Regression to predict median house values

Import data from California_Houses.csv file

In [1]:
```
import numpy as np
data_set = np.genfromtxt('California_Houses.csv', delimiter=',', dtype=None, encoding='utf-8')
headers=data_set[0]
data_set_data=data_set[1:].astype('f')
data_set
```

Out[1]:
```
array([['"Median_House_Value"', '"Median_Income"', '"Median_Age"', ...,
        '"Distance_to_SanDiego"', '"Distance_to_SanJose"',
        '"Distance_to_SanFrancisco"'],
       ['452600', '8.3252', '41', ..., '735501.80698384',
        '67432.5170008434', '21250.2137667799'],
       ['358500', '8.3014', '21', ..., '733236.884360166',
        '65049.9085739663', '20880.6003997074'],
       ...,
       ['92300', '1.7', '17', ..., '830699.57316343', '240172.220489273',
        '212097.936231564'],
       ['84700', '1.8672', '18', ..., '834672.461886794',
        '238193.865908775', '207923.199166252'],
       ['89400', '2.3886', '16', ..., '825569.179027675',
        '233282.76906299', '205473.376575195']], dtype='<U26')
```

Randomizing data, shuffling and splitting

We are shuffling all our data using numpy Spliting data to Training, Testing, Validation data sets

In [2]:
```
rng = np.random.default_rng()
rng.shuffle(data_set_data)
train,test_validate=np.array_split(data_set_data,[int(0.70 * len(data_set_data))])
test,validation=np.array_split(test_validate,[int(0.50 * len(test_validate))])
print(f"train: {train.shape[0]}\n"
      f"test: {test.shape[0]}\n"
      f"validation: {validation.shape[0]}")
```

```
train: 14447
test: 3096
validation: 3097
```

Separating features from the perdiction

We are taking the first ten features and putting them in array x_data and the last feature (the predection) in array y_data

In [3]:
```
def return_x_y_arrays(data_set_to_be_sliced):
    buf=data_set_to_be_sliced.tolist()
    x_data= []
    y_data=[]
    for data in buf:
        x=data[1:]
        x_data.append(x)
        y=np.array([data[0]])
        y_data.append(y)
    x_data=np.array(x_data)
    y_data=np.array(y_data).ravel()
    return x_data,y_data
```

In [4]:
```
x_train,y_train=return_x_y_arrays(train)
x_validation,y_validation=return_x_y_arrays(validation)
x_test,y_test=return_x_y_arrays(test)
```

Normalize data to make them all have the same range

In order to prevent that certain data have more influence on the predection than others

In [5]:
```
from sklearn.preprocessing import MinMaxScaler
def normailze(data):
    scaler = MinMaxScaler()
    scaler.fit(data)
    normalized_data = scaler.transform(data)
    return normalized_data

x_test=normailze(x_test)
x_train=normailze(x_train)
x_validation=normailze(x_validation)
```

Using Linear regression and calculating scores

Also calculating mean absolute and squared errors

In [6]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train,y_train)
y_predict=model.predict(x_train)
print(model.score(x_train,y_train))
print(model.score(x_test,y_test))
print(model.score(x_validation,y_validation))
print("mean absolute error of linear:", mean_absolute_error(y_true=y_train,y_pred=y_predict))
print("mean square error of linear:", mean_squared_error(y_true=y_train,y_pred=y_predict))
```

```
0.6484124996301643
0.4367558308392615
0.576352706639417
mean absolute error of linear: 50017.940867707366
mean square error of linear: 4691762040.748949
```

Using Lasso regression

In [7]:
```
from sklearn.linear_model import Lasso
model_laso=Lasso(alpha=100)
model_laso.fit(x_train,y_train)
y_predict=model_laso.predict(x_train)
print(model_laso.score(x_train,y_train))
print(model_laso.score(x_test,y_test))
print(model_laso.score(x_validation,y_validation))
print("mean absolute error of lasso:",mean_absolute_error(y_true=y_train,y_pred=y_predict))
print("mean square error of lasso:",mean_squared_error(y_true=y_train,y_pred=y_predict))
```

```
0.6377024883086774
0.5843123454622934
0.628579460906055
mean absolute error of lasso: 51165.984855248484
mean square error of lasso: 4834681867.31072
```

Using Ridge regression

In [8]:
```
from sklearn.linear_model import Ridge
model_ridge=Ridge(alpha=1)
model_ridge.fit(x_train,y_train)
y_predict=model_ridge.predict(x_train)
print(model_ridge.score(x_train,y_train))
print(model_ridge.score(x_test,y_test))
print(model_ridge.score(x_validation,y_validation))
print("mean absolute error of ridge:",mean_absolute_error(y_true=y_train,y_pred=y_predict))
print("mean square error of ridge:",mean_squared_error(y_true=y_train,y_pred=y_predict))
```

```
0.6461267444769891
0.5315973949764232
0.6113186860383444
mean absolute error of ridge: 50355.7678144383
mean square error of ridge: 4722264317.567187
```