Odds: The Language Behind Logistic Regression

When we talk about probabilities, we usually think in terms of values between 0 and 1. But in many domains—especially **gambling, statistics, and logistic regression**—probability is often expressed using a different but closely related concept: **odds.**

Understanding odds is the key to understanding *why logistic regression looks the way it does.*

Odds

Odds are another way of quantifying the probability of an event, commonly used in gambling (and logistic regression).

Odds

For some event E,

$$\text{odds}(E) = \frac{P(E)}{P(E^c)} = \frac{P(E)}{1 - P(E)}$$

Similarly, if we are told the odds of E are x to y then

$$\text{odds}(E) = \frac{x}{y} = \frac{x/(x+y)}{y/(x+y)}$$

which implies

$$P(E) = x/(x+y), \quad P(E^c) = y/(x+y)$$

What Are Odds?

For an event E, the **odds** are defined as:

$$\text{odds}(E) = \frac{P(E)}{P(E^c)} = \frac{P(E)}{1 - P(E)}$$

Interpretation:
Odds compare how likely an event is to occur **relative to it not occurring.**

Example

If:

$$P(E) = 0.75$$

Then:

$$\text{odds}(E) = \frac{0.75}{0.25} = 3$$

This is read as:

> **"3 to 1 in favor of the event."**

Odds Written as "x : y"

Odds are often written as $x : y$, meaning:

- x favorable outcomes
- y unfavorable outcomes

From the definition:

$$\text{odds}(E) = \frac{x}{y}$$

To convert odds back into probabilities, we normalize:

$$P(E) = \frac{x}{x + y}$$

$$P(E^c) = \frac{y}{x + y}$$

> **Important**
>
> Odds and probabilities contain the **same information** — they are just different representations.

Example: The Donner Party

In **1846**, the **Donner and Reed families** left *Springfield, Illinois* for *California* by covered wagon. In **July**, the group—later known as the **Donner Party**—reached *Fort Bridger, Wyoming*. There, its leaders decided to attempt a **new and untested route** to the *Sacramento Valley*.

At its full size, the party consisted of **87 people and 20 wagons**. The journey was delayed first by a difficult crossing of the **Wasatch Range**, and again while crossing the desert west of the **Great Salt Lake**. These delays proved disastrous.

In late **October**, the group became stranded in the **eastern Sierra Nevada mountains** when the region was hit by heavy snowfall. Trapped by extreme weather and dwindling supplies, the party faced severe famine and exposure.

By the time the last survivor was rescued on **April 21, 1847, 40 of the 87 members had died** due to starvation and extreme cold.

Source

Ramsey, F. L., & Schafer, D. W. (2002). *The Statistical Sleuth: A Course in Methods of Data Analysis* (2nd ed.).

```
df = pd.read_csv('donner_party1.csv')
df.head()
```

✓ 0.0s Python

	Survived	Name	Sex	Age	Marital Status
0	1	Patrick Breen	M	51.0	M
1	1	Margaret Breen	F	40.0	M
2	1	John Breen	M	14.0	S
3	1	Patrick Breen Jr.	M	9.0	S
4	1	Simon Preston Breen	M	8.0	S

An example EDA on the dataset- Donner Party

Example:

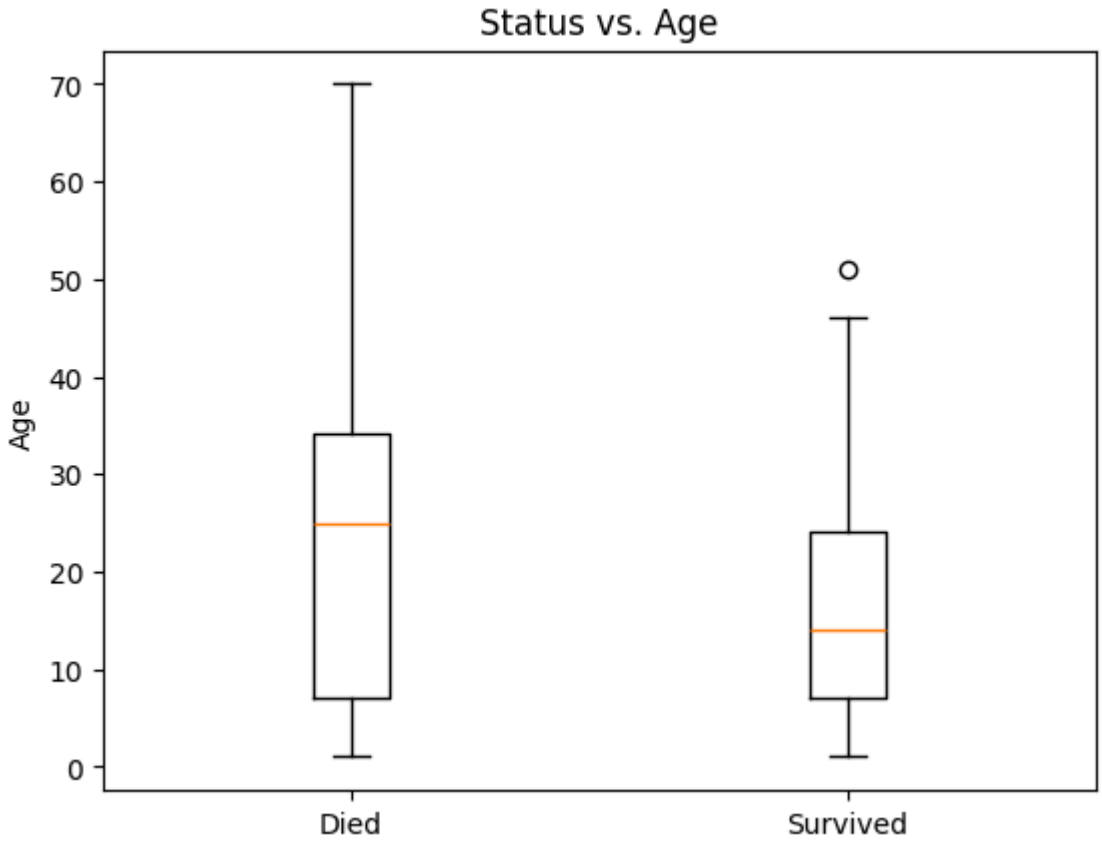
Status Vs Gender

```
pd.crosstab(df["Survived"], df["Sex"])
```

Sex	F	M
Survived		
0	10	31
1	23	25

Box plot of Age vs Survived

```
import matplotlib.pyplot as plt



ages_died = df[df["Survived"] == 0]["Age"].dropna()

ages_survived = df[df["Survived"] == 1]["Age"].dropna()



plt.figure()

plt.boxplot([ages_died, ages_survived], labels=["Died", "Survived"])

plt.ylabel("Age")

plt.title("Status vs. Age")



plt.show()
```



It seems clear that both age and gender have an effect on someone's survival, how do we come up with a model that will let us explore this
relationship?

Even if we set Died to 0 and Survived to 1, this isn't something we can
transform our way out of - we need something more

One way to think about the problem - we can treat Survived and Died as
successes and failures arising from a binomial distribution where the
probability of a success is given by a transformation of a linear model of
the predictors.

Generalized Linear Models (GLMs)

Generalized Linear Models (GLMs) provide a **unified framework** for handling regression problems where the response variable does **not** follow a normal distribution.

Instead of modeling the response directly, GLMs relate the **expected value of the response** to a **linear predictor** through a *link function*.

Why GLMs Are Needed

Classical linear regression assumes:

- Normally distributed errors
- Constant variance
- A continuous response variable

Many real-world problems violate these assumptions. For example:

- Binary outcomes (survived / died)
- Counts (number of events)
- Rates and proportions

GLMs were developed to handle exactly these cases.

Structure of a GLM

A generalized linear model consists of three components:

1. **Random component**
   Specifies the probability distribution of the response
   (e.g., Bernoulli, Binomial, Poisson)
2. **Systematic component**
   A linear predictor

   $\eta = w^T x$
3. **Link function**
   Connects the mean of the response to the linear predictor

   $g(E[Y]) = \eta$

Logistic Regression as a GLM

Logistic regression is a **special case** of a generalized linear model where:

- Response distribution: **Bernoulli**
- Link function: **Logit**
- Linear predictor: wTxw^T xwTx

log(P(y=1|x)1-P(y=1|x))=wTx\log\left(\frac{P(y=1 \mid x)}{1 - P(y=1 \mid x)}\right) = w^T xlog(1-P(y=1|x)P(y=1|x))=wTx

This formulation allows logistic regression to model **binary outcomes** while maintaining the simplicity of a linear model.

> **Key Takeaway**
>
> **Logistic regression is just one example of a generalized linear model.**
> GLMs provide a flexible framework for extending linear regression to many different types of data.

Logistic Regression

Objective

Predict the probability of a binary class using the logistic (sigmoid) function.

Algorithm

1. Initialize weight vector `w` and bias `b`
2. Choose learning rate `α` and number of iterations `T`
3. Repeat for `T` iterations:
   - Compute linear output
     `z = X · w + b`
   - Apply sigmoid
     `y_hat = 1 / (1 + exp(-z))`

- Compute gradients
  `dw = (1/n) · Xᵀ · (y_hat − y)`
  `db = (1/n) · sum(y_hat − y)`
- Update parameters
  `w = w − α · dw`
  `b = b − α · db`

4. Return `w` and `b`

Logistic Regression Algorithm

```
Algorithm: LogisticRegression
Input:
    X  → training data (n × d)
    y  → labels (n)
    α  → learning rate
    T  → number of iterations

Output:
    w  → weights
    b  → bias

Initialize w = zeros(d)
Initialize b = 0

for t = 1 to T:
    z = X · w + b
    y_hat = 1 / (1 + exp(-z))

    dw = (1/n) · Xᵀ · (y_hat − y)
    db = (1/n) · sum(y_hat − y)

    w = w − α · dw
    b = b − α · db

return w, b
```

Sigmoid Function

```
sigmoid(z) = 1 / (1 + exp(-z))
```