 'RMSE']
original_scores = [82696.07, 129762.05]
tuned_scores = [79774.56, 127210.37]
```

```
x = np.arange(len(metrics))  # [0, 1]
width = 0.35  # Width of each bar
```

```
# Create figure
plt.figure(figsize=(8, 5))

# Plot bars
plt.bar(x - width/2, original_scores, width, label='Original Model', color='skyblue')
plt.bar(x + width/2, tuned_scores, width, label='Tuned Model', color='lightgreen')
```

```
<BarContainer object of 2 artists>
```



Actual vs Predicted (Tuned Model) Visualization

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test_actual, y_pred_best, alpha=0.5, color='green')
plt.plot([y_test_actual.min(), y_test_actual.max()],
         [y_test_actual.min(), y_test_actual.max()],
         '--', color='red')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual vs Predicted Prices (Tuned Model)')
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```