

Test Project Data Science Vacancy: Housing Price

Prepared by Melina BASHA

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

Overview of the dataset

```
In [2]: housing = pd.read_csv("housing.csv")
housing.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
```

```
In [3]: housing.shape
```

```
Out[3]: (904966, 10)
```

```
In [4]: housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 904966 entries, 0 to 904965
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
 --- 
 0   status            904966 non-null   object 
 1   bed               775126 non-null   float64
 2   bath              791082 non-null   float64
 3   acre_lot          638324 non-null   float64
 4   city              904894 non-null   object 
 5   state             904966 non-null   object 
 6   zip_code          904762 non-null   float64
 7   house_size        612080 non-null   float64
 8   prev_sold_date   445865 non-null   object 
 9   price             904895 non-null   float64
dtypes: float64(6), object(4)
memory usage: 69.0+ MB
```

As seen above the housing dataset has 9 variables. The target variable is price and the other variables are its features. Among these 9 features we can distinguish 5 numerical variables and 4 categorical variables. Note that there are some variables that are in numerical format, but actually should be categorial variables, like zip code.

```
In [5]: print("Numerical variables in dataset: {housing.select_dtypes(exclude = ['object']).columns.tolist()}")
print("Categorical variables in dataset: {housing.select_dtypes(include = ['object']).columns.tolist()}"
```

```
Numerical variables in dataset: ['bed', 'bath', 'acre_lot', 'zip_code', 'house_size', 'price']
Categorical variables in dataset: ['status', 'city', 'state', 'prev_sold_date']
```

Below are presented statistics indicators for numerical and categorical variables (in separated tables).

```
In [6]: housing.describe()
```

```
Out[6]:      bed      bath    acre_lot    zip_code    house_size      price
count  775126.000000  791082.000000  638324.000000  904762.000000  6.120800e+05  9.048950e+05
mean    3.332190      2.484236     17.317292    6519.464582  2.138437e+03  8.774382e+05
std     2.065312      1.931622     970.707378   3816.713093  3.046600e+03  2.457698e+06
min     1.000000      1.000000     0.000000    601.000000  1.000000e+02  0.000000e+00
25%    2.000000      2.000000     0.110000    2908.000000  1.132000e+03  2.685000e+05
50%    3.000000      2.000000     0.290000    6811.000000  1.650000e+03  4.750000e+05
75%    4.000000      3.000000     1.150000    8854.000000  2.495000e+03  8.300000e+05
max   123.000000     198.000000  100000.000000  99999.000000  1.450112e+06  8.750000e+08
```

```
In [7]: housing.describe(include = 'object')
```

```
Out[7]:
```

	status	city	state	prev_sold_date
count	904966	904894	904966	445865
unique	2	2487	18	9870
top	for_sale	New York City	New Jersey	2018-07-25
freq	903373	47502	231958	317

Data Cleaning

In order of conducting Exploratory Data Analysis and Model Development we need to be sure that the dataset can be usable, otherwise it will produce problems. So we have to do checks related to missing values in the dataset and ways of handling with it. Based on the variables we will decide which of them to keep for this project, which to drop, which to fill with imputations mode.

```
In [8]: values_missing = housing.isna().sum()*100/len(housing)
print('Percentage Missing Values %')
values_missing
```

```
Out[8]:
```

	Percentage Missing Values %
status	0.000000
bed	14.347500
bath	12.584340
acre_lot	29.464311
city	0.007956
state	0.000000
zip_code	0.022542
house_size	32.364310
prev_sold_date	50.731298
price	0.007846

dtype: float64

As seen above there are variables that have missing values. First we will remove columns city, zip_code and prev_sold_date, because they will be not used.

```
In [9]: housing = housing.drop(['city', "zip_code", "prev_sold_date"], axis=1)
```

From Out[8] we can see that the target variables price has 0.78% missing values, so it is better to just remove the missing values from it.

```
In [10]: housing = housing.drop(housing[housing['price'].isnull()].index)
```

Also we will remove the rows that have 2 or more missing values.

```
In [11]: housing = housing[~(housing.isna().sum(axis=1) >= 2)]
```

Below we will make a check to see the state of missing values in the variables.

```
In [12]: ((housing.isna().sum() / len(housing)) * 100).sort_values(ascending=False)
```

```
Out[12]:
```

acre_lot	24.884798
house_size	15.764261
bed	0.576647
bath	0.408540
status	0.000000
state	0.000000
price	0.000000

dtype: float64

Also we will remove the missing values from variables bed and bath as they are < 1 %.

```
In [13]: housing = housing.drop(housing[housing['bed'].isnull()].index, axis=0)
housing = housing.drop(housing[housing['bath'].isnull()].index, axis=0)
```

Now we will do a check related to status variable and we will remove it as it has only one value "for_sale".

```
In [14]: housing["status"].value_counts()
```

```
Out[14]:
```

for_sale	700912
Name:	status, dtype: int64

```
In [15]: housing = housing.drop("status", axis=1)
```

For variables like acre_lot and house_size we will use Median imputation to handle missing values. Median is more recommended to be used with numeric variables.

```
In [16]: housing['acre_lot'].fillna(housing['acre_lot'].median(), inplace=True)
housing['house_size'].fillna(housing['house_size'].median(), inplace=True)
```

```
In [17]: housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 700912 entries, 0 to 904965
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   bed         700912 non-null   float64
 1   bath        700912 non-null   float64
 2   acre_lot    700912 non-null   float64
 3   state       700912 non-null   object 
 4   house_size  700912 non-null   float64
 5   price       700912 non-null   float64
dtypes: float64(5), object(1)
memory usage: 37.4+ MB
```

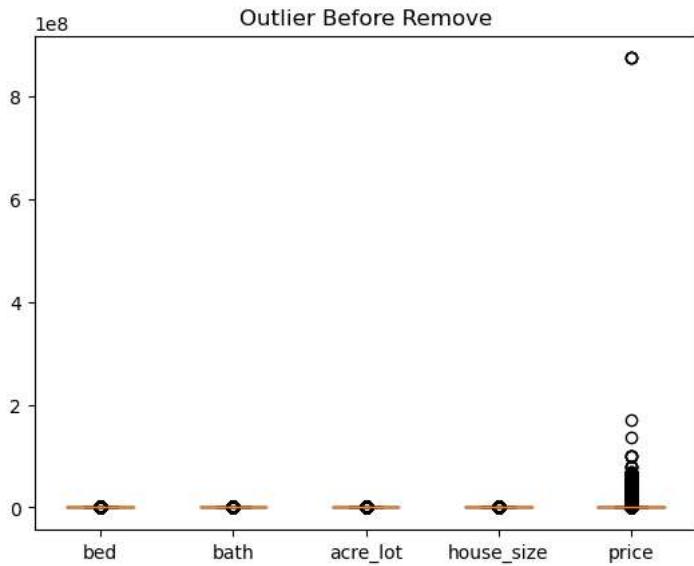
We will make another check if we have missing values.

```
In [18]: housing.isnull().sum()
```

```
Out[18]: bed      0
bath     0
acre_lot 0
state    0
house_size 0
price    0
dtype: int64
```

Data cleaning has not finished yet. We need to detect for outliers and remove them.

```
In [19]: var_num = ['bed','bath','acre_lot','house_size','price']
plt.boxplot(housing[var_num])
plt.xticks([1, 2, 3, 4, 5], var_num)
plt.title('Outlier Before Remove')
plt.show()
print(f'Total Row With Outlier: {housing.shape[0]}')
```



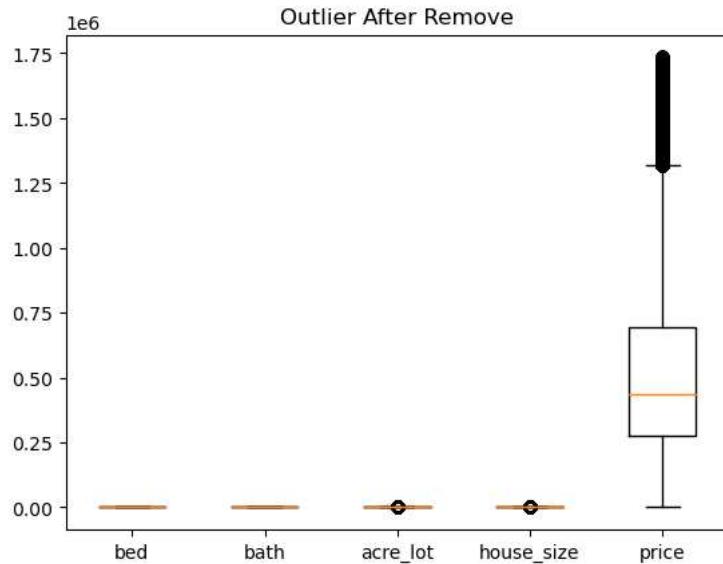
Total Row With Outlier: 700912

As seen outliers are detected for price in the visualization above, so we will remove.

```
In [20]: Q1 = housing[var_num].quantile(0.25)
Q3 = housing[var_num].quantile(0.75)
IQR = Q3 - Q1

housing = housing[~((housing[var_num] < (Q1 - 1.5 * IQR)) | (housing[var_num] > (Q3 + 1.5 * IQR))).any(axis=1)]
```

```
In [21]: var_num = ['bed','bath','acre_lot','house_size','price']
plt.boxplot(housing[var_num])
plt.xticks([1, 2, 3, 4, 5], var_num)
plt.title('Outlier After Remove')
plt.show()
print(f'Total Row Without Outlier: {housing.shape[0]}')
```



Exploratory Data Analysis and Visualizations

In [22]: `housing.describe()`

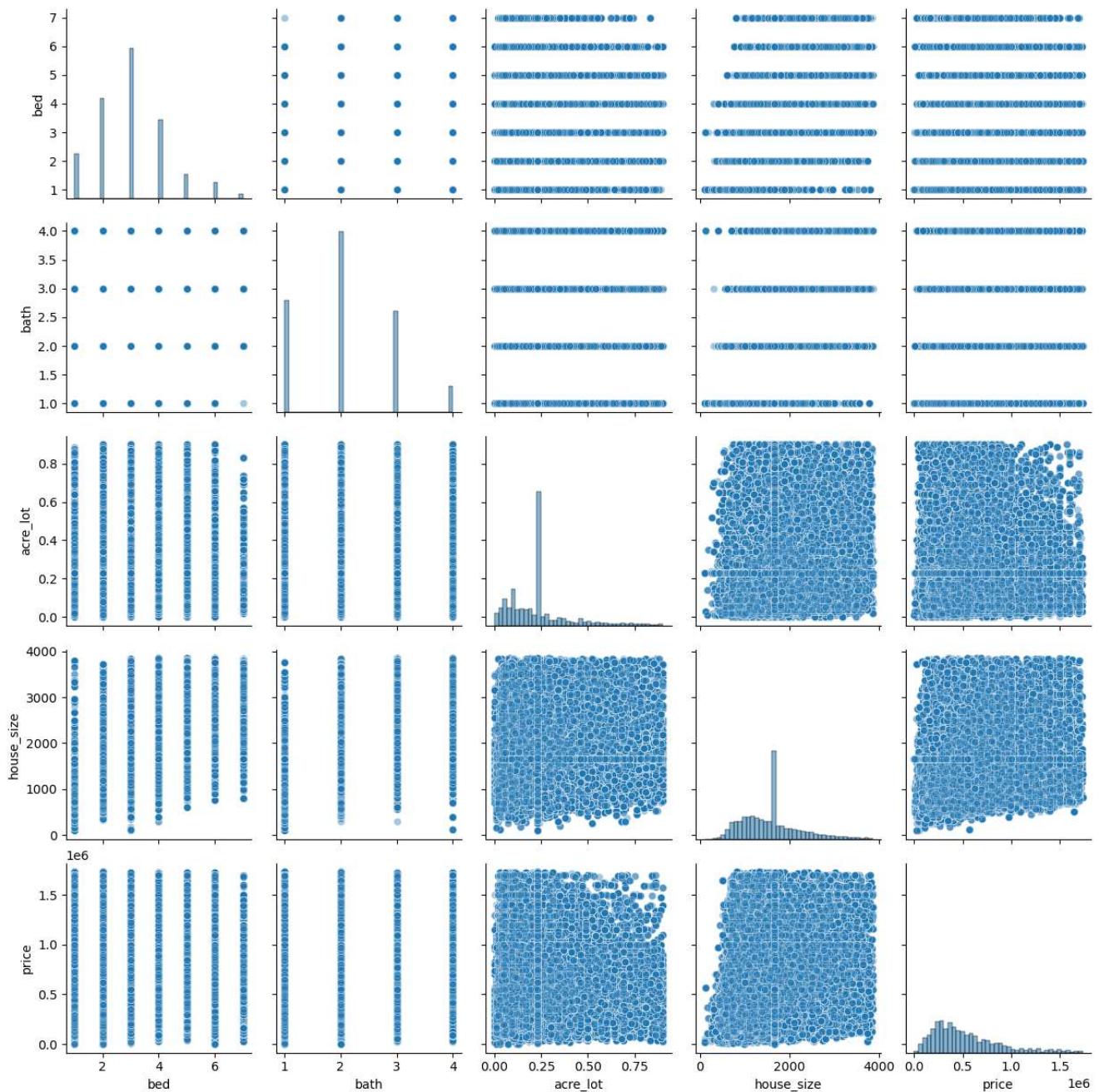
	bed	bath	acre_lot	house_size	price
count	494209.000000	494209.000000	494209.000000	494209.000000	4.942090e+05
mean	3.007060	2.098831	0.226893	1569.741067	5.217125e+05
std	1.263706	0.863229	0.162786	631.238299	3.400852e+05
min	1.000000	1.000000	0.000000	100.000000	1.000000e+00
25%	2.000000	1.000000	0.110000	1100.000000	2.749000e+05
50%	3.000000	2.000000	0.230000	1600.000000	4.350000e+05
75%	4.000000	3.000000	0.230000	1808.000000	6.919650e+05
max	7.000000	4.000000	0.900000	3852.000000	1.735000e+06

Below it is presented the pairplot of each of 2 variables which show the relationship between them. As presented there is no indication for linear regression between the variables.

In [23]: `sns.pairplot(housing,
kind='scatter',
plot_kws={'alpha':0.4},
diag_kws={'alpha':0.55, 'bins':40})`

Out[23]: `<seaborn.axisgrid.PairGrid at 0x17270eddf90>`

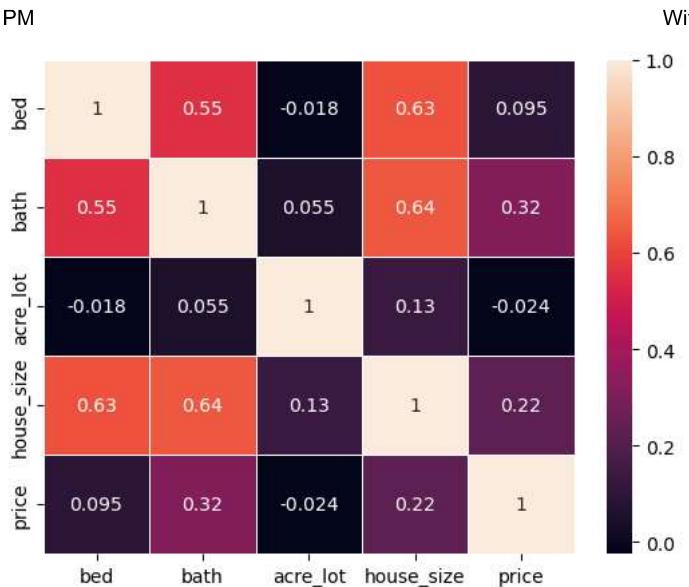
With_duplicates



Below it is presented the correlation matrix for the variables. Considering price as target, there is no evidence for strong correlation between its features.

In [24]: `sns.heatmap(housing.corr(), annot=True, linewidths=0.5);`

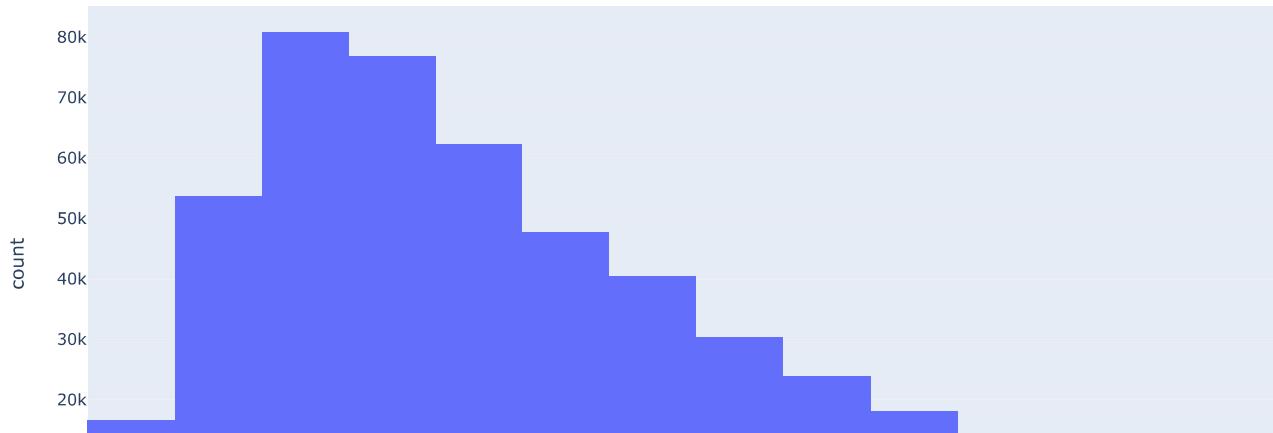
```
C:\Users\HP\AppData\Local\Temp\ipykernel_11424\1901337508.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
sns.heatmap(housing.corr(), annot=True, linewidths=0.5);
```



Below it is presented the Distribution of Price.

```
In [25]: fig = px.histogram(housing, x="price", nbins=25, template="plotly")
fig.update_layout(title="Distribution of Price")
fig.show()
```

Distribution of Price

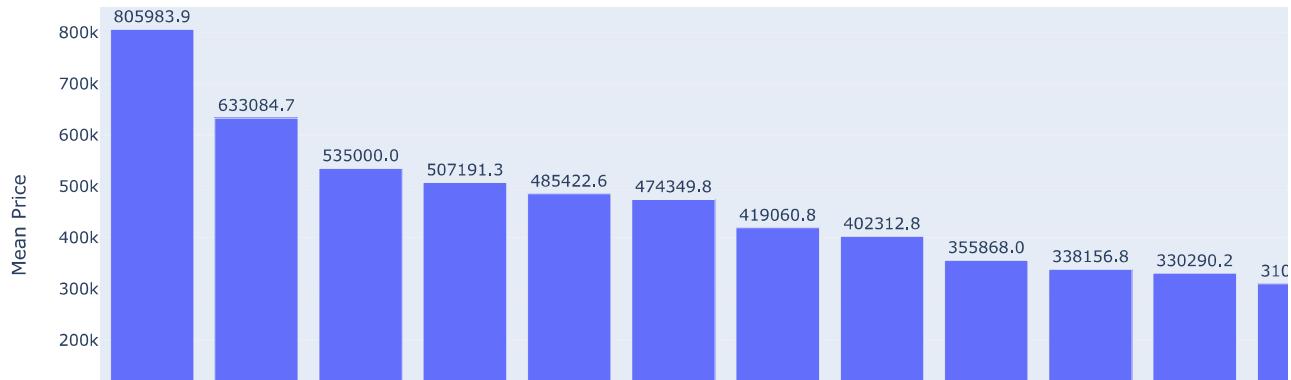


Below it is presented the chart of Top 10 States with Highest Mean Price

```
In [26]: df_mean = housing.groupby('state')['price'].mean().reset_index()
df_mean_sort = df_mean.sort_values(by='price', ascending=False)

fig = px.bar(df_mean_sort, x='state', y='price',
             title='Top 10 States with Highest Mean Price',
             labels={'state': 'State', 'price': 'Mean Price'})
fig.update_traces(texttemplate='%{y:.1f}', textposition='outside')
fig.show()
```

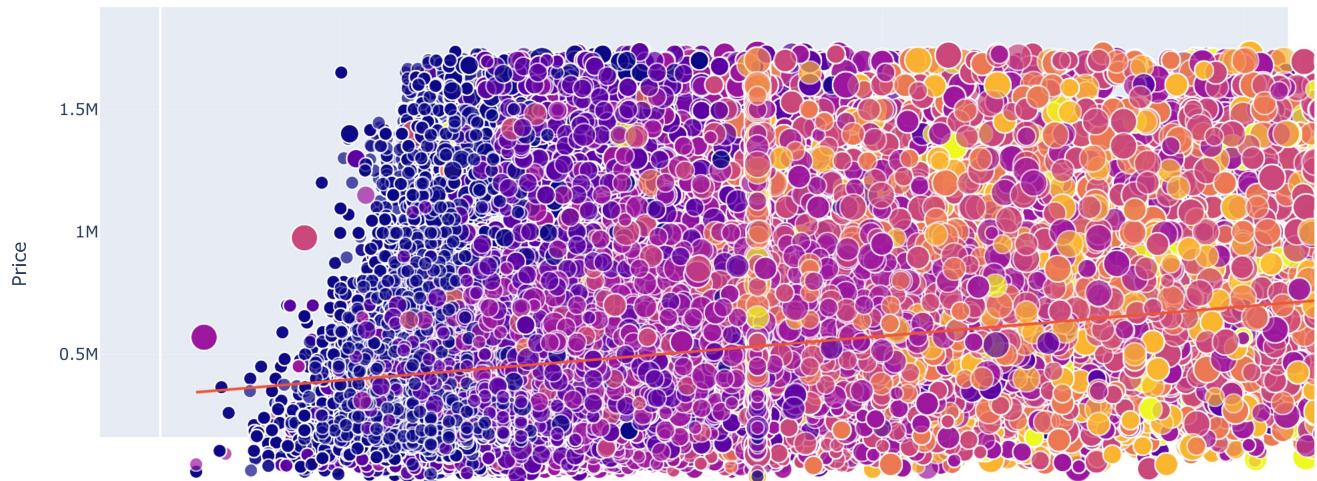
Top 10 States with Highest Mean Price



Below it is presented the chart of House Size, Bed, Bath to Price.

```
In [27]: fig = px.scatter(housing, x='house_size', y='price', color='bed', size='bath', trendline='ols')
fig.update_layout(title='House Size vs Price',
                  xaxis_title='House Size',
                  yaxis_title='Price')
fig.show()
```

House Size vs Price



Keeping the unique values of state, removing the ones that have less than 50 counts and fitting and transforming the "state" column to obtain numeric labels.

```
In [28]: housing["state"].value_counts()
```

```
Out[28]: New Jersey      153223
          Massachusetts   103135
          New York         86723
          Connecticut     53410
          Rhode Island    21126
          New Hampshire   18140
          Puerto Rico      17488
          Pennsylvania     13579
          Vermont          13357
          Maine            11804
          Delaware         1780
          Virgin Islands   390
          Georgia          48
          Wyoming          3
          West Virginia    3
          Name: state, dtype: int64
```

```
In [29]: housing = housing.drop(housing[housing["state"].map(housing["state"].value_counts()) < 50]["state"].index)
```

```
In [30]: housing["state"].unique()
```

```
Out[30]: array(['Puerto Rico', 'Virgin Islands', 'Massachusetts', 'Connecticut',
       'New Jersey', 'New Hampshire', 'Vermont', 'New York',
       'Rhode Island', 'Maine', 'Pennsylvania', 'Delaware'], dtype=object)
```

```
In [31]: from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
housing['state_numeric'] = label_encoder.fit_transform(housing['state'])
housing = housing.drop("state", axis=1)
housing
```

```
Out[31]:   bed  bath  acre_lot  house_size      price  state_numeric
0      3.0    2.0     0.12      920.0  105000.0          8
1      4.0    2.0     0.08     1527.0  80000.0          8
2      2.0    1.0     0.15      748.0  67000.0          8
3      4.0    2.0     0.10     1800.0  145000.0          8
4      6.0    2.0     0.05     1655.0  65000.0          8
...
904958  1.0    1.0     0.23      700.0  245000.0          6
904959  1.0    1.0     0.23      720.0  265000.0          6
904960  2.0    2.0     0.23     1200.0  399000.0          6
904962  2.0    2.0     0.23     1200.0  299000.0          6
904965  3.0    3.0     0.23     1677.0  850000.0          6
```

494155 rows × 6 columns

```
In [32]: housing.describe()
```

```
Out[32]:      bed      bath    acre_lot    house_size      price  state_numeric
count  494155.000000  494155.000000  494155.000000  494155.000000  4.941550e+05  494155.000000
mean    3.00686        2.098690     0.226893     1569.560421   5.217166e+05  4.566901
std     1.26362        0.863148     0.162794     631.011452   3.401018e+05  2.377142
min     1.00000        1.000000     0.000000     100.000000   1.000000e+00  0.000000
25%    2.00000        1.000000     0.110000     1100.000000   2.749000e+05  3.000000
50%    3.00000        2.000000     0.230000     1600.000000   4.350000e+05  5.000000
75%    4.00000        3.000000     0.230000     1807.000000   6.930000e+05  6.000000
max    7.00000        4.000000     0.900000     3852.000000   1.735000e+06  11.000000
```

Training the Model with multivariable regression using Scikit Learn

```
In [33]: from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import math
```

Standardizing data

```
In [34]: housing['house_size'] = StandardScaler().fit_transform(housing['house_size'].values.reshape(len(housing), 1))
housing['price'] = StandardScaler().fit_transform(housing['price'].values.reshape(len(housing), 1))
```

```
In [35]: housing['bed'] = MinMaxScaler().fit_transform(housing['bed'].values.reshape(len(housing), 1))
housing['bath'] = MinMaxScaler().fit_transform(housing['bath'].values.reshape(len(housing), 1))
housing['acre_lot'] = MinMaxScaler().fit_transform(housing['acre_lot'].values.reshape(len(housing), 1))
```

Splitting the data

X are the predictores, and y is the output. What we want to do is create a model that will take in the values in the X variable and predict y with a linear regression algorithm and random forest regressor algorithm. We will use the SciKit Learn library to create the model.

In the model we will use even categororical variable state, by using One-hot encoding, in order to use them as dummy variables in the model.

```
In [36]: X = housing[['bed', 'bath', 'acre_lot', 'house_size', 'state_numeric']]
y = housing['price']

X = pd.get_dummies(X, columns=['state_numeric'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Linear Regression

```
In [37]: lm = LinearRegression()
```

```
In [38]: lm.fit(X_train, y_train)
```

```
Out[38]: ▾ LinearRegression
LinearRegression()
```

```
In [39]: lm.coef_
```

```
Out[39]: array([-0.37218256,  1.130083 ,  0.0772693 ,  0.1351292 , -0.37376231,
   -0.41614545, -0.13809235,  0.62272378, -0.0503748 ,  0.09431653,
   1.23621904, -0.22260922, -0.49691699, -0.0396621 , -0.29976771,
   0.08407156])
```

```
In [40]: lm.score(X, y)
```

```
Out[40]: 0.39915832588727773
```

```
In [41]: # The coefficients in a dataframe
cdf = pd.DataFrame(lm.coef_, X.columns, columns=['Coef'])
print(cdf)
```

	Coef
bed	-0.372183
bath	1.130083
acre_lot	0.077269
house_size	0.135129
state_numeric_0	-0.373762
state_numeric_1	-0.416145
state_numeric_2	-0.138092
state_numeric_3	0.622724
state_numeric_4	-0.050375
state_numeric_5	0.094317
state_numeric_6	1.236219
state_numeric_7	-0.222609
state_numeric_8	-0.496917
state_numeric_9	-0.039662
state_numeric_10	-0.299768
state_numeric_11	0.084072

```
In [42]: predictions = lm.predict(X_test)
```

```
In [43]: print('Mean Absolute Error:', mean_absolute_error(y_test, predictions))
print('Mean Squared Error:', mean_squared_error(y_test, predictions))
print('Root Mean Squared Error:', math.sqrt(mean_squared_error(y_test, predictions)))
```

```
Mean Absolute Error: 0.5594266344647285
Mean Squared Error: 0.5995482472664143
Root Mean Squared Error: 0.7743050091962561
```

```
In [45]: print(predictions)
```

```
[-0.19227954 -1.07777078 -0.58209547 ... -0.15342556 -0.10717391
 -1.05121261]
```

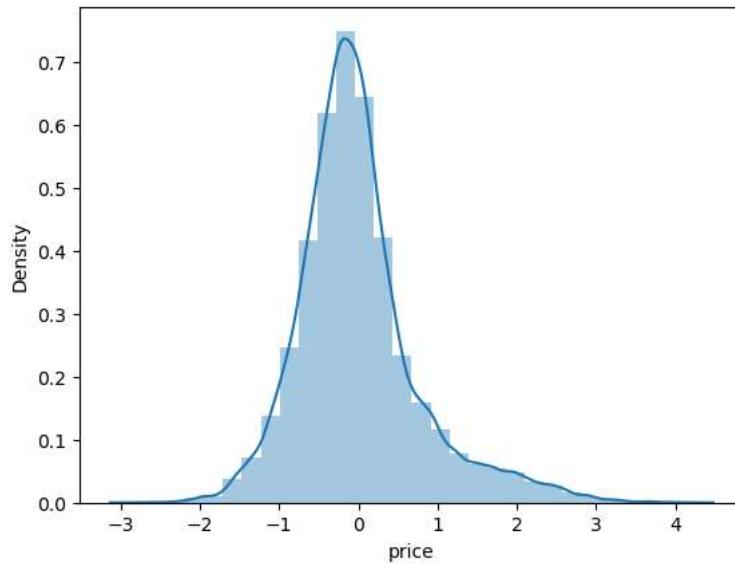
Residuals

Distribution plot of the residuals of the model's predictions. They should be normally distributed.

```
In [46]: residuals = y_test-predictions
sns.distplot(residuals, bins=30)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_11424\339810671.py:2: UserWarning:
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```

Out[46]: <Axes: xlabel='price', ylabel='Density'>



The Linear Regression shows a value of R² in the value 39.9% (that is not a very good value). On the other hand the value of MAE and MSE are near 0, which is a very good value.

Random Forest Regressor

```
In [47]: # creating a Random Forest model and train it using training data
model_RF = RandomForestRegressor(n_estimators=100, random_state=42)
model_RF.fit(X_train, y_train)

# Make predictions using testing data
y_pred = model_RF.predict(X_test)

# calculate average error value using MSE metrics
mse_RF = mean_squared_error(y_test, y_pred)
rmse_RF = mean_squared_error(y_test, y_pred, squared=False)
mae_RF = mean_absolute_error(y_test, y_pred)
r2_RF = r2_score(y_test, y_pred)
```

```
In [48]: result = {'Random Forest': {'MSE': mse_RF, 'RMSE': rmse_RF, 'MAE': mae_RF, 'R^2': r2_RF}}
```

```
In [49]: data = pd.DataFrame.from_dict(result, orient='index')
data = data.applymap(lambda x: f'{x:.2f}')
print(data)
```

	MSE	RMSE	MAE	R ²
Random Forest	0.14	0.37	0.16	0.86

The Random Forest Regressor shows that all indicators MSE, RMSE, MAE, R² are very good and this algorithm is appropriate for the relationship of the variables.

Training the model with multivariable regression using OLS

```
In [50]: import statsmodels.api as sm
X = sm.add_constant(X_train)
model = sm.OLS(y_train, X)
model_fit = model.fit()
print(model_fit.summary())
```

OLS Regression Results

```
=====
Dep. Variable:      price   R-squared:          0.399
Model:              OLS     Adj. R-squared:      0.399
Method:             Least Squares   F-statistic:       1.749e+04
Date:              Sat, 21 Oct 2023   Prob (F-statistic):    0.00
Time:                15:39:58     Log-Likelihood:   -4.6035e+05
No. Observations:  395324     AIC:            9.207e+05
Df Residuals:      395308     BIC:            9.209e+05
Df Model:           15
Covariance Type:   nonrobust
```

	coef	std err	t	P> t	[0.025	0.975]
const	-0.5575	0.005	-106.549	0.000	-0.568	-0.547
bed	-0.3722	0.008	-46.622	0.000	-0.388	-0.357
bath	1.1301	0.006	194.094	0.000	1.119	1.141
acre_lot	0.0773	0.007	10.612	0.000	0.063	0.092
house_size	0.1351	0.002	73.699	0.000	0.132	0.139
state_numeric_0	-0.4202	0.005	-78.992	0.000	-0.431	-0.410
state_numeric_1	-0.4626	0.019	-23.980	0.000	-0.500	-0.425
state_numeric_2	-0.1846	0.008	-21.944	0.000	-0.201	-0.168
state_numeric_3	0.5763	0.005	123.131	0.000	0.567	0.585
state_numeric_4	-0.0968	0.007	-13.517	0.000	-0.111	-0.083
state_numeric_5	0.0479	0.004	10.655	0.000	0.039	0.057
state_numeric_6	1.1898	0.005	245.409	0.000	1.180	1.199
state_numeric_7	-0.2691	0.008	-33.787	0.000	-0.285	-0.253
state_numeric_8	-0.5434	0.007	-74.821	0.000	-0.558	-0.529
state_numeric_9	-0.0861	0.007	-12.693	0.000	-0.099	-0.073
state_numeric_10	-0.3462	0.008	-43.480	0.000	-0.362	-0.331
state_numeric_11	0.0376	0.040	0.932	0.351	-0.041	0.117

```
=====
Omnibus:            68868.226   Durbin-Watson:        1.997
Prob(Omnibus):      0.000     Jarque-Bera (JB):    146406.691
Skew:                  1.033     Prob(JB):            0.00
Kurtosis:               5.149     Cond. No.        1.57e+15
=====
```

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The smallest eigenvalue is 2.5e-25. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

In []: