Real-World Applications Assignment

Have you ever wondered how professionals who work in buying and selling homes know how to appropriately price a house? It's a big problem to solve -- set a price too low and both you and your client lose money. Set a price too high and the house stays on the market for a long time -- also costing you and your client money. One effective way to solve this problem is using matrix operations to evaluate a housing dataset, which can provide you with a data-driven answer to the problem.

In this peer review assignment, you'll use matrix operations to find a predictive value for housing prices using a real-world dataset.

In finding the predictive value, you'll run a model based on the normal equation to find historical housing prices and the predictors (number of bathrooms, lot size, upgrades, etc) for a homeowner who wants to sell. The predictors will indicate the price at which the homeowner should sell the house. You'll then provide another example of a real-world scenario for which you could use matrices to solve a problem.

Once you complete the assignment, you'll provide your peers with feedback on their assignments. The peer review will help improve yours and your peers' understanding of how to use concrete examples to study real-world datasets to solve a problem, and matrix operations to find a predictive value.

==========

Import Libraries

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

import sklearn
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder, OneHo

from sklearn.model_selection import cross_val_score, train_test_split, cross_valida
from sklearn.model_selection import KFold, cross_val_predict, RandomizedSearchCV, S

from sklearn.metrics import accuracy_score, auc, classification_report, confusion_m
from sklearn.metrics import plot_confusion_matrix, plot_roc_curve, precision_score,
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

from sklearn.linear_model import ElasticNet, Lasso, LinearRegression, LogisticRegre
```

```
%matplotlib inline
#sets the default autosave frequency in seconds
%autosave 60
sns.set_style('dark')
sns.set(font_scale=1.2)

plt.rc('axes', titlesize=9)
plt.rc('axes', labelsize=14)
plt.rc('xtick', labelsize=12)
plt.rc('ytick', labelsize=12)

import warnings
warnings.filterwarnings('ignore')


pd.set_option('display.max_columns',None)
#pd.set_option('display.max_rows', None)
pd.set_option('display.width', 1000)
pd.set_option('display.float_format','{:.2f}'.format)

random.seed(0)
np.random.seed(0)
np.set_printoptions(suppress=True)
```

```
Autosaving every 60 seconds
```

==========

Quick Data Glance

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

In [3]:
```
df.head()
```

Out[3]:

	size	price
0	479.75	154282.13
1	487.29	281626.34
2	518.38	183459.49
3	525.81	168047.26
4	525.81	191486.90

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   size    100 non-null    float64
 1   price   100 non-null    float64
dtypes: float64(2)
memory usage: 1.7 KB
```

==========

Simple Linear Regression (SciKit Learn)

Simple Linear Regression is a method to help us understand the relationship between two variables:

- The predictor/independent variable (X)
- The response/dependent variable (that we want to predict)(Y)

The result of Linear Regression is a **linear function** that predicts the response (dependent) variable as a function of the predictor (independent) variable.

$$Y : Response\ Variable$$
$$X : Predictor\ Variables$$

Linear Function

$$Yhat = a + bX$$

- a refers to the **intercept** of the regression line, in other words: the value of Y when X is 0
- b refers to the **slope** of the regression line, in other words: the value with which Y changes when X increases by 1 unit

In [5]:
```
df.head()
```

Out[5]:

	size	price
0	479.75	154282.13
1	487.29	281626.34
2	518.38	183459.49
3	525.81	168047.26
4	525.81	191486.90

In [6]:
```
X = df[['size']] # predictors
y = df[['price']] # target
```

In [7]:
```
#X.values, y.values
```

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [9]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[9]:
```
((80, 1), (20, 1), (80, 1), (20, 1))
```

In [10]:
```
lr = LinearRegression()
lr
```

Out[10]:
```
LinearRegression()
```

In [11]:
```
lr.fit(X_train,y_train)
```

Out[11]:
```
LinearRegression()
```

In [12]:
```
lr_pred = lr.predict(X_test)
lr_pred[0:5]
```

Out[12]:
```
array([[245542.43087405],
       [379800.28348123],
       [216132.02772416],
       [257158.99345649],
       [333135.86826037]])
```

In [13]:
```
prediction = pd.DataFrame(data=lr_pred, columns=['predicted'])
prediction
```

Out[13]:

	predicted
0	245542.43
1	379800.28
2	216132.03
3	257158.99
4	333135.87
5	412324.38
6	238606.66
7	332156.43
8	257111.16
9	438892.14
10	257111.16
11	401933.25
12	339581.92
13	229071.97
14	223288.74
15	247528.64
16	242282.96
17	244537.94
18	253188.86
19	223288.74

In [14]:
```
y_test.reset_index(drop=True, inplace=True)
y_test
```

Out[14]:

	price
0	225656.12
1	354512.11
2	183459.49
3	263311.70
4	320345.52
5	440201.62
6	262423.50
7	299416.98
8	262477.86
9	454512.76
10	266684.25
11	365863.94
12	293044.50
13	245050.28
14	211724.10
15	298926.50
16	257828.42
17	282683.54
18	225452.32
19	252460.40

In [15]:
```
tableslr = pd.concat([y_test, prediction], axis=1)
tableslr
```

Out[15]:

	price	predicted
0	225656.12	245542.43
1	354512.11	379800.28
2	183459.49	216132.03
3	263311.70	257158.99
4	320345.52	333135.87
5	440201.62	412324.38
6	262423.50	238606.66
7	299416.98	332156.43
8	262477.86	257111.16
9	454512.76	438892.14
10	266684.25	257111.16
11	365863.94	401933.25
12	293044.50	339581.92
13	245050.28	229071.97
14	211724.10	223288.74
15	298926.50	247528.64
16	257828.42	242282.96
17	282683.54	244537.94
18	225452.32	253188.86
19	252460.40	223288.74

Find the predictive value of a house with size=3000

In [16]:
```
predicted_price = lr.predict([[3000]])
pd.DataFrame(data=predicted_price, columns=["Price"])
```

Out[16]:

	Price
0	781384.85

==========