Week 3: Probability Distributions

POP88162 Introduction to Quantitative Research Methods

Tom Paskhalis

Department of Political Science, Trinity College Dublin

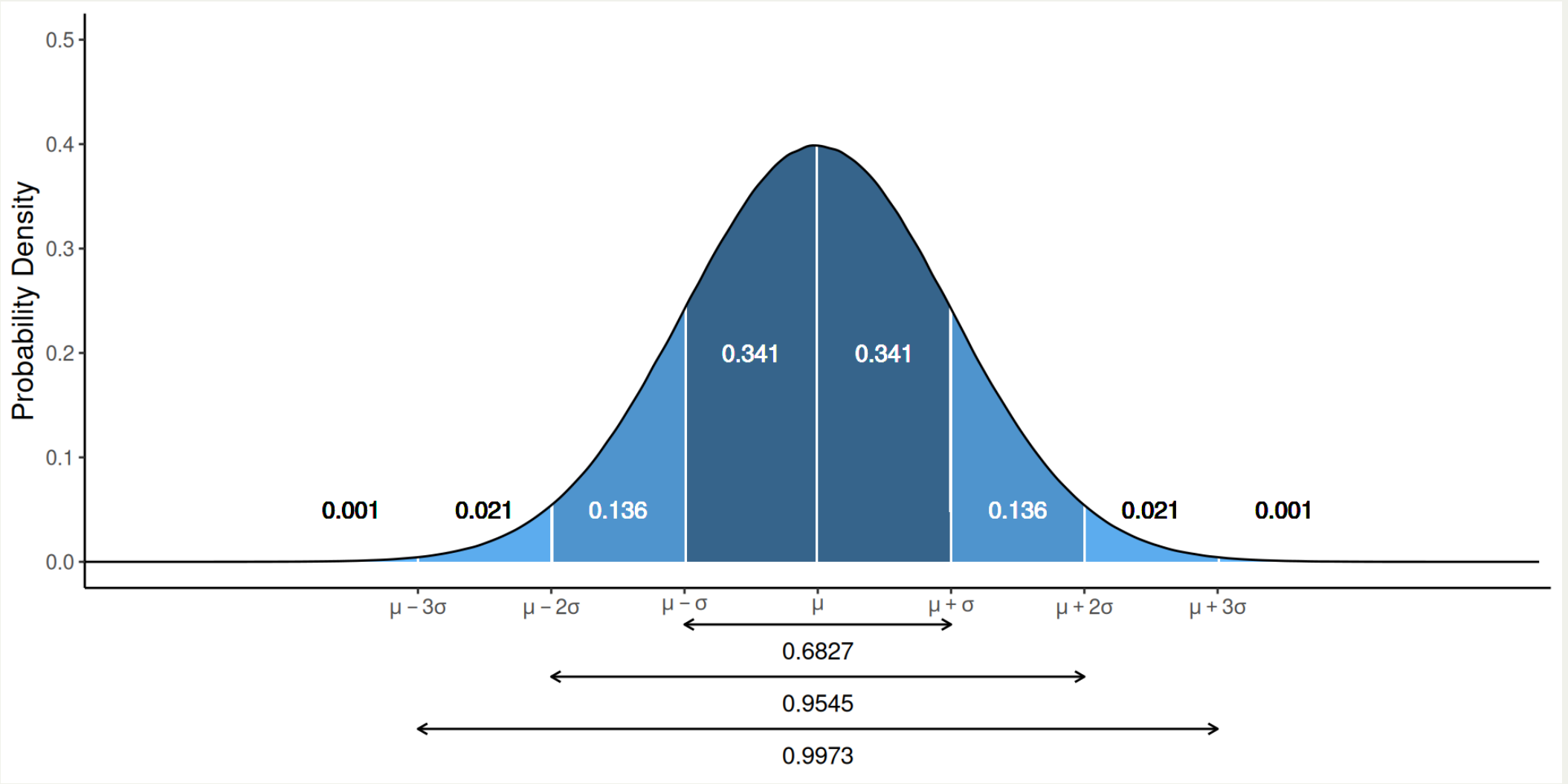
So Far

- Code in R can be distributed as R Scripts and R Markdown.
- Everything in R is an object, the references to which are established through *assignment*.
- *Vector* is the core *data structure* in R.
- They hold only data of the same type and are used to represent individual variables.
- Vectors can be one of the three main types (*character*, *numeric*, *logical*).
- Lists are used to store different data types (multiple individual vectors).

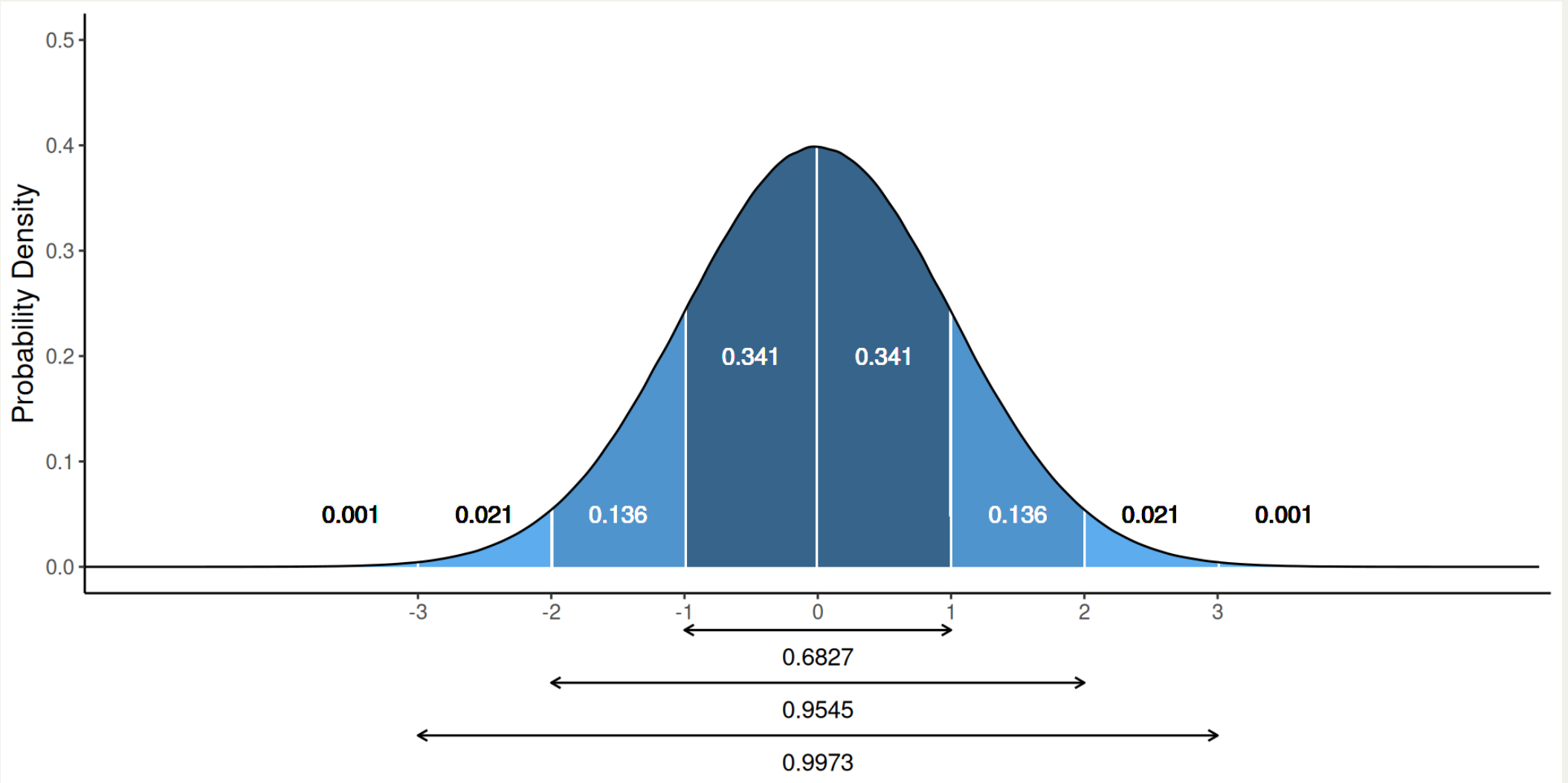
Topics for Today

- Probability functions in R
- Cumulative distribution function
- Quantiles
- Random number generation

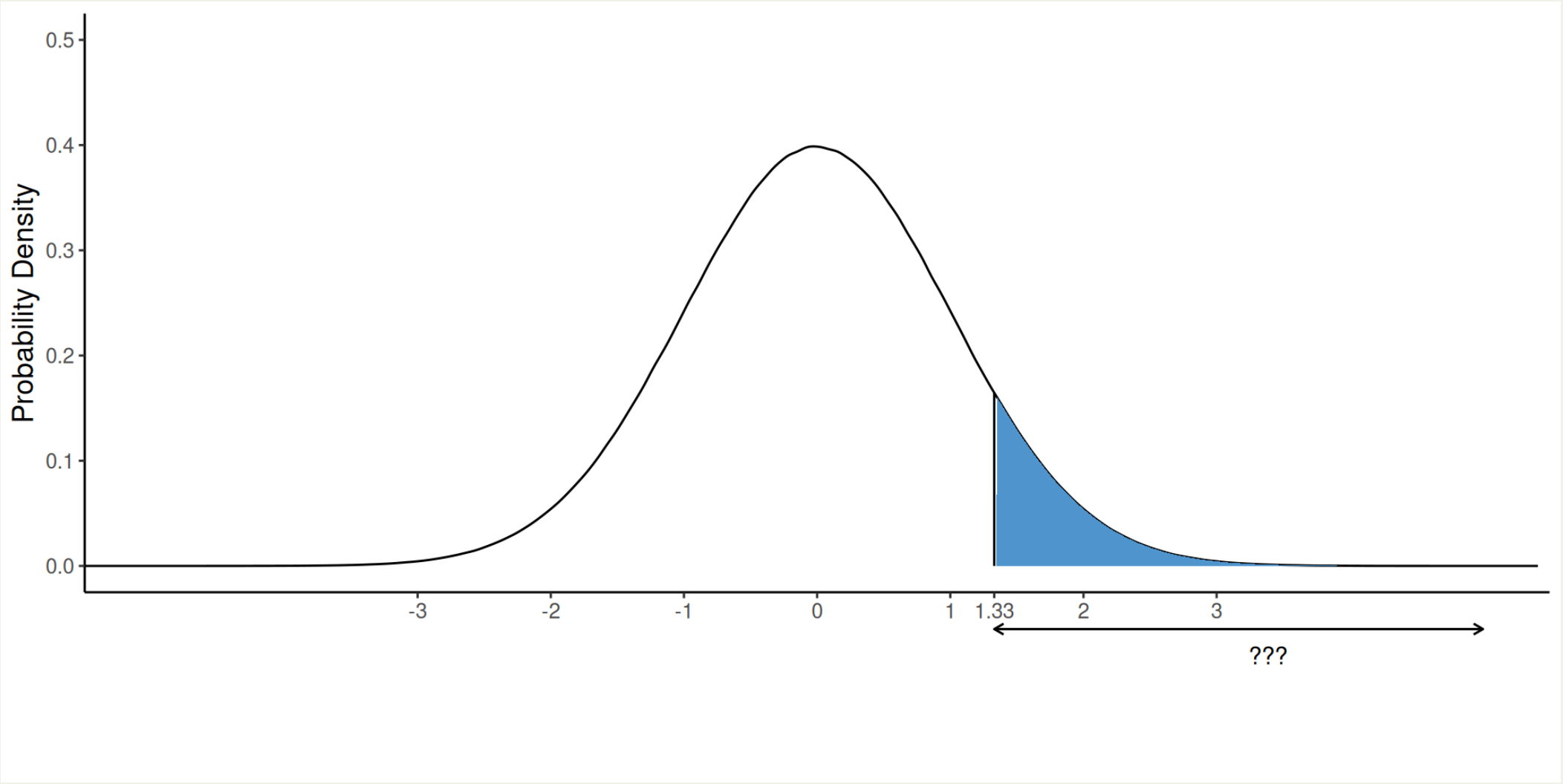
Normal Distribution



Standard Normal Distribution



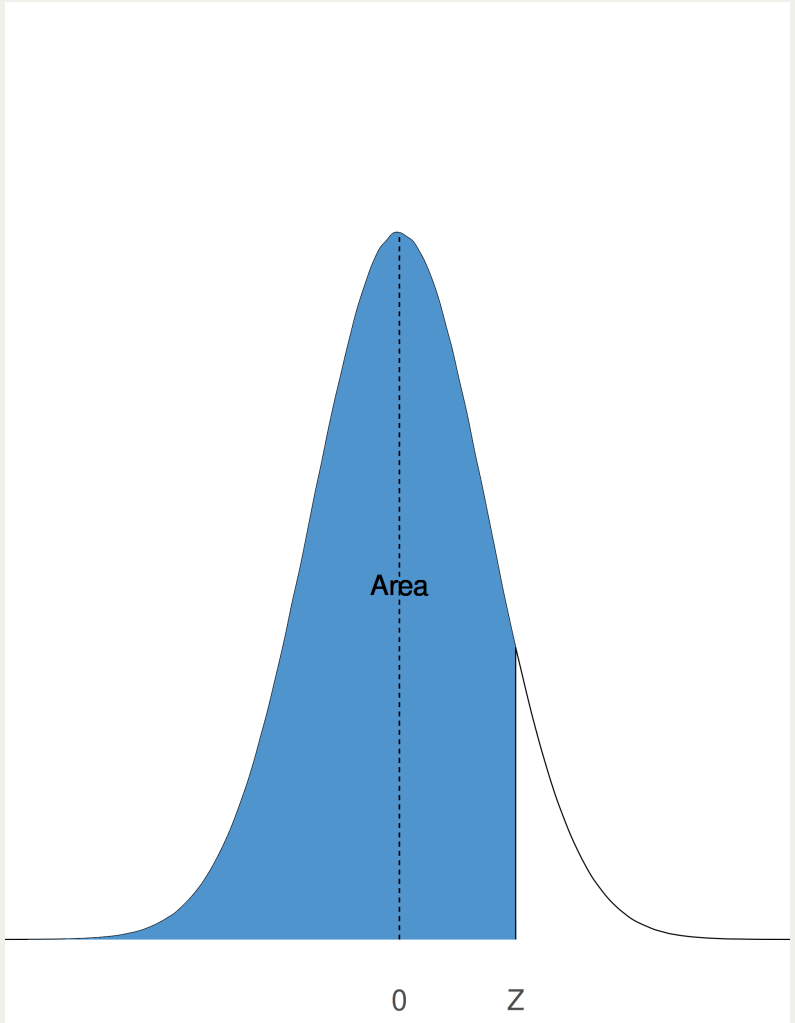
How to Find an Arbitrary Value?



Statistical Tables

Areas under the Normal Curve

Z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
-3.1	0.0010	0.0009	0.0009	0.0009	0.0008	0.0008	0.0008	0.0008	0.0007	0.0007
-3.0	0.0013	0.0013	0.0013	0.0012	0.0012	0.0011	0.0011	0.0011	0.0010	0.0010
-2.9	0.0019	0.0018	0.0018	0.0017	0.0016	0.0016	0.0015	0.0015	0.0014	0.0014
-2.8	0.0026	0.0025	0.0024	0.0023	0.0023	0.0022	0.0021	0.0021	0.0020	0.0019
-2.7	0.0035	0.0034	0.0033	0.0032	0.0031	0.0030	0.0029	0.0028	0.0027	0.0026
-2.6	0.0047	0.0045	0.0044	0.0043	0.0041	0.0040	0.0039	0.0038	0.0037	0.0036
-2.5	0.0062	0.0060	0.0059	0.0057	0.0055	0.0054	0.0052	0.0051	0.0049	0.0048
-2.4	0.0082	0.0080	0.0078	0.0075	0.0073	0.0071	0.0069	0.0068	0.0066	0.0064
-2.3	0.0107	0.0104	0.0102	0.0099	0.0096	0.0094	0.0091	0.0089	0.0087	0.0084
-2.2	0.0139	0.0136	0.0132	0.0129	0.0125	0.0122	0.0119	0.0116	0.0113	0.0110
-2.1	0.0179	0.0174	0.0170	0.0166	0.0162	0.0158	0.0154	0.0150	0.0146	0.0143
-2.0	0.0228	0.0222	0.0217	0.0212	0.0207	0.0202	0.0197	0.0192	0.0188	0.0183
-1.9	0.0287	0.0281	0.0274	0.0268	0.0262	0.0256	0.0250	0.0244	0.0239	0.0233
-1.8	0.0359	0.0351	0.0344	0.0336	0.0329	0.0322	0.0314	0.0307	0.0301	0.0294
-1.7	0.0446	0.0436	0.0427	0.0418	0.0409	0.0401	0.0392	0.0384	0.0375	0.0367
-1.6	0.0548	0.0537	0.0526	0.0516	0.0505	0.0495	0.0485	0.0475	0.0465	0.0455
-1.5	0.0668	0.0655	0.0643	0.0630	0.0618	0.0606	0.0594	0.0582	0.0571	0.0559
-1.4	0.0808	0.0793	0.0778	0.0764	0.0749	0.0735	0.0721	0.0708	0.0694	0.0681
-1.3	0.0968	0.0951	0.0934	0.0918	0.0901	0.0885	0.0869	0.0853	0.0838	0.0823
-1.2	0.1151	0.1131	0.1112	0.1093	0.1075	0.1056	0.1038	0.1020	0.1003	0.0985



Built-in Distributions

- Instead of relying on statistical tables, we can use built-in R functions.
- R provides functions for working with all common probability distributions.
- We can calculate critical values and probabilities.
- Conducting many statistical tests requires calculation of probabilities for continuous random variables.

Four Key Quantities

- The four key quantities for a probability distribution are:
  - **Density** or **point probability**
  - **Cumulated probability**
  - **Quantiles**
  - **Pseudo-random numbers**
- R has the tools for calculating those for a number of distributions!

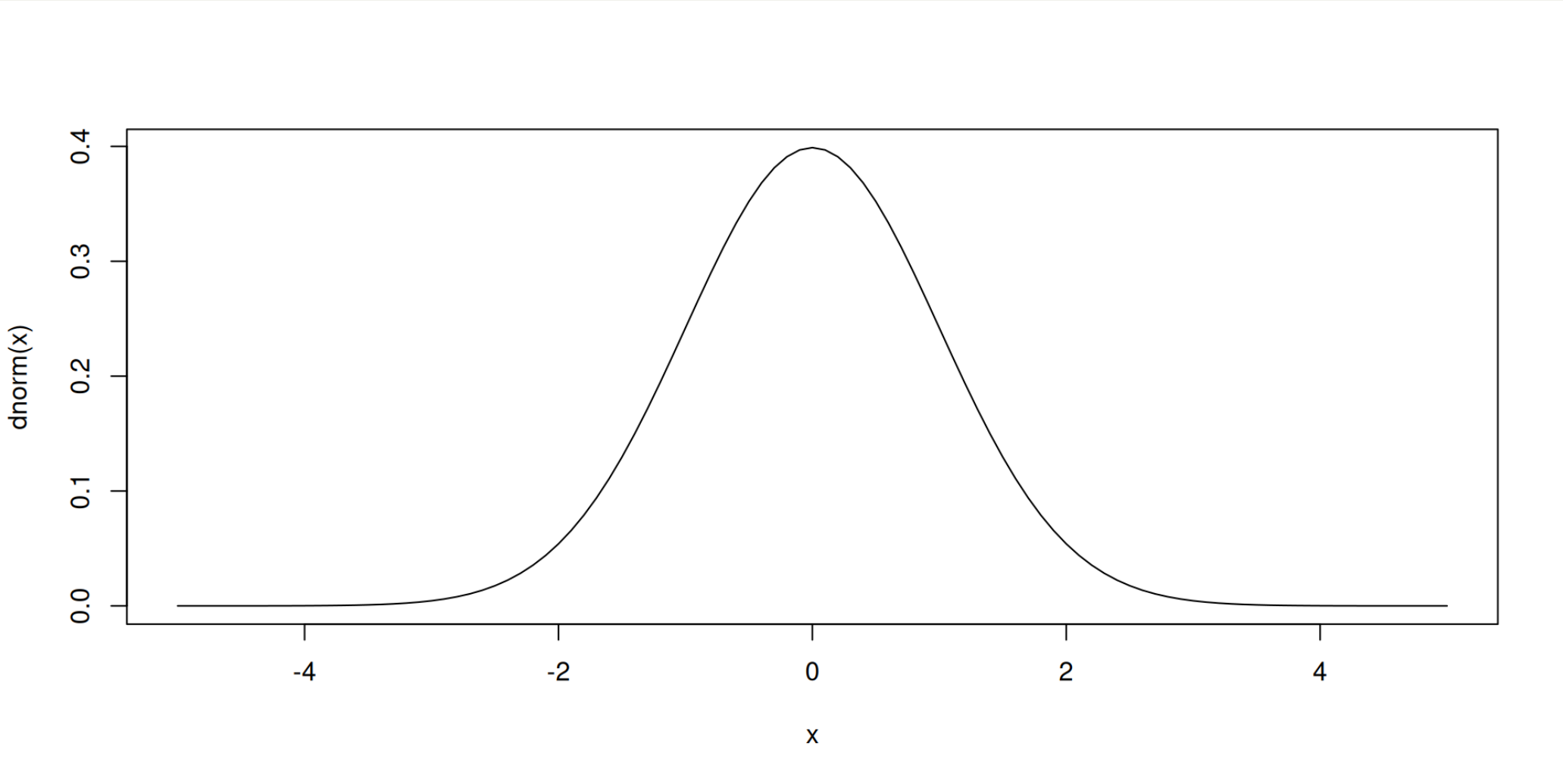
Density

- For discrete random variables (discrete distributions) we can calculate point probability (probability that a random variable takes a specific value).
- For continuous random variables we can only calculate **probability density** (area under the curve) for a specific interval.
- In R functions that calculate the density start with `d`:

```
dnorm(x, mean = 0, sd = 1)
dchisq(x, df)
dexp(x)
dpois(x, lambda)
dt(x, df, ncp)
dunif(x)
```

Density for Normal Distribution

```
1 x <- seq(-5, 5, 0.1)
2 plot(x, dnorm(x), type = "l")
```

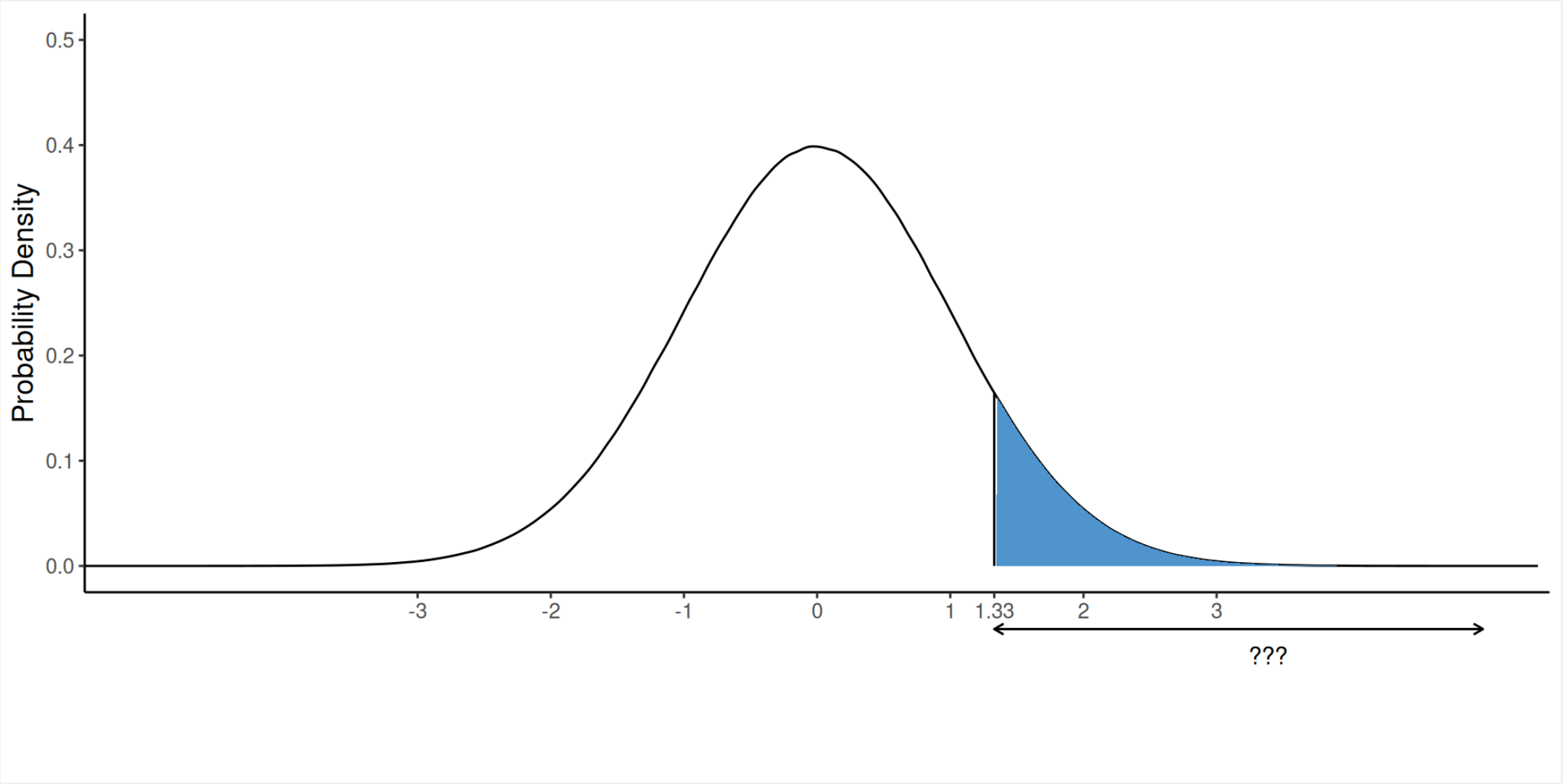


Cumulated Distribution Function (CDF)

- Describes the probability of getting x or less in a given distribution.
- While density functions in R are used more for visualisations, CDFs are often used for calculating actual probabilities.
- Corresponding R functions start with `p` (for **p**robability)

```
pnorm(q, mean = 0, sd = 1)
pchisq(q, df)
pexp(q)
ppois(q, lambda)
pt(q, df, ncp)
punif(q)
```

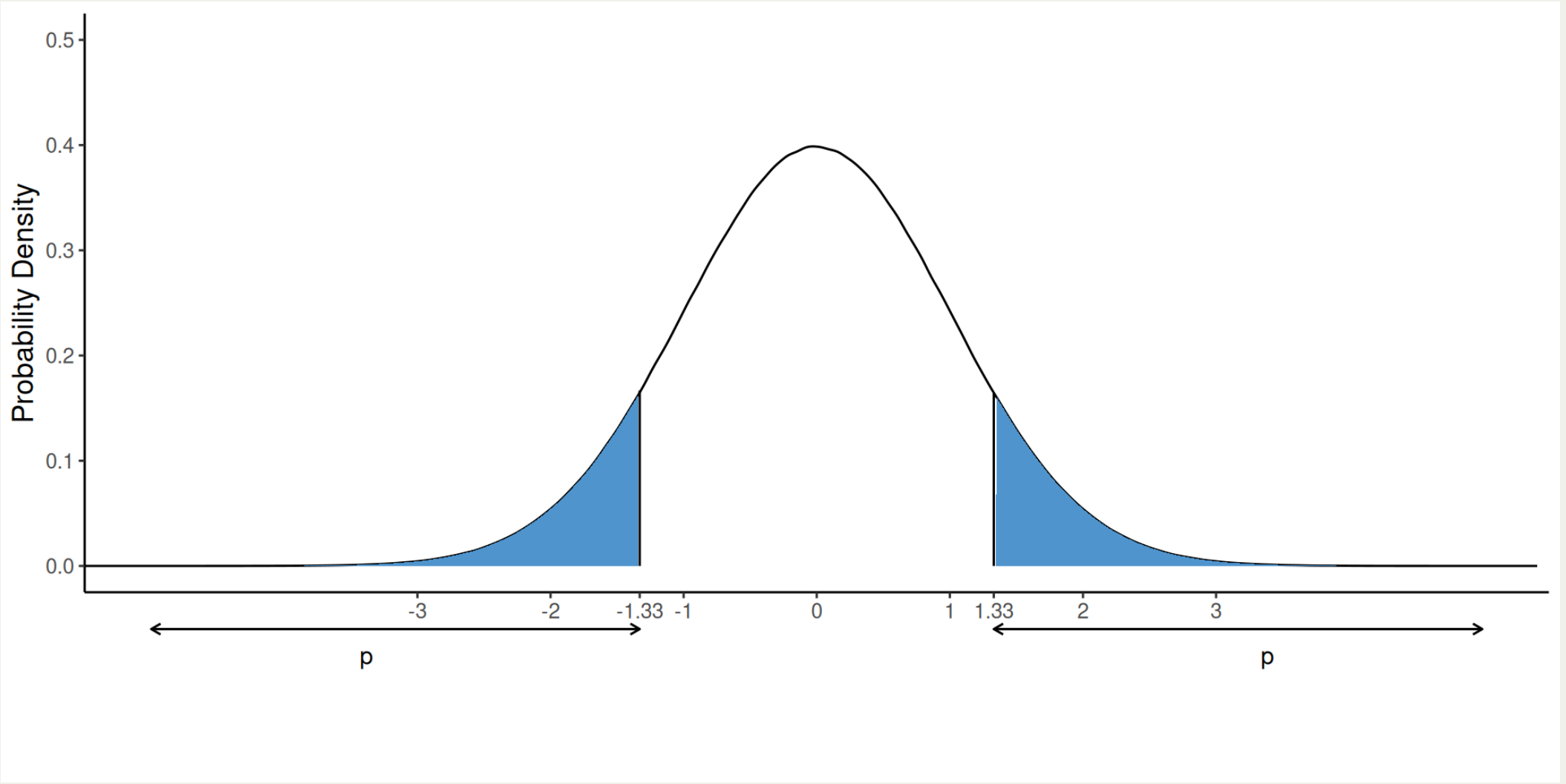
How to Find an Arbitrary Value?



2 Ways

1. We note that standard normal distribution ($\mu = 0, \sigma = 1$) is symmetric.
    - Thus, the area under the curve to the left from a symmetric point on the other end should be the same
2. We note that the total area under the curve is 1.
    - So we can calculate the area to the left from a given point and subtract it from 1

First Way



First Way

- Let's now calculate it in R using `pnorm()` function.

```
1 pnorm(-1.33)
```

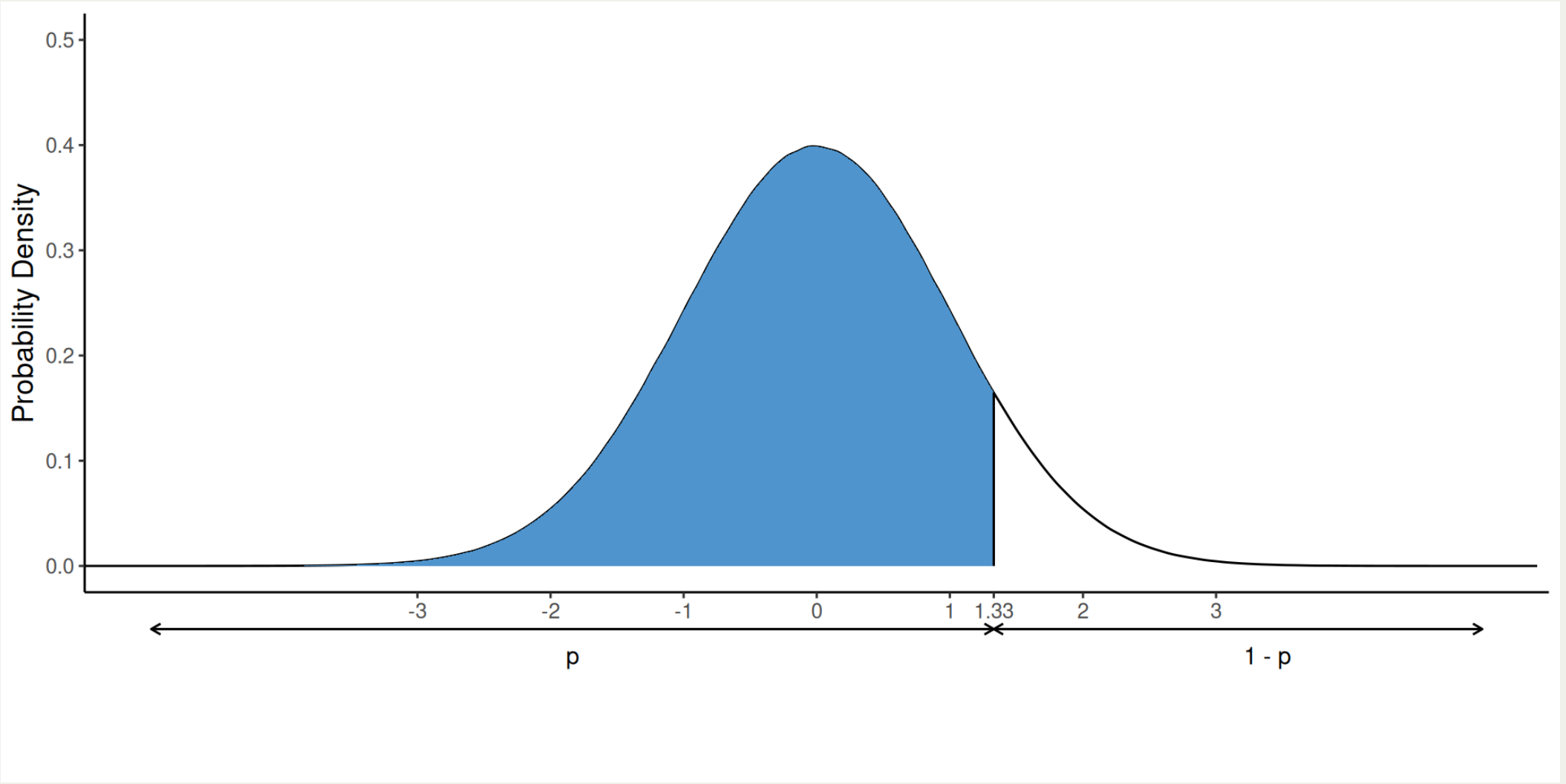
```
[1] 0.09175914
```

- Note that doing it for the original point gives results to the left and not to the right from it.

```
1 pnorm(1.33)
```

```
[1] 0.9082409
```

Second Way



Second Way

- Let's now calculate it in R using `pnorm()` function.

```r
1 - pnorm(1.33)
```

```
[1] 0.09175914
```

- Note that both methods (as we would expect) produce identical results.

```r
pnorm(-1.33)
```

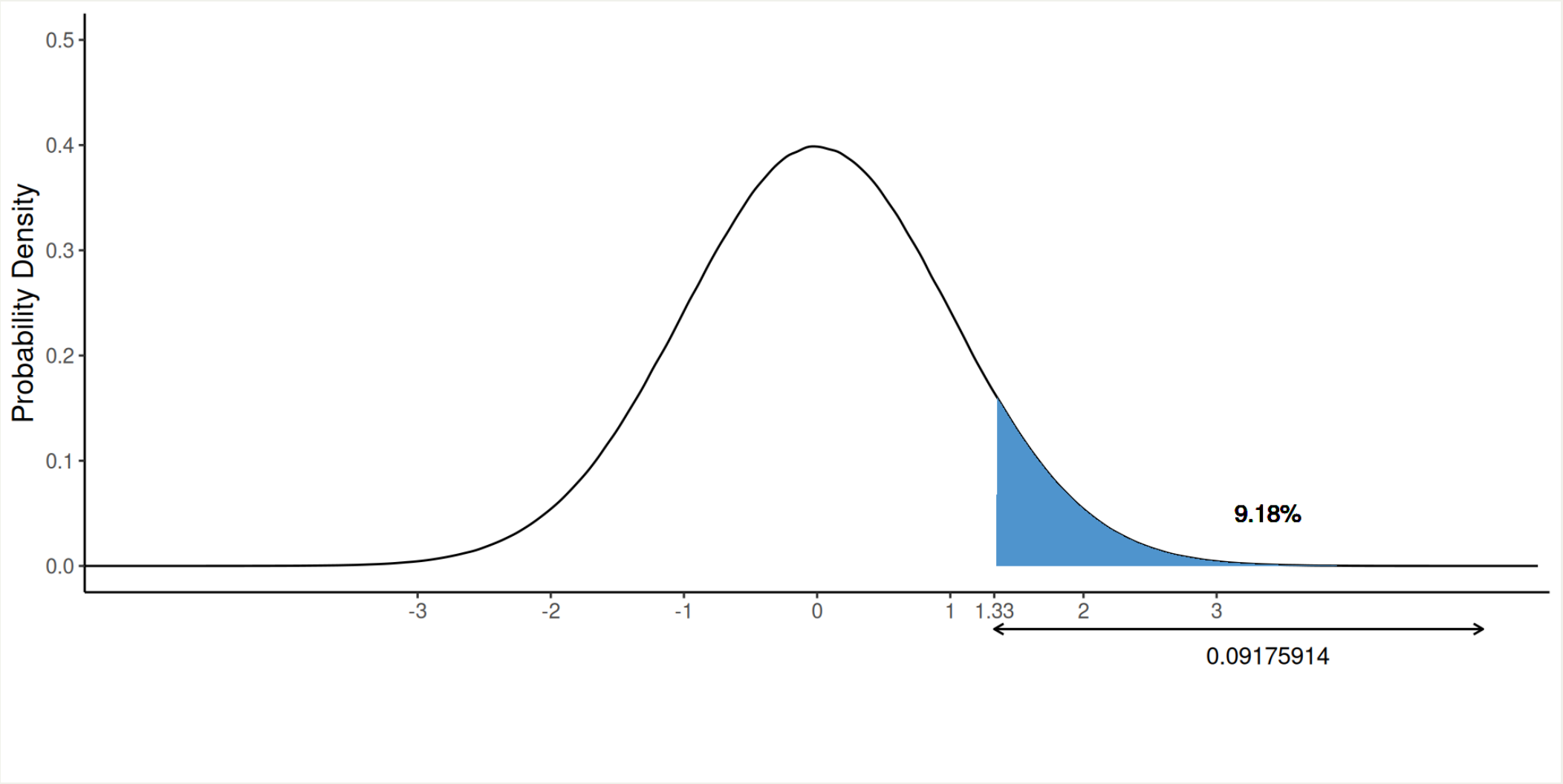
```
[1] 0.09175914
```

- R provides a 3rd way (with a helpful function argument).

```r
pnorm(1.33, lower.tail = FALSE)
```

```
[1] 0.09175914
```

Finding an Arbitrary Value



Back to Statistical Table

- Analogous to the first way.

Areas under the Normal Curve

Z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
-3.1	0.0010	0.0009	0.0009	0.0009	0.0008	0.0008	0.0008	0.0008	0.0007	0.0007
-3.0	0.0013	0.0013	0.0013	0.0012	0.0012	0.0011	0.0011	0.0011	0.0010	0.0010
-2.9	0.0019	0.0018	0.0018	0.0017	0.0016	0.0016	0.0015	0.0015	0.0014	0.0014
-2.8	0.0026	0.0025	0.0024	0.0023	0.0023	0.0022	0.0021	0.0021	0.0020	0.0019
-2.7	0.0035	0.0034	0.0033	0.0032	0.0031	0.0030	0.0029	0.0028	0.0027	0.0026
-2.6	0.0047	0.0045	0.0044	0.0043	0.0041	0.0040	0.0039	0.0038	0.0037	0.0036
-2.5	0.0062	0.0060	0.0059	0.0057	0.0055	0.0054	0.0052	0.0051	0.0049	0.0048
-2.4	0.0082	0.0080	0.0078	0.0075	0.0073	0.0071	0.0069	0.0068	0.0066	0.0064
-2.3	0.0107	0.0104	0.0102	0.0099	0.0096	0.0094	0.0091	0.0089	0.0087	0.0084
-2.2	0.0139	0.0136	0.0132	0.0129	0.0125	0.0122	0.0119	0.0116	0.0113	0.0110
-2.1	0.0179	0.0174	0.0170	0.0166	0.0162	0.0158	0.0154	0.0150	0.0146	0.0143
-2.0	0.0228	0.0222	0.0217	0.0212	0.0207	0.0202	0.0197	0.0192	0.0188	0.0183
-1.9	0.0287	0.0281	0.0274	0.0268	0.0262	0.0256	0.0250	0.0244	0.0239	0.0233
-1.8	0.0359	0.0351	0.0344	0.0336	0.0329	0.0322	0.0314	0.0307	0.0301	0.0294
-1.7	0.0446	0.0436	0.0427	0.0418	0.0409	0.0401	0.0392	0.0384	0.0375	0.0367
-1.6	0.0548	0.0537	0.0526	0.0516	0.0505	0.0495	0.0485	0.0475	0.0465	0.0455
-1.5	0.0668	0.0655	0.0643	0.0630	0.0618	0.0606	0.0594	0.0582	0.0571	0.0559
-1.4	0.0808	0.0793	0.0778	0.0764	0.0749	0.0735	0.0721	0.0708	0.0694	0.0681
-1.3	0.0968	0.0951	0.0934	0.0918	0.0901	0.0885	0.0869	0.0853	0.0838	0.0823
-1.2	0.1151	0.1131	0.1112	0.1093	0.1075	0.1056	0.1038	0.1020	0.1003	0.0985

Quantiles

- We already discussed one quantile - *median* (50% quantile).
- Quantile function can also be viewed as an inverse of CDF.
- In that we know the probability (percentage) and want to find out which value corresponds to it.
- Other than 50, other common values are 25%, 75% (*quartiles*).
- *Percentiles* break up the distribution into 100 groups (1%, 2%, …, 100%).
- *Quantiles* do not have to be whole numbers (e.g. 2.5% quantile).
- Functions that calculate quantiles in R start with `q`:

```
qnorm(p, mean = 0, sd = 1)
qchisq(p, df)
qexp(p)
qpois(p, lambda)
qt(p, df, ncp)
qunif(p)
```

Quantiles Examples

```r
# median of the standard normal distribution
qnorm(0.5)
```

```
[1] 0
```

```r
# 95% of such distribution falls between -1.96 * sigma
qnorm(0.025)
```

```
[1] -1.959964
```

```r
# and +1.96 * sigma
qnorm(0.975)
```

```
[1] 1.959964
```

Random Numbers

- Generating *random numbers* on a deterministic computer sounds like an oxymoron.
- What we in fact can generate are *pseudo-random numbers*, that for practical purposes behave as if they were random.
- They use some computer parameters that are almost random (time, load, process identifier)
- In R functions for random number generation (RNG) start with `r`:

```
rnorm(n, mean = 0, sd = 1)
rchisq(n, df)
rexp(n)
rpois(n, lambda)
rt(n, df, ncp)
runif(n)
```

RNG for Normal Distribution

```r
# Draw 10 pseudo-random numbers from standard normal distribution
rnorm(10)
```

```
[1]  0.05270152  1.53800577 -0.07023770 -0.23166383  1.06150143 -0.85755747
[7] -2.78842719 -1.00184274  0.69095688  1.58075134
```

```r
rnorm(10)
```

```
[1]  0.256239843  1.158994256 -0.136316800  1.529497541  0.808011988
[6]  0.251033913  1.552438351  0.981181616  0.005545037 -1.833525246
```

```r
# Change the parameters (mean and sd) of a normal distribution
rnorm(10, mean = 7, sd = 5)
```

```
[1] 14.737266  6.623805  3.248738  6.787346 12.444408  6.822736  5.675426
[8]  4.188826  8.197629  6.690349
```

Making Random Numbers Replicable

```r
# set.seed() function sets the initial state
# Makes random number generation replicable
set.seed(2023)
rnorm(10)
```

```
[1] -0.08378436 -0.98294375 -1.87506732 -0.18614466 -0.63348570  1.09079746
[7] -0.91372727  1.00163971 -0.39926660 -0.46812305
```

```r
set.seed(2023)
rnorm(10)
```

```
[1] -0.08378436 -0.98294375 -1.87506732 -0.18614466 -0.63348570  1.09079746
[7] -0.91372727  1.00163971 -0.39926660 -0.46812305
```

Next

- Tutorial:
  - Working with distributions, sampling
- Next week:
  - Data Frames