

# Setting up experiments

EXPERIMENTAL DESIGN IN PYTHON



James Chapman

Curriculum Manager, DataCamp

# Experimental Design definition

- A process
- An **objective** and **controlled** way
- Draw **specific conclusions**
  - In reference to a **hypothesis**



<sup>1</sup> <https://www.sciencedirect.com/topics/earth-and-planetary-sciences/experimental-design>

# Forming robust statements

X probably had an effect on Y. There is likely some small risk of error

- Precise and Quantified language

P-value analysis indicates X had an effect on Y with a 10% risk of Type I error

- Type I error: incorrectly reject null hypothesis

- Goal: Experimental design and statistical analysis

# Why experimental design?

Useful in many fields:

- Medical research
- Marketing
- Product analytics
- Agriculture
- Government policy



# Some terminology...

- **Subjects** = *what* we are experimenting on  
(people, employees, users, etc.)



= Subject

# Some terminology...

- **Subjects** = *what* we are experimenting on (people, employees, users, etc.)
- **Treatment** = some change given to one group



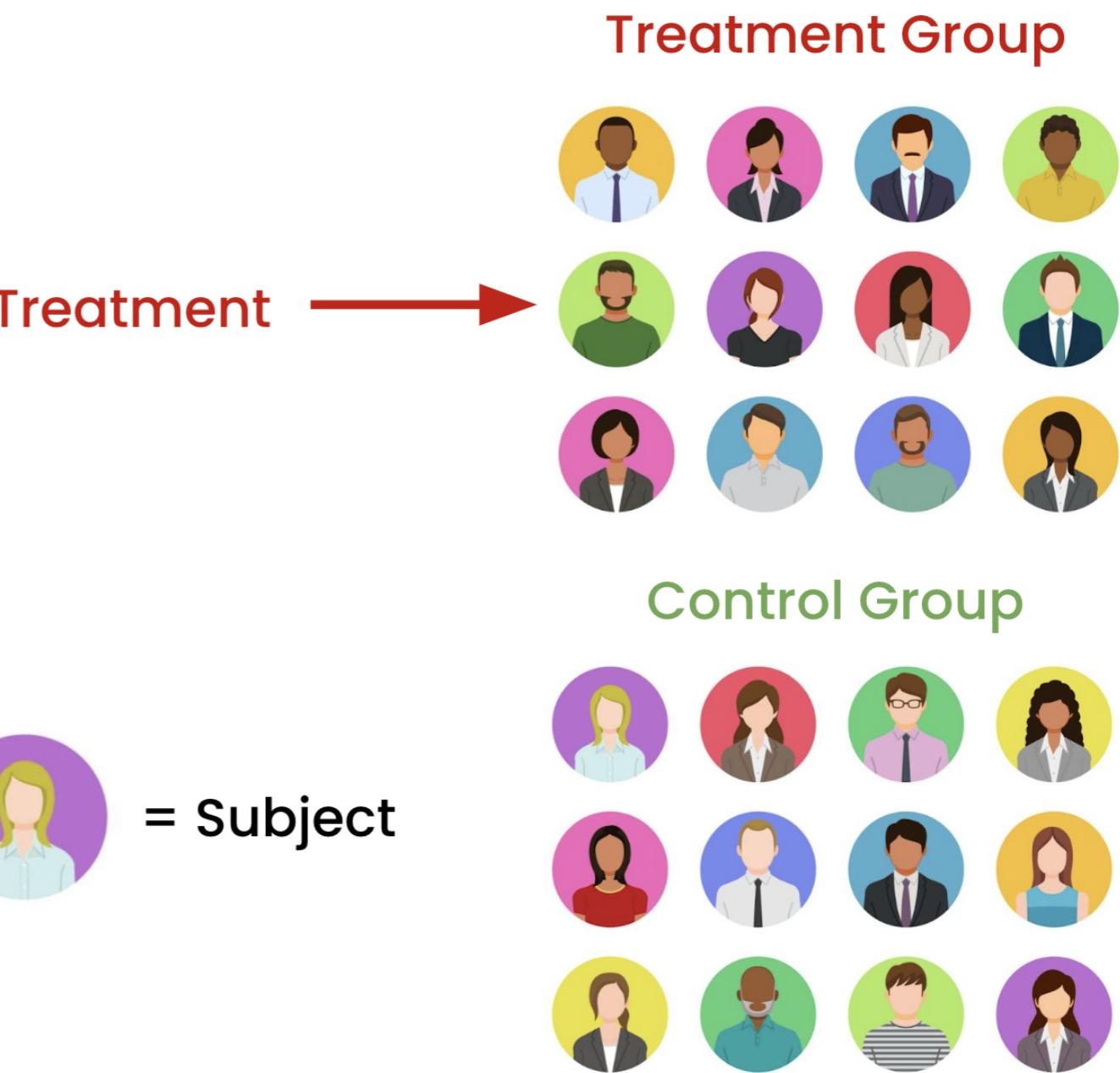
# Some terminology...

- **Subjects** = *what* we are experimenting on (people, employees, users, etc.)
- **Treatment** = some change given to one group



# Some terminology...

- **Subjects** = *what* we are experimenting on (people, employees, users, etc.)
- **Treatment** = some change given to one group
- **Control** = the group not given any change



# Assigning subjects to groups

- How to assign subjects to groups?
  - Option 1 - *non-random* ('split' the DataFrame)
  - Option 2 - *random assignment*
- Example: 200 subjects heights in `heights` DataFrame

	<code>id</code>	<code>height</code>
0	0	177.98
1	1	174.17
2	2	178.89

# Non-random assignment

## Assignment by slicing the DataFrame

```
group1_nonrandom = heights.iloc[0:100,:]  
group2_nonrandom = heights.iloc[100:,:]  
compare_df = pd.concat(  
    [group1_nonrandom['height'].describe(),  
     group2_nonrandom['height'].describe()],  
    axis=1)  
compare_df.columns = ['group1', 'group2']  
print(compare_df)
```

	group1	group2
count	100.00	100.00
mean	170.32	179.19 <--
std	3.28	3.50
min	159.28	175.03
25%	168.06	176.57
50%	170.75	178.03
75%	173.09	180.79
max	174.92	191.32

- Very different! (Mean 9cm apart)

# Random assignment

- Use `.sample()`
  - `n` or `frac` (fraction 0-1)

```
group1 = heights.sample(frac=0.5,  
                        replace=False,  
                        random_state=42)
```

```
group2 = heights.drop(group1.index)
```

```
print(compare_df)
```

- Much closer! (<1cm)

	group1	group2
count	100.00	100.00
mean	175.10	174.41 $\leftarrow$
std	5.39	5.78
min	163.07	159.28
25%	171.32	170.17
50%	175.22	174.86
75%	178.32	177.85
max	189.78	191.32

# Assignment summary

- Subjects should be *randomly* assigned to groups
  - Observed changes correctly attributed
- Random subject assignment: `.sample()`
- Verify differences: `.describe()`

Treatment Group



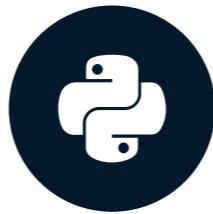
Control Group

# **Let's practice!**

**EXPERIMENTAL DESIGN IN PYTHON**

# Experimental data setup

EXPERIMENTAL DESIGN IN PYTHON

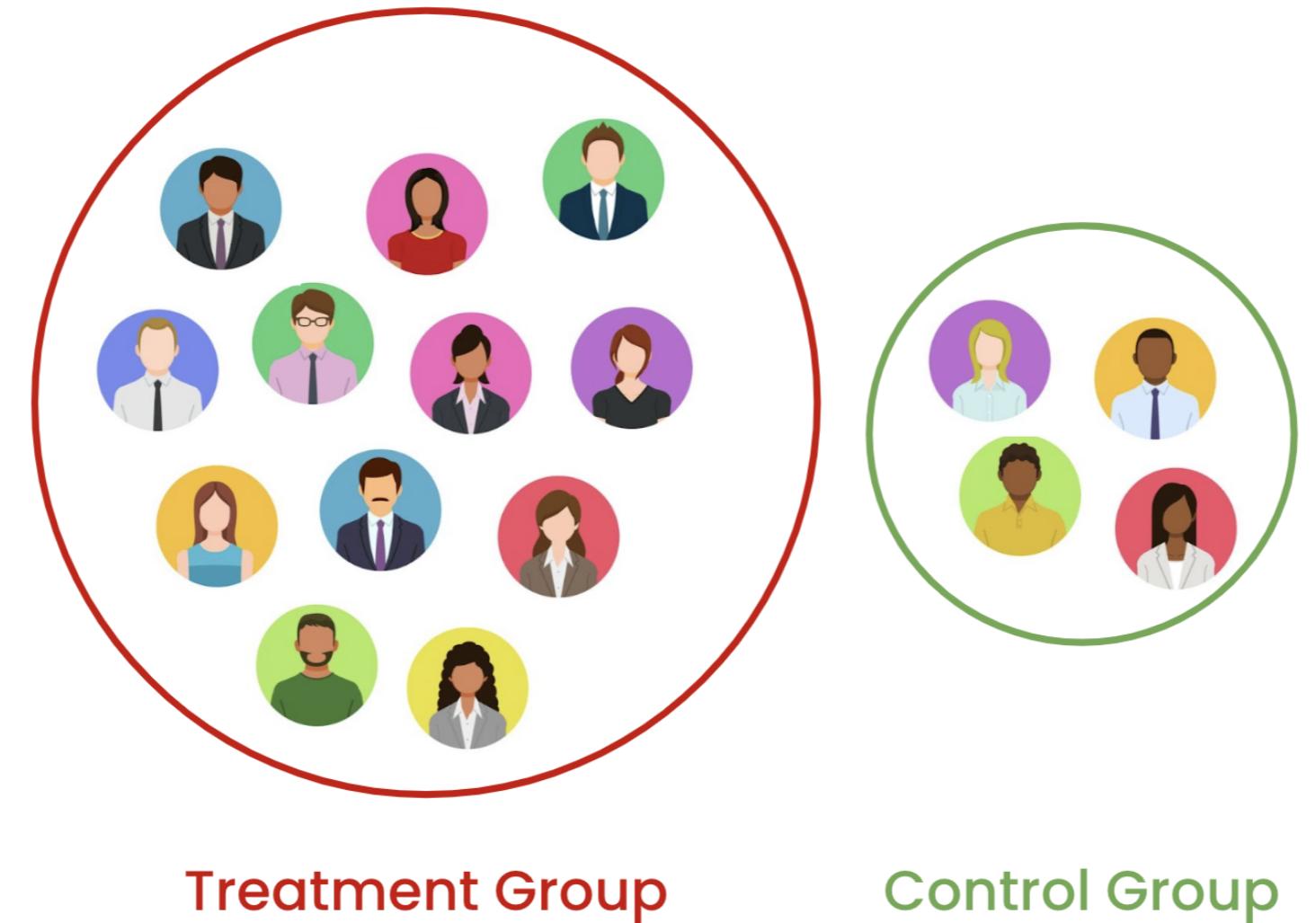


James Chapman

Curriculum Manager, DataCamp

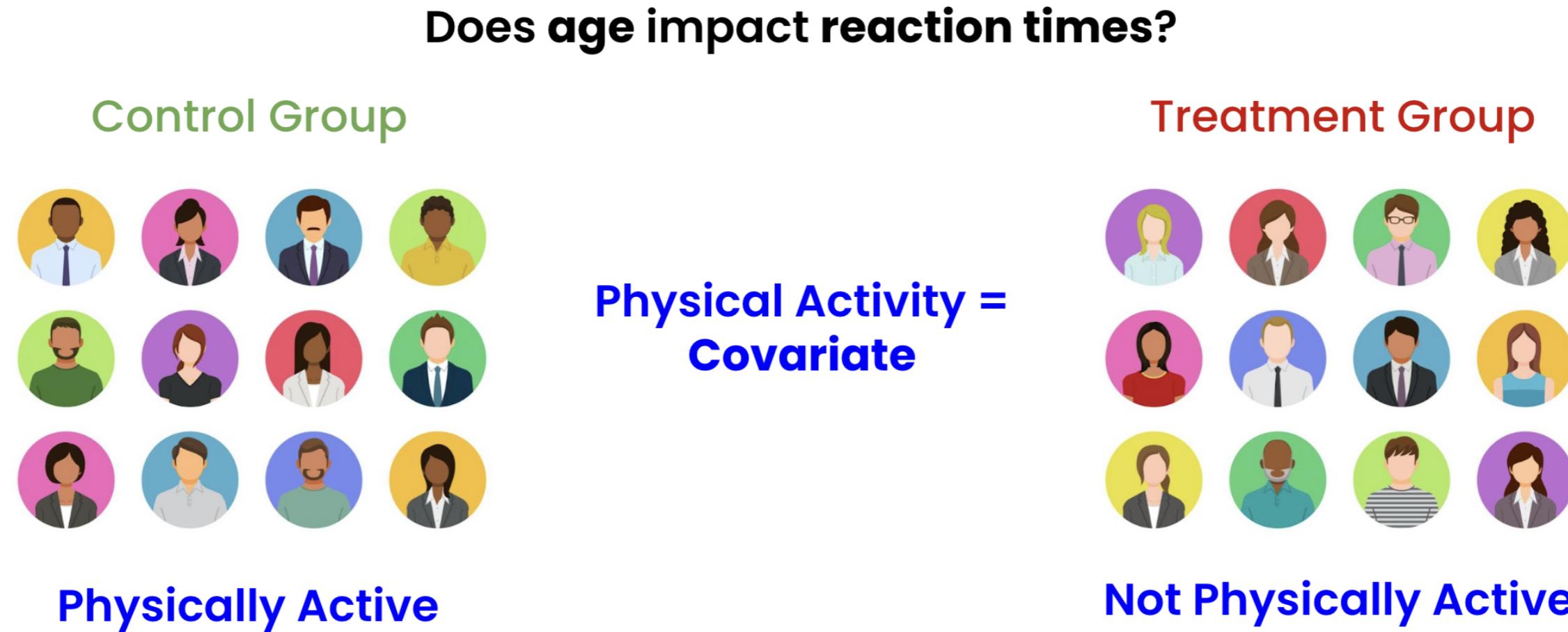
# The problem with randomization

1) Uneven issue: different number of subjects in groups



# The problem with randomization

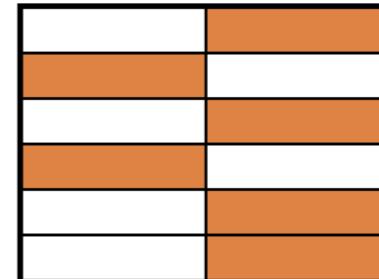
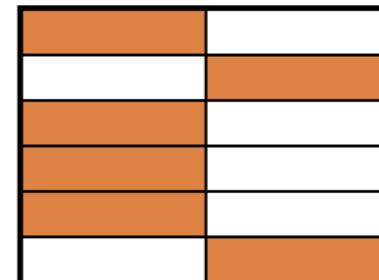
2) Covariate issue: High variability in some covariates → group imbalances in randomization



**Result:** harder to measure treatment effect!

# Block randomization

- Split into blocks of size  $n$  first then randomly split
  - Fixes uneven issue



- 24 subjects split into two groups then randomized

# Our dataset

- E-commerce dataset ( ecom ) (1000 subjects)
  - Average basket size
  - Average time on the website
  - Power user (Average 40+ daily minutes on website)

	basket_size	web_time	power_user
0	227	7	0
1	123	5	0
2	98	16	0
3	211	45	1
4	133	17	0

# Block randomization in Python

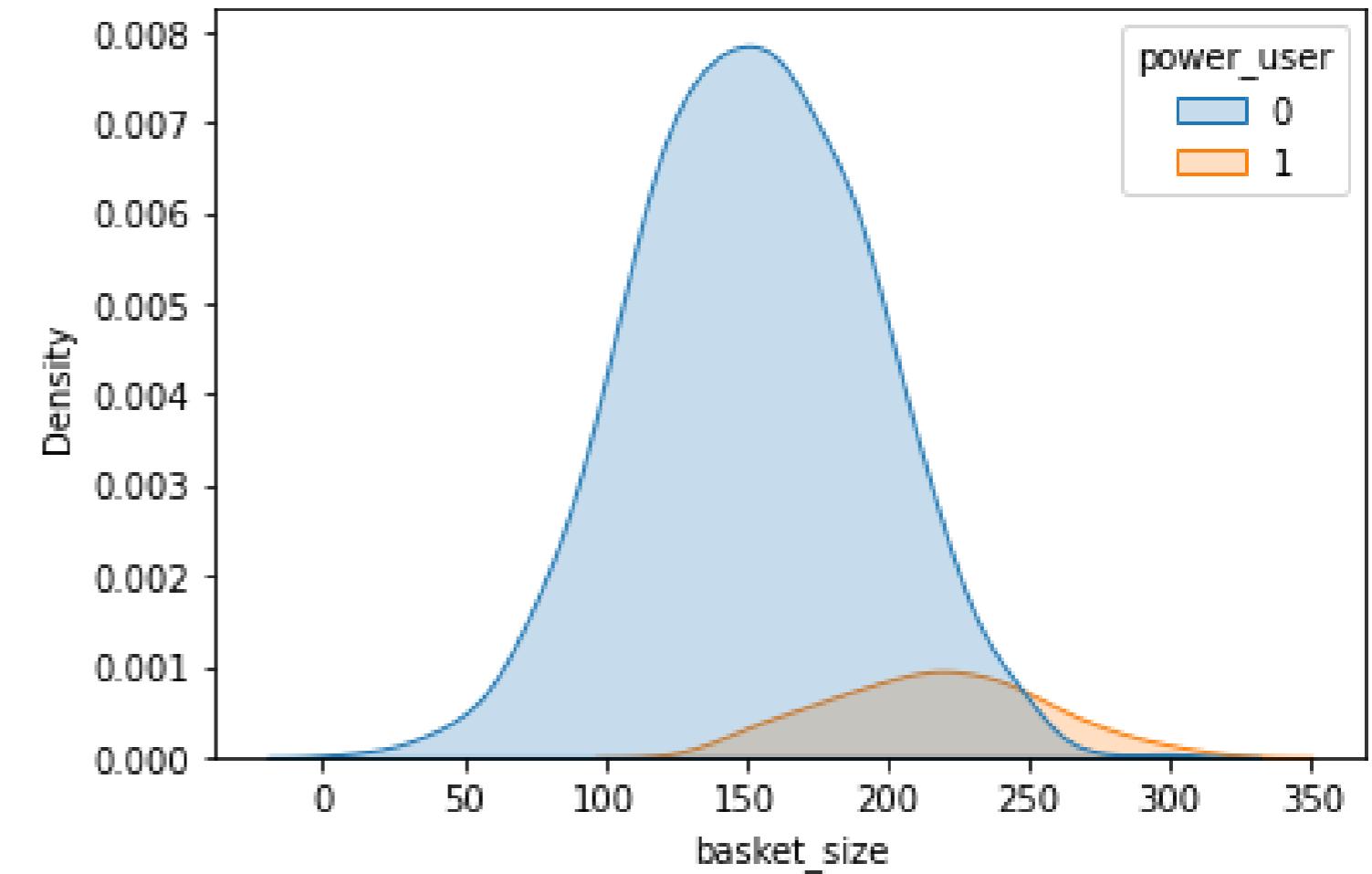
```
group1 = ecom.sample(frac=0.5, random_state=42, replace=False)  
group1['Block'] = 1  
group2 = ecom.drop(group1.index)  
group2['Block'] = 2  
print(len(group1), len(group2))
```

500,500

# Visualizing splits

```
import seaborn as sns  
import matplotlib.pyplot as plt  
sns.displot(data=ecom,  
             x='basket_size',  
             hue='power_user',  
             fill=True,  
             kind='kde')  
plt.show()
```

Confounding = variable *might* cause the effect rather than treatment



# Stratified randomization

- Splitting based on covariate(s) *first*
  - Then randomization
- Green = All power users (Yellow = Not power users)
  - Then split Treatment/Control
- Fixes covariate/confounding issue
- Can be done for multiple covariates - but gets complex!

**Not Power User**

T	T
C	T
C	T
T	T
C	C
C	C

T	C
C	T
T	C
T	C

**Power Users**

# Our first strata

- Separate power users
- Sample into Treatment/Control

```
strata_1 = ecom[ecom['power_user'] == 1]
strata_1['Block'] = 1
strata_1_g1 = strata_1.sample(frac=0.5, replace=False)
strata_1_g1['T_C'] = 'T'
strata_1_g2 = strata_1.drop(strata_1_g1.index)
strata_1_g2['T_C'] = 'C'
```

# The second strata

- Separate not power users
- Sample into Treatment/Control

```
strata_2 = ecom.drop(strata_1.index)
strata_2['Block'] = 2
strata_2_g1 = strata_2.sample(frac=0.5, replace=False)
strata_2_g1['T_C'] = 'T'
strata_2_g2 = strata_2.drop(strata_2_g1.index)
strata_2_g2['T_C'] = 'C'
```

# Confirming stratification

- Join blocks and groups
- Use groupby to check allocation

```
ecom_stratified = pd.concat([strata_1_g1, strata_1_g2, strata_2_g1, strata_2_g2])
ecom_stratified.groupby(['Block', 'T_C', 'power_user']).size()
```

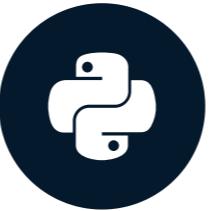
Block	T_C	power_user	
1	C	1	50
	T	1	50
2	C	0	450
	T	0	450

# **Let's practice!**

**EXPERIMENTAL DESIGN IN PYTHON**

# Normal data

EXPERIMENTAL DESIGN IN PYTHON



James Chapman

Curriculum Manager, DataCamp

# The normal distribution

- The familiar 'bell curve' shape

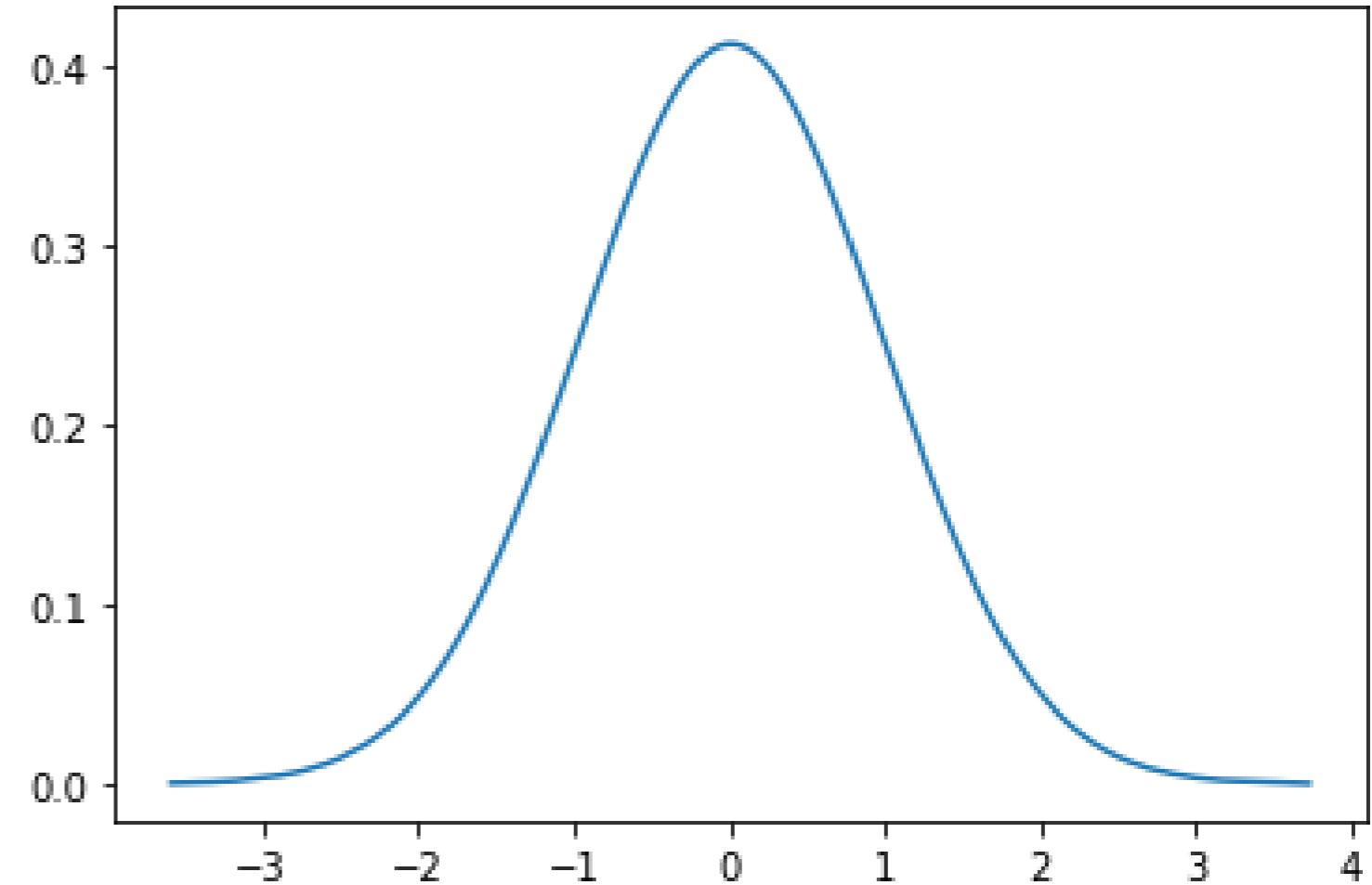
- Related to z-score work

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

- Mean = 0, std = 1

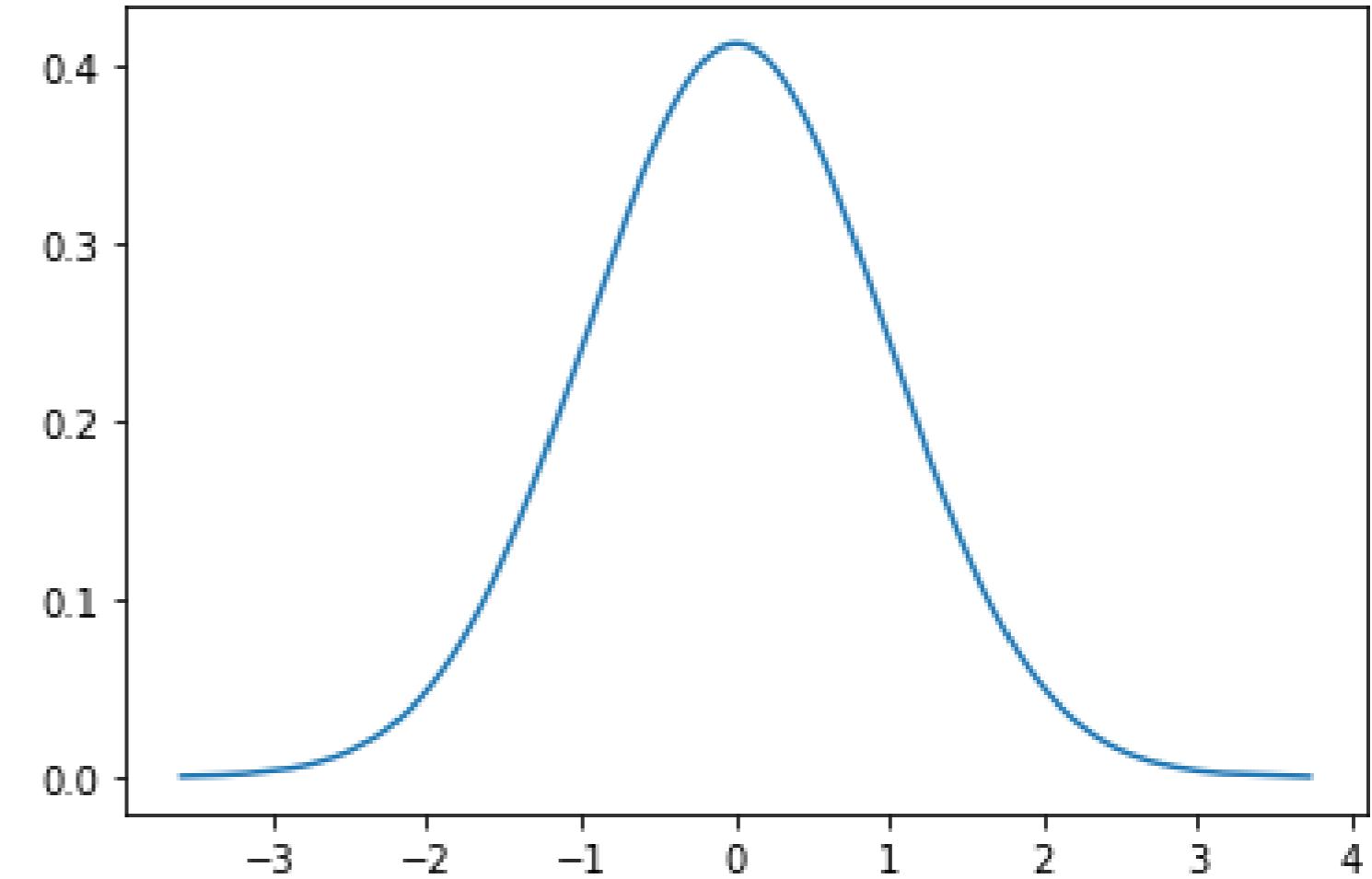
- 'How many standard deviations is this point from the mean?'

- 'What is the probability of obtaining this score?'



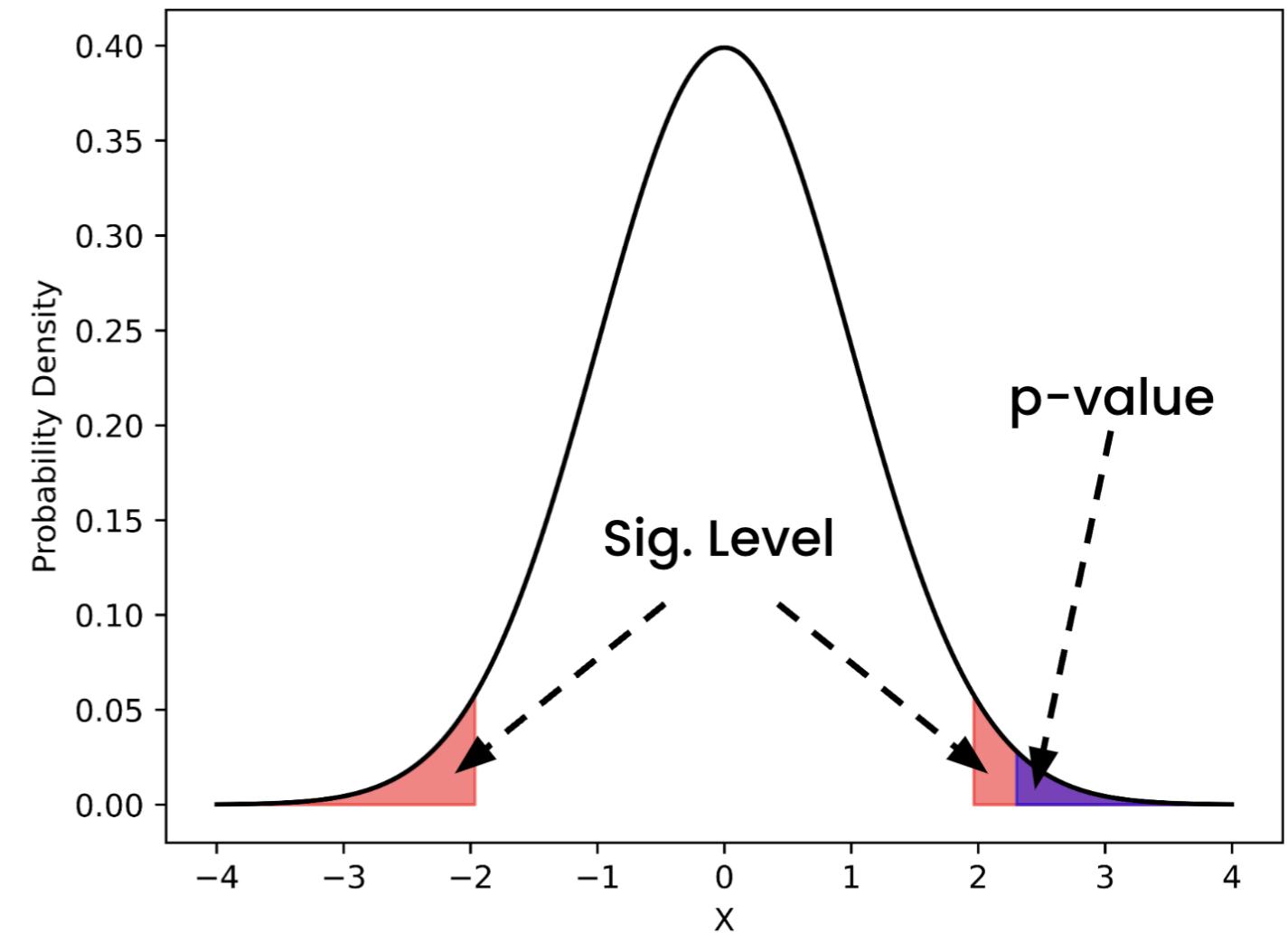
# Normal data and statistical tests

- Required for **parametric** tests
- **Nonparametric tests:** don't assume normal data



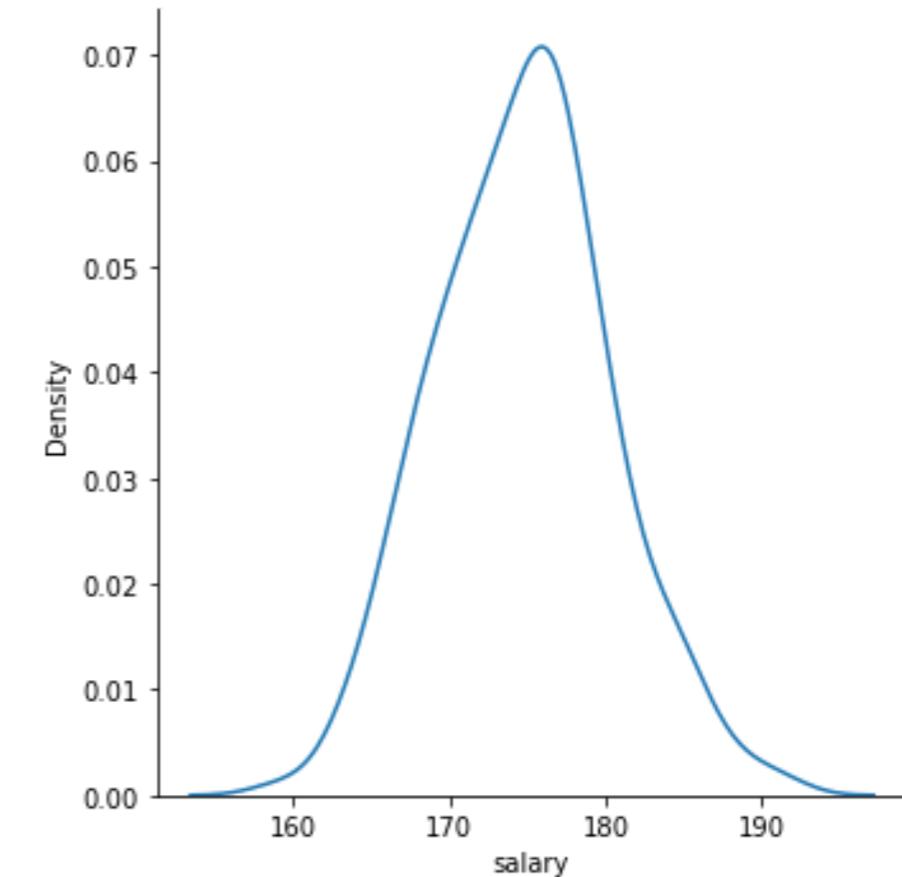
# Normal, Z, and alpha

- Crucial link to significance level ( $\alpha$ )
- Compare p-value to  $\alpha$
- Probability of a Type I error



# Visualizing normal data

```
sns.displot(data=salaries,  
            x='salary',  
            kind="kde")  
  
plt.show()
```

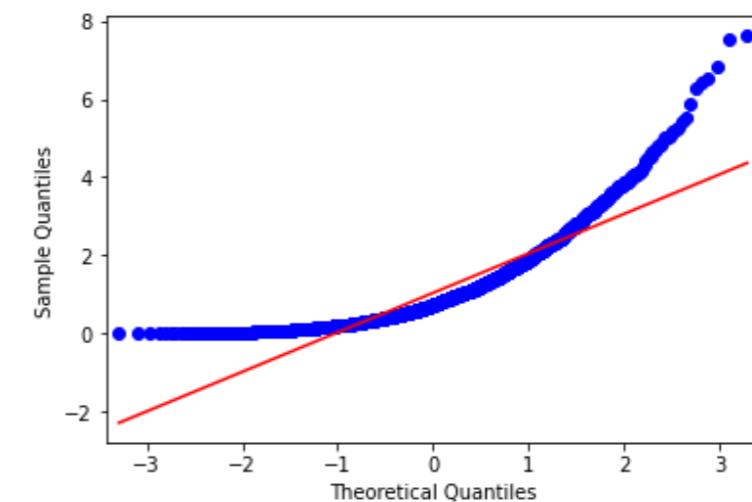
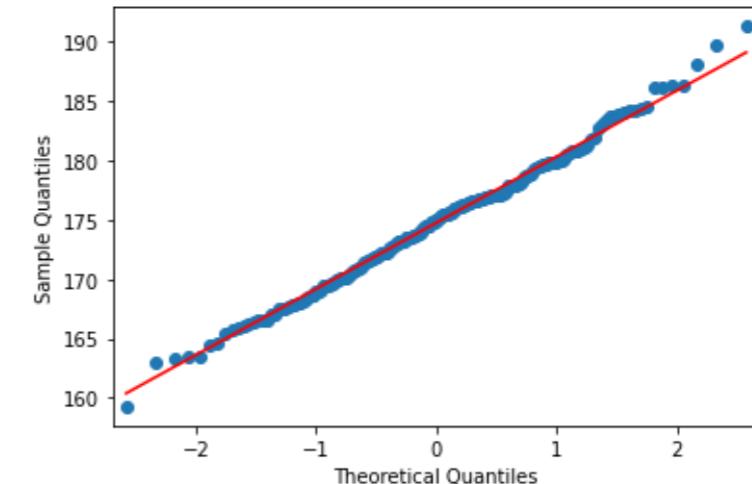


# QQ plots

QQ plot: compare data to a particular distribution

```
from statsmodels.graphics.gofplots import qqplot  
from scipy.stats.distributions import norm  
  
qqplot(salaries['salary'],  
       line='s',  
       dist=norm)  
  
plt.show()
```

- Ideal: dots hugging line
- Bad: bow out at ends



# Tests for normality

- Shapiro-Wilk (good for smaller datasets)
- D'Agostino  $K^2$  (uses kurtosis and skewness)
- Anderson-Darling (returns list of values)

$H_0$  = "Data is drawn from a Normal Distribution"

# A Shapiro-Wilk test

```
from scipy.stats import shapiro  
alpha = 0.05  
stat, p = shapiro(salaries['salary'])  
print(f"p: {round(p, 4)} test stat: {round(stat, 4)}")
```

```
p: 0.8293 test stat: 0.9956
```

- `p > alpha`
  - Fail to reject  $H_0 \rightarrow$  likely normal

# An Anderson-Darling test

```
from scipy.stats import anderson  
result = anderson(x=salaries['salary'], dist="norm")
```

```
print(round(result.statistic,4))  
print(result.significance_level)  
print(result.critical_values)
```

```
0.2748  
[15.  10.   5.   2.5  1.]  
[0.572 0.651 0.781 0.911 1.084]
```

- $0.2748 < [0.572 \ 0.651 \ 0.781 \ 0.911 \ 1.084]$ 
  - Fail to reject  $H_0 \rightarrow$  likely normal

# **Let's practice!**

**EXPERIMENTAL DESIGN IN PYTHON**