# MSBA Workshop: Linear Regression and Matrix Algebra

## Linear Regression

Suppose we want to predict (or estimate) the sale price ($y$) of a house based on its square footage ($x$). Linear regression provides one way to do this by "fitting a line" to the data. That is, our predictions ($\hat{y}$) will be of the form

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

where $\hat{\beta}_0$ is the *intercept* and $\hat{\beta}_1$ is the *slope* or *coefficient* for square footage.

We'll use the Ames housing data as an example; `Gr Liv Area` ($x$) is square footage and `SalePrice` ($y$) is measured in 2010 US dollars.

```
import pandas as pd

import numpy as np
```

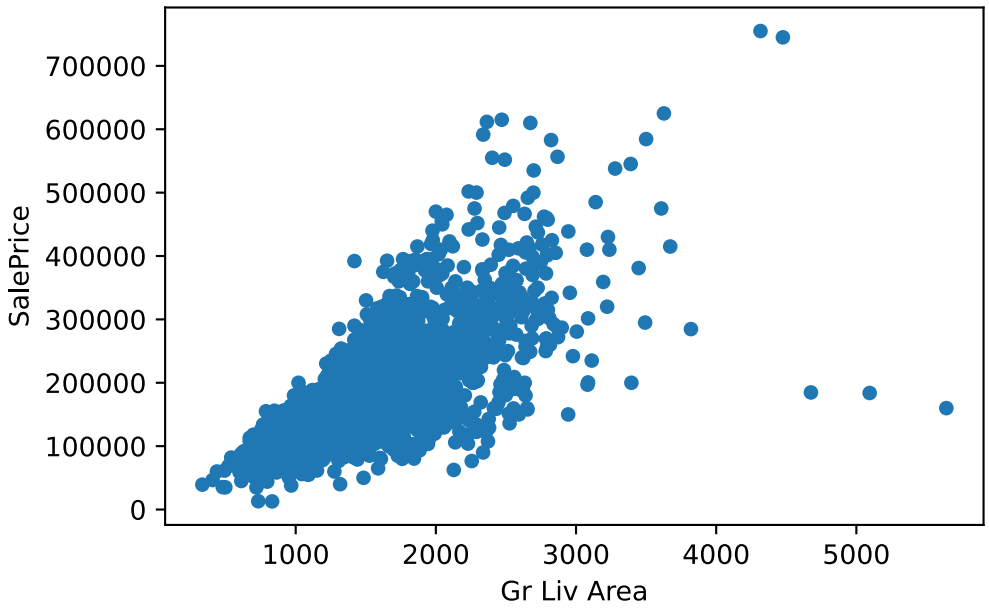
```
df_ames = pd.read_csv("http://dlsun.github.io/pods/data/AmesHousing.txt", sep
= "\t")

df = df_ames[["Gr Liv Area", "SalePrice"]]

df
```

| | Gr Liv Area | SalePrice |
|---|---|---|
| 0 | 1656 | 215000 |
| 1 | 896 | 105000 |
| 2 | 1329 | 172000 |
| 3 | 2110 | 244000 |
| 4 | 1629 | 189900 |
| … | … | … |
| 2925 | 1003 | 142500 |
| 2926 | 902 | 131000 |
| 2927 | 970 | 132000 |

| | Gr Liv Area | SalePrice |
|---|---|---|
| 2928 | 1389 | 170000 |
| 2929 | 2000 | 188000 |

First we summarize the data.

```
df.plot.scatter(x="Gr Liv Area", y="SalePrice")
```




The number of observations, $n$

```
len(df)
```

```
2930
```

The mean of each variable, $\bar{x}$, $\bar{y}$

```
df[["Gr Liv Area", "SalePrice"]].mean()
```

```
Gr Liv Area      1499.690444
SalePrice      180796.060068
dtype: float64
```

The standard deviation of each variable, $s_x$, $s_y$.

```
df[["Gr Liv Area", "SalePrice"]].std()
```

```
Gr Liv Area      505.508887
SalePrice      79886.692357
dtype: float64
```

The correlation between the variables $r$

```
df[["Gr Liv Area", "SalePrice"]].corr()
```

| | Gr Liv Area | SalePrice |
|---|---|---|
| Gr Liv Area | 1.00000 | 0.70678 |
| SalePrice | 0.70678 | 1.00000 |

Linear regression is one *machine learning* technique. Machine learning in Python is usually performed using the `sciKit-Learn` package. This library not only provides convenient functions for fitting machine learning models, it also enforces a strict structure of workflow that will help you make responsible choices. (You'll cover machine learning and `scikit-learn` in much more detail in GSB 544.)

```
from sklearn.linear_model import LinearRegression
```

```
sqft_model = LinearRegression()
sqft_model.fit(
    X=df[["Gr Liv Area"]],
    y=df["SalePrice"]
)
```

```
LinearRegression()
```

The above code "fits the line" using the Ames data. The fitted slope (`coef_`) and intercept (`intercept_`) are

```
sqft_model.coef_, sqft_model.intercept_
```

```
(array([111.69400086]), np.float64(13289.63436475952))
```

So our prediction equation is

$$\hat{y} = 13289 + 111.694x$$

The predicted sale price for a 3000 sqft house is

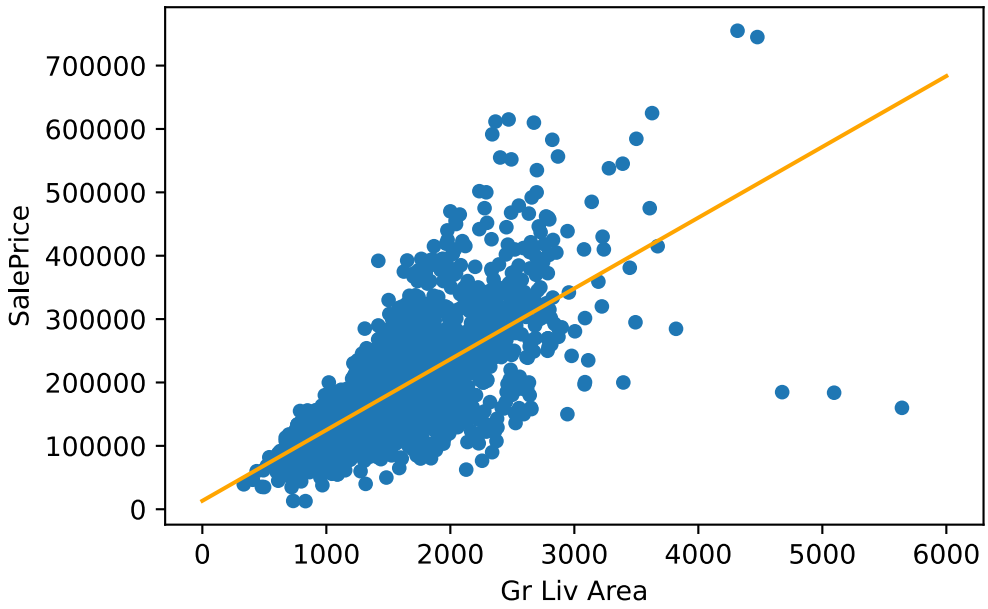
$$\hat{y} = 13289 + 111.694 \times 3000 = 348371$$

```
sqft_model.predict([[3000]])
```

```
array([348371.63693816])
```

The following code computes the predicted value for $x = 0, 1, 2, ...5999, 6000$ and plots the predicted values on the original scatter plot. We see that the predicted values all lie along a straight line.

```
X_new = pd.DataFrame()
X_new["Gr Liv Area"] = np.linspace(0, 6000, num=6000)
y_new_ = pd.Series(
    sqft_model.predict(X_new),
    index=X_new["Gr Liv Area"]
)

df.plot.scatter(x="Gr Liv Area", y="SalePrice")
y_new_.plot.line(c="orange")
```




## Least Squares

In the previous section we fitted a line to the data, resulting in a line with a specific slope and intercept. But why that particular line instead of some other line? How do we know which line is "best"?

A line is defined by its slope and intercept. Given a particular slope $\beta_1$ and intercept $\beta_0$, the predicted value for a certain $x$ will be $\beta_0 + \beta_1 x$. For the observed $(x_i, y_i)$ values in the data set, we can compare the actual value $y_i$ and the predicted value $\beta_0 + \beta_1 x_i$; the difference is the "prediction error" or *residual* and is represented on the scatterplot by the vertical distance between the actual value and the predicted value. The *least squares* criteria says to choose the line that minimizes the sum of the squared residuals for all the observations

$$S(\beta_0, \beta_1) = \sum_i (y_i - (\beta_0 + \beta_1 x_i))^2$$

To see the idea of least squares try this applet:

- There is some data loaded in (context doesn't matter, but it's predicting height from shoe size)
- Click on "Show Movable Line". A horizontal line should appear on the scatter plot. This line predicts the same value of $y$ (the observed mean $\bar{y}$) regardless of $x$
- Click on "Show Residuals". You should now see the vertical distances on the plot. These represent the prediction errors if you predicted the same value of $y$ regardless of $x$.
- Now click and drag one of the green boxes on the end of the line to change its slope, and click and drag the green box in the middle to move the line up and down (changing the intercept). Notice how "Sum Abs Errors (SAE)" changes as you move the line around
- Least squares minimizes the sum of the *squared* residuals (rather than absolute). Click on the box for "Show Squared Residuals" and you'll see a bunch of squares on the plot representing the squared residuals.
- Move the line around and observed how the squared residuals and "Sum Squared Errors (SSE)" changes.
- Keep moving the line around until you have made the SSE as small as possible.
- Now click on the box for "Show Regression Line" which will display the least squares regression line. How close was your line to the regression line?
- Under "Show Regression Line", click on the box for "Show Squared Residuals". Continue to move your line around; can you beat the SSE for the regression line?

Least squares regression can be set up as a calculus minimization problem.

The slope and intercept of the least squares regression line are

$$\hat{\beta}_1 = \frac{r s_y}{s_x}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

## Linear Regression Model

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

- $\beta_1$: "true" slope
- $\beta_0$: "true" intercept

- $\epsilon_i$: "true" residual, assumed to have some distribution (usually Normal with mean 0 and constant variance)

The line $\beta_0 + \beta_1 x$ represents how the *mean* of $y$ varies with $x$; $\epsilon$ represents how the actual $y$ values vary about their mean.

What's the difference between the "model" and the "fitted model"?

- The model is defined by some "true" slope $\beta_1$ and intercept $\beta_0$ which represents what we would see if we had data on a large population of many, many observations. The values $\beta_0$ and $\beta_1$ are *unknown parameters*.
- The fitted model estimates the unknown parameters $\beta_1$ and $\beta_0$ based on the observed sample data. The fitted intercept $\hat{\beta}_0$ and slope $\hat{\beta}_1$ are *observed statistics* computed from the sample data.

You might encounter formulas like the following when dealing with linear regression; what does it all mean?

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$
$$\hat{\mathbf{Y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$$
$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

To understand the above, we need a little matrix algebra. We only introduce what we need in the context of a tabular data set with rows representing observations and columns representing variables.

## Vector

You can think of a vector as an ordered list of values. For example, a vector might be a row/observation in a data set; the vector $[1656, 215000]$ represents a house with 1656 square feet and a 215000 sale price. Most often we think of vectors as columns in a data set; the `SalePrice` vector has 2930 elements $\mathbf{y} = [2150000, 10500, 172000, \ldots, 188000]$.

Vectors can either be row vectors or column vectors, but we usually think of them as columns. If $\mathbf{x}$ is a column vector then $\mathbf{x}^T$ is a row vector.

## Vectorization

Operations on vectors are performed element-by-element

$$10 \times \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 10 \times 1 \\ 10 \times 2 \\ 10 \times 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = \begin{bmatrix} 1 + 4 \\ 2 + 5 \\ 3 + 6 \end{bmatrix}$$

## Vector dot product

Suppose you course grade is based on scores $\mathbf{x} = [100, 70, 80, 90]$ with weights $\mathbf{y} = [0.1, 0.2, 0.3, 0.4]$. Your weighted course average is

$$0.1 \times 100 + 0.2 \times 70 + 0.3 \times 80 + 0.4 \times 90 = 84$$

This an example of a vector dot product of two vectors with the same length

$$\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i$$

If $\mathbf{x}$ and $\mathbf{y}$ are both column vectors of the same length, the dot product can be written as $\mathbf{x}^T \mathbf{y}$ (see more on matrix multiplication below).

```
import numpy as np

x = np.array([100, 70, 80, 90])
y = np.array([0.1, 0.2, 0.3, 0.4])

np.dot(x, y)
```

```
np.float64(84.0)
```

## Matrix

A matrix is a rectangular array of values. We often think of a data frame as a matrix with one row for each observation and one column for each variable. For matrix $\mathbf{A}$, $a_{ij}$ denotes the entry in row $i$ and column $j$. An $n \times m$ matrix has $n$ rows and $m$ columns.

A column vector can be considered an $n \times 1$ matrix, and a row vector as a $1 \times m$ matrix.

A square matrix has the same number of rows and columns, $n \times n$.

## Matrix Transpose

To find the transpose of a matrix, flip the rows and columns. If $\mathbf{A}$ is an $n \times m$ matrix then $\mathbf{A}^T$ is an $m \times n$ matrix.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

## Matrix multiplication

Continuing the weighted course average example above, suppose we have scores for three students (each student is a different row; each column is a different grade item)

$$\mathbf{X} = \begin{bmatrix} 100 & 70 & 80 & 90 \\ 95 & 75 & 80 & 100 \\ 90 & 90 & 80 & 80 \end{bmatrix}$$

with weights

$$\mathbf{Y} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix}$$

To find the weighted average for each student, we take the row corresponding to the student and dot product it with the weight vector, resulting in a vector with 3 rows and 1 column.

$$\mathbf{XY} = \begin{bmatrix} 0.1 \times 100 + 0.2 \times 70 + 0.3 \times 80 + 0.4 \times 90 \\ 0.1 \times 95 + 0.2 \times 75 + 0.3 \times 80 + 0.4 \times 100 \\ 0.1 \times 90 + 0.2 \times 90 + 0.3 \times 80 + 0.4 \times 80 \end{bmatrix} = \begin{bmatrix} 84 \\ 88.5 \\ 83 \end{bmatrix}$$

This is an example of matrix multiplication.

Two matrices $\mathbf{A}$ and $\mathbf{B}$ can only be multiplied if their dimensions align; the matrix product $\mathbf{AB}$ can only be computed if the number of columns of $\mathbf{A}$ is the same as the number of rows of $\mathbf{A}$.

If $\mathbf{A}$ is an $n \times m$ matrix and $\mathbf{B}$ is an $m \times o$ matrix then the matrix product $\mathbf{AB}$ is an $n \times o$ matrix where the $ij$ entry is the dot product of the $i$th row of $\mathbf{A}$ and the $j$ column of $\mathbf{B}$

```
X = np.array([
    [100, 70, 80, 90],
    [95, 75, 80, 100],
    [90, 90, 80, 80]])

y = np.array([0.1, 0.2, 0.3, 0.4])

X @ y
```

```
array([84. , 88.5, 83. ])
```

## Identity matrix

The identity matrix is a square matrix with 1s on the diagonals and 0 everywhere else.

For example, the $3 \times 3$ identity matrix is

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

If $\mathbf{A}$ is $n \times n$ and $\mathbf{I}$ is $n \times n$ then

$$\mathbf{AI} = \mathbf{IA} = \mathbf{A}$$

## Matrix inverse

An $n \times n$ square matrix $\mathbf{A}$ is *invertible* if there exists a $n \times n$ square matrix $\mathbf{A}^{-1}$, called the *matrix inverse*, which satisfies

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I} = \mathbf{A}^{-1}\mathbf{A}$$

If the inverse exists then it is unique, but some matrices do not have an inverse.

See Wikipedia for an example