In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set_theme(color_codes=True)
```

In [2]:
```
# import dataset
df = pd.read_csv('realtor-data.csv')
df.head()
```

Out[2]:

	status	bed	bath	acre_lot	city	state	zip_code	house_size	prev_sold_date	price
0	for_sale	3.0	2.0	0.12	Adjuntas	Puerto Rico	601.0	920.0	NaN	105000.0
1	for_sale	4.0	2.0	0.08	Adjuntas	Puerto Rico	601.0	1527.0	NaN	80000.0
2	for_sale	2.0	1.0	0.15	Juana Diaz	Puerto Rico	795.0	748.0	NaN	67000.0
3	for_sale	4.0	2.0	0.10	Ponce	Puerto Rico	731.0	1800.0	NaN	145000.0
4	for_sale	6.0	2.0	0.05	Mayaguez	Puerto Rico	680.0	NaN	NaN	65000.0

Data Preprocessing Part 1

In [3]:
```
#Check the number of unique value for object datatypes
df.select_dtypes(include='object').nunique()
```

Out[3]:
```
status              2
city              525
state              12
prev_sold_date   3604
dtype: int64
```

In [4]:
```
#Check the number of missing value
df.prev_sold_date.isnull().sum()
```

Out[4]:
```
71255
```

In [5]:
```
#Check the number of row and column
df.shape
```

Out[5]:
```
(100000, 10)
```

In [6]:
```
# Drop prev_sold_date because the missing value is around 70%
df.drop(columns='prev_sold_date', inplace=True)
df.head()
```

Out[6]:

	status	bed	bath	acre_lot	city	state	zip_code	house_size	price
0	for_sale	3.0	2.0	0.12	Adjuntas	Puerto Rico	601.0	920.0	105000.0
1	for_sale	4.0	2.0	0.08	Adjuntas	Puerto Rico	601.0	1527.0	80000.0
2	for_sale	2.0	1.0	0.15	Juana Diaz	Puerto Rico	795.0	748.0	67000.0
3	for_sale	4.0	2.0	0.10	Ponce	Puerto Rico	731.0	1800.0	145000.0
4	for_sale	6.0	2.0	0.05	Mayaguez	Puerto Rico	680.0	NaN	65000.0

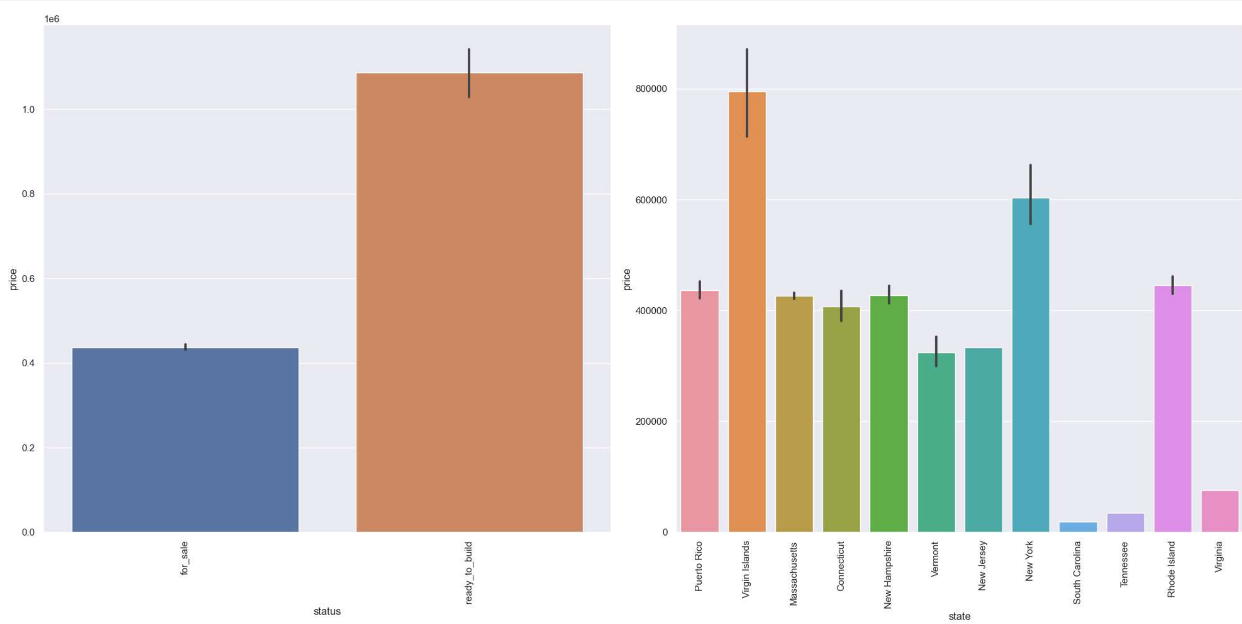
In [7]:
```
# Drop city because the number of unique value for object datatypes is alot
df.drop(columns='city', inplace=True)
df.head()
```

Out[7]:

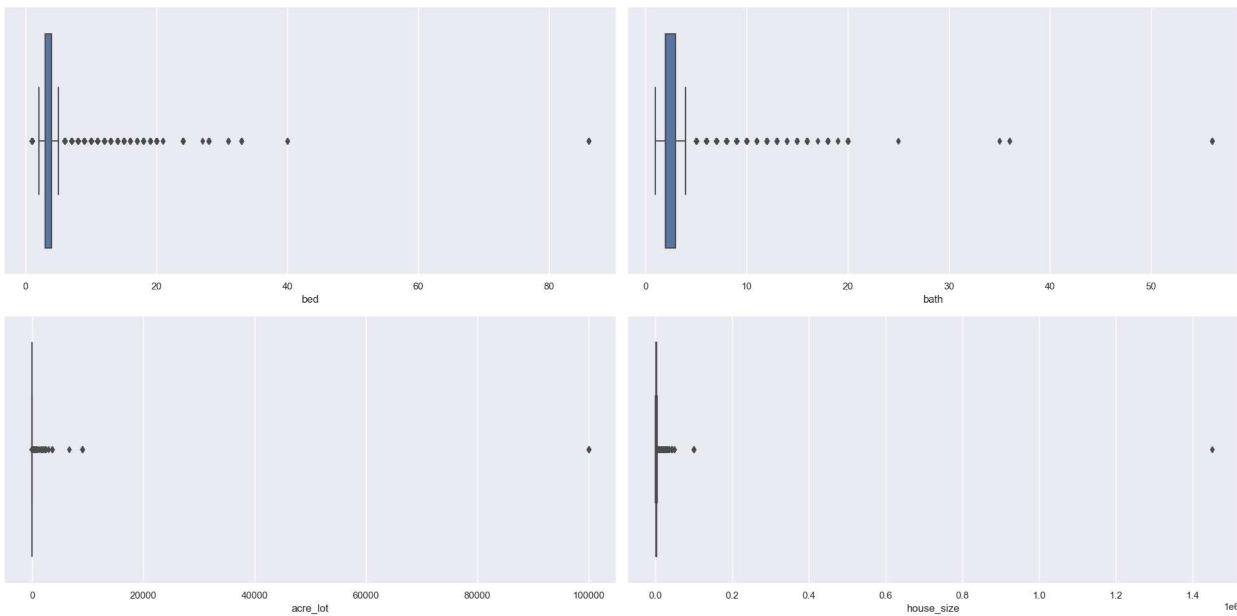
	status	bed	bath	acre_lot	state	zip_code	house_size	price
0	for_sale	3.0	2.0	0.12	Puerto Rico	601.0	920.0	105000.0
1	for_sale	4.0	2.0	0.08	Puerto Rico	601.0	1527.0	80000.0
2	for_sale	2.0	1.0	0.15	Puerto Rico	795.0	748.0	67000.0
3	for_sale	4.0	2.0	0.10	Puerto Rico	731.0	1800.0	145000.0
4	for_sale	6.0	2.0	0.05	Puerto Rico	680.0	NaN	65000.0

Exploratory Data Analysis

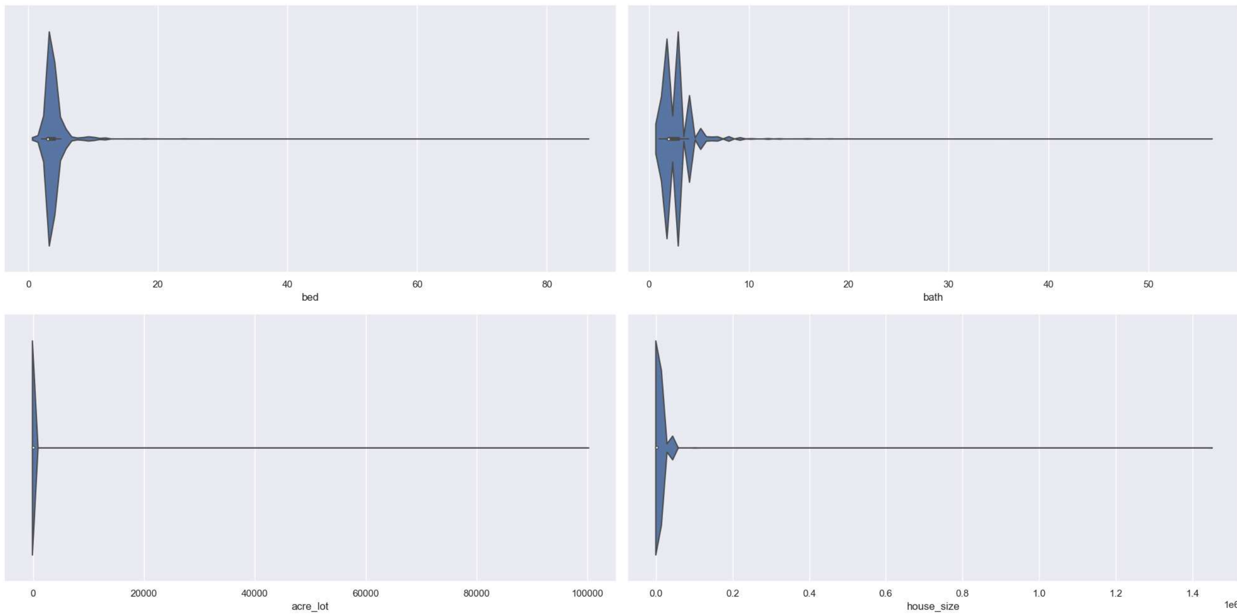
In [8]:
```
# list of categorical variables to plot
cat_vars = ['status', 'state']

# create figure with subplots
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(20, 10))
axs = axs.flatten()

# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.barplot(x=var, y='price', data=df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```



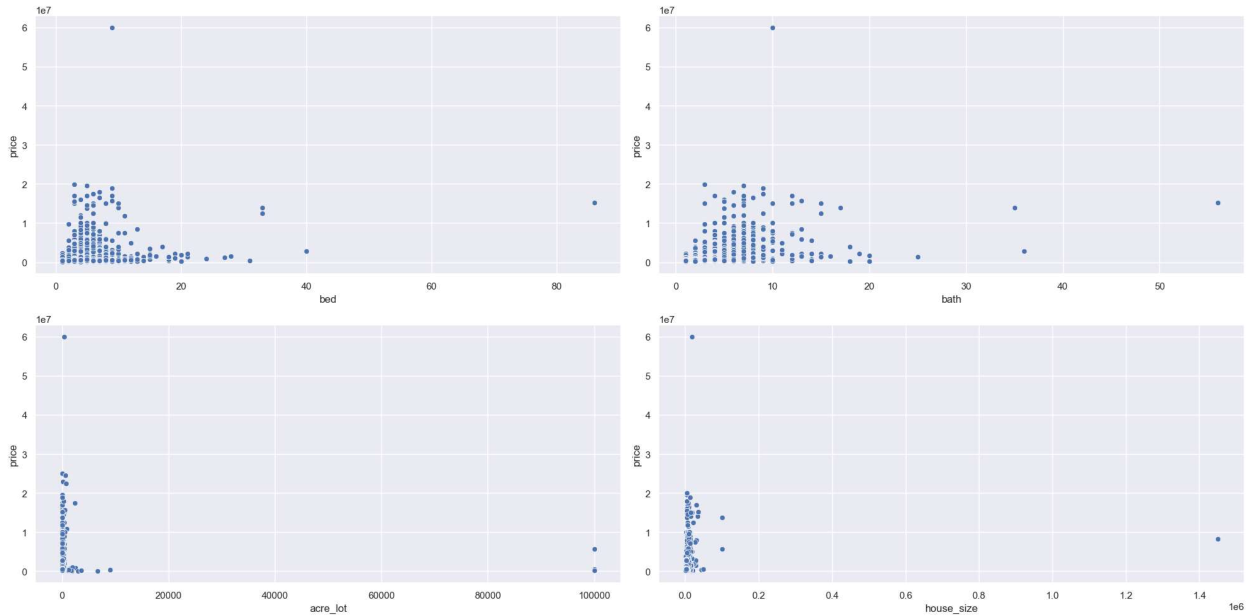
In [9]:
```
num_vars = ['bed', 'bath', 'acre_lot', 'house_size']

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.boxplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [10]:
```
num_vars = ['bed', 'bath', 'acre_lot', 'house_size']

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [11]:
```
num_vars = ['bed', 'bath', 'acre_lot', 'house_size']

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.scatterplot(x=var, y='price', data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



Data Preprocessing Part 2

In [12]:
```
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[12]:
```
bed           24.950
house_size    24.918
bath          24.888
acre_lot      14.013
zip_code       0.195
dtype: float64
```

In [13]:
```
df.drop(columns='zip_code', inplace=True)
df.shape
```

Out[13]:
```
(100000, 7)
```

In [14]:
```
# Fill null value with median and mean
df['bed'].fillna(df['bed'].median(), inplace=True)
df['bath'].fillna(df['bath'].median(), inplace=True)
df['house_size'].fillna(df['house_size'].mean(), inplace=True)
df['acre_lot'].fillna(df['acre_lot'].mean(), inplace=True)
```

In [15]:
```
df.head()
```

Out[15]:

	status	bed	bath	acre_lot	state	house_size	price
0	for_sale	3.0	2.0	0.12	Puerto Rico	920.000000	105000.0
1	for_sale	4.0	2.0	0.08	Puerto Rico	1527.000000	80000.0
2	for_sale	2.0	1.0	0.15	Puerto Rico	748.000000	67000.0
3	for_sale	4.0	2.0	0.10	Puerto Rico	1800.000000	145000.0
4	for_sale	6.0	2.0	0.05	Puerto Rico	2180.081737	65000.0

Label encoding for object datatypes

In [16]:
```
# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")
```

```
status: ['for_sale' 'ready_to_build']
state: ['Puerto Rico' 'Virgin Islands' 'Massachusetts' 'Connecticut'
 'New Hampshire' 'Vermont' 'New Jersey' 'New York' 'South Carolina'
 'Tennessee' 'Rhode Island' 'Virginia']
```

In [17]:
```
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
status: [0 1]
state: [ 5 10  1  0  2  9  3  4  7  8  6 11]
```

Remove Outliers using IQR

In [18]:
```
# Print the number of rows before outliers removal
df.shape
```

Out[18]:
```
(100000, 7)
```

In [19]:
```
# define a function to remove outliers using IQR
def remove_outliers_iqr(df, columns):
    for col in columns:
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)
        iqr = q3 - q1
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr
        df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
    return df

# specify the columns to remove outliers from
columns_to_check = ['bed', 'bath', 'acre_lot', 'house_size']

# call the function to remove outliers using IQR
df_clean = remove_outliers_iqr(df, columns_to_check)

# print the resulting dataframe
df_clean.head()
```

Out[19]:

	status	bed	bath	acre_lot	state	house_size	price
0	0	3.0	2.0	0.12	5	920.0	105000.0
1	0	4.0	2.0	0.08	5	1527.0	80000.0
3	0	4.0	2.0	0.10	5	1800.0	145000.0
7	0	3.0	2.0	0.08	5	1050.0	71600.0
10	0	3.0	2.0	13.39	5	1106.0	89000.0

In [20]:
```
df_clean.shape
```

Out[20]:
```
(50558, 7)
```

Correlation Heatmap

In [21]:
```
#Correlation Heatmap
plt.figure(figsize=(20, 16))
sns.heatmap(df_clean.corr(method='pearson'), fmt='.2g', annot=True)
```

Out[21]: <AxesSubplot:>

In [23]:
```
df_clean.drop(columns='bath', inplace=True)
df_clean.head()
```

Out[23]:

	status	bed	acre_lot	state	house_size	price
0	0	3.0	0.12	5	920.0	105000.0
1	0	4.0	0.08	5	1527.0	80000.0
3	0	4.0	0.10	5	1800.0	145000.0
7	0	3.0	0.08	5	1050.0	71600.0
10	0	3.0	13.39	5	1106.0	89000.0

Train Test Split

In [24]:
```
X = df_clean.drop('price', axis=1)
y = df_clean['price']
```

In [25]:
```
#test size 20% and train size 80%
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_state=0)
```

Decision Tree Regressor

In [26]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV

# Create a DecisionTreeRegressor object
dtree = DecisionTreeRegressor()

# Define the hyperparameters to tune and their values
param_grid = {
    'max_depth': [2, 4, 6, 8],
    'min_samples_split': [2, 4, 6, 8],
    'min_samples_leaf': [1, 2, 3, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
    'random_state': [0, 42]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(dtree, param_grid, cv=5, scoring='neg_mean_squared_error')

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 8, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2,
'random_state': 42}
```

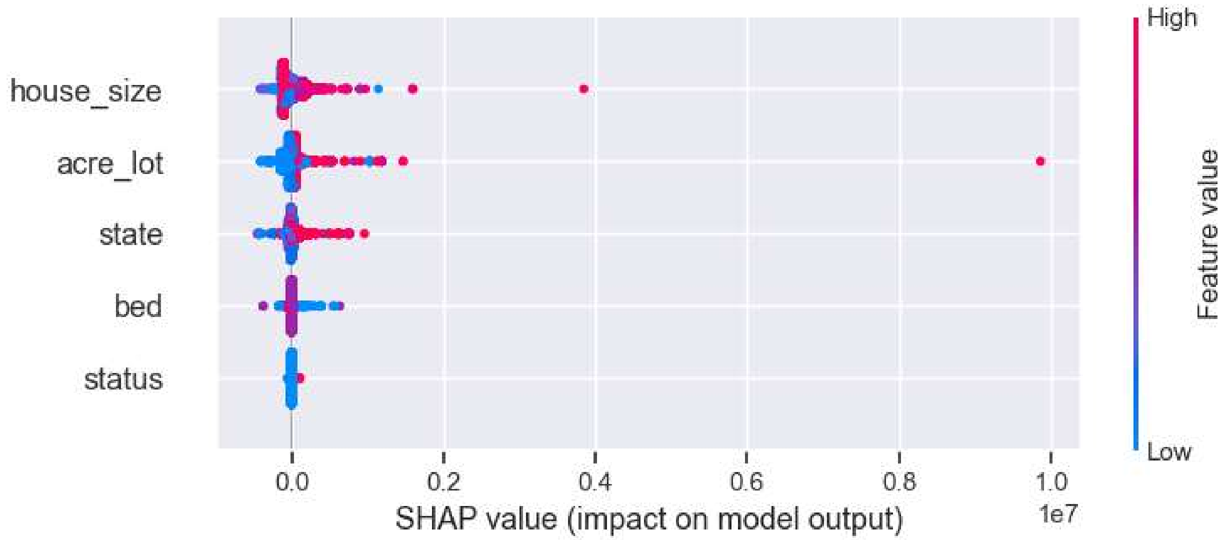
In [27]:
```
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(random_state=42, max_depth=8, max_features='auto', min_sam
dtree.fit(X_train, y_train)
```

Out[27]:
```
DecisionTreeRegressor(max_depth=8, max_features='auto', random_state=42)
```

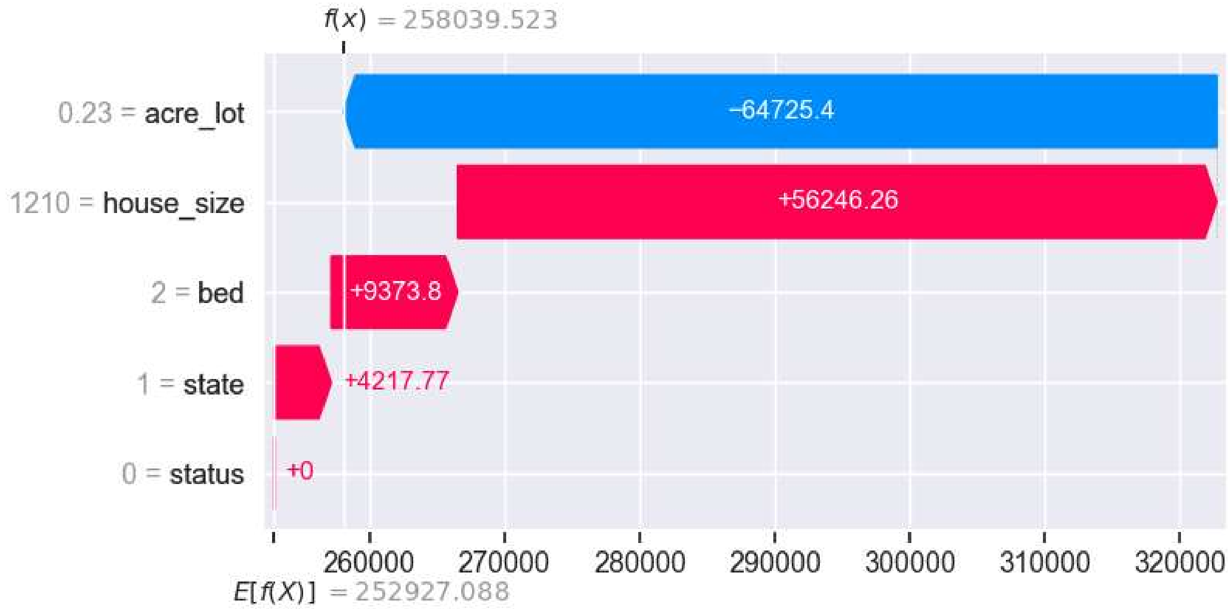
In [28]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = dtree.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 99905.67946085877
MAPE is 1.3304384099501423
MSE is 50681781752.41356
R2 score is 0.5766792701837287
RMSE score is 225126.14631004893
```

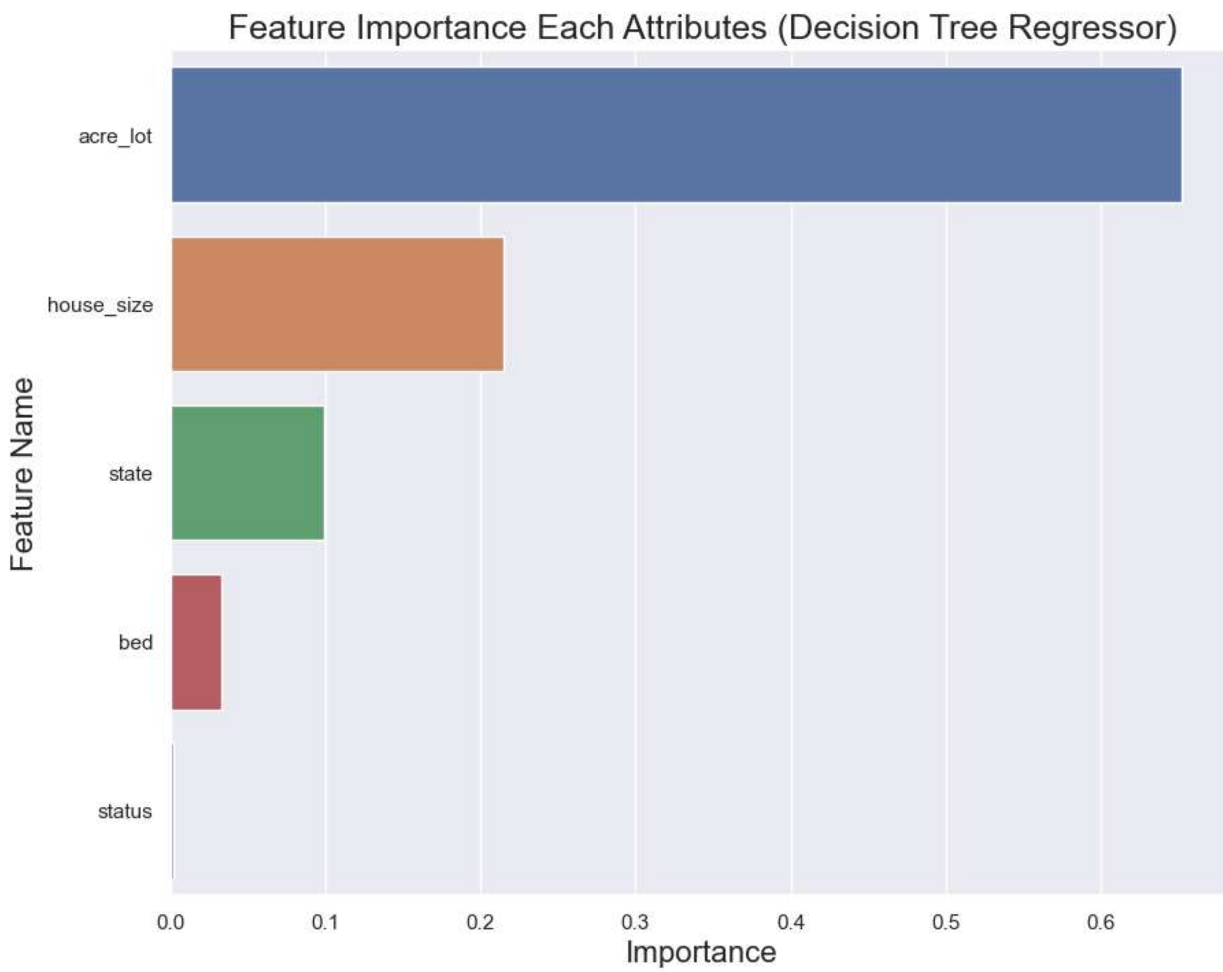
In [29]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [30]:
```
explainer = shap.Explainer(dtree, X_test)
shap_values = explainer(X_test)
shap.plots.waterfall(shap_values[0])
```



In [31]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (Decision Tree Regressor)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



AdaBoost Regressor

In [33]:
```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import GridSearchCV

# Create an AdaBoost Regressor object
ada = AdaBoostRegressor()

# Define the hyperparameter grid
param_grid = {'n_estimators': [50, 100, 150],
              'learning_rate': [0.01, 0.1, 1]}

# Create a GridSearchCV object
grid = GridSearchCV(ada, param_grid, cv=5, scoring='neg_mean_absolute_error')

# Fit the GridSearchCV object to the training data
grid.fit(X_train, y_train)

# Print the best hyperparameters
print("Best hyperparameters: ", grid.best_params_)
```

```
Best hyperparameters:  {'learning_rate': 0.01, 'n_estimators': 100}
```

In [34]:
```
from sklearn.ensemble import AdaBoostRegressor
ada = AdaBoostRegressor(n_estimators=100, learning_rate=0.01, random_state=0)
ada.fit(X_train, y_train)
```

Out[34]:
```
AdaBoostRegressor(learning_rate=0.01, n_estimators=100, random_state=0)
```

In [35]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = ada.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 142898.84136857474
MAPE is 1.7810437954933251
MSE is 88172321437.5971
R2 score is 0.26353868846015893
RMSE score is 296938.24515814247
```

In [36]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": ada.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (AdaBoost Regressor)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```