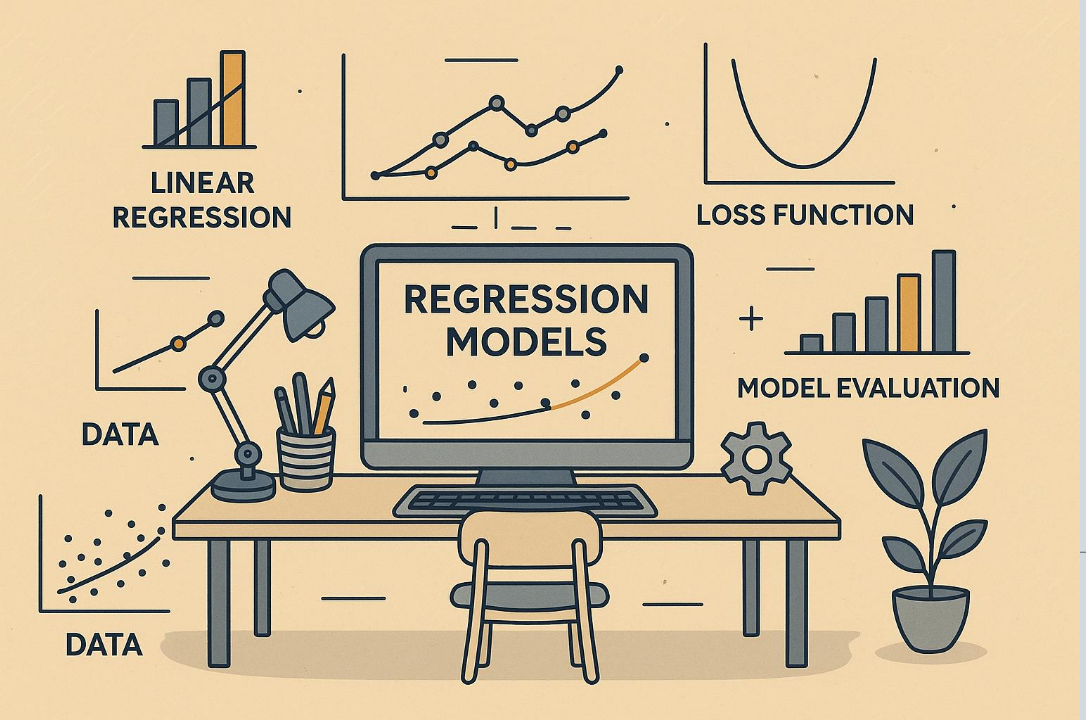
# Regression Models




DR. ZVI BEN AMI

# Agenda

- Fitting a Model to Data
- Modeling types
- Linear regression
- Curve fitting
- Gradient Descent
- Regularization
- Logistic regression

# Fitting a Model to Data

- We build (train) our model using historical data.
- We want to find the “optimal” model parameters given our training data.
- In practice, we need to find the hyperparameters that will predict our target variable as closely as possible, using our available features.
- The process of finding such hyperparameters or patterns in our data is called “fitting”.
- model.fit(features,target)

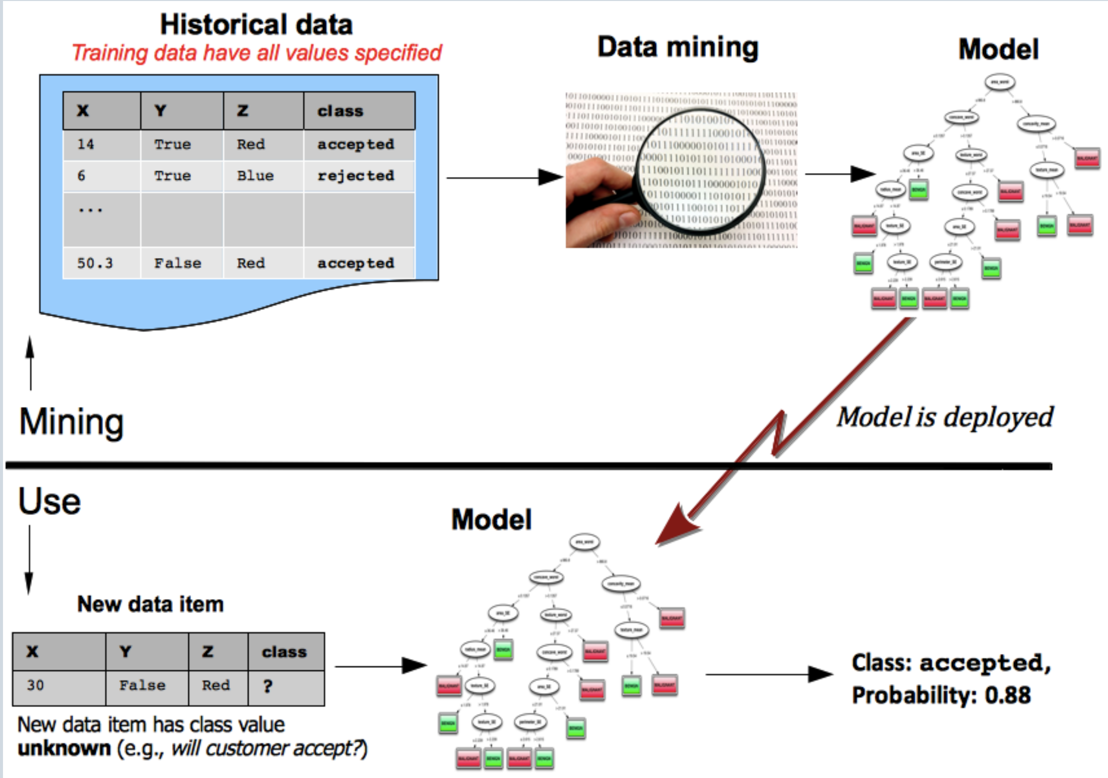



Image from “Data Science for Business”, Provost, Fawcett, 2013

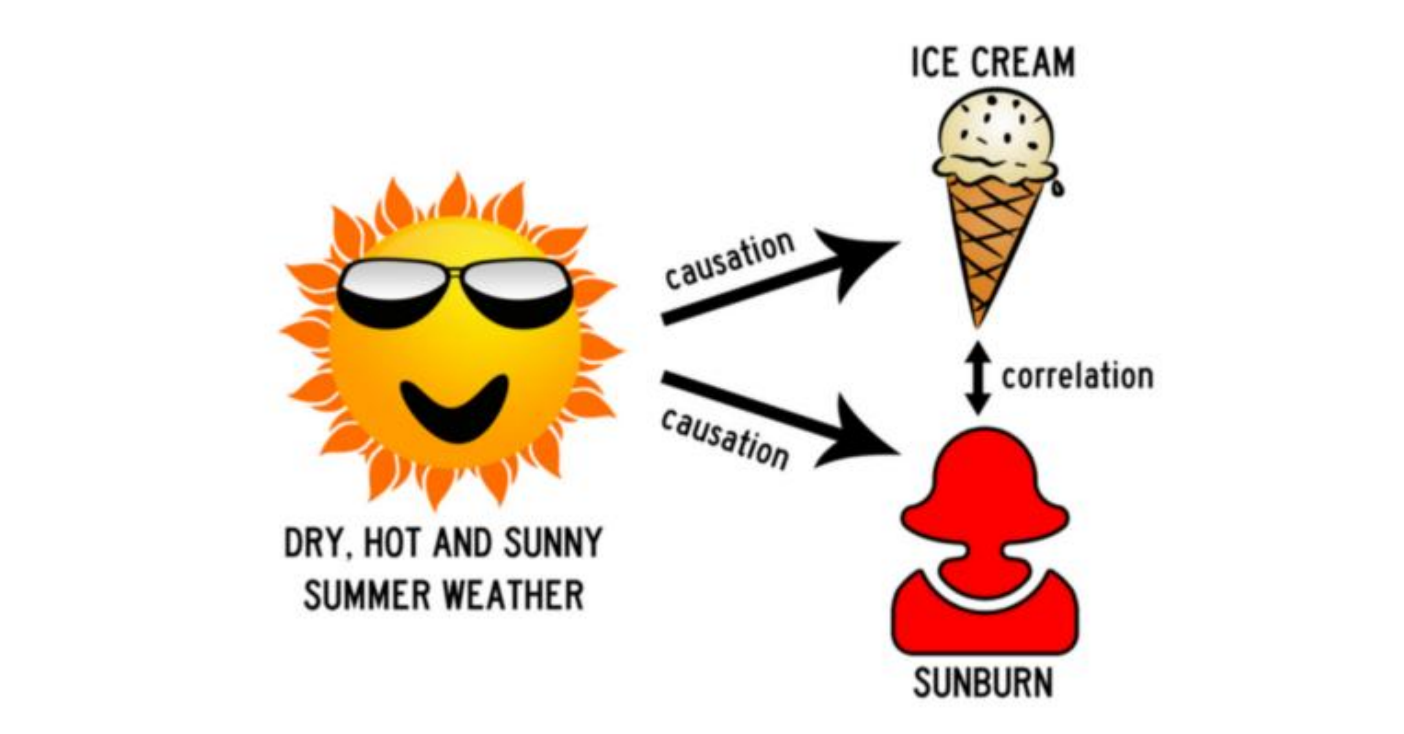
# What is the Purpose of our Model?

- Descriptive Modeling:
  - Summarize or represent the data (EDA)
- Explanatory Modeling:
  - Test causal hypotheses
- Predictive Modeling:
  - Predict new/future observations

Some references:

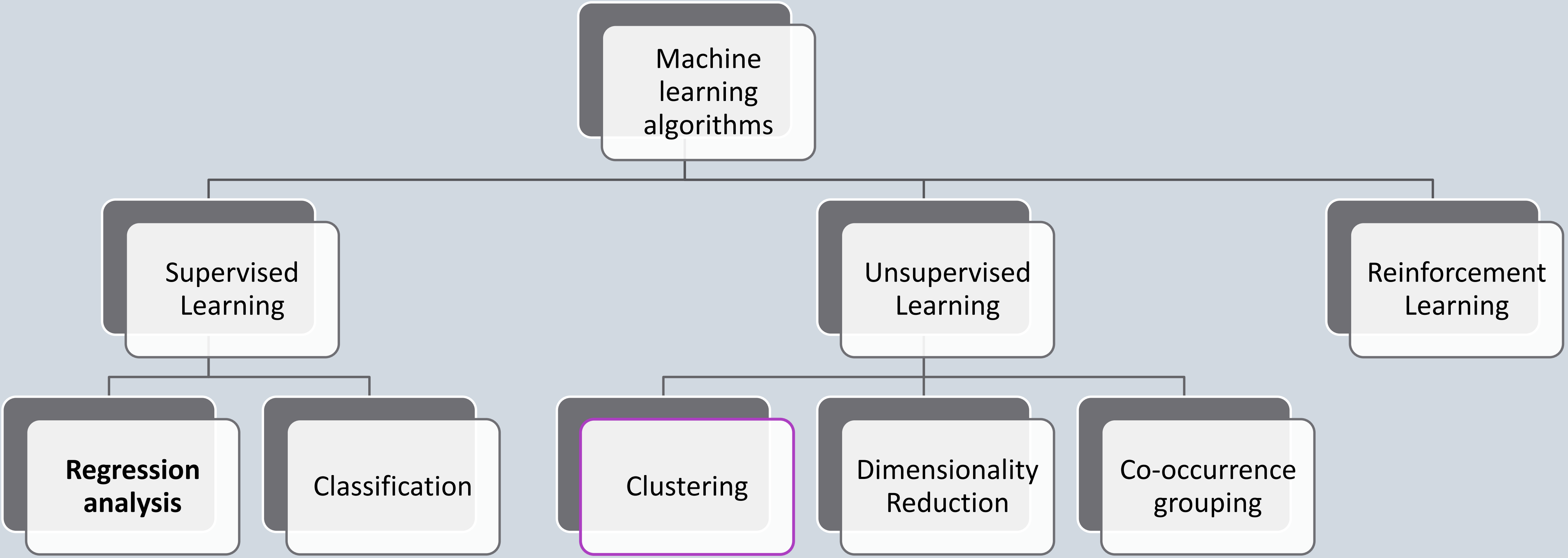
- https://repub.eur.nl/pub/25837/63604908.pdf
- https://www.stat.berkeley.edu/~aldous/157/Papers/shmueli.pdf
- https://onlinelibrary.wiley.com/doi/abs/10.1016/j.pmrj.2014.08.941

# Correlation vs. Causality




Source : https://medium.com/p/66b6cfa702f0

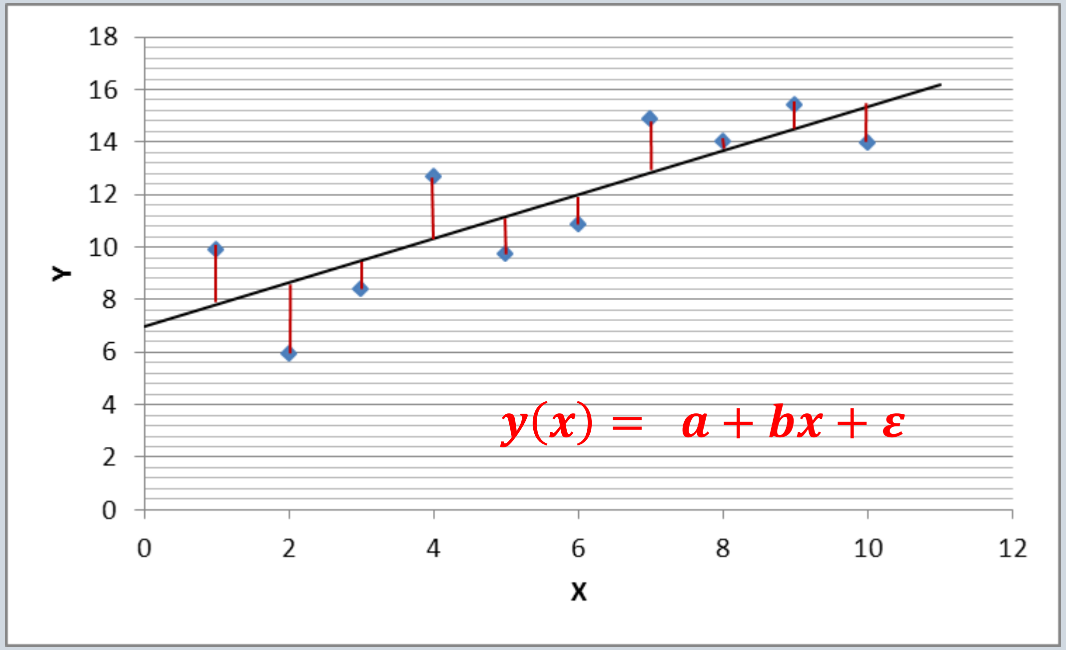
# Model Types


Machine learning algorithms
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Regression analysis
Classification
Clustering
Dimensionality Reduction
Co-occurrence grouping

# Linear Regression

- Model the relation between a numeric target using a regression function that with one or more predicting variables.
- Fit a model to existing (historical) data
- The objective is to find optimal parameters

- Cost Function
  - Mathematical function of the model parameters (α, b1,..) given the (training) data.
  - MSE, RMSE, AMSE, etc.
- Once the cost function is specified, there are various (automated) methods to find the optimal α, b1,… values. For example:
  - Normal equations
  - Gradient descent
- Use fitted model to Predict the numeric target of new data (variable / feature vector)



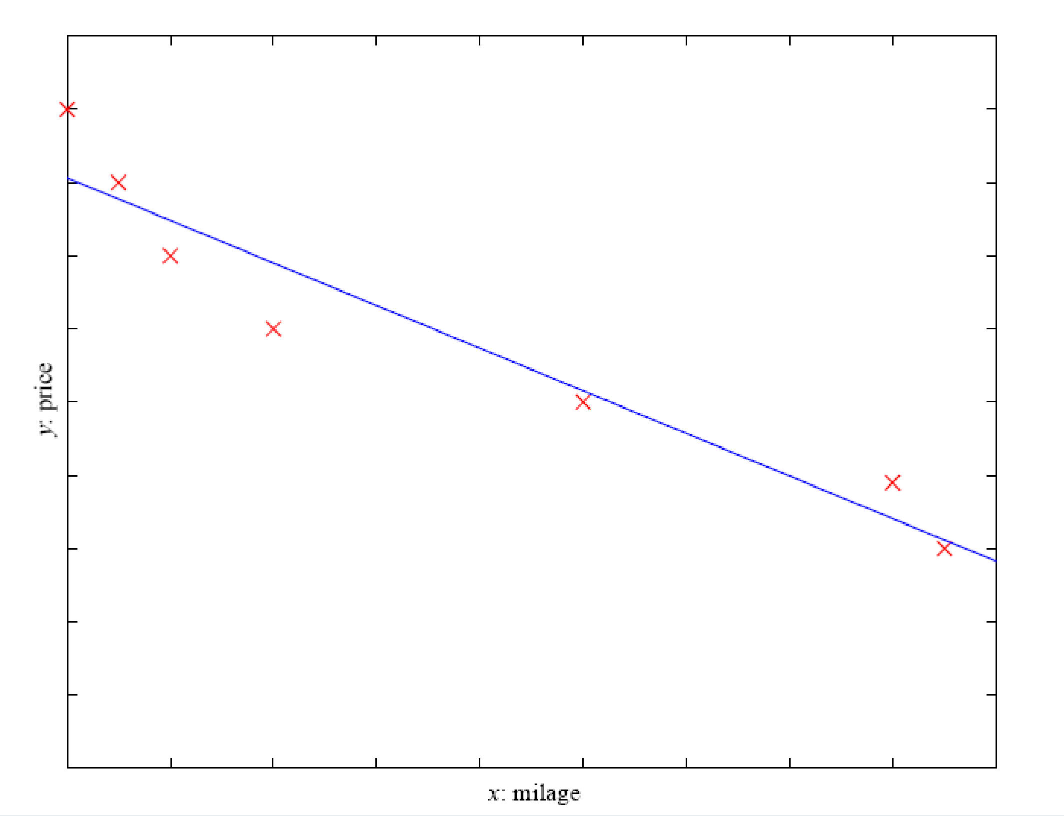
# Linear Regression

Regressions typically applied to a subset of the population and can be used to:

- Learn relationship between parameters
  - i.e. “explain” observed outcomes
  - Note: **not causally!**
- Make predictions for the entire population

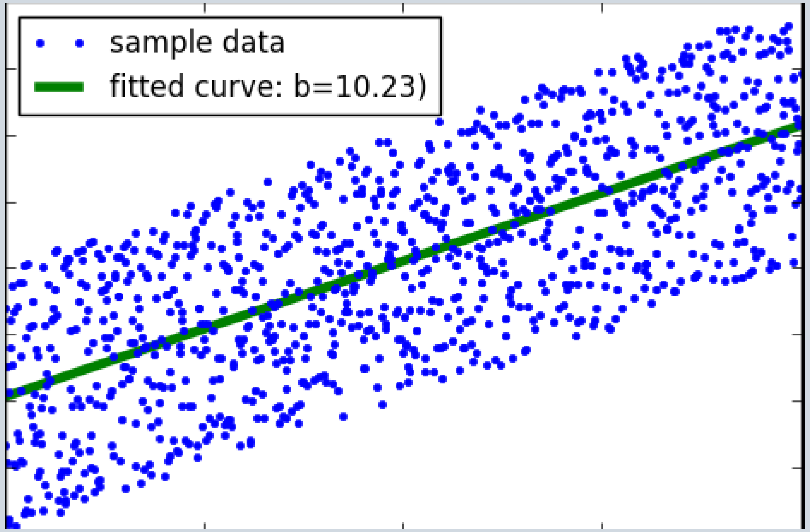
# Supervised Learning: Prediction With Regression

- Example: Price of a used car
  - $x$ : car attributes
  - $y$ : price
  - $y = g(x \mid \beta)$
  - $g(\,)$ model,
  - $\beta$ parameters



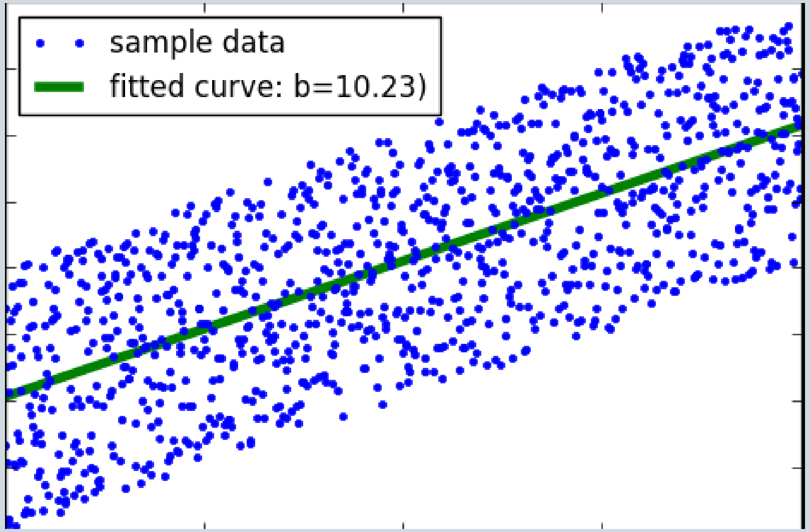
# ML: Curve Fitting

- Curve fitting is the simplest case of machine learning.
- Curve fitting can be used to:
  - Confirm that the data is consistent with a certain pattern and validate the underlying mechanism
  - Discover curve parameters. Useful to
    - draw insights (how x relates to y?) & compare different cases
    - make predictions of what if (e.g. what if we could change how x relates to y by certain percentage)?
  - Design more sophisticated models (regressions)
  - Complement the sample (inter/extra-polate) – predict what kind of patterns will be observed



# ML: Curve Fitting

- Curve fitting goes well beyond linear curves
  - (e.g. high-degree polynomials; exponents, logs and power laws; trigonometric functions; splines)
- Always start with scatter plot of the raw data
- Consider scaling of x and/or y axis for different datasets.



# ML - 1st Order Polynomial

- 1st order polynomial fit **assumes** linear relationship between input (independent variable) - $x$ and output (dependent variable) – $y$.
- Given two vectors, $x$ and $y$, find $a$ and $b$ (scalars) so that

$$y(x) = \quad bx + a + \varepsilon$$

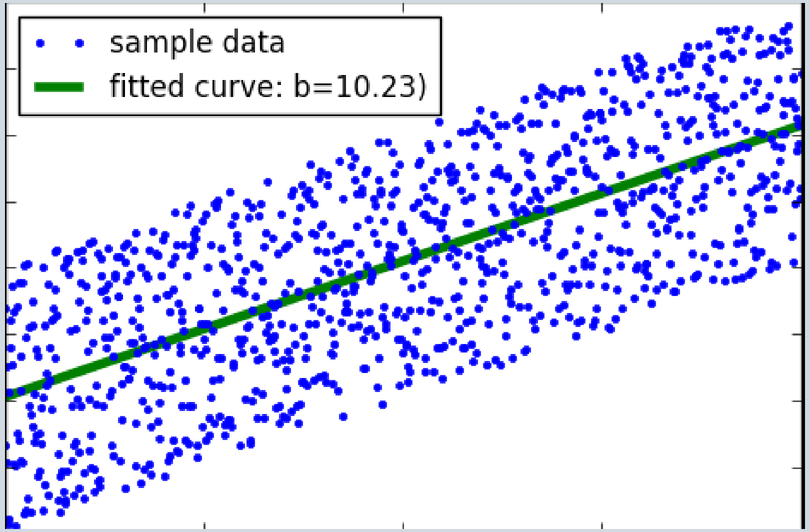
  And $\varepsilon$ is minimal (in some sense, eg. $\sum \varepsilon_i^2$ - least square)

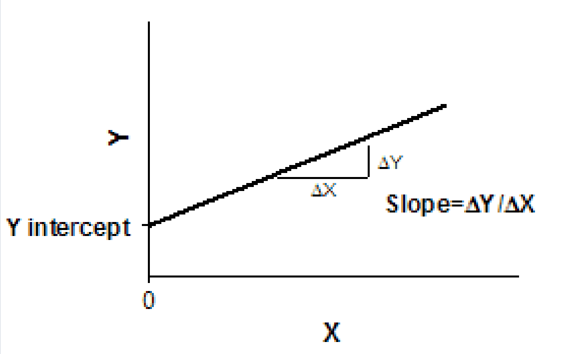
  $a$ – intercept. a is the point at which the fitted line hits y axis (at $x = 0$)

  $b$ – slope of the line. Define how fast $y$ changes with $x$

- Quality of fit depends on the assumption:

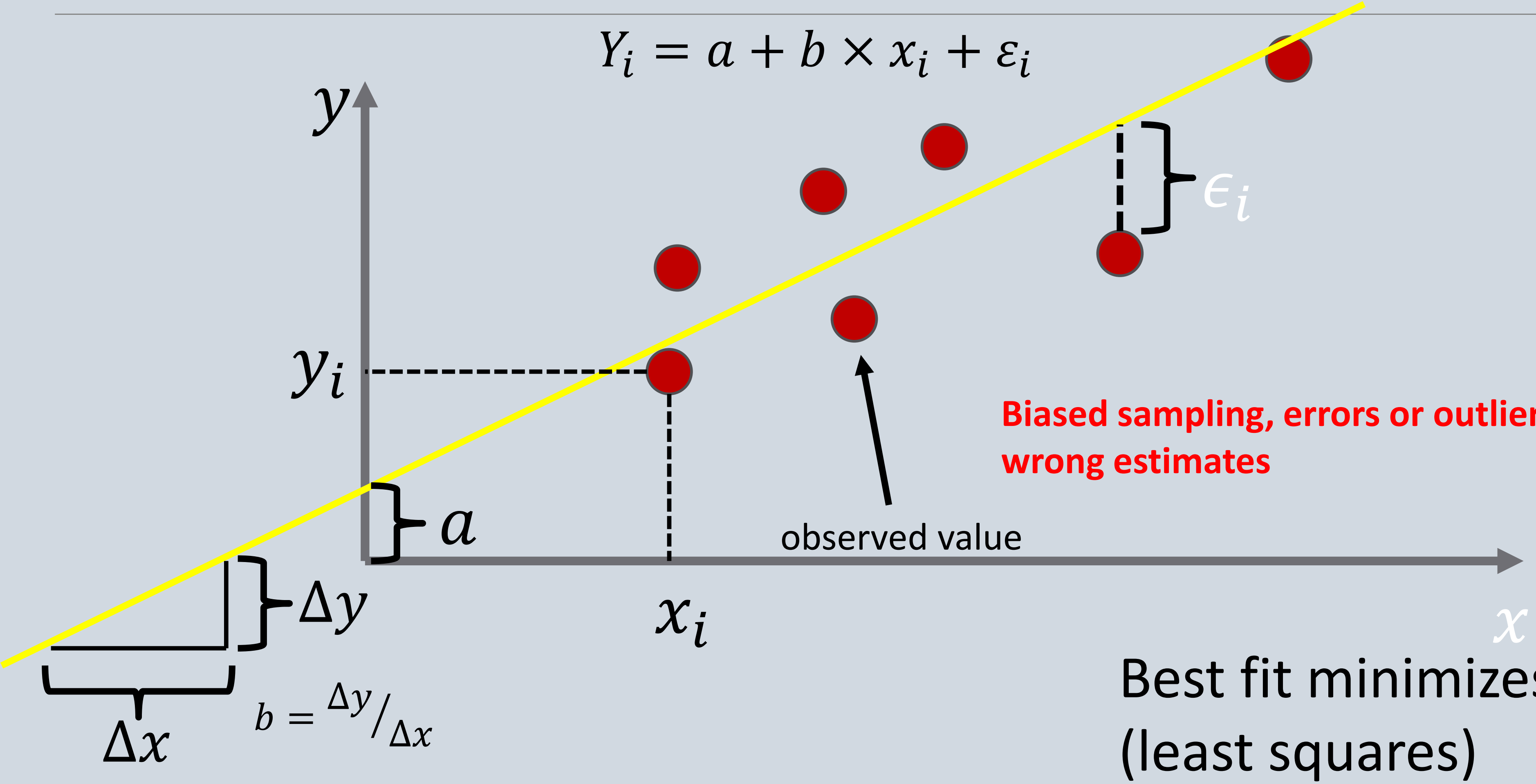
  **data may not be linear**



# ML - 1st Order Polynomial

$$Y_i = a + b \times x_i + \varepsilon_i$$




**Biased sampling, errors or outliers may result in wrong estimates**

Best fit minimizes $\sum_i \varepsilon_i^2$ (least squares)

# Regression Metrics

- $y_i$ – actual value for instance i
- $\hat{y}_i$ – predicted value for instance i
- n – number of instances

Mean Squared Error (MSE) $= \frac{1}{n}\sum_i^n (y_i - \hat{y}_i)^2$

Root Mean Squared Error (RMSE) $= \sqrt{\frac{1}{n}\sum_i^n (y_i - \hat{y}_i)^2}$

Mean Absolute Error (MAE) $= \frac{1}{n}\sum_i^n |y_i - \hat{y}_i|$

Mean Absolute Percentile Error (MAPE) $= \frac{1}{n}\sum_i^n \left|\frac{y_i - \hat{y}_i}{y_i}\right|$

# Linear Regression: Supervised Learning

**Regression_models.ipynb**

- 1st order polynomial fit is a private case of linear regression fitting.

$$y(x) = \ a + bx + \varepsilon$$

  $x$, $y$ and $\varepsilon$ are vectors, a and b are scalars

  The objective is to discover a and b

- General linear regression finds how y depends on several parameters

$$y(x) \sim \ a + \beta x$$

  $x$ is a matrix, $N \times M$

    $N$ – number of observations

    $M$ – number of parameters

  y is a vector, $N \times 1$

  $\beta -$ vector $1 \times M$ – translates how y depends on each parameter in $y$

- $y$ is a **dependent** variable (it depends) or **target** variable
- $x$ *is* ***independent*** *or* ***explanatory*** *variable (helps explaining y) or* ***features***
- *Linear regression is a model!*
- It is essential that the data fits the model

**Simple linear regression** – 1st order polynomial (1 variable)
**Multiple linear regression** – 2 or more explanatory variables

# Ordinary Least Squares (OLS)

- Ordinary Least Squares:
  - A method to estimate the parameters (β) of a linear regression model.
- Goal:
  - Find the line (or hyperplane) that minimizes the sum of squared residuals: $min \sum(y_i - \hat{y}_i)^2$
- Key Idea:
  - Residual = difference between actual value (y) and predicted value (ŷ).
- Why Squared Errors?
  - Penalizes large errors more strongly.
  - Gives a unique, closed-form solution (Normal Equation).
- Result:
  - Provides the "best fit" line under the assumptions of linear regression.

# Interpreting the OLS Output

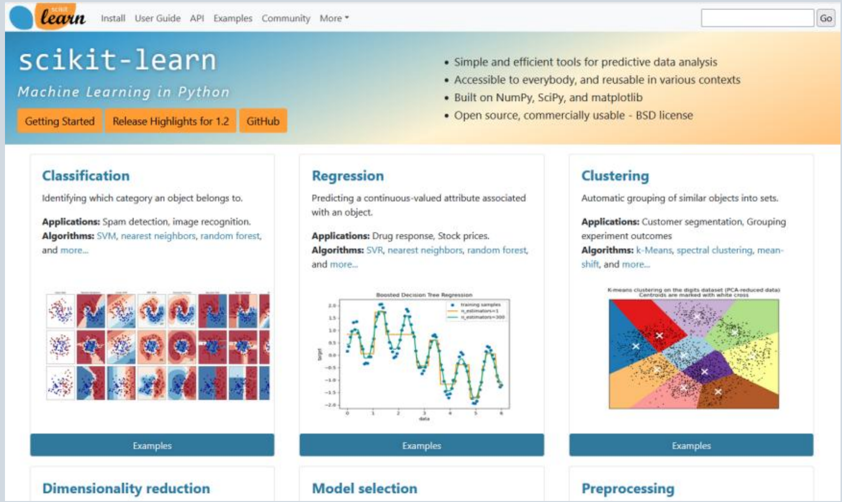
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.894
Model:                            OLS   Adj. R-squared:                  0.894
Method:                 Least Squares   F-statistic:                     4194.
Date:                Fri, 01 Apr 2022   Prob (F-statistic):               0.00
Time:                        16:34:53   Log-Likelihood:                -7742.7
No. Observations:                1000   AIC:                         1.549e+04
Df Residuals:                     997   BIC:                         1.551e+04
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
intercept    -28.1727     26.494     -1.063      0.288     -80.162      23.817
x              9.5522      0.306     31.255      0.000       8.952      10.152
x2             0.5090      0.006     86.089      0.000       0.497       0.521
==============================================================================
Omnibus:                      534.067   Durbin-Watson:                   1.941
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               55.841
Skew:                           0.022   Prob(JB):                     7.49e-13
Kurtosis:                       1.843   Cond. No.                     6.72e+03
==============================================================================
```

- Target variable
- Model type
- Number of observations
- Degrees of Freedom
- Numbers our predicting variables
- Normalcy distribution of residuals (0=prefect)
- Probability residuals normally distributed. (1=prefect)
- Peakiness of data
- Symmetry in data
- R2- How much of the independent variable is explained by changes in our dependent variables
- Adj-R2 – important when there are multiple features
- F-score of the null hypothesis
- Accuracy of the null hypothesis
- measurement of homoscedasticity (uneven distribution)
- Alternate measure for Omnibus
- Measurement of sensitivity

Reference: https://medium.com/swlh/interpreting-linear-regression-through-statsmodels-summary-4796d359035a

# Scikit-learn

- Scikit-learn is a Python library for machine learning and data analysis.
- It provides tools for classification, regression, clustering, and dimensionality reduction.
- Scikit-learn includes algorithms for supervised and unsupervised learning, feature extraction, and model selection and evaluation.
- It has a user-friendly interface and is widely used in industry and academia for various applications.
- Scikit-learn is constantly updated and improved with new features and algorithms.




https://scikit-learn.org/stable/

# Linear Regression Using Scikit-learn

Regression_models.ipynb

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error


data_encoded = pd.get_dummies(data, drop_first=True)
y = data_encoded['price']
X = data_encoded.drop(columns=['price'])


# Fit model
lm = LinearRegression()
lm.fit(X, y)
```

# Gradient Descent – Numeric Optimization

- It is not likely that *loss* function ($C_\beta$) have analytic solution.
- Numeric Analysis solution:
  - Start with arbitrary* solution ($\beta_0$,$\beta_1$)
  - Iteratively change ($\beta_0$,$\beta_1$) to improve the solution (minimize the *loss* function $C_\beta$ )
  - Continue until you reach a minimum point (or termination condition)

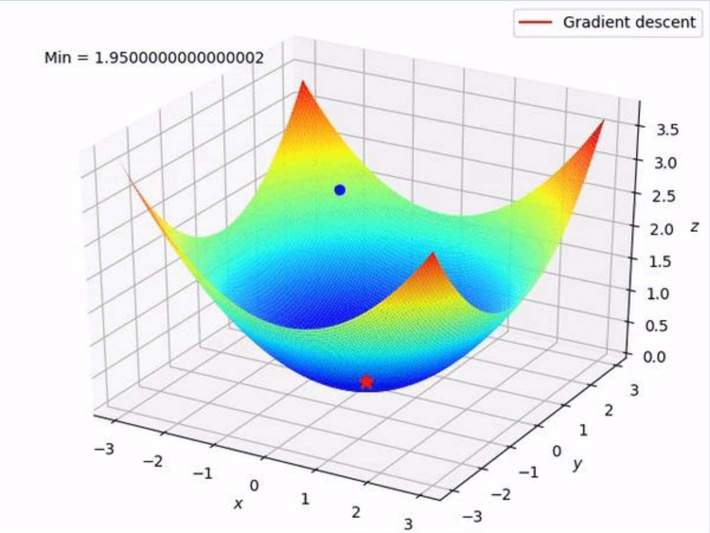
* You may want to use some logic here and assign values in the range you expect the actual βs values to be.

# Gradient Descent

- Cost functions
  - Linear regression:  min(MSE )
  - Logistic regression: min(-Log Llikelihood)

$$LL = \sum_{i=1}^{n} \{ y_i ln(p(x_i)) + (1 - y_i)\left(ln(1 - p(x_i))\right)\}$$

# Gradient Descent

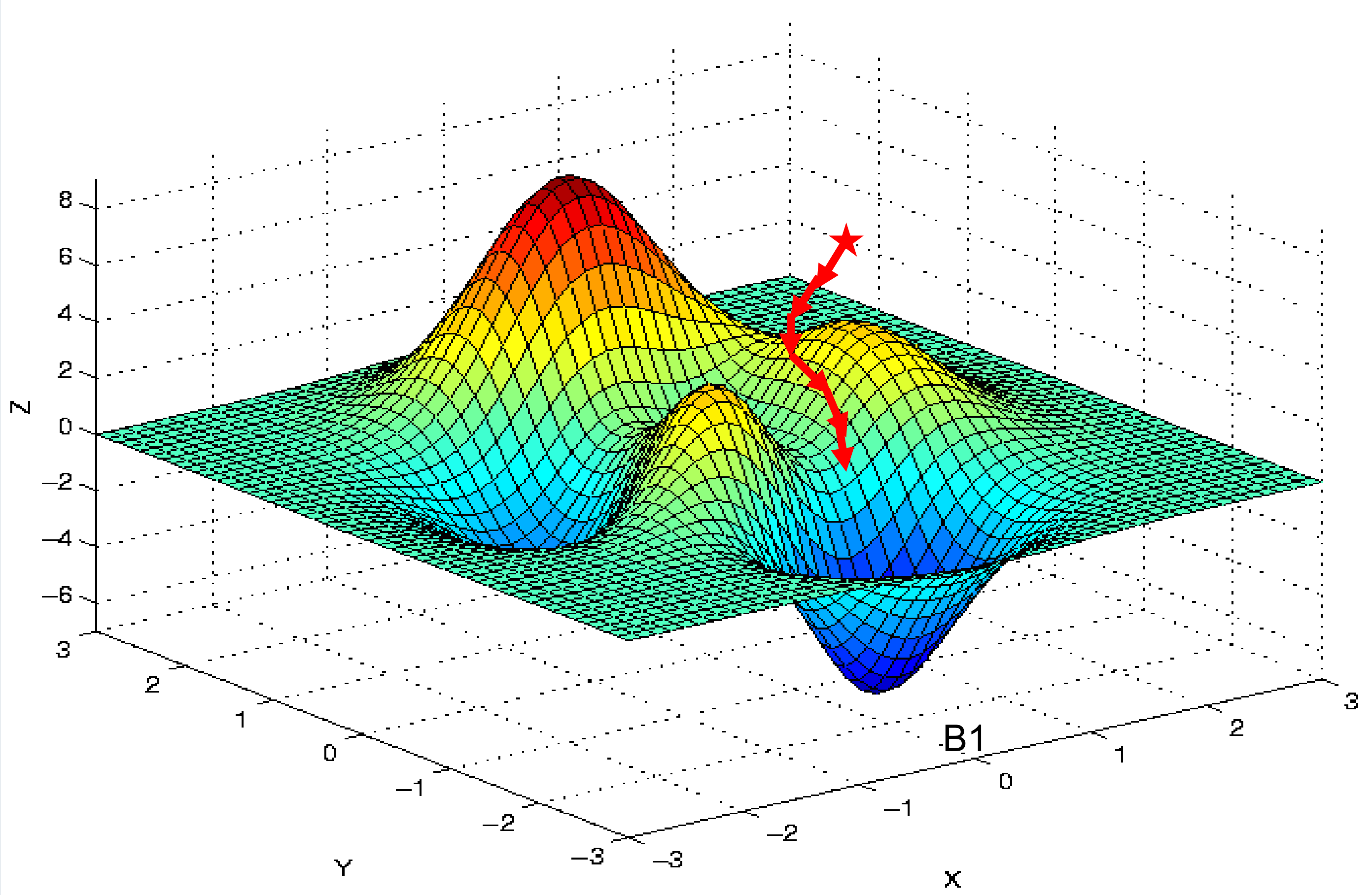



Source: https://jed-ai.github.io/py1_gd_animation/

Source: https://www.youtube.com/watch?v=vWFjqgb-ylQ&feature=youtu.be

Linear regression using gradate descent: https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931

# Gradient Descent



# Gradient Descent

1. Define the "learning rate" $\gamma$
2. Select initial solution $a_0$
3. Calculate the slope of the Loss function at current step (gradient)
   - $\nabla\mathcal{L}(a_i)$
4. Calculate the next step $a_{i+1}$ as
   - $a_{i+1} = a_i - \gamma\nabla\mathcal{L}(a_i)$
5. Termination?
   - If not reached the termination condition go to step 3
   - Else terminate

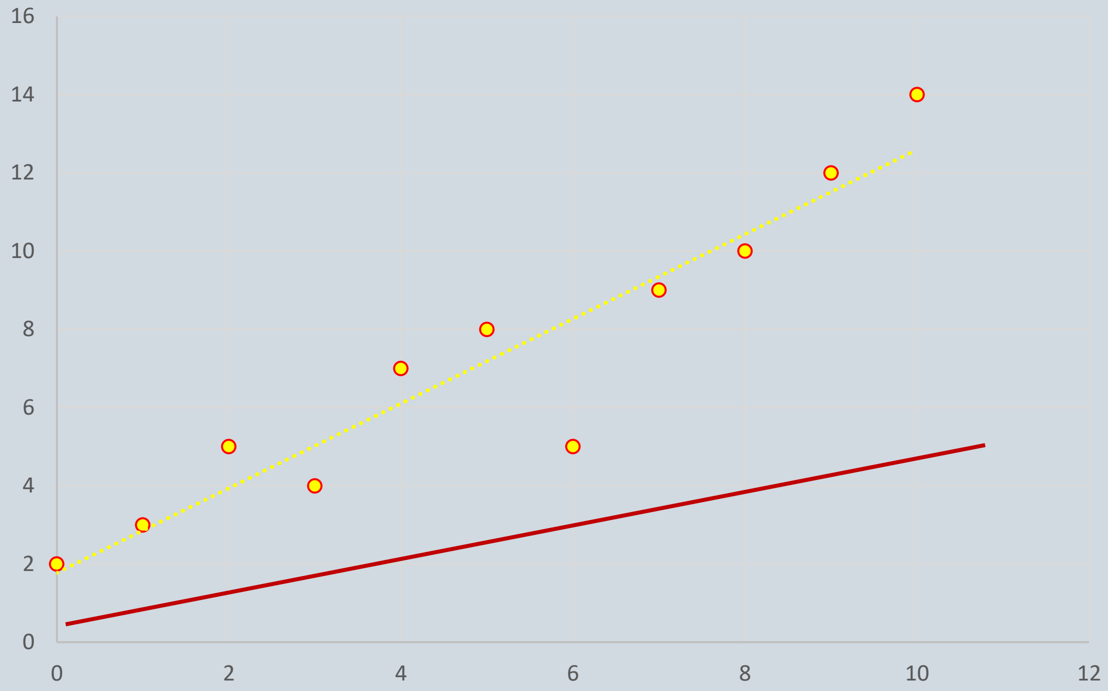
If $\gamma$ is small enough than $\mathcal{L}(a_1) \geq \mathcal{L}(a_2) \geq \mathcal{L}(a_3) \geq \dots$

# Gradient Descent

1. Define the “learning rate” $\gamma$
   - Represent the size of each parameter adjustment

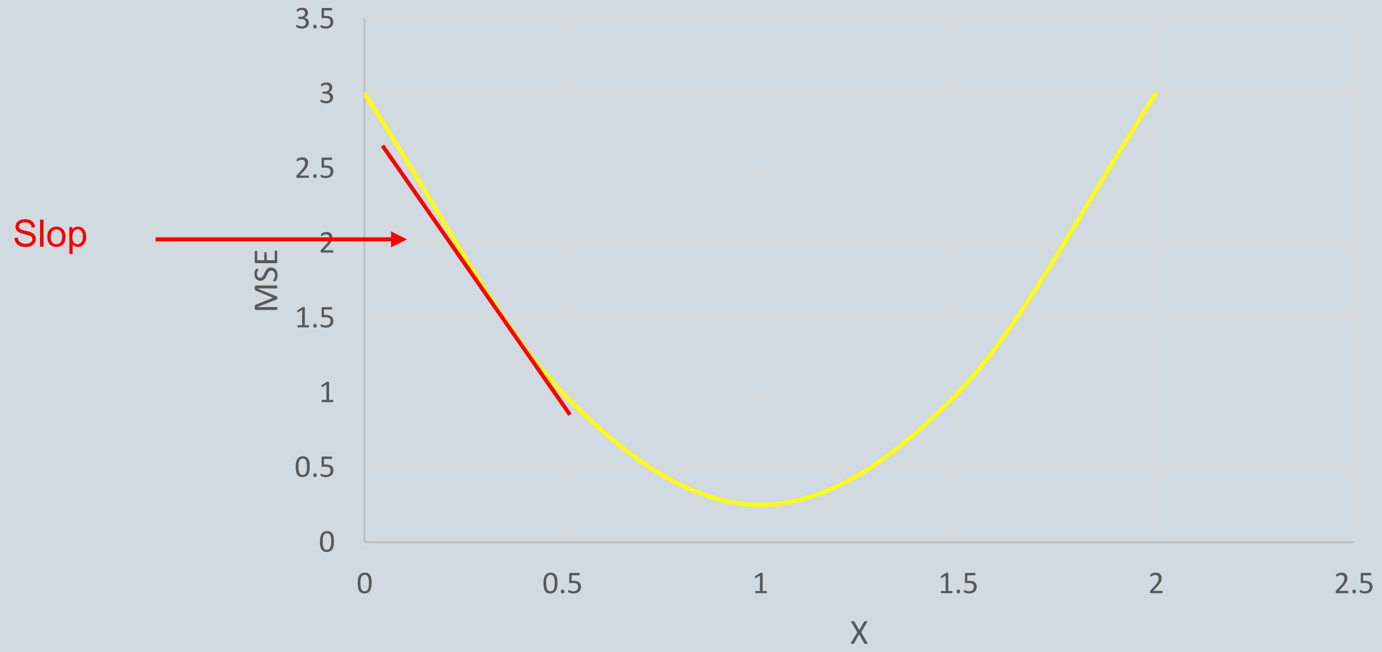
# Gradient Descent

2. Select initial solution $a_0$
    1. Set initial intersection and parameters



# Gradient Descent

3. Calculate the slope of the Loss function at current step (gradient) $\nabla\mathcal{L}(a_i)$
   - The lost function could be the MSE or any other lost function



# Gradient Descent

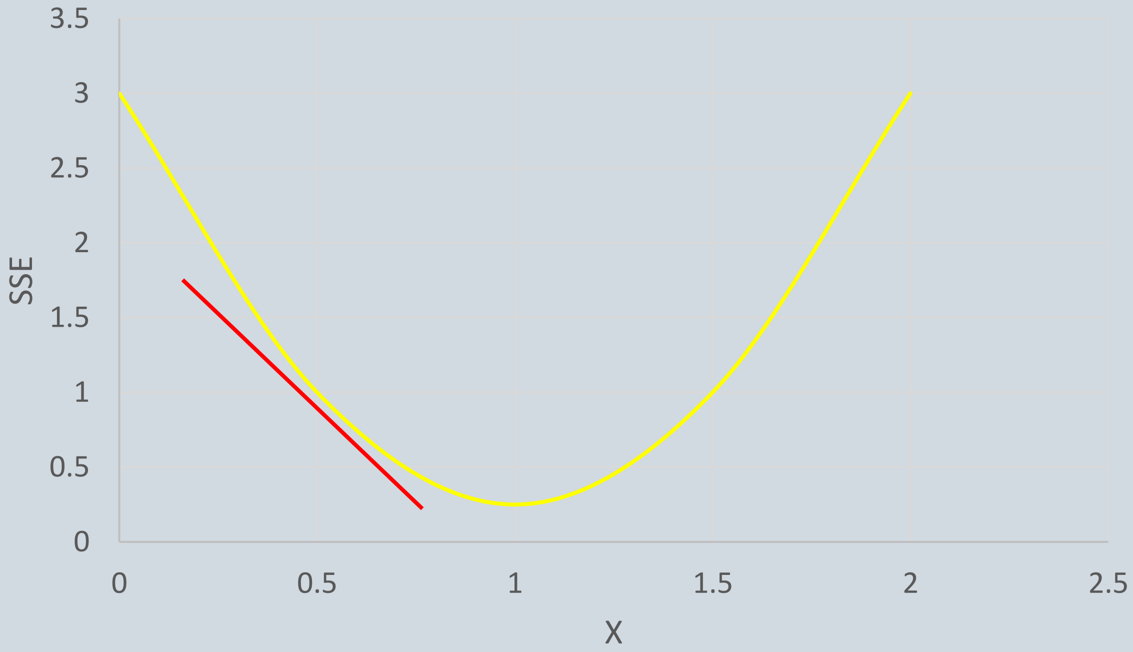
4. Calculate the next step $a_{i+1}$ as
   - $a_{i+1} = a_i - \gamma \nabla \mathcal{L}(a_i)$
   - Basically, we adjust the parameters by the learning rate * the slop of the derivative

# Gradient Descent

5. Termination?
   - Termination condition:
     - The adjusted learning rate is very small (normally close to 0)
     - Or the maximum number of iteration reached.
   - Terminate if reached the termination

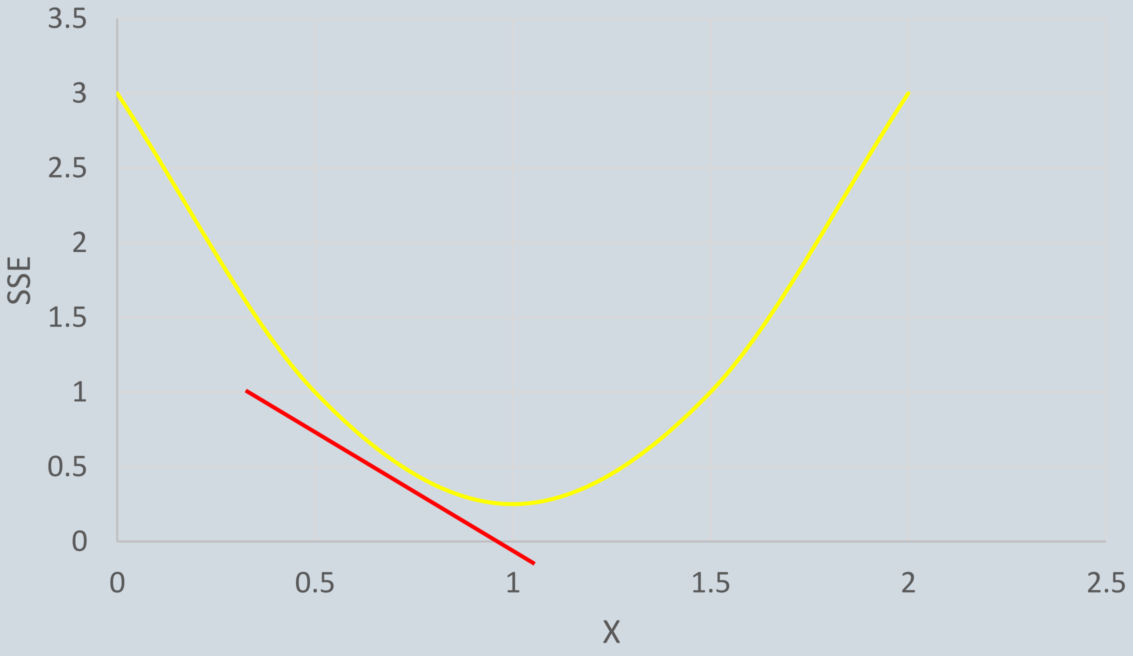
# Gradient Descent

5. Termination?
   - If not reached the termination condition go to step 3
   - Calculate the slope of the Loss function at current step (gradient) $\nabla\mathcal{L}(a_i)$



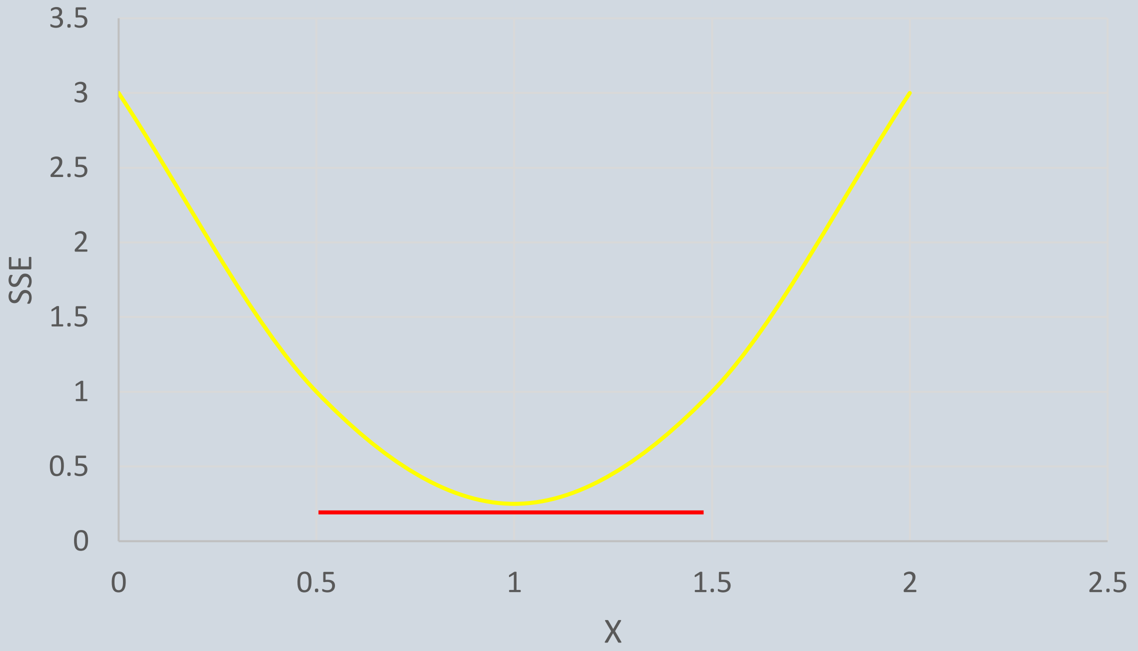
# Gradient Descent

5. Termination?
   - If not reached the termination condition go to step 3
   - Calculate the slope of the Loss function at current step (gradient) $\nabla\mathcal{L}(a_i)$

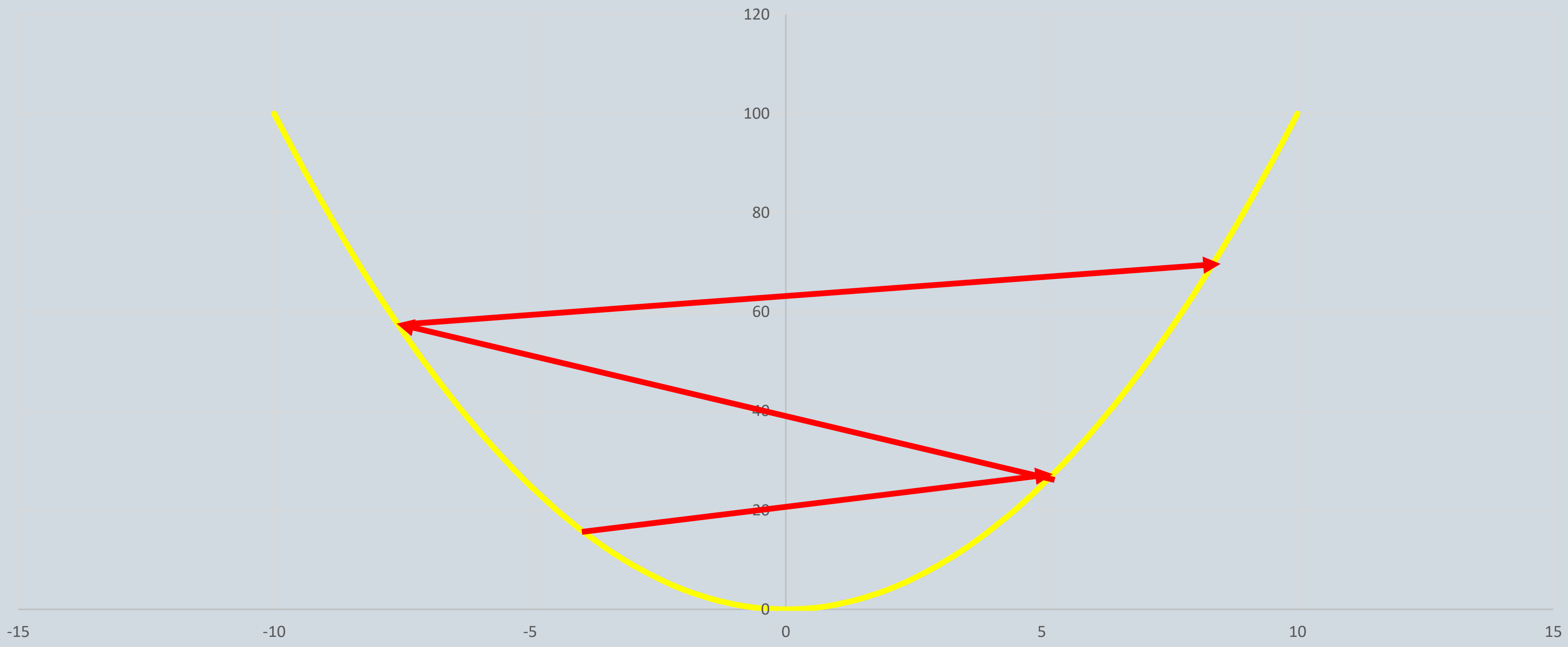


# Gradient Descent

5. Termination?
   - The adjusted learning rate is very small
   - Terminate



# Gradient Descent



# Gradient Descent – Linear Regression Model

Algorithm:

Start with initial values for $\beta_0,\beta_1$…

Repeat until convergence:

$$\beta_0^{i+1} \Leftarrow \beta_0^i - \alpha \frac{\partial \mathcal{L}_{\beta^i}}{\partial \beta_0}$$

$$\beta_1^{i+1} \Leftarrow \beta_1^i - \alpha \frac{\partial \mathcal{L}_{\beta^i}}{\partial \beta_1}$$

…

# Gradient Descent – Linear Regression Model

Algorithm:

$\mathcal{L} = MSE = \frac{SSE}{n}$

Initialize $\beta_0$,$\beta_1$

Repeat until convergence:

- $SSE' = -2X^{\mathrm{T}}Y + 2X^{\mathrm{T}}X\beta$
- $\beta_0^{i+1} = \beta_0^i - \gamma\frac{1}{n}\sum_{j=i}^{n}\left(\beta_0^{i+1} + \beta_1^{i+1}x_j - y_j\right)$
- $\beta_1^{i+1} = \beta_1^i - \gamma\frac{1}{n}\sum_{j=i}^{n}\left(\beta_0^{i+1} + \beta_1^{i+1}x_j - y_j\right)x_j$

# Main Linear Regression Assumptions

- Linearity: $E(\varepsilon_i)=0$
  - response variable is a linear combination of the predictors.
- Constant variance: Var(εi)=0
  - variance does not depends on the value of the predictors
- Independence of error: Cov(εi εj)=0
  - Errors are not correlated
- Normal Distribution of errors: ε~Norm(0,σ)

# Regularization

- Motivation for Regularization
  - Instead of eliminating variables – we reduce their effect by lowering their coefficient values
  - Smaller coefficient values results in “simpler hypotheses”
  - Which in turn reduces the chances of over-fitting (to-be discussed)
- **Ridge, Lasso Regression and Elastic Net**
  - Popular techniques in machine learning used for **regularizing linear models to avoid overfitting and improve predictive performance**.
  - Add a **penalty** term to the model’s cost function to constrain the coefficients, but they differ in how they apply this penalty.

# Ridge Regression

- **Ridge Regression**
  - A.K.A. L2 regularization
  - Add penalty equivalent to the square of coefficients
  - Minimize MSE + λ * sum of square of coefficients
  - λ = regularization strength
  - If λ=0 than we deal with normal linear regression

$$RidgeLoss = \sum_{i=1}^{m}(y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{m} \beta_j^2$$

- Example:
  - Suppose we want to predict a student's exam score using two features:
    - Study hours
    - Sleep hours
  - The model gives coefficients:
    - $\beta_1 = 8$ (effect of study hours)
    - $\beta_2 = -2$ (effect of sleep hours)
    - Regularization parameter: $\lambda = 0.05$
  - Step 1: Penalty Term
    - $\lambda(\beta_1^2 + \beta_2^2) = 0.05 \cdot (8^2 + (-2)\ ^2) = 3.4$
  - Step 2: Ridge Loss
    - The **prediction error term** $\sum(y_i - \hat{y}_i)^2$ depends on the dataset.
    - The **regularization penalty** we just calculated is **3.4**.
    - So the Ridge Loss = prediction error + **3.4**.

# Lasso Regression

- **Lasso Regression**
  - A.K.A. L1 regularization
  - Add penalty equivalent to the <u>absolute</u> square of coefficients
  - Minimize MSE + λ * sum of absolute values of coefficients
  - λ = regularization strength
  - If λ=0 than we deal with normal linear regression

$$LassoLoss = \sum_{i=1}^{m}(y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{m} |\beta_i|$$

- Example:
  - Suppose we want to predict a student's exam score using two features:
    - Study hours
    - Sleep hours
  - The model gives coefficients:
    - $\beta_1 = 8$ (effect of study hours)
    - $\beta_2 = -2$ (effect of sleep hours)
    - Regularization parameter: $\lambda = 0.05$
  - Step 1: Penalty Term
    - $\lambda(|\beta_1| + |\beta_2|) = 0.05 \cdot (8 + 2)$ =0.5
  - Step 2: Lasso Loss
    - The **prediction error term** $\sum(y_i - \hat{y}_i)^2$ depends on the dataset.
    - The **regularization penalty** we just calculated is **0.5**.
    - So the Ridge Loss = prediction error + **0.5**.

# Effect of Regularization on Coefficients

- Ridge never zeros coefficients while Lasso can
- Ridge:
  - Ridge penalty is proportional to the square of the coefficient.
  - The shrinkage is smooth and continuous — the red line (solution) just gets pulled closer to zero.
  - **Result:** coefficients are reduced in size but never exactly zero.
- Lasso:
  - Lasso penalty is proportional to the absolute value of the coefficient.
  - At small values, the penalty is "flat" (seen as the horizontal red line at the origin).
  - This means that if the benefit of including a predictor is smaller than the penalty λ, the coefficient jumps to exactly zero.
  - **Result:** Lasso can completely remove predictors.

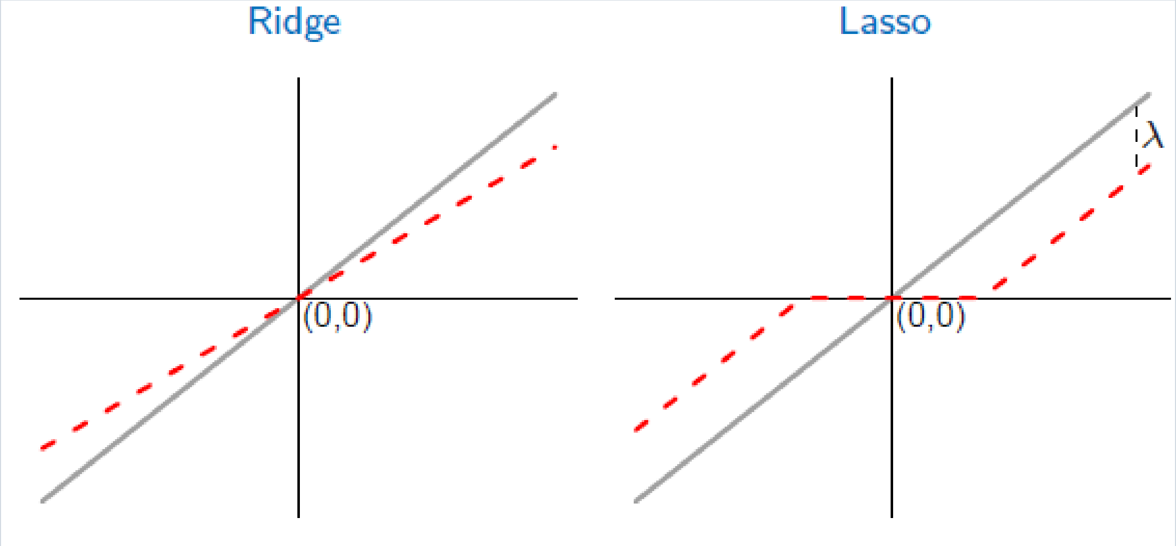



Figure from "An Introduction to Statistical Learning", James et al., 2013

# Elastic Net Regression

- **Elastic Net Regression**
  - A.K.A. L1+L2 regularization
  - Combines both L1 (Lasso) and L2 (Ridge) penalties.
  - Performs feature selection while also controlling the size of coefficients.
  - Helps handle multicollinearity by distributing weights across correlated features.
  - Unlike Lasso, which might arbitrarily select one feature and drop the rest, Elastic Net provides a more balanced solution when many predictors are correlated.

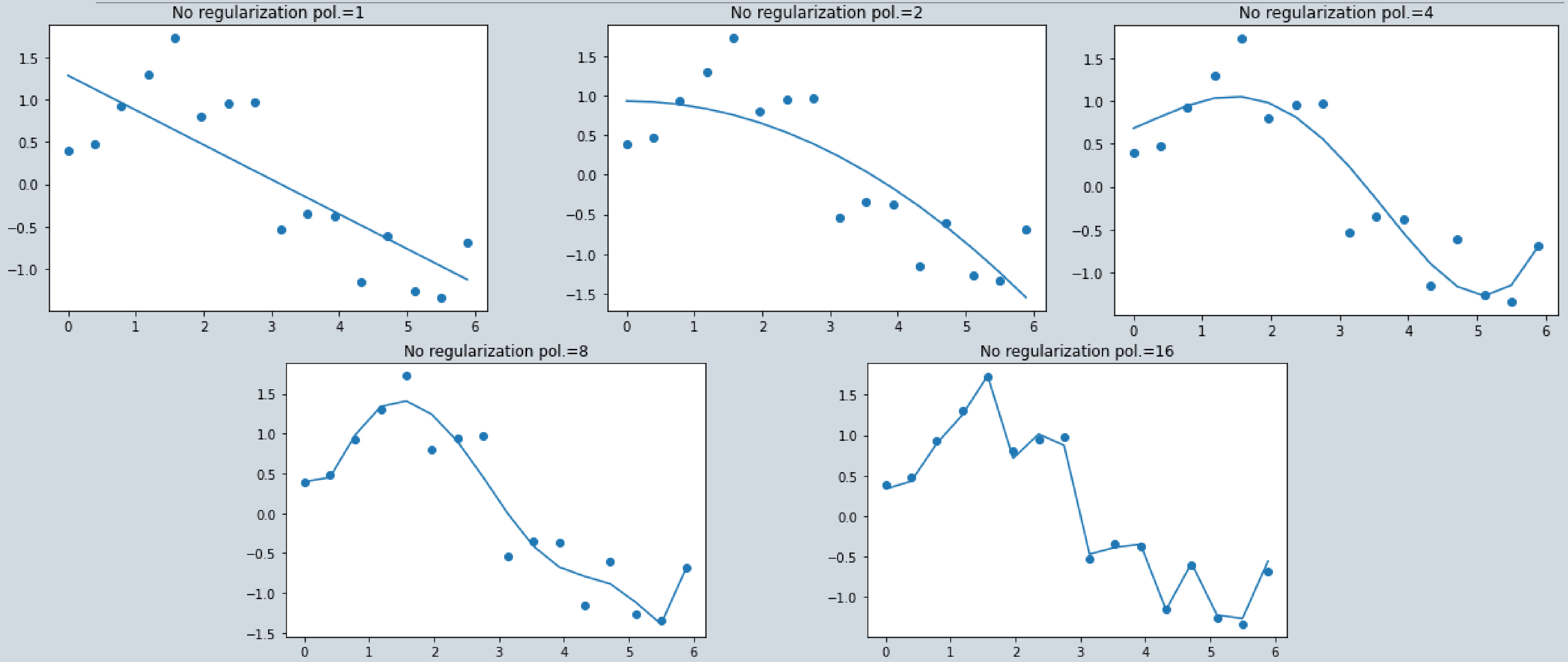
$$ElasticNetLoss = \sum_{i=1}^{m}(y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{m} |\beta_i| + \lambda \sum_{i=1}^{m} \beta_j^2$$

- Example:
  - Suppose we want to predict a student's exam score using two features:
    - Study hours
    - Sleep hours
  - The model gives coefficients:
    - $\beta_1 = 8$ (effect of study hours)
    - $\beta_2 = -2$ (effect of sleep hours)
    - Regularization parameter: $\lambda = 0.05$
    - Mix between L1 and L2:
      - 1_ratio=ρ=0.6
      - (ρ=1ρ=1 → pure Lasso; ρ=0ρ=0 → pure Ridge)
  - Penalty Term
    - $Loss = \sum(y_i - \hat{y}_i)^2 + \lambda\left[p \sum |\beta_i| + (1-p) \sum \beta_j^2\right]$
    - $Loss 0.05 \cdot [0.6 \cdot (|8| + |-1|) + 0.4 \cdot (8^2 + (-2)\ ^2)] = 1.66$
    - So the **Elastic Net loss = prediction error + 1.66**.

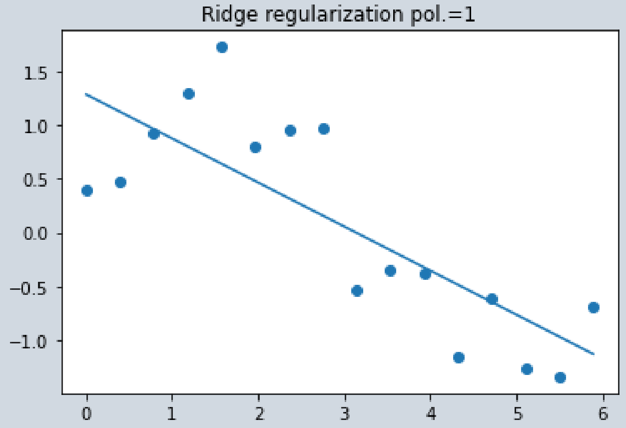
# When to Use What?

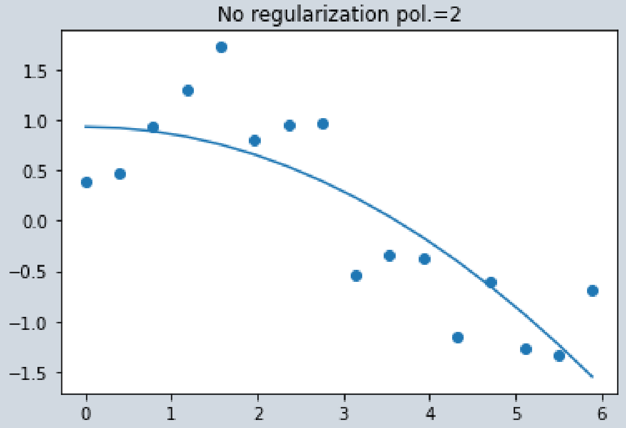
- **Ridge Regression (L2)**
  - Use when all predictors are potentially relevant.
  - Goal: reduce overfitting without removing features.
  - Works well when predictors are moderately correlated.
  - Example: House price prediction where size, location, rooms, and year built all matter.
- **Lasso Regression (L1)**
  - Use when only a subset of predictors is important.
  - Goal: perform automatic feature selection.
  - Best when you want a simpler model with fewer features.
  - Example: Genetic studies where only a few out of thousands of genes matter.
- **Elastic Net Regression (L1 + L2)**
  - Use when predictors are many and highly correlated.
  - Goal: balance shrinkage (Ridge) and feature selection (Lasso).
  - Avoids instability of Lasso dropping correlated predictors.
  - Example: Text or genomics data where features overlap heavily.

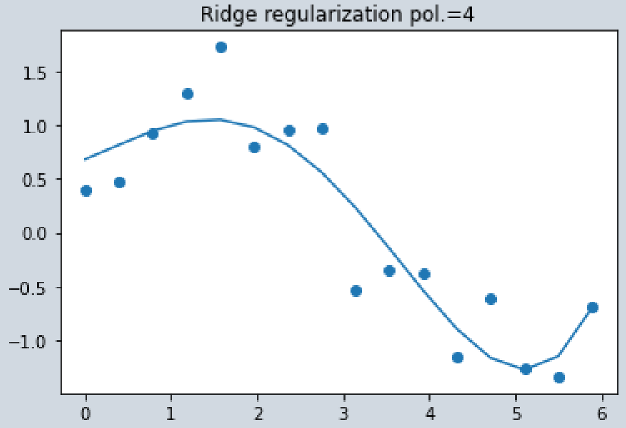
# Polynomial Regression

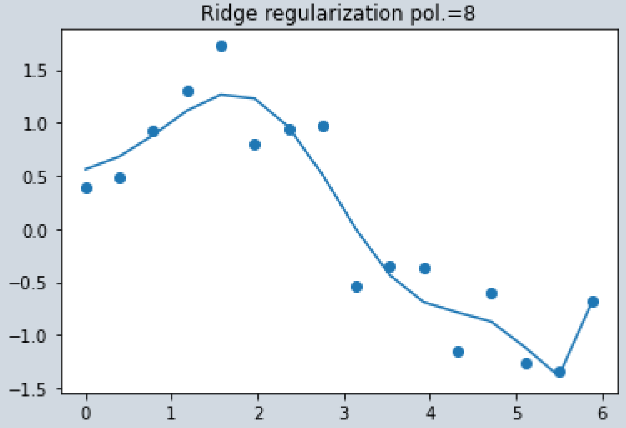


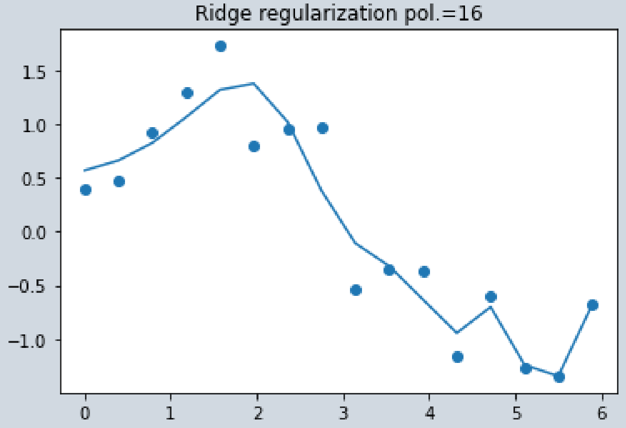
# Polynomial Regression – Ridge $\alpha = 1.5$



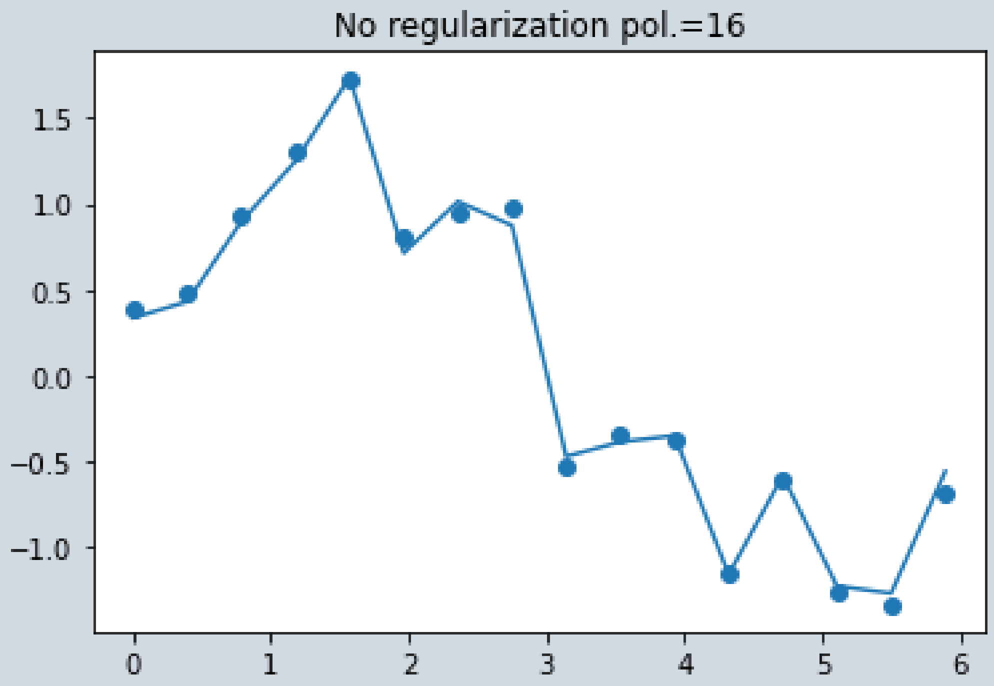



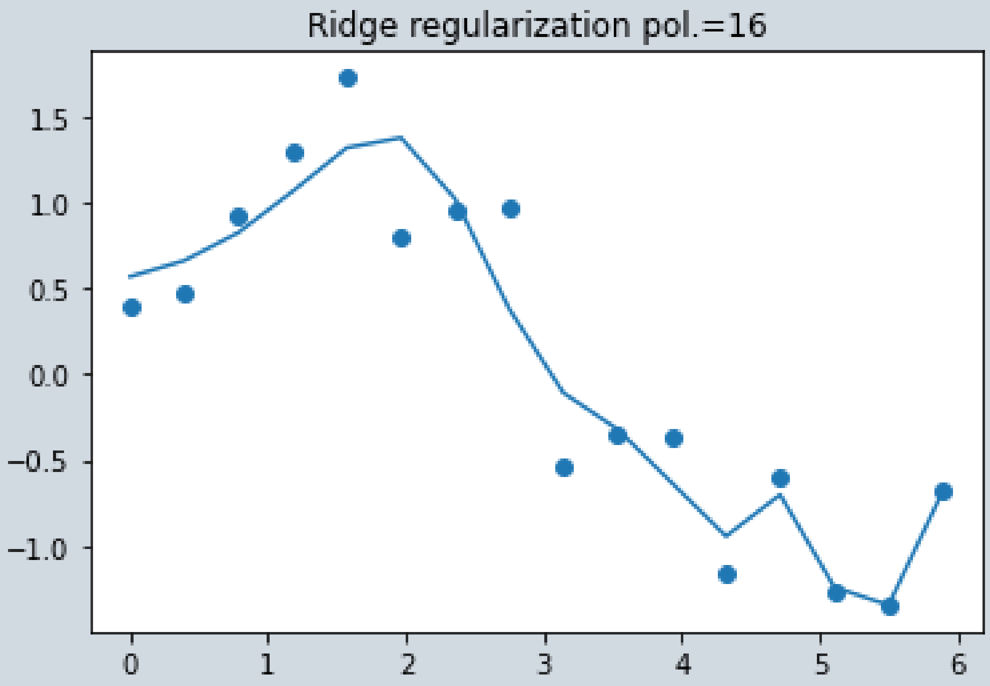



# Polynomial Regression – Before and After



# Lasso Regression in Python

```
>>> from sklearn import linear_model
>>> clf = linear_model.Lasso(alpha=0.1)
>>> clf.fit([[0,0], [1, 1], [2, 2]], [0, 1, 2])
Lasso(alpha=0.1)
>>> print(clf.coef_)
[0.85 0.  ]
>>> print(clf.intercept_)
0.15...
```

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html#sklearn.linear_model.Lasso

**Regression_models.ipynb**

# Ridge Regression in Python

```
>>> from sklearn.linear_model import Ridge
>>> import numpy as np
>>> n_samples, n_features = 10, 5
>>> rng = np.random.RandomState(0)
>>> y = rng.randn(n_samples)
>>> X = rng.randn(n_samples, n_features)
>>> clf = Ridge(alpha=1.0)
>>> clf.fit(X, y)
Ridge()
```

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge

**Regression_models.ipynb**

# Elastic Net Regression in Python

```
>>> from sklearn.linear_model import ElasticNet
>>> from sklearn.datasets import make_regression
```

```
>>> X, y = make_regression(n_features=2, random_state=0)
>>> regr = ElasticNet(random_state=0)
>>> regr.fit(X, y)
ElasticNet(random_state=0)
>>> print(regr.coef_)
[18.83816048 64.55968825]
>>> print(regr.intercept_)
1.451
>>> print(regr.predict([[0, 0]]))
[1.451]
```

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html

**Regression_models.ipynb**

# Discrete Models: Logistic Regression

- Linear regression is useful for problems in which outcomes change gradually with input

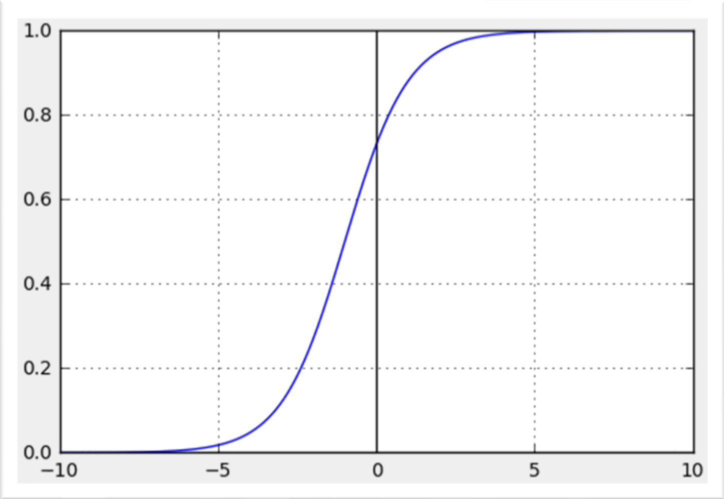
However, there are many cases in which the dependent variable is discrete (binary):

  - Voting (for/against), survival/mortality, adoption of a product, gender

- Rather than linear dependence on the input, we need a function that would switch quickly from one output (0) to another (+1)
- Essentially, we want to predict probability of an outcome: $P(0 \mid X)$ and $P(1 \mid X)$ – probability of an outcome given some set of independent variables.

# Logit

- The logistic distribution constrains the estimated probabilities to lie between 0 and 1.




$$p(x) = \frac{1}{1+exp(-\alpha-\beta \cdot x)}$$

Note:

- $\alpha + \beta \cdot x = 0 \quad \Longrightarrow p(x) = 0.5$
- $\alpha + \beta \cdot x \longrightarrow \infty \Longrightarrow p(x) = 1$
- $\alpha + \beta \cdot x = -\infty \Longrightarrow p(x) = 0$

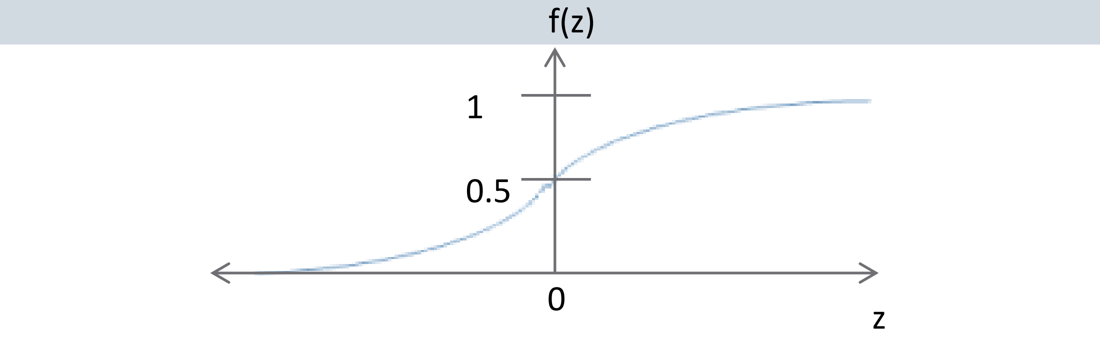
```
import numpy
import matplotlib.pyplot as plt

alpha = 1
beta = 1
x = numpy.linspace(-10,10,10000)
p_x = 1/(1+numpy.exp(-alpha-beta*x))

fig, ax = plt.subplots()
plt.plot(x,p_x,'-')
plt.grid()
```

# Logistic Regression

- Binary Dependent Variable y=0 “no” ; y=1 “yes”
- We want to estimate $h_\beta(x)$, where $h_\beta(x)$ is our hypothesis (guess) for the probability that y=1 given a feature vector x and parameters $\beta=\beta_0, \beta_1, \beta_2, \ldots$
  - $P(y = 1|x, \beta)$
- We can predict “y=1” if $h_\beta(x)>0.5$
- We define $h_\beta(x) = f(\beta_0+\beta_1 X_1+ \beta_2 X_2 \ldots)$
- Where f is the sigmoid (logistic) function: $f(z) = \frac{1}{1+e^{-z}}$
- Inserting $z=\beta_0+\beta_1 X_1+ \beta_2 X_2 \ldots$
- We can predict “y=1” if $\frac{1}{1+e^{-(\beta_0+\beta_1 X_1+ \beta_2 X_2 \ldots)}} > 0.5$

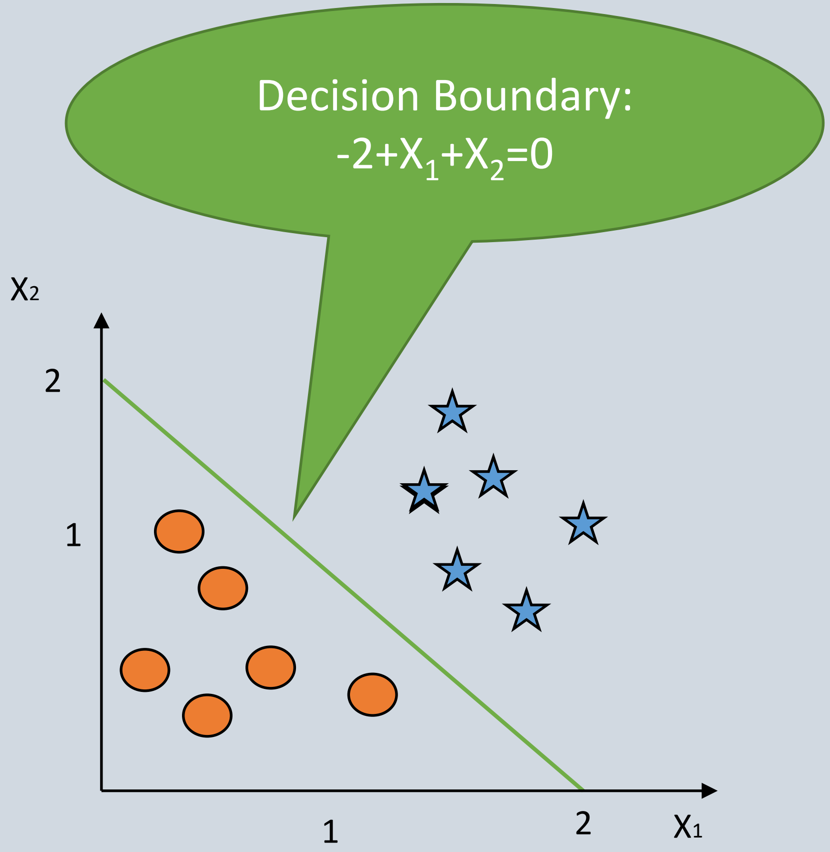



$z= \beta_0+\beta_1 X_1+ \beta_2 X_2 \ldots$

# Decision Boundary

- Classify "y=1" if $Pr(y\ is\ 1|X) > 0.5$
- Or $\frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\ \beta_2 X_2\ldots)}} > 0.5$
- Or $(\beta_0+\beta_1 X_1+\ \beta_2 X_2\ldots) > 0$

- Example:
  - Predict "y=1" if $-2+X_1+X_2>0$
  - parameter values: $\beta_0 = -2$, $\beta_1=1$, $\beta_2=1$




**Regression_models.ipynb**

# Logistic Regression Using Scikit-learn

Regression_models.ipynb

## Logit using sklearn

```
from sklearn.linear_model import LogisticRegression
X=data[['age','yrs_married','children']].values
y=data['had_affairs'].values
log_reg = LogisticRegression().fit(X,y)
```

executed in 79ms, finished 22:52:16 2023-03-26

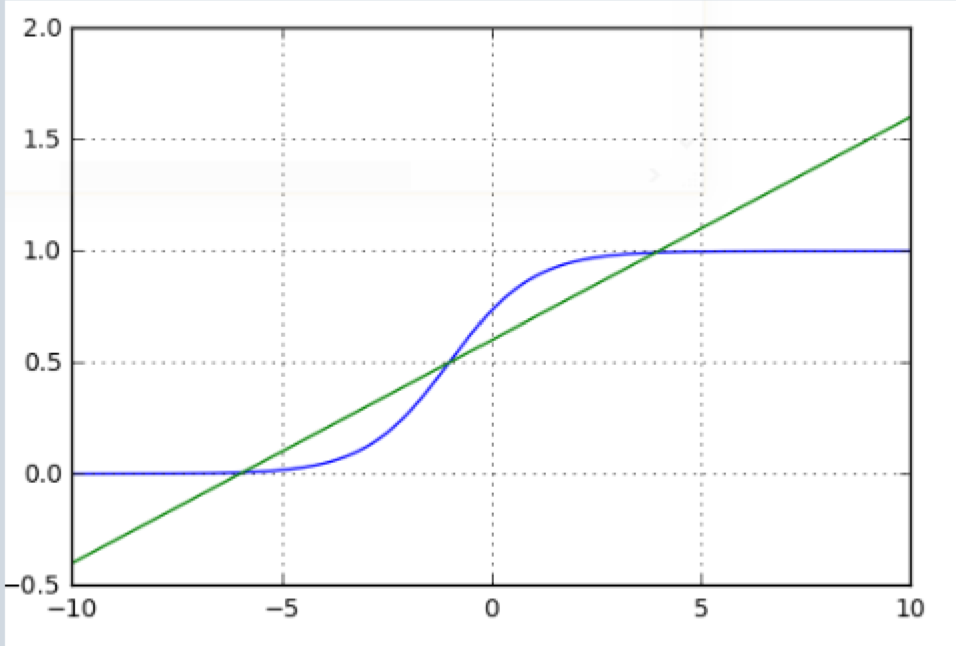
```
log_reg.predict(np.array([28,4,2]).reshape(1, -1))
```

executed in 15ms, finished 22:52:16 2023-03-26

# Logit vs Linear Model

```
%matplotlib inline
import numpy
import matplotlib.pyplot as plt

alpha = 1
beta = 1
x = numpy.linspace(-10,10,10000)
p_x = 1/(1+numpy.exp(-alpha-beta*x))
p_x1 = 0.6 + 0.1*x
fig, ax = plt.subplots()
plt.plot(x,p_x,'-')
plt.hold(True)
plt.plot(x,p_x1,'g-')
plt.grid()
```




See: http://nbviewer.jupyter.org/gist/justmarkham/6d5c061ca5aee67c4316471f8c2ae976

# THANK YOU FOR LISTENING

ZVI.BENAMI@MAIL.HUJI.AC.IL