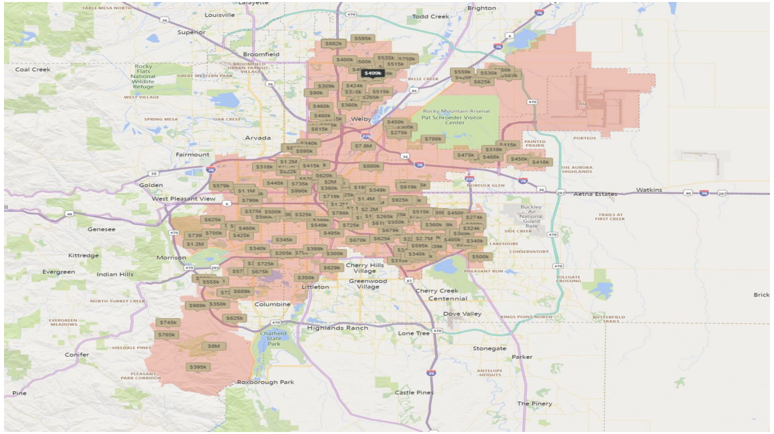
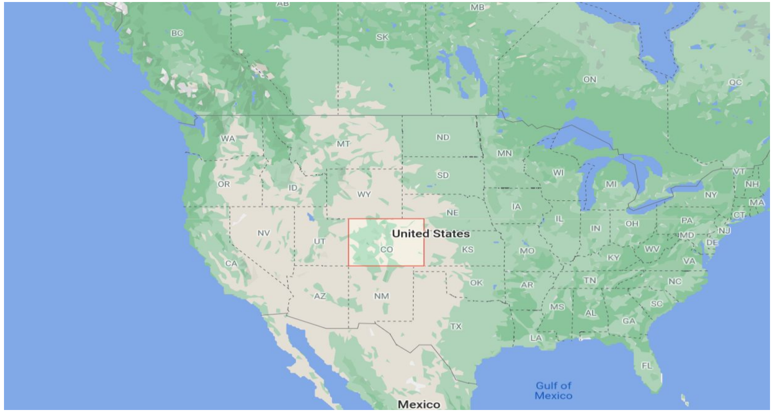
Denver,Colorado,USA House Price Prediction With Machine Learning in Python

ISTANBUL DATA SCIENCE ACADEMY

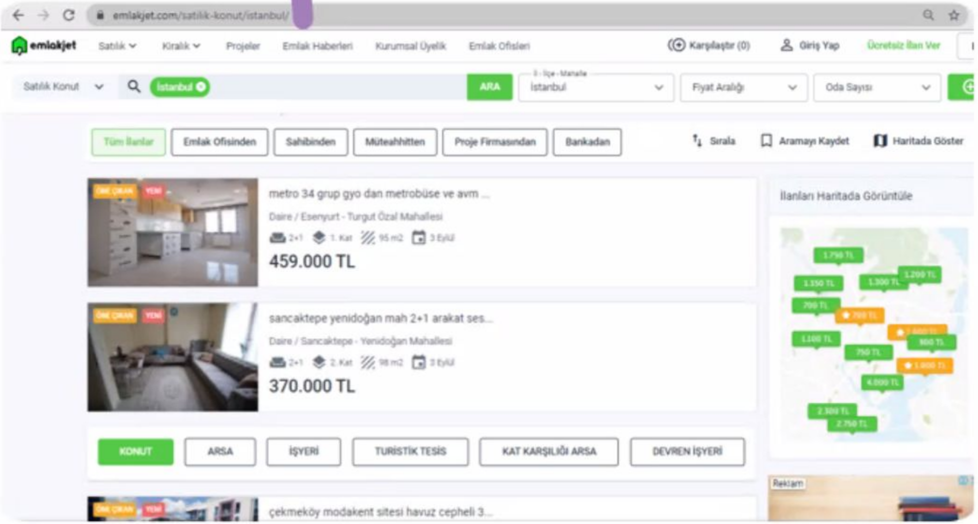




Serdar Celebi

INTRODUCTION

- **FOCUSING**: Make a Regression model that shows a good estimation on prices.
- **OBJECTIVE** : What is shown in emlakjet dataset or what features are crucial in dataset?
- **GOALS**: Finding the fit model on the prices with the help of distinct columns.



METHODOLOGIES AND TOOLS

- To gather dataset, we made some web scraping from emlakjet by applying BeatifulSoup model .
- After that approach, we used libraries and their coding tools such as;

-matplotlib, seaborn (to visualize and make users closer)

-Linear Regression tools such as OLS,Ridge, ElasticNet Regression, Lasso etc.

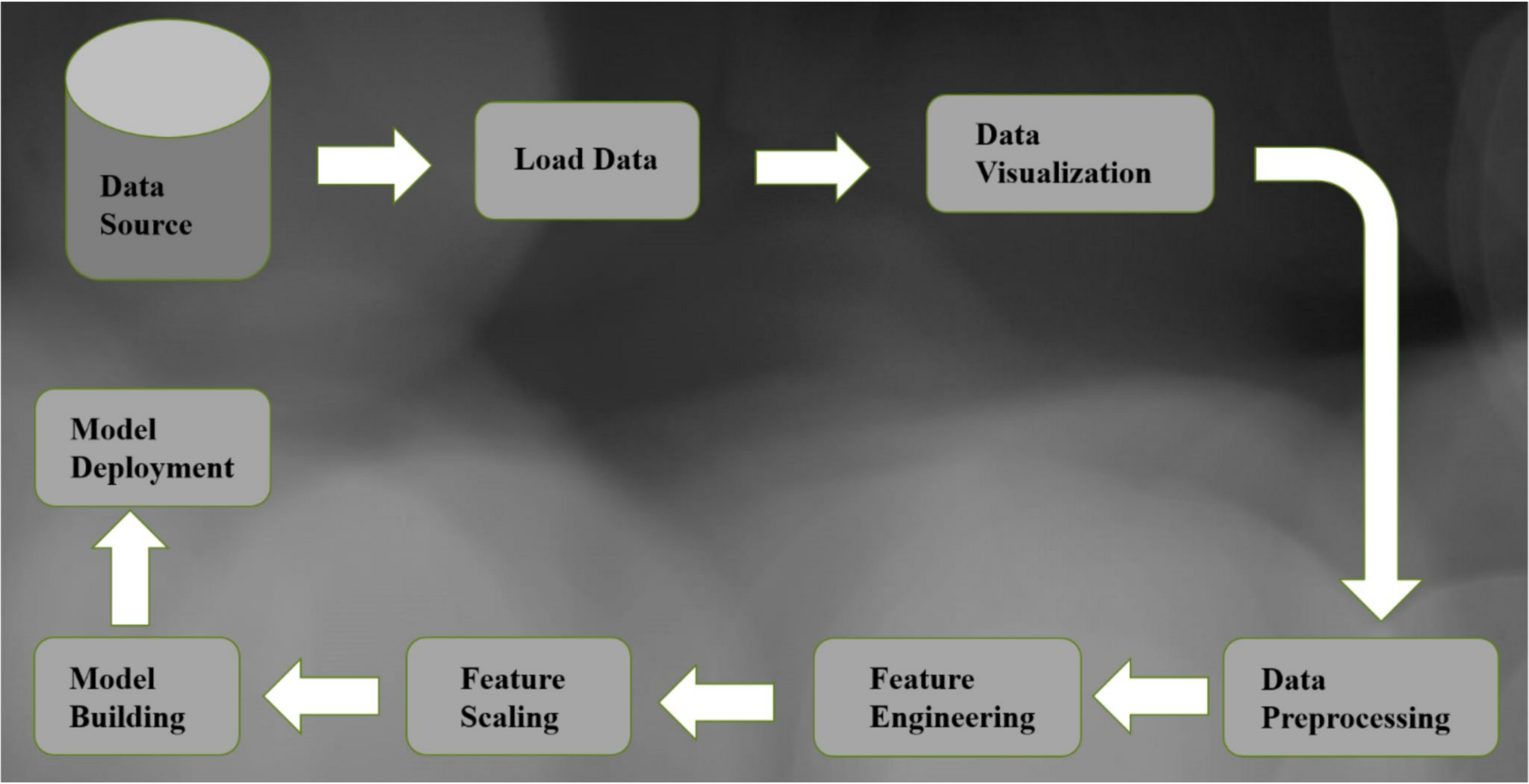
Gathering the data using BeautifulSoup

Introduction

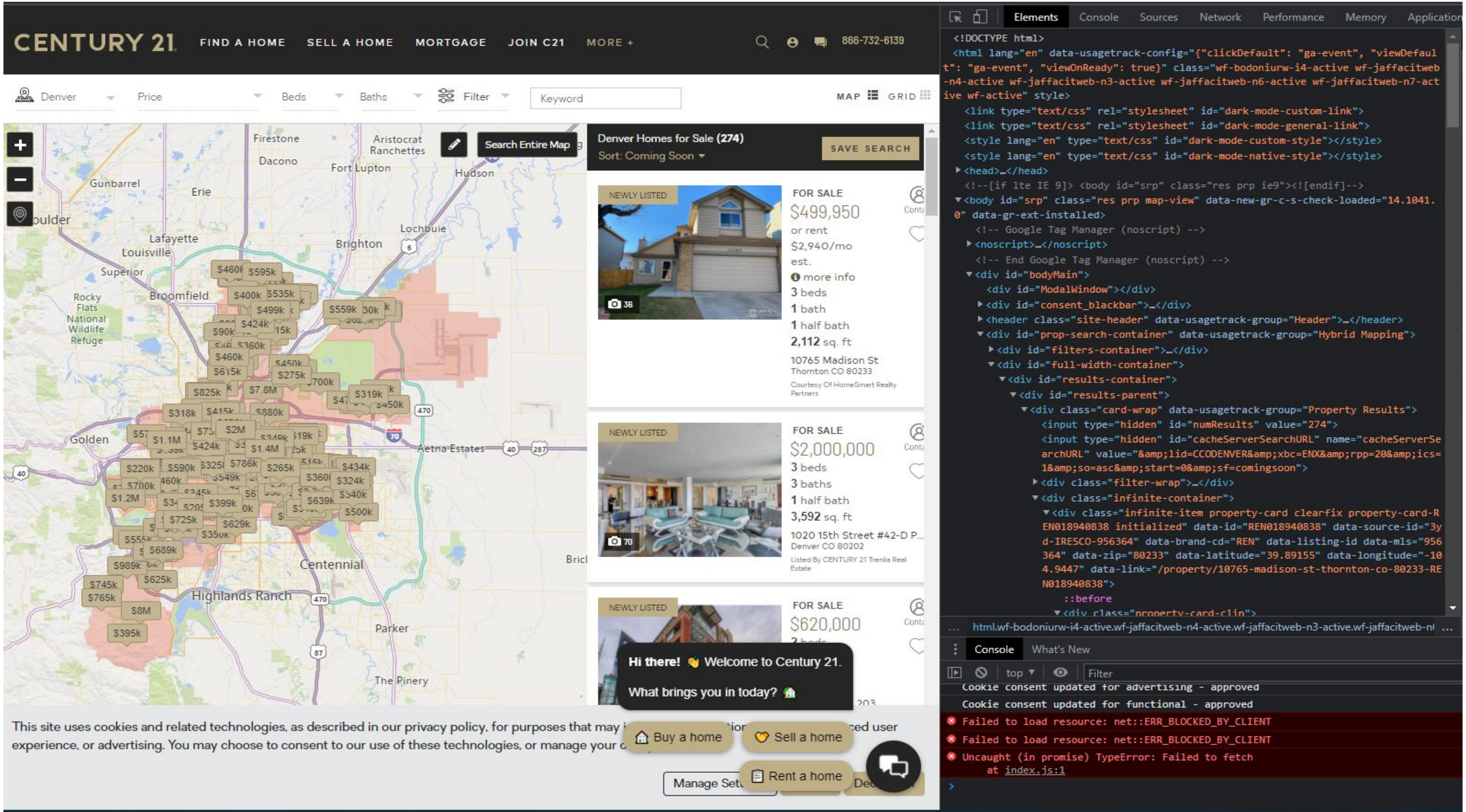
Business need: Predicting movie scores on IMDB

Solution: In order to predict the Movie ratings we will develop a linear regression model.

Web Scraping → Data Cleaning & EDA → Basic Model → Model with all numeric features → Model with dummy variables → Regularization


Data
Source
Load Data
Data
Visualization
Data
Preprocessing
Feature
Engineering
Feature
Scaling
Model
Building
Model
Deployment

Get the data- Web scraping



Get the data- Web scraping

```
properties = soup.find_all("div", {"class": "property-card-primary-info"})
properties
```

```
[12] len(properties)
```

```
20
```

```
properties[0]
```

```
[14] all_info = []

for i in properties:
  info = {}
  info['Title'] = i.find("div", {"class": "pdp-listing-type sale"}).text
  info['Price'] = i.find("a", {"class":"listing-price"}).text.strip()
  info['Beds'] = i.find("div", {"class": "property-beds"}).text.strip()
  info['Bath'] = i.find("div", {"class": "property-baths"}).text.strip()
  info['SqFt'] = i.find("div", {"class": "property-sqft"}).text.strip()
  info['city_info'] = i.find("div", {"class": "property-city"}).text.split()[-1]

  all_info.append(info)
```

```
[15] print(all_info)
```

```
[{'Title': 'FOR SALE', 'Price': '$595,000', 'Beds': '3 beds', 'Bath': '2 baths', 'SqFt': '2,366 sq. ft', 'city_info': '80231'}, {'Title': 'FOR SALE', 'Price': '$360,000', 'Beds': '2 beds', 'Bath': '2 baths', 'SqFt': '936 sq. ft', 'city_info': '80204'}, {'Title': 'FOR SALE', 'Price': '$499,950', 'Beds': '3 beds', 'Bath': '1 bath', 'SqFt': '2,1
```

```
[16] len(all_info)
```

```
20
```

```
all_properties = []

for i in tqdm(range(1,34)):
  r = requests.get(f"https://www.century21.com/real-estate/denver-co/LCCODENVER/?sa=CCOARVADA%2CCCOAURORA%2CCCOBRIGHTON%2CCCOBROOMFIELD%2CCCOCOMMERCECITY%2CCCOENGLEWOOD%2CCCOHENDERSON%2CCCOLAFAYETTE%2CCCOLITTLETON%2CCCOLOUISVILLE%2CCCOMORRISON%2CCCOTHORNTON%2CCCOWESTMINSTER%2CCCOWHEATRIDGE&searchKey=f505681d-a16a-4836-a430-9ea664844d0b&o=com
      headers={'User-agent': 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:61.0) Gecko/20100101 Firefox/61.0'})

  properties = soup.find_all("div", {"class": "property-card-primary-info"})
  for item in properties:
    info = {}
    info['Title'] = item.find("div", {"class": "pdp-listing-type sale"}).text
    info['Price'] = item.find("a", {"class":"listing-price"}).text.strip()
    info['Beds'] = item.find("div", {"class": "property-beds"}).text.strip()
    info['Bath'] = item.find("div", {"class": "property-baths"}).text.strip()
    info['SqFt'] = item.find("div", {"class": "property-sqft"}).text.strip()
    info['city_info'] = item.find("div", {"class": "property-city"}).text.split()[-1]

    all_properties.append(info)
print(all_properties)
```

Data output

```
[22] df
```

	Title	Price	Beds	Bath	SqFt	city_info
0	FOR SALE	$595,000	3 beds	2 baths	2,366 sq. ft	80231
1	FOR SALE	$360,000	2 beds	2 baths	936 sq. ft	80204
2	FOR SALE	$499,950	3 beds	1 bath	2,112 sq. ft	80233
3	FOR SALE	$350,000	3 beds	2 baths	1,144 sq. ft	80003
4	FOR SALE	$843,030	4 beds	4 baths	4,571 sq. ft	80016
...	...	...	...	...	...	...
655	FOR SALE	$646,200	3 beds	3 baths	2,507 sq. ft	80027
656	FOR SALE	$620,000	2 beds	1 bath	1,084 sq. ft	80202
657	FOR SALE	$1,400,000	4 beds	2 baths	3,088 sq. ft	80206
658	FOR SALE	$775,000	6 beds	4 baths	4,968 sq. ft	80016
659	FOR SALE	$735,900	3 beds	3 baths	1,675 sq. ft	80204

660 rows × 6 columns

Data Processing

```
[27] df['Bath']= df['Bath'].apply(lambda x : x.replace('s',''))
```

```
[28] df['Bath'] = df['Bath'].apply(lambda x : x.replace('bath',''))
```

```
[29] df['Beds'] = df['Beds'].apply( lambda x : x.replace('beds',''))
     df['Beds']
```

```
df['Beds'] = df['Beds'].apply( lambda x : x.replace('bed',''))
```

```
[31] df['Price'] = df['Price'].apply(lambda x : x.replace('$',''))
```

```
[32] df['Price'] = df['Price'].apply(lambda x : x.replace(',',''))
```

```
[33] df['SqFt'] = df['SqFt'].apply(lambda x : x.replace('sq. ft',''))
```

```
[34] df['SqFt'] = df['SqFt'].apply(lambda x : x.replace(',',''))
```

```
[35] # type conversion
     df[['Price','SqFt','Beds','Bath']]= df[['Price','SqFt','Beds','Bath']].astype(int)
```

```
[36] df.info()
```

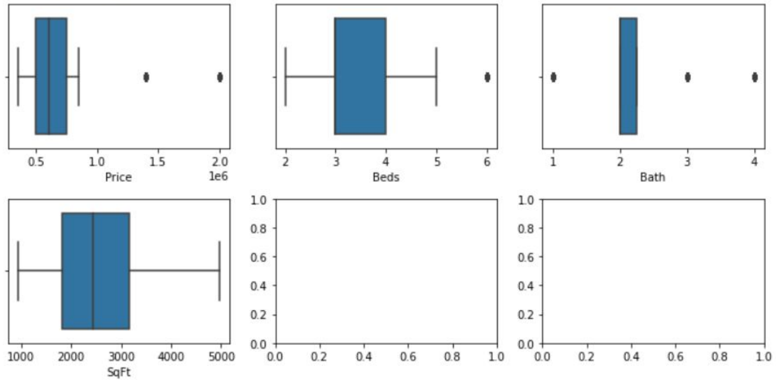
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 660 entries, 0 to 659
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   Title      660 non-null    object
 1   Price      660 non-null    int64 
 2   Beds       660 non-null    int64 
 3   Bath       660 non-null    int64 
 4   SqFt       660 non-null    int64 
 5   city_info  660 non-null    object
dtypes: int64(4), object(2)
memory usage: 31.1+ KB
```

Data Processing

```
# Outlier Analysis
fig, axs = plt.subplots(2,3, figsize = (10,5))
plt1 = sns.boxplot(df['Price'], ax = axs[0,0])
plt2 = sns.boxplot(df['Beds'], ax = axs[0,1])
plt3 = sns.boxplot(df['Bath'], ax = axs[0,2])
plt1 = sns.boxplot(df['SqFt'], ax = axs[1,0])


plt.tight_layout();
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable
  FutureWarning
```

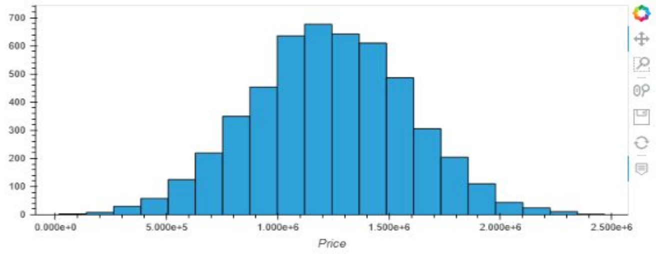


Exploratory Data Analysis

In [15]:

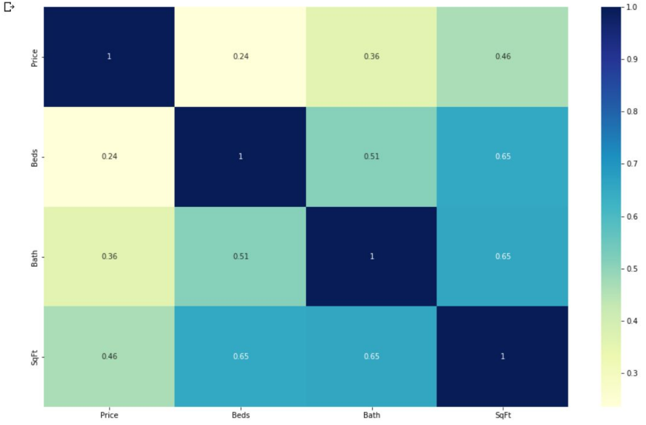
```
USAhousing.hvplot.hist("Price")
```

Out[15]:

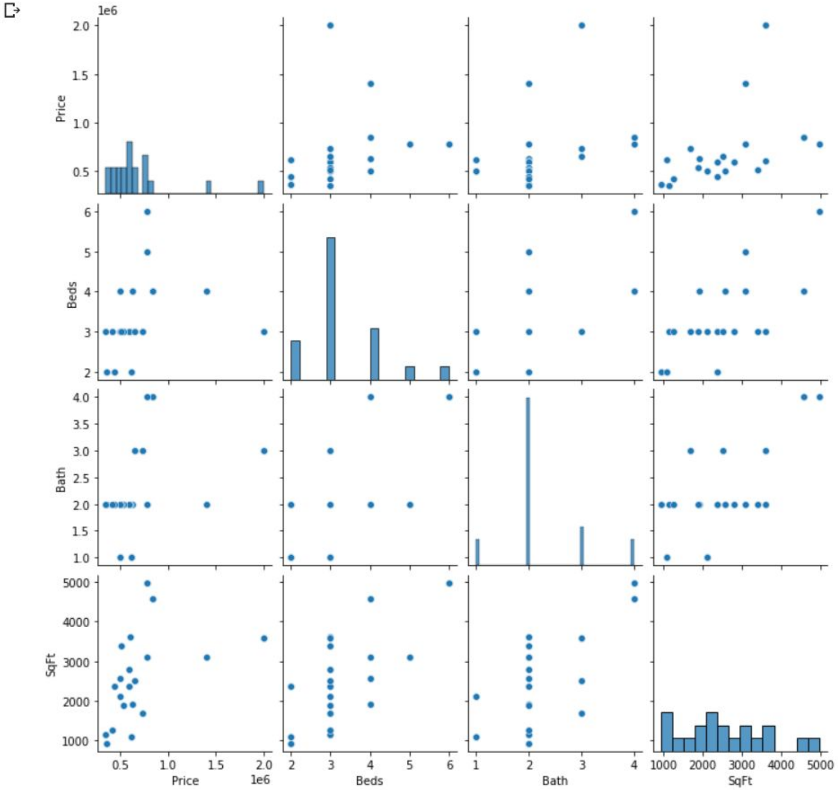



```
# Let's check the correlation coefficients to see which variables are highly correlated

plt.figure(figsize = (16, 10))
sns.heatmap(df.corr(), annot = True, cmap="YlGnBu")
plt.show()
```




```
sns.pairplot(df)
plt.show()
```



✔ Regression Evaluation Metrics

Here are three common evaluation metrics for regression problems:

- **Mean Absolute Error** (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **Mean Squared Error** (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- **Root Mean Squared Error** (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

📌 Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, because we want to minimize them.

Model Results

Linear Regression

R^2: 0.8975574856866375

Root Mean Squared Error: 115272.59421

CV R^2: 0.910746

Ridge

R^2: 0.8975574856866375

Root Mean Squared Error: 114728.35240476725

Lasso

R^2: 0.8935089334837565
Root Mean Squared Error: 113692.11382114442

Q & A