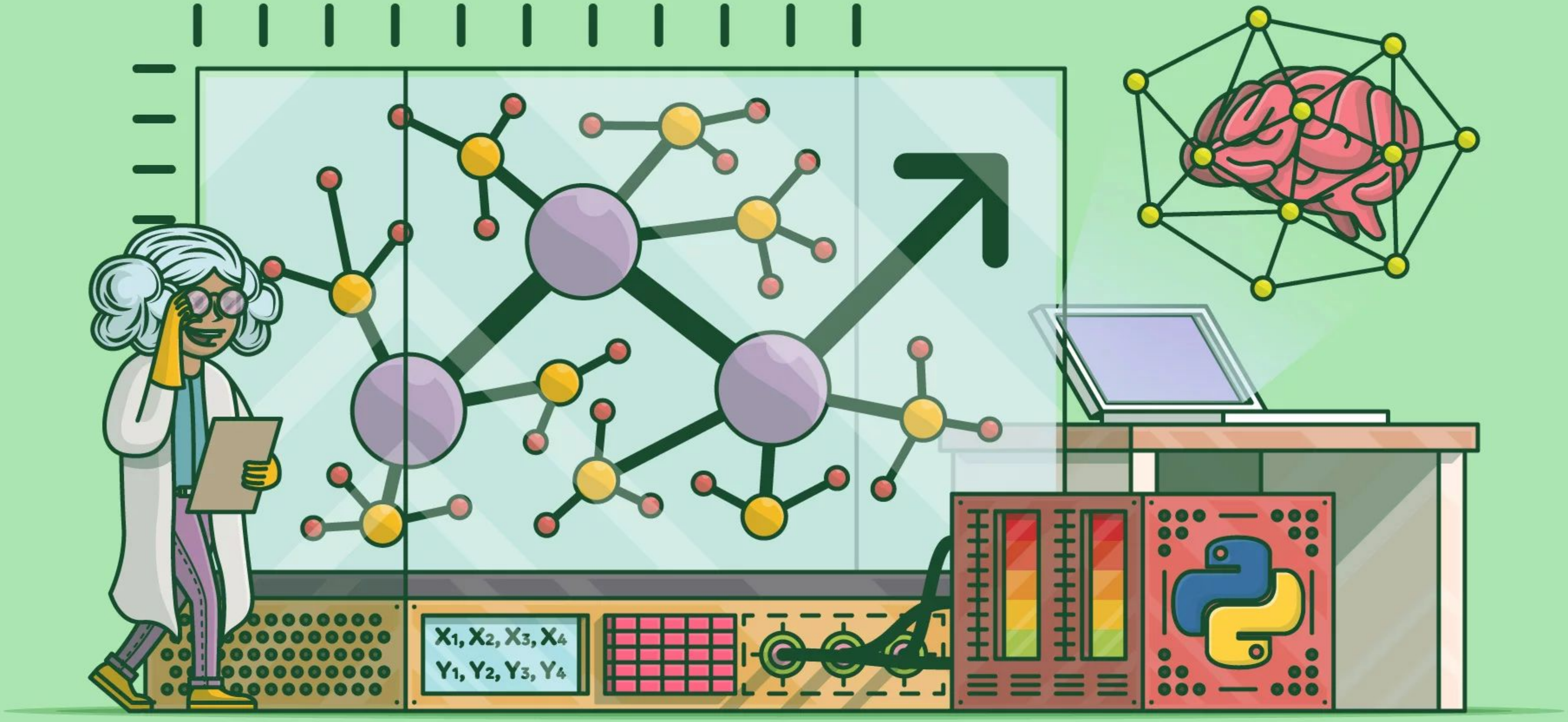


Lesson #04 - Linear Regression

notas

- We are going to start by covering **linear regression**
- We discuss the application of linear regression to **housing price prediction**
- Present the notion of a **cost function**
- Introduce the **gradient descent** method for learning.
- Refresher on **linear algebra concepts**.

notas

Instance-Based Learning
Vs
Model-Based Learning

Instance-Based Learning

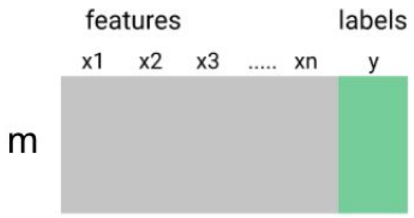
Model-Based Learning

$$f(x,\alpha,\beta)$$

Instance-Based Learning: KNN

Training Process

Training Data



Select Features

Done! No computational work in training process.

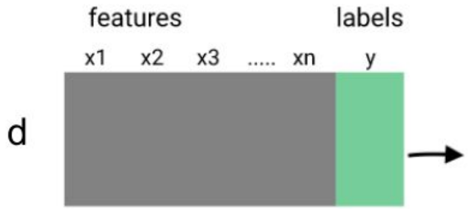
m >> n
d >> n
m >= d

Testing Process

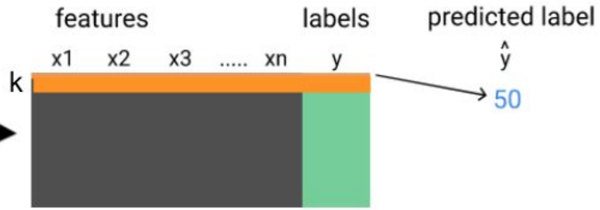
Using INSTANCES from training set to make predictions

Test Data

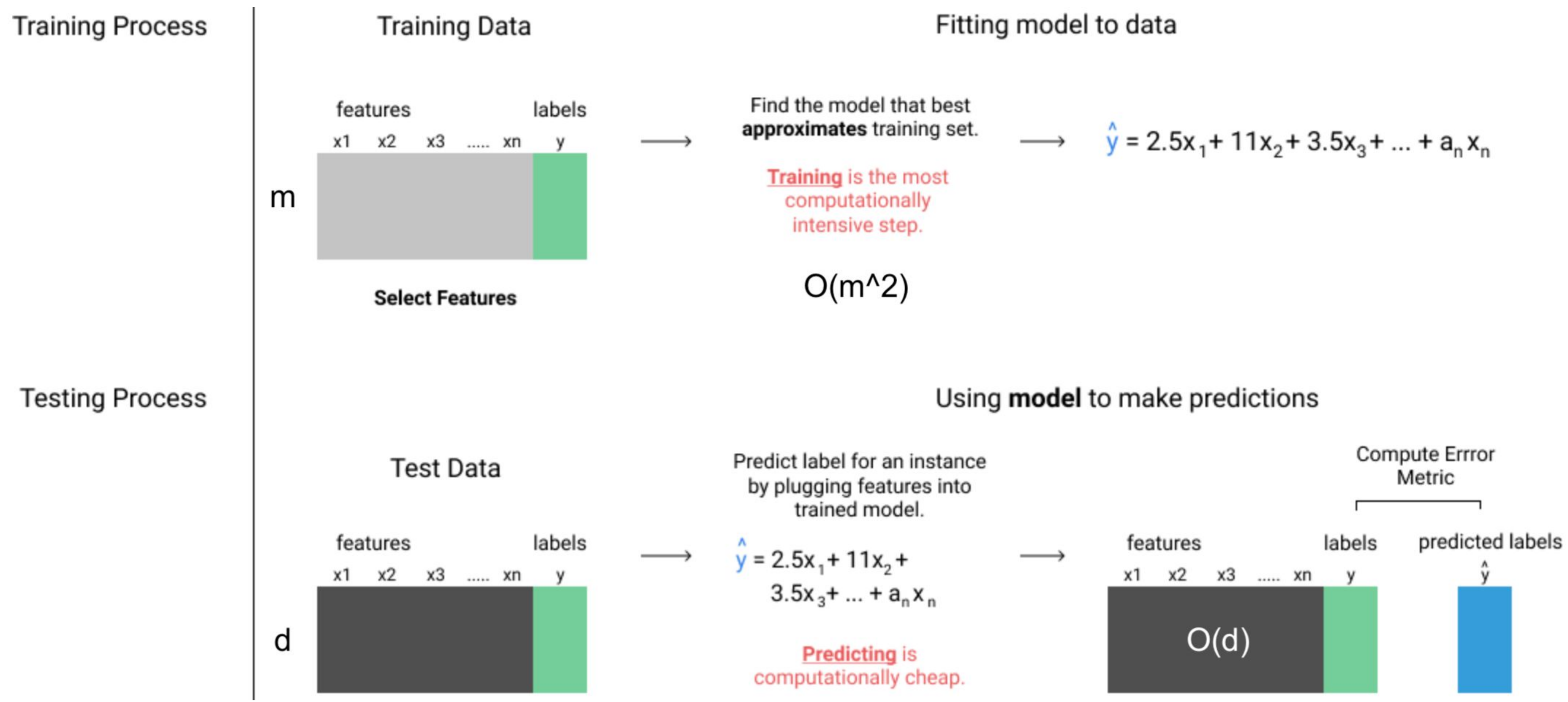
Predicting is computationally intensive.
O(mn + m^2 = m^2) per prediction



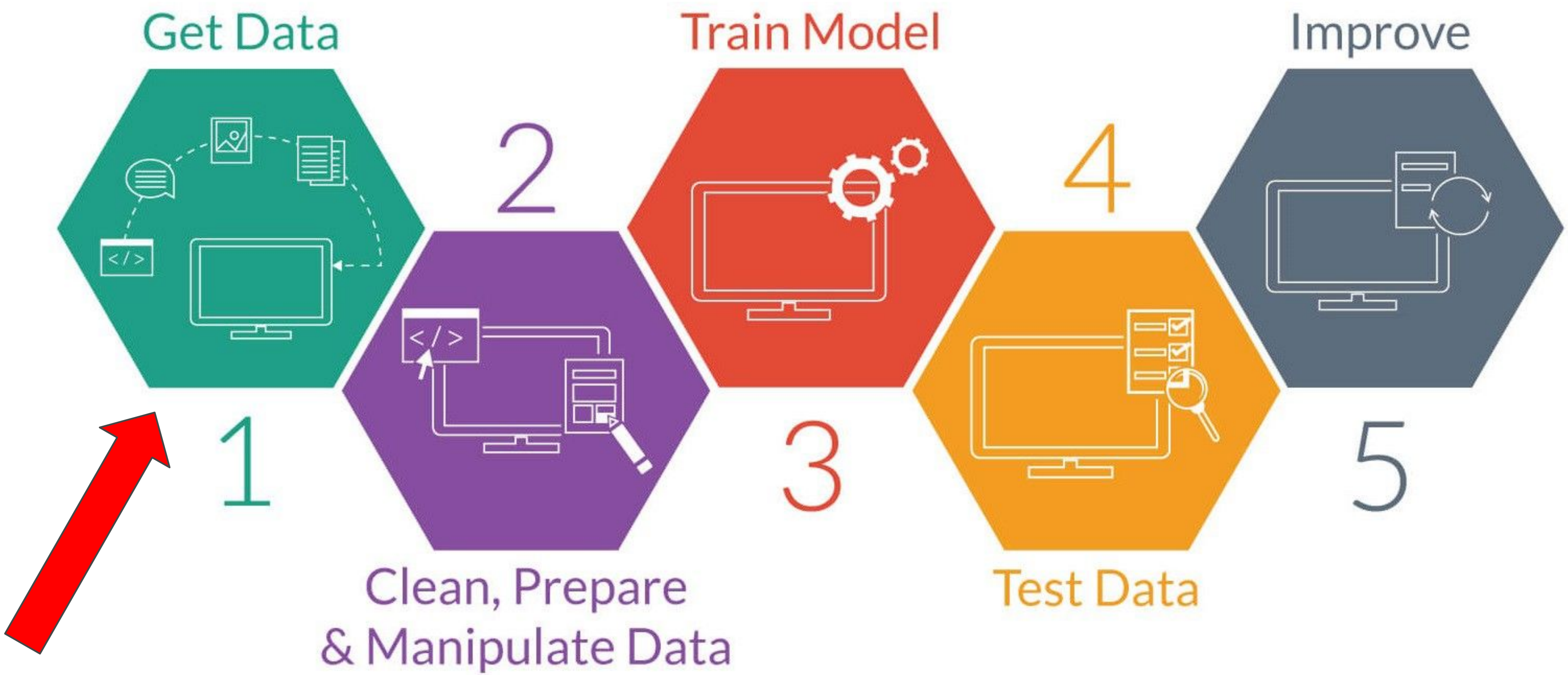
For each test intance:
1. Sort m train instances by Euclidean Distance
2. Select top k train instances form sorted training set and average their labels.



Model-Based Learning: Linear Regression



A general ML workflow



Journal of
Statistics Education

Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project

Dean De Cock
Truman State University

Journal of Statistics Education Volume 19, Number 3(2011),
www.amstat.org/publications/jse/v19n3/decock.pdf

Key Words: Multiple Regression; Linear Models; Assessed Value; Group Project.

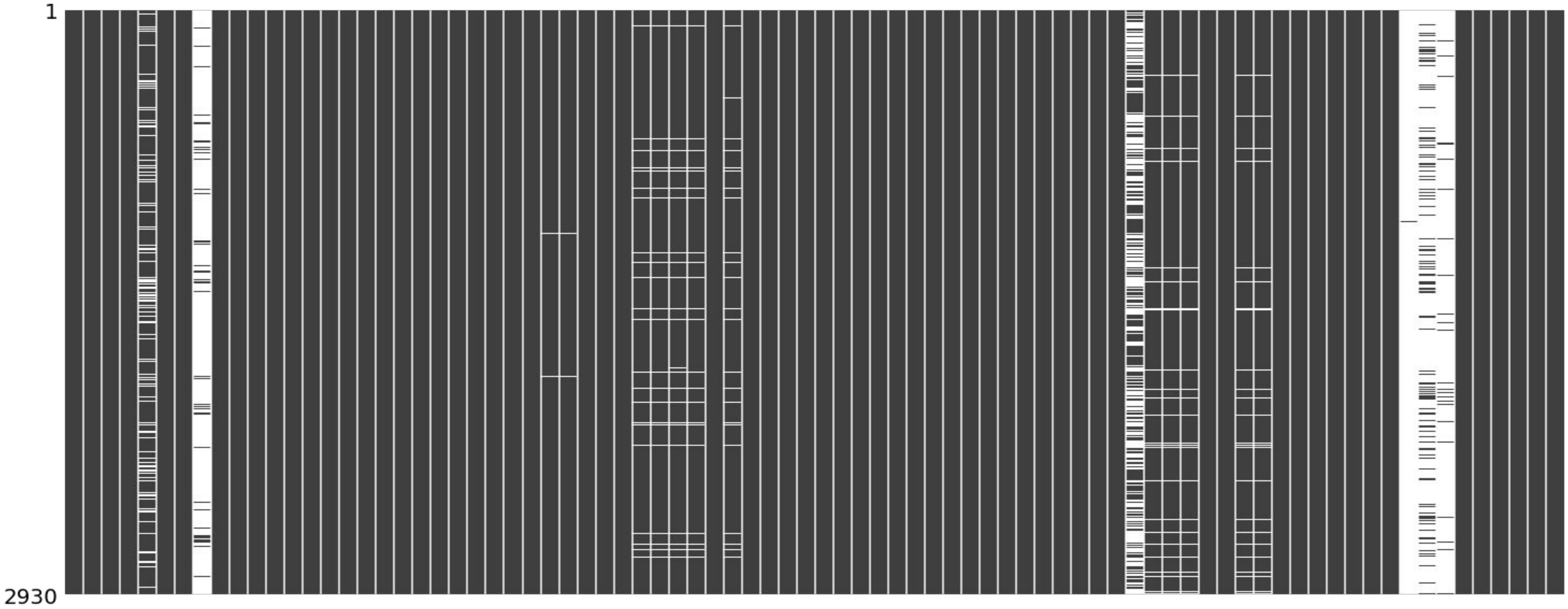
Abstract

This paper presents a data set describing the sale of individual residential property in Ames, Iowa from 2006 to 2010. The data set contains 2930 observations and a large number of explanatory variables (23 nominal, 23 ordinal, 14 discrete, and 20 continuous) involved in assessing home values. I will discuss my previous use of the Boston Housing Data Set and I will suggest methods for incorporating this new data set as a final project in an undergraduate regression course.

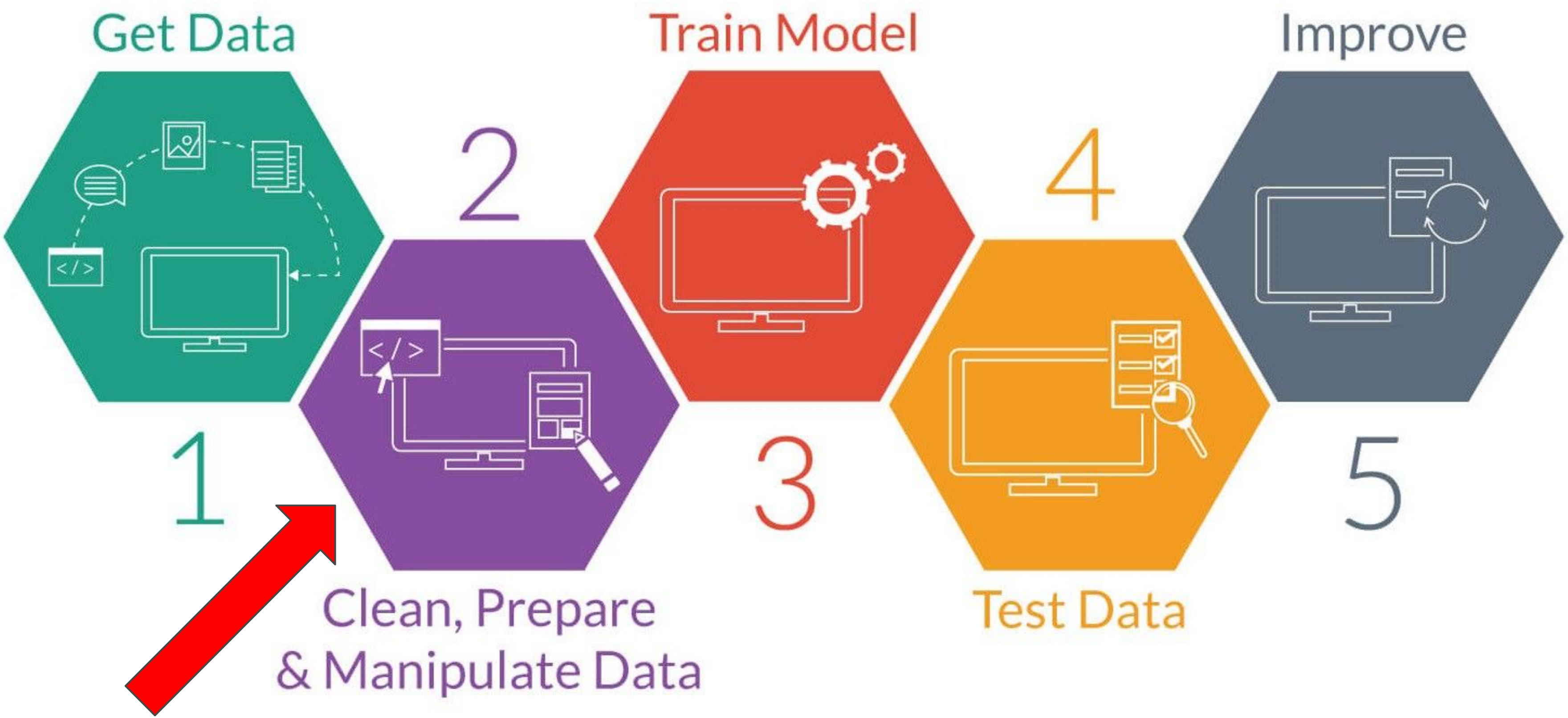
(2930, 82)

Visualizing Missing Values

```python
import pandas as pd
import missingno as msno
# get the data
data = pd.read_csv("AmesHousing.txt", sep='\t')
# visualize missing values
msno.matrix(data,sparkline=False)
```



A general ML workflow

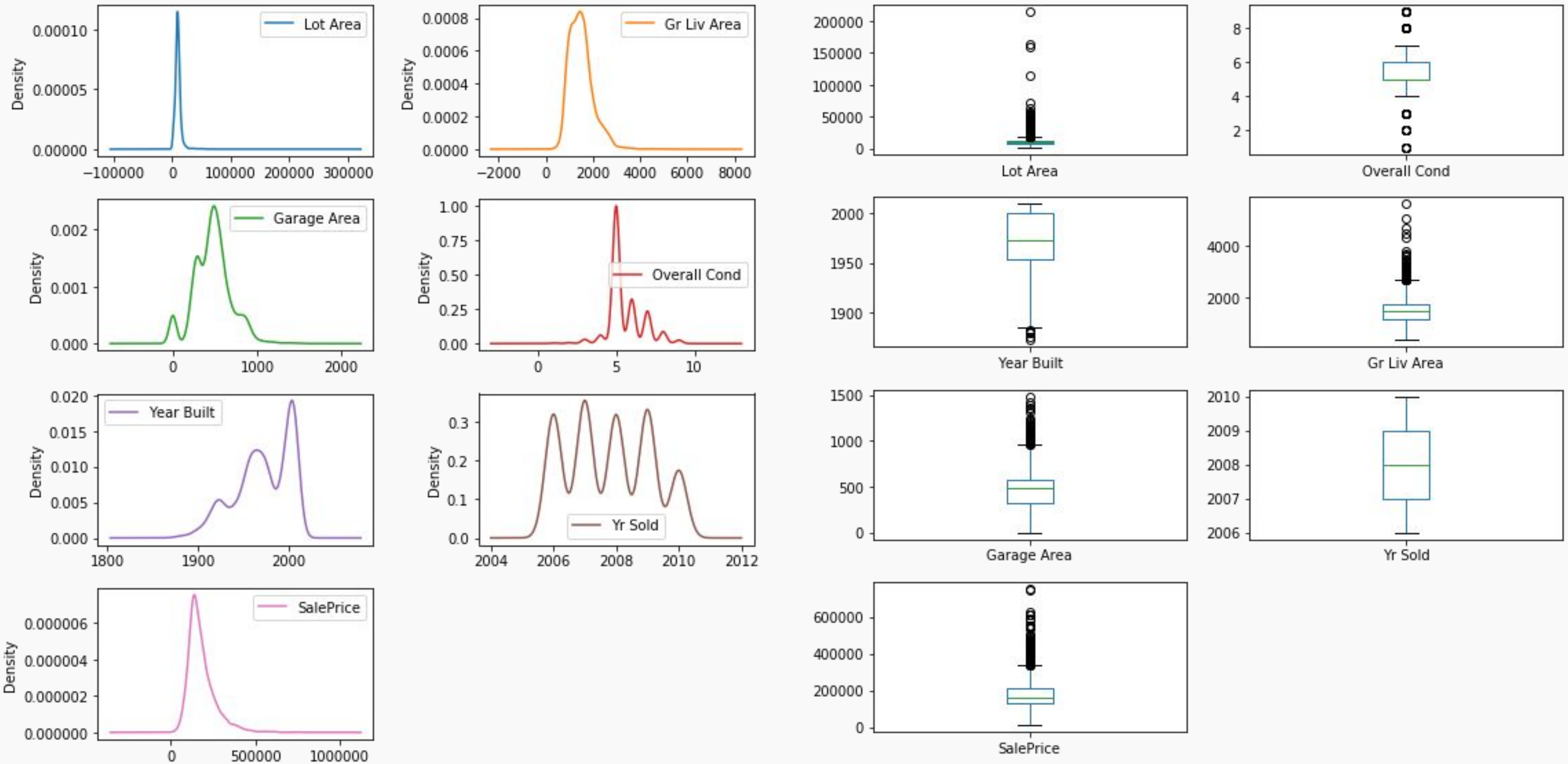


	Lot Area	Gr Liv Area	Garage Area	Overall Cond	Year Built	Yr Sold	SalePrice
1118	9428	1874	880.0	5	2007	2008	297900
2786	9800	894	552.0	7	1972	2006	149900
2633	7000	864	336.0	6	1962	2006	105000
2002	9439	1248	160.0	5	1930	2007	87000
268	4435	848	420.0	5	2003	2010	143750

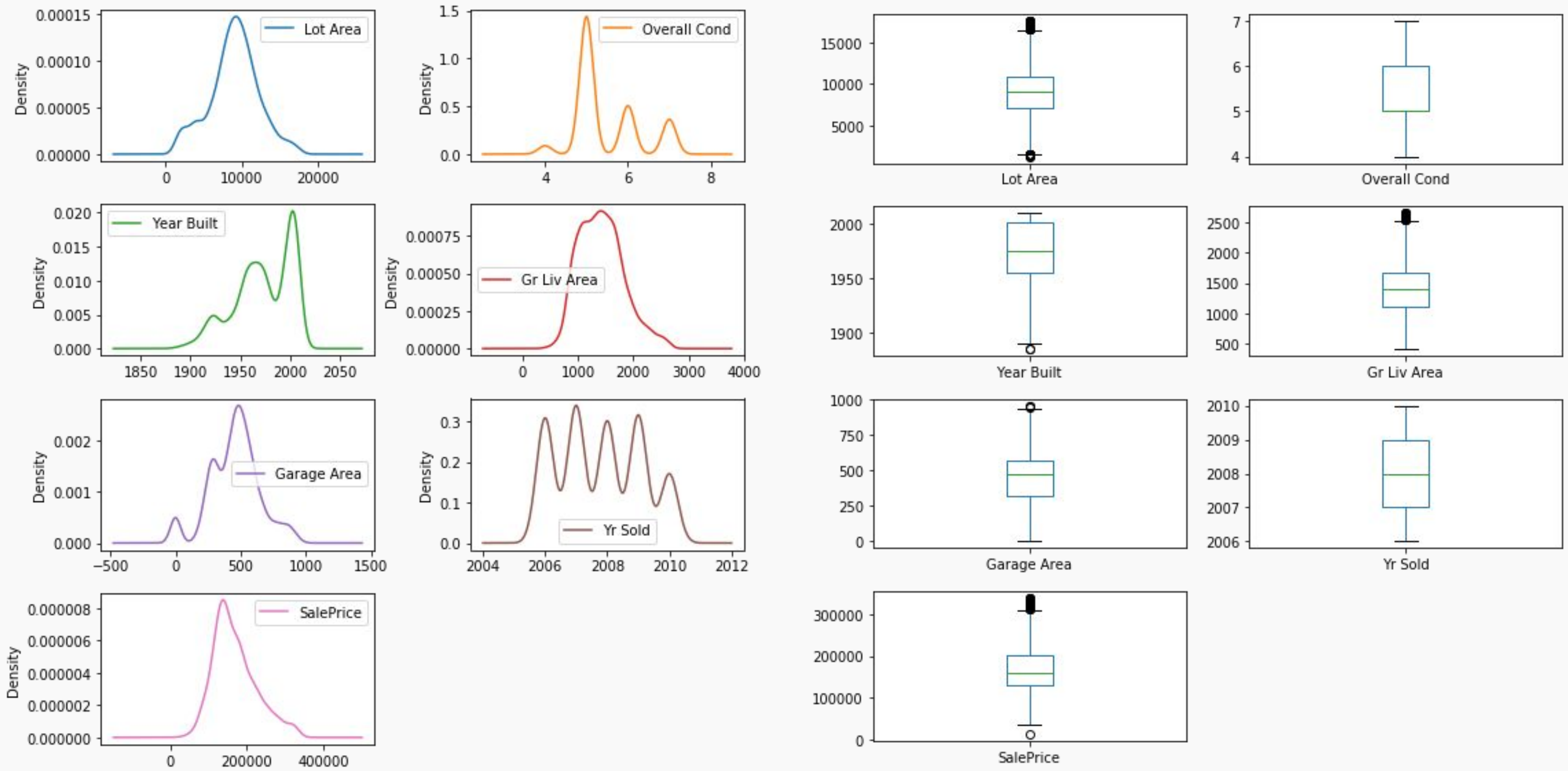
EDA

	Lot Area	Gr Liv Area	Garage Area	Overall Cond	Year Built	Yr Sold	SalePrice
count	2930.000000	2930.000000	2929.000000	2930.000000	2930.000000	2930.000000	2930.000000
mean	10147.921843	1499.690444	472.819734	5.563140	1971.356314	2007.790444	180796.060068
std	7880.017759	505.508887	215.046549	1.111537	30.245361	1.316613	79886.692357
min	1300.000000	334.000000	0.000000	1.000000	1872.000000	2006.000000	12789.000000
25%	7440.250000	1126.000000	320.000000	5.000000	1954.000000	2007.000000	129500.000000
50%	9436.500000	1442.000000	480.000000	5.000000	1973.000000	2008.000000	160000.000000
75%	11555.250000	1742.750000	576.000000	6.000000	2001.000000	2009.000000	213500.000000
max	215245.000000	5642.000000	1488.000000	9.000000	2010.000000	2010.000000	755000.000000

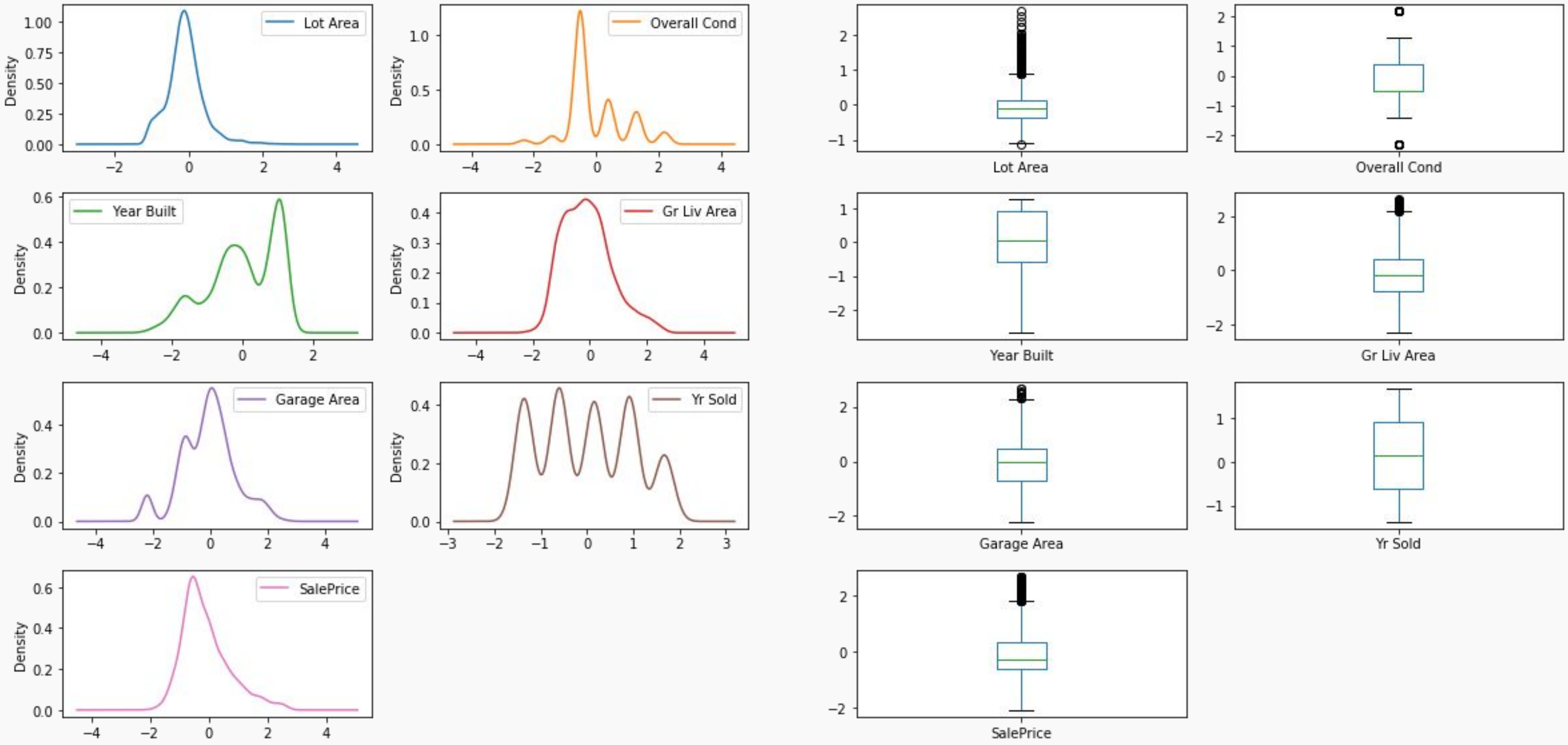
Original Data (2929,7)



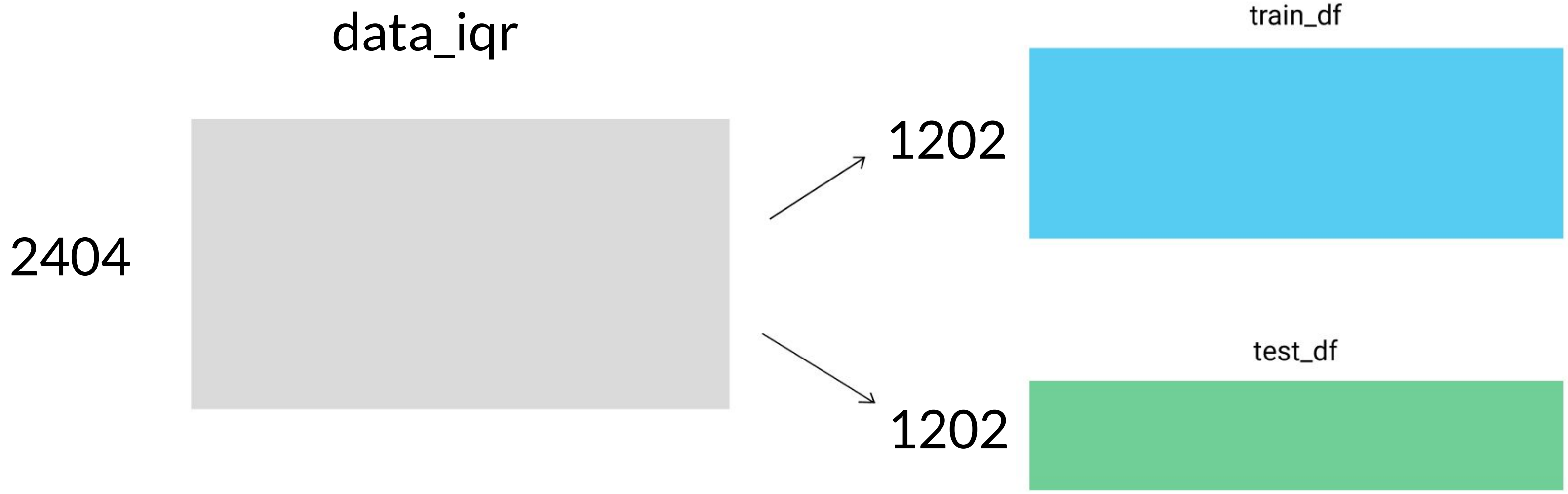
Outlier elimination - IQR method (2404,7) 17.90% of original data were removed



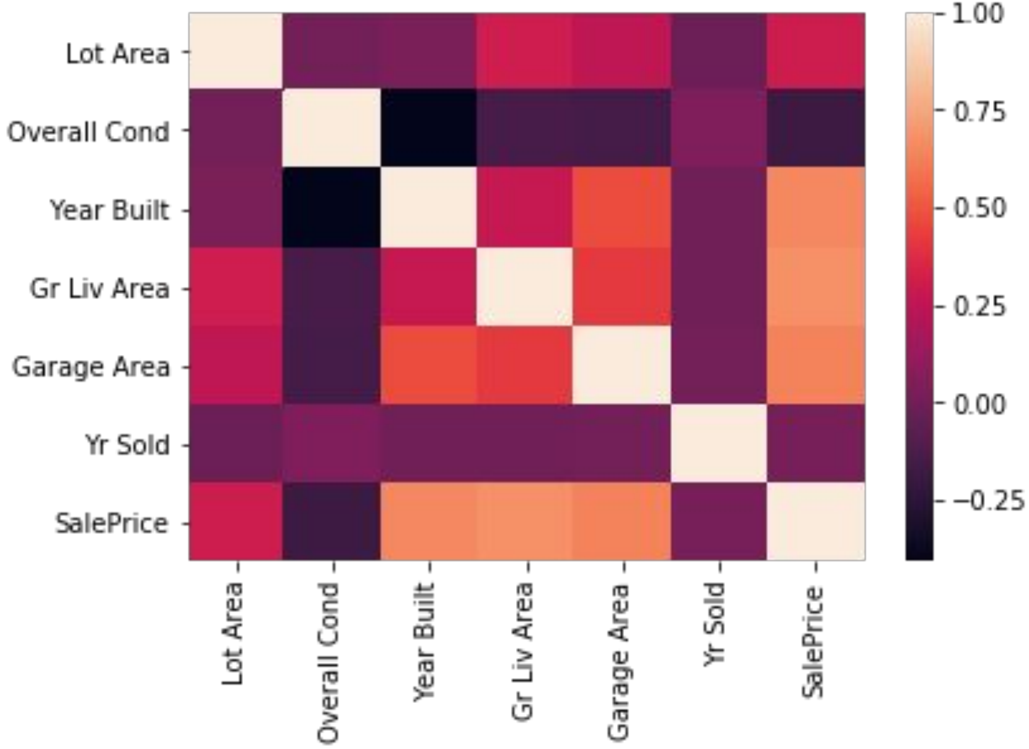
Outlier elimination - Z-Score method (2745,7) 6.28% of original data were removed



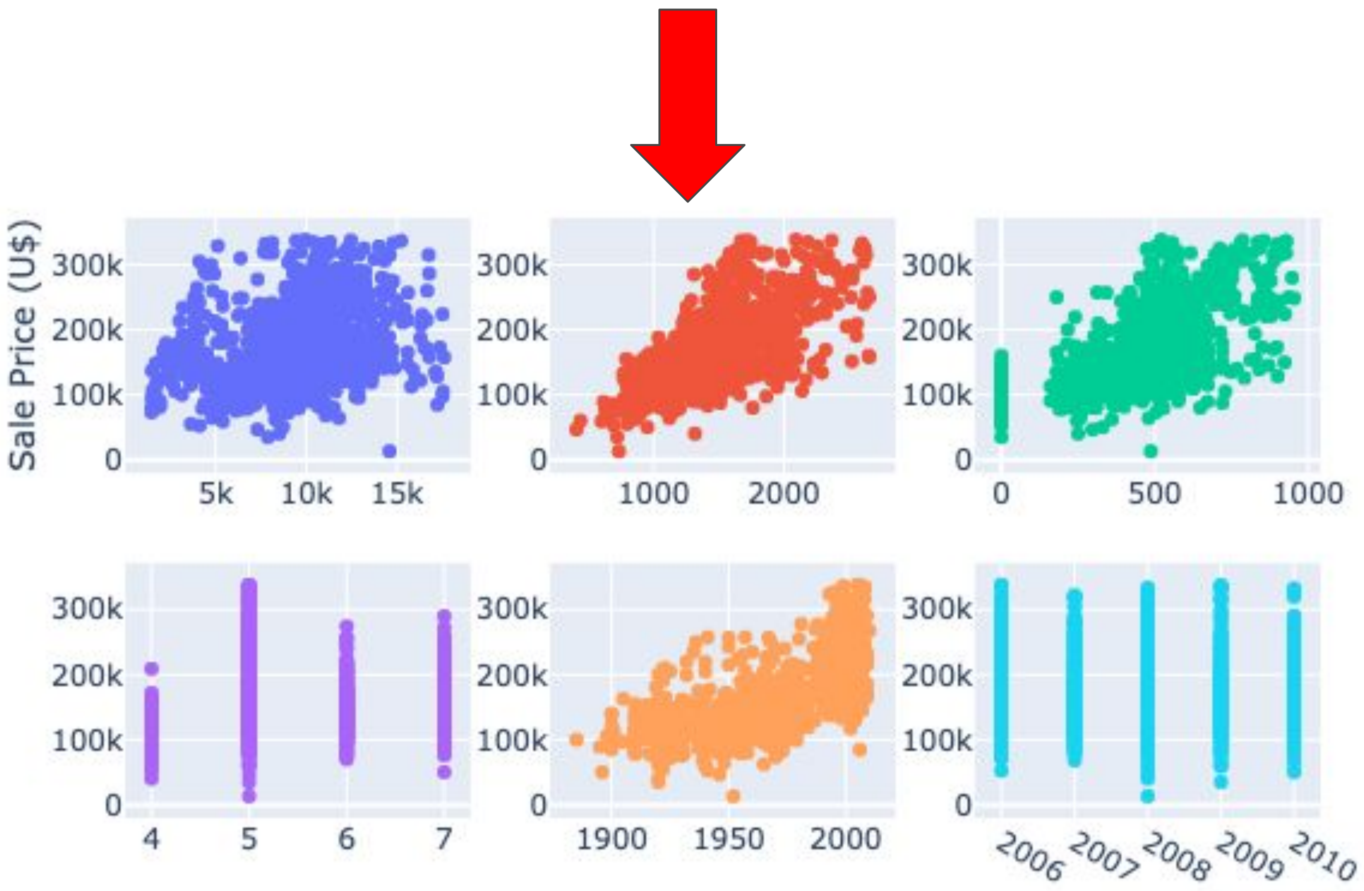
```
X_train, X_test, Y_train, Y_test = train_test_split(data_iqr.drop(axis=1,labels=["SalePrice"]),
                                                    data_iqr["SalePrice"],
                                                    test_size=0.5,
                                                    random_state=42)
```



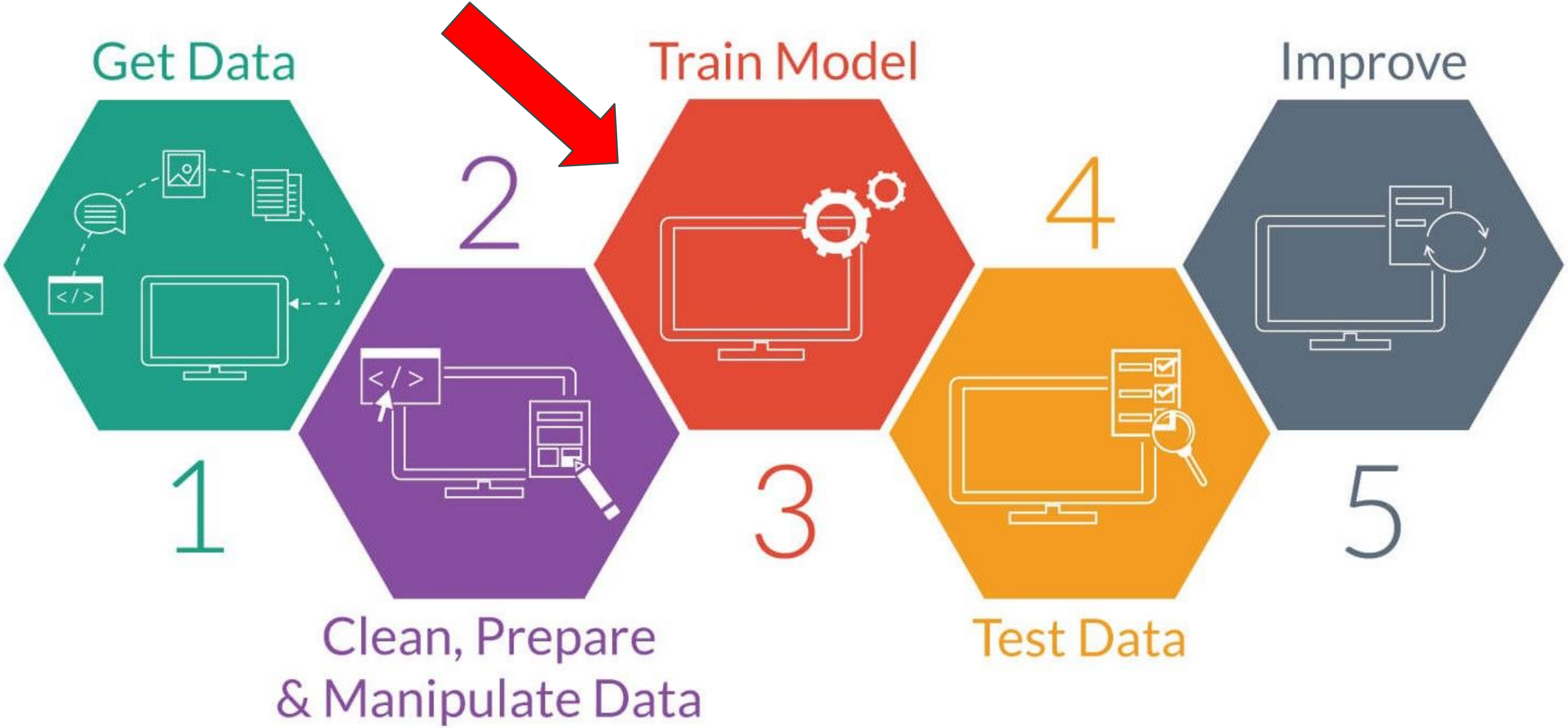
```python
pd.concat([X_train,Y_train],axis=1).corr()["SalePrice"].sort_values()
```




```
Overall Cond    -0.186210
Yr Sold          0.017669
Lot Area         0.305555
Garage Area      0.632527
Year Built       0.650917
Gr Liv Area      0.676906
SalePrice        1.000000
```


Sale Price (U$)
300k
200k
100k
0
5k
10k
15k
1000
2000
0
500
1000
Lot Area
Gr Liv Area
Garage Area
Overall Cond
Year Built
Yr Sold
4
5
6
7
1900
1950
2000
2006
2007
2008
2009
2010

A general ML workflow



Linear Regression with **One Variable**

Notation:

- m - number of training examples
- X's - input variable/features
- y's - output variable/ target variable

Gr Liv Area	SalePrice
1173	170000
1096	138800
1012	127500
1797	231000
1436	225000

X — y

m = 1202

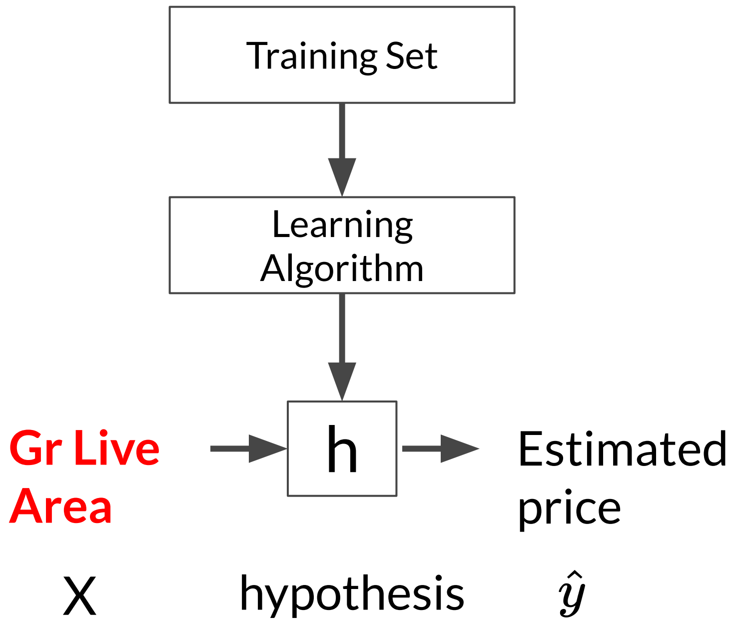
$X^{(1)} = 1173$ $y^{(1)} = 170000$

$X^{(2)} = 1096$ $y^{(2)} = 138800$

$X^{(3)} = 1012$ $y^{(3)} = 127500$

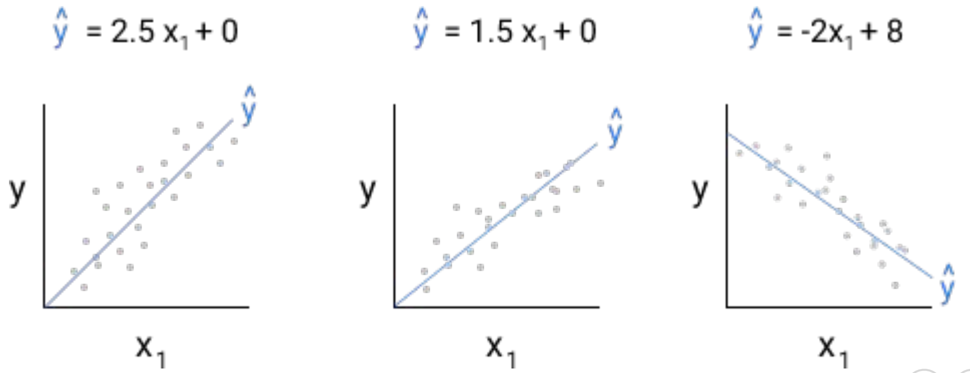
$(X^{(i)}, y^{(i)})$ = i^{th} training example

Model Representation (Linear Reg. **One Variable**)



How do we represent h?

$$\hat{y} = h_\theta(x) = \theta_1 x_1 + \theta_0$$



Cost Function

$f(x)$

"minimize the error"

Cost Function (Linear Reg. One Var.)

Hypothesis $h_\theta(x) = \theta_0 + \theta_1 x$

θ_i = parameters

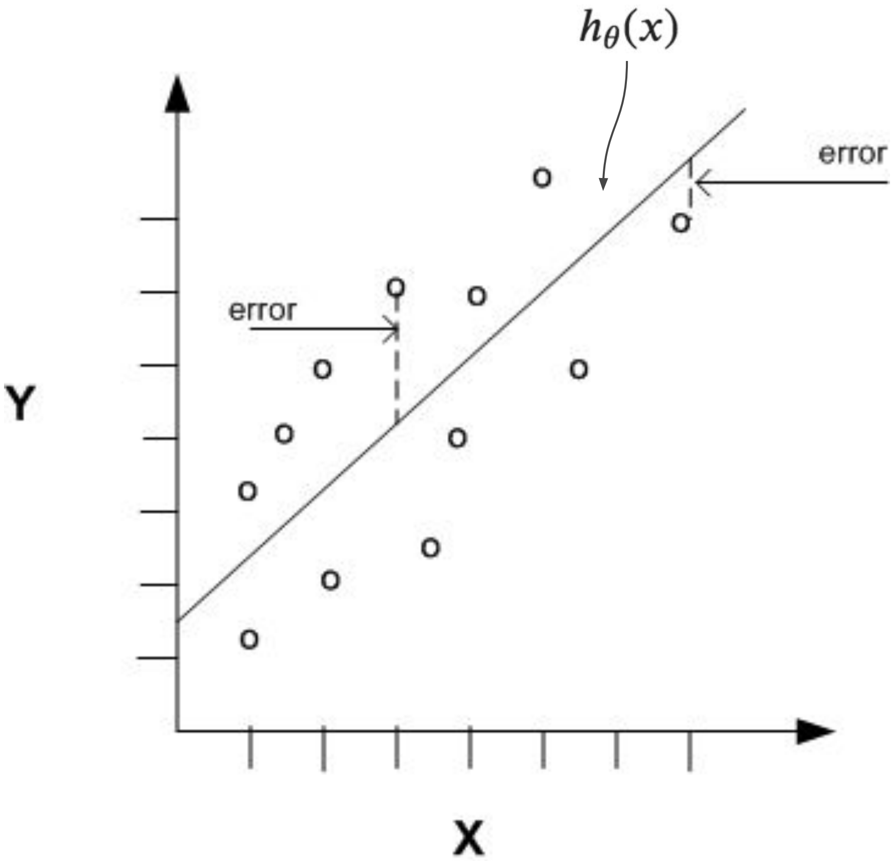
How to choose θ_i ?

m = 1202

X	y
Gr Liv Area	**SalePrice**
1173	170000
1096	138800
1012	127500
1797	231000
1436	225000

Cost Function Intuition #01 (Linear Reg. One Var.)

Cost Function

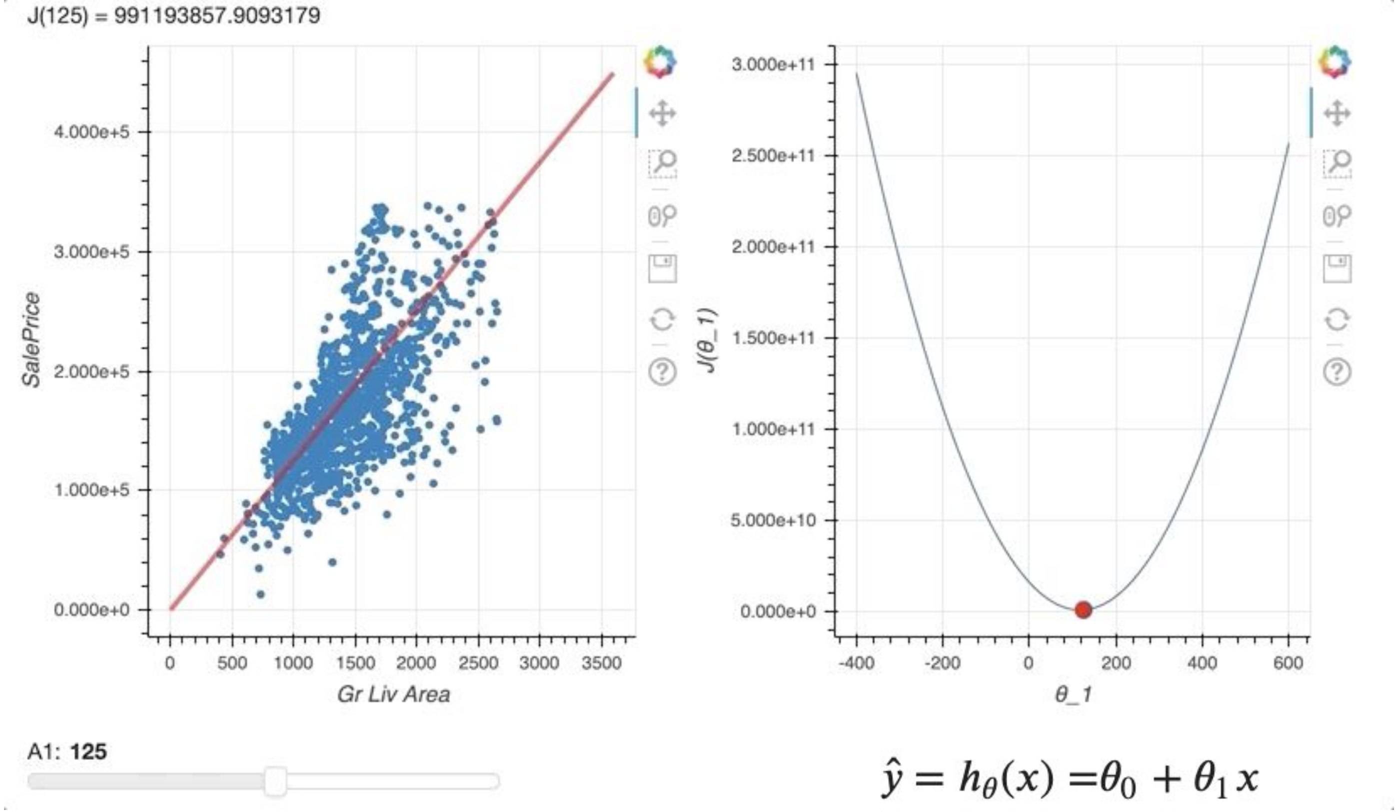


$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left[h_\theta(x^{(i)}) - y^{(i)}\right]^2$$

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Idea:

- choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples $(x^{(i)},y^{(i)})$
- minimize (θ_0, θ_1)



$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$$

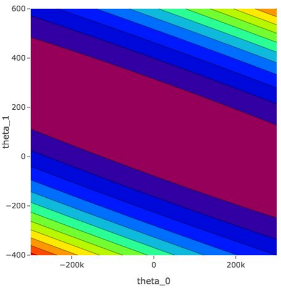
$$\hat{y} = h_\theta(x) = \theta_1 x$$

Cost Function Intuition #01

($\theta_0 = 0$)

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left[\theta_1 x^{(i)} - y^{(i)}\right]^2$$

Cost Function Intuition #02 (Linear Reg. One var)

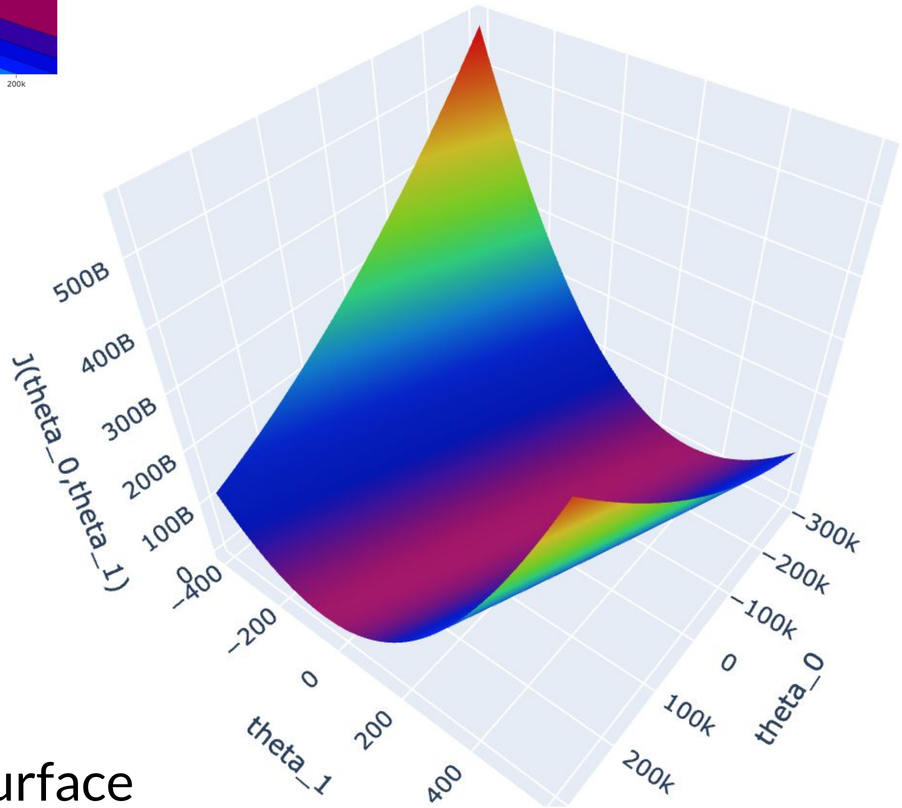


Contour

Cost Function
Intuition #02
(θ_0 and θ_1 are defined)

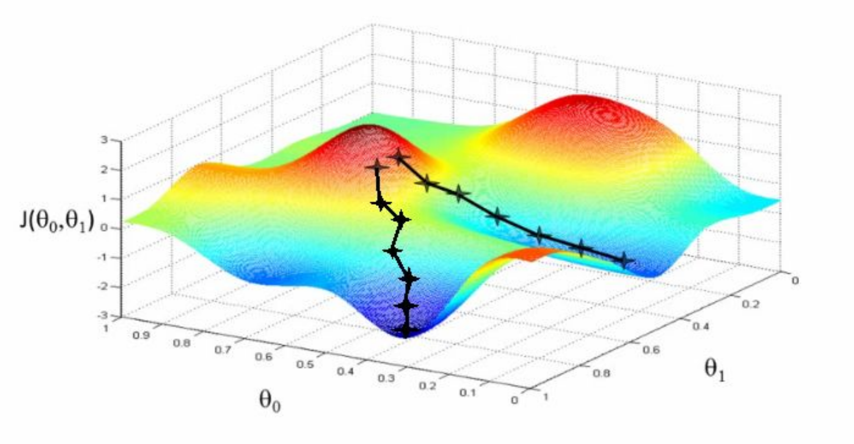
$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left[h_\theta(x^{(i)}) - y^{(i)} \right]^2$$

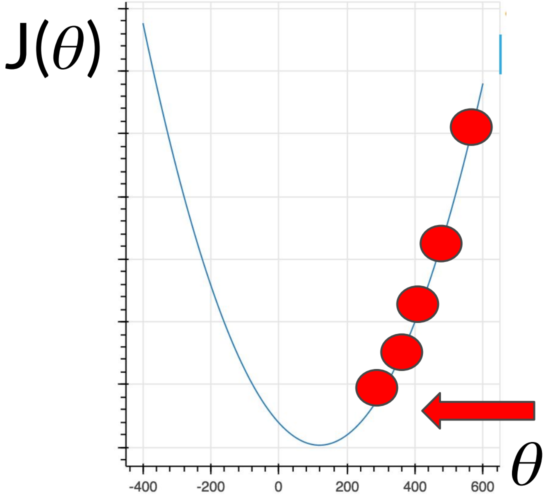


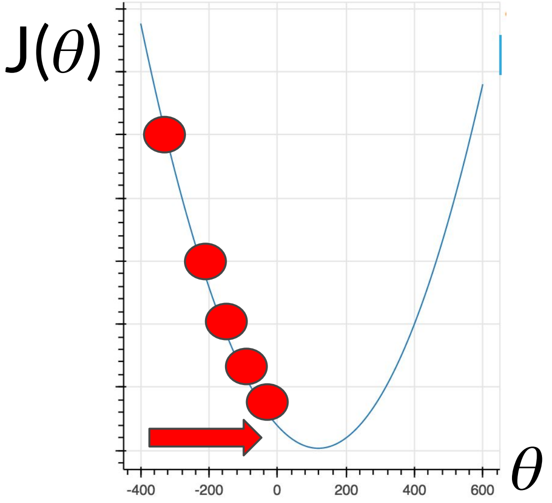
Surface

Gradient Descent (Linear Reg. One var)



1. Iteratively
   1.1. Evaluate parameters
   1.2. Compute loss
   1.3. Take small steps in the direction that will minimize loss





repeat until converge {

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

}

α - learning rate

repeat until converge {

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

}

Correct update

$$aux_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$aux_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_0 = aux_0$$

$$\theta_1 = aux_1$$

Incorrect update

$$aux_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\theta_0 = aux_0$$

$$aux_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_1 = aux_1$$

repeat until converge {

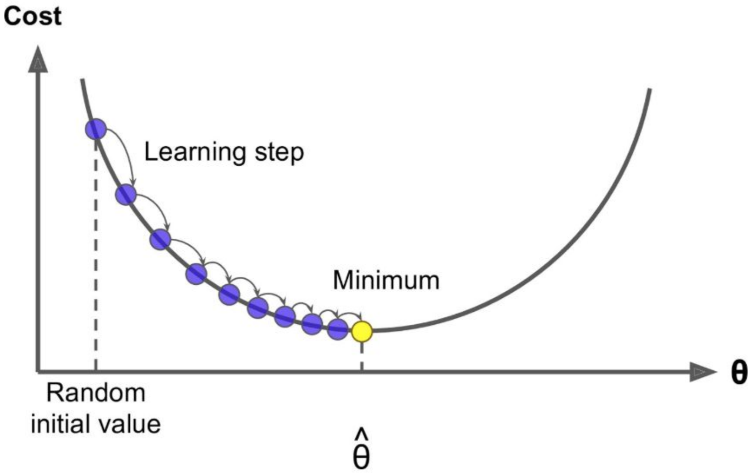
$$aux_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left[h_\theta(x^{(i)}) - y^{(i)}) \right]$$

$$aux_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left[h_\theta(x^{(i)}) - y^{(i)}) \right] x^{(i)}$$

$$\theta_0 = aux_0$$

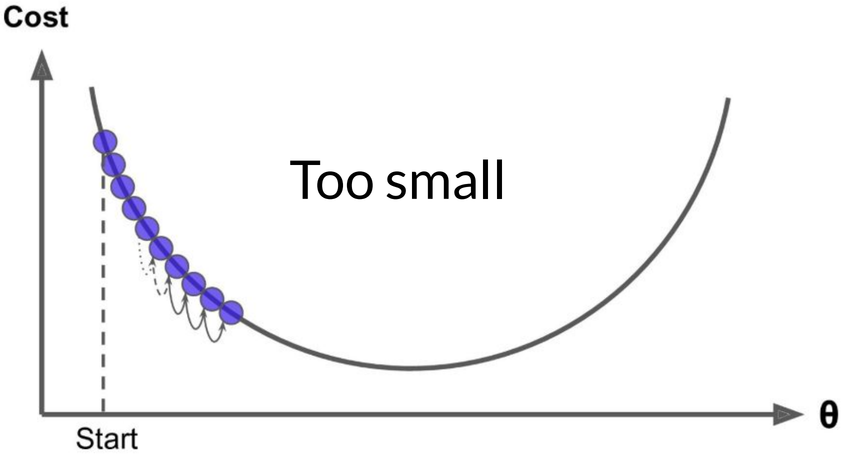
$$\theta_1 = aux_1$$

}

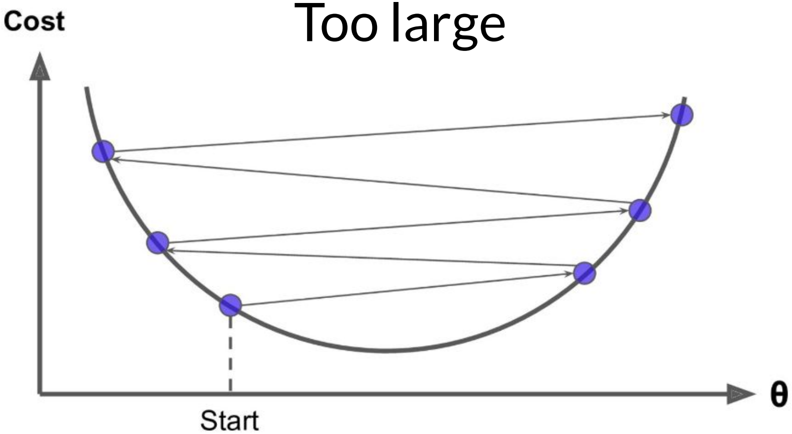


Learning rate tradeoff

Too small

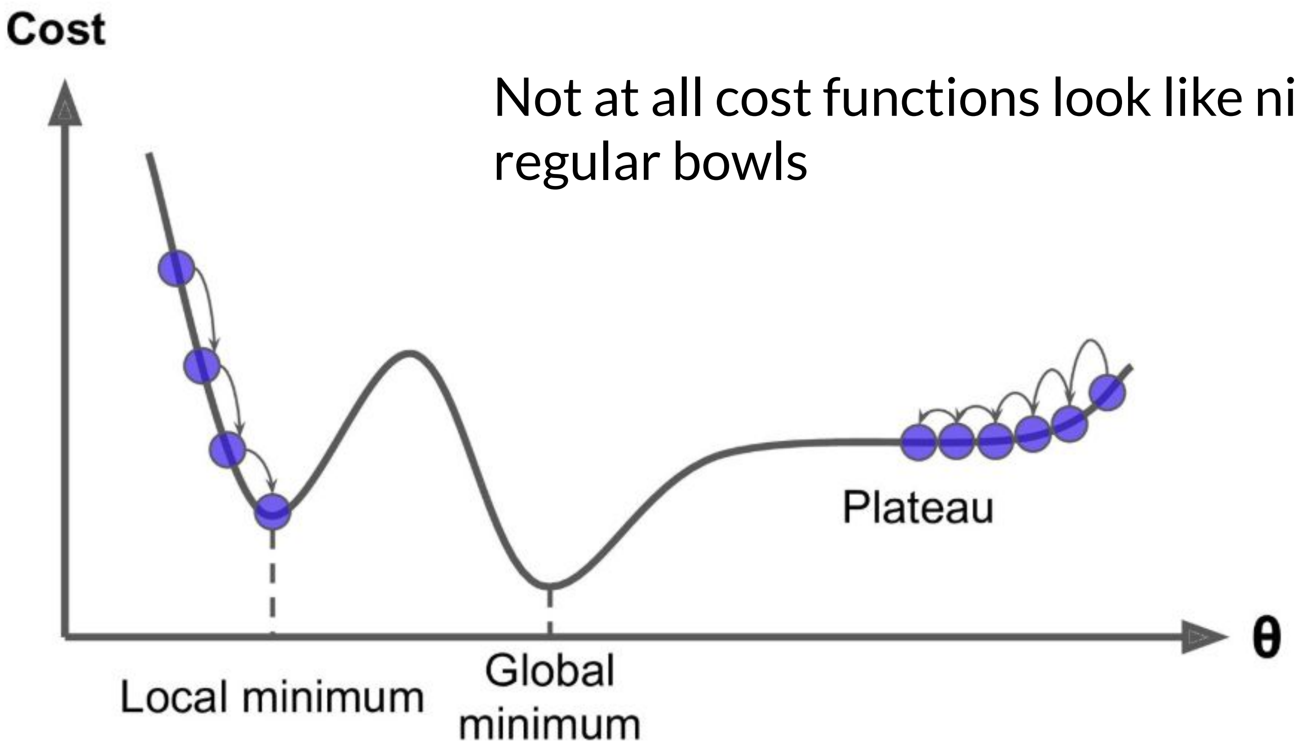


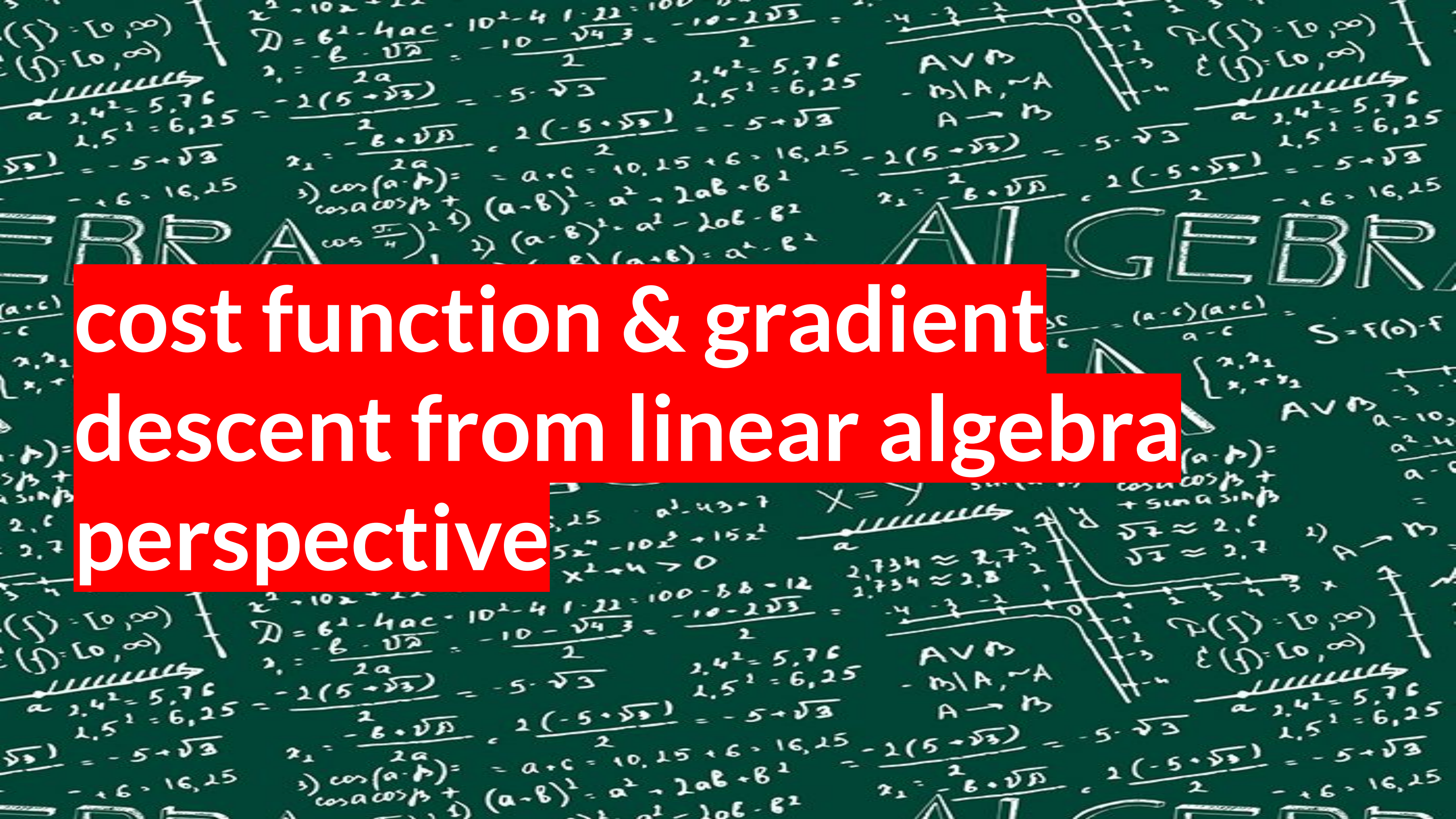
Too large



Gradient Descent Pitfalls

Not at all cost functions look like nice regular bowls




cost function & gradient descent from linear algebra perspective

Hypothesis

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$$

Gr Liv Area	SalePrice
2480	205000
1829	237000
2673	249000
1005	133500
1768	224900

Export to plot.ly »

$$hypothesis = \begin{bmatrix} 1 & 2480 \\ 1 & 1829 \\ 1 & 2679 \\ 1 & 1005 \\ 1 & 1768 \end{bmatrix} \times \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} 2480\ \theta_1 + \theta_0 \\ 1829\ \theta_1 + \theta_0 \\ 2679\ \theta_1 + \theta_0 \\ 1005\ \theta_1 + \theta_0 \\ 1768\ \theta_1 + \theta_0 \end{bmatrix}$$

```python
def cost_function(X, y, theta):
    return np.sum(np.square(np.matmul(X, theta) - y)) / (2 * len(y))
```

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left[h_\theta(x^{(i)}) - y^{(i)}\right]^2$$

Gr Liv Area	SalePrice
2480	205000
1829	237000
2673	249000
1005	133500
1768	224900

Export to plot.ly »

$$J(\theta_0, \theta_1) = \frac{1}{2 \times 5} \sum \left(\begin{bmatrix} 2480\ \theta_1 + \theta_0 \\ 1829\ \theta_1 + \theta_0 \\ 2679\ \theta_1 + \theta_0 \\ 1005\ \theta_1 + \theta_0 \\ 1768\ \theta_1 + \theta_0 \end{bmatrix} - \begin{bmatrix} 205000 \\ 237000 \\ 249000 \\ 133500 \\ 224900 \end{bmatrix} \right)^2$$

repeat until converge {

$$aux_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left[h_\theta(x^{(i)}) - y^{(i)} \right]$$

$$aux_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left[h_\theta(x^{(i)}) - y^{(i)} \right] x^{(i)}$$

$$\theta_0 = aux_0$$

$$\theta_1 = aux_1$$

}

```
def gradient_descent(X, y, alpha, iterations, theta):
    m = len(y)
    cost_history = []

    for i in range(iterations):
        t0 = theta[0] - (alpha / m) * np.sum(np.dot(X, theta) - y)
        t1 = theta[1] - (alpha / m) * np.sum((np.dot(X, theta) - y) * X[:,1])
        theta = np.array([t0, t1])
        cost_history.append(cost_function(X, y, theta))

    return theta, cost_history
```

- Involves calculations over the full training set X at each gradient step
- It uses the whole batch of training data at every step.
- This is why the algorithm called **Batch Gradient Descent**

1. Batch gradient descent: use all **m examples** in each iteration
2. Stochastic gradient descent: use **1 example** in each iteration
3. Mini-batch gradient descent: use **b examples** in each iteration

Stochastic gradient descent

Randomly shuffle (reorder) training examples

Repeat {

for $i := 1, \ldots, m$ {

$$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

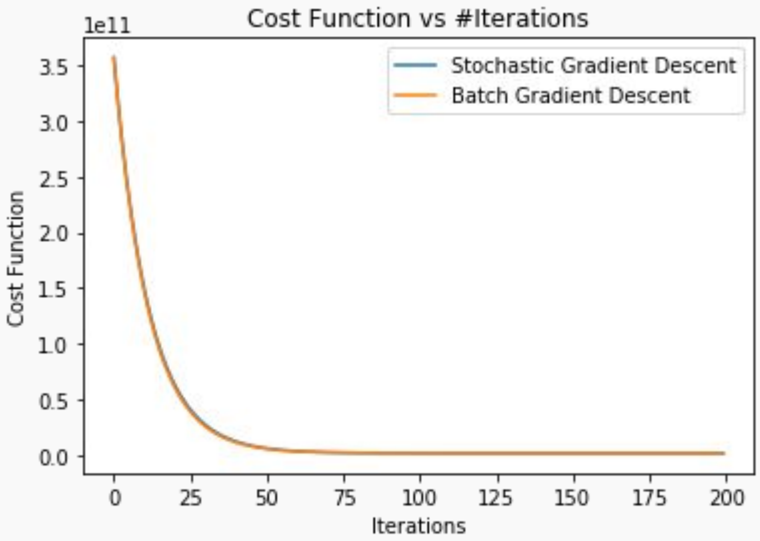
(for every $j = 0, \ldots, n$)

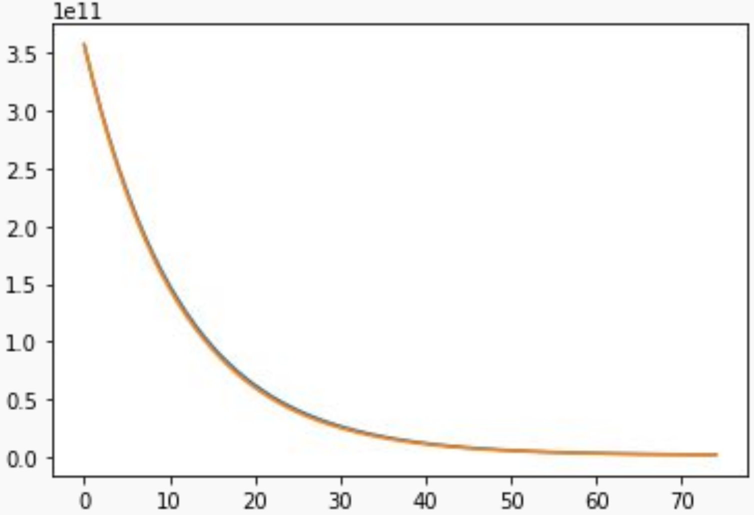
}

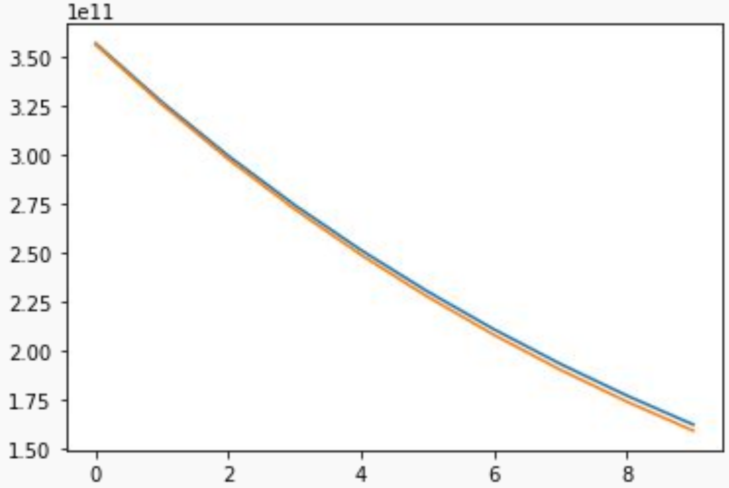
}

Stochastic gradient descent

```
from sklearn.utils import shuffle

# stochastic gradient descent
def stochastic_gradient_descent(X, y, alpha, iterations, theta):
  m = len(y)
  cost_history = []

  # ramdomly suffle the training dataset
  X,y = shuffle(X,y,random_state=42)

  for i in range(iterations):
    for j in range(m):
      t0 = theta[0] - alpha * np.sum(np.dot(X[j,:], theta) - y[j])
      t1 = theta[1] - (alpha / m) * np.sum((np.dot(X[j,:], theta) - y[j]) * X[j,1])
      theta = np.array([t0, t1])
    cost_history.append(cost_function(X, y, theta))
  return theta, cost_history
```







Stochastic Gradient Descent

- Pro: faster learning, can avoid local minima
- Cons: computationally expensive

Batch Gradient Descent

- Pro: computationally efficient, stable convergence
- Cons: memory--

Mini-Batch gradient descent

Say $b = 10, m = 1000$.

Repeat {

for $i = 1, 11, 21, 31, \ldots, 991$ {

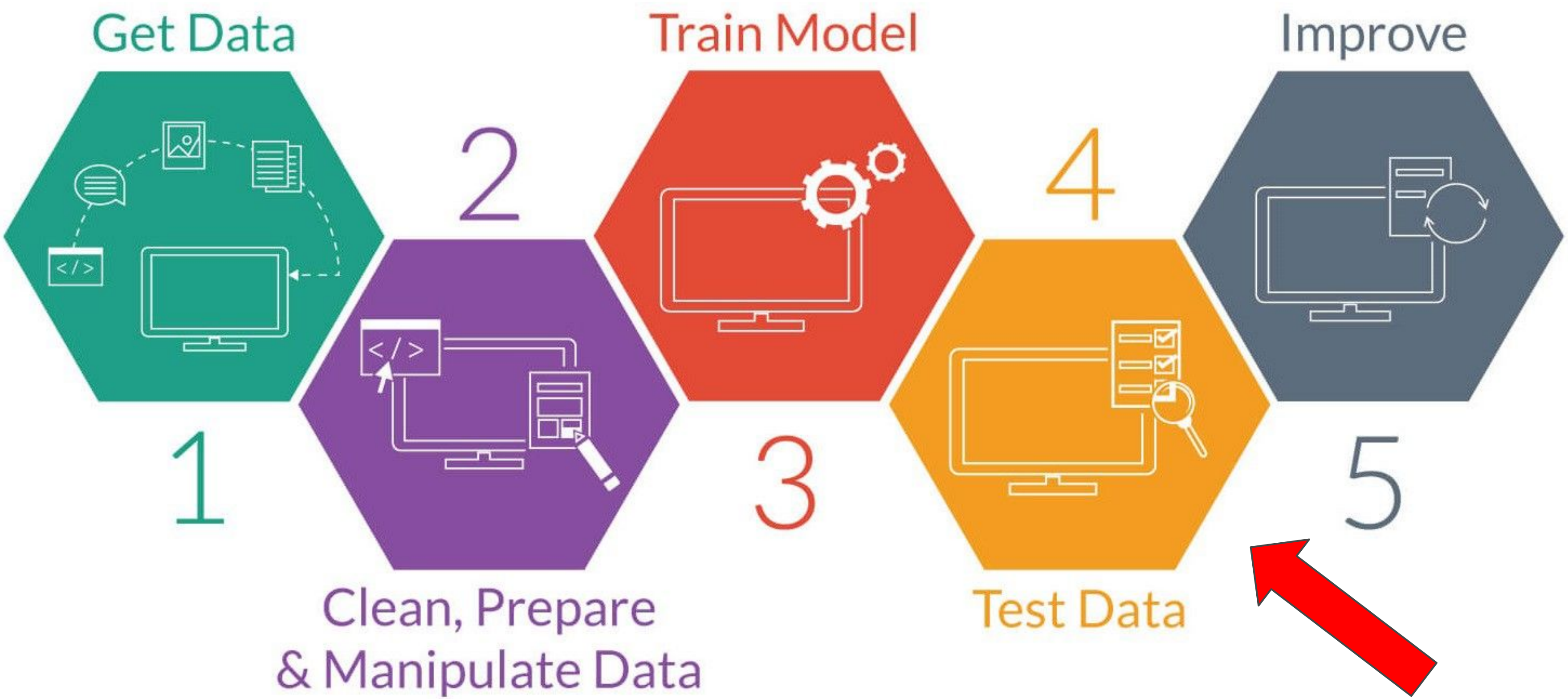
$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every $j = 0, \ldots, n$)

}

}

A general ML workflow



The Linear Regression Model.ipynb
Section #01

- We are going to start by covering linear regression
  - Multiple variables
- We discuss the application of linear regression to **housing price prediction**

Linear Regression with Multiple Variables

Notation:

- m - number of training examples
- n - number of features
- $x^{(i)}$ - input features of i^{th} training example
- $x_j^{(i)}$ - value of feature j in i^{th} training example
- $y^{(i)}$ - target value of i^{th} training examples

$$x^{(2)} = \begin{bmatrix} 11622 \\ 5 \\ 1961 \\ 2010 \\ 105000 \end{bmatrix}$$

$$x_3^{(2)} = 1961$$

$$n = 4$$

$m = 5$

x_1 Lot Area	x_2 Overall Qual	x_3 Year Built	x_4 Yr Sold	y SalePrice
31770	6	1960	2010	215000
11622	5	1961	2010	105000
14267	6	1958	2010	172000
11160	7	1968	2010	244000
13830	5	1997	2010	189900

Hypothesis (previously)

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Multivariable case

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

For convenience of notation, define $x_0=1$. In other words:

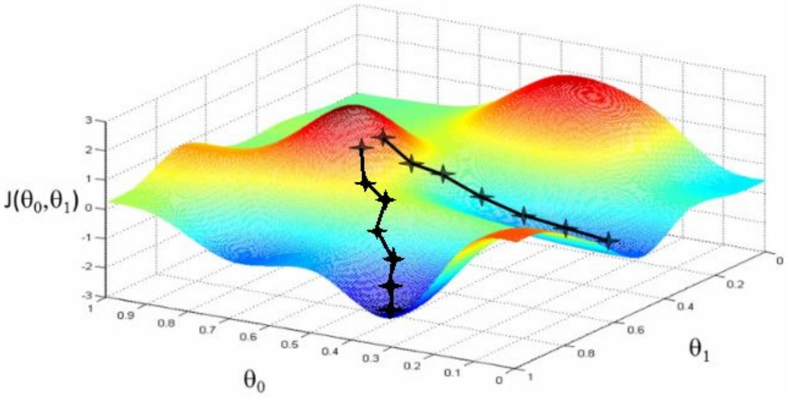
$x_0^{(i)}=1$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \theta_1 & \dots & \theta_n \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

Gradient Descent
(Linear Reg. Multiple Variables)



Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \theta_2, \ldots, \theta_n$

Cost function:

$$J(\theta_0, \theta_1, \theta_2, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Gradient descent:

repeat {

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

} Simultaneously update for every j (0 to n)

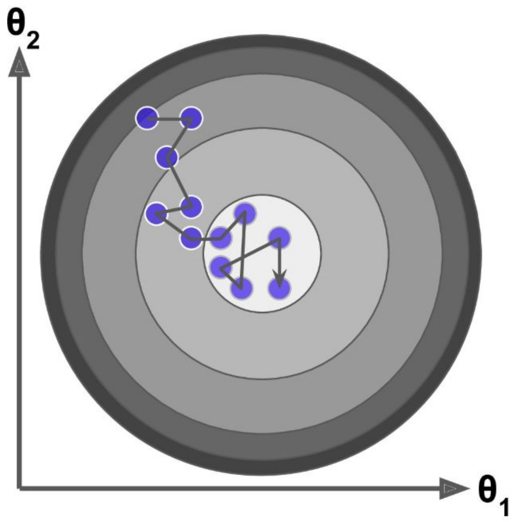
Gradient descent:

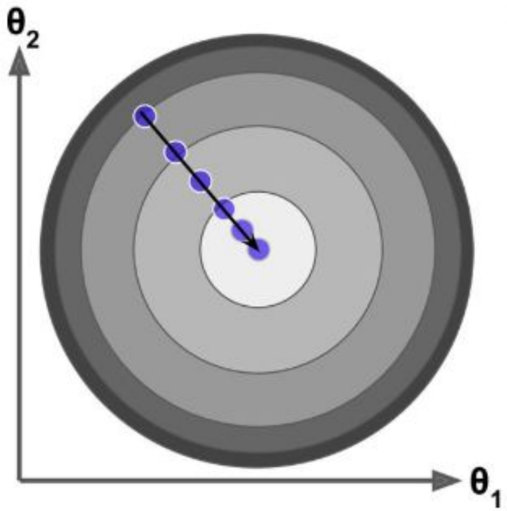
repeat until the convergence {

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \qquad \text{for } j = 0, \ldots, n$$

}

Gradient Descent: trick #1 - Feature Scaling





Gradient Descent: trick #1 - Feature Scaling

Z-Score or Standardization

$$z = \frac{x - \mu}{\sigma}$$

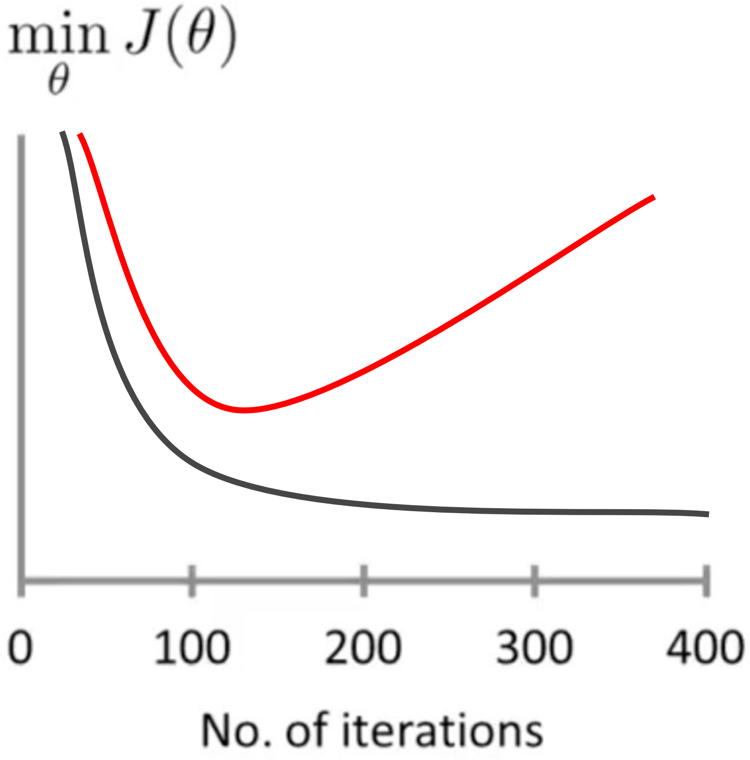
μ - 0
σ - 1

Min-Max Scaling

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

range (0,1)

Gradient Descent: trick #2 - Debugging α



1. Make a plot with number of iterations on the x-axis.
2. Now plot the cost function, J(θ) over the number of iterations of gradient descent.
3. If J(θ) ever increases, then you probably need to decrease α.
4. It has been proven that if learning rate α is sufficiently small, then J(θ) will decrease on every iteration.
5. Automatic convergence test

Gradient Descent: trick #2 - Debugging α

- If α is too small:
  - Slow convergence
- If α is too large:
  - J(Θ) may not decrease on every iteration;
  - J(Θ) may not converge.

To choice α:
…, 0.0001, ….., 0.001, …, 0.01, …, 0.1, …, 1, …, 10, ...

<u>normal equation</u>: method to solve for θ analytically

Normal Equation

Lot Area	Overall Qual	Year Built	Yr Sold	SalePrice
31770	6	1960	2010	215000
11622	5	1961	2010	105000
14267	6	1958	2010	172000
11160	7	1968	2010	244000
13830	5	1997	2010	189900

$$X\theta = y$$
$$X^T X\theta = X^T y$$
$$\theta = (X^T X)^{-1} X^T y$$

$$\begin{bmatrix} 1 & 31770 & 6 & 1960 & 2010 \\ 1 & 11622 & 5 & 1961 & 2010 \\ 1 & 14267 & 6 & 1958 & 2010 \\ 1 & 11160 & 7 & 1968 & 2010 \\ 1 & 13830 & 5 & 1997 & 2010 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix} = \begin{bmatrix} \theta_0 + 31770\theta_1 + 6\theta_2 + 1960\theta_3 + 2010\theta_4 \\ \theta_0 + 11622\theta_1 + 5\theta_2 + 1961\theta_3 + 2010\theta_4 \\ \theta_0 + 14267\theta_1 + 6\theta_2 + 1958\theta_3 + 2010\theta_4 \\ \theta_0 + 11160\theta_1 + 7\theta_2 + 1968\theta_3 + 2010\theta_4 \\ \theta_0 + 13830\theta_1 + 5\theta_2 + 1997\theta_3 + 2010\theta_4 \end{bmatrix}$$

m **training examples**, n **features**

Gradient Descent

- Need to choose α
- Needs many iterations
- Works well even when *n* is large

Normal Equation

- No need to choose α
- Don't needs to iterate
- Need to compute $(X^TX)^{-1}$
- Slow if *n* is very large

If (X^TX) is noninvertible, the common causes might be having:

- Redundant features, where two features are very closely related (i.e. they are linearly dependent)
- Too many features (e.g. m <= n). In this case, dele some features or use "regularization" (to be explained in a later lesson)