CS 534
Artificial Intelligence

Week 5: Supervised Machine Learning Model I

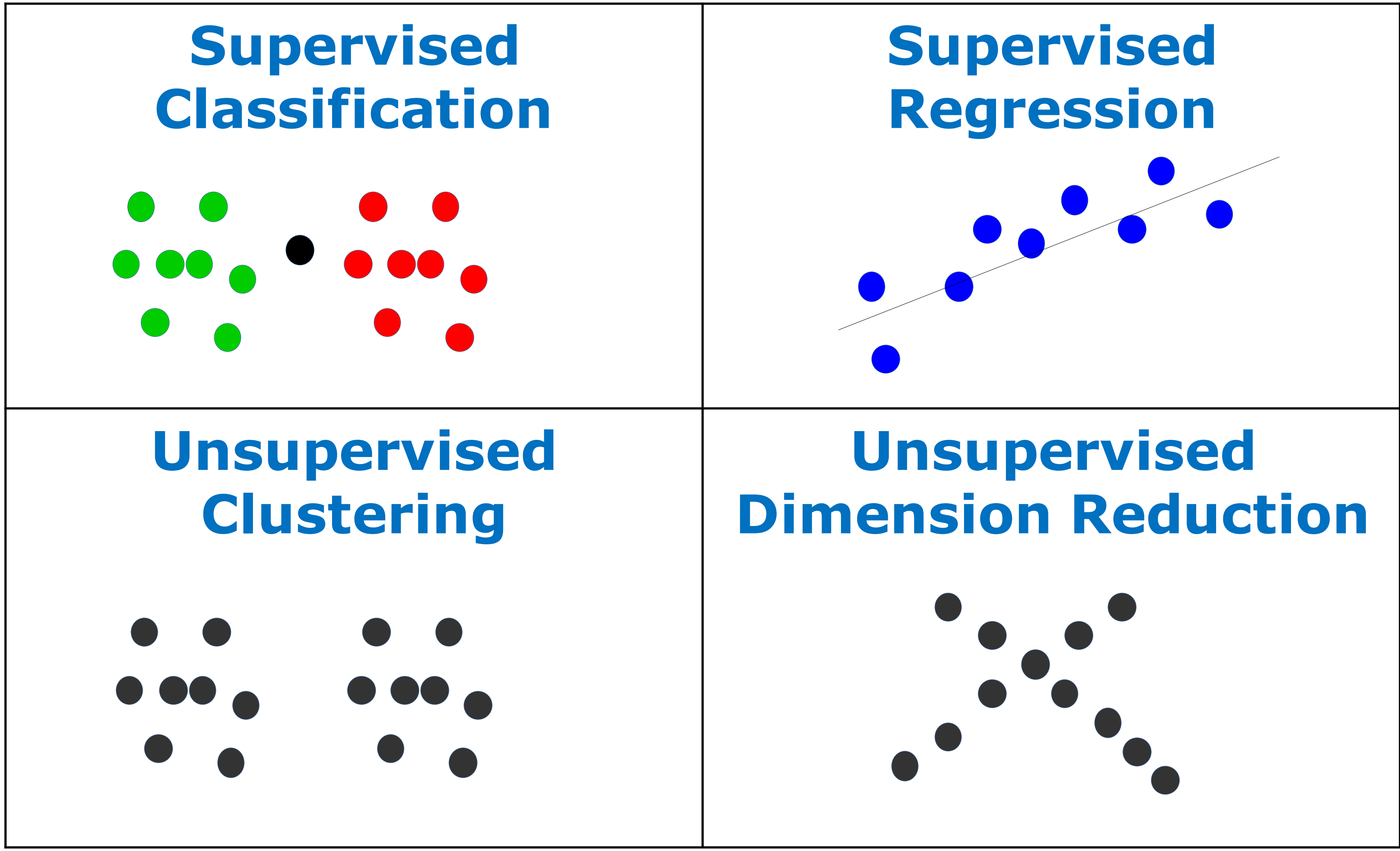
By

Ben C.K. Ngan

Machine Learning

- ***Machine Learning*** is the study of computer algorithms that can learn from the data and use that knowledge to make predictions on data they have not seen before.

- This learning is achieved via **a parameterized model** with tunable parameters that are automatically adjusted according to **different performance criteria**.

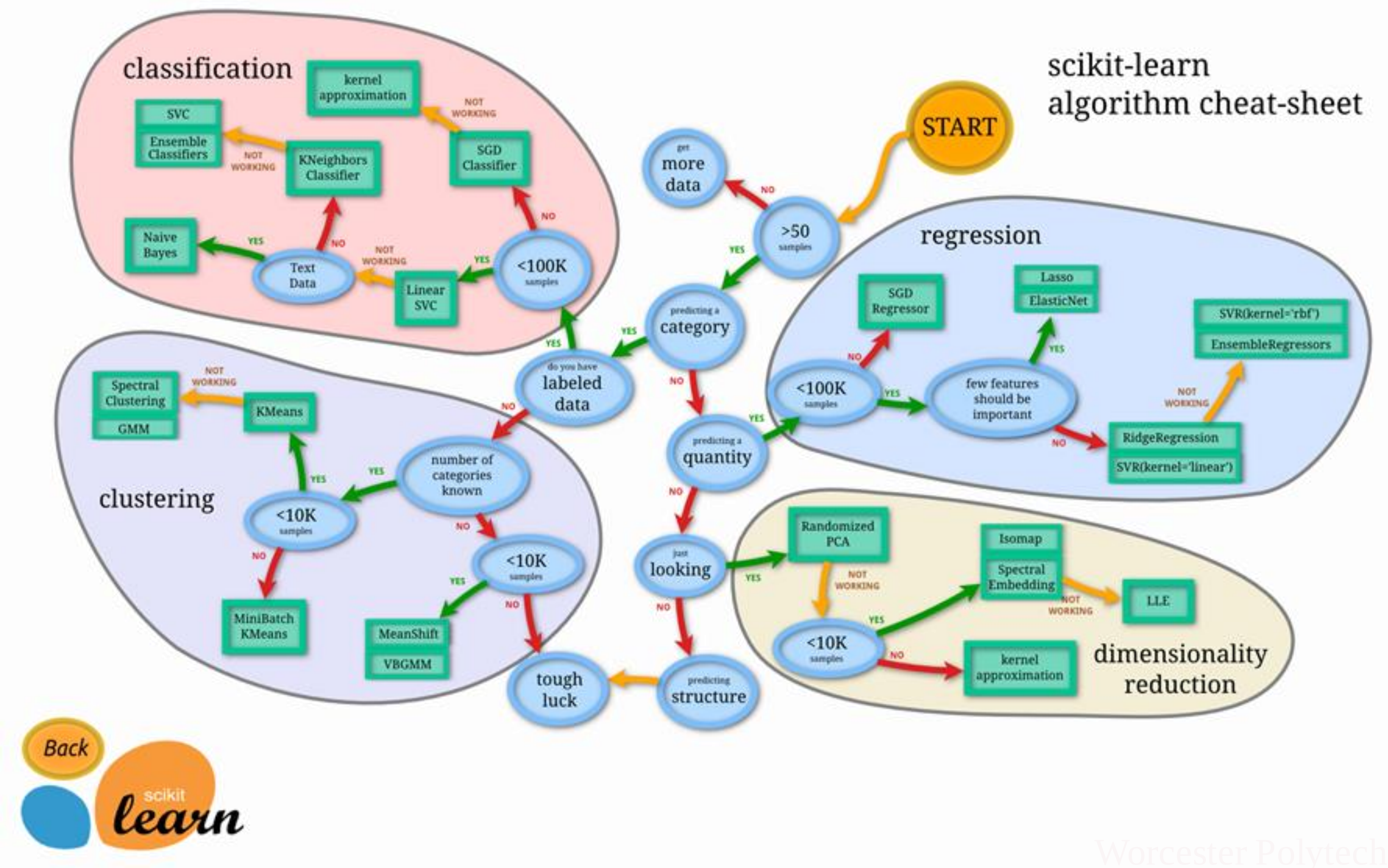


Supervised and Unsupervised Learning

- Supervised: Algorithms which learn from a training set of **labeled** examples to generalize to the set of **all possible inputs**.
  - **Classification:** decision tree, random forest, support vector classifier, etc.
  - **Regression:** linear regression, regression tree, support vector regressor, etc.
  - **Time Series:** ARIMA, SARIMA, exponential smoothing, etc.

- Unsupervised: Algorithms that learn from a training set of **unlabeled** examples. Used to explore data according to some statistical, geometric or similarity criterion.
  - **Clustering:** k-means clustering, hierarchical clustering, expectation maximization, etc.
  - **Dimension Reduction:** principal component analysis, linear discriminant analysis, kernel density estimation, etc.
  - **Association Analysis:** Apriori Algorithm, FP-Growth, ZeroR, etc.

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



Supervised Learning - Regression

- If our question is a prediction of **a real-valued quantity**, we are faced with a regression problem.

  – Given the description of an apartment, what is the expected market value of this apartment? What will the value be if the apartment has an elevator?

  – Given the past records of user activity on our Apps, how long will a certain client be connected to our App?

  – Given my skills and marks in computer science and math, what mark will I achieve in a data science course?

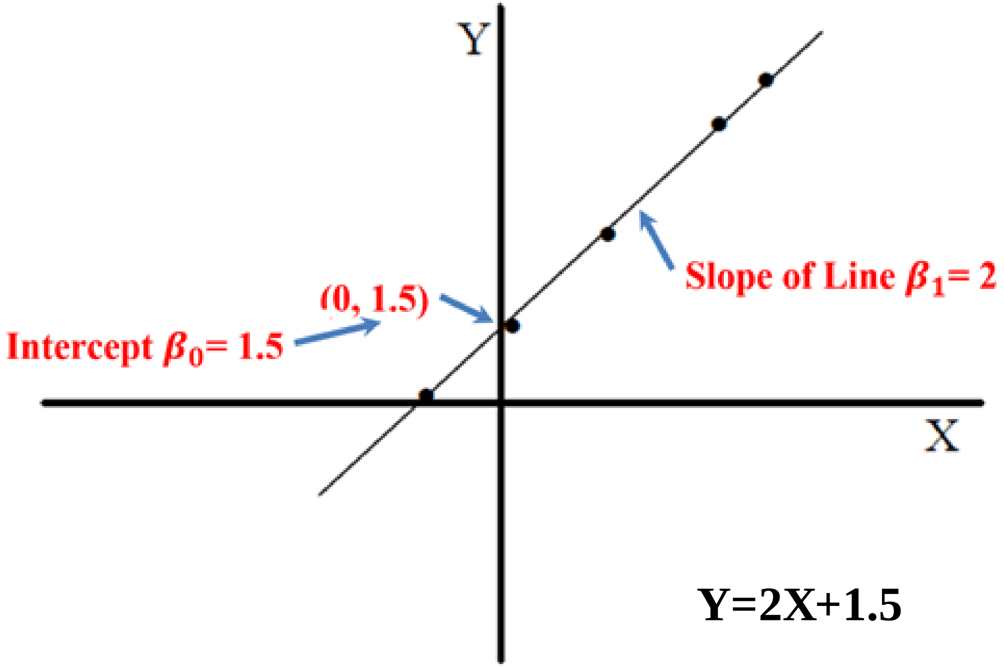
- Some of the most popular regression models are linear regression, regression tree, support vector regressor, etc.

Linear Regression Model

- Goal of Linear Regression Analysis:
  - Estimate the linear relationship between one or more explanatory variables and a single continuous variable.
  - Predict the value of this single continuous variable based upon the values of those input explanatory variables.
- Simple Linear Regression (SLR) Model:
  - $Y = \beta_1 X + \beta_0$, where β_1 is the slope (regression coefficient) and β_0 is the intercept (constant coefficient).
  - **Both X and Y must be continuous variables**

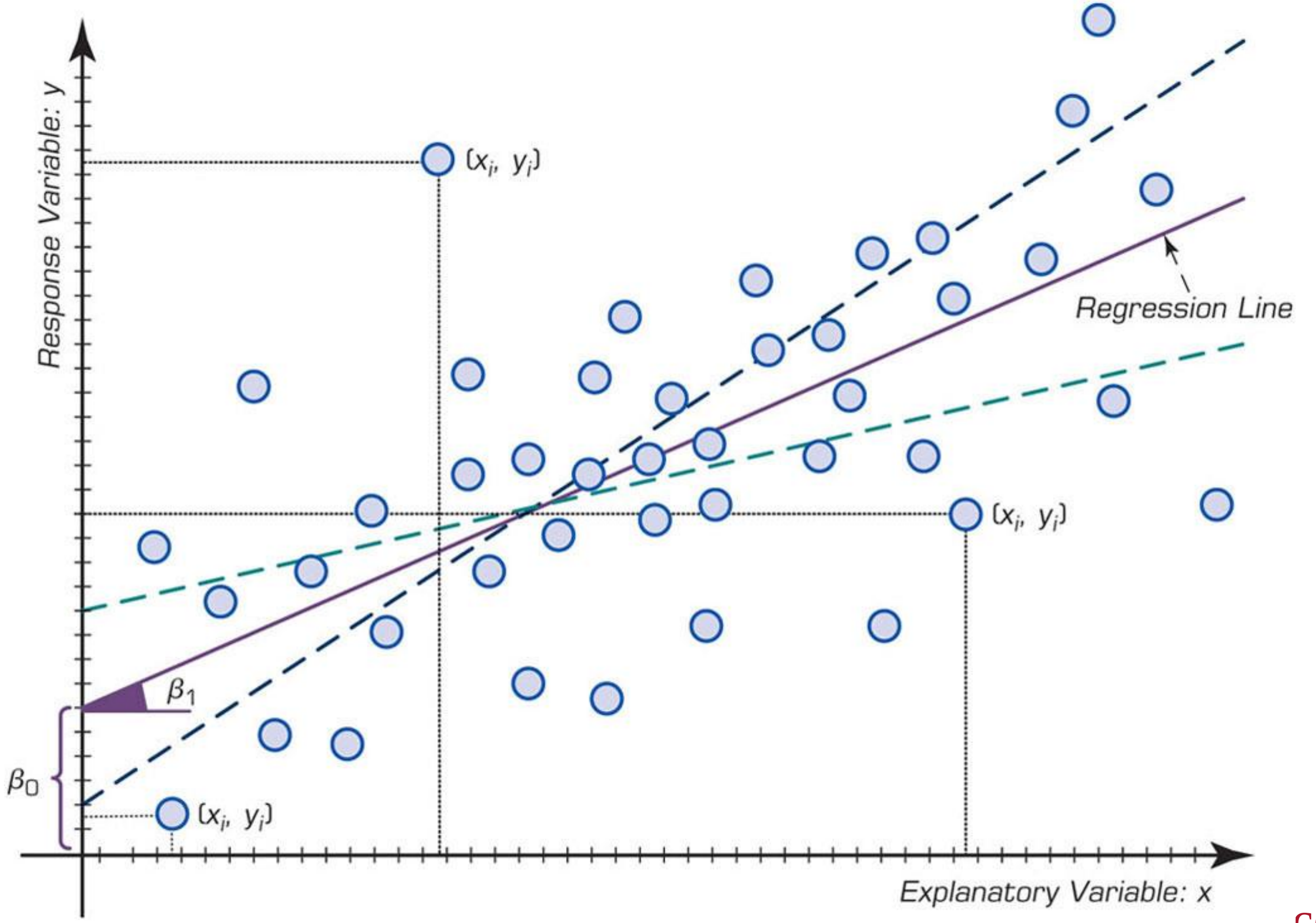
Simple Linear Regression Model

- $Y = \beta_1 X + \beta_0$, where β_1 is the slope (regression coefficient) and β_0 is the intercept (constant coefficient).



Simple Linear Regression Model

- $Y = \beta_1 X + \beta_0$, where β_1 is the slope (regression coefficient) and β_0 is the intercept (constant coefficient).



Simple Linear Regression Model

- Y = β_1X + β_0, where β_1 is the slope (regression coefficient) and β_0 is the intercept (constant coefficient).

- Ordinary Least Squares (OLS): Obtain the minimum value for the following equation: **Sum of Squared Errors (SSE) or Cost/Error Function (C)**

$$= \frac{1}{N}\sum_{i=1}^{N}\overbrace{\left(Y_i - \underbrace{(\beta_1 X_i + \beta_0)}_{\hat{Y}_i \text{ is a predicted value}}\right)}^{\textit{Residual Error}}{}^2$$, where N is the size of the learning data set

Y_i **is an actual value**

  - OLS minimizes the SSE
    - If SSE = 0, the straight line fits the data points perfectly → **Correlation Coefficient *r* = +1 or -1**.
    - If SSE > 0, the straight regression line does not go through each data point → **Correlation Coefficient -1 < r < +1**.

Covariance and Correlation Coefficient

- When two variables share the **same tendency**, we speak about covariance.

- Let us consider two series, $\{x_i\}$ and $\{y_i\}$. Let us center the data with respect to their mean, where n is the length of both sets:
  - $d_{xi} = x_i - \mu_X$ and $d_{yi} = y_i - \mu_Y$.
  - It is easy to show that when $\{x_i\}$ and $\{y_i\}$ vary together, their deviations tend to have the same sign.
  - The covariance is defined as the mean of the following products:

$$Cov(X, Y) = \frac{1}{n} \sum_{i=1}^{n} dx_i dy_i \,,$$

Covariance and Correlation Coefficient

- If we normalize the data with respect to their deviation, that leads to the standard scores; and then multiplying them, we get:

$$\rho_i = \frac{x_i - \mu_X}{\sigma_X}\frac{y_i - \mu_Y}{\sigma_Y}.$$

- The mean of this product is $\rho = \frac{1}{n}\sum_{i=1}^{n} \rho_i$

- Equivalently, we can rewrite ρ in terms of the covariance, and thus obtain the correlation coefficient:

$$\rho = \frac{Cov(X, Y)}{\sigma_X \sigma_Y}.$$

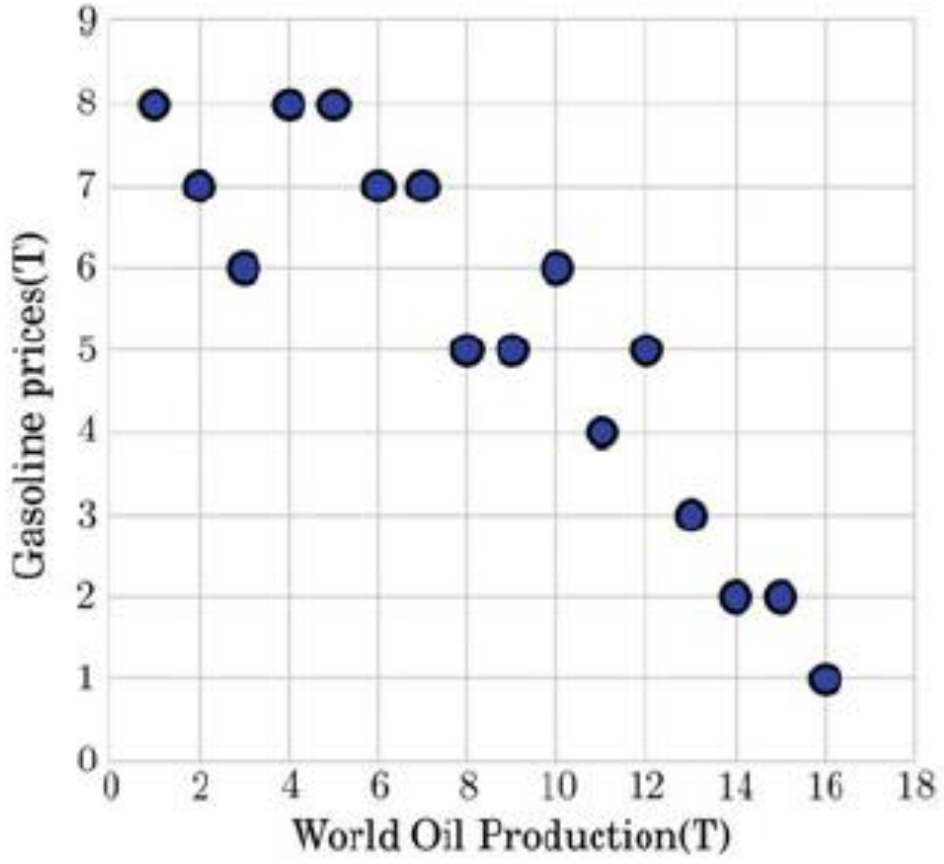
- The correlation coefficient is always **between −1 and +1**, where the magnitude depends on the degree of correlation.
  - If the correlation coefficient is **1 (or −1)**, it means that the variables are **perfectly correlated** (positively or negatively). This means that one variable can predict the other very well.
  - Having **ρ = 0**, does not necessarily mean that the variables are not correlated! The correlation coefficient captures correlations of first order, but not **nonlinear correlations**.

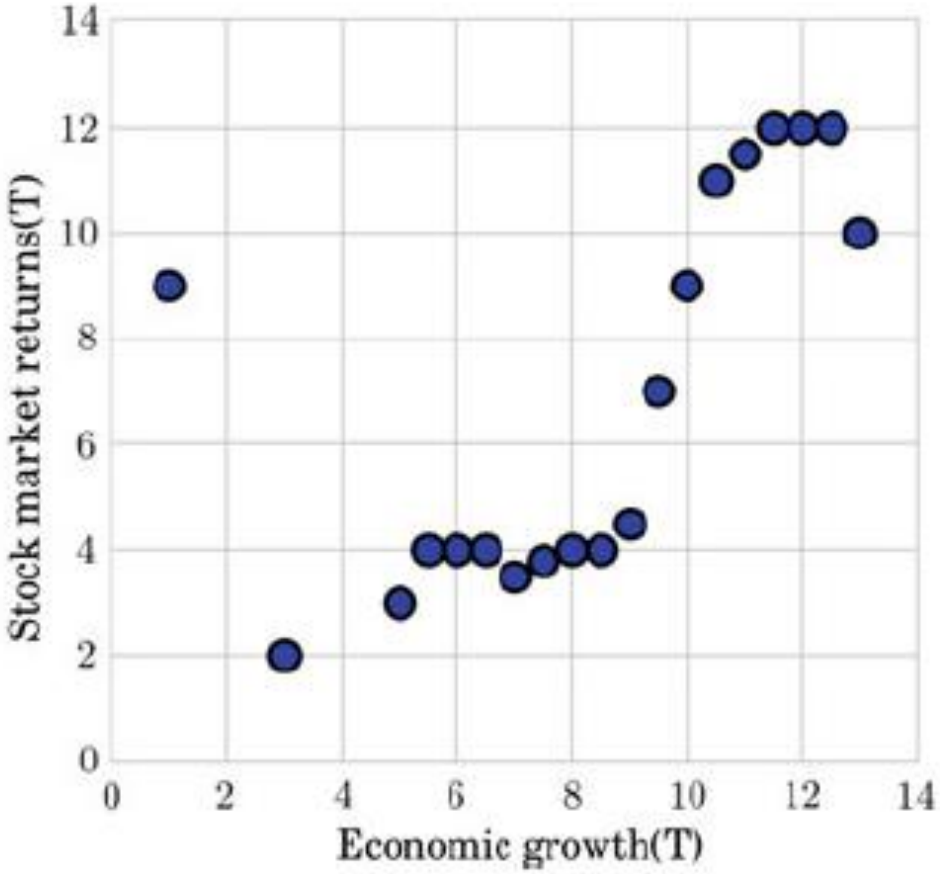
Covariance and Correlation Coefficient

- The correlation coefficient is always **between −1 and +1**, where the magnitude depends on the degree of correlation.
  – If the correlation coefficient is **1 (or −1)**, it means that the variables are **perfectly correlated** (positively or negatively). This means that one variable can predict the other very well.
  – Having **ρ = 0**, does not necessarily mean that the variables are not correlated! The correlation coefficient captures correlations of first order, but not **nonlinear correlations**.

- The linear strength of the correlation suggests for the **absolute** value of **ρ**:
  – 0.00-0.19 - "Very Weak"
  – 0.20-0.39 - "Weak"
  – 0.40-0.59 - "Moderate"
  – 0.60-0.79 - "Strong"
  – 0.80-1.00 - "**Very Strong**"

Covariance and Correlation Coefficient

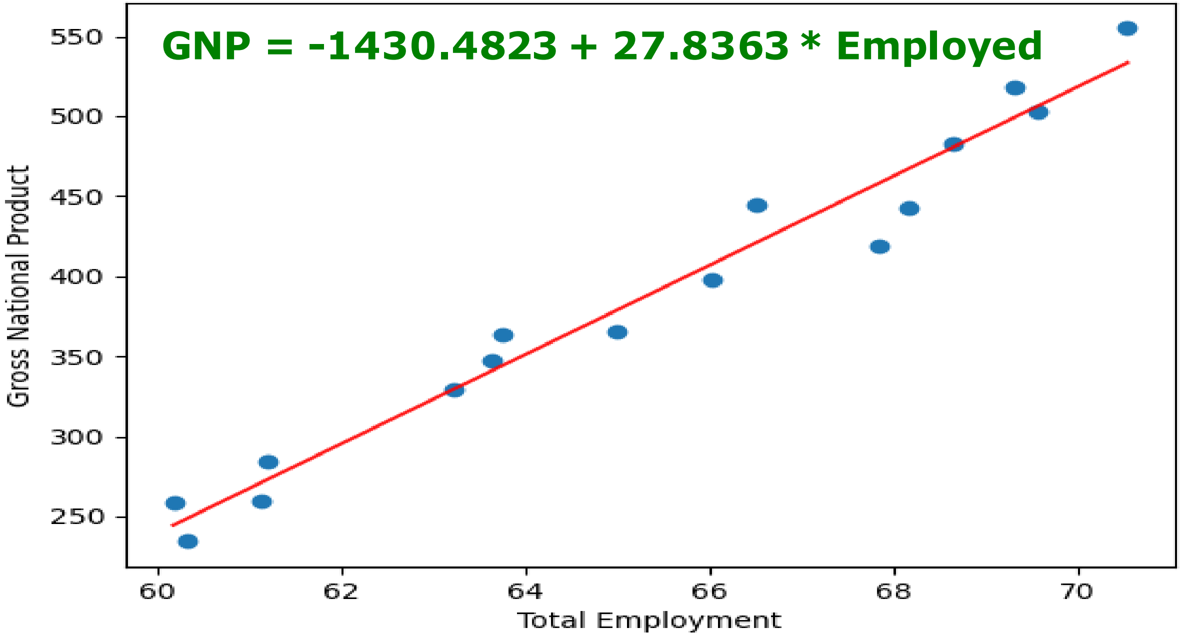




Hands-on Example: Simple Linear Regression

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm

def main():
    df = pd.read_csv('longley.csv', index_col=0)
    print("Correlation coefficient = ", np.corrcoef(df.Employed,df.GNP)[0,1])

    X = df.Employed  # predictor (independent variable)
    y = df.GNP  # response (dependent variable)
    X = sm.add_constant(X)  # Adds a constant term to the predictor
    lr_model = sm.OLS(y, X).fit()
    print(lr_model.summary())

    # We pick 100 points equally spaced from the min to the max
    X_prime = np.linspace(X.Employed.min(), X.Employed.max(), 100)
    X_prime = sm.add_constant(X_prime)  # Add a constant as we did before

    # Now we calculate the predicted values
    y_hat = lr_model.predict(X_prime)

    plt.scatter(X.Employed, y)  # Plot the raw data
    plt.xlabel("Total Employment")
    plt.ylabel("Gross National Product")
    plt.plot(X_prime[:, 1], y_hat, 'red', alpha=0.9)  # Add the regression line, colored in red
    plt.show()

if __name__ == "__main__":
    main()
```

	GNP.deflator	GNP	Unemployed	Armed.Forces	Population	Year	Employed
1947	83	234.289	235.6	159	107.608	1947	60.323
1948	88.5	259.426	232.5	145.6	108.632	1948	61.122
1949	88.2	258.054	368.2	161.6	109.773	1949	60.171
1950	89.5	284.599	335.1	165	110.929	1950	61.187
1951	96.2	328.975	209.9	309.9	112.075	1951	63.221
1952	98.1	346.999	193.2	359.4	113.27	1952	63.639
1953	99	365.385	187	354.7	115.094	1953	64.989
1954	100	363.112	357.8	335	116.219	1954	63.761
1955	101.2	397.469	290.4	304.8	117.388	1955	66.019
1956	104.6	419.18	282.2	285.7	118.734	1956	67.857
1957	108.4	442.769	293.6	279.8	120.445	1957	68.169
1958	110.8	444.546	468.1	263.7	121.95	1958	66.513
1959	112.6	482.704	381.3	255.2	123.366	1959	68.655
1960	114.2	502.601	393.1	251.4	125.368	1960	69.564
1961	115.7	518.173	480.6	257.2	127.852	1961	69.331
1962	116.9	554.894	400.7	282.7	130.081	1962	70.551



Multiple Linear Regression Model

- Multiple Linear Regression (MLR) Model: $Y = \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \beta_0$
  - Y must be a continuous variable.
  - $X_1, X_2, \ldots, X_n$ must be continuous variables
  - **Rule of Thumb**: At least two or more X_i variables.

- The linear strength of the correlation suggests for the **absolute** value of r:
  - 0.00-0.19 - "Very Weak"
  - 0.20-0.39 - "Weak"
  - 0.40-0.59 - "Moderate"
  - 0.60-0.79 - "Strong"
  - 0.80-1.00 - "**Very Strong**"

Multiple Linear Regression Model

- Multiple Linear Regression Model that describes how a target variable Y relates to two or more X variables.
  - Multiple: $Y = \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \beta_0$
  - Ordinary Least Squares (OLS): Obtain the minimum value for the following equation: **Sum of Squared Errors (SSE) or Cost/Error Function**

$$= \frac{1}{N}\sum_{i=1}^{N} \overbrace{\left(Y_i - \underbrace{(\beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_n X_{ni} + \beta_0)}_{\hat{Y}_i \text{ is a predicted value}} \right)^2}^{\textit{Residual Error}}$$

$\hat{Y}_i$ is a predicted value

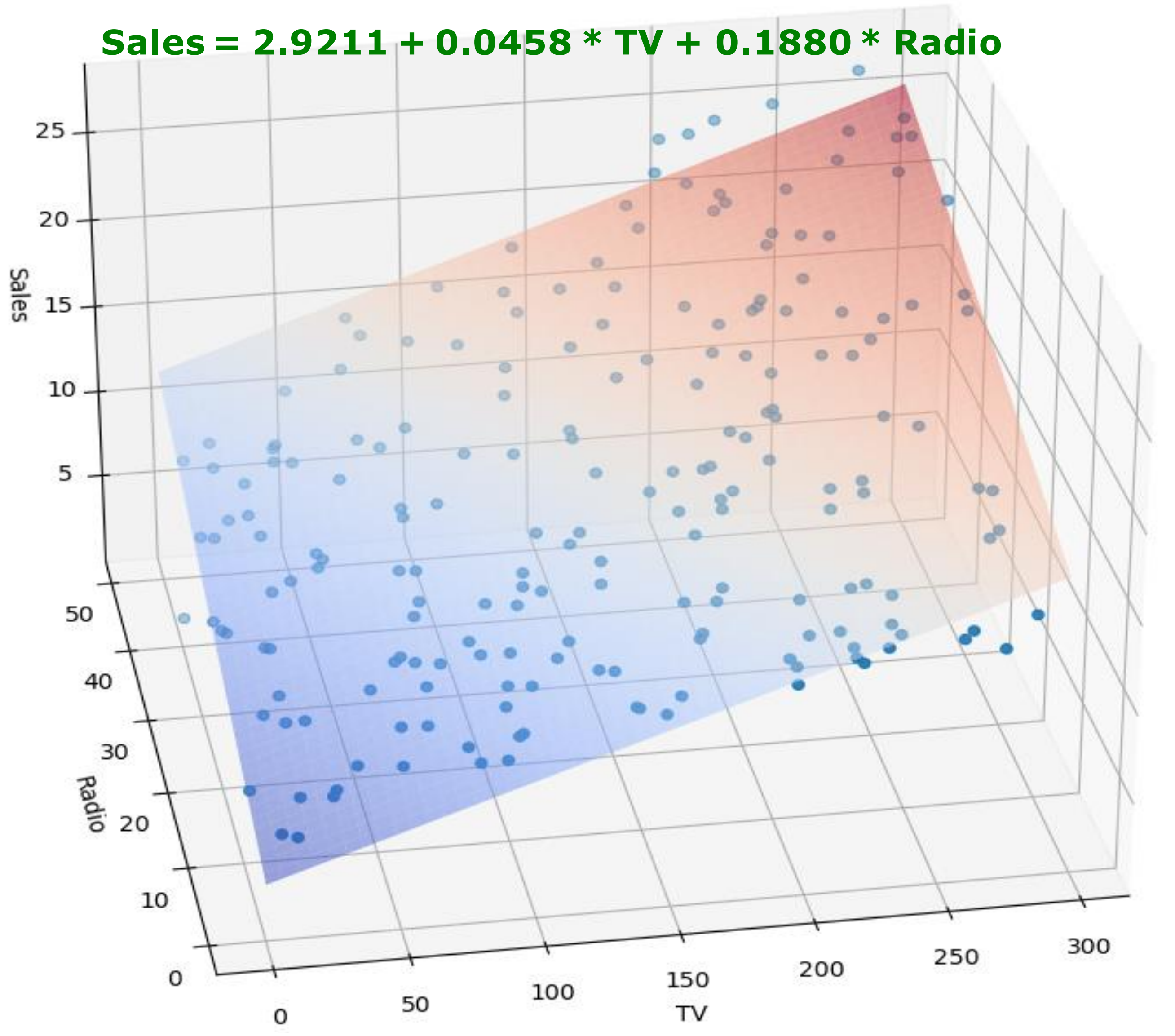
Y_i is an actual value

Hands-on Example: Multiple Linear Regression

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from mpl_toolkits.mplot3d import Axes3D

def main():
    # Load the advertising dataset into a pandas data frame
    df = pd.read_csv('Advertising.csv', index_col=0)
    y = df['Sales']
    X = df[['TV','Radio']]
    X = sm.add_constant(X)

    lr_model = sm.OLS(y,X).fit()
    print(lr_model.summary())
    print(lr_model.params)

    # Figure out X and Y axis using ranges from TV and Radio
    X_axis, Y_axis = np.meshgrid(np.linspace(X.TV.min(), X.TV.max(), 100), np.linspace(X.Radio.min(), X.Radio.max(), 100))

    # Plot the hyperplane by calculating corresponding Z axis (Sales)
    Z_axis = lr_model.params[0] + lr_model.params[1] * X_axis + lr_model.params[2] * Y_axis

    # Create matplotlib 3D axes
    fig = plt.figure(figsize=(12, 8))   # figsize refers to width and height of the figure
    ax = Axes3D(fig, azim=-100)

    # Plot hyperplane
    ax.plot_surface(X_axis, Y_axis, Z_axis, cmap=plt.cm.coolwarm, alpha=0.5, linewidth=0)

    # Plot data points
    ax.scatter(X.TV, X.Radio, y)

    # Set axis labels
    ax.set_xlabel('TV')
    ax.set_ylabel('Radio')
    ax.set_zlabel('Sales')

    plt.show()

if __name__ == "__main__":
    main()
```

	TV	Radio	Newspaper	Sales
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9
6	8.7	48.9	75	7.2
7	57.5	32.8	23.5	11.8
8	120.2	19.6	11.6	13.2
9	8.6	2.1	1	4.8
10	199.8	2.6	21.2	10.6

Sales = 2.9211 + 0.0458 * TV + 0.1880 * Radio



Gradient Descent

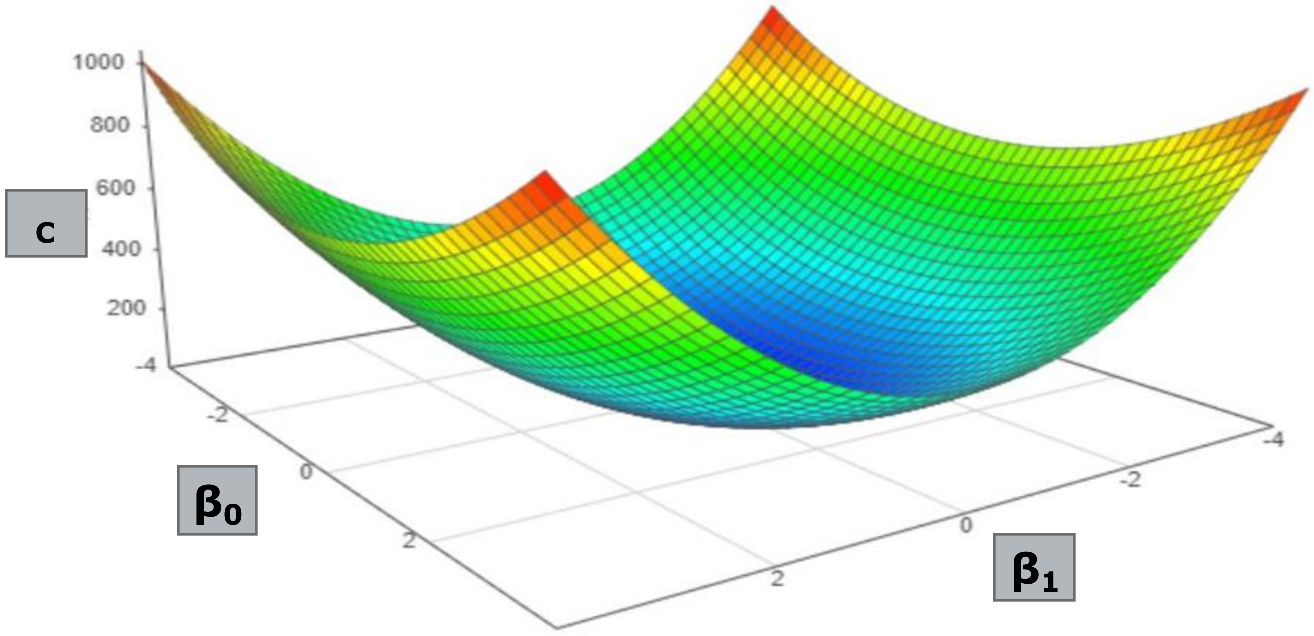
- Y = β_1X + β_0, where $\beta_1$is the slope (regression coefficient) and β_0 is the intercept (constant coefficient). C is the **cost or error** function, i.e., **the convex function**.

$$\text{Min } C = \frac{1}{2N}\sum_{i=1}^{N}(Y_i - (\beta_1 X_i + \beta_0))^2$$

- Find the best β_1 and β_0 set that will minimize C using gradient descent.

- The darker blue color indicates the lowest the error function value and the better it fits the data.

- The objective is to keep adjusting the function by picking different values for its parameters and seeing if that lowers the cost. Whenever we find the lowest cost, we stop and note the values of the parameters. Those parameter values comprise the most fitting model for the data. This is the essence of a technique called **gradient descent**.



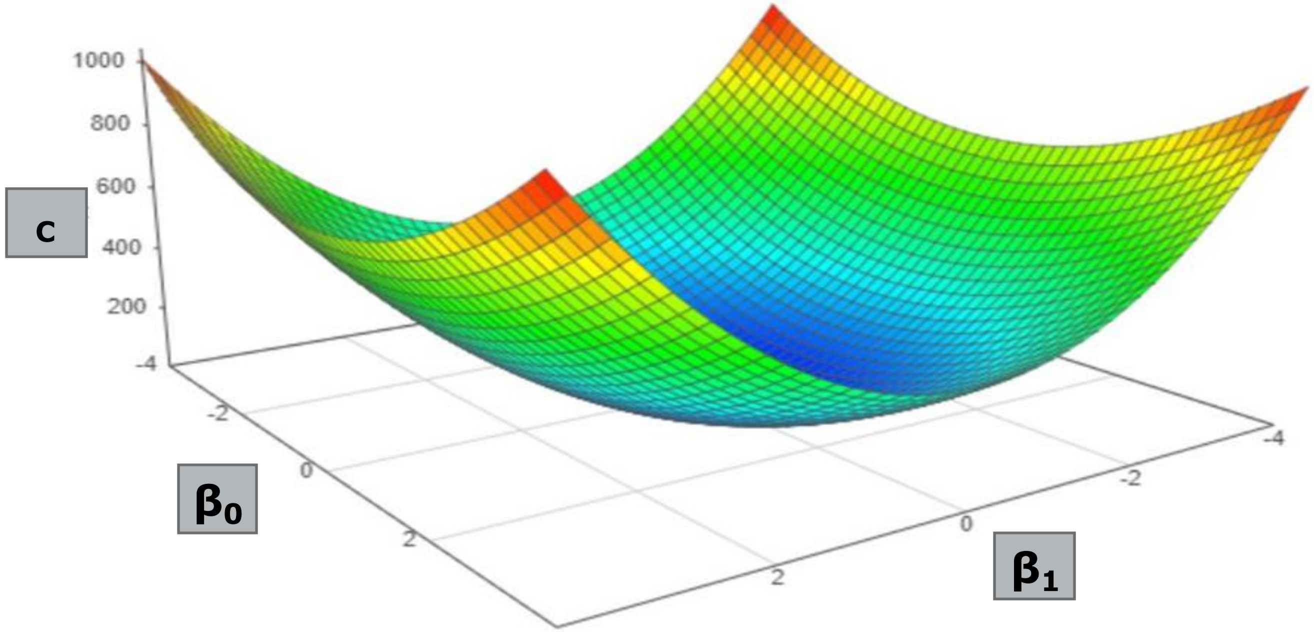
- It is an approach for looking for minima - points where the error is at its lowest.

Gradient Descent

- Compute the gradient or slope of C by differentiating C using the partial derivative for each parameter, i.e., β_1 and β_0.
  - $\frac{\partial C}{\partial \beta_1} = \frac{1}{N}\sum_{i=1}^{N}((\beta_1 X_i + \beta_0) - Y_i)X_i$
  - $\frac{\partial C}{\partial \beta_0} = \frac{1}{N}\sum_{i=1}^{N}((\beta_1 X_i + \beta_0) - Y_i)$

$$\text{Min } C = \frac{1}{2N}\sum_{i=1}^{N}(Y_i - (\beta_1 X_i + \beta_0))^2$$

- Initialize the search to start at ***any pair*** of β_1 and β_0 values (i.e., any line) and let the gradient descent algorithm march downhill on C toward the best line.



- Each point in this 3D figure represent a line, where the point has three dimensions.

- This 3D figure presents a whole bunch of possible lines that could fit the data.

- The darker blue the color at a point, the lower the error function value, and the better that point fits our data.

Gradient Descent

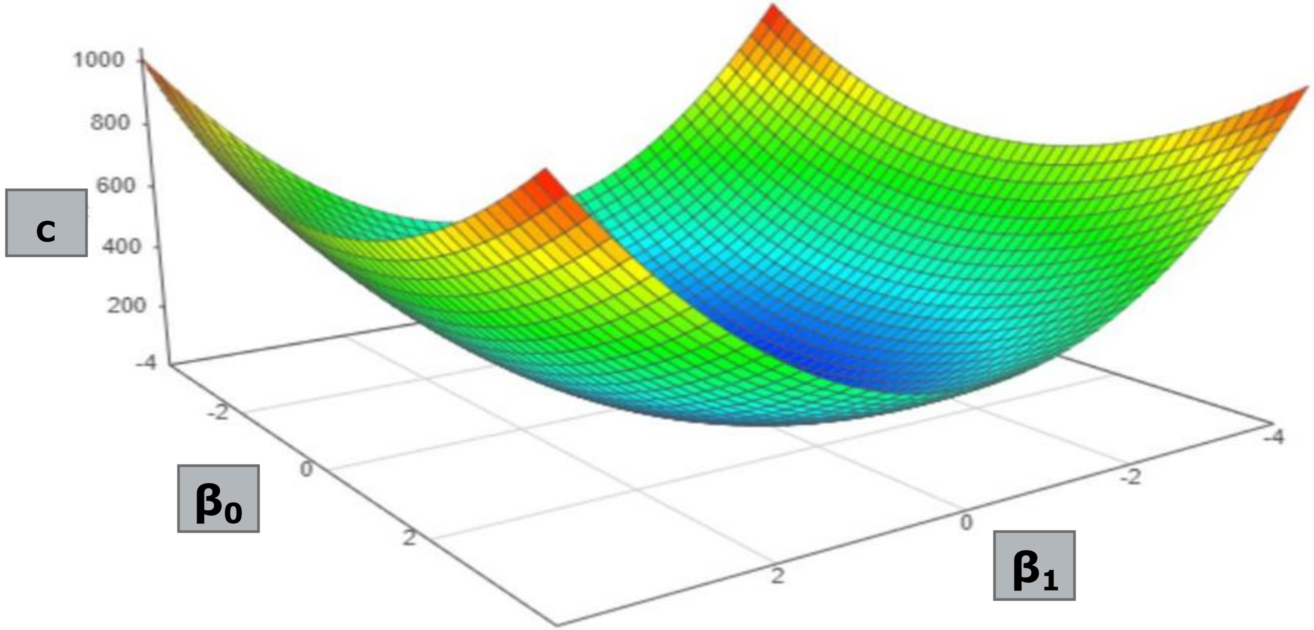
- Compute the gradient or slope of C by differentiating C using the partial derivative for each parameter, i.e., β_1 and β_0.

$$\text{Min } C = \frac{1}{2N}\sum_{i=1}^{N}(Y_i - (\beta_1 X_i + \beta_0))^2$$

- $\frac{\partial C}{\partial \beta_1} = \frac{1}{N}\sum_{i=1}^{N}((\beta_1 X_i + \beta_0) - Y_i)X_i$

- $\frac{\partial C}{\partial \beta_0} = \frac{1}{N}\sum_{i=1}^{N}((\beta_1 X_i + \beta_0) - Y_i)$

- Each iteration will ***update*** ***β_1 and β_0*** to a line that yields slightly lower error than the previous iteration.



Gradient Descent

- Image a Model that Could Have Any Number of Parameters of Inputs.

- $h(X) = \sum_{j=0}^{n} \theta_j X_j \text{ , } where\ \theta_0 = \beta_0,\ \theta_1 = \beta_1, and\ X_0 = 1$

- $J(\theta) = \frac{1}{2N} \sum_{i=1}^{N} (h(X_i) - Y_i)^2$

, where N is the number of data instances

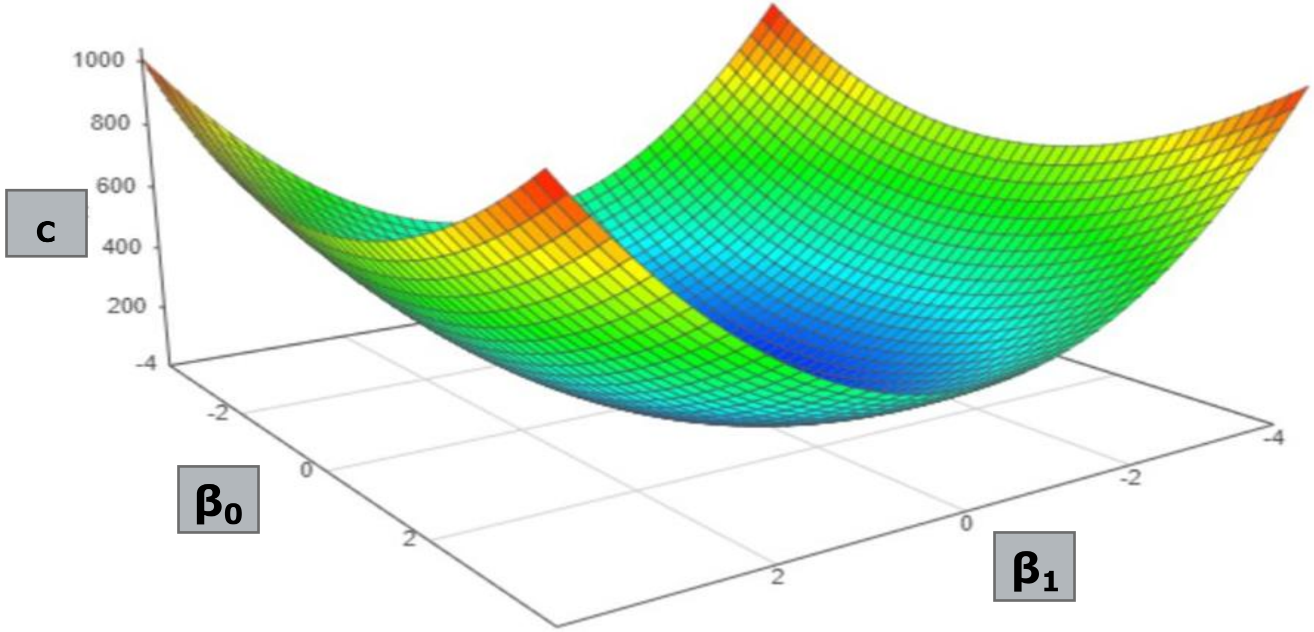
- $\frac{\partial J(\theta)}{\partial \theta_j} = \underbrace{\frac{1}{N} \sum_{i=1}^{N} (h(X_i) - Y_i) X_{ji}}_{\text{Slope = Gradient}}$

- $\theta_j = \theta_j - \alpha \frac{1}{N} \sum_{i=1}^{N} (h(X_i) - Y_i) X_{ji},$

where α is the learning rate with

the value between 0 and 1 & 0 ≤ j ≤ n.

$$\text{Min } C = \frac{1}{2N} \sum_{i=1}^{N} (Y_i - (\beta_1 X_i + \beta_0))^2$$



Gradient Descent

- Image a Model that Could Have Any Number of Parameters of Inputs.

- $h(X) = \sum_{j=0}^{n} \theta_j X_j \, , where \; \theta_0 = \beta_0, \; \theta_1 = \beta_1, and \; X_0 = 1$

- $J(\theta) = \frac{1}{2N}\sum_{i=1}^{N}(h(X_i) - Y_i)^2$

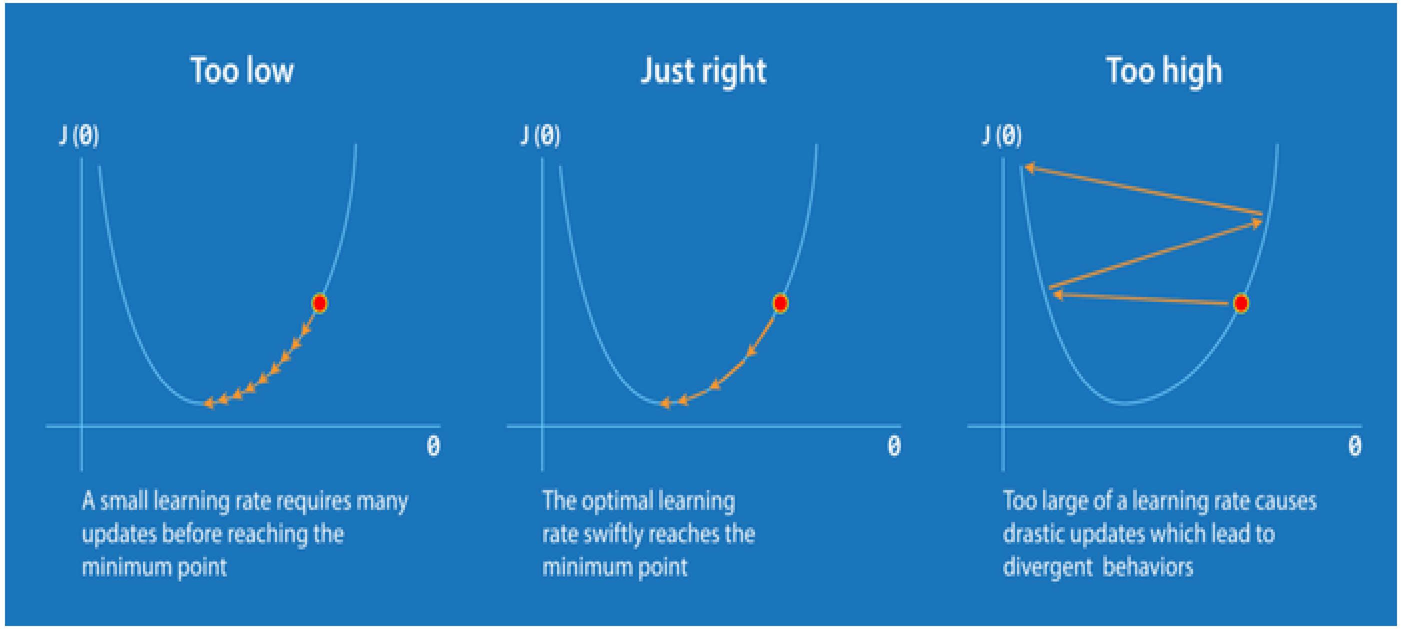
, where N is the number of data instances

- $\frac{\partial J(\theta)}{\partial \theta_j} = \underbrace{\frac{1}{N}\sum_{i=1}^{N}(h(X_i) - Y_i)X_{ji}}_{\text{Slope = Gradient}}$

- $\theta_j = \theta_j - \alpha\frac{1}{N}\sum_{i=1}^{N}(h(X_i) - Y_i)X_{ji},$

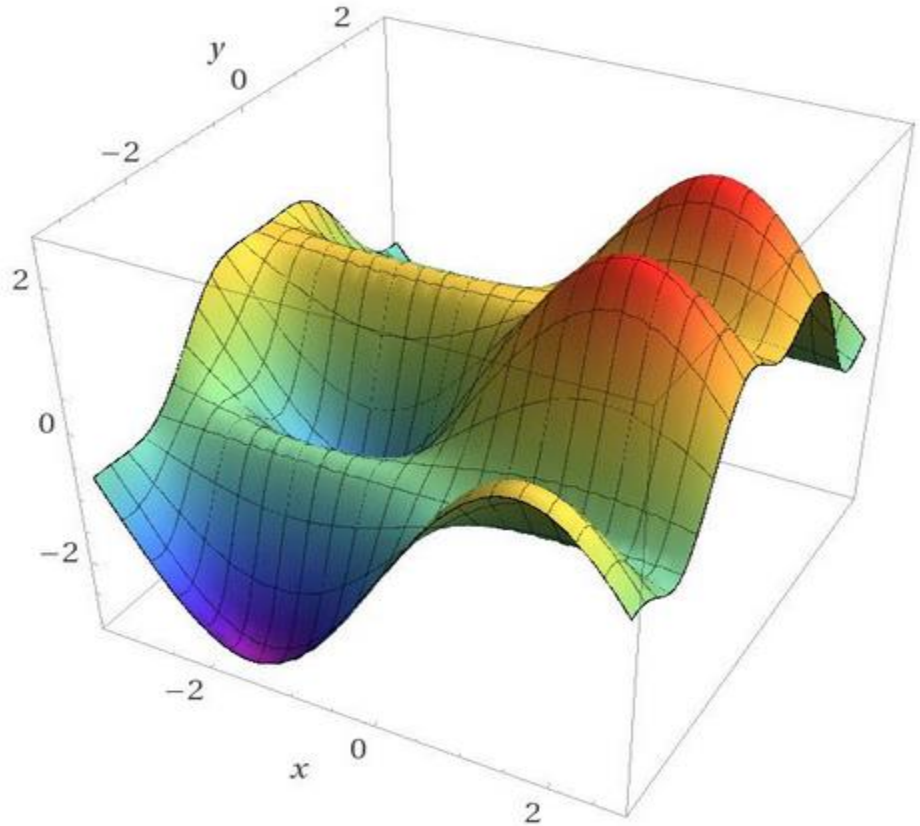
where α is the learning rate with

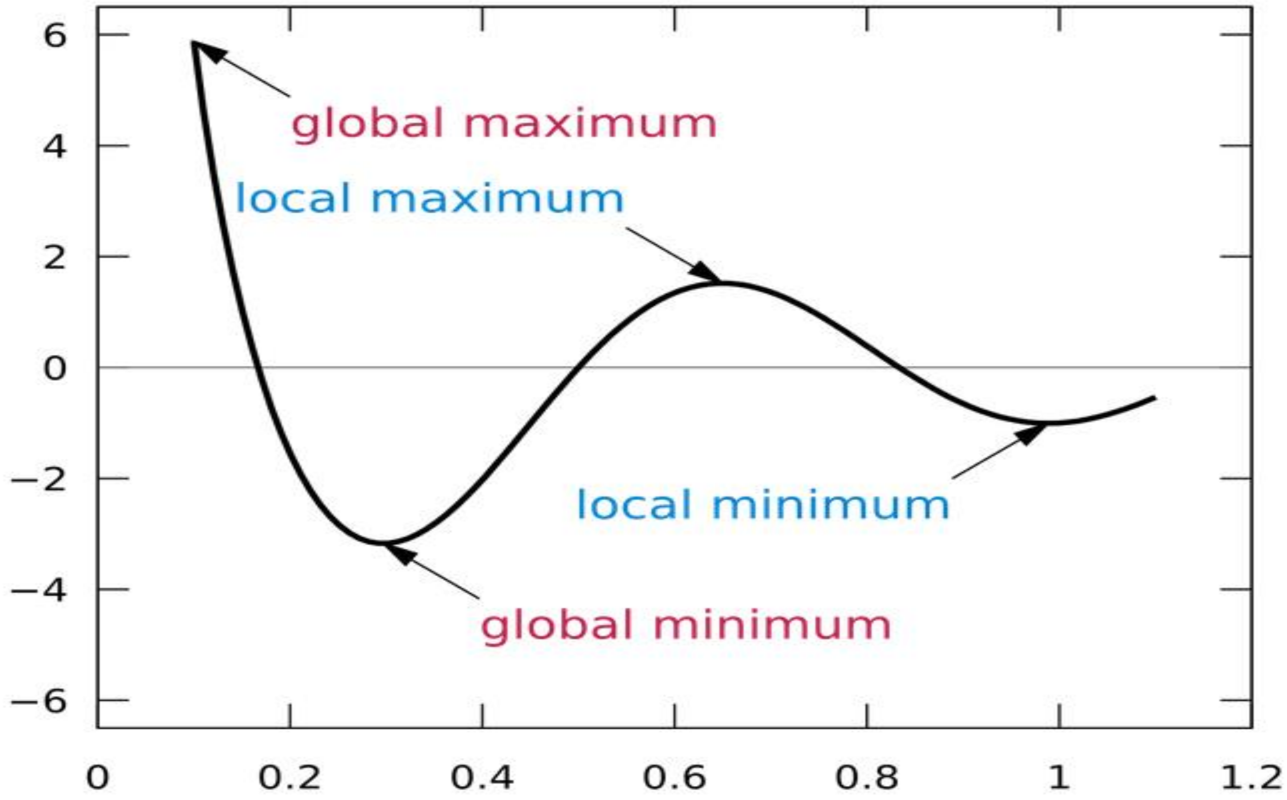
the value between 0 and 1 & $0 \le j \le n$.



Gradient Descent

- Gradient Descent is an iterative process that finds the minima of a function. This is an optimization algorithm that finds the parameters or coefficients of a function where the function has a minimum value. **Note that this function does not always guarantee to find a global minimum and can get stuck at a local minimum.**

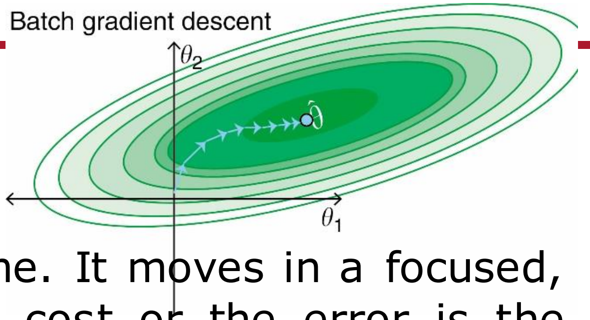




Gradient Descent

```
for i in range(num_epochs):
    grad = compute_gradient(data, params)
    params = params — learning_rate * grad
```

Batch Gradient Descent



In BGD, it looks at **ALL** the data point at a time. It moves in a focused, almost linear fashion to the point where the cost or the error is the lowest. It takes fewer steps, but each step is quite expensive.

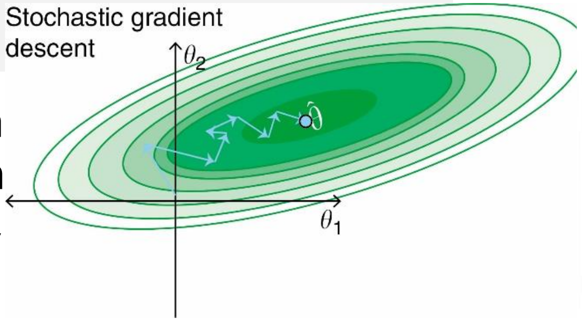
```
for i in range(num_epochs):
    np.random.shuffle(data)
    for batch in radom_minibatches(data, batch_size=32):
        grad = compute_gradient(batch, params)
        params = params — learning_rate * grad
```

Mini-Batch Gradient Descent

```
for i in range(num_epochs):
    np.random.shuffle(data)
    for example in data:
        grad = compute_gradient(example, params)
        params = params — learning_rate * grad
```

Stochastic Gradient Descent

In SGD, it only looks at **ONE** data point at a time. It could bounce around a lot moving in all kinds of directions. It takes far more steps, but each step is computationally cheap.



	STOCHASTIC	MINI-BATCH	BATCH
VELOCITY	✓	✓	✗
ACCURACY	✗	✓	✓
UPDATES	Every time	Every mini-batch	Once

Hands-on Example: Gradient Descent

epsilon is a threshold to determine the next iteration for the theta update.

The evolution of the regression line

m is the total number of data instances

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

x_prime = []

def batch_gradient_descent1(y, X, alpha, epsilon,
                            plot_on=-1, plot_func=None):
    # Initialize our "guess" for each coefficient theta_j to 1
    # and store these in a single column
    theta = np.ones(shape=(X.shape[1], 1))

    m = X.shape[0] # number of data points

    # Calculate a column of predicted y values for each data point
    y_hat = X @ theta

    # calculate a 1 by 1 matrix that holds the sum of the squared
    # differences between each y_hat and y
    cost = np.transpose(y_hat - y) @ (y_hat - y)

    # initialize list of costs to contain the cost associated with
    # our initial coefficients (scaled by 1/2m in accordance with
    # the cost formula)
    costs = [cost[0][0] / (2 * m)]

    i = 0 # number of iterations
    delta = 1 # Change in cost
```

```
    while (delta > epsilon):
        if (plot_on > 0 and i % plot_on == 0 and plot_func is not None):
            plot_func(theta)

        # calculate a column that holds the difference between y_hat and
        # y for each data point
        differences = X @ theta - y

        # Update each theta_j by the partial derivative of the cost with
        # respect to theta_j, scaled by learning rate

        # Note: np.transpose(X) gives us the observed values (x_j) for
        # a parameter j in the j'th row of a matrix
        theta = theta - (alpha / m) * ((np.transpose(X)) @ differences)

        # Using the updated coefficient values, append the new cost value
        cost = np.transpose(X @ theta - y) @ (X @ theta - y)
        costs.append(cost[0][0] / (2 * m))
        delta = abs(costs[i + 1] - costs[i])

        if (costs[i + 1] > costs[i]):
            print('Cost is increasing. Try reducing alpha.')
            break
        i += 1

    if (plot_on > 0 and i % plot_on == 0 and plot_func is not None):
        plot_func(theta)

    print('Completed in', i, 'iterations.')
    return theta
```

Hands-on Example: Gradient Descent

```python
def plot_model(theta):
    y_hat = [xp * theta[0] for xp in x_prime]
    plt.plot(x_prime, y_hat, color='dimgrey')

# Perform batch gradient descent, as in the previous example,
# but plots the evolution of cost as opposed to the evolution of
# the regression line
def batch_gradient_descent2(y, X, alpha, epsilon):
    theta = np.ones(shape=(X.shape[1], 1))
    m = X.shape[0]
    cost = np.transpose(X @ theta - y) @ (X @ theta - y)
    costs = [cost[0][0] / (2 * m)]

    i = 0
    delta = 1

    while (delta > epsilon):
        theta = theta - (alpha / m) * ((np.transpose(X)) @ (X @ theta - y))

        cost = np.transpose(X @ theta - y) @ (X @ theta - y)
        costs.append(cost[0][0] / (2 * m))
        delta = abs(costs[i + 1] - costs[i])

        if (costs[i + 1] > costs[i]):
            print('Cost is increasing. Try reducing alpha.')
            break
        i += 1

    print('Completed in', i, 'iterations.')
    # Plot the cost versus iteration
    plt.plot([i for i in range(len(costs))], costs)
    return theta
```

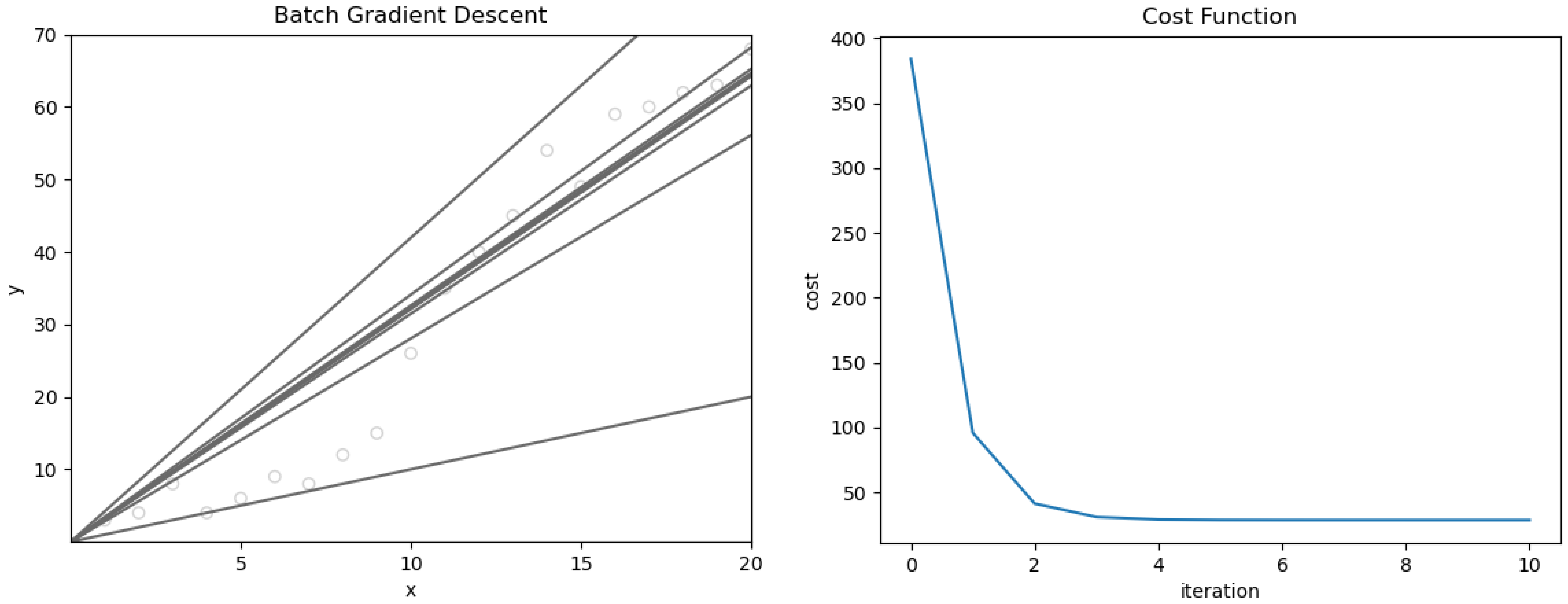
The evolution of cost over iteration

```python
def main():
    df = pd.read_csv('regression.csv')
    X = df[['x']]
    y = df[['y']]
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Batch Gradient Descent')
    xt = np.arange(0, df.x.max() + 5, 5)
    yt = np.arange(0, df.y.max() + 10, 10)
    plt.axis([xt[0], xt[-1], yt[0], yt[-1]])
    plt.xticks(xt[1:])
    plt.yticks(yt[1:])

    # originial data
    plt.scatter(df.x, df.y, facecolors='none', edgecolors='lightgray')
    global x_prime
    x_prime = [xt[0], xt[-1]]
    X = X.to_numpy()
    y = y.to_numpy()
    batch_gradient_descent1(y=y, X=X, alpha=0.01, epsilon=10**-4,
                            plot_on=1, plot_func=plot_model)
    plt.show()

    # Set up our predictor / response columns
    df = pd.read_csv('regression.csv')
    X = df[['x']].to_numpy()
    y = df[['y']].to_numpy()

    # Set up our plotting environment
    plt.xlabel('iteration')
    plt.ylabel('cost')
    plt.title('Cost Function')
    batch_gradient_descent2(y=y, X=X, alpha=0.01, epsilon=10**-4)
    plt.show()
```

x	y
1	3
2	4
3	8
4	4
5	6
6	9
7	8
8	12
9	15
10	26

```python
if __name__ == "__main__":
    main()
```

Hands-on Example: Gradient Descent



Simple Linear Regression Model

- Y = β_1X + β_0, where β_1 is the slope (regression coefficient) and β_0 is the intercept (constant coefficient).

- β_0 and β_1 Formula and Calculation

$$\frac{\partial C}{\partial \beta_1} = \frac{1}{N}\sum_{i=1}^{N}((\beta_1 X_i + \beta_0) - Y_i)X_i = 0$$

$$\frac{\partial C}{\partial \beta_0} = \frac{1}{N}\sum_{i=1}^{N}((\beta_1 X_i + \beta_0) - Y_i) = 0$$

Co-variance of X and Y

$$\beta_1 = \frac{\boxed{\sum_{i=1}^{N}(X_i - \bar{X})(Y_i - \bar{Y})}}{\boxed{\sum_{i=1}^{N}(X_i - \bar{X})^2}}$$

Variance of X

$$\beta_0 = \bar{Y} - (\beta_1 * \bar{X})$$

Linear Regression Model Assessment

- Coefficient of Determination - Multiple R-squared R^2:

$$R^2 = 1 - \boxed{\frac{\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{N}(Y_i - \bar{Y})^2}}$$ **Residual Error%**

  - $\hat{Y}_i$ is the predicted value of Y_i, where $\hat{Y}_i = \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_n X_{ni} + \beta_0$, and Y_i is the actual value, for $1 \le i \le N$
  - $\bar{Y}$ is the mean Y
  - $0 \le R^2 \le 1$
  - $R^2 \times 100\%$: The variation in Y is 'explained by' the variation in the predictors $(X_1, X_2, \ldots X_n)$.

Linear Regression Model Assessment

- Problems with R^2 – Too Many Predictor Variables: **Overfitting**

- Every time you add a predictor to a model, the *R*-squared increases. It never decreases.

- If a model has too many predictors , it begins ***modeling and accounting the random noise and variation in the variable data***.

- This condition is known as **overfitting** the model and it produces **misleadingly high *R*-squared values**.

What problem can you think of if you have an overfitting model?

Linear Regression Model Assessment

- Solution for **Overfitting**:

$$Adjusted\ R^2 = 1 - \left(\frac{N-1}{N-p}\right)(1 - R^2),$$

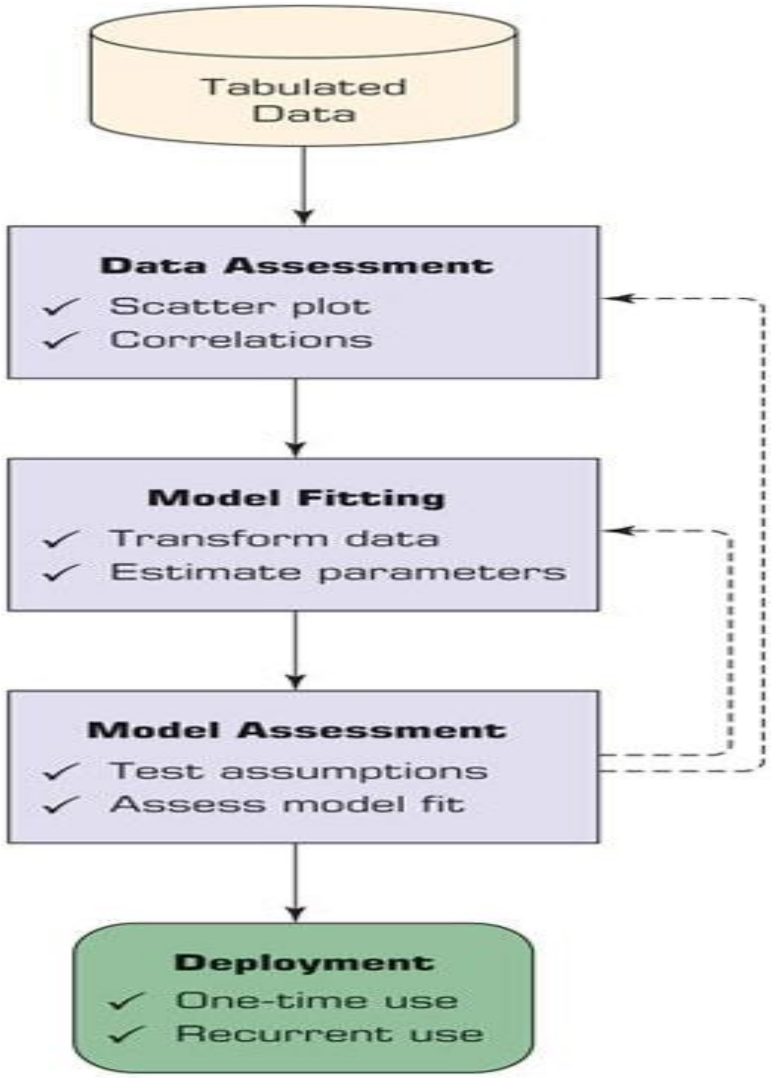
$$where\ N\ is\ the\ data\ size, and\ p = n + 1,$$

$$i.e., the\ number\ of\ \beta\ coefficients\ (\beta_0, \beta_1, \beta_2, \dots, \beta_n)\ in\ the\ model$$

- The adjusted R-squared R^2
  - Increase when a new predictor variable improves the model more than would be expected by chance ***OR***
  - Decrease when a new predictor variable improves the model by less than expected by chance.

Linear Regression Model

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



Supervised Learning - Classification

- If our question is answered by **YES/NO/MAYBE**, we are facing a classification problem.

- Classifiers are also the models to use if our question admits only **a discrete set of answers**, i.e., we want to select from a finite number of choices.
  - Given the results of a clinical test, e.g., does this patient suffer from diabetes?
  - Given a magnetic resonance image, is it a tumor shown in the image?
  - Given the past activity associated with a credit card, is the current operation fraudulent?

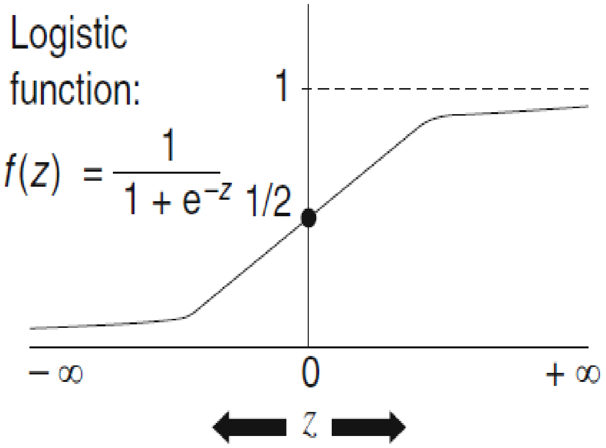
- According to **the cardinality of the target variable**, one usually distinguishes between **binary classifiers** when the target output only takes **two values**, i.e., the classifier answers questions with a yes (1) or a no (0); and **multiclass classifiers**, for **a larger number of classes**.

Logistic Regression Model

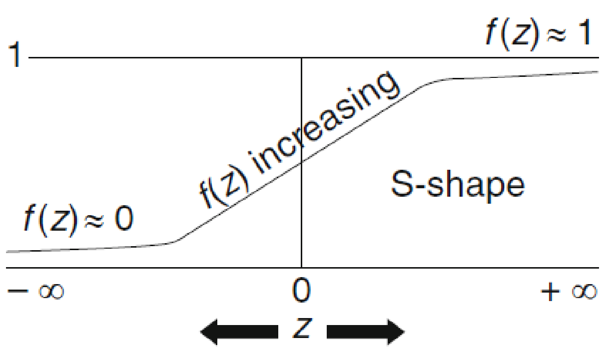
- Purpose of Logistic Regression Model:
  - Predict the decision outcome/response of an event (i.e., do or not do) by **learning regression coefficients of the logistic distribution function of the explanatory variables**.
  - Binary Decision-Making Model, i.e., **a binary target variable**.
  - Some Decision Examples include:
    - A customer should purchase or rent a car
    - An investor decides whether or not to buy or sell a stock
    - An employee should be promoted or not
    - A company determines whether or not they should target a customer to sell a long-term deposit.
    - A bank should give a person a loan or not?
    - A person should decide to vote against a new law or not?
    - The school will accept my admittance or not?

Logistic Regression Model

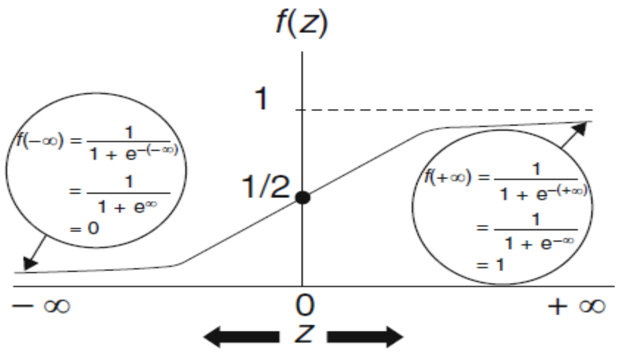
- Logistic Function f(z)
  - Describes the mathematical form on which the nonlinear logistic regression model is based.



Shape:



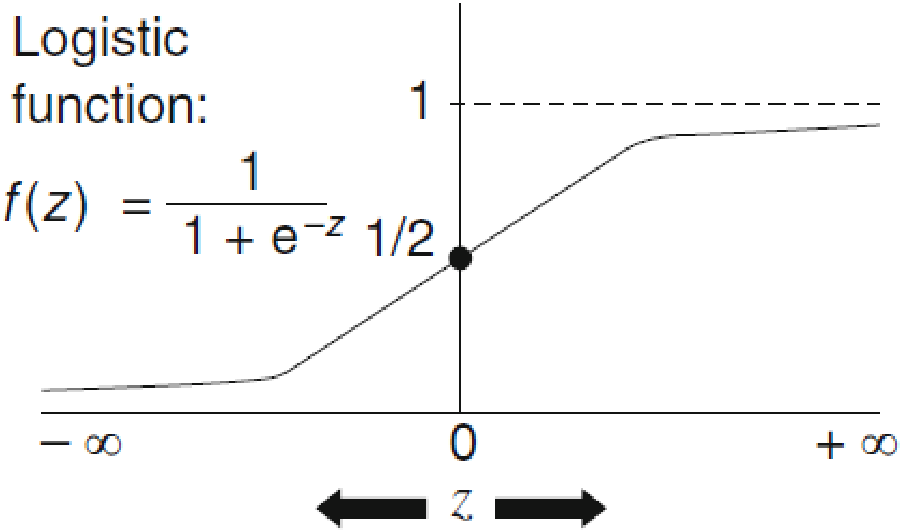
  - What is e? e = 2.7182818284590452
  - -∞ ≤ Z ≤ +∞



Range: $0 \le f(z) \le 1$

Logistic Regression Model

- Logistic Function f(z)
  - Describes the mathematical form on which the nonlinear logistic regression model is based.
  - $z = \alpha + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k$ is a linear regression function.



$$z = \alpha + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k$$

$$f(z) = \frac{1}{1 + e^{-z}}$$

$$= \frac{1}{1 + e^{-(\alpha + \sum \beta_i X_i)}}$$

Logistic Regression Model

- Logistic Function f(z) → P(X)
  - Describes the mathematical form on which the nonlinear logistic regression model is based.
  - $z = \alpha + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k$ is a linear function.
  - $X = (X_1, X_2, \ldots, X_k)$

$$\underbrace{z = \alpha + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k}$$

$$f(z) = \frac{1}{1 + e^{-z}}$$

$$= \frac{1}{1 + e^{-(\alpha + \sum \beta_i X_i)}}$$

Model formula:

$$\boxed{P(\mathbf{X}) = \frac{1}{1 + e^{-(\alpha + \sum \beta_i X_i)}}}$$

If P(X) ≥ 0.5, then 1 indicates a decision event occurs.
If P(X) < 0.5, then 0 indicates a decision event does not occur.

Logistic Regression Model

- Example for Applying the Logistic Regression Model
  - Coronary Heart Disease (CHD): 0 without CHD and 1 with CHD
    - Catecholamine Level (CAT): 1 if high and 0 if low
    - AGE: Continuous Numbers
    - Electrocardiogram Status (ECG): 1 if abnormal and 0 if normal

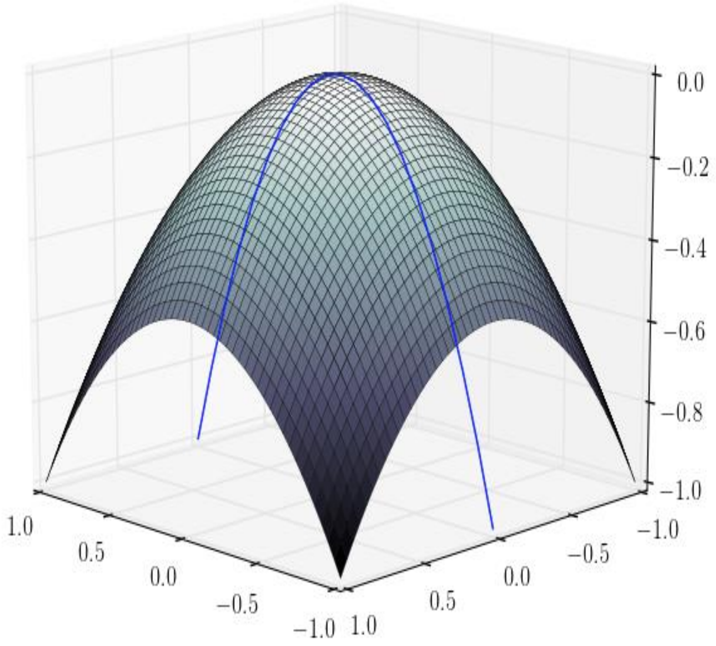
$$P(X) = \frac{1}{1 + e^{-(\alpha + \beta_1 CAT + \beta_2 AGE + \beta_3 ECG)}}$$

  - $P(X) = \frac{1}{1+e^{-(\alpha + \sum_{i=1}^{3} \beta_i X_i)}}$
  - X = (CAT=1, AGE=40, ECG=0)
  - Use the dataset to estimate parameters α = -3.911, β_1=0.652, β_2=0.029, and β_3=0.342.
  - $P(X) = \frac{1}{1+e^{-(-3.911+0.652(1)+0.029(40)+0.342(0))}}$
  - $P(X) = \frac{1}{1+e^{-(-2.101)}} = \frac{1}{1+8.173} = 0.110 < 0.5 \rightarrow Without\ CHD$

Logistic Regression Model

- Example for Applying the Logistic Regression Model
  - Coronary Heart Disease (CHD): 0 without CHD and 1 with CHD
    - Catecholamine Level (CAT): 1 if high and 0 if low
    - AGE: Continuous Numbers
    - Electrocardiogram Status (ECG): 1 if abnormal and 0 if normal

$$P(X) = \frac{1}{1 + e^{-(\alpha + \beta_1 CAT + \beta_2 AGE + \beta_3 ECG)}}$$



  - Maximum Likelihood Estimation: https://online.stat.psu.edu/stat415/lesson/1/1.2
  - $\theta_j = \theta_j - \sigma \sum_{i=1}^{N} (h_\theta(X_i) - Y_i) X_{ji}$ - Gradient Ascent

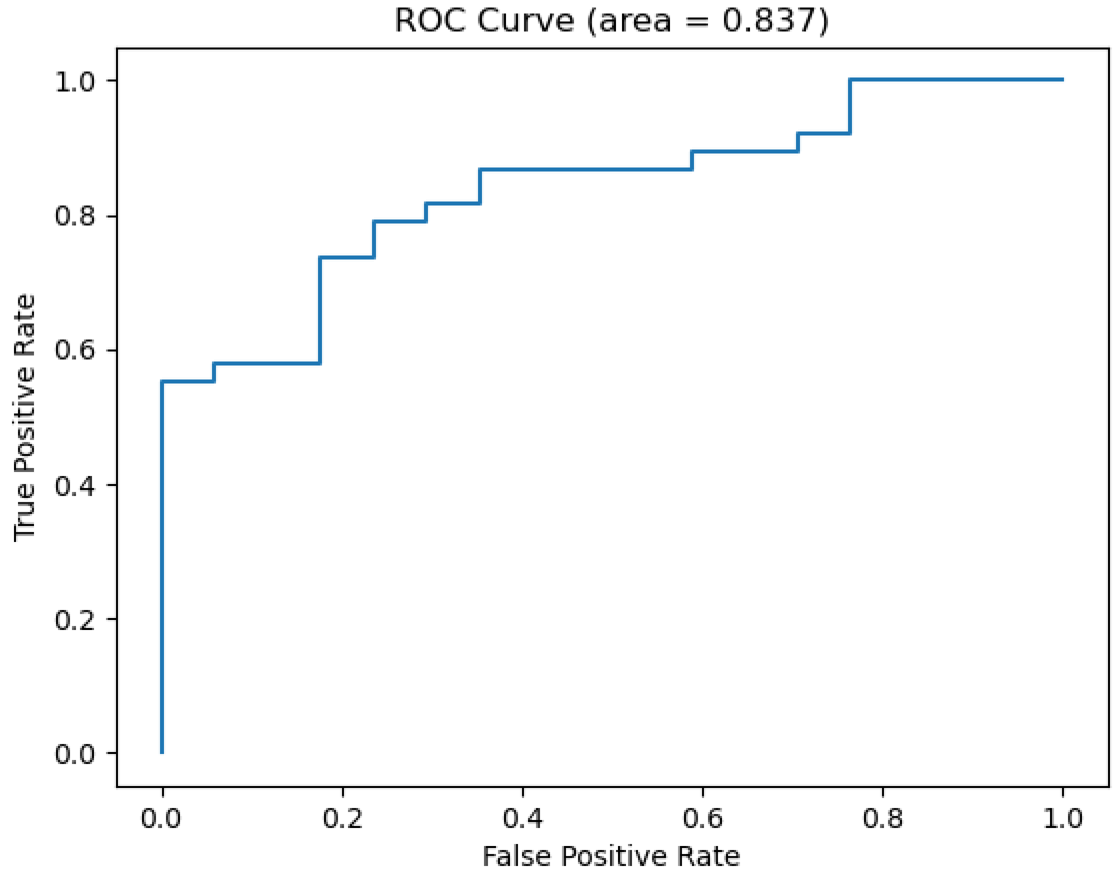
Logistic Regression Model

- $0 \leq$ McFadden $R^2 \leq 1$
  - The closer this statistic is to one, the better the predictive ability of the model.
  - The value of the McFadden R^2 never decreases but increases as additional variables are added to the model, leading to potential ***overfitting*** problems.
  - If the model with additional variables fits perfectly, the McFadden R^2 equals to one.
- ***Solution***: Minimal Akaike Information Criterion (AIC)

Hands-on Example: Logistic Regression Model

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25		S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925		S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S
6	0	3	Moran, Mr. James	male		0	0	330877	8.4583		Q

```python
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

def main():
    t_df = pd.read_csv('titanic_data.csv', index_col='PassengerId')
    t_df = t_df.dropna()
    t_df.drop(columns=['Name', 'Cabin', 'Ticket'], inplace=True)
    t_df['Sex'].replace(['male', 'female'], [1, 0], inplace=True)
    t_df['Embarked'].replace(['S', 'C', 'Q'], [0, 1, 2], inplace=True)
    X = t_df.drop(columns=['Survived'])
    y = t_df['Survived']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)

    logmodel = sm.Logit(y_train, sm.add_constant(X_train)).fit(disp=False)
    print(logmodel.summary())

    # Form our predictions, convert continuous [0, 1] predictions to binary
    predictions = logmodel.predict(sm.add_constant(X_test))
    bin_predictions = [1 if x >= 0.5 else 0 for x in predictions]

    # We can now assess the accuracy and print out the confusion matrix
    print(accuracy_score(y_test, bin_predictions))
    print(confusion_matrix(y_test, bin_predictions))

    fpr, tpr, thresholds = roc_curve(y_test, predictions)
    roc_auc = roc_auc_score(y_test, predictions)
    plt.plot(fpr, tpr, label='ROC Curve (area = %0.3f)' % roc_auc)
    plt.title('ROC Curve (area = %0.3f)' % roc_auc)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.show()

if __name__ == "__main__":
    main()
```



Logistic Regression Model

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

MLR vs. LR

Multiple Linear Regression Model		Logistic Regression Model	
Input Predictors	Continuous Target Variable	Input Predictors	Binary Target Variable
$\beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \beta_0$	Y	$\beta_1 X_1 + \beta_2 X_2 + \ldots +$ βnXn $+ \beta_0$	Y
Model Builder - Learning		Model Builder - Learning	
$\hat{Y} = \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \beta_0$		$P(X_1, X_2, \ldots, X_k) = \frac{1}{1+e^{-(\beta_0 + \sum_{i=1}^{k} \beta_i x_i)}}$ If $P(X_1, X_2, \ldots, X_k) \geq 0.5$, $\hat{Y} = 1$ If $P(X_1, X_2, \ldots, X_k) < 0.5$, $\hat{Y} = 0$	

Y is an actual value, and $\hat{Y}$ is the predicted value.