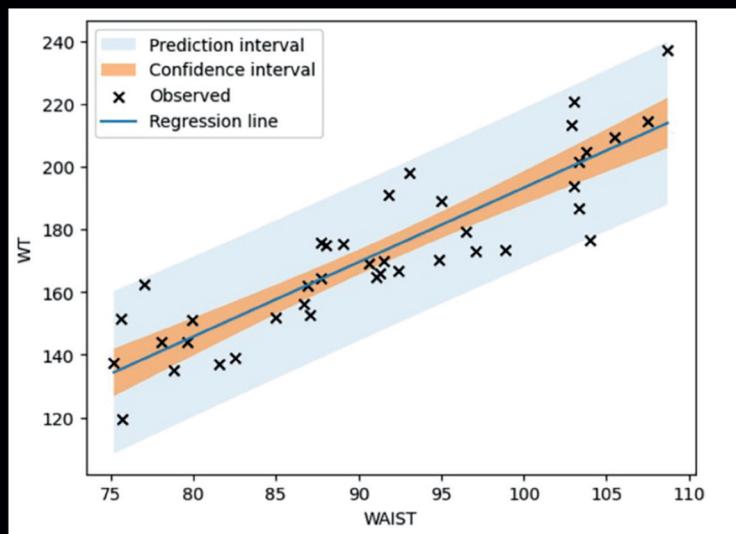


# Applied Statistics with Python

Volume I: Introductory Statistics and Regression



LEON KAGANOVSKIY

A Chapman & Hall Book

 CRC Press  
Taylor & Francis Group

# Applied Statistics with Python

*Applied Statistics with Python: Volume I: Introductory Statistics and Regression* concentrates on applied and computational aspects of statistics, focusing on conceptual understanding and Python-based calculations. Based on years of experience teaching introductory and intermediate Statistics courses at Touro University and Brooklyn College, this book compiles multiple aspects of applied statistics, teaching the reader useful skills in statistics and computational science with a focus on conceptual understanding. This book does not require previous experience with statistics and Python, explaining the basic concepts before developing them into more advanced methods from scratch. *Applied Statistics with Python* is intended for undergraduate students in business, economics, biology, social sciences, and natural science, while also being useful as a supplementary text for more advanced students.

## Key Features:

- Concentrates on more introductory topics such as descriptive statistics, probability, probability distributions, proportion and means hypothesis testing, as well as one-variable regression.
- The book's computational (Python) approach allows us to study Statistics much more effectively. It removes the tedium of hand/calculator computations and enables one to study more advanced topics.
- Standardized sklearn Python package gives efficient access to machine learning topics.
- Randomized homework as well as exams are provided in the author's course shell on My Open Math web portal (free).

**Leon Kaganovskiy** is an Associate Professor at the Mathematics Department of Touro University. He received a M.S. in Theoretical Physics from Kharkov State University, and M.S. and Ph.D. in Applied Mathematics from the University of Michigan. His most recent interest is in a broad field of Applied Statistics, and he has developed new courses in Bio-Statistics with R, Statistics for Actuaries with R, and Business Analytics with R. In addition, he teaches Statistics research courses at the Graduate Program in Speech-Language Pathology at Touro College.



Taylor & Francis  
Taylor & Francis Group  
<http://taylorandfrancis.com>

# Applied Statistics with Python

Volume I: Introductory Statistics and  
Regression

Leon Kaganovskiy



CRC Press

Taylor & Francis Group

Boca Raton London New York

---

CRC Press is an imprint of the  
Taylor & Francis Group, an **informa** business  
A CHAPMAN & HALL BOOK

Designed cover image: © Leon Kaganovskiy

First edition published 2025  
by CRC Press  
2385 NW Executive Center Drive, Suite 320, Boca Raton FL 33431

and by CRC Press  
4 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

*CRC Press is an imprint of Taylor & Francis Group, LLC*

© 2025 Leon Kaganovskiy

Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, access [www.copyright.com](http://www.copyright.com) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. For works that are not available on CCC please contact [mpkbookspermissions@tandf.co.uk](mailto:mpkbookspermissions@tandf.co.uk)

*Trademark notice:* Product or corporate names may be trademarks or registered trademarks and are used only for identification and explanation without intent to infringe.

ISBN: 978-1-032-75193-1 (hbk)  
ISBN: 978-1-032-75263-1 (pbk)  
ISBN: 978-1-003-47311-4 (ebk)

DOI: [10.1201/9781003473114](https://doi.org/10.1201/9781003473114)

Typeset in Latin Modern font  
by KnowledgeWorks Global Ltd.

*Publisher's note:* This book has been prepared from camera-ready copy provided by the authors.

---

# *Contents*

---

<b>Preface</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Introduction to Statistics . . . . .	1
1.2 Data Introduction . . . . .	2
1.3 Sampling Methods . . . . .	7
1.4 Statistical Experiments . . . . .	8
1.5 Roadmap for the Book . . . . .	9
<b>2 Descriptive Data Analysis</b>	<b>11</b>
2.1 Numerical Data . . . . .	11
2.1.1 Sample and Population Means . . . . .	11
2.1.2 Numerical Distribution Shape . . . . .	14
2.1.3 Variance and Standard Deviation . . . . .	18
2.1.4 Median, Quartiles, and Boxplot . . . . .	22
2.2 Categorical Data . . . . .	26
2.2.1 Tables and Barplots . . . . .	26
2.2.2 Contingency Tables . . . . .	28
2.2.3 Comparing numerical data across groups . . . . .	30
<b>3 Probability</b>	<b>35</b>
3.1 Basic Concepts of Probability . . . . .	35
3.2 Addition Rule . . . . .	37
3.3 Simple Finite Probability Distributions . . . . .	38
3.4 Complement of an Event . . . . .	41
3.5 Independence . . . . .	42
3.6 Conditional Probability . . . . .	45
3.7 Contingency Tables Probability . . . . .	46
3.8 General Multiplication Rule and Tree Diagrams . . . . .	56
3.9 Expected Value of Probability Distribution . . . . .	62
<b>4 Probability Distributions</b>	<b>65</b>
4.1 Probability Distributions . . . . .	65
4.2 Normal Distribution . . . . .	67
4.2.1 Normal Distribution Model . . . . .	67
4.2.2 Normal Probability Calculations . . . . .	70

4.2.3	Central Limit Theorem . . . . .	74
4.2.4	Normal Distribution Examples . . . . .	74
4.2.5	68-95-99.7 Rule . . . . .	83
4.3	Binomial Distribution . . . . .	86
4.3.1	Permutations and Combinations . . . . .	87
4.3.2	Binomial Formula . . . . .	89
4.3.3	Normal Approximation of the Binomial Distribution . . . . .	95
<b>5</b>	<b>Inferential Statistics and Tests for Proportions</b>	<b>99</b>
5.1	Introduction to Inference . . . . .	99
5.2	Confidence Interval for Proportions . . . . .	104
5.2.1	General Confidence Intervals . . . . .	107
5.2.2	More Examples of Proportion Confidence Intervals . . . . .	110
5.3	Hypothesis Testing for Proportions . . . . .	112
5.3.1	More Proportion Hypothesis Tests Examples . . . . .	116
5.3.2	Decision Errors and Choosing a Significance (Error) Level . . . . .	126
5.3.3	Sample Size for Proportion . . . . .	128
5.3.4	One-Proportion Tests for Data Files . . . . .	131
5.4	Difference of Two Proportions . . . . .	134
<b>6</b>	<b>Goodness of Fit and Contingency Tables</b>	<b>143</b>
6.1	Goodness of Fit . . . . .	143
6.2	Chi-Squared Test of Independence in a Two-Way Table . . . . .	157
<b>7</b>	<b>Inference for Means</b>	<b>171</b>
7.1	One-Sample Mean Tests . . . . .	171
7.1.1	T-distribution . . . . .	178
7.1.2	Confidence Interval for One-Sample Mean . . . . .	181
7.1.3	One-Sample Means t-tests . . . . .	184
7.1.4	One-Sample Means Tests on Data Files . . . . .	195
7.1.5	Choosing a Sample Size When Estimating a Mean . . . . .	204
7.1.6	Effect Size and Power . . . . .	206
7.2	Means Tests for Paired Data Samples . . . . .	210
7.3	Means Tests for Two Independent Samples . . . . .	220
<b>8</b>	<b>Correlation and Regression</b>	<b>237</b>
8.1	Correlation . . . . .	237
8.1.1	Spearman Correlation . . . . .	245
8.1.2	Partial Correlation . . . . .	246
8.2	Least Squares Regression Line . . . . .	248
8.2.1	Assumptions for the Least Squares Model . . . . .	251
8.2.2	Statistical Analysis of Regression Coefficients . . . . .	254
8.2.3	Leverage and Influence . . . . .	258
8.2.4	Regression ANOVA . . . . .	261
8.2.5	Linear Model Prediction . . . . .	263

8.3 Linear Model Examples . . . . .	267
8.4 Linearized Models - Exponential and Logarithmic Transformations . . . . .	282
8.4.1 Exponential Regression . . . . .	282
8.4.2 Logarithmic Regression . . . . .	287
8.4.3 S-Shaped (Logistic) Curve . . . . .	291
8.5 Categorical Predictors . . . . .	296
<b>Bibliography</b>	<b>305</b>
<b>Index</b>	<b>307</b>



Taylor & Francis  
Taylor & Francis Group  
<http://taylorandfrancis.com>

---

## **Preface**

---

This textbook is based on years of experience teaching introductory and intermediate Statistics courses at Touro University and Brooklyn College. Similar to other authors, I became frustrated with splicing available textbook materials and wanted to create my own textbook that covers the topics the way that I like and teaches the skills I believe are the most important for the students. In this textbook, I focus on Applied Statistics leaving Mathematical derivations for the appropriate Mathematical Statistics courses and instead concentrating on the conceptual understanding of the subject. In Applied Statistics, we naturally, avoid any long-hand calculations and teach students useful computational science skills. The audience includes undergraduate students in business, economics, biology, social sciences, and natural science, but other more advanced students could benefit from it as well. It was determined in collaboration with the publisher's production team that it is best to split the large textbook into two more manageable parts. This book is Volume I of Applied Statistics with Python. It concentrates on more introductory topics such as descriptive statistics, probability, probability distributions, proportion and means hypothesis testing, as well as one-variable regression. Volume II of the book will look at ANOVA, multiple regression, model selection, lasso, ridge, logistic regression, KNN, support vector classifiers, non-linear models, tree models, clustering, etc.

Oftentimes, Statistics is students' least favorite subject. I would dare to say that some of this could be blamed on the teaching style that is often either too Mathematical and/or too reliant on long, old-style hand calculations with minimal use of a scientific calculator at best and heavy use of last-century tables. No one does actual Statistics like that anymore; so why do we teach students outdated, impractical, and time-consuming skills? It is like teaching logs with a log ruler. I strongly believe that we should concentrate on conceptual understanding and applicability of statistical tests and use the appropriate computational tools to actually compute. There are also a number of newer computational methods essential for bigger data files like regularization methods (lasso, ridge, etc.), cross-validation, tree structures, ensemble methods, etc. All of these require a mature software implementation and cannot be done in Excel and the like.

There are dozens of different texts in introductory and intermediate Statistics. A number of them introduce computational software like SPSS, STATA,

Minitab, R, etc., but it is usually done separately in computer labs and only for some problems based on data files [15, 13, 5, 14, 12, 18, 3, 16, 2, 7]. On the other hand, there are a number of software-based Statistics textbooks which assumes that students have already taken a Statistics course [8, 17, 4, 9, 1, 6, 19, 10, 11, 20]. Some of them use Python, but most use R. In this book, I don't assume previous experience in Statistics. We learn the basic concepts and develop more advanced methods from scratch. I use Python as a computational tool at every step because of its amazing versatility and flexibility. It is a free programming language that is used by beginners and advanced practitioners alike with many free add-on libraries for Statistics and its applications. It is a scripting language that allows turning otherwise tedious hand calculations into step-by-step functions where students can see the results of every computation to obtain information from the data in real-time.

The main difference between this textbook and others is weaving code into the text at every step in a clear and accessible way to give students a hands-on computational tool that can benefit them in this and many other courses. The homework for each chapter and exams are provided in my course shell on My Open Math web portal. It is an amazing free platform to teach Mathematics and Statistics. All problems are randomized. I provide scripting files for each chapter with clear instructions for each problem, and students have remarked that they liked the course a lot compared to their peers who had to deal with old-style approaches to Statistics.

# 1

---

## Introduction

---

### 1.1 Introduction to Statistics

Statistics is the science of working with data collection (-polls, surveys, experiments, etc.) and subsequent data analysis and interpretation. It has become an integral part of almost every subject from Business and Economics to Medicine, Biology, Engineering, etc.

Consider a typical example from Medicine. A new cholesterol-reducing drug is invented and its efficacy needs to be tested against a standard drug used in medical practice. In the study of 200 patients who had high cholesterol, each one was **randomly** assigned to one of the two groups:

**Treatment group** - 98 patients treated with the new drug.

**Control group** - 102 patients treated with the standard drug.

The results were assessed after a month. For the treatment group,  $77/98 = 0.786 = 78.6\%$  had more than 10 points reduction in their cholesterol levels. In the control group,  $75/102 = 0.735 = 73.5\%$  had more than 10 points reduction in their cholesterol levels.

The treatment group had about 5% better chance of having a sizeable cholesterol reduction. The main question of Statistics is if it happened by chance or if it is a **statistically significant** result. There are always **natural variations (fluctuations)** in any data - a perfectly fair coin flipped 200 times does not land on 100 heads and 100 tails. Each such trial would come out slightly differently. Then the question becomes, how likely is it to obtain a 5% difference if the new drug and old drug perform about the same? We have to wait until a later chapter to develop computational tools to answer this question, but intuitively, the larger the difference, the less probable it is to obtain it by chance.

## 1.2 Data Introduction

The files have to be in the right data format for Statistical software to read and analyze. Two standard files will be used extensively: HELPrct (medical data on substance abusers) and CPS85 (-economics data on hourly wages). We show the summary of a portion of the first file below. The lines of Python code are commented out with text starting with a hashtag `#`. These are explanations for us humans to understand what is going on. Python approaches Statistics and Data Science by introducing many libraries like pandas imported below. Pandas library has the command `read_csv()` that reads in data files in comma-separated-values (.csv) format, which is used below. The `.shape` command shows that there are 453 substance abusers in the HELPrct file with 31 columns describing them. Here, we show only a very small portion of it selecting variables (columns) of interest with double brackets `mydata[['cesd',...]]`.

```
[ ]: import pandas as pd # pandas library works with data frames in Python
url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrct.csv"
mydata = pd.read_csv(url) # save as mydata file
print('Data Frame dimensions: ',mydata.shape) # #rows and # cols

mydata = mydata[['cesd','age','mcs','pcs','homeless','substance','sex']];
mydata # NOTE that Python counts from row 0!!!
```

Data Frame dimensions: (453, 31)

	cesd	age	mcs	pcs	homeless	substance	sex
0	49	37	25.111990	58.413689	housed	cocaine	male
1	30	37	26.670307	36.036942	homeless	alcohol	male
2	39	26	6.762923	74.806328	housed	heroin	male
3	15	39	43.967880	61.931679	housed	heroin	female
4	39	32	21.675755	37.345585	homeless	cocaine	male
...	...	...	...	...	...	...	...
448	28	33	41.943066	56.968681	housed	heroin	male
449	37	49	62.175503	57.253838	housed	alcohol	male
450	28	39	33.434536	40.045715	homeless	heroin	female
451	11	59	54.424816	53.732044	homeless	cocaine	male
452	35	45	30.212227	43.476074	homeless	alcohol	male

[453 rows x 7 columns]

The file above has a **data frame (data matrix)** form. Each row (case) is a patient, and each column is a variable describing that patient. Variables have different types. The `age`, `cesd` (depression score), `mcs` (mental health score), and `pcs` (physical health score) are all **numerical (quantitative) variables**. The `homeless`, `sex`, and `substance` are all **categorical (qualitative) variables** (labels).

The CPS85.csv data set on wages is shown below:

```
[2]: import pandas as pd
url="https://raw.githubusercontent.com/leonkag/Statistics0/main/CPS85.csv"
mydata = pd.read_csv(url) # save as mydata file
print('Data Frame dimensions: ',mydata.shape) # shape #rows and # cols
mydata = mydata[['wage','educ','race','sex','south','married','exper','sector']];
mydata.head(10)
```

Data Frame dimensions: (534, 12)

	wage	educ	race	sex	south	married	exper	sector
0	9.00	10	W	M	NS	Married	27	const
1	5.50	12	W	M	NS	Married	20	sales
2	3.80	12	W	F	NS	Single	4	sales
3	10.50	12	W	F	NS	Married	29	clerical
4	15.00	12	W	M	NS	Married	40	const
5	9.00	16	W	F	NS	Married	27	clerical
6	9.57	12	W	F	NS	Married	5	service
7	15.00	14	W	M	NS	Single	22	sales
8	11.00	8	W	M	NS	Married	42	manuf
9	5.00	12	W	F	NS	Married	14	sales

Variables `wage`, `age`, `educ`, and `exper` are numerical, while `race`, `sex`, `south`, `married`, and `sector` are categorical.

Lastly, MHEALTH.csv data file contains a lot of information about the health measures of 40 males. All variables are numerical.

```
[3]: import pandas as pd
url="https://raw.githubusercontent.com/leonkag/Statistics0/main/MHEALTH.csv"
mydata = pd.read_csv(url) # save as mydata file
print('Data Frame dimensions: ',mydata.shape) # #rows and # cols
mydata = mydata[['AGE','HT','WT','WAIST','PULSE','BMI']];
mydata.head(10)
```

Data Frame dimensions: (40, 14)

	AGE	HT	WT	WAIST	PULSE	BMI
0	20	69.7	137.4	75.2	88	19.9
1	25	67.6	151.3	75.6	64	23.3
2	34	63.7	119.5	75.7	56	20.7
3	29	70.0	162.4	77.0	56	23.4
4	22	66.2	144.2	78.1	64	23.2
5	41	66.5	135.0	78.8	60	21.5
6	26	68.5	144.1	79.6	64	21.6
7	20	66.3	151.0	79.9	72	24.2
8	36	70.3	137.1	81.6	64	19.6
9	54	65.6	139.0	82.5	60	22.7

In addition, numerical (quantitative) variables can be subdivided into **discrete** and **continuous**. **Discrete variables** can only take on distinct, separate values, which are typically counted in whole numbers (0,1,2,...). Unlike **continuous variables**, which can take on any value within a range, discrete

variables have a finite or countably infinite number of possible values. For example, the number of students in a class, the number of cars in a garage, the number of books on a shelf, etc. are discrete, while distance, time, height, weight, etc. are continuous.

Categorical (qualitative) variables can also be subdivided into **nominal** and **ordinal**. **Nominal variables** represent categories or labels without any inherent order or relationship. For example, gender, eye color, marital status, experimental group (Treatment vs. Control), country, etc. The values of nominal variables are called **levels**. **Ordinal variables** are categorical variables that have order or ranking, but the intervals in between the levels are not necessarily uniform or meaningful. For example, education level (high school < bachelor < master < Ph.D.), income (low < medium < high), and Likert scale in surveys (strongly disagree < disagree < neutral < agree < strongly agree).

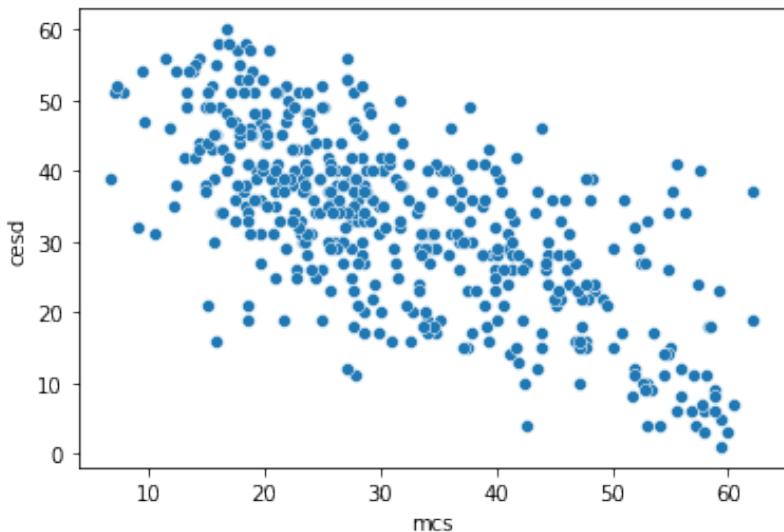
For numerical variables both discrete and continuous, numerical statistics like mean, and median, as well as transformations, make sense. For categorical variables, whether nominal or ordinal, only counting in each group (level) makes sense. Also note that there are some variables represented via numbers that are categorical, like zip codes. The mean zip code of students in a class doesn't make any sense.

In addition to the individual variable classification, relationships between variables play an important role in Statistics. For two numerical variables, a scatterplot illustrates a possible relationship. For example, consider the dependence between **cesd** depression score and **mcs** mental score for substance abusers shown in the Figure below:

```
[ ]: import pandas as pd;
      import seaborn as sns  # - seaborn is a Statistics graphics library

      url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrc.csv"
      mydata = pd.read_csv(url)    # save as mydata file
      # mydata.head(10)
      sns.scatterplot(data=mydata, x="mcs", y="cesd")
```

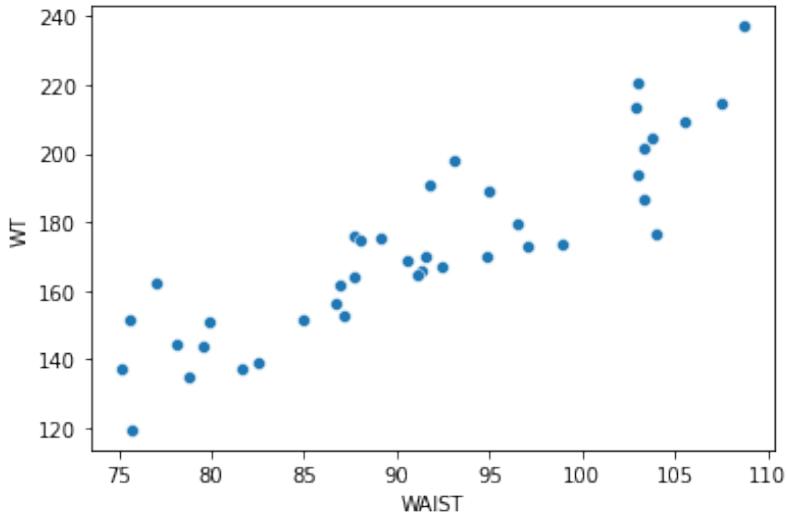
```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3e15d4c460>
```



Not surprisingly, as the mental score becomes higher, the depression score decreases, so these variables are **negatively associated (dependent)**. In this setting, the x-axis **mcs** would be an **explanatory (independent, predictor) variable**, while the y-axis **cesd** would be a **response (dependent) variable**. On the other hand, men's weight (**WT**) is **positively associated** with waist measurement (**WAIST**) as shown in the Figure below.

```
[ ]: url="https://raw.githubusercontent.com/leonkag/Statistics0/main/MHEALTH.csv"
mydata = pd.read_csv(url)    # save as mydata file
# mydata.head(10)
sns.scatterplot(data=mydata, x="WAIST", y="WT")
```

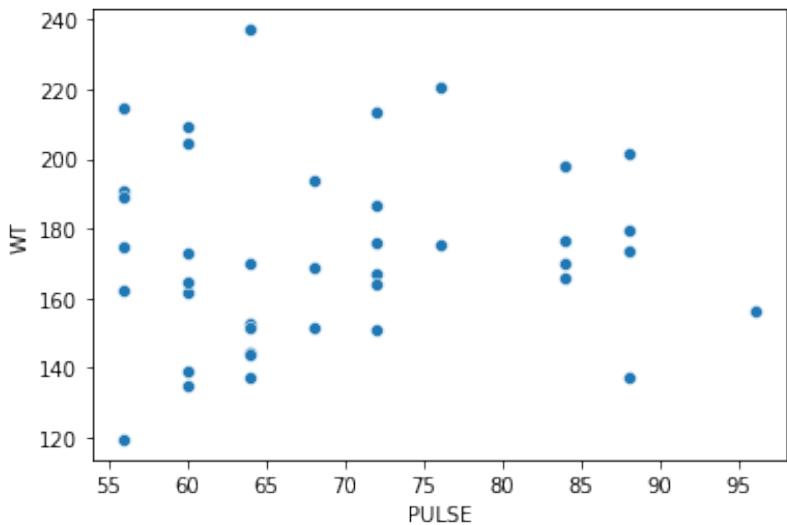
```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3e15ce27f0>
```



Finally, men's weight (WT) is **not associated (independent)** with pulse (PULSE) as shown in Figure below.

```
[ ]: sns.scatterplot(data=mydata, x="PULSE", y="WT")
```

```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3e157198b0>
```



Remember that **association does not imply causation**. The mcs does not cause the reduction in cesd score; they are just measurements of depression and mental state which are related to each other. A classic example of association without causation is NYC subway fare and average pizza slice price, which have stayed almost the same throughout the years since the 1950s. Both prices have grown over the years due to inflation, etc., not one causing the another.

The data is collected in two primary ways: **observational studies and experiments**. If researchers do not directly interfere with how the data arises, it is an **observational study**. The data sets considered in this section are all observational. Any opinion polls, surveys, etc. are observational. A study looking back in time at medical or business records (*retrospective studies*), and *prospective studies* where a cohort (group) of similar individuals is followed over many years into the future are all observational. **Experiments**, however, involve manipulating one or more independent variables to observe their effect on a dependent variable. The example at the beginning of the chapter on the effects of a new cholesterol-reducing drug was an experimental study. Researchers randomized patients into treatment and control groups and proportions were compared. The explanatory group variable has two levels (treatment and control). The proportions by group are the responses. *Observational studies may only show the association/relationship between variables, not cause and effect. Only a carefully designed randomized experiment may be used to establish cause and effect.*

To illustrate why it would be wrong to infer cause and effect based on a simple association, let's consider the following example. A medical study focusing on blood pressure and mortality may find that low blood pressure is associated with a higher risk of mortality. However, it does not cause death directly, rather, it is due to the confounding effect of heart disease. A variable that is associated with both the explanatory (low blood pressure) and response variable (mortality) but is not directly accounted for by the study is called a **confounding (lurking) variable**. There are ways to account for confounding variables; they should not be dismissed.

---

### 1.3 Sampling Methods

In statistics, a **population** is the pool of all individuals or items from which a much smaller **statistical sample** is drawn for a study. For example, in the drug study considered before, 200 patients with high cholesterol constitute a random sample out of the whole population of such patients. It is impractical

to take **census** of the entire population, instead, a small randomized sample is considered. For example, presidential approval polls are conducted almost every day. It is impossible each time to run a census of many millions of American voters, so a much smaller 1200-1500 random sample is obtained. A sample should be **representative of a target population**, which is not an easy task undertaken by polling agencies. Most of them failed to correctly predict Donald Trump's election in 2016 missing new "Trump Voters" giving him an edge. A characteristic of the entire population is called a **parameter** and denoted by a Greek letter like a population mean  $\mu$ . The same quantity measured on a sample is called a **statistic** denoted by a regular letter with some special sign like a sample mean  $\bar{x}$ .

In Statistics, we almost always use a **simple random sample** - an **unbiased** approach where a subset is chosen randomly with all items equally likely to be selected (like a computerized raffle). Systematic bias is different from natural variation in the data, and cannot be corrected by any statistical techniques. For example, a survey on smartphone usage conducted on tech-savvy young users would be highly inaccurate in reflecting the entire population's phone usage.

Another possible flaw in sampling is a **voluntary response** sample. For example, a voting questionnaire was mailed to a large sample of individuals, but only a much smaller sub-sample responded corresponding to a high **non-response rate**. It could be that most people who felt strongly about the election responded, which would produce an additional *bias*. Similarly, self-selected polls of viewers of highly partisan networks like FOX and MSNBC are not representative of the entire population. There are other more specialized sampling techniques like **stratified sampling**, **cluster sampling**, etc., but they are mostly used in polling and will not be considered in this textbook.

---

## 1.4 Statistical Experiments

In this section, we consider **randomized experimental design** that involves randomly assigning participants to different experimental conditions to control for confounding and allow causal inferences. A control group must always be included to provide a baseline for comparison with one or more of the experimental groups. For example, in the new drug study described at the beginning of the chapter, there was a control group of patients taking the standard drug. Also, the patients in the two groups should take the new drug and the standard one in a similar manner, say after a meal, same time of the day, etc.. Randomization can be achieved via random number generators. This helps to account for other variables that cannot be controlled. For example,

some patients may be healthier due to their lifestyle, so randomizing helps to even out such differences and avoid bias. The experiment should be replicated a sufficient number of times with different samples and/or in different settings. It strengthens the confidence in being able to generalize the observed effects to a much larger population.

In some studies, a **randomized block design** is called for that involves dividing participants into homogeneous blocks before random assignment. It can be particularly useful when there are factors that may influence the outcome but are not the primary focus of the study (confounding variables). For example, considering again the new drug experiment, it seems logical to assume that the drug reaction could be different for patients with moderately high cholesterol levels (moderate) vs. patients with very high cholesterol levels (high). Therefore, we create a blocking variable Chol with two levels of moderate and high, separate all 200 patients according to this variable, and then take simple random samples of equal sizes from moderate and high groups, respectively. This strategy ensures each treatment group has an equal number of moderate and high cholesterol level patients.

In addition, it is important to reduce unintended human bias via **double blinding** where neither the patient nor the doctor knows whether a particular patient is on a new drug or a standard drug. If a new drug is tested against no treatment, the patients in the control (no treatment) group should still receive a placebo to avoid possible **placebo effects**.

---

## 1.5 Roadmap for the Book

At the end of this first chapter of the book, let's discuss the roadmap for the rest of the text. In this chapter, some of the basic ideas of Statistics have been introduced. The next chapter reviews the main approaches of the **Descriptive Statistics**. It summarizes a sample in terms of means, standard deviations, and counts as well as introduces important visualizations like histograms, boxplots, etc. Within the descriptive approach, we only try to describe the sample at hand, not to infer anything about the larger population which it came from; that would be the task of **Inferential Statistics**.

The inferential Statistics method uses sample statistics to make inferences about population parameters. First, however, basic probability ideas and probability distributions relevant to Statistics (normal and binomial) must be introduced/reviewed. The next chapters introduce the main approaches of inferential statistics such as confidence intervals and hypothesis testing. They are applied to both categorical and numerical data leading to proportions,

goodness of fit, independence, and means tests. Next, Part 2 of the book will look at more advanced topics such as ANOVA, linear and nonlinear models, multiple and logistic regression, tree models, principal component analysis, clustering, etc.

# 2

---

## *Descriptive Data Analysis*

---

This chapter reviews the ideas of **Descriptive Statistics** that summarize a sample in terms of means, standard deviations, counts, visualizations, etc. There are no attempts to generalize sample statistics to the entire population (**Inferential Statistics**).

---

### 2.1 Numerical Data

We defined in the previous introductory chapter numerical data as quantitative data that could be described with numbers. In this chapter, we review standard measures of center and spread of such data. In addition to introducing basic Statistics definitions, we introduce some new ideas such as dealing with NAs, trimmed mean, etc.

#### 2.1.1 Sample and Population Means

We start with the most standard measure of center of a numerical sample, **mean** (average).

For a sample of  $n$  measurements  $x_1, x_2, \dots, x_n$ , the mean is defined as:

$$\bar{x} = \frac{x_1 + x_2 + \dots + x_n}{n} = \frac{\sum_{i=1}^n x_i}{n} = \frac{1}{n} \sum_{i=1}^n x_i \quad (2.1)$$

The way to compute it depends on whether you are dealing with a small set of data to be typed into an array manually or a data file.

Let's first assume that we just have a small data set of heights for students in a class, and we want to find the average. Python array can be defined with `[]`, but *numpy* library needs to be imported to use numerical arrays and statistical functions effectively. Numpy arrays are better for Statistics, as they are vectorized (addition, subtraction, and many other operations are automatically done element-by-element) and have a rich set of mathematical functions. First, `np.array()` is defined, then `np.mean` is used to compute the mean. The results are printed with format specifications like `{: .4f}` to display the desired number of digits rather than the full 16 double-digits-accuracy

```
[2]: import numpy as np # numerical library
x = np.array([69,64,63,73,74,64,66,70,70,68,67])
xbar = np.mean(x);
print('xbar = {:.4f} for n = {:d} observations'.format(xbar, len(x)))
```

```
xbar = 68.0000 for n = 11 observations
```

Note that if any of the data are missing and represented by *nan*, the mean computes to *nan*. To avoid it, we have to adjust our command:

```
[4]: x = np.array([69,64,63,73,74,64,66,70,70,68,67,float("nan")])
xbar = np.mean(x); print('regular mean xbar = {:.4f}'.format(xbar))
xbar = np.nanmean(x); print('nanmean() xbar = {:.4f}'.format(xbar))
```

```
regular mean xbar = nan
nanmean() xbar = 68.0000
```

Alternatively, we can trim the mean to a specified percentage on low and high extremes.

```
[5]: from scipy import stats
x = np.array([69,64,63,73,74,64,66,70,70,68,67,float("nan")])
xbar = stats.trim_mean(x,0.1);
print('trimmed mean xbar = {:.4f}'.format(xbar))
```

```
trimmed mean xbar = 68.5000
```

Next, let's find the average for a variable defined in the data file. We use `cesd` depression score in `HELPrc` data file, which we used before:

```
[7]: import pandas as pd; import numpy as np
url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrc.csv"
mydata = pd.read_csv(url) # save as mydata file
print(mydata[['cesd','pcs','mcs']].head(10)) # first 10 lines for 3 vars
print('\n')

xbar = np.mean(mydata['cesd']) # data column is referenced as ['cesd']
print('mean of cesd = {:.4f}'.format(xbar))
```

	cesd	pcs	mcs
0	49	58.413689	25.111990
1	30	36.036942	26.670307
2	39	74.806328	6.762923
3	15	61.931679	43.967880
4	39	37.345585	21.675755
5	6	46.475212	55.508991
6	52	24.515039	21.793024
7	32	65.138008	9.160530
8	50	38.270878	22.029678
9	46	22.610598	36.143761

```
mean of cesd = 32.8477
```

We can also find the mean of several columns at once with `np.mean()` function or `.mean` method as illustrated below.

```
[34]: print('Mean of the two columns using np.mean:')
print( np.mean(mydata[['cesd', 'mcs']],axis=0) )

print('\nMean of the two columns .mean():')
print( mydata[['cesd', 'mcs']].mean() )
```

Mean of the two columns using np.mean:

```
cesd    32.847682
mcs     31.676678
dtype: float64
```

Mean of the two columns .mean():

```
cesd    32.847682
mcs     31.676678
dtype: float64
```

We can also trim the means of data frame columns:

```
[20]: print('\nTrimmed mean from stats library : ',
      stats.trim_mean(mydata[['cesd', 'mcs']], 0.05))
```

Trimmed mean from stats library : [33.06356968 31.31445885]

The `cesd` depression scores represent a sample from a population of such depression scores for all substance abusers. The **population mean** of all  $N$  such patients' scores is found similarly:

$$\mu = \frac{x_1 + x_2 + \dots + x_N}{N} = \frac{\sum_{i=1}^N x_i}{N}$$

It is usually impractical to find the mean of the entire population, so  $\mu$  is estimated using the sample mean  $\bar{x}$ . For the depression score,  $\bar{x} = 32.85$  is a **point estimate** of the true population mean of the depression score for all such substance abusers. Naturally, for a larger sample, the accuracy improves. In a later chapter, we develop a quantitative way (confidence interval) to quantify the accuracy of such point estimates.

Consider the following interesting example of computing means. Assume an exam is given to 29 out of 30 students in the class on a regular date and the class mean is 72. The last remaining student takes a makeup exam due to family reasons and gets 84. What is the new mean? In this case, the actual exam scores are not given, however, the sum of scores for the 29 students who took the exam on the regular date can be found:

$$\bar{x} = \frac{\sum x}{n} \implies \sum x = \bar{x} \cdot n = 72 \cdot 29 = 2088$$

Then the new mean can be found as:

$$\frac{72 \cdot 29 + 84}{30} = \frac{2088 + 84}{30} = 72.4$$

which is a little bit higher than before, as it should be since the missing student's score is higher than the mean.

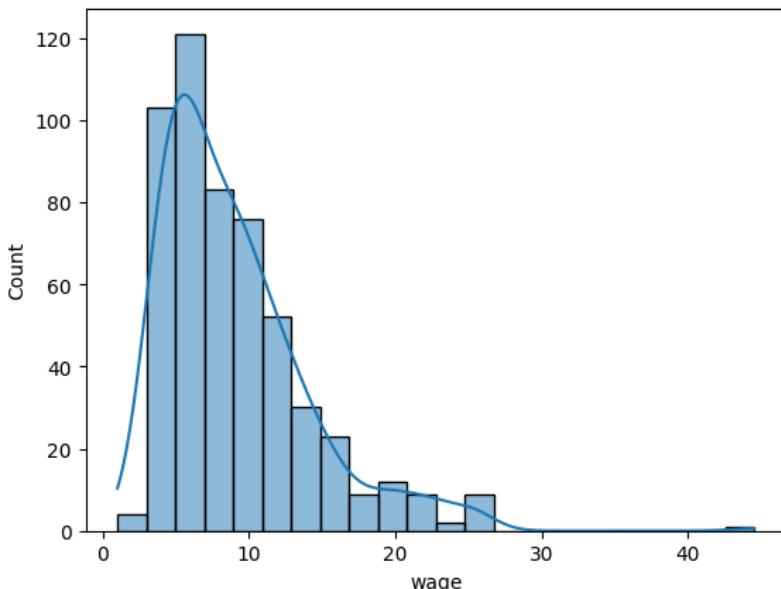
### 2.1.2 Numerical Distribution Shape

A **histogram** is a graphical representation of numerical data that groups observations into bins. The width of the bins can be varied. The y-axis shows frequencies (number of observations) per bin. The high-level seaborn graphics library is used below to display the histogram of hourly wages from the CPS85 file (kde=True adds smoothed-out density plot on top of the histogram):

```
[ ]: import pandas as pd; import numpy as np; import seaborn as sns
url="https://raw.githubusercontent.com/leonkag/Statistics0/main/CPS85.csv"
mydata = pd.read_csv(url) # save as mydata file
# print(mydata.head())

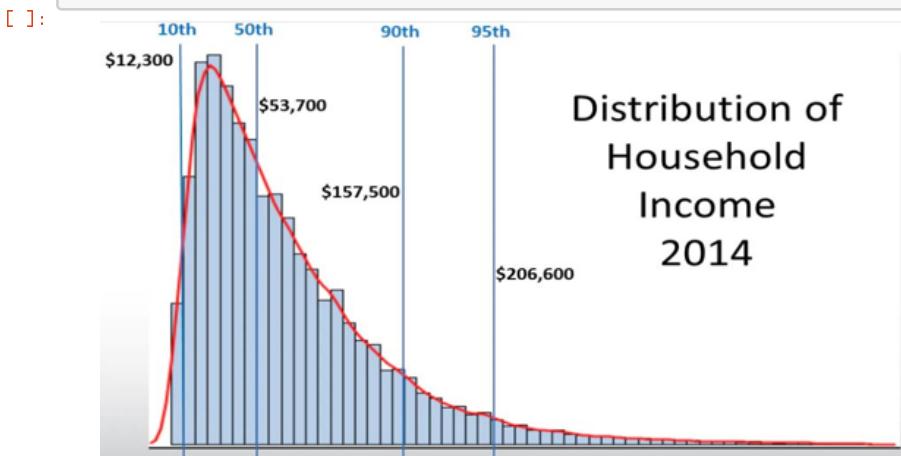
sns.histplot(data=mydata, x="wage", binwidth=2, kde=True)
# kde adds kernel density estimate smoothing histogram shape
# of the distribution. The binwidth allows varying the number of bins.
```

```
[ ]: <Axes: xlabel='wage', ylabel='Count'>
```



The **shape** of the histogram above indicates that most hourly workers made below \$10/hour, much fewer made around \$20/hour, and only one made over \$40/hour (an outlier). The shape of hourly wages trails off to the right - **right-skewed shape**. The most common example of a right-skewed distribution is income. Most people stay in a lower income range with only a few making a lot resulting in a long, thin tail to the right illustrated in the distribution of household income in 2014 reproduced below:

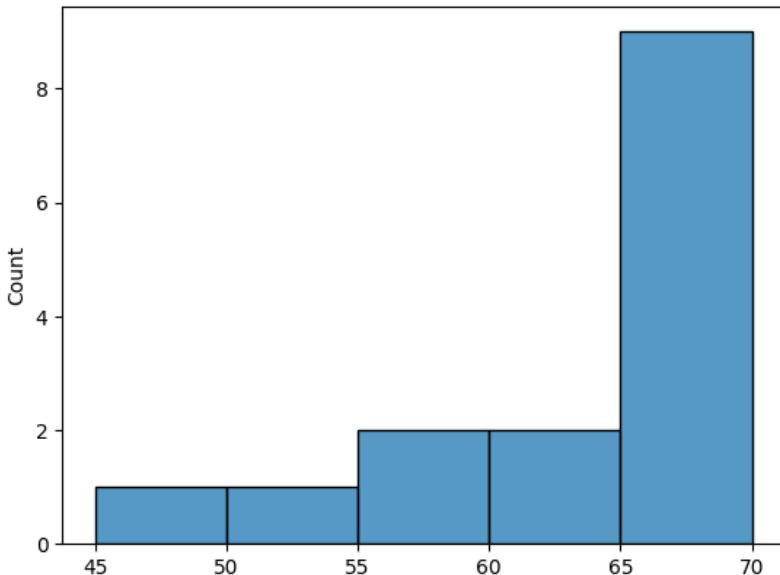
```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main/Income2014.
->JPG'
page = requests.get(url); img = Image.open(BytesIO(page.content));
img.resize((550, 300))
```



Next, the histogram of the retirement age data below has a longer tail to the left - **left-skewed shape**. Most people retire in their 60s and 70s, but a few could take early retirement in their 40s and 50s.

```
[ ]: import pandas as pd; import numpy as np; import seaborn as sns
x = np.array([69,45,67,52,57,59,65,64,65,65,70,67,67,62,65])
sns.histplot(x)
```

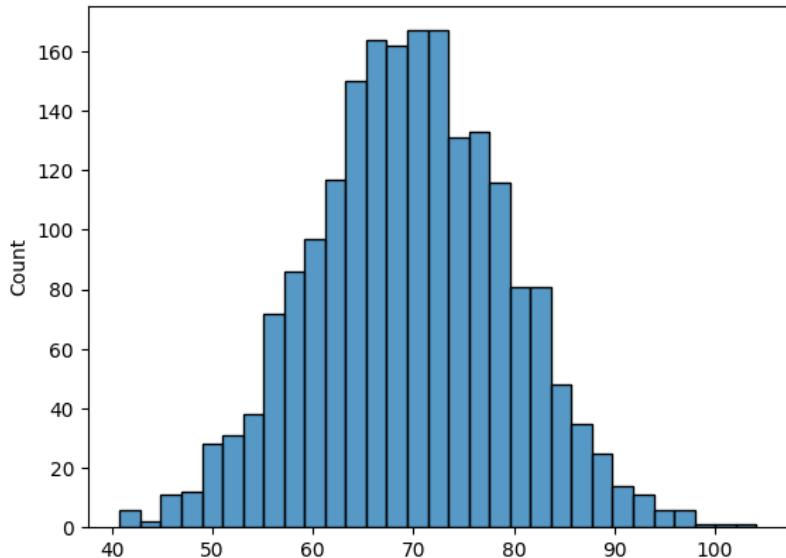
```
[ ]: <Axes: ylabel='Count'>
```



Finally, the Python code below generates **symmetric** (bell-shaped) distribution of test scores that trail off about the same left and right.

```
[ ]: import numpy as np; import seaborn as sns
# Set seed for reproducibility (optional) np.random.seed(1)
# Mean and standard deviation for the normal distribution
mean = 70; std_dev = 10
# Number of test scores to generate
num_scores = 2000
# Generate normally distributed test scores
test_scores = np.random.normal(mean, std_dev, num_scores)
# Plot the histogram of the generated test scores
sns.histplot(test_scores)
```

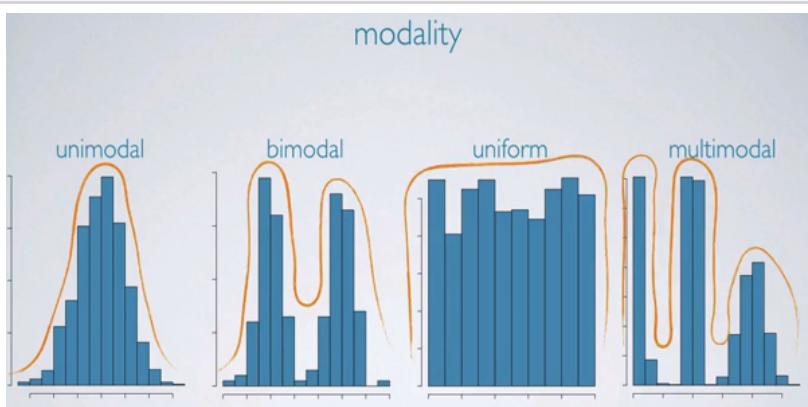
```
[ ]: <Axes: ylabel='Count'>
```



Histograms can help us to identify distribution **modes** - prominent peaks in the distribution. The histograms considered so far in this section all had only one mode (peak) - **unimodal**. A distribution with two modes is called **bimodal**, while a distribution with more than two modes is called **multimodal** as illustrated in the Figure below:

```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main/
→UniBiMultiModal.jpg'
page = requests.get(url); img = Image.open(BytesIO(page.content));
img.resize((600, 300))
```

[ ]:



The modes are not very rigorously defined but help to get a better sense of the data. For example, a class grade distribution is often unimodal with only one prominent peak, which is assigned B or B- range. However, sometimes there are two separate groups of students with good and bad grades with their own peaks - bimodal.

### 2.1.3 Variance and Standard Deviation

The mean describes the distribution's center, but the data's variability around the mean is equally important. In finance, variability (*volatility*) of investment returns measures its risk. The standard way to measure variability around the data mean is the **standard deviation**.

Define  $d_i = x_i - \bar{x}$ ,  $i = 1, 2, \dots, n$  as deviations of data from the overall mean  $\bar{x}$ . The mean is at the center of the data, so some deviations are positive, while others are negative; therefore, they would cancel each other out in a sum or average:

$$\sum d_i = \sum (x_i - \bar{x}) = \sum x_i - \sum \bar{x} = \bar{x}n - \bar{x}n = 0$$

Instead, the **sample standard deviation** averages squared deviations:

$$s = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}} \quad (2.2)$$

The **sample variance** is just the square of sample standard deviation:

$$s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}$$

The division by the **degree of freedom**  $n - 1$  rather than  $n$  ensures that sample variance is an unbiased estimate of the true **population variance**:

$$\sigma^2 = \frac{\sum_{i=1}^n (x_i - \mu)^2}{N}$$

Let's illustrate it first with the grades list considered before. Note the difference between population vs. sample standard deviation calculation in Python.

```
[ ]: import numpy as np; import pandas as pd
x = np.array([69, 64, 63, 73, 74, 64, 66, 70, 70, 68, 67])

# this part of the code is FYI #####
xbar = np.mean(x)
d = x - xbar;
d2 = d**2      # d^2
```

```
stdframe = pd.DataFrame({'x':x, 'xbar':np.repeat(xbar,len(x)), 'd':d, 'd2':d2})
print(stdframe, '\n')
print('sum of deviations =', np.sum(d), '\n')
#####
s = np.std(x,ddof=1);      # sample standard deviation requires ddof=1!!
sig = np.std(x);           # population standard deviation (default)
print('s, sig = ', s, sig)
```

	x	xbar	d	d2
0	69	68.0	1.0	1.0
1	64	68.0	-4.0	16.0
2	63	68.0	-5.0	25.0
3	73	68.0	5.0	25.0
4	74	68.0	6.0	36.0
5	64	68.0	-4.0	16.0
6	66	68.0	-2.0	4.0
7	70	68.0	2.0	4.0
8	70	68.0	2.0	4.0
9	68	68.0	0.0	0.0
10	67	68.0	-1.0	1.0

sum of deviations = 0.0

s, sig = 3.63318042491699 3.4641016151377544

Now apply it to a data file column:

```
[24]: import pandas as pd;   import numpy as np
url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPPrct.csv"
mydata = pd.read_csv(url)    # save as mydata file

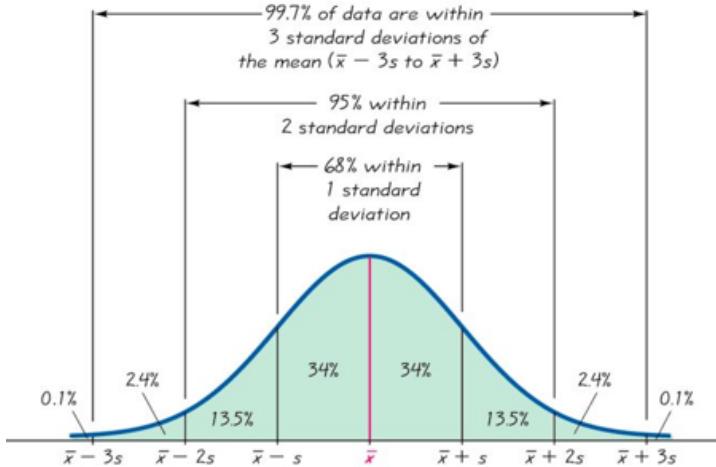
s = np.std(mydata['cesd'],ddof=1)  # data column is referenced as ['cesd']
print('s = {:.4f}'.format(s))
```

s = 12.5145

The standard deviation meaning is defined in terms of a given data distribution. We derive in the later chapter that for the **bell-shaped normal distribution**, about 68% of observations are within 1 standard deviation of the mean, 95% are within 2, and 99.7% are within 3 as shown in the Figure below:

```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main/
→Normal1s2s3s.JPG'
page = requests.get(url); img = Image.open(BytesIO(page.content));
img.resize((450, 300))
```

[ ]:



For example, the IQ score distribution tends to be bell-shaped and usually formatted so that the mean is 100 with a standard deviation of 15. Therefore:

$$\text{Mean} \pm 1 \cdot sd = 100 \pm 1 \cdot 15 = (85, 115) \text{ contains } 68\% \text{ of the data.}$$

$$\text{Mean} \pm 2 \cdot sd = 100 \pm 2 \cdot 15 = (70, 130) \text{ contains } 95\% \text{ of the data.}$$

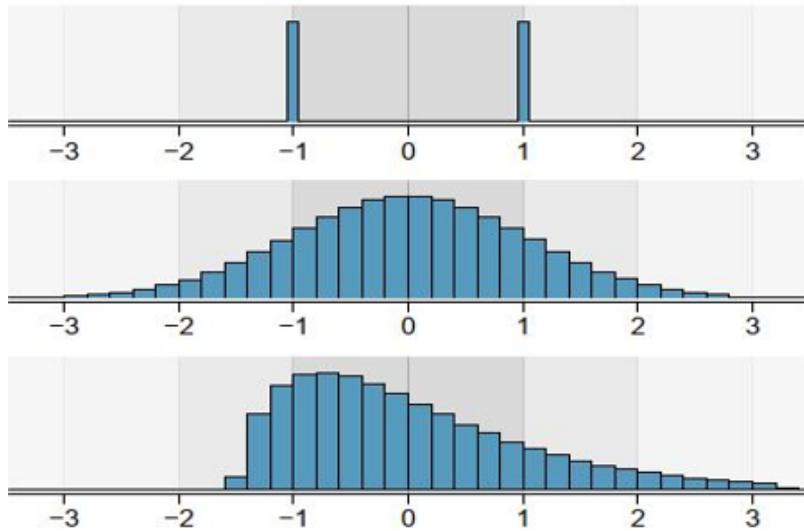
$$\text{Mean} \pm 3 \cdot sd = 100 \pm 3 \cdot 15 = (55, 145) \text{ contains } 99.7\% \text{ of the data.}$$

This is *only true for the symmetric bell-shaped distribution*; however, a general Chebyshev Theorem in Probability shows that the proportion of observations within  $k$  standard deviations of the mean is at least  $1 - \frac{1}{k^2}$ . Therefore, even highly skewed bank accounts distribution is guaranteed to have at least  $1 - \frac{1}{2^2} = 1 - \frac{1}{4} = \frac{3}{4} = 75\%$  of the data within 2 standard deviations of the mean.

It should be noted that mean, standard deviations, and any other summary statistics do not completely describe the data. It is still crucial to look at the actual histogram. For example, the Figure below shows three distributions that look quite different, but all have the same mean 0 and standard deviation 1. Using modality, we can distinguish between the first plot (bimodal) and the last two (unimodal). Using skewness, we can distinguish between the last plot (right-skewed) and the first two.

```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main/
↪SameMeanSd3distr.JPG'
page = requests.get(url); img = Image.open(BytesIO(page.content));
img.resize((450, 300))
```

[ ]:



We made a note before that the standard deviation is in the same units as the original observations. Say, if the stock price is given in dollars, so is the standard deviation. However, how would you compare the variability of two data sets given in *different units* (say, stock prices in dollars vs. in pesos)? Analogously, how to compare the variability of two stocks with very different overall prices - say, one stock sells in hundreds of dollars and another in cents.

To facilitate comparisons like that, define **coefficient of variation**:

$$CV = \frac{s}{\bar{x}} 100\%$$

A sample standard deviation and mean have the same units, so once divided, a non-dimensional number is obtained that scales with the magnitude of the observations. For high values of CV, the standard deviation is equal to the large percentage of the mean and the data is very volatile, while low CVs, say below 10%, are indicative of small volatility. Consider that data on two companies A and B stock prices below:

```
[28]: import numpy as np
StockA = np.array([1.23, 1.50, 2.27, 0.54, 2.40, 3.10, 5.18])
StockB = np.array([171, 169, 182, 159, 164, 175, 180, 173, 176, 177])

Abar = np.mean(StockA); sa = np.std(StockA, ddof=1); CVa = sa/Abar*100;
print('Company A: ')
print('Mean = {:.3f}, standard deviation = {:.3f}, CV = {:.3f}'.
      format(Abar, sa, CVa))

Bbar = np.mean(StockB); sb = np.std(StockB, ddof=1); CVb = sb/Bbar*100;
print('Company B: ')
print('Mean = {:.3f}, standard deviation = {:.3f}, CV = {:.3f}'.
      format(Bbar, sb, CVb))
```

```
Company A:
Mean = 2.317, standard deviation = 1.518, CV = 65.511
Company B:
Mean = 172.600, standard deviation = 7.106, CV = 4.117
```

So stock B is a lot less volatile.

### 2.1.4 Median, Quartiles, and Boxplot

The **median** is the middle value of the **ordered** data. Without the ordering, this definition makes no sense, so the first step is to rearrange the data into an ordered array. Then, the median is the value such that half of the observations are larger, and half are smaller. It is also the 50th percentile (50% below, 50% above). If the sample size  $n$  is odd, the median is the middle observation of the ordered array. If  $n$  is even, it is the average of the two central observations. In the code below we illustrate the computation of the median for small sets of numbers. Note that the `np.sort()` command is only used to see the sorted list for our convenience, the `np.median()` command sorts it by itself.

```
[ ]: import numpy as np
x = np.array([3, 99, 100, 5, 20, 2, 0])
n = len(x); print('number of data points = ',n)
print('sorted array: ', np.sort(x))
xmed = np.median(x);
print('xmed = ', xmed)

x = np.array([3, 99, 5, 20, 2, 0])
n = len(x); print('number of data points = ',n)
print('sorted array: ', np.sort(x))
xmed = np.median(x);
print('xmed = ', xmed)
```

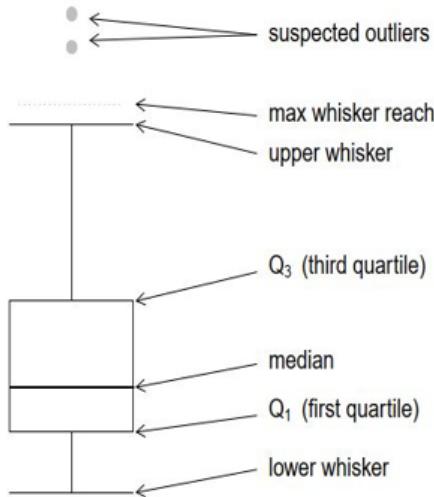
```
number of data points =  7
sorted array:  [ 0   2   3   5   20  99 100]
xmed =  5.0
number of data points =  6
sorted array:  [ 0   2   3   5  20 99]
xmed =  4.0
```

The most important property of the median is that, unlike the mean, it is **not affected as much by extreme values (outliers)**. If we replace 100 by 100 million in the example above, the mean changes dramatically, but the median is still 5.

The median breaks ordered data in two halves; in turn, quartiles break the halves in half. The **first quartile**  $Q_1$  is the 25th percentile (25% of the data are below this value and 75% above). The **third quartile**  $Q_3$  is the 75th percentile (75% below, 25% above). A **boxplot** represents the data graphically as shown in the diagram below.

```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main/BoxPlotdef.
↪JPG'
page = requests.get(url); img = Image.open(BytesIO(page.content));
img.resize((300, 350))
```

```
[ ]:
```

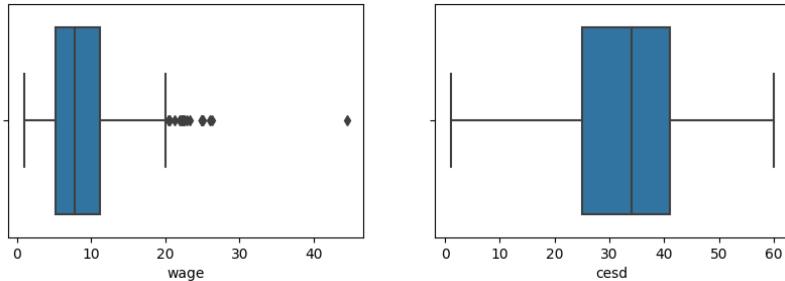


The actual box is from  $Q_1$  to  $Q_3$  and the median is drawn as a line inside this box. The **interquartile range** is the length of the box:  $IQR = Q_3 - Q_1$ . It contains the middle 50% of the data, so it is also a measure of variability. The **upper whisker** extends from  $Q_3$  to the highest observation below  $Q_3 + 1.5 \cdot IQR$ . If there is no observation at or close to this value, the upper whisker may be considerably below this value. The observations above the upper whisker are considered to be **outliers** and denoted by dots or stars. Similarly, the lower whisker extends from  $Q_1$  down to the lowest data point above  $Q_1 - 1.5 \cdot IQR$  (it could be considerably above). The observations below this value are outliers as well. The Figure below compares boxplots of `wage` from CPS85 and `cesd` depression scores from HELPrct.

```
[ ]: import pandas as pd; import numpy as np;
import seaborn as sns; import matplotlib.pyplot as plt;

url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrct.csv"
HELPrct = pd.read_csv(url)
url="https://raw.githubusercontent.com/leonkag/Statistics0/main/CPS85.csv"
CPS85 = pd.read_csv(url)
fig, axes = plt.subplots(1, 2, figsize=(10, 3), sharey=False)
sns.boxplot(data=CPS85, x="wage", ax=axes[0])
sns.boxplot(data=HELPrct, x="cesd", ax=axes[1])
```

```
[ ]: <Axes: xlabel='cesd'>
```



The `quantile()` function can be used to compute any percentile.

The  $Q_1$  is 25th percentile and  $Q_3$  is 75th. Below, we use these values to compute the interquartile range and whisker positions.

```
[ ]: x = CPS85.wage # we can refer to just one variable with dot .wage
Q1 = np.quantile(x,0.25);
Q3 = np.quantile(x,0.75);
IQR = Q3-Q1;
print('Q1, Q3, IQR, whisker lower bound, whisker upper bound = ',
      Q1, Q3, IQR, Q1-1.5*IQR, Q3+1.5*IQR)
```

```
Q1, Q3, IQR, lower bound, upper bound =  5.25 11.25 6.0 -3.75 20.25
```

The boxplot of `wage` is highly skewed, and many observations are identified as outliers above the upper whisker at 20.25. The lower whisker would have been at  $-3.75$ , which is below the lowest wage value, so it stops at that value. The `cesd` boxplot on the right side of the Figure above is approximately symmetric and doesn't show any outliers.

We have considered mean, median, and quartile in separate commands, but there is a way to get a complete summary of several variables at once:

```
[ ]: HELPrct[['cesd','mcs','pcs']].describe()
```

	cesd	mcs	pcs
count	453.000000	453.000000	453.000000
mean	32.847682	31.676678	48.048542
std	12.514460	12.839337	10.784603
min	1.000000	6.762923	14.074291
25%	25.000000	21.675755	40.384377
50%	34.000000	28.602417	48.876808
75%	41.000000	40.941338	56.953285
max	60.000000	62.175503	74.806328

```
[ ]: CPS85[['wage','exper','educ']].describe()
```

```
[ ]:      wage      exper      educ
count  534.000000  534.000000  534.000000
mean   9.024064   17.822097   13.018727
std    5.139097   12.379710   2.615373
min   1.000000   0.000000   2.000000
25%   5.250000   8.000000  12.000000
50%   7.780000  15.000000  12.000000
75%  11.250000  26.000000  15.000000
max   44.500000  55.000000  18.000000
```

The most extreme outlier in wage observations is particularly large, it is more than 40\$ per hour. Let's compare sample statistics measurements with and without it by introducing a subset of data with .loc[].

```
[ ]: CPS85new = CPS85.loc[CPS85.wage<30] # select subset of data frame
print('Original wage summary: \n', CPS85.wage.describe())
print()
print('Updated wage summary: \n', CPS85new.wage.describe())
```

```
Original wage summary:
count      534.000000
mean      9.024064
std       5.139097
min      1.000000
25%     5.250000
50%     7.780000
75%    11.250000
max     44.500000
Name: wage, dtype: float64

Updated wage summary:
count      533.000000
mean      8.957505
std       4.908140
min      1.000000
25%     5.250000
50%     7.780000
75%    11.250000
max     26.290000
Name: wage, dtype: float64
```

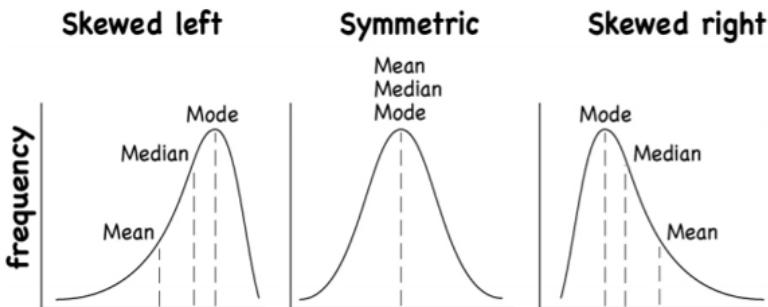
The median and quartiles Q1 and Q3 (therefore IQR) did not change. They are so-called **robust statistics** - extreme observations have little effect. However, the mean and standard deviations changed (**non-robust statistics** - affected strongly by extremes). In principle, if observations are highly skewed like incomes, median and IQR are better measures of center and spread, while for more symmetrical data like IQ scores and heights, mean and standard deviation are better measures, respectively.

The center graph in the Figure below shows that for the symmetrical bell-shaped data, mean, median and mode coincide. The right-skewed data shows that extreme tail values pull the mean to the right much further than the median (robust). Analogously, for the left-skewed data, the extreme left tail

values pull the mean to the left much more than the median.

```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main/
→MeanMedModeGraph.png'
page = requests.get(url); img = Image.open(BytesIO(page.content));
img.resize((600, 250))
```

[ ]:



[ ]:

## 2.2 Categorical Data

Categorical data, whether nominal or ordinal, can only be counted in each level, mean, median, and other numerical summaries cannot be applied.

### 2.2.1 Tables and Barplots

The `value_counts()` method applied to a categorical variable produces the level counts, then `plot.bar()` method creates a barplot graph. Consider for example the barplot of the substance variable in the substance abusers HELPrct data file shown in the Figure below:

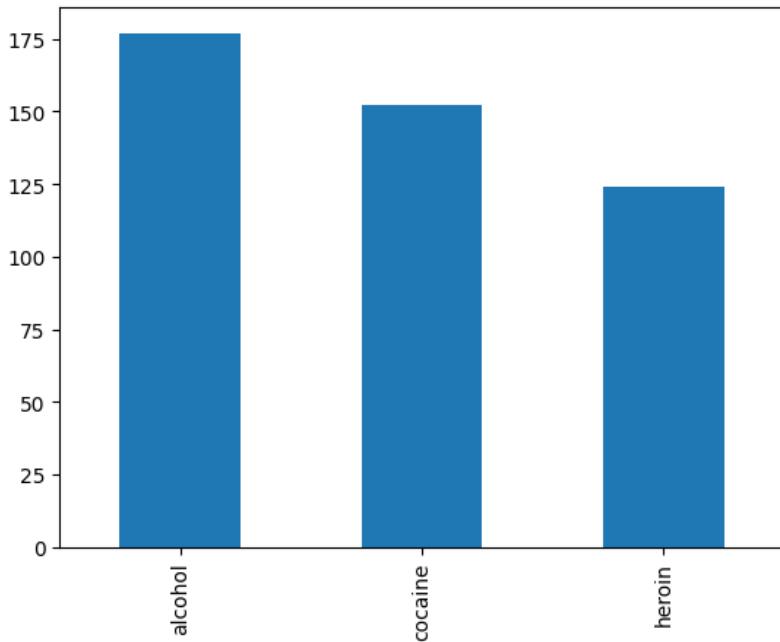
```
[ ]: import pandas as pd; import numpy as np; import seaborn as sns
url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrct.csv"
mydata = pd.read_csv(url) # save as mydata file

mytable = mydata['substance'].value_counts()
print(mytable)
mytable.plot.bar()
```

alcohol	177
cocaine	152

```
heroin      124  
Name: substance, dtype: int64
```

```
[ ]: <Axes: >
```

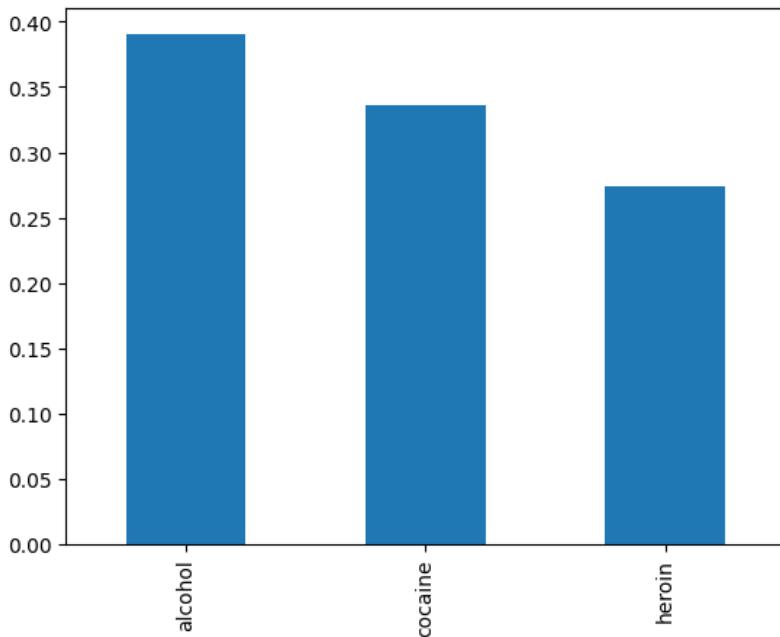


It shows the counts for each type of substance with alcohol the highest, then cocaine, and finally heroin. The `normalize=True` option allows to convert counts into proportions, which is more informative.

```
[ ]: mytable = mydata['substance'].value_counts(normalize=True)  
print(mytable)  
mytable.plot.bar()
```

```
alcohol      0.390728  
cocaine     0.335541  
heroin      0.273731  
Name: substance, dtype: float64
```

```
[ ]: <Axes: >
```



The same information as in a bar graph could be presented in a pie chart; however, human eyes discern differences in height far better than in angles and areas. Therefore, pie charts are *not* used in scientific literature.

### 2.2.2 Contingency Tables

In this section, a relationship between two categorical variables is investigated leading to **contingency tables**. Continuing with the previous section on substance abuse, add homeless status to the consideration. The `pd.crosstab()` function produces the contingency table with or without the margins. The `.plot.bar()` method creates barplots whether stacked or not.

```
[ ]: import pandas as pd;   import numpy as np;
import seaborn as sns; import matplotlib.pyplot as plt;
url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrc.csv"
mydata = pd.read_csv(url)    # save as mydata file

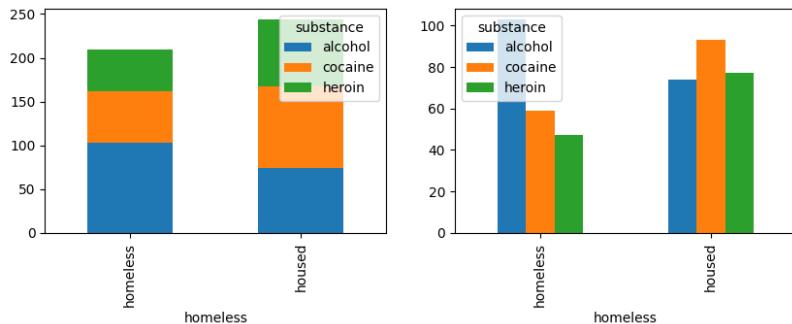
O  = pd.crosstab(mydata.homeless, mydata.substance, margins=False)
Om = pd.crosstab(mydata.homeless, mydata.substance, margins=True)
print(O, '\n')
print(Om, '\n')

fig, axes = plt.subplots(1, 2, figsize=(10, 3), sharey=False)
O.plot.bar(stacked=True, ax=axes[0])
```

```
0.plot.bar(stacked=False,ax=axes[1])
```

substance	alcohol	cocaine	heroin	
homeless				
homeless	103	59	47	
housed	74	93	77	
All	177	152	124	453
substance	alcohol	cocaine	heroin	All
homeless				
homeless	103	59	47	209
housed	74	93	77	244
All	177	152	124	453

```
[ ]: <Axes: xlabel='homeless'>
```



Each value in the table represents the number of times a particular combination of variable outcomes occurred. For example, there are 59 homeless cocaine abusers and 74 housed alcoholics. The margins provide row and column totals. For example, the total number of alcoholics is  $103+74 = 177$ , and the total number of housed patients is  $74+93+77 = 244$ , and the grand total of all patients is 453. We can also create a table of proportions. The overall proportions are obtained by dividing each entry by the grand total:

```
[ ]: pd.crosstab(mydata.homeless, mydata.substance, margins=True, normalize = u
    ↪'all')
```

substance	alcohol	cocaine	heroin	All
homeless	0.227373	0.130243	0.103753	0.461369
housed	0.163355	0.205298	0.169978	0.538631
All	0.390728	0.335541	0.273731	1.000000

For example, the proportion of housed alcoholics out of the total is  $74/453 = 0.163$ . Analogously, the proportion of homeless cocaine abusers is  $59/453 = 0.13$ .

It is more informative to investigate conditional (row and column) proportions.  
Row proportions:

```
[ ]: pd.crosstab(mydata.homeless, mydata.substance, margins=True, normalize = u
    ↪'index')
```

```
[ ]: substance    alcohol    cocaine    heroin
homeless
homeless    0.492823    0.282297    0.224880
housed      0.303279    0.381148    0.315574
All         0.390728    0.335541    0.273731
```

The output provided a proportions breakdown for housed and homeless. For example,  $103/209 = 0.493$  of all homeless are alcoholics,  $59/209 = 0.282$  of all homeless are cocaine abusers, etc... Analogously,  $74/244 = 0.303$  of all housed are alcoholics, etc...

Column proportions:

```
[ ]: pd.crosstab(mydata.homeless, mydata.substance, margins=True, normalize = u
    ↪'columns')
```

```
[ ]: substance    alcohol    cocaine    heroin      All
homeless
homeless    0.581921    0.388158    0.379032    0.461369
housed      0.418079    0.611842    0.620968    0.538631
```

The output provided proportions by substance. For example,  $103/177 = 0.582$  of all alcoholics are homeless, and  $93/152 = 0.612$  of all cocaine abusers are housed. Note that the conditional proportions above vary for each group, which implies association. In [Chapter 6](#), a formal chi-squared test for independence in contingency tables is derived.

```
[ ]:
```

### 2.2.3 Comparing numerical data across groups

Oftentimes, it is essential to consider numerical data broken into groups by one or more categorical variables. For example, we may want to investigate depression scores broken down by gender using `groupby()`.

```
[ ]: import pandas as pd;   import numpy as np;
import seaborn as sns;   import matplotlib.pyplot as plt;
url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrc.csv"
mydata = pd.read_csv(url)  # save as mydata file

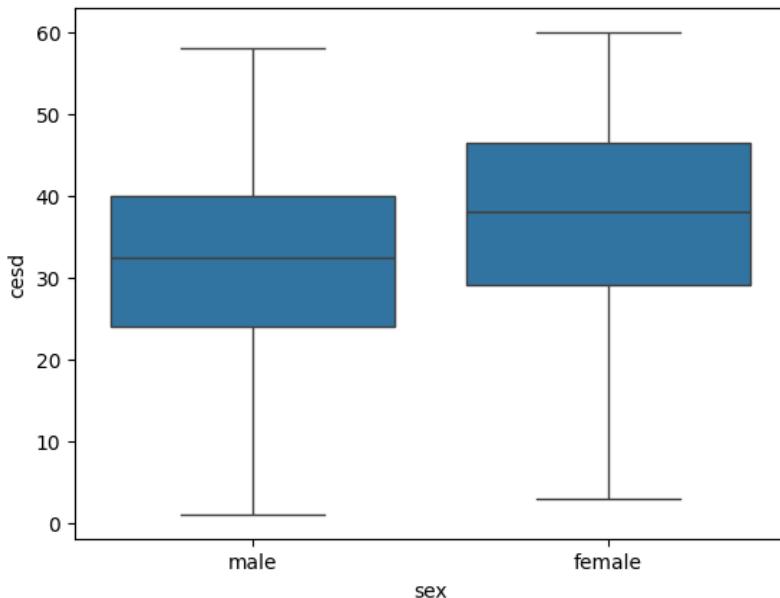
mydata.groupby('sex')['cesd'].describe()
```

	count	mean	std	min	25%	50%	75%	max
sex								
female	107.0	36.887850	13.017642	3.0	29.0	38.0	46.5	60.0
male	346.0	31.598266	12.103318	1.0	24.0	32.5	40.0	58.0

We can present graphical displays like boxplots, histograms, etc. broken down by categorical variables too.

```
[ ]: sns.boxplot(data=mydata,x='sex',y='cesd')
```

```
[ ]: <Axes: xlabel='sex', ylabel='cesd'>
```



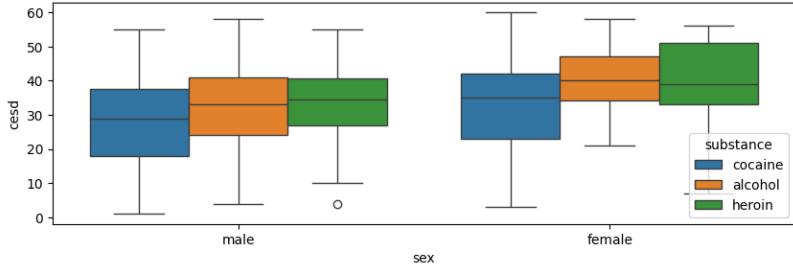
We can break up the `cesd` by more than one categorical variable - here `gender` and `substance` and represent the 2nd categorical variable by the hue of the graph:

```
[ ]: mydata.groupby(['sex','substance'])['cesd'].describe()
```

		count	mean	std	min	25%	50%	75%	max
female	alcohol	36.0	40.277778	9.808484	21.0	34.25	40.0	47.25	58.0
	cocaine	41.0	32.975610	14.487042	3.0	23.00	35.0	42.00	60.0
	heroin	30.0	38.166667	13.274511	7.0	33.00	39.0	51.00	56.0
male	alcohol	141.0	32.865248	12.134496	4.0	24.00	33.0	41.00	58.0
	cocaine	111.0	28.108108	12.791582	1.0	18.00	29.0	37.50	55.0
	heroin	94.0	33.819149	10.309159	4.0	27.00	34.5	40.75	55.0

```
[ ]: plt.figure(figsize=(10,3))
sns.boxplot(data=mydata,x='sex',y='cesd',hue='substance')
```

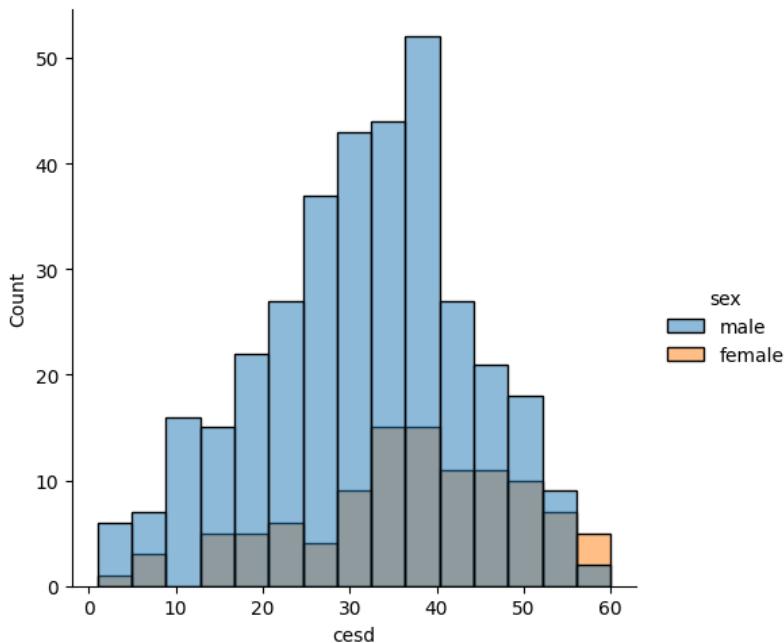
```
[ ]: <Axes: xlabel='sex', ylabel='cesd'>
```



On the other hand, two histograms on the same graph overlap each other and are harder to distinguish. It may be better to display them separately as shown in the two figures below. The option `col="sex"` creates two graphs in two columns.

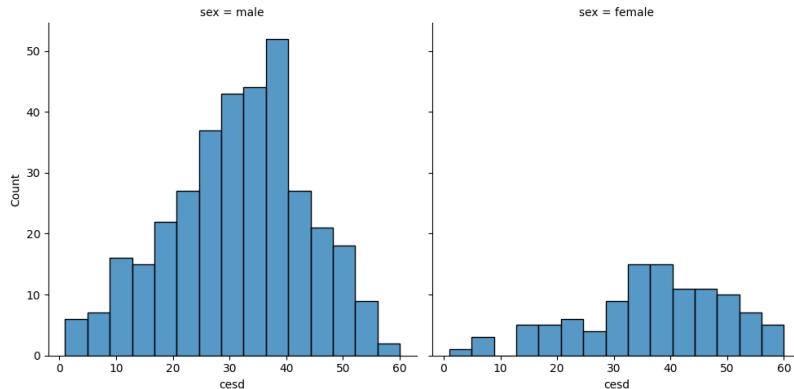
```
[ ]: sns.displot(mydata, x="cesd", hue="sex") #
```

```
[ ]: <seaborn.axisgrid.FacetGrid at 0x7f0a4f3b76a0>
```



```
[ ]: sns.displot(mydata, x="cesd", col="sex")
```

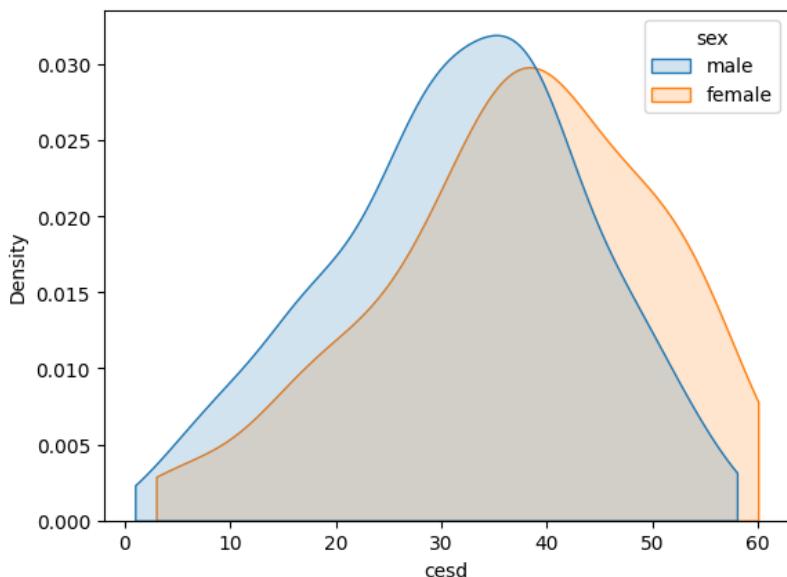
[ ]: <seaborn.axisgrid.FacetGrid at 0x7f0a1793b520>



The graph with two separate histograms for males and females is preferable. Alternatively, we can produce a density plot which is a smoothed version of a histogram as shown in the Figure below:

```
[ ]: sns.kdeplot(data=mydata, x="cesd", hue="sex", cut=0, fill=True,
                 common_norm=False, alpha=0.2)
# alpha provides transparency
```

[ ]: <Axes: xlabel='cesd', ylabel='Density'>



In this case, overlapping is less of a problem.



Taylor & Francis  
Taylor & Francis Group  
<http://taylorandfrancis.com>

# 3

---

## Probability

---

As mentioned in the book's roadmap, probability ideas must be developed to understand inferential statistics. Probability is a vital subject with many applications in Medicine, Biology, Genetics, Engineering, Business, Economics, etc. This chapter introduces basic concepts and additional techniques like probability simulations and applications of Bayes Theorem.

---

### 3.1 Basic Concepts of Probability

**Probability** deals with **random processes** giving rise to outcomes that cannot be predicted in advance. The simplest examples of such processes are flipping a coin or rolling a die, but random processes apply to far more complex situations like fluctuations in the stock market, genetic mutations, etc.

Let's begin the discussion of probability with a **sample space**, which is the set of all possible outcomes of an experiment. Any subset of the sample space is an **event**. A simple event consists of exactly one element of the sample space. For example, the sample space for tossing two coins is  $S = \{HH, HT, TH, TT\}$ . Any of the above outcomes would be a simple event. A more general event would be, say, tossing one head and one tail  $A = \{HT, TH\}$ . For equally likely outcomes, the probability could be defined simply as a ratio of the number of outcomes in the event of interest to the total number of simple events in the sample space. Then, the probability of one head and one tail is  $P(A) = 2/4 = 1/2$ .

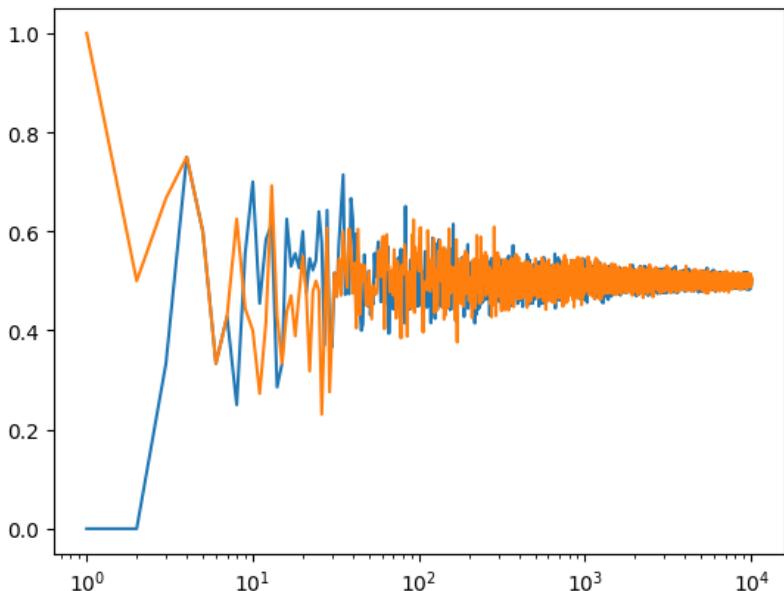
More generally, we use a frequentist approach where the probability of an outcome is the proportion of times the outcome would occur if the process is observed an *infinite number of times*. Let's consider a simulation of a large, but of course finite number of repetitions of tossing a fair coin. Let  $\hat{p}_n$  be the proportion of heads in the first  $n$  coin tosses. The Figure below shows two sequences of fair coin tosses. Initially, they are very different, but as the number of tosses increases,  $\hat{p}_n$  eventually converges to the theoretical probability  $p = 1/2$ . Note that after just a few tosses or even 10 or 20, the proportion still fluctuates a lot, only in a long run of many thousands of tosses,

$\hat{p}_n$  stabilizes around true theoretical probability. It is a common misconception that if you flip a fair coin 10 or 20 times, the proportion of heads should be close to half, but that is only achieved in the long run (large numbers). For the same reason, if the “red” came out a few times in a row at a casino table, it is *not* more likely for “black” to appear, they only equilibrate in a long run!

```
[ ]: import matplotlib.pyplot as plt
import numpy as np

def coinFlip(size):
    flips = np.random.randint(0, 2, size=size)
    return flips.mean()
coinFlip = np.frompyfunc(coinFlip, 1, 1)

x = np.arange(1, 10000, 1)
y1 = coinFlip(x)
y2 = coinFlip(x)
fig, ax = plt.subplots()
ax.plot(x, y1); ax.plot(x, y2);
ax.set_xscale('log')
```

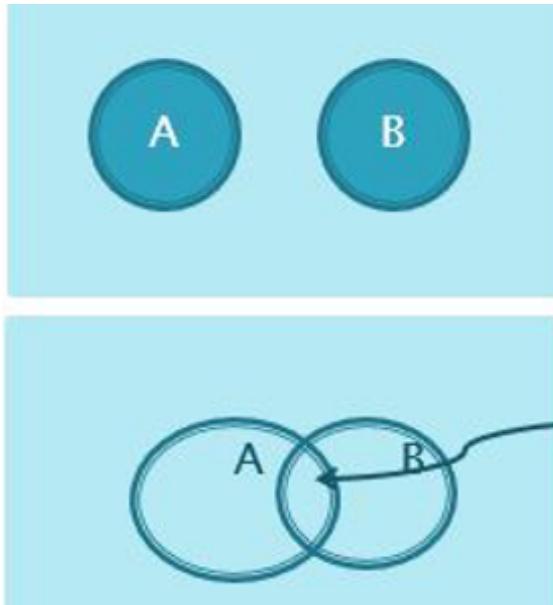


### 3.2 Addition Rule

The events are **disjoint (mutually exclusive)** if there are no elements in common. Venn diagrams help to understand disjoint vs. non-disjoint events (sets) as shown in the Figure below:

```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main/
→VennDiagram1.JPG'
page = requests.get(url); img = Image.open(BytesIO(page.content));
img.resize((400, 300))
```

[ ]:



For example, rolling an odd number on a die ( $\{1,3,5\}$ ) is disjoint from rolling an even number ( $\{2,4,6\}$ ). However, rolling a number less than 3 ( $\{1,2\}$ ) and rolling an even number ( $\{2,4,6\}$ ) are not disjoint because they share  $\{2\}$  (intersection is  $\{2\}$ ).

Given two **disjoint** events  $A_1$  and  $A_2$ , the probability that one occurs **or** the other is given by the **addition rule**:

$$P(A_1 \text{ or } A_2) = P(A_1) + P(A_2) \quad (3.1)$$

For example, the probability of picking a queen or a king out of a standard

deck of cards is:

$$P(Q \text{ or } K) = P(Q) + P(K) = \frac{4}{52} + \frac{4}{52} = \frac{8}{52} \approx 0.154$$

Consider now a non-disjoint case. For example, let event  $K$  be randomly selecting a king from a shuffled card deck ( $P(K) = \frac{4}{52}$ ) and event  $R$  be randomly selecting a red card from a deck ( $P(R) = \frac{26}{52}$ ).  $K$  and  $R$  are not disjoint (King of Hearts and King of Diamonds are in the intersection). If probabilities are simply added, the king of hearts and king of diamonds are double-counted  $P(K \text{ and } R) = \frac{2}{52}$ , so it should be subtracted:

$$P(K \text{ or } R) = P(K) + P(R) - P(K \text{ and } R) = \frac{4}{52} + \frac{26}{52} - \frac{2}{52} = \frac{28}{52} = \frac{7}{13}$$

Thus, we obtain the **General Addition Rule**:

Given two events  $A_1$  and  $A_2$  (disjoint or not):

$$P(A_1 \text{ or } A_2) = P(A_1) + P(A_2) - P(A_1 \text{ and } A_2) \quad (3.2)$$

In the case they are disjoint,  $P(A_1 \text{ and } A_2) = 0$  and we have a simple addition rule.

Here is another example relevant to college students. Assume 10% of students major in accounting (A), 15% major in business (B), and 3% are double majors (both A and B). What is the probability that a randomly chosen student is an accounting or a business major? The events are clearly not mutually exclusive because of double majors, so

$$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B) = .10 + .15 - .03 = .22$$

### 3.3 Simple Finite Probability Distributions

For the case of a simple finite sample space consisting of a fixed number of outcomes, a probability distribution is just a table of all possible outcomes and their associated probabilities. For example, for a fair coin, it is just:

x	Head (H)	Tail (T)
p	0.5	0.5

The coin need not be fair, so an equally valid probability distribution might be:

x	Head (H)	Tail (T)
p	0.6	0.4

Analogously, a fair die distribution would be given by

x	1	2	3	4	5	6
p	1/6	1/6	1/6	1/6	1/6	1/6

A fair die and coin are examples of **uniform** distribution where all outcomes are equally likely; an unfair coin is not.

The sum of outcomes while rolling two fair dice provides a very interesting example of non-uniform distribution. Possible sums are given by:

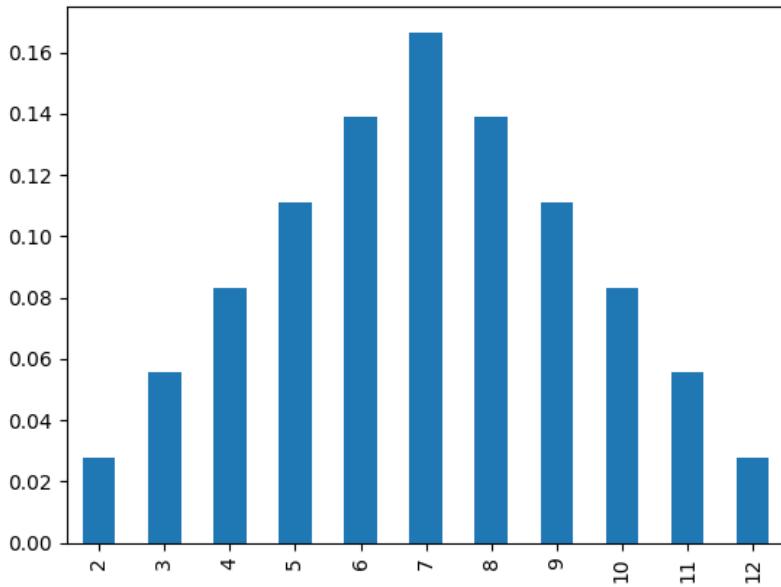
.	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

The probabilities are obtained by counting how many ways to get a particular sum out of all possible  $6 \cdot 6 = 36$  equally likely combinations:

```
[ ]: import numpy as np; import pandas as pd
x = np.arange(start=2,stop=13,step=1);
p = np.array([1,2,3,4,5,6,5,4,3,2,1])/36;
df = pd.DataFrame({'x':x, 'p':p}); print(df)
df.index = ["2","3","4","5","6","7","8","9","10","11","12"]
df['p'].plot(kind='bar')
```

	x	p
0	2	0.027778
1	3	0.055556
2	4	0.083333
3	5	0.111111
4	6	0.138889
5	7	0.166667
6	8	0.138889
7	9	0.111111
8	10	0.083333
9	11	0.055556
10	12	0.027778

[ ]: <Axes: >



To be valid, a probability distribution must satisfy these three properties:

1. All finite outcomes must be disjoint.
2. All probabilities are between 0 and 1.
3. The probabilities add up to 1.

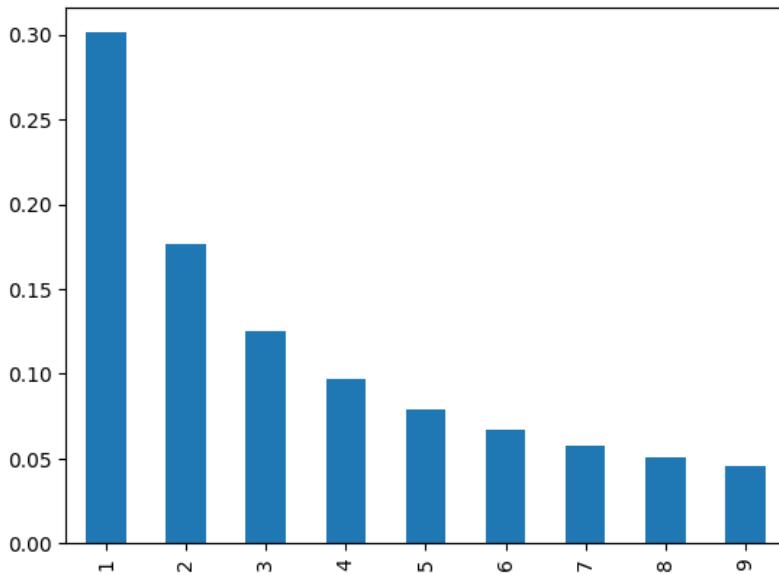
Another interesting finite discrete distribution is *Benford's law*: the distribution of the leading (first digit) in many real data sets is not uniform, but rather heavily skewed right. As the Figure below shows, "1" appears as the leading significant digit about 30% of the time, "2" 17.6%, ..., "9" less than 5% of the time. Financial data follows this law quite well, so the IRS flags tax returns failing to follow this law as a possible fraud.

```
[ ]: import numpy as np; import pandas as pd
x = np.arange(start=1,stop=10,step=1);
p = np.log(1+1/x)/np.log(10)
df = pd.DataFrame({'x':x, 'p':p})
df.index = ["1","2","3","4","5","6","7","8","9"]
print(df)
df['p'].plot(kind='bar')
```

	x	p
1	1	0.301030
2	2	0.176091
3	3	0.124939

4	4	0.096910
5	5	0.079181
6	6	0.066947
7	7	0.057992
8	8	0.051153
9	9	0.045757

[ ]: <Axes: >



### 3.4 Complement of an Event

A **complement** of an event  $A$  (denoted by  $A^c$ ) consists of all outcomes in the sample space that are not in  $A$ . For example, let the event  $A$  be rolling any of the 1 through 4  $A = \{1, 2, 3, 4\}$ , then  $A^c = \{5, 6\}$ . Note that  $P(A) = \frac{4}{6}$  and  $P(A^c) = \frac{2}{6}$ , which add up to 1. Based on the definition above,  $A$  and  $A^c$  cover the entire sample space and are clearly disjoint, so:

$$1 = P(S) = P(A \text{ or } A^c) = P(A) + P(A^c) \implies P(A) = 1 - P(A^c)$$

This simple algebraic formula is surprisingly useful for many complex probability questions. It is often hard or impossible to find the probability of the event, but the probability of the complement can be found and subtracted

from 1 to answer the original question.

For rolling two dice, say, what is the probability that the sum is 9 or less? One can count directly, but the complement rule is more efficient:

$$P(\text{sum} \leq 9) = 1 - P(\text{sum} \geq 10) = 1 - \frac{6}{36} = \frac{30}{36} = \frac{5}{6}$$

There are more applications of the complement rule in future sections, but first independence must be discussed.

---

### 3.5 Independence

Random processes or events are **independent** if the occurrence of one event does not affect the occurrence of the other. Sometimes it is obvious - one die roll does not affect another, and one coin flip does not affect another. Other times it requires research; for example, it has been shown that eye color is independent of gender, but it was not obvious to start with. On the other hand, it is well known by insurance companies, that age and rate of accidents are not independent. Younger, less experienced, and more reckless drivers get into accidents more often and, as a result, are charged higher insurance premiums. Stock prices day to day are not independent of each other, but the nature of this dependence is still a mystery.

The **Multiplication Rule for Independent Events (Processes)**  $A$  and  $B$  states that probability of both events happening is the product of the probabilities of each event happening individually:

$$P(A \text{ and } B) = P(A) \cdot P(B) \tag{3.3}$$

For example, about 27% of the US population has blue eyes. Suppose two people are selected at random independently from each other. The probability that both have blue eyes is:

$$P(B \text{ and } B) = P(B) \cdot P(B) = 0.27 \cdot 0.27 = 0.0729$$

The probability that at least one of the two randomly chosen people has blue eyes is best found using the complement rule:

$$\begin{aligned} P(\text{at least one blue eyes}) &= 1 - P(\text{none}) = 1 - P(\text{both not blue}) = \\ &= 1 - 0.73 \cdot 0.73 = 0.4671 \end{aligned}$$

In the case of only two people, the probability of at least one blue-eyed could have been computed directly, as there are only four total options

$\{\text{BB}, \text{BN}, \text{NB}, \text{NN}\}$  of which three have at least one blue-eyed person (still you would have to compute three products and add them). If we have three people there are  $2^3 = 8$  total options  $\{\text{BBB}, \text{BBN}, \text{BNB}, \text{BNN}, \text{NBB}, \text{NBN}, \text{NNB}, \text{NNN}\}$ , and there are seven combinations in which at least one person has blue eyes. The complement is much easier to compute.

$$P(\text{at least one blue eyes}) = 1 - P(\text{none}) = 1 - P(\text{NNN}) = 1 - 0.73^3 = 0.61098$$

Finally, for a larger number of people, for example 10, using the complement is the only option:

$$P(\text{at least one blue eyes}) = 1 - P(\text{none}) = 1 - 0.73^{10} = 0.9570$$

We will return to the questions of this type in the next chapter when we study Binomial distribution.

Independence can also be used in the classic problem of redundant alarm clocks. Assume a student has a very important exam and sets two *independent* alarm clocks. Each clock has a probability of working 0.99 (fails to work  $1 - 0.99 = 0.01$ ). Then

$$P(\text{at least one works}) = 1 - P(\text{both fail}) = 1 - 0.01 \cdot 0.01 = 0.9999$$

Thus, the chance that at least one works and a student wakes up is much higher.

Note that the multiplication rule is often used as a definition of independence, so it can be used to establish independence. For example, consider picking two cards out of a well-shuffled deck at random. The probability that they are “red kings” is

$$P(\text{Red and Kings}) = \frac{2}{52}$$

Consider also

$$P(\text{Red}) \cdot P(\text{Kings}) = \frac{26}{52} \cdot \frac{4}{52} = \frac{104}{52 \cdot 52} = \frac{2}{52}$$

These probabilities are the same, so picking red cards and picking kings at random is independent, which was not obvious.

Note also that we should not confuse independent events and disjoint events. Compare the above with computations we have done before for the probability of picking a king **OR** a red card (*Addition Rule*):

$$P(K \text{ or } R) = P(K) + P(R) - P(K \text{ and } R) = \frac{4}{52} + \frac{26}{52} - \frac{2}{52} = \frac{28}{52} = \frac{7}{13}$$

The probability  $P(K \text{ and } R) = \frac{2}{52} \neq 0$ , so these events are not disjoint, but they are independent. It is a common misconception that mutual exclusiveness (disjoint events) and independence are the same. On the contrary, for

any nontrivial disjoint events  $A$  and  $B$ , independence cannot even hold.

$$P(A \text{ and } B) = 0 \neq P(A) \cdot P(B)$$

since  $P(A)$  and  $P(B)$  are not 0.

### *Birthday Problem*

This is a classical problem in probability that illustrates ideas of complement and independence. Suppose that  $k$  people are selected at random from the general population. What are the chances that at least two of those  $k$  were born on the same day?

“At least two” gives way too many options, but the complement (none) is much easier to track.

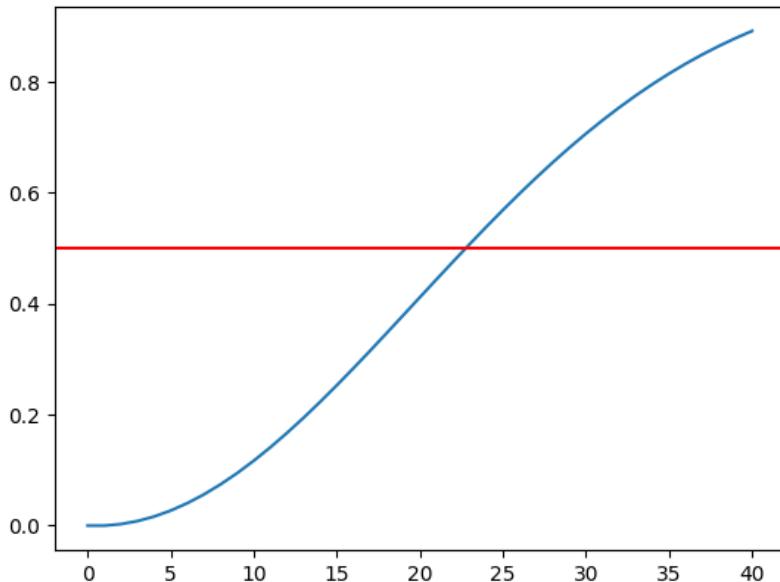
$$P(\text{at least two same bday}) = 1 - P(\text{none}) = 1 - \frac{365}{365} \frac{364}{365} \frac{363}{365} \cdots \frac{365 - k + 1}{365}$$

To compute it for general  $k$ , we need a program that iteratively multiplies the probabilities for 1st person to “choose” a day times independently 2nd person to “choose” a day not the same as the 1st, etc. The iterative computation is necessary since computing powers of 365 quickly exceed highest number that a computer can store.

```
[ ]: import numpy as np; import pandas as pd
import matplotlib.pyplot as plt
kmax = 40;
kv = np.arange(0,kmax+1)
pv = np.repeat(1.0,kmax+1)
for j in range(2, kmax+1):
    pv[j] = pv[j-1]*(365-j+1)/365

pv = 1 - pv # complement
df = pd.DataFrame({'class_size':kv, 'probability': pv});
# print(df)
plt.plot(kv,pv)
plt.axhline(y = 0.5, color = 'r', linestyle = '-')
```

```
[ ]: <matplotlib.lines.Line2D at 0x7eb397a1ee30>
```



The Figure above shows that the probabilities greatly exceed what our intuition would suggest. It is more than a 50% chance for a class of 23 already, and by 40, it is almost 90%.

---

### 3.6 Conditional Probability

Sometimes, we know that a certain event has already occurred and it may have a bearing on the probability we are trying to find - **conditional probability**. For instance, consider rolling a fair die and the event of rolling 3 or less:  $A = \{1,2,3\}$ . The sample space is  $\{1,2,\dots,6\}$ , and they are all equally likely, so  $P(A) = \frac{3}{6} = \frac{1}{2}$ . But suppose someone watched the die roll and tells you, not the exact outcome, but that an odd number appeared. Denote it as the event  $B = \{1, 3, 5\}$ . This changes sample space to smaller *conditional sample space*  $B = \{1,3,5\}$ , still equally likely. Therefore, the updated **conditional probability**  $P(A|B) = \frac{2}{3}$ .

Note that

$$P(A|B) = \frac{2}{3} = \frac{\frac{2}{6}}{\frac{3}{6}} = \frac{P(A \text{ and } B)}{P(B)}$$

The equation

$$P(A|B) = \frac{P(A \text{ and } B)}{P(B)} \quad (3.4)$$

is actually used as the **definition of the conditional probability**.

As a more practical example, a car insurance company will consider information about a person's driving history (conditioned on past history) to assess the probability of an accident.

Conditional probability is also involved in the computations of sampling with and without replacement. If a person/item cannot be picked more than once, it is sampling **without replacement**. Conversely, if a person or item is put back, it is sampling **with replacement**.

For example, what is the probability of picking an Ace first and a Queen second vs. the probability of picking two Aces?

First, without replacement:

$$\begin{aligned} P(\text{Ace first and Queen second}) &= \frac{4}{52} \cdot \frac{4}{51} \\ P(\text{Ace first and Ace second}) &= \frac{4}{52} \cdot \frac{3}{51} \end{aligned}$$

Second, with replacement:

$$\begin{aligned} P(\text{Ace first and Queen second}) &= \frac{4}{52} \cdot \frac{4}{52} = \left(\frac{4}{52}\right)^2 \\ P(\text{Ace first and Ace second}) &= \frac{4}{52} \cdot \frac{4}{52} = \left(\frac{4}{52}\right)^2 \end{aligned}$$

Note that in sampling without replacement, the 2nd probability is conditional, while in sampling with replacement, all probabilities are independent and actually the same.

### 3.7 Contingency Tables Probability

Conditional probability is very useful in the context of contingency tables.

#### Example

Consider a marketing study on social class and purchasing habits for two brands A and B. The code below to display data frame with margins is a bit technical.

	Lower	Middle	Upper	Sum
Brand A	2	22	21	45
Brand B	24	28	3	55
Sum	26	50	24	100

```
[ ]: import pandas as pd; import numpy as np;
df = pd.DataFrame({})

nr = 2; nc = 3;
O = pd.DataFrame({'Lower':[2,24], 'Middle':[22,28], 'Upper':[21,3]},
                 index= ['Brand A','Brand B'])
print(O)
Om = pd.DataFrame.copy(O)    # creating data frame with margins
Om['Sum'] = Om.sum(axis=1);
c = Om.sum(axis=0);
Om.loc[len(Om.index)] = c.tolist() + [Om.sum().sum()]
print(Om)
```

	Lower	Middle	Upper	
Brand A	2	22	21	
Brand B	24	28	3	
	Lower	Middle	Upper	Sum
Brand A	2	22	21	45
Brand B	24	28	3	55
2	26	50	24	100

We already encountered in the previous chapter contingency tables based on data. Here, we are given counts in each contingency pair directly. In this example, there are, say, 2 in the Lower class who insist on buying more expensive Brand A, and 28 in the Middle class who are ok with a less fancy Brand B. The row and column totals are given. The total number of people who preferred Brand A is  $2 + 22 + 21 = 45$ , regardless of social class, and the total number of the middle class is  $22 + 28 = 50$ . The grand total is 100. Every entry divided by the grand total gives overall probabilities. For example:

$$P(\text{Brand } B \text{ and Lower}) = \frac{24}{100} = 0.24$$

$$P(\text{Brand } A \text{ and Middle}) = \frac{22}{100} = 0.22$$

The probabilities above are **joint probabilities** connecting events with **and**. On the other hand, **marginal probabilities** are row and column total probabilities for each variable separately:

$$P(\text{Brand } B) = \frac{55}{100} = 0.55$$

$$P(Lower) = \frac{26}{100} = 0.26$$

However, **row and column (conditional) probabilities** are more informative.

First, row probabilities:

$$P(Lower | Brand A) = \frac{2}{45} = 0.0444$$

$$P(Middle | Brand A) = \frac{22}{45} = 0.489$$

Column probabilities:

$$P(Brand A | Lower) = \frac{2}{26} = 0.0769$$

$$P(Brand A | Middle) = \frac{22}{50} = 0.44$$

Note above that switching the order of conditioning produces different probabilities  $P(Lower | Brand A) \neq P(Brand A | Lower)$ .

A ratio of probabilities formula can also be used to find any of the conditional probabilities. For example:

$$P(Brand B | Upper) = \frac{3}{24} = \frac{\frac{3}{100}}{\frac{24}{100}} = \frac{P(Brand B and Upper)}{P(Upper)} = 0.125$$

The answer is the same, but it is more work. However, it is needed when the full contingency table is not given.

The main question in a contingency table is whether row and column variables depend on each other. Let's compare conditional and marginal probabilities:

$$P(Brand A | Lower) = \frac{2}{26} = 0.077$$

$$P(Brand A | Middle) = \frac{22}{50} = 0.44$$

$$P(Brand A | Upper) = \frac{21}{24} = 0.875$$

$$P(\text{Brand A}) = \frac{45}{100} = 0.45$$

These probabilities are very different, so the row variable (Brand) is dependent on the column variable (Social Class). What if they were close, but not exactly the same? The chi-squared test in [chapter 6](#) will answer how close is close enough for independence.

Let's also check if the joint probability is the product of marginal ones:

$$P(\text{Brand A and Lower}) ?=? P(\text{Brand A}) \cdot P(\text{Lower})$$

$$\frac{2}{100} ?=? \frac{45}{100} \cdot \frac{26}{100} \implies \\ 0.02 ?=? 0.45 \cdot 0.26 = 0.117$$

No it is not, so that proves dependence in another way.

Finally, let's illustrate the Addition Rule for a contingency table.

$$P(\text{Brand A or Upper}) = P(\text{Brand A}) + P(\text{Upper}) - P(\text{Brand A and Upper}) = \\ \frac{45}{100} + \frac{24}{100} - \frac{21}{100} = \frac{48}{100}$$

### Example

Let's now consider a similar problem on a two-by-two contingency table on Dove soap usage by gender. Such tables are very common and calling entries  $a, b, c, d$ , we can set up the table and answer any probability questions in terms of ratios, which is done in the Python code below:

	Male	Female	Sum
Dove Yes	a=80	b=120	a+b=200
Dove No	c=320	d=480	c+d=800
Sum	a+c=400	b+d=600	a+b+c+d=1000

```
[ ]: # Contingency table for Dove Soap
import pandas as pd; import numpy as np;
a = 80; b = 120;
c = 320; d = 480;
Om = np.matrix( [[a, b, a+b],
                 [c, d, c+d],
                 [a+c, b+d, a+b+c+d]]);
Om = pd.DataFrame(Om, columns=['Male', 'Female', 'Sum'],
                  index=['DoveYes', 'DoveNo', 'Sum']); Om
```

```
[ ]:      Male  Female  Sum
DoveYes    80     120   200
DoveNo     320     480   800
Sum        400     600  1000
```

```
[ ]: JointMarginal = Om.div(a+b+c+d); JointMarginal
```

```
[ ]:      Male  Female  Sum
DoveYes  0.08     0.12   0.2
DoveNo   0.32     0.48   0.8
Sum      0.40     0.60   1.0
```

```
[ ]: print('P(Male | DoveYes) = a/(a+b): ', a/(a+b))
print('P(Female | DoveNo) = d/(c+d): ', d/(c+d))
print('P(DoveYes | Male) = a/(a+c): ', a/(a+c))
print('P(DoveNo | Female) = d/(b+d): ', d/(b+d))

print('\nDependence check')
print('Two conditional probabilities and the marginal:')
print('P(DoveYes | Male) = a/(a+c): ', a/(a+c))
print('P(DoveYes | Female) = b/(b+d): ', b/(b+d))
print('P(DoveYes) = (a+b)/(a+b+c+d): ', (a+b)/(a+b+c+d))

print('\nJoin probability vs. product of marginal ones')
print('P(DoveYes and Male) = a/(a+b+c+d): ', a/(a+b+c+d))
print('P(DoveYes)=(a+b)/(a+b+c+d): ', (a+b)/(a+b+c+d))
print('P(Male)=(a+c)/(a+b+c+d): ', (a+c)/(a+b+c+d))
print('P(DoveYes) * P(Male) = (a+b)/(a+b+c+d)*(a+c)/(a+b+c+d): ', (a+b)/(a+b+c+d)*(a+c)/(a+b+c+d))
```

```
P(Male | DoveYes) = a/(a+b):  0.4
P(Female | DoveNo) = d/(c+d):  0.6
P(DoveYes | Male) = a/(a+c):  0.2
P(DoveNo | Female) = d/(b+d):  0.8
```

```
Dependence check
Two conditional probabilities and the marginal:
P(DoveYes | Male) = a/(a+c):  0.2
P(DoveYes | Female) = b/(b+d):  0.2
P(DoveYes) = (a+b)/(a+b+c+d):  0.2
```

```
Join probability vs. product of marginal ones
P(DoveYes and Male) = a/(a+b+c+d):  0.08
P(DoveYes)=(a+b)/(a+b+c+d):  0.2
P(Male)=(a+c)/(a+b+c+d):  0.4
P(DoveYes) * P(Male) = (a+b)/(a+b+c+d)*(a+c)/(a+b+c+d):  0.08
```

The dependence check shows that the gender and Dove soap usage are independent of each other.

We can also illustrate the Addition Rule. For example:

$$P(\text{DoveYes or Male}) = P(\text{DoveYes}) + P(\text{Male}) - P(\text{DoveYes and Male})$$

```
[ ]: print('P(DoveYes or Male) = (a+b)/(a+b+c+d) + (a+c)/(a+b+c+d) - a/(a+b+c+d) :')
    ↪ ,
    (a+b)/(a+b+c+d), ' + ', (a+c)/(a+b+c+d), ' - ', a/(a+b+c+d), ' = ',
    np.round((a+b)/(a+b+c+d) + (a+c)/(a+b+c+d) - a/(a+b+c+d), 2) )
```

$$P(\text{DoveYes or Male}) = (a+b)/(a+b+c+d) + (a+c)/(a+b+c+d) - a/(a+b+c+d) : 0.2 + 0.4 - 0.08 = 0.52$$

## Example

In the next problem, we consider a similar two-by-two contingency table of a new scan technique to check for muscle tear against the “gold standard” (it could be a standard scan or surgery). In addition, we introduce standard terminology for these types of problems.

	Has Tear	No Tear	Sum
Test Positive	a=47 (True Positive)	b=5 (False Positive)	a+b=52 (Total Positive)
Test Negative	c=2 (False Negative)	d=46 (True Negative)	c+d=48 (Total Negative)
Sum	a+c=49 (Total Condition)	b+d=51 (Total No Condition)	a+b+c+d=100 (Grand Total)

The gold standard (“truth”) is in the columns and new test results are in the rows, although it does not matter - just be consistent. The table shows standard terminology. The patients who have the condition and tested positive are called **True Positive**, and those who tested negative are called **False Negative**. The patients who don’t have the condition and tested positive are called **False Positive**, and those who tested negative are called **True Negative**. The row totals give the total number of positive and negative tests respectively, while column totals give the total number of patients with and without the condition. There is also the grand total in the lower right corner. First, the code below sets up the contingency table, and the overall probabilities are computed via dividing each entry by the grand total of 100:

```
[ ]: import pandas as pd; import numpy as np; import matplotlib.pyplot as plt
a = 47;    # true positive
b = 5;     # false positive
c = 2;     # false negative
d = 46;    # true negative
Om = np.matrix( [[a ,b ,a+b],
                 [ c ,d ,c+d],
                 [ a+c,b+d,a+b+c+d ]]);
Om = pd.DataFrame(Om,columns=['Had Tear','No Tear','Sum'],
                  index=['Test Positive','Test Negative','Sum']); Om
```

[ ]: Had Tear No Tear Sum  
Test Positive 47 5 52

Test	Negative	2	46	48
Sum		49	51	100

```
[ ]: JointMarginal = Om.div(a+b+c+d); JointMarginal
```

	Had Tear	No Tear	Sum
Test Positive	0.47	0.05	0.52
Test Negative	0.02	0.46	0.48
Sum	0.49	0.51	1.00

This gives the following *joint* probabilities:

$$P(\text{has tear and test positive}) = \frac{47}{100} = 0.47$$

$$P(\text{has tear and test negative}) = \frac{2}{100} = 0.02$$

$$P(\text{no tear and test positive}) = \frac{5}{100} = 0.05$$

$$P(\text{no tear and test negative}) = \frac{46}{100} = 0.46$$

The *marginal* probabilities:

$$P(\text{has tear}) = \frac{49}{100} = 0.49$$

$$P(\text{no tear}) = \frac{51}{100} = 0.51$$

$$P(\text{test positive}) = \frac{52}{100} = 0.52$$

$$P(\text{test negative}) = \frac{48}{100} = 0.48$$

Next, we set row probabilities with new test terminology.

First, **positive predictive value** is the probability that a person who tests positive indeed has a condition  $P(\text{has tear}|\text{test positive})$ .

```
[ ]: print('\nPos Predict Value = a/(a+b) = ',a,'/',a+b,' = ', a/(a+b))
```

```
Pos Predict Value = a/(a+b) = 47 / 52 = 0.9038461538461539
```

The high positive predictive values close to 100% is important for a useful test. A value close to 50/50 indicates a useless (uninformative) test.

Similarly, **negative predictive value** is the probability that the person who tests negative indeed has no condition  $P(\text{no tear}|\text{test negative})$ .

```
[ ]: print('Neg Predict Value = d/(c+d) = ',d,'/',c+d,' = ', d/(c+d))
```

Neg Predict Value = d/(c+d) = 46 / 48 = 0.9583333333333334

Now consider column proportions.

The **sensitivity** (true positive rate) is the probability that the person who has a condition tests positive  $P(\text{test positive}|\text{has tear})$ .

```
[ ]: print('Sensitivity = a/(a+c) = ',a,'/',a+c,' = ', a/(a+c))
```

Sensitivity = a/(a+c) = 47 / 49 = 0.9591836734693877

The **specificity** (true negative rate) is the probability that the person who has no condition tests negative  $P(\text{test negative}|\text{no tear})$ .

```
[ ]: print('Specificity = d/(b+d) = ',d,'/',b+d,' = ', d/(b+d))
```

Specificity = d/(b+d) = 46 / 51 = 0.9019607843137255

Finally, the overall accuracy of the test is defined as the proportion correct results:

```
[ ]: print('P(Correct Test) = (a+d)/(a+b+c+d) = (' ,a,'+',d,')/',a+b+c+d,' = '
    ↪',(a+d)/(a+b+c+d))
```

P(Correct Test) = (a+d)/(a+b+c+d) = ( 47 + 46 )/ 100 = 0.93

We can summarize all Python steps above into a single function.

```
[ ]: def ProbabilityTable22(a,b,c,d,rnames,cnames):
    import pandas as pd; import numpy as np;

    Om = np.matrix( [[a ,b ,a+b],
                    [ c ,d ,c+d],
                    [ a+c,b+d,a+b+c+d ]]);
    Om = pd.DataFrame(Om,columns=[cnames[0],cnames[1],'Sum'],
                      index =[rnames[0],rnames[1],'Sum']);

    print(Om, '\n')
    JointMarginal = round(Om.div(a+b+c+d),4);
    print('Joint and Marginal Probabilities: \n\n',JointMarginal)

    print('\nPos Predict Value = P({:s}|{:s}) = a/(a+b) = {:d}/{:d} = {:.4f}'
        .format(cnames[0],rnames[0],a,a+b,a/(a+b)))
    print('\nNeg Predict Value = P({:s}|{:s}) = d/(c+d) = {:d}/{:d} = {:.4f}'
        .format(cnames[1],rnames[1],d,c+d,d/(c+d)))
    print('\nSensitivity = P({:s}|{:s}) = a/(a+c) = {:d}/{:d} = {:.4f}'
        .format(rnames[0],cnames[0],a,a+c,a/(a+c)))
    print('\nSpecificity = P({:s}|{:s}) = d/(b+d) = {:d}/{:d} = {:.4f}'
        .format(rnames[1],cnames[1],d,b+d,d/(b+d)))
```

```

print('\nP(Correct Test) = (a+d)/(a+b+c+d) = {:.d}/{:.d} = {:.4f}'.format(a+d,a+b+c+d,(a+d)/(a+b+c+d)))
ProbabilityTable22(a=47,b=5,c=2,d=46,rnames=['Test Positive','Test Negative'],
cnames=['Had Tear','No Tear'])

```

	Had Tear	No Tear	Sum
Test Positive	47	5	52
Test Negative	2	46	48
Sum	49	51	100

Joint and Marginal Probabilities:

	Had Tear	No Tear	Sum
Test Positive	0.47	0.05	0.52
Test Negative	0.02	0.46	0.48
Sum	0.49	0.51	1.00

$$\text{Pos Predict Value} = P(\text{Had Tear}|\text{Test Positive}) = a/(a+b) = 47/52 = 0.9038$$

$$\text{Neg Predict Value} = P(\text{No Tear}|\text{Test Negative}) = d/(c+d) = 46/48 = 0.9583$$

$$\text{Sensitivity} = P(\text{Test Positive}|\text{Had Tear}) = a/(a+c) = 47/49 = 0.9592$$

$$\text{Specificity} = P(\text{Test Negative}|\text{No Tear}) = d/(b+d) = 46/51 = 0.9020$$

$$P(\text{Correct Test}) = (a+d)/(a+b+c+d) = 93/100 = 0.9300$$

## Example

In the previous problem on the new scan for muscle tear diagnostics, the number of patients with and without the muscle tear was about the same. Sometimes, however, we are testing for a rare condition/infection. For example, for the infection disease data below, we apply the same function to compute all the probabilities.

	Has Infection	No Infection	Sum
Test Positive	a=90 (True Positive)	b=625 (False Positive)	a+b=715 (Total Positive)
Test Negative	c=10 (False Negative)	d=2575 (True Negative)	c+d=2585 (Total Negative)
Sum	a+c=100 (Total Condition)	b+d=3200 (Total No Condition)	a+b+c+d=3300 (Grand Total)

```
[ ]: ProbabilityTable22(a=90,b=625,c=10,d=2575,rnames=['Test Positive','Test Negative'], cnames=['Has Infection','No Infection'])
```

	Has Infection	No Infection	Sum
Test Positive	90	625	715
Test Negative	10	2575	2585
Sum	100	3200	3300

Joint and Marginal Probabilities:

	Has Infection	No Infection	Sum
Test Positive	0.0273	0.1894	0.2167
Test Negative	0.0030	0.7803	0.7833
Sum	0.0303	0.9697	1.0000

Pos Predict Value =  $P(\text{Has Infection}|\text{Test Positive}) = a/(a+b) = 90/715 = 0.1259$

Neg Predict Value =  $P(\text{No Infection}|\text{Test Negative}) = d/(c+d) = 2575/2585 = 0.9961$

Sensitivity =  $P(\text{Test Positive}|\text{Has Infection}) = a/(a+c) = 90/100 = 0.9000$

Specificity =  $P(\text{Test Negative}|\text{No Infection}) = d/(b+d) = 2575/3200 = 0.8047$

$P(\text{Correct Test}) = (a+d)/(a+b+c+d) = 2665/3300 = 0.8076$

The Specificity and Sensitivity are quite high, but the Positive Predictive Value is very low. It does not necessarily invalidate the test, rather the infection has a low **prevalence** - the proportion of the population with the condition is only about 3% here. Retesting for positive cases might be in order or, if possible, focusing on subgroups of the population with a higher occurrence of the condition.

### Example

Here is another practical application. A firm introduces a new lie detector test and advertises it to the Police, FBI, etc. Here are the results of the test run on 100 people divided about equally between actors who lied and who did not lie.

```
[ ]: ProbabilityTable22(a=44,b=16,c=8,d=32,rnames=['Test Positive','Test Negative'], cnames=['Lied','Not Lied'])
```

	Lied	Not Lied	Sum
Test Positive	44	16	60
Test Negative	8	32	40
Sum	52	48	100

Joint and Marginal Probabilities:

	Lied	Not Lied	Sum
Test Positive	0.44	0.16	0.6
Test Negative	0.08	0.32	0.4
Sum	0.52	0.48	1.0

Pos Predict Value =  $P(\text{Lied}|\text{Test Positive}) = a/(a+b) = 44/60 = 0.7333$

Neg Predict Value =  $P(\text{Not Lied}|\text{Test Negative}) = d/(c+d) = 32/40 = 0.8000$

Sensitivity =  $P(\text{Test Positive}|\text{Lied}) = a/(a+c) = 44/52 = 0.8462$

Specificity =  $P(\text{Test Negative}|\text{Not Lied}) = d/(b+d) = 32/48 = 0.6667$

$P(\text{Correct Test}) = (a+d)/(a+b+c+d) = 76/100 = 0.7600$

Overall, it is not very impressive, but you have to compare it to the competition.

[ ]:

### 3.8 General Multiplication Rule and Tree Diagrams

The definition of conditional probability given before

$$P(A|B) = \frac{P(A \text{ and } B)}{P(B)}$$

can be used to trivially solve for the joint probability  $P(A \text{ and } B)$  to obtain the **General Multiplication Rule**:

$$P(A \text{ and } B) = P(A|B) \cdot P(B) = P(B) \cdot P(A|B) \quad (3.5)$$

For the independent events, the conditional probability is the same as the original probability  $P(A|B) = P(A)$ , so we get independent events Multiplication Rule again  $P(A \text{ and } B) = P(A) \cdot P(B)$ .

It would be redundant, but the general multiplication rule can be illustrated for the muscle tear example:

	Has Tear	No Tear	Sum
Test Positive	a=47 (True Positive)	b=5 (False Positive)	a+b=52 (Total Positive)
Test Negative	c=2 (False Negative)	d=46 (True Negative)	c+d=48 (Total Negative)
Sum	a+c=49 (Total Condition)	b+d=51 (Total No Condition)	a+b+c+d=100 (Grand Total)

$$P(\text{test positive and has tear}) = P(\text{ has tear}) \cdot P(\text{test positive} \mid \text{has tear})$$

$$\frac{a}{a+b+c+d} = \frac{a+c}{a+b+c+d} \cdot \frac{a}{a+c}$$

$$\frac{47}{100} = \frac{49}{100} \cdot \frac{47}{49}$$

$$0.47 = 0.49 \cdot 0.959$$

**Tree diagrams** help to organize outcomes and their probabilities according to the conditional structure of the data. They are most useful when processes occur in a sequence and each process is conditioned on its predecessors. Let's first represent an infectious disease example as a tree. First, let's remind ourselves of the set-up below.

	Has Infection	No Infection	Sum
Test Positive	a=90 (True Positive)	b=625 (False Positive)	a+b=715 (Total Positive)
Test Negative	c=10 (False Negative)	d=2575 (True Negative)	c+d=2585 (Total Negative)
Sum	a+c=100 (Total Condition)	b+d=3200 (Total No Condition)	a+b+c+d=3300 (Grand Total)

In reality, we usually don't have a full contingency table, but rather we know the overall incidence of the disease:

$$P(\text{has infection}) = \frac{a+c}{a+b+c+d} = \frac{100}{3300} = 0.0303$$

As well as manufacturer provided Sensitivity and Specificity:

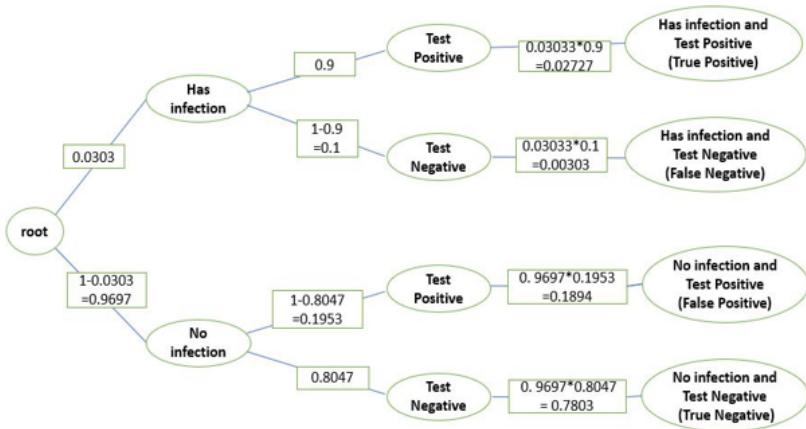
$$\text{Sensitivity} = P(\text{test positive} \mid \text{has infection}) = \frac{90}{100}$$

$$\text{Specificity} = P(\text{test negative} \mid \text{no infection}) = \frac{2575}{3200} = 0.8047$$

Based on the above information only, the probability tree below can be constructed:

```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main//'
→TreeInfecti.JPG'
page = requests.get(url); img = Image.open(BytesIO(page.content));
img.resize((600, 320))
```

[ ]:



A tree starts with a root. The next level gives the marginal probabilities of having an infection or not (add up to 1). The next column gives conditional probabilities of testing positive or negative given the previous branch of infection (use givensensitivity and specificity and the complement law). Finally, the last column lists joint probabilities as products of marginal and conditional:

$$P(\text{has infection and test positive}) =$$

$$P(\text{has infection}) \cdot P(\text{test positive}|\text{has infection}) = 0.0303 \cdot 0.9 = 0.02727$$

$$P(\text{has infection and test negative}) =$$

$$P(\text{has infection}) \cdot P(\text{test negative}|\text{has infection}) = 0.0303 \cdot 0.1 = 0.00303$$

$$P(\text{no infection and test positive}) =$$

$$P(\text{no infection}) \cdot P(\text{test positive}|\text{no infection}) = 0.9697 \cdot 0.19531 = 0.18939$$

$$P(\text{no infection and test negative}) =$$

$$P(\text{no infection}) \cdot P(\text{test negative}|\text{no infection}) = 0.9697 \cdot 0.80469 = 0.7803$$

Comparing it to the original table of joint and marginal probabilities, we see that we get the same answers.

More importantly, a probability tree can be used to find the reversed conditional probability  $P(\text{has infection} | \text{tested positive})$  - Positive Predictive Value

and  $P(\text{no infection} \mid \text{tested negative})$  - Negative Predictive Value.

The **Bayes Theorem** switches the conditioning:

$$P(\text{has infection} \mid \text{tested positive}) = \frac{P(\text{has infection and tested positive})}{P(\text{tested positive})} =$$

$$\frac{P(\text{infect}) \cdot P(\text{test pos|infect})}{P(\text{infect}) \cdot P(\text{test pos|infect}) + P(\text{no infect}) \cdot P(\text{test pos|no infect})}$$

Following the tree above, we obtain Positive Predictive Value:

$$P(\text{has infection} \mid \text{tested positive}) = \frac{0.0303 \cdot 0.9}{0.0303 \cdot 0.9 + 0.9697 \cdot 0.19531} = 0.12587$$

Negative Predictive Value:

$$P(\text{no infection} \mid \text{tested negative}) = \frac{0.9697 \cdot 0.80469}{0.9697 \cdot 0.80469 + 0.0303 \cdot 0.1} = 0.99613$$

These are exactly the conditional probabilities we obtained before.

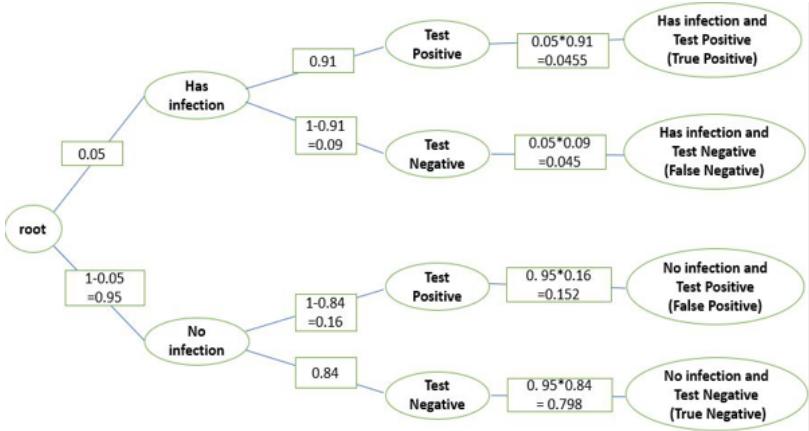
[ ]:

### Example

Let's consider a more realistic infectious disease problem where the contingency table is not given. Assume a manufacturer produced a new test for infectious disease with a prevalence of 0.05 ( $P(\text{randomly selected person has the disease})=0.05$ ). Also, the manufacturer claims that their test has Sensitivity =  $P(\text{test positive} \mid \text{has infection}) = 0.91$  and Specificity =  $P(\text{test negative} \mid \text{no infection}) = 0.84$ . Note that this formulation works for any new test scan or device; for example, we can assess the effectiveness of a new alarm system, lie detector test, etc. Note that another way to give conditional probabilities would be to say that the false negative rate is 9%, and the false positive rate is 16%. Based on this information we can create the following tree shown in the Figure below:

```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main//'
→TreeInfect2.JPG'
page = requests.get(url); img = Image.open(BytesIO(page.content)); img.
→resize((600, 320))
```

[ ]:



The **Bayes Theorem** gives us:

$$P(\text{has infection}|\text{tested positive}) = \frac{P(\text{has infection and tested positive})}{P(\text{tested positive})} =$$

$$\frac{P(\text{infect}) \cdot P(\text{test pos|infect})}{P(\text{infect}) \cdot P(\text{test pos|infect}) + P(\text{no infect}) \cdot P(\text{test pos|no infect})}$$

Following the tree, we obtain for Positive Predictive Value:

$$P(\text{has infection}|\text{tested positive}) = \frac{0.05 \cdot 0.91}{0.05 \cdot 0.91 + 0.95 \cdot 0.16} = 0.23038$$

Analogously for the Negative Predictive Value:

$$P(\text{no infection}|\text{tested negative}) = \frac{0.95 \cdot 0.84}{0.95 \cdot 0.84 + 0.05 \cdot 0.09} = 0.99439$$

The Positive Predictive Value is low again due to low prevalence.

### Example

Conditional trees and Bayes Theorem have many applications. In this example, a business application for quality control is investigated. Assume a computer manufacturer buys computer chips from firms A, B, and C with

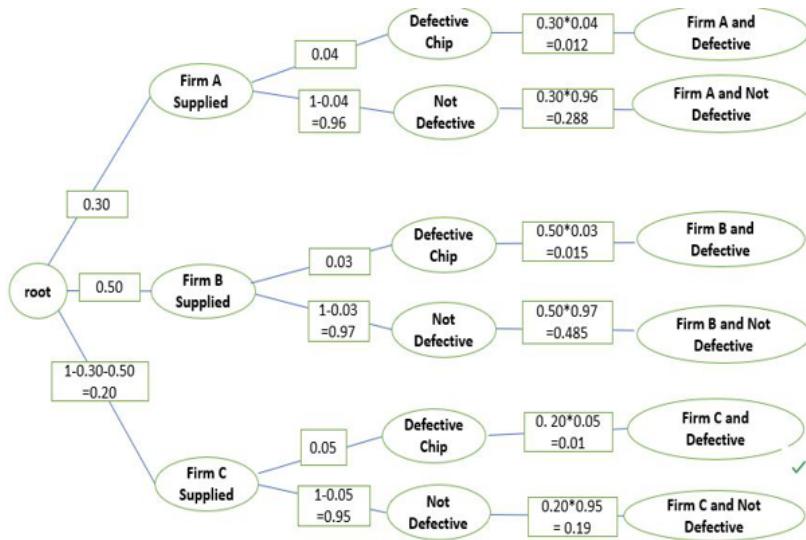
supply shares and defective percentages given in the table below. If a computer was returned with a defective chip, how likely is it that it came from each of the firms?

.	Firm A	Firm B	Firm C
Supply %	30	50	20
Defective %	4	3	5

Based on the supply percentages, the main branches of the probability tree below are created and the defective percentages are conditional probabilities for each supplier.

```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main//'
→TreeFirmChips.JPG'
page = requests.get(url); img = Image.open(BytesIO(page.content));
img.resize((600, 400))
```

[ ]:



The **Bayes Theorem** gives us:

$$P(\text{Firm A|defective}) = \frac{P(\text{Firm A and defective})}{P(\text{defective})} =$$

$$\frac{P(\text{Firm A}) \cdot P(\text{defective|Firm A})}{P(A) \cdot P(\text{defect|A}) + P(B) \cdot P(\text{defect|B}) + P(C) \cdot P(\text{defect|C})}$$

Following the tree we obtain:

$$P(\text{Firm A|defective}) = \frac{0.30 \cdot 0.04}{0.30 \cdot 0.04 + 0.50 \cdot 0.03 + 0.20 \cdot 0.05} = 0.3243$$

Analogously for Firm B and Firm C:

$$P(\text{Firm B|defective}) = \frac{0.50 \cdot 0.03}{0.30 \cdot 0.04 + 0.50 \cdot 0.03 + 0.20 \cdot 0.05} = 0.4054$$

$$P(\text{Firm C|defective}) = \frac{0.20 \cdot 0.05}{0.30 \cdot 0.04 + 0.50 \cdot 0.03 + 0.20 \cdot 0.05} = 0.2703$$

It is perhaps somewhat surprising given a defective chip, that it is most likely coming from firm B even though they have the smallest percentage of defectives. However, they supply most chips by far, 50%, compared to 30% and 20% by the two other firms.

[ ]:

### 3.9 Expected Value of Probability Distribution

We defined a simple finite probability distribution before as a table of all possible outcomes and their associated probabilities. In the previous chapter, we investigated measures of center and spread for the data in [Chapter 2](#). In this section, we introduce such measures for a probability distribution.

#### Example

Consider an example of placing a bet of \$10 on a single number in a casino roulette, say 27, but it does not matter which one. There are 38 equally likely slots on a roulette. Most likely you are going to lose your bet with probability  $\frac{37}{38}$ . In an unlikely event of winning with the probability  $\frac{1}{38}$ , the net gain is \$350. Therefore the probability distribution is:

Payout	−10	350
Probability	$\frac{37}{38}$	$\frac{1}{38}$

Let's assume you bet 100 times. You would expect to win on average  $\$350 \cdot 100 \cdot \frac{1}{38} = \$921.05$  and lose  $\$10 \cdot 100 \cdot \frac{37}{38} = \$973.68$  for the net expected gain of

$-\$52.63$ . Per game, this amounts to  $\frac{-\$52.63}{100} = -\$0.53$  dollars loss per game. Note that you cannot lose this amount in a single game, most of the time you lose your \$10 bets and occasionally win \$350. However, over 100 games, on average, you expect to lose this amount per game. Note that this amount does not depend on the number of bets you place:

$$\frac{\$350 \cdot 100 \cdot \frac{1}{38} - \$10 \cdot 100 \cdot \frac{37}{38}}{100} = \$350 \cdot \frac{1}{38} - \$10 \cdot \frac{37}{38}$$

Thus, we obtain the definition of the **Expected Value for a Probability Distribution**:

$$\mu = E(X) = \sum_{i=1}^n x_i \cdot P(X = x_i) = \\ x_1 \cdot P(X = x_1) + x_2 \cdot P(X = x_2) + \dots + x_n \cdot P(X = x_n)$$

It is the weighted sum of possible outcomes weighted by the probabilities (center of mass of probability weights located at x's).

The standard deviation for a data set was defined in [Chapter 2](#) using squared deviations from the mean:  $s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}}$ . Analogously, **probability distribution standard deviation** is defined using squared deviations from the expected value:

$$\sigma = \sqrt{\sum_{i=1}^n (x_i - \mu)^2 \cdot P(X = x_i)} = \\ \sqrt{(x_1 - \mu)^2 \cdot P(X = x_1) + \dots + (x_n - \mu)^2 \cdot P(X = x_n)}$$

The **variance** is the square of the standard deviation:

$$\sigma^2 = \sum_{i=1}^n (x_i - \mu)^2 \cdot P(X = x_i) = \\ (x_1 - \mu)^2 \cdot P(X = x_1) + \dots + (x_n - \mu)^2 \cdot P(X = x_n)$$

For the roulette probability distribution we obtain:

```
[ ]: import numpy as np
x = np.array([-10,350])
p = np.array([37/38,1/38])
mu = np.sum(x*p); sig = np.sqrt(np.sum((x-mu)**2*p));
print('Expected Value = {:.3f}, Standard Deviation = {:.3f}'.format(mu,sig))
print('mu +- 2sig = {:.3f}, {:.3f}'.format(mu-2*sig, mu+2*sig))
```

```
Expected Value = -0.526, Standard Deviation = 57.626
mu +- 2sig = -115.779, 114.726
```

$$\mu = E = -10 \cdot 0.974 + 350 \cdot 0.026 = -0.526$$

$$\sigma = \sqrt{(-10 - 0.526)^2 \cdot 0.974 + (350 - 0.526)^2 \cdot 0.026} = 57.626$$

In the Python code above, the bounds of the expected value plus/minus 2 standard deviations were included to illustrate the spread of the most likely results.

### Example

In the context of insurance, expected value refers to the average amount the company expects to pay to a customer. It is a key concept in risk management to determine the pricing of insurance premiums.

Consider auto insurance with a premium of \$800 per year for coverage against accidents. Based on the historical data, the following probabilities and corresponding payouts are used:

	No Accidents	One Accident	Two Accidents
Payout	0	5000	10000
Probability	0.96	0.03	0.01

```
[ ]: import numpy as np
x = np.array([0 , 5000, 10000])
p = np.array([0.96 , 0.03, 0.01])
mu = np.sum(x*p); sig = np.sqrt(np.sum((x-mu)**2*p));
print('Expected Value = {:.3f}, Standard Deviation = {:.3f}'.format(mu,sig))
print('mu +- 2sig = {:.3f}, {:.3f}'.format(mu-2*sig, mu+2*sig))
```

```
Expected Value = 250.000, Standard Deviation = 1299.038
mu +- 2sig = -2348.076, 2848.076
```

So, the expected value for the insurance company in this example is \$250 - the average expected payout per policy. This information is useful for making a business decision on the level of premium accounting for administrative costs, profit margin, etc. The actual payouts in any given period vary, but averaged over many policies, it gives a measure of risk.

# 4

---

## Probability Distributions

---

### 4.1 Probability Distributions

There are many important probability distributions with numerous applications in Engineering, Biology, Medicine, Finance, etc. In this chapter, only a few distributions that are important in Statistics are considered.

We have already studied simple finite probability distributions in [Chapter 3](#). The table below, for example, reviews payout distributions for an insurance company.

Table	No Accidents	One Accident	Two Accidents
Payout	0	5000	10000
Probability	0.96	0.03	0.01

The above is an example of a **discrete random variable** with a finite set of distinct values, each assigned a specific probability. More generally, a discrete probability distribution can have a **countably infinite** set of outcomes. The only requirements are:

1.  $0 \leq p(s) \leq 1$  for each event  $s$  in the sample space  $S$
2.  $\sum_{s \in S} p(s) = 1$

On the other hand, a **continuous random variable** can take any value within a given range. It is often associated with measurements like height, weight, time, temperature, distance, etc. Unlike discrete random variables, continuous ones have an uncountably infinite number of possible outcomes. Any real number range contains infinitely many values, so the probability of any specific value is 0. Instead, areas under a continuous probability distribution function are used to compute the probabilities. The two requirements for a continuous probability distribution function are:

- 1  $0 \leq f(x) \leq 1$  for all  $x$
- 2  $\int_{-\infty}^{\infty} f(x)dx = 1$

For example, an electric outlet voltage varies slightly around the prescribed value of 120 volts. One assumption might be that it has a **uniform distribution** in the range  $119 - 121$  V, i.e. the values are spread evenly over the range of possibilities.

For a general **uniform distribution** on the interval  $[a, b]$ , the probability distribution function is given by:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } a < x < b \\ 0 & \text{if } x < a \text{ or } x > b \end{cases} = \begin{cases} \frac{1}{121-119} = \frac{1}{2} & \text{if } 119 < x < 121 \\ 0 & \text{if } x < 119 \text{ or } x > 121 \end{cases}$$

Note that the area under any probability distribution is always 1.

$$\text{Total Area} = \text{base} \cdot \text{height} = (b - a) \cdot \frac{1}{(b - a)} = 1$$

The probability of any particular event (interval) is equal to the area under the probability distribution function over that interval (see Figure below). For example, to find the probability that the voltage is between 119 and 120.5 V, we compute.

$$P(119 < V < 120.5) = \text{base} \cdot \text{height} = (120.5 - 119) \cdot \frac{1}{2} = 0.75$$

Also, note that a single particular value has no area, so the probability of it is 0.  $P(V = 120) = 0$ . We could ask instead for a tight interval around the value of interest:  $P(119.9 < V < 120.1) = (120.1 - 119.9) \cdot \frac{1}{2} = 0.1$ .

```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main/
    ↪UniformDistrib.JPG'
page = requests.get(url); img = Image.open(BytesIO(page.content));
img.resize((900, 200))
```

[ ]:



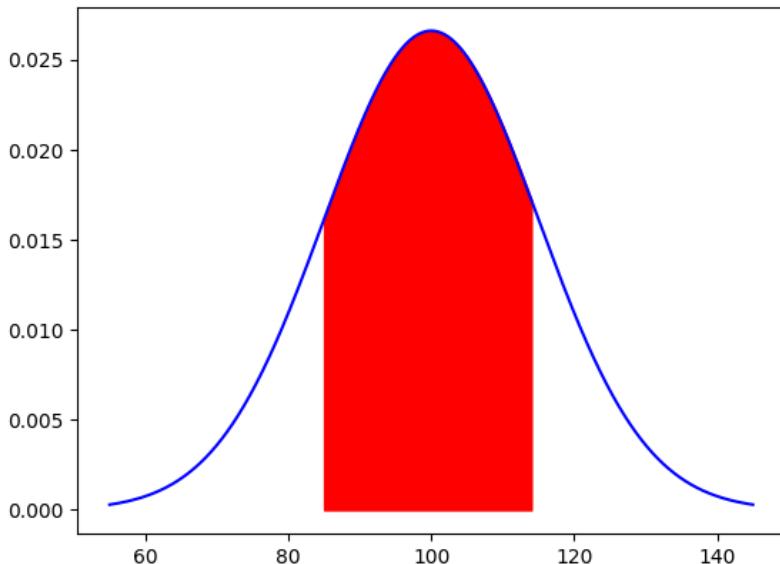
## 4.2 Normal Distribution

### 4.2.1 Normal Distribution Model

One of the most important distributions in Statistics is **normal (bell-shaped)** distribution, which is symmetric and unimodal. The IQ scores, SAT scores, heights, baby weights, and many other practical quantities closely follow such a shape with low probability for more extreme values on both ends and a bell shape in the middle. It is completely described by mean  $\mu$  (center of the bell curve) and standard deviation  $\sigma$  (spread). The Figure below shows a typical distribution of IQ scores with mean 100 and standard deviation 15 (one standard deviation away from the mean is shaded red). Generally we denote normal distribution as  $N(\mu, \sigma)$ .

```
[ ]: import numpy as np; import matplotlib.pyplot as plt;
from scipy.stats import norm;
mu=100; sig=15
x=np.linspace(mu-3*sig, mu+3*sig,1000)
plt.plot(x,norm.pdf(x,loc=mu,scale=sig), 'b')
px=np.arange(85,115,1)
plt.fill_between(px,norm.pdf(px,loc=mu,scale=sig),color='r')
```

```
[ ]: <matplotlib.collections.PolyCollection at 0x7bf3b5d1f040>
```



The probability distribution function is given by:

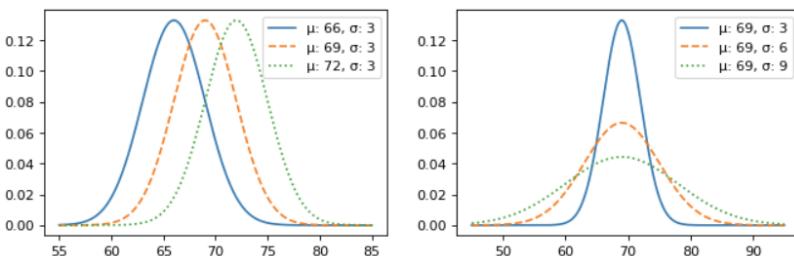
$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (4.1)$$

However, one cannot use it to find areas under the curve analytically, only numerically. Changing the mean  $\mu$  shifts the normal curve horizontally, while  $\sigma$  specifies the spread as illustrated in the Figure below. On the left, we show distributions of height for populations of three different countries with means 66 in, 69 in, and 72 in, respectively, but with the same standard deviation of 3 in. On the right, all distributions have the same mean of 69 in, but the standard deviations are 3 in, 6 in, and 9 in, respectively.

```
[1]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
figure(figsize=(10, 3), dpi=80)
plt.subplot(1, 2, 1)
x = np.arange(55, 85, 0.01)
#define multiple normal distributions
plt.plot(x, norm.pdf(x, 66, 3), label='μ: 66, σ: 3')
plt.plot(x, norm.pdf(x, 69, 3), label='μ: 69, σ: 3')
plt.plot(x, norm.pdf(x, 72, 3), label='μ: 72, σ: 3')
plt.legend()

plt.subplot(1, 2, 2)
x = np.arange(45, 95, 0.01)
#define multiple normal distributions
plt.plot(x, norm.pdf(x, 69, 3), label='μ: 69, σ: 3')
plt.plot(x, norm.pdf(x, 69, 6), label='μ: 69, σ: 6')
plt.plot(x, norm.pdf(x, 69, 9), label='μ: 69, σ: 9')
plt.legend()
```

[2]: <matplotlib.legend.Legend at 0x7bf3b575bf40>



The normal distribution with mean  $\mu = 0$  and standard deviation  $\sigma = 1$  is called **standard normal distribution**. Its probability distribution function is:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \quad (4.2)$$

It is used to compute the areas under any other normal distribution  $N(\mu, \sigma)$ , but first, the given distribution must be standardized to a Z-score with mean 0 and standard deviation 1. The **Z-score** is the number of standard deviations above or below the mean:

$$z = \frac{x - \mu}{\sigma} \quad (4.3)$$

For example, someone with an IQ score of 130 on the standard IQ scale with mean  $\mu = 100$  and standard deviation  $\sigma = 15$  has the standardized score:

$$z = \frac{x - \mu}{\sigma} = \frac{130 - 100}{15} = \frac{30}{15} = 2$$

On the other hand, a raw score of 85 implies

$$z = \frac{x - \mu}{\sigma} = \frac{85 - 100}{15} = \frac{-15}{15} = -1$$

Generally, any score above the mean leads to positive  $z$ , and any score below the mean produces negative  $z$ , while the mean score produces  $z = \frac{\mu - \mu}{\sigma} = 0$ . Note that you can always subtract the mean and divide by the standard deviation for any distribution, so z-scores can be defined for any type of data/distribution, not just normally distributed.

Z-scores are very convenient to compare variables measured on different scales. For example, let's say in a given year the SAT scores are normally distributed with mean  $\mu = 1000$  and standard deviation  $\sigma = 250$ , and Regents scores are also normally distributed with mean  $\mu = 60$  and standard deviation  $\sigma = 20$ . Let's say Jane got 1210 on her SAT and John got 70 on his Regents exam. The scores are on completely different scales, so we cannot compare them directly. However, let's find the corresponding standardized z-scores.

```
[ ]: muSAT = 1000; sigSAT = 250; x1 = 1210;
z1 = (x1-muSAT)/sigSAT; print('Jane z score = ', z1)
muReg = 60; sigReg = 20; x2 = 70;
z2 = (x2-muReg)/sigReg; print('John z score = ', z2)
```

```
Jane z score =  0.84
John z score =  0.5
```

$$z = \frac{1210 - 1000}{250} = \frac{210}{250} = 0.84$$

$$z = \frac{70 - 60}{20} = \frac{10}{20} = 0.5$$

Therefore, Jane did better on her SAT than John on his Regents test.

### 4.2.2 Normal Probability Calculations

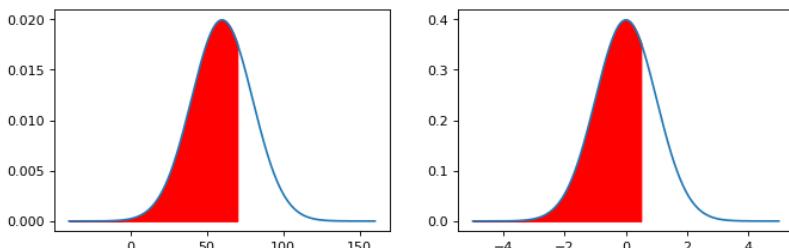
Let's now start discussing probability questions for normal distributions. For example, what is the fraction (percentage) of students who scored **below** John's score? Or equivalently, if we select a random student, what is the probability that their score is below 70? This is given by the left tail area under the normal probability distribution function shown in the Figure below. I created my own (user) function `plot_normal()` with `def` command. In parentheses, it has its input parameters. This function does not produce any output; it plots the areas under the normal curve with given parameters and the corresponding standard normal area.

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 60; # mean
sig = 20; # standard deviation
x = 70; # raw x value
z = (x-mu)/sig; print('z score = {:.4f}'.format(z))
p = norm.cdf(z); print('probability = {:.4f} \n\n'.format(p))
```

`z score = 0.5000`  
`probability = 0.6915`

```
[ ]: def plot_normal(mu,sig,x1,x2,z1,z2):
    # Plotting the shaded areas for normal distribution
    figure(figsize=(10, 3), dpi=80)
    plt.subplot(1, 2, 1)
    xv = np.arange(mu-5*sig, mu+5*sig, 0.01)
    plt.plot(xv, norm.pdf(xv, mu, sig))
    px=np.arange(x1,x2,0.01)
    plt.fill_between(px,norm.pdf(px,mu,sig),color='r')
    plt.subplot(1, 2, 2)
    zv = np.arange(-5, 5, 0.01)
    plt.plot(zv, norm.pdf(zv, 0, 1))
    pz=np.arange(z1,z2,0.01)
    plt.fill_between(pz,norm.pdf(pz),color='r')
```

```
[ ]: plot_normal(mu=mu,sig=sig,x1=mu-5*sig,x2=x,z1=-5,z2=z)
```



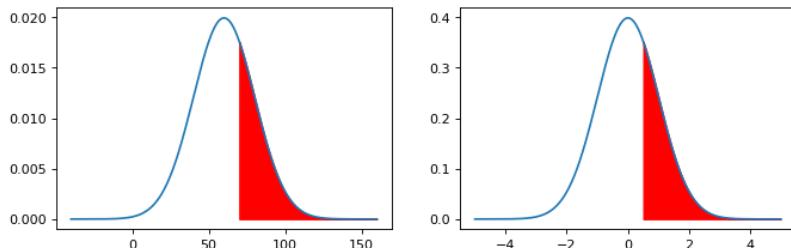
Therefore, 69.15% of students have scores below John's score of 70.

The next common question to ask would be what is the fraction (percentage) of students, who scored **above** John's score (**at least** as good as John's)? Or equivalently, if we select a random student, what is the probability that their score is above John's score of 70? This is given by the right tail area under the normal probability distribution function shown in the Figure below. The total area under any probability distribution is 1, so we can find the *complement area* to the left and then subtract it from 1:

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 60; # mean
sig = 20; # standard deviation
x = 70; # raw x value
z = (x-mu)/sig; print('z score = {:.4f}'.format(z))
p = 1-norm.cdf(z); print('probability = {:.4f}\n\n'.format(p))

plot_normal(mu=mu,sig=sig,x1=x,x2=mu+5*sig,z1=z,z2=5)
```

`z score = 0.5000  
probability = 0.3085`

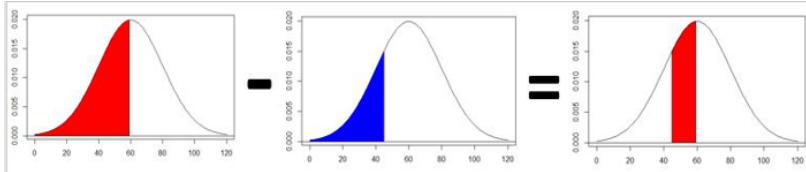


Finally, a school board may decide that students who scored between 45 and 59 require extra help. What proportion of students falls in **between** these bounds? Or equivalently, what is the probability that a randomly chosen student falls in these bounds?

This probability is best obtained as a **difference** between the probability of scoring below 59 and scoring below 45 as shown in the Figure below:

```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main/
    ↪ProbSubtract.JPG'
page = requests.get(url); img = Image.open(BytesIO(page.content));
img.resize((700, 150))
```

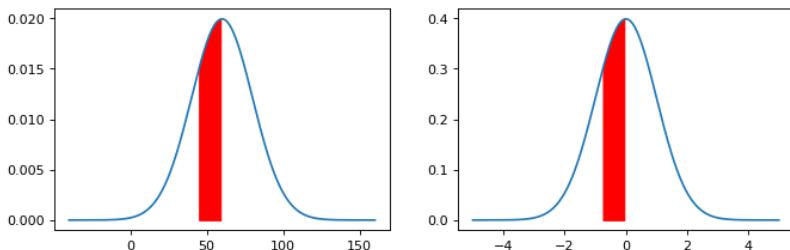
[ ]:



```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 60; # mean
sig = 20; # standard deviation
x1 = 45; x2 = 59; # raw x value
z1 = (x1-mu)/sig; z2 = (x2-mu)/sig;
print('z1, z2 scores = {:.4f}, {:.4f}'.format(z1,z2))
p = norm.cdf(z2)-norm.cdf(z1);
print('probability = {:.4f}\n\n'.format(p))

plot_normal(mu=mu,sig=sig,x1=x1,x2=x2,z1=z1,z2=z2)
```

z1, z2 scores = -0.7500, -0.0500  
probability = 0.2534



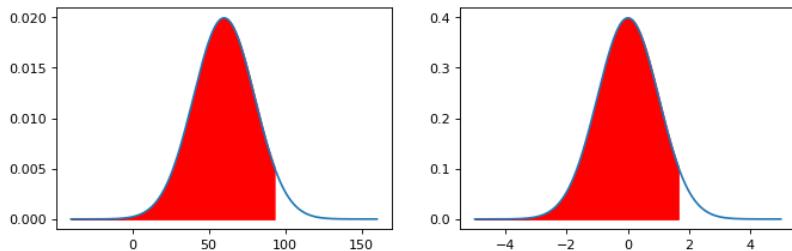
The last several problems have focused on finding the percentile (lower tail area) or upper tail area for a particular observation. There is an opposite kind of question - what value (**quantile**) corresponds to a particular percentile? For example, what is the value corresponding to the 95 percentile on the Regents test? Or equivalently, which score corresponds to the top 5% on the Regents exam? To answer this question, first, we need to use the inverse function of cumulative probability distribution (CDF) to find the standardized z-score corresponding to 95% of the data below it, as illustrated in the Figure below. Then, find the raw score  $x$  from the definition of  $z$ :

$$z = \frac{x - \mu}{\sigma} \implies z \cdot \sigma = x - \mu \implies x = \mu + z \cdot \sigma \quad (4.4)$$

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 60; # mean
sig = 20; # standard deviation
ProbLeft = 0.95
z = norm.ppf(ProbLeft); print('z = {:.4f}'.format(z))
x = mu + z*sig; print('raw score x = {:.4f}\n\n'.format(x))

plot_normal(mu=mu,sig=sig,x1=mu-5*sig,x2=x,z1=-5,z2=z)
```

`z = 1.6449  
raw score x = 92.8971`



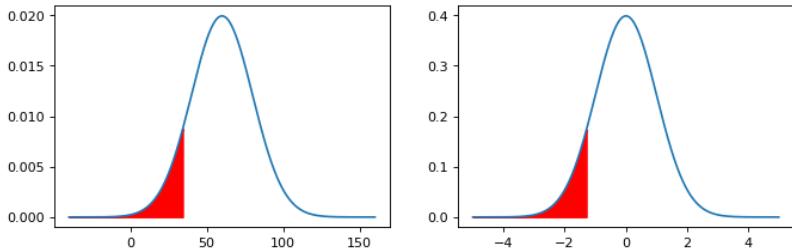
The value (quantile) corresponding to the 95th percentile is 92.897. The 50th percentile corresponds to  $z = 0$ , so the raw score is  $x = \mu + z \cdot \sigma = \mu + 0 \cdot \sigma = \mu$ .

Let's also compute a quantile corresponding to the lower 10th percentile on the Regents test. Or equivalently, which score corresponds to the bottom 10% scores on the Regents exam? The Figure below illustrates the bottom 10% area under the normal distribution.

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 60; # mean
sig = 20; # standard deviation
ProbLeft = 0.10
z = norm.ppf(ProbLeft); print('z = {:.4f}'.format(z))
x = mu + z*sig; print('raw score x = {:.4f}\n\n'.format(x))

plot_normal(mu=mu,sig=sig,x1=mu-5*sig,x2=x,z1=-5,z2=z)
```

`z = -1.2816  
raw score x = 34.3690`



The value (quantile) corresponding to the 10th percentile is 34.369. In this case,  $z$  is negative and the quantile is below the mean.

#### 4.2.3 Central Limit Theorem

The Central Limit Theorem (CLT) is perhaps the most amazing scientific result. We will see it again and again throughout the book. For now, we just want to understand it in terms of adjusting standard deviation in normal distribution computation for sample means. The CLT proves that for a population of **ANY distribution** with mean  $\mu$  and standard deviation  $\sigma$ , the distribution of the sample means converges to a normal distribution as the sample size  $n$  increases ( $n \geq 30$  is a common guideline). The mean of this distribution of sample means is the same  $\mu$ , but the standard deviation is much smaller  $\frac{\sigma}{\sqrt{n}}$ , which produces much “sharper” normal distribution more centered around the mean. If the original population is already normally distributed, then CLT holds for a sample of any size.

#### 4.2.4 Normal Distribution Examples

There are many other examples of normal distribution.

##### Example

Let's consider a business-oriented example. Suppose that the gas mileage of a particular type of car is normally distributed with a mean of 22 mpg and a standard deviation of 6 mpg.

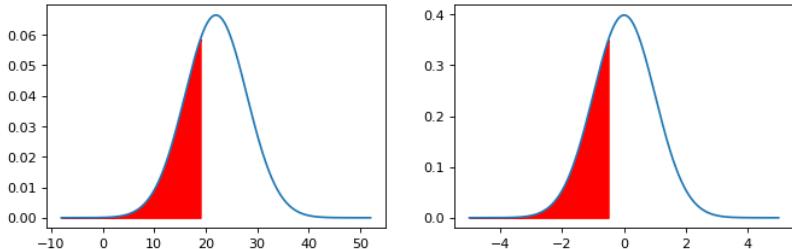
First, which percentage of cars has mpg **below** 19 (left tail on the graph shown below)? Or equivalently, if we select a random car, what is the probability that its mpg is below 19?

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 22; # mean
sig = 6; # standard deviation
x = 19; # raw x value
```

```
z = (x-mu)/sig; p = norm.cdf(z);
print('z score = {:.4f}, p = {:.4f}'.format(z,p))

plot_normal(mu=mu,sig=sig,x1=mu-5*sig,x2=x,z1=-5,z2=z)
```

`z score = -0.5000, p = 0.3085`



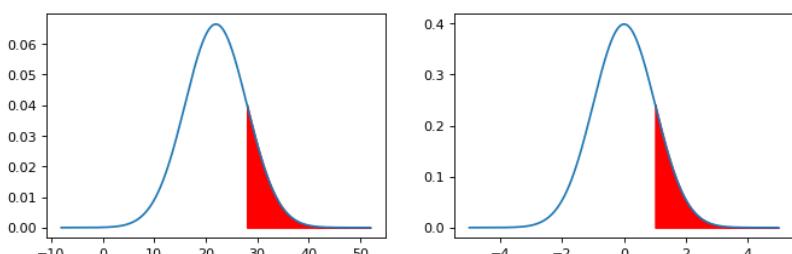
Therefore, 30.85% of cars have below 19 mpg.

Next, what is the fraction (percentage) of cars with mpg **above** 28 (right tail)? Or equivalently, if we select a random car, what is the probability that its mpg is above 28?

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 22; # mean
sig = 6; # standard deviation
x = 28; # raw x value
z = (x-mu)/sig; p = 1-norm.cdf(z);
print('z score = {:.4f}, p = {:.4f}'.format(z,p))

plot_normal(mu=mu,sig=sig,x1=x,x2=mu+5*sig,z1=z,z2=5)
```

`z score = 1.0000, p = 0.1587`



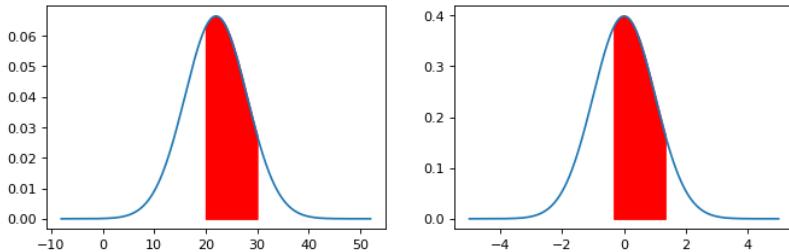
Therefore, 15.86% of cars have an mpg above 28.

Which percentage of cars has “good mpg” that falls in **between** 20 and 30 mpg as shown in the Figure below? Or equivalently, what is the probability that a randomly chosen car mpg falls in these bounds?

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 22; # mean
sig = 6; # standard deviation
x1 = 20; x2 = 30; # raw x value
z1 = (x1-mu)/sig; z2 = (x2-mu)/sig;
p = norm.cdf(z2)-norm.cdf(z1);
print('z1 = {:.4f}, z2 = {:.4f}, p = {:.4f}'.format(z1,z2,p))

plot_normal(mu=mu,sig=sig,x1=x1,x2=x2,z1=z1,z2=z2)
```

`z1 = -0.3333, z2 = 1.3333, p = 0.5393`



Therefore, 53.93% of cars have an mpg between 20 and 30.

Let's also consider the mean of a **sample** of  $n = 30$  cars. What is the probability that a **sample mean** mpg is **below** 19? The standard deviation for sample means is much smaller according to the CLT:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}} = \frac{6}{\sqrt{30}} = 1.095$$

Note that with much smaller  $\sigma_{\bar{x}}$ , we get a much sharper (tighter) normal distribution and, as a result, much less chance that a sample mean will be below 19 as can be seen in the Figure below.

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 22; # mean
sig = 6; # standard deviation
x = 19; # raw x value
z1 = (x-mu)/sig; p1 = norm.cdf(z1);
print('1 item: z1 score = {:.4f}, probability = p1 = {:.4f}\n'.format(z1,p1))

n = 30; # sample size
zn = (x-mu)/(sig/np.sqrt(n)); pn = norm.cdf(zn);
```

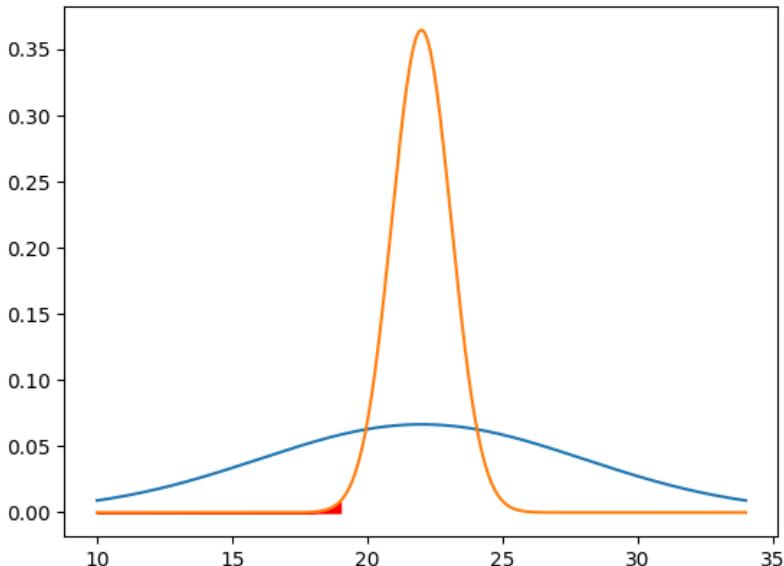
```
print('mean of n items: zn = {:.4f}, pn = {:.4f}\n'.format(zn,pn))

xv = np.arange(mu-2*sig, mu+2*sig, 0.01)
plt.plot(xv, norm.pdf(xv, mu, sig))
plt.plot(xv, norm.pdf(xv, mu, sig/np.sqrt(n)))
px=np.arange(mu-2*sig,x,0.01)
plt.fill_between(px,norm.pdf(px,mu,sig/np.sqrt(n)),color='r')
```

1 item: z1 score = -0.5000, probability = p1 = 0.3085

mean of n items: zn = -2.7386, pn = 0.0031

[ ]: <matplotlib.collections.PolyCollection at 0x7bf3b57581c0>

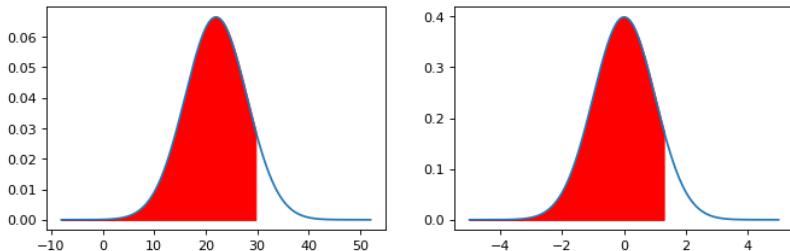


Continuing to quantiles, what is the value (quantile) corresponding to the 90th percentile of mpg of cars (this is the value such that 90% of cars have mpg below it)? Or equivalently, which score corresponds to the top 10% of mpg's? First, let's find the standardized z-score corresponding to the 90th percentile as illustrated in the Figure below.

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 22; # mean
sig = 6; # standard deviation
ProbLeft = 0.90
z = norm.ppf(ProbLeft); print('z = {:.4f}'.format(z))
```

```
x = mu + z*sig; print('raw score x = {:.4f}'.format(x))
plot_normal(mu=mu,sig=sig,x1=mu-5*sig,x2=x,z1=-5,z2=z)
```

```
z = 1.2816
raw score x = 29.6893
```



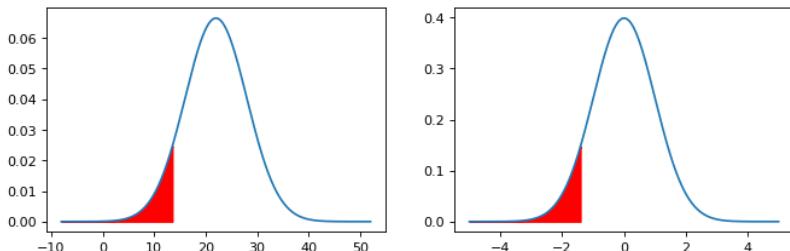
The value (quantile) corresponding to the 90th percentile is 29.6893.

Let's also compute a quantile corresponding to the lower 8th percentile (the value such that only 8% of cars have mpg below it). Or equivalently, which score corresponds to the bottom 8% of the values? As before, find the standardized z-score corresponding to 8% of the data as illustrated in the Figure below, and compute the corresponding raw score.

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 22; # mean
sig = 6; # standard deviation
ProbLeft = 0.08
z = norm.ppf(ProbLeft); print('z = {:.4f}'.format(z))
x = mu + z*sig; print('raw score x = {:.4f}'.format(x))

plot_normal(mu=mu,sig=sig,x1=mu-5*sig,x2=x,z1=-5,z2=z)
```

```
z = -1.4051
raw score x = 13.5696
```



The value (quantile) corresponding to the 8th percentile is 13.57. In this case,  $z$  is negative, and the quantile is below the mean.

### Example

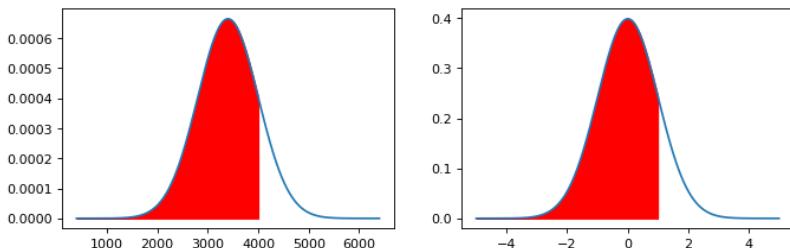
The distribution of baby weights is known to be normal with a mean around 3400 g and a standard deviation of approximately 600 g.

Which percentage of baby' weights is **below** 4000 g? Equivalently, for a randomly chosen baby, what is the probability that the weight is below 4000 g? This is given by the left tail area under the normal probability distribution function shown the Figure below.

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 3400; # mean
sig = 600; # standard deviation
x = 4000; # raw x value
z = (x-mu)/sig; p = norm.cdf(z);
print('z score = {:.4f}, probability = {:.4f}'.format(z,p))

plot_normal(mu=mu,sig=sig,x1=mu-5*sig,x2=x,z1=-5,z2=z)
```

$z$  score = 1.0000, probability = 0.8413

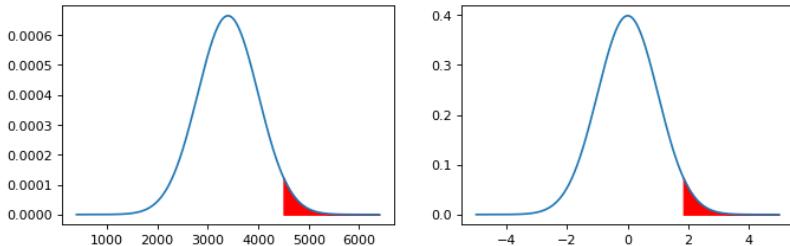


Next, what is the fraction (percentage) of babies who are heavier than (weighing more than) 4500? Equivalently, for a randomly chosen baby, what is the probability that the weight is **above** 4500? This is given by the right tail area under the normal probability distribution function shown below:

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 3400; # mean
sig = 600; # standard deviation
x = 4500; # raw x value
z = (x-mu)/sig; p = 1-norm.cdf(z);
print('z score = {:.4f}, probability = {:.4f}'.format(z,p))
```

```
plot_normal(mu=mu,sig=sig,x1=x, x2=mu+5*sig,z1=z,z2=5)
```

`z score = 1.8333, probability = 0.0334`



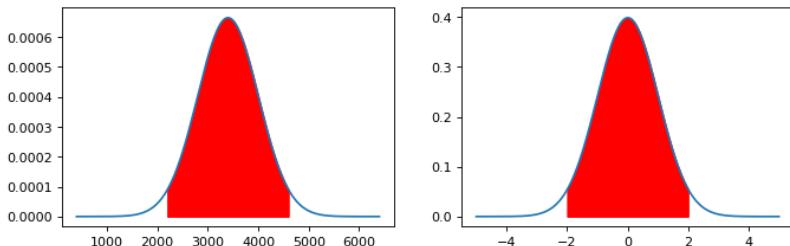
Which fraction (percentage) of babies have a weight **between** 2200 and 4600 g as shown in the Figure below? Equivalently, for a randomly chosen baby, what is the probability that their weight is within the above bounds?

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 3400; # mean
sig = 600; # standard deviation
x1 = 2200; x2 = 4600; # raw x value

z1 = (x1-mu)/sig; z2 = (x2-mu)/sig;
print('z1, z2 scores = {:.4f}, {:.4f}'.format(z1,z2))
p = norm.cdf(z2)-norm.cdf(z1);
print('probability = {:.4f}\n\n'.format(p))

plot_normal(mu=mu,sig=sig,x1=x1,x2=x2,z1=z1,z2=z2)
```

`z1, z2 scores = -2.0000, 2.0000  
probability = 0.9545`



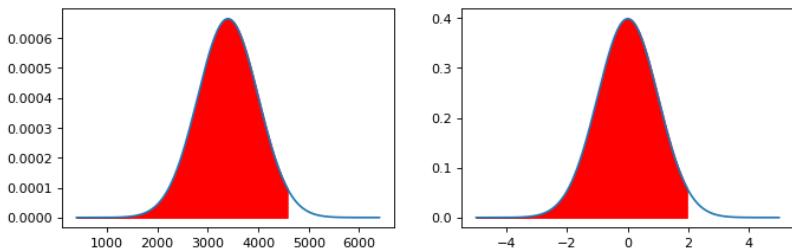
The above is close to 95%, but not quite. Let's consider the middle 95% of data in more details. First, let's determine the quantile or baby's weight

corresponding to the 97.5th percentile (i.e, the weight such that 97.5% of babies have lower weights)? Equivalently, which weight separates the top 2.5% of baby weights?

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 3400; # mean
sig = 600; # standard deviation
ProbLeft = 0.975
z = norm.ppf(ProbLeft); print('z = {:.4f}'.format(z))
x = mu + z*sig; print('raw score x = {:.4f}'.format(x))

plot_normal(mu=mu,sig=sig,x1=mu-5*sig,x2=x,z1=-5,z2=z)
```

`z = 1.9600  
raw score x = 4575.9784`

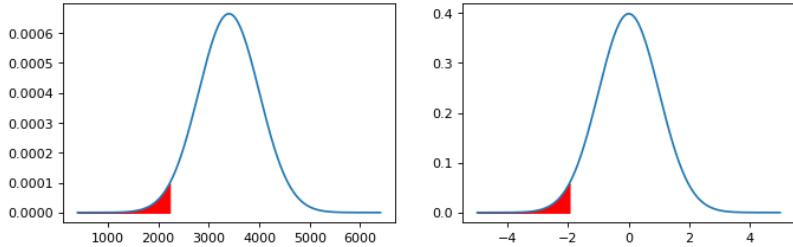


Analogously which weight (quantile) corresponds to a lower 2.5th percentile of baby weights (i.e, what is the weight such that only 2.5% of babies have weights below it)? Equivalently, which weight separates the bottom 2.5% of baby weights?

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 3400; # mean
sig = 600; # standard deviation
ProbLeft = 0.025
z = norm.ppf(ProbLeft); print('z = {:.4f}'.format(z))
x = mu + z*sig; print('raw score x = {:.4f}'.format(x))

plot_normal(mu=mu,sig=sig,x1=mu-5*sig,x2=x,z1=-5,z2=z)
```

`z = -1.9600  
raw score x = 2224.0216`



The quantiles are approximately 2224 g for the 2.5th percentile and 4576 g for the 97.5th percentile. These bounds contain the middle 95% of the data. Medical professionals usually use them to define cutoffs for low birth weight and high birth weight babies requiring special medical attention.

Let's also illustrate the CLT by comparing the probability for one baby to have a weight more than, say, 3600 g and for a **sample** of  $n = 30$  babies to have average weight above 3600 g. For the sample, we must adjust the sample standard deviation

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}} = \frac{600}{\sqrt{30}} = 109.545$$

As always,  $\sigma_{\bar{x}}$  is much smaller than  $\sigma$ , so the sample means normal distribution is much sharper (tighter), and there is much less chance that a sample mean will be above 3600 as can be seen in the Figure below.

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
mu = 3400; # mean
sig = 600; # standard deviation
x = 3600; # raw x value
z1 = (x-mu)/sig; p1 = 1-norm.cdf(z1);
print('1 item: z1 score = {:.4f}, probability = p1 = {:.4f}\n'.format(z1,p1))

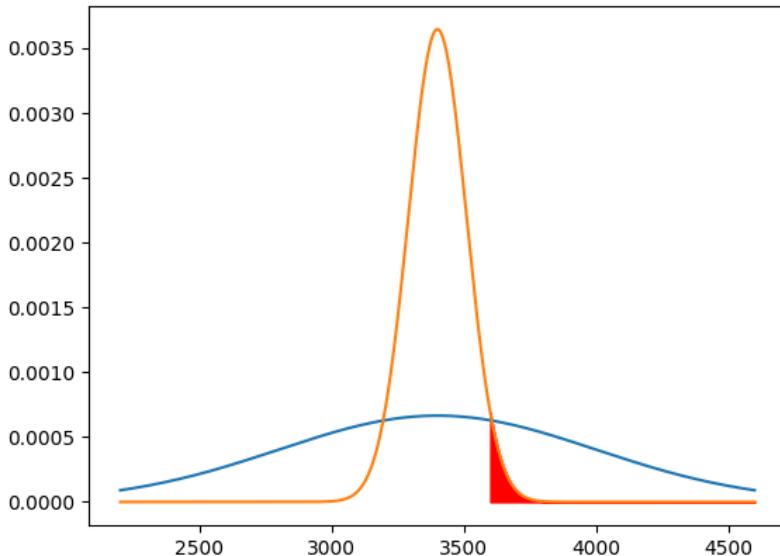
n = 30; # sample size
zn = (x-mu)/(sig/np.sqrt(n)); pn = 1-norm.cdf(zn);
print('mean of n items: zn = {:.4f}, pn = {:.4f}\n'.format(zn,pn))

xv = np.arange(mu-2*sig, mu+2*sig, 0.01)
plt.plot(xv, norm.pdf(xv, mu, sig))
plt.plot(xv, norm.pdf(xv, mu, sig/np.sqrt(n)))
px=np.arange(x,mu+2*sig,0.01)
plt.fill_between(px,norm.pdf(px,mu,sig/np.sqrt(n)),color='r')
```

1 item: z1 score = 0.3333, probability = p1 = 0.3694

mean of n items: zn = 1.8257, pn = 0.0339

```
[ ]: <matplotlib.collections.PolyCollection at 0x7bf3b5cf000>
```



#### 4.2.5 68-95-99.7 Rule

We mentioned in the previous chapter that for the bell-shaped data, there are certain percentages of the data falling within 1, 2, and 3 standard deviations of the mean. Let's formalize it more precisely here. First, rewrite this in terms of  $z$ . Say, 1 standard deviation away from the mean results in

$$z = \frac{(\mu \pm \sigma) - \mu}{\sigma} = \frac{\pm \sigma}{\sigma} = \pm 1$$

Analogously, for the 2 standard deviations:

$$z = \frac{(\mu \pm 2\sigma) - \mu}{\sigma} = \frac{\pm 2\sigma}{\sigma} = \pm 2$$

The area under the normal probability distribution function contained within  $z = \pm 1, \pm 2$ , and  $\pm 3$  is computed as follows:

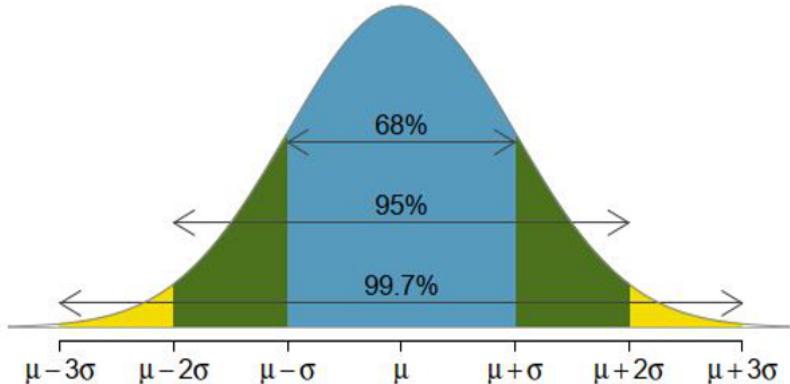
```
[ ]: from scipy.stats import norm;
print('Areas under the standard normal probability distribution')
p1 = norm.cdf(1)-norm.cdf(-1);
print('Between -1 to 1 = {:.4f}'.format(p1))
p2 = norm.cdf(2)-norm.cdf(-2);
print('Between -2 to 2 = {:.4f}'.format(p2))
p3 = norm.cdf(3)-norm.cdf(-3);
print('Between -3 to 3 = {:.4f}'.format(p3))
```

```
Areas under the standard normal probability distribution
Between -1 to 1 = 0.6827
Between -2 to 2 = 0.9545
Between -3 to 3 = 0.9973
```

Therefore, all the standard percentages given in the Figure below check out.

```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main/Normal6895.
↪JPG'
page = requests.get(url); img = Image.open(BytesIO(page.content));
img.resize((600, 300))
```

[ ]:



Let's make this statement just a bit more precise with a quantile approach. If we want exactly 95% of the data in the middle, we need to find  $z$ -scores corresponding to 2.5% on the left and 97.5% on the right.

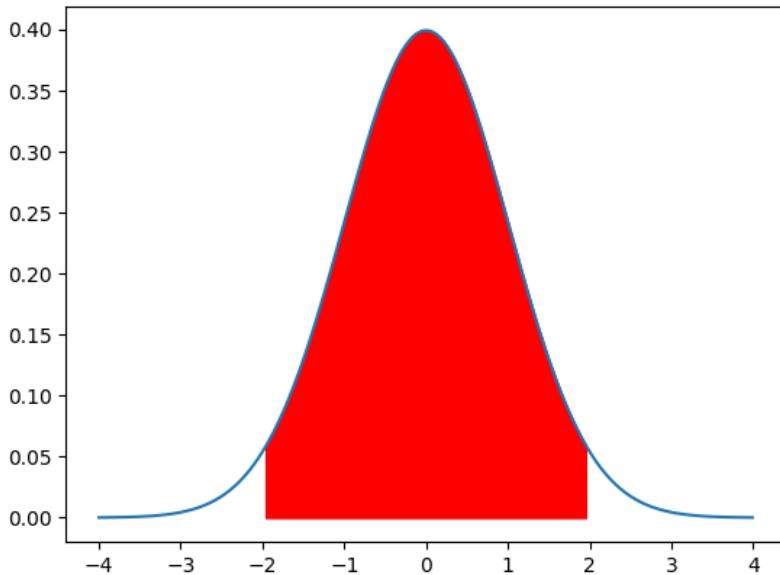
The normal distribution is symmetric, so these values have the same magnitude, just opposite signs. They are approximately  $z = \pm 1.960$ , which is quite close to  $z = \pm 2$ , but not the same, as shown in the Figure below.

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; from matplotlib.pyplot import figure
ProbLeft = 0.025; z1 = norm.ppf(ProbLeft);
ProbLeft = 0.975; z2 = norm.ppf(ProbLeft);
print('z1 = {:.4f}, z2 = {:.4f}\n'.format(z1,z2))

zv = np.arange(-4, 4, 0.01)
plt.plot(zv, norm.pdf(zv, 0, 1))
pz=np.arange(z1,z2,0.01)
plt.fill_between(pz,norm.pdf(pz),color='r')
```

$z1 = -1.9600$ ,  $z2 = 1.9600$

```
[ ]: <matplotlib.collections.PolyCollection at 0x7bf3b53adff0>
```



The Standard Normal distribution has  $e^{-x^2/2}$  term which decreases to 0 very fast but never touches or crosses it, so a normal random variable can fall 4, 5, or even more standard deviations away from the mean, but it is very unlikely as shown below.

```
[ ]: from scipy.stats import norm;
print('Areas under the standard normal probability distribution')
print('Area above 4 = {:.16f}'.format(1-norm.cdf(4)))
print('Area above 5 = {:.16f}'.format(1-norm.cdf(5)))
print('Area above 6 = {:.16f}'.format(1-norm.cdf(6)))
print('Area above 10 = {:.16f}'.format(1-norm.cdf(10)))
```

```
Areas under the standard normal probability distribution
Area above 4 = 0.0000316712418331
Area above 5 = 0.0000002866515719
Area above 6 = 0.0000000009865877
Area above 10 = 0.0000000000000000
```

```
[ ]:
```

### 4.3 Binomial Distribution

Consider flipping a coin 10 times, say. Each coin flip (**trial**) can result in only two outcomes H or T with fixed probability, **independent** of each other. Define a random variable as a number of heads in these 10 coin flips. This is an example of **Binomial Distribution**. Thus, it is used to describe the following situation: *there are  $n$  independent trials, each one is either success or failure with fixed probability, and we look at the distribution of the number of successes*. Another basic example might be rolling a fair die. One might argue that it does not provide two outcomes, but 6. However, if we denote success as rolling 6 (or any other outcome), then we can have, say,  $n = 20$  independent die rolls (trials), each one is either a success (rolling 6 with probability 1/6) or failure (rolling anything else with probability 5/6). Then, the distribution of the number of successes is Binomial.

Individual trials resulting in success (probability  $p$ ) or failure ( $1 - p$ ) are called Bernoulli trials. The Binomial distribution is a sum of such Bernoulli trials. **Bernoulli random variable** has distribution:

$$f(x) = \begin{cases} p & \text{if } x = 1 \text{ (success)} \\ (1 - p) & \text{if } x = 0 \text{ (failure)} \end{cases} \quad (4.5)$$

The expected value is:

$$\mu = E(X) = \sum x \cdot P(X = x) = 1 \cdot p + 0 \cdot (1 - p) = p$$

The standard deviation is:

$$\sigma = \sqrt{\sum (x - \mu)^2 \cdot P(X = x)} = \sqrt{(1 - p)^2 \cdot p + (0 - p)^2 \cdot (1 - p)} =$$

$$\sqrt{p(1 - p)(1 - p + p)} = \sqrt{p(1 - p)}$$

Bernoulli's random variable provides a way to connect means and proportions. Assume 10 Bernoulli trials are observed and result in  $x = (1, 0, 0, 1, 0, 1, 0, 0, 0, 1)$ . The mean is

$$\bar{x} = \frac{\sum x_i}{n} = \frac{1 + 0 + 0 + 1 + 0 + 1 + 0 + 0 + 0 + 1}{10} = \frac{4}{10} =$$

$$\frac{\text{number of successes}}{\text{number of trials}} = \text{proportion successes} = \hat{p}$$

Therefore, proportion can be represented as a mean.

### 4.3.1 Permutations and Combinations

To derive the Binomial formula, we need to make a very short detour into an intricate subject of Combinatorics.

First, let's remind ourselves of the definition of factorial:

$$n! = n \cdot (n - 1) \cdot (n - 2) \cdot (n - 3) \cdot \dots \cdot 3 \cdot 2 \cdot 1 = n \cdot (n - 1)!$$

For example

$$0! = 1$$

$$1! = 1$$

$$2! = 2 \cdot 1 = 2$$

$$3! = 3 \cdot 2 \cdot 1 = 6$$

$$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$$

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$$

$$6! = 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 720$$

The factorials grow very fast.

Consider a Business department with 7 faculty members. How many ways are there to choose a chair, associate chair, and treasurer (7 distinct items, 3 slots)? Note here that the **order matters**. There are 7 ways to choose the chair, 6 ways to choose an associate chair from the remaining 6 people, and 5 ways to choose the treasurer for the remaining 5. This is called the **number of permutations**. We can represent it with factorials.

$${}_7P_3 = 7 \cdot 6 \cdot 5 = \frac{7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{4 \cdot 3 \cdot 2 \cdot 1} = \frac{7!}{4!} = \frac{7!}{(7-3)!} = 210$$

Generally, if we are given  $n$  different items to select  $r$  items when **order is important**, then the **number of permutations** is

$${}_nP_r = \frac{n!}{(n-r)!} \quad (4.6)$$

How many ways are there to sit these faculty members around the table? Once again order matters, and we have 7 ways to choose a person for the 1st chair, 6 ways for the 2nd, etc., so

$$7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 7! = {}_7P_7$$

Generally,

$${}_nP_n = \frac{n!}{(n-n)!} = \frac{n!}{(0)!} = \frac{n!}{1} = n!$$

On the other hand, let's consider how many ways to choose 3 faculty members for a committee. In this case, the **order does NOT matter**. We know that there are  ${}_7P_3$  ways to choose 3 out of 7 when order does matter. For each of these choices, there are  $3!$  ways to arrange 3 faculty members inside the chosen group. Therefore, **the number of combinations (order does not matter)** of 3 items selected from 7 different items is:

$${}_7C_3 = \frac{{}_7P_3}{3!} = \frac{\frac{7!}{4!}}{3!} = \frac{7!}{3!4!} = \frac{7 \cdot 6 \cdot 5}{6} = 35$$

Generally, **the number of combinations (order does not matter)** of  $r$  items selected from  $n$  different items is:

$${}_nC_r = \frac{n!}{r!(n-r)!} \quad (4.7)$$

There are many examples of the number of combinations. For example, how many different Poker hands can one draw from a standard deck of 52 cards? A poker hand is 5 cards, and it does not matter in which order they come; therefore, the number of combinations is:

$${}_{52}C_5 = \frac{52!}{5!47!} = 2598960$$

Another standard example is a State Lottery. How many ways to choose 6 balls out of 50 numbered balls? Note that you win if you have the correct numbers in **any order**, so order does not matter and we use combinations:

$${}_{50}C_6 = \frac{50!}{6!44!} = 15890700$$

Therefore, if you buy one lottery ticket, your chance to win is  $\frac{1}{15890700}$ . The computations above are illustrated in the Python code below.

```
[ ]: import math
fact = math.factorial(7); print('factorial of 7 = ',fact )
fact = math.factorial(50); print('factorial of 50 = ',fact )
nPr = math.perm(7,3); print('7P3 = ',nPr )
nCr = math.comb(7,3); print('7C3 = ',nCr )
nCr = math.comb(52,5); print('52C5 = ',nCr )
nCr = math.comb(50,6); print('50C6 = ',nCr )
P = 1/nCr; print('Probability to win = ',P )
```

```
factorial of 7 =  5040
factorial of 50 =
304140932017133780436126081660647688443776415689605120000000000000
7P3 =  210
7C3 =  35
52C5 =  2598960
50C6 =  15890700
Probability to win =  6.292988980976294e-08
```

### 4.3.2 Binomial Formula

Consider a medical science example to introduce the Binomial formula. Let's assume a new drug is tested. It either has an effect (success) or does not (failure). In the course of a clinical trial, this drug is administered to a group of  $n$  independent patients, and the outcome for each patient is binary - either there is an intended effect with pre-clinical trial probability  $p$  or not (with a complement probability  $1 - p$ ). To understand the formula, let's start with an extremely small sample of size  $n = 3$  patients recorded as A, B, and C and probability of success  $p = 0.87$ . What is the probability that the drug has the intended effect in exactly 2 patients? There are 3 options for that and the probability is found using *independence* of the patients:

$$P(A = \text{yes}, B = \text{yes}, C = \text{no}) = 0.87 \cdot 0.87 \cdot 0.13$$

$$P(A = \text{yes}, B = \text{no}, C = \text{yes}) = 0.87 \cdot 0.13 \cdot 0.87$$

$$P(A = \text{no}, B = \text{yes}, C = \text{yes}) = 0.13 \cdot 0.87 \cdot 0.87$$

These probabilities are the same, and the probability that the drug works on exactly two patients is

$$P = (\text{number of ways}) \cdot P(\text{one way}) = 3 \cdot 0.87^2 \cdot 0.13 = {}_3C_2 \cdot 0.87^2 \cdot 0.13$$

The number of combinations  ${}_3C_2$  is used because the order of successes and failures makes no difference in the equation.

Generalizing, consider a binomial distribution with  $n$  independent trials, each one is either a success with probability  $p$  or failure with probability  $1 - p$ . Let  $X$  = be a number of successes. Then

$$P(X = k) = (\text{number of ways } k \text{ successes and } (n-k) \text{ failures}) \cdot P(\text{one way}) =$$

$${}_nC_k \cdot p \cdot p \dots \cdot p \cdot (1 - p) \cdot (1 - p) \cdot \dots \cdot (1 - p)$$

$$P(X = k) = {}_nC_k p^k (1 - p)^{n-k} = \frac{n!}{k!(n - k)!} p^k (1 - p)^{n-k} \quad (4.8)$$

Like any other standard distribution, there are well-known formulas for mean and standard deviation of the Binomial Distribution:

$$\mu = E = n \cdot p \quad \sigma = \sqrt{n \cdot p \cdot (1 - p)} \quad (4.9)$$

We would not have to compute using basic formulas  $\mu = E = \sum x_i \cdot P(X = x_i)$  and  $\sigma = \sqrt{\sum (x_i - \mu)^2 \cdot P(X = x_i)}$  from scratch.

A very small sample of size  $n = 3$  was used just to illustrate the formula. More realistically, consider a random sample of  $n = 25$  **independent** patients, for each one either the drug works (success  $p = 0.87$ ) or it doesn't (failure  $1 - p = 1 - 0.87 = 0.13$ ). Let's illustrate typical questions for Binomial distribution.

First, let's find what is the probability that the drug works for 20 patients?

$$P(X = 20) = \frac{25!}{20!5!} 0.87^{20} 0.13^5$$

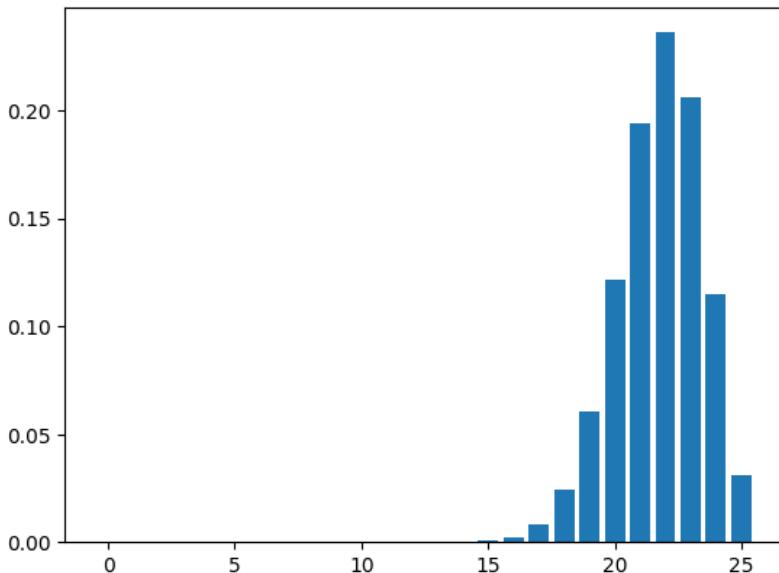
```
[ ]: from scipy.stats import binom
n = 25; p = 0.87
p1 = binom.pmf(k=20,n=n,p=p); print('p1 = {:.4f}'.format(p1))
```

p1 = 0.1217

We can compute this probability for any number of successes in the range  $k = 0..n$ , which produces **Probability Distribution Function (PDF)** shown as a barplot below.

```
[ ]: import matplotlib.pyplot as plt; import numpy as np
kv = np.arange(n+1);
pv = binom.pmf(k=kv,n=25,p=0.87);
plt.bar(kv, pv)
```

```
[ ]: <BarContainer object of 26 artists>
```



Next, what is the probability that 20 **or fewer** (fewer than 21) patients responded well to this drug?

$$P(X \leq 20) = P(X = 0) + P(X = 1) + P(X = 2) + \dots + P(X = 20)$$

It is exactly what is called the **Cumulative Distribution Function (CDF)**:

```
[ ]: p2 = binom.cdf(k=20,n=n,p=p); print('p2 = {:.4f}'.format(p2))
```

p2 = 0.2183

Alternatively, we can ask what is the probability that 21 **or more** (more than 20) patients responded well to the drug? This is the complement of the previous question, so it can be computed as:

```
[ ]: p3 = 1-p2; print('p3 = {:.4f}'.format(p3)) # OR
      p3 = 1-binom.cdf(k=20,n=n,p=p); print('p3 = {:.4f}'.format(p3))
```

p3 = 0.7817  
p3 = 0.7817

Next, let's consider the probability that between 16 and 21 patients responded well to the drug. Unlike a continuous distribution, for any discrete distribution, the in-between question requires subtracting one from the lower limit so that this value is included:

```
[ ]: p4 = binom.cdf(k=21,n=n,p=p) - binom.cdf(k=15,n=n,p=p);
      print('p4 = {:.4f}'.format(p4))
```

p4 = 0.4116

Finally, let's find the mean and standard deviation of this distribution. In the code below, we have also included  $\mu \pm 2\sigma$  bounds of usual values (at least 75% by Chebyshev Theorem). The barplot of the probability distribution shows a similar range of likely outcomes.

```
[ ]: mu = n*p; sig = np.sqrt(n*p*(1-p));
      print('mean mu = {:.4f}, standard deviation = {:.4f}'.format(mu,sig))
      print('usual range {:.4f}, {:.4f} '.format(mu-2*sig, mu+2*sig))
```

mean mu = 21.7500, standard deviation = 1.6815  
usual range 18.3870, 25.1130

Note that the designation of success or failure in the Binomial distribution setup is somewhat arbitrary and may not correspond to common sense success. For example, in the problem below the event is a patient experiencing negative side effects of the drug, which is not a success. However, all that matters is that the outcome is binary and we stay consistent.

### Example

Consider a drug with a 3% side effect rate. In a random sample of  $n = 20$  (**independent**) patients, each one either has a side effect ( $p = 0.03$ ) or does not ( $1 - p = 1 - 0.03 = 0.97$ ).

In the Python code below, the same questions as in the previous problem are considered. We only show the theoretical answer for one of them in terms of the Binomial formula with factorials. What is the probability that 2 patients have side effects?

$$P(X = 2) = \frac{20!}{2!18!} 0.03^2 0.97^{18}$$

```
[ ]: from scipy.stats import binom
import matplotlib.pyplot as plt; import numpy as np

n = 20; p = 0.03; k = 2;
p1 = binom.pmf(k=k,n=n,p=p);
print('Probability of {:d} out of {:d} = p1 = {:.4f}\n'.format(k,n,p1))

kv = np.arange(n+1);
pv = binom.pmf(k=kv,n=n,p=p);
plt.bar(kv, pv)

p2 = binom.cdf(k=k,n=n,p=p);
print('Probability of {:d} or less out of {:d} = p2 = {:.4f}\n'.
      format(k,n,p2))

p3 = 1-binom.cdf(k=k,n=n,p=p);
print('Probability of {:d} or more out of {:d} = p3 = {:.4f}\n'.
      format(k+1,n,p3))

p4 = binom.cdf(k=3,n=n,p=p) - binom.cdf(k=1,n=n,p=p);
print('Probability between 2 to 3 = {:.4f}\n'.format(p4))

mu = n*p; sig = np.sqrt(n*p*(1-p));
print('mean mu = {:.4f}, standard deviation = {:.4f}'.format(mu,sig))
print('usual range {:.4f}, {:.4f} '.format(mu-2*sig, mu+2*sig))
```

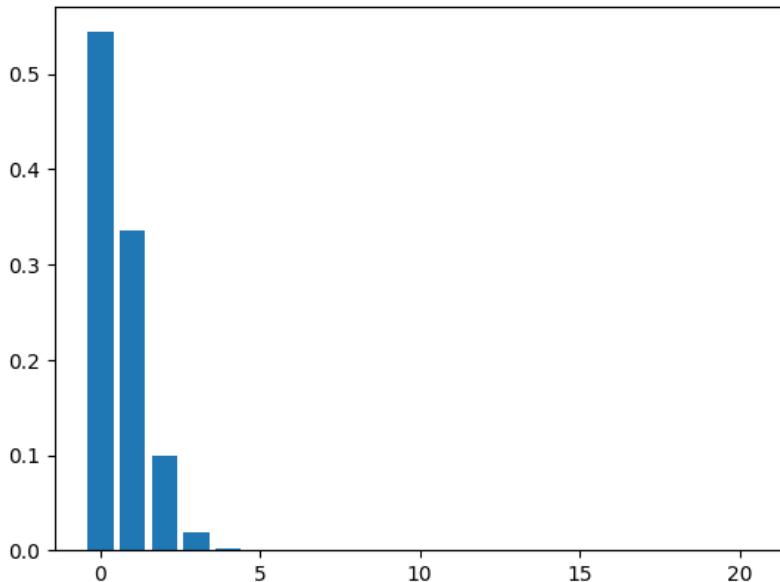
Probability of 2 out of 20 = p1 = 0.0988

Probability of 2 or less out of 20 = p2 = 0.9790

Probability of 3 or more out of 20 = p3 = 0.0210

Probability between 2 to 3 = 0.1172

mean mu = 0.6000, standard deviation = 0.7629  
 usual range -0.9258, 2.1258



The barplot above shows the probability distribution for the number of side effects.

### Example

Here is a quality control example. Suppose about 8% of all computer chips produced by a manufacturing firm are defective. Assume there are  $n = 45$  (**independent**) chips selected at random; each chip is either defective ( $p = 0.08$ ) or not ( $1 - p = 1 - 0.08 = 0.92$ ).

Once again, the questions are shown computationally; we only mention one Binomial formula with factorials.

What is the probability that 4 chips are defective?

$$P(X = 4) = \frac{45!}{4!41!} 0.08^4 0.92^{41}$$

```
[ ]: from scipy.stats import binom; import pandas as pd;
import matplotlib.pyplot as plt; import numpy as np;

n = 45; p = 0.08; k = 4;
p1 = binom.pmf(k=k,n=n,p=p);
print('Probability of {:d} out of {:d} = p1 = {:.4f}\n'.format(k,n,p1))

kv = np.arange(n+1);
pv = binom.pmf(k=kv,n=n,p=p);
plt.bar(kv, pv)
```

```

p2 = binom.cdf(k=k,n=n,p=p);
print('Probability of {:d} or less out of {:d} = p2 = {:.4f}\n'.
    format(k,n,p2))

p3 = 1-binom.cdf(k=k,n=n,p=p);
print('Probability of {:d} or more out of {:d} = p3 = {:.4f}\n'.
    format(k+1,n,p3))

# I would add here a calculation for different number of defective chips
kv = np.arange(11)
pv = 1-binom.cdf(k=kv,n=n,p=p);
df = pd.DataFrame({'kv':kv,'pv':pv});
print('Probability to get k or more defective chips:')
pd.set_option("display.precision", 4); print(df, '\n')

p4 = binom.cdf(k=9,n=n,p=p) - binom.cdf(k=1,n=n,p=p);
print('Probability between 2 to 9 = {:.4f}\n'.format(p4))

mu = n*p; sig = np.sqrt(n*p*(1-p));
print('mean mu = {:.4f}, standard deviation = {:.4f}'.format(mu,sig))
print('usual range {:.4f}, {:.4f}'.format(mu-2*sig, mu+2*sig))

```

Probability of 4 out of 45 = p1 = 0.1999

Probability of 4 or less out of 45 = p2 = 0.7098

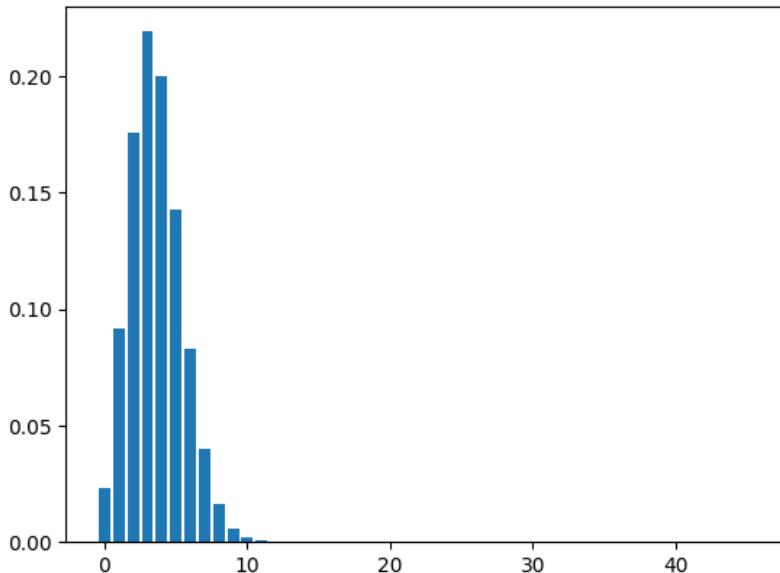
Probability of 5 or more out of 45 = p3 = 0.2902

Probability to get k or more defective chips:

	kv	pv
0	0	0.9765
1	1	0.8847
2	2	0.7090
3	3	0.4901
4	4	0.2902
5	5	0.1476
6	6	0.0650
7	7	0.0250
8	8	0.0084
9	9	0.0025
10	10	0.0007

Probability between 2 to 9 = 0.8822

mean mu = 3.6000, standard deviation = 1.8199  
 usual range -0.0398, 7.2398



Once again, the Figure above shows the probability distribution.

#### 4.3.3 Normal Approximation of the Binomial Distribution

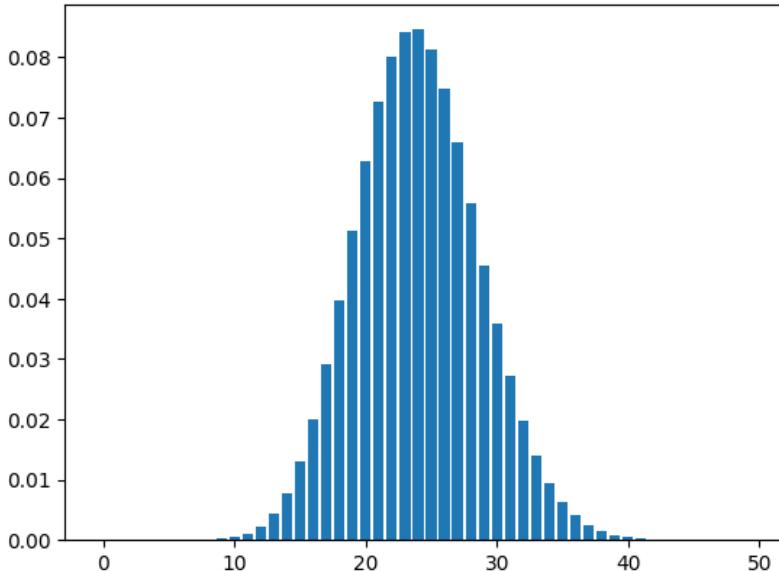
For larger sample sizes  $n$ , Binomial Distribution is well approximated by Normal Distribution. It also avoids the computation of large factorials  $n!$  which may not even fit in computer memory directly, although Python *binom.pmf()* and *binom.cdf()* can still compute precisely even for large sample sizes.

In the previous example about defective computer chips, the sample size was not small,  $n = 45$ , but the distribution was a little skewed right. Let's increase the sample (batch) size to  $n = 300$ , then as the Figure below shows, the binomial distribution looks exactly bell-shaped (normal distribution).

```
[ ]: from scipy.stats import binom; import pandas as pd;
      import matplotlib.pyplot as plt; import numpy as np;

n = 300; p = 0.08;
kv = np.arange(50);
pv = binom.pmf(k=kv,n=n,p=p);
plt.bar(kv, pv)

[ ]: <BarContainer object of 50 artists>
```



The binomial distribution  $B(n, p)$  with size  $n$  and probability  $p$  is **nearly normal** when the sample size is large enough to satisfy the **(success/failure) conditions**:

$$n \cdot p \geq 10 \quad \text{and} \quad n \cdot (1 - p) \geq 10$$

Then, the approximate normal has the same mean and standard deviation as the “parent” binomial distribution:

$$\mu = n \cdot p \quad \sigma = \sqrt{n \cdot p \cdot (1 - p)}$$

Returning to the defective chips problem:

For  $n = 45$  and  $p = 0.08$  then

$$n \cdot p = 45 \cdot 0.08 = 3.6 < 10, \quad n \cdot (1 - p) = 45 \cdot 0.92 = 41.4 \geq 10$$

so the success/failure conditions fail, the distribution is still a bit skewed, and normal approximation should not be used. For  $n = 300$  and  $p = 0.08$ , the conditions become:

$$n \cdot p = 300 \cdot 0.08 = 24 \geq 10, \quad n \cdot (1 - p) = 300 \cdot 0.92 = 276 \geq 10$$

so the success/failure conditions are satisfied and the distribution is bell-shaped. In this case, the normal approximation is appropriate.

Consider the probability that there would be 30 or more defective chips. Using Binomial formula directly, we obtain:

$$P(X \geq 30) = \sum_{k=30}^{300} \frac{300!}{k!(300-k)!} 0.08^k 0.92^{300-k}$$

This is impossible to compute directly term-by-term. Python's binom.cdf() can still give the exact answer to be compared to the normal approximation.

```
[ ]: from scipy.stats import binom, norm; import numpy as np;

# Exact computation
n = 300; p = 0.08;
pb = 1-binom.cdf(k=29,n=n,p=p);
print('Binomial probability 30 or more defective = pb = {:.6f}\n'.format(pb))

# Apply normal approximation
print('Success/failure validity check >= 10: n*p,n*(1-p)', n*p,n*(1-p))
mu = n*p; sig = np.sqrt(n*p*(1-p));
print('mean mu = {:.4f}, standard deviation = {:.4f}'.format(mu,sig))

# without the continuity correction:
z = (30 - mu)/sig; pn = 1-norm.cdf(z);
print('Standard z = {:.4f}, approx prob pn ={:4f}'
      .format(z,pn))
```

Binomial probability 30 or more defective = pb = 0.122641

Success/failure validity check >= 10: n\*p,n\*(1-p) 24.0 276.0  
 mean mu = 24.0000, standard deviation = 4.6989  
 Standard z = 1.2769, approx prob pn =0.1008

The answers are somewhat close, but it is not a reasonable approximation. When we approximate discrete distribution with a continuous probability distribution, the **continuity correction** is needed, which adds or subtracts 0.5 as necessary to correct for the difference.

```
[ ]: # with the continuity correction:
z = (30 -0.5 - mu)/sig; pn = 1-norm.cdf(z);
print('Standardized z = {:.4f}, approximate probability pn ={:4f}'
      .format(z,pn))
```

Standardized z = 1.1705, approximate probability pn =0.1209

This approximation with the continuity correction is much closer to the exact answer. Such continuity correction is even more important for the probability of a small interval. For example, what is the probability that the number of defective chips is between 25 and 35?

```
[ ]: pb = binom.cdf(k=35,n=n,p=p) - binom.cdf(k=24,n=n,p=p);
print('Binomial probability pb = {:.4f}\n'.format(pb))

print('Without the continuity correction:')
z1 = (25 - mu)/sig;
z2 = (35 - mu)/sig;
print('standardized scores: z1 = {:.4f}, z2 = {:.4f}'.format(z1, z2))
pn = norm.cdf(z2)-norm.cdf(z1);
print('approximate probability pn = {:.4f}'.format(pn))

print('\nWith the continuity correction:')
z1 = (25-0.5 - mu)/sig;
z2 = (35+0.5 - mu)/sig;
print('standardized scores: z1 = {:.4f}, z2 = {:.4f}'.format(z1, z2))
pn = norm.cdf(z2)-norm.cdf(z1);
print('approximate probability pn = {:.4f}'.format(pn))
```

Binomial probability pb = 0.4359

Without the continuity correction:  
standardized scores: z1 = 0.2128, z2 = 2.3410  
approximate probability pn = 0.4061

With the continuity correction:  
standardized scores: z1 = 0.1064, z2 = 2.4474  
approximate probability pn = 0.4504

The exact probability is much closer to the one computed with the continuity correction.

# 5

---

# Inferential Statistics and Tests for Proportions

---

## 5.1 Introduction to Inference

In the previous chapters, we developed the main ideas of Statistics and Probability. In this chapter, we start with the main topics of Inferential Statistics that makes inferences about the entire population from the available sample data.

For example, many polling agencies (Pew, Gallup, etc.) conduct regular presidential approval polls. A sample of approximately 1000-2000 people is randomly selected. A sample proportion  $\hat{p}$  estimates the true population proportion  $p$ . Consider a particular poll of sample size  $n = 1600$  with 660 approving presidential performance. Then sample proportion is  $\hat{p} = \text{yes}/\text{total} = 660/1600 = 0.4125$  - **point estimate**. To find the true population proportion  $p$  we must conduct a **census** of, say,  $N = 250$  million people, which is not realistic. However,  $\hat{p}$  approximates  $p$ .

A different random sample of the same size would produce a slightly different sample proportion, say,  $\hat{p} = \text{yes}/\text{total} = 665/1600 = 0.416$ . Variation in this point estimate from one sample to another is quantified by **sampling error**. Note that an **unbiased simple random sample** is always assumed to avoid any systematic error.

To understand the behavior of the **sampling error**, let's simulate its variation with a known population approval rating of  $p = 0.41$ . It is not known in reality, but it is set to this particular value for simulation purposes.

```
[ ]: # Simulation of Sample Proportions
import numpy as np; import seaborn as sns
from sklearn.utils import resample
p = 0.41    # true population proportion
N = 250000000; # population size
n = 1600; # sample size
numsamples = 20000 # number of samples in the simulation
numYes = p*N;
numNo = (1-p)*N;
```

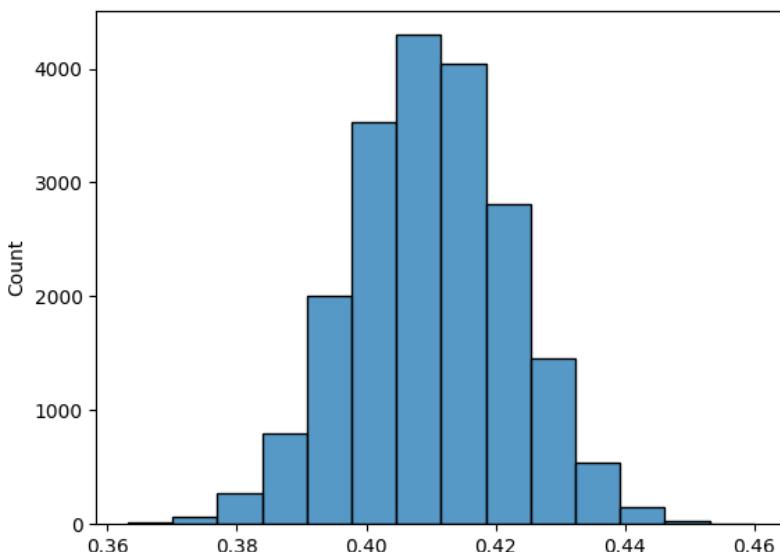
```

print('numYes, numNo = {:.1f}, {:.1f}'.format(numYes, numNo))
possible_entries = np.concatenate((np.repeat(1,numYes), np.repeat(0,numNo)))
pv = np.repeat(0.0,numsamples) # sample proportions
for j in range(numsamples):
    # resample below produces a SIMPLE RANDOM SAMPLE!!!
    onesample = resample(possible_entries, n_samples=n, replace=True)
    pv[j] = np.sum(onesample == 1) / n

sns.histplot(pv,bins=14) # create histogram of sample proportions
pvbar = np.mean(pv); # mean of sample proportions
pvsd1 = np.std(pv,ddof=1); # standard deviation of sample proportions
print('pvbar, pvsd1 = {:.4f}, {:.4f}'.format(pvbar, pvsd1))

```

numYes, numNo = 102500000.0, 147500000.  
 pvbar, pvsd1 = 0.4101, 0.0123



From the Figure above, the distribution of sample proportions (**sampling distribution**) is symmetric and bell-shaped (*normal distribution*). The center (*mean*) of this distribution is almost the same as the true population proportion  $p = 0.41$ . The spread of the sampling distribution (its standard deviation) is called **standard error SE** and is computed as well. Only one sample out of the entire population is really taken, so such sampling distribution is not observed, but it provides invaluable information about the properties of the point estimate.

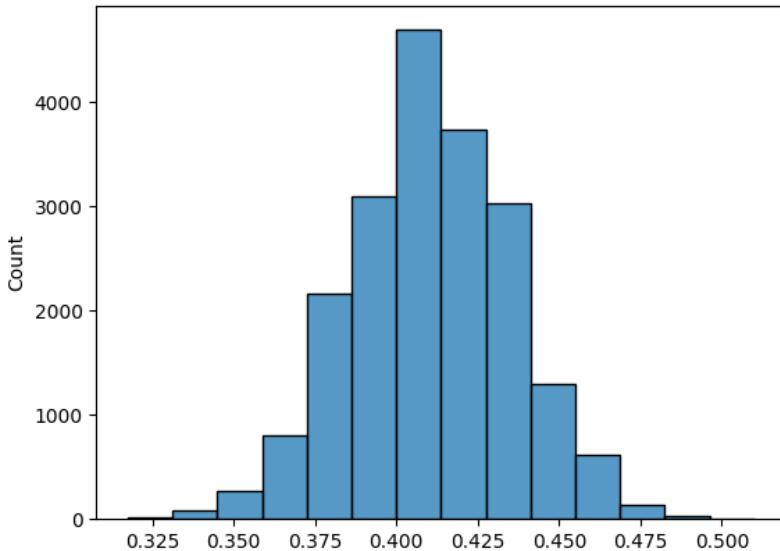
If a smaller sample size  $n = 400$  is taken, the sampling distribution is still bell-shaped and centered at the same true population proportion, but the

distribution is more spread out as shown in the Figure below.

```
[ ]: # Simulation of Sample Proportions
import numpy as np; import seaborn as sns
from sklearn.utils import resample
p = 0.41    # true population proportion
N = 250000000; # population size
n = 400; # sample size
numsamples = 20000 # number of samples in the simulation
numYes = p*N;
numNo = (1-p)*N;
print('numYes, numNo = {:.1f}, {:.1f}'.format(numYes, numNo))
possible_entries = np.concatenate((np.repeat(1,numYes), np.repeat(0,numNo)))
pv = np.repeat(0.0,numsamples) # sample proportions
for j in range(numsamples):
    # resample below produces a SIMPLE RANDOM SAMPLE!!!
    onesample = resample(possible_entries, n_samples=n, replace=True)
    pv[j] = np.sum(onesample == 1) / n

sns.histplot(pv,bins=14) # create histogram of sample proportions
pvbar = np.mean(pv); # mean of sample proportions
pvsd1 = np.std(pv,ddof=1); # standard deviation of sample proportions
print("pvbar, pvsd1 = {:.4f}, {:.4f}".format(pvbar, pvsd1))
```

numYes, numNo = 102500000.0, 147500000.0  
 pvbar, pvsd1 = 0.4101, 0.0247



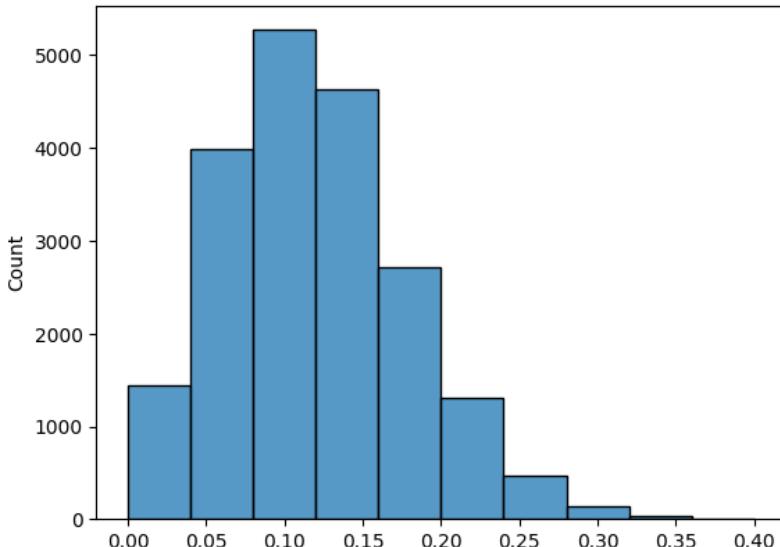
The standard error is about twice as much as the one obtained for the  $n = 1600$  sample size. However, if the sample size is very small  $n = 25$  and the true

proportion  $p = 0.1$  is further away from 0.5 than 0.41, the resulting sampling distribution is skewed right (not bell-shaped) as shown in the Figure below:

```
[ ]: # Simulation of Sample Proportions
import numpy as np; import seaborn as sns
from sklearn.utils import resample
p = 0.10    # true population proportion
N = 250000000; # population size
n = 25; # sample size
numsamples = 20000 # number of samples in the simulation
numYes = p*N;
numNo = (1-p)*N;
print('numYes, numNo = {:.1f}, {:.1f}'.format(numYes, numNo))
possible_entries = np.concatenate((np.repeat(1,numYes), np.repeat(0,numNo)))
pv = np.repeat(0.0,numsamples) # sample proportions
for j in range(numsamples):
    # resample below produces a SIMPLE RANDOM SAMPLE!!!
    onesample = resample(possible_entries, n_samples=n, replace=True)
    pv[j] = np.sum(onesample == 1) / n

sns.histplot(pv,bins=10) # create histogram of sample proportions
pvbar = np.mean(pv); # mean of sample proportions
pvsd1 = np.std(pv,ddof=1); # standard deviation of sample proportions
print('pvbar, pvsd1 = {:.4f}, {:.4f}'.format(pvbar, pvsd1))
```

```
numYes, numNo = 25000000.0, 225000000.0
pvbar, pvsd1 = 0.0999, 0.0598
```



The simulations above confirm the Central Limit Theorem:

### Central Limit Theorem for Proportions:

Provided that the observations are independent, and the sample size is sufficiently large to satisfy the **success/failure** conditions:  $np \geq 10$  and  $n(1 - p) \geq 10$ , the sample proportion  $\hat{p}$  follows a normal distribution with the mean  $\mu_{\hat{p}} = p$  and standard deviation (**standard error**)  $SE_{\hat{p}} = \sqrt{\frac{p(1-p)}{n}}$

First, let's verify the conditions of this CLT Theorem for the above examples:

1.  $n = 1600, p = 0.41 \implies np = 656 \geq 10, n(1 - p) = 944 \geq 10$
2.  $n = 400, p = 0.41 \implies np = 164 \geq 10, n(1 - p) = 236 \geq 10$
3.  $n = 25, p = 0.10 \implies np = 2.5 < 10, n(1 - p) = 22.5 \geq 10$

Thus, for the first two simulations, the CLT conditions hold, and normal distributions is a valid approximation, but in the third case it fails and it is skewed right. Independence of individual observations must hold and is ensured, for example, for a simple random sample of less than 10% of the population size. In an unlikely case when the sample size  $n$  exceeds 10% of the population size  $N$ , the standard error must be multiplied by a **finite correction factor**  $\sqrt{\frac{N-n}{N-1}}$ , which is very close to 1 for  $n < 0.1N$ .

Let's compare the results of the first simulation with  $n = 1600$  to the predictions of the CLT. In the simulation  $p = 0.41$ ; therefore, the corresponding normal distribution mean and standard errors are:

$$\mu = p = 0.41 \quad \text{and} \quad SE = \sqrt{\frac{p(1-p)}{n}} = \sqrt{\frac{0.41(1-0.41)}{1600}} \approx 0.0123$$

The theoretical standard error is very close to the simulation standard error 0.0122.

In reality, the true population proportion  $p$  is not known. Assume one poll of  $n = 1600$  is taken and  $x = 675$  participants approve the presidential performance, so sample proportion  $\hat{p} = 675/1200 \approx 0.422$ . We cannot check the success failure condition with unknown true population proportion  $p$ , so instead the sample proportion  $\hat{p}$  is **substituted**:

$$np \approx n\hat{p} = 1600 \cdot 0.422 = 675.2 \geq 10$$

$$n(1 - p) \approx n(1 - \hat{p}) = 1600 \cdot (1 - 0.422) = 924.8 \geq 10$$

Therefore, the sample proportion approximately follows a normal distribution with mean and standard error:

$$\mu = 0.422 \quad SE = \sqrt{\frac{p(1-p)}{n}} \approx \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} = \sqrt{\frac{0.422(1-0.422)}{1600}} = 0.01235$$

The standard error is extremely close to the one computed with the exact  $p = 0.41$ .

## 5.2 Confidence Interval for Proportions

From the previous section, we know that sample proportions have a normal distribution centered at the true population proportion. Sample proportions naturally vary from sample to sample, so a range of plausible values (**confidence interval**) would be a better description of true population proportion than one sample estimate. It is like using a net to catch a fish instead of trying to hit it with a spear.

Provided the conditions of CLT Theorem are satisfied,  $\hat{p}$  follows normal distribution with mean and standard deviation (called standard error) given by:

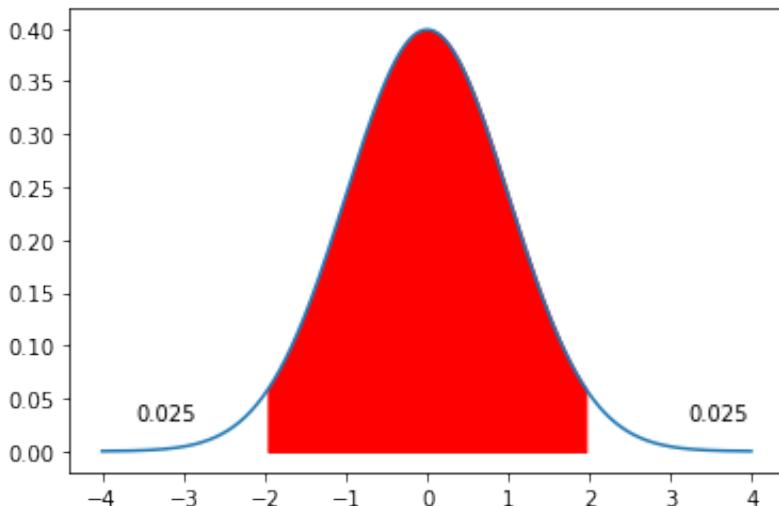
$$\mu = \hat{p} \text{ and } SE = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}.$$

Note that the unknown true population proportion  $p$  was substituted by its sample proportion approximation  $\hat{p}$ .

As we know, for a normal distribution, 95% of the data are within about 1.96 standard deviations away from the mean, as shown in the Figure below.

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm
xv = np.arange(-4, 4, 0.01)
plt.plot(xv, norm.pdf(xv, 0, 1))
px=np.arange(-1.96,1.96,0.01)
plt.fill_between(px,norm.pdf(px,0,1),color='r')
plt.text(-3.6,0.03,"0.025"); plt.text(3.2,0.03,"0.025")
```

```
[ ]: Text(3.2, 0.03, '0.025')
```



Therefore, the interval:

$$\text{point estimate} \pm 1.96 \cdot SE = \hat{p} \pm 1.96 \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

contains the true population proportion  $p$  with 95% certainty - **95% confidence interval (CI)**.

To better understand CIs, we perform a simulation of 100 such confidence intervals as shown in the Figure below. Each one has a slightly different center  $\hat{p}$  and slightly different length, but about 95% contain the true population proportion  $p$ . Therefore, if the procedure of selecting a simple random sample and computing confidence interval is repeated, about 95% of resulting CIs are expected to contain the true population proportion.

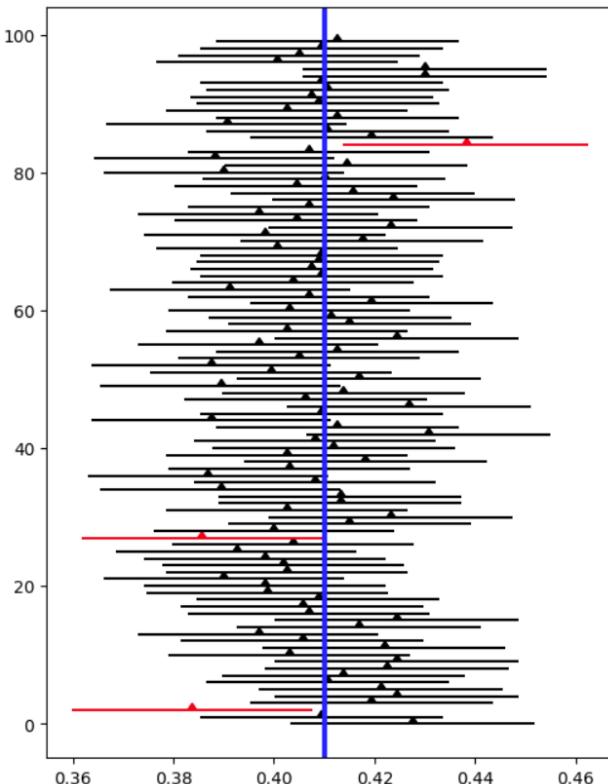
A few confidence intervals do not contain the true population proportion, but this does *not* invalidate the theory. It is a manifestation of natural sampling variability. Just as some observations naturally occur more than 1.96 standard deviations from the mean, analogously some point estimates naturally are further away from the true proportion. Confidence interval gives a plausible range of values. The values outside of it are implausible (much less likely), but not impossible!

```
[ ]: # Simulation of Sample Proportions
import numpy as np; import seaborn as sns
from sklearn.utils import resample; import matplotlib.pyplot as plt
p0 = 0.41    # true population proportion
N = 250000000; # population size
n = 1600; # sample size
numsamples = 100 # number of samples in the simulation
numYes = p0*N;
numNo = (1-p0)*N;
numin = 0; # number of p's inside
# print('numYes, numNo = ', numYes, numNo)
possible_entries = np.concatenate((np.repeat(1,numYes), np.repeat(0,numNo)))
CIL = np.repeat(0.0,numsamples); CIR = np.repeat(0.0,numsamples);
_, ax = plt.subplots(1, 1, figsize=(6, 8))
for j in range(numsamples):
    # resample below produces a SIMPLE RANDOM SAMPLE!!!
    onesample = resample(possible_entries, n_samples=n, replace=True)
    phat = np.sum(onesample == 1) / n
    SE = np.sqrt(phat*(1-phat)/n)
    CIL[j] = phat - 1.96*SE
    CIR[j] = phat + 1.96*SE
    if (p0 > CIL[j] and p0 < CIR[j]):
        # interval contains p
        numin = numin + 1
        ax.errorbar(phat, j, lolims=True, xerr=1.96*SE, yerr=0.0, 
→linestyle='', c='black')
    else:
```

```
# interval does not contain p
ax.errorbar(phat, j, lolims=True, xerr=1.96*SE, yerr=0.0, ls='--', c='red')

ax.axvline(p0, color='darkorange')
print('Proportion inside = ', numin/numsamples)
```

Proportion inside = 0.97



## Example

A sample of  $n = 1600$  randomly chosen voters is taken and  $x = 675$  of them approve presidential performance. Construct and interpret a 95% confidence interval for the population proportion.

```
[ ]: import numpy as np; from scipy.stats import norm; import pandas as pd
alpha = 0.05 # error level corresponding to 95% confidence level
x = 675; # number of Yes
n = 1600; # sample size
phat = x/n # sample proportion
print('success/failure conditions n*phat, n*(1-phat) <=10?? ', n*phat, n*(1-phat))
SE = np.sqrt(phat*(1-phat)/n); # standard error
zstar = norm.ppf(1-alpha/2); # z critical
MarginErr = zstar*SE; # margin of error
CIL = phat - MarginErr; CIR = phat + MarginErr;
df = pd.DataFrame({'phat':phat,'SE':SE,'zstar':zstar, \
'MarginErr':MarginErr,'CIL':CIL,'CIR':CIR},index=[0]);
pd.set_option("display.precision", 4); print(df,'\n')

success/failure conditions n*phat, n*(1-phat) <=10??  675.0 925.0
          phat      SE   zstar  MarginErr      CIL      CIR
0  0.4219  0.0123    1.96     0.0242  0.3977  0.4461
```

The interpretation is that we are 95% confident that the actual proportion of American voters who approve of the presidential performance is between  $CIL = 0.400$  and  $CIR = 0.446$ .

### 5.2.1 General Confidence Intervals

The most common level of confidence is 95%, but any level of confidence  $1 - \alpha$  equivalent to  $\alpha$  level of error can be used. The Figures below show confidence level areas for 90%, 95%, and 99% on the standard normal curve.

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; import pandas as pd
fig, axs = plt.subplots(3, 1, figsize=(6, 9), sharey=True)
xv = np.arange(-4, 4, 0.01)
ConfLevel = np.array([0.9,0.95,0.99])
alpha = 1-ConfLevel
zstar = norm.ppf(1-alpha/2);
df = pd.DataFrame({'ConfLevel':ConfLevel,'alpha':alpha,'zstar':zstar});

axs[0].plot(xv, norm.pdf(xv, 0, 1))
px=np.arange(-zstar[0],zstar[0],0.01)
axs[0].fill_between(px,norm.pdf(px,0,1),color='r')
axs[0].text(-3.6,0.03,"0.05"); axs[0].text(3.2,0.03,"0.05")

axs[1].plot(xv, norm.pdf(xv, 0, 1))
px=np.arange(-zstar[1],zstar[1],0.01)
axs[1].fill_between(px,norm.pdf(px,0,1),color='r')
```

```

axs[1].text(-3.6,0.03,"0.025"); axs[1].text(3.2,0.03,"0.025")

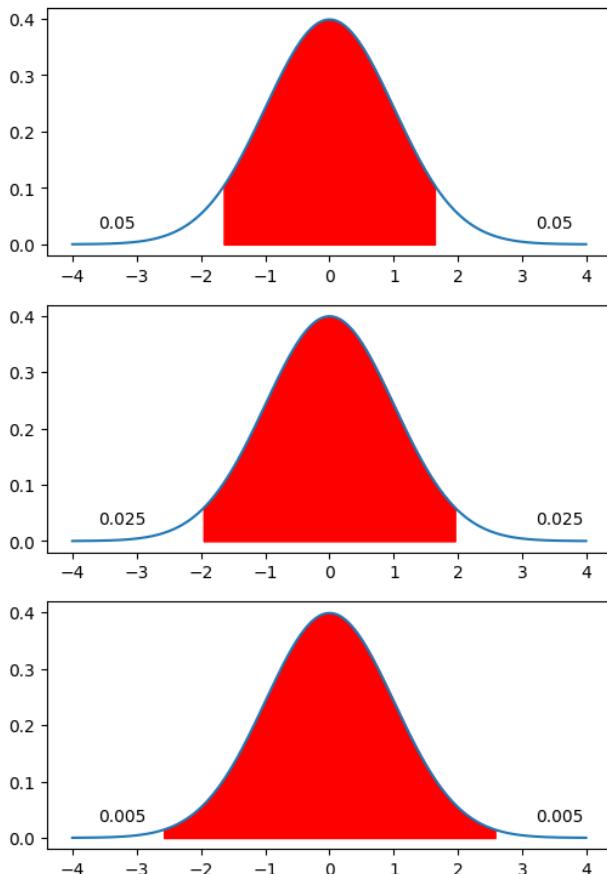
axs[2].plot(xv, norm.pdf(xv, 0, 1))
px=np.arange(-zstar[2],zstar[2],0.01)
axs[2].fill_between(px,norm.pdf(px,0,1),color='r')
axs[2].text(-3.6,0.03,"0.008"); axs[2].text(3.2,0.03,"0.005")
df

```

```

[ ]: ConfLevel    alpha      zstar
0         0.90    0.10   1.644854
1         0.95    0.05   1.959964
2         0.99    0.01   2.575829

```



The critical  $z^*$  is found in such a way that the area between  $-z^*$  and  $z^*$  under the standard normal distribution  $N(0, 1)$  corresponds to the confidence level. These  $z^*$  bounds for standard levels are given in the Python code above.

A **general confidence interval** for any critical  $z^*$  is:

$$\text{point estimate} \pm z^* \cdot SE = \hat{p} \pm z^* \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \quad (5.1)$$

As the level of confidence increases, the area of the confidence region in the middle increases, so the corresponding **critical  $z^*$  bound** increases and the interval becomes wider. Thus, we are more confident but about a wider interval. To use a fishing analogy again: to be more sure to trap the fish, a wider net must be used (**trade-off relationship**).

The best way to write a confidence interval is in terms of **margin of error**  $e$ :

$$e = z^* \cdot SE = z^* \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \Rightarrow CI = \hat{p} \pm e \quad (5.2)$$

### Example

In the previous example, construct 90%, 95%, and 99% confidence intervals.

```
[ ]: import numpy as np; from scipy.stats import norm; import pandas as pd
ConfLevel = np.array([0.9, 0.95, 0.99]) # confidence level
alpha = 1-ConfLevel # error level corresponding to confidence level
x = 675; # number of Yes
n = 1600; # sample size
phat = x/n # sample proportion
print('sample proportion phat = ', x , '/', n, '=', phat)
print('success/failure conditions n*phat, n*(1-phat) <=10?? ', n*phat, n*(1-phat))
SE = np.sqrt(phat*(1-phat)/n); # standard error
print("Standard error = sqrt(phat*(1-phat)/n) = ", SE)
zstar = norm.ppf(1-alpha/2); # z critical
MarginErr = zstar*SE; # margin of error
CIL = phat - MarginErr; CIR = phat + MarginErr;
df = pd.DataFrame({'ConfLevel':ConfLevel, 'alpha':alpha, 'zstar':zstar, 'SE':
    ↪SE, ↪
    'e':MarginErr, 'phat':phat, 'CIL':CIL, 'CIR':CIR});
pd.set_option("display.precision", 4); df
```

```
sample proportion phat = 675 / 1600 = 0.421875
success/failure conditions n*phat, n*(1-phat) <=10?? 675.0 925.0
Standard error = sqrt(phat*(1-phat)/n) = 0.012346469241624303
```

	ConfLevel	alpha	zstar	SE	e	phat	CIL	CIR
0	0.90	0.10	1.6449	0.0123	0.0203	0.4219	0.4016	0.4422
1	0.95	0.05	1.9600	0.0123	0.0242	0.4219	0.3977	0.4461
2	0.99	0.01	2.5758	0.0123	0.0318	0.4219	0.3901	0.4537

The table above shows column by column all the steps for each confidence level. The sample proportion  $\hat{p} = \frac{x}{n} = \frac{675}{1600} \approx 0.422$  and standard error  $SE = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \approx \sqrt{\frac{0.422(1-0.422)}{1600}} \approx 0.0123$  are the same regardless of the

level. The margin of error is the product  $e = z^* \cdot SE$ , which is increasing with confidence level. The confidence interval is  $\hat{p} \pm e$ , which gets wider and wider as the level of confidence increases.

There are common mistakes in the interpretation of the confidence intervals. First, the confidence intervals above apply to the *population proportion for the entire population of American voters*. Confidence intervals predict **nothing** about sample proportion from another sample! For example, if another random sample of 1600 voters is collected, it would be *wrong* to say that there is a 99% chance that the interval (39.01%, 45.37%) obtained in the previous example contains a new sample proportion. Also, it is better to avoid incorrect language implying that the true population proportion has, say, a 99% chance *of falling inside* the above confidence interval. The true population proportion is a fixed number - it does not vary. Instead, the confidence interval is variable and has, a chance to contain (catch) the true population proportion.

### 5.2.2 More Examples of Proportion Confidence Intervals

#### Example

A company claims that only about 8% of its widgets can be defective. A sample of  $n = 100$  of randomly chosen widgets is taken and  $x = 10$  of them turn out to be defective. Construct 90%, 95%, and 99% confidence intervals and interpret the results.

It is much more efficient to write a Python generic function to compute all confidence intervals in one shot.

```
[ ]: def OneProportionCI(x,n,ConfidenceLevels):
    import numpy as np; from scipy.stats import norm; import pandas as pd
    import matplotlib.pyplot as plt
    print('One Proportion Confidence Interval function')
    print('Number of successes x = ', x)
    print('Sample size n = ', n)
    print('Confidence Levels',ConfidenceLevels)

    phat = x/n; phat # sample proportion
    print('sample proportion phat = {:.2f}/{:d} = {:.4f}'.format(x,n,phat))

    ConfLevel = np.array(ConfidenceLevels) # confidence level
    alpha = 1-ConfLevel # error level corresponding to confidence level
    print('success/failure: n*phat = {:.3f}, n*(1-phat) = {:.3f} <=10?? ? '
          .format(n*phat, n*(1-phat)))
    SEci = np.sqrt(phat*(1-phat)/n); # standard error for CI
    print('Standard Error = SEci =sqrt(phat*(1-phat)/n) = ')
    print('      sqrt({:.4f}*(1-{:.4f})/{:d}) = {:.4f}'.
        format(phat,phat,n,SEci))
    zstar = norm.ppf(1-alpha/2); # z critical
    MarginErr = zstar*SEci; # margin of error
    CIL = phat - MarginErr; CIR = phat + MarginErr;
    df = pd.DataFrame({'ConfLevel':ConfLevel, 'zstar':zstar,'SEci':SEci, \
```

```
'MarginErr':MarginErr,'phat':phat,'CIL':CIL,'CIR':CIR});
pd.set_option("display.precision", 4); print(df, '\n')
```

```
OneProportionCI(x=10,n=100,ConfidenceLevels=[0.9,0.95,0.99])
```

```
One Proportion Confidence Interval function
Number of successes x = 10
Sample size n = 100
Confidence Levels [0.9, 0.95, 0.99]
sample proportion phat = 10.00/100 = 0.1000
success/failure: n*phat = 10.000, n*(1-phat) = 90.000 <=10??
Standard Error = SEci =sqrt(phat*(1-phat)/n) =
sqrt(0.1000*(1-0.1000)/100) = 0.0300
ConfLevel zstar SEci MarginErr phat CIL CIR
0 0.90 1.6449 0.03 0.0493 0.1 0.0507 0.1493
1 0.95 1.9600 0.03 0.0588 0.1 0.0412 0.1588
2 0.99 2.5758 0.03 0.0773 0.1 0.0227 0.1773
```

The most common 95% confidence interval contains the claimed 8% proportion of defective items, so *no reason to reject the company's claim*. In fact, in this case, all common levels of confidence produce intervals containing the claimed 8%.

### Example

To assess the efficacy of a new drug, a random sample of  $n = 159$  patients was taken and the drug was effective in 86% of them. The company claimed that it is at least 80% effective. Construct 90%, 95%, and 99% confidence intervals and interpret the results.

In this case, the actual number of people in which the drug was effective was not given, rather a percentage - 86%. Therefore, it is computed on the fly with  $x = 0.86 \cdot 159$ .

$$\hat{p} = \frac{x}{n} \Rightarrow x = \hat{p} \cdot n$$

```
[ ]: OneProportionCI(x=0.86*159,n=159,ConfidenceLevels=[0.9,0.95,0.99])
```

```
One Proportion Confidence Interval function
Number of successes x = 136.74
Sample size n = 159
Confidence Levels [0.9, 0.95, 0.99]
sample proportion phat = 136.74/159 = 0.8600
success/failure: n*phat = 136.740, n*(1-phat) = 22.260 <=10??
Standard Error = SEci =sqrt(phat*(1-phat)/n) =
sqrt(0.8600*(1-0.8600)/159) = 0.0275
ConfLevel zstar SEci MarginErr phat CIL CIR
0 0.90 1.6449 0.0275 0.0453 0.86 0.8147 0.9053
1 0.95 1.9600 0.0275 0.0539 0.86 0.8061 0.9139
2 0.99 2.5758 0.0275 0.0709 0.86 0.7891 0.9309
```

The 90% and 95% confidence intervals are completely above 80% confirming the company's claim, but the 99% confidence interval is not. Thus, the choice of the confidence level makes a difference. There is *no right or wrong choice* of the confidence level; it is a subjective decision guided by cost-benefit analysis. If there is already an established drug on the market, a higher 99% confidence level may be needed. On the other hand, if there is no drug at all, perhaps a 90% confidence level should be used to give this drug more of a chance.

### Example

A poll of US adults found that 26% of 310 Republicans supported a generic “National Health Plan”. A politician claims that more than 1/3 of Republicans supported such a plan. Construct standard confidence intervals and interpret the results.

```
[ ]: OneProportionCI(x=310*0.26,n=310,ConfidenceLevels=[0.9,0.95,0.99])
```

```
One Proportion Confidence Interval function
Number of successes x = 80.60000000000001
Sample size n = 310
Confidence Levels [0.9, 0.95, 0.99]
sample proportion phat = 80.60/310 = 0.2600
success/failure: n*phat = 80.600, n*(1-phat) = 229.400 <=10??
Standard Error = SEci =sqrt(phat*(1-phat)/n) =
sqrt(0.2600*(1-0.2600)/310) = 0.0249
   ConfLevel      zstar      SEci    MarginErr    phat      CIL      CIR
0        0.90    1.6449    0.0249     0.0410    0.26    0.2190    0.3010
1        0.95    1.9600    0.0249     0.0488    0.26    0.2112    0.3088
2        0.99    2.5758    0.0249     0.0642    0.26    0.1958    0.3242
```

The most common 95% confidence interval is well below  $1/3 \approx 0.3333 = 33.33\%$ ; therefore, the claim could be rejected. In fact, in this case, all common levels of confidence produce confidence intervals below the claimed  $1/3$ .

```
[ ]:
```

## 5.3 Hypothesis Testing for Proportions

In many situations, a claim is made about a population that requires a direct answer with a pre-defined certainty, which requires a **hypothesis testing framework**. We will introduce it with an example.

### Example

A politician claims that the population is indifferent to imposing 2-term limits for governors (50% supporting, 50% not). A poll of  $n = 1200$  of likely voters

is conducted, and  $x = 550$  of them supported it. Investigate this hypothesis claim at the *error (significance level)* of 5%.

The first step is to state **null and alternative hypotheses**. The **null hypothesis** is generally the skeptical perspective (“no difference”), agreeing with the status quo. The **alternative hypothesis** is rejecting this status quo. In this problem, it would be *not 50/50*, i.e. there is preference one way or another - majority supports or majority opposes (*two-sided* alternative).

**H0** :  $p = 0.5$  (there are 50% supporting term limits).

**H1** :  $p \neq 0.5$  (there is less than 50% or more than 50% supporting term limits).

The hypothesis test determines whether the sample proportion  $\hat{p} = \frac{550}{1200} = 0.4583$  is significantly different from the claimed null hypothesis proportion  $p_0 = 0.5$ .

In hypothesis testing, it is always **initially assumed that the null hypothesis H0 is true**. Then, the probability of observing the actual data or something even more extreme is found (**p-value**). Thus, assuming  $H0: p = 0.5$  is true sets **null proportion**  $p_0 = 0.5$  that is used in all computations.

First, check the conditions of applicability of the Central Limit Theorem:

1. The poll was based on a simple random sample, so *independence* holds.
2.  $n \cdot p_0 = 1200 \cdot 0.5 = 600 \geq 10$  and  $n \cdot (1 - p_0) = 1200 \cdot 0.5 = 600 \geq 10 \Rightarrow$  success/failure conditions hold.

Therefore, the distribution of sample proportions  $\hat{p}$  (**null distribution**) is normal with mean and standard deviation given by:

$$\mu = p_0 = 0.5 \text{ and } SE = \sqrt{\frac{p_0(1-p_0)}{n}} = \sqrt{\frac{0.5(1-0.5)}{1200}} = 0.014.$$

For the confidence interval approach of the previous section,  $p = \hat{p}$  was used in the success/failure conditions and the standard error (SE) computation. Now,  $H0$  is assumed true, so  $p = p_0$  is used.

The **p-value** is the probability to observe the sampling proportion  $\hat{p} = x/n = 0.4583$  or something even more extreme under our bell-shaped null distribution centered at  $p_0 = 0.5$  with standard error  $SE = 0.014$ .

The alternative  $H1 : p \neq 0.5$  is two-sided; therefore both left and right tails must be accounted for. The normal distribution is symmetric, so the p-value can be found as twice the area of a single tail (see Figure below).

To find this tail area, we first find z corresponding to the observed  $\hat{p} = 0.458$ :

$$z = \frac{\hat{p} - p_0}{SE} = \frac{0.458 - 0.5}{0.014} = -2.887$$

Then, Python is used to find the area under the tail and multiply it by 2 to obtain  $p\text{-value} = 0.0038924$ . It measures how likely to observe sample proportion  $\hat{p} = 0.458$  or something more extreme and is much less than given significance (error) level  $\alpha = 0.05$ , so the initial assumption  $H0: p = 0.5$  can be rejected. Thus, the proportion of voters supporting 2-term limits for governors  $\hat{p} = 0.458$  is significantly different from  $0.5 = 50\%$ . Note that even though  $\hat{p} = 0.458 < 0.5$ , it is incorrect to change the alternative hypothesis

H1:  $p \neq 0.5$  to one-sided H1:  $p < 0.5$  post-factum. It would have negative accuracy implications as will be explained later.

Using the confidence interval approach in the code below, we are 95% confident (for 5% significance error rate) that the true population proportion is between 43.01% and 48.65%, which does *not* contain the claimed H0:  $p = 0.5$ . This confirms our conclusion to reject H0.

Note that the confidence interval standard error computed with  $\hat{p}$ :  $SE_{ci} = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$  is *not* the same as standard error for hypothesis test computed with  $p_0$ :  $SE = \sqrt{\frac{p_0(1-p_0)}{n}}$ . Therefore, in rare **borderline cases**, the conclusions on accepting or rejecting H0 may be the opposite. In Chapter 7 on hypothesis tests for means, standard error for the confidence interval will be the same as for the hypothesis test, so the conclusions will always match. The Python code below summarizes these steps into one function.

```
[38]: def OneProportionTest(x,n,p0,ConfidenceLevels):
    import numpy as np; from scipy.stats import norm; import pandas as pd
    import matplotlib.pyplot as plt
    print('One Proportion Test function')
    print('Number of successes x = ', x)
    print('Sample size n = ', n)
    print('Null hypothesis claimed proportion p0 = ', p0)
    print('Confidence Levels',ConfidenceLevels,'\\n')

    print('H0: p = p0')
    print('H1: p not p0  or one-sided test\\n')

    phat = x/n; phat  # sample proportion
    print('sample proportion phat = {:.2f}/{:d} = {:.4f}'.format(x,n,phat))

    print('Hypothesis Testing Approach:')
    print('success/failure: n*p0 = {:.4f}, n*(1-p0) = {:.4f}  <=10?? '
          .format(n*p0, n*(1-p0)))
    SE = np.sqrt(p0*(1-p0)/n);  # standard error
    print('Standard Error = SE =sqrt(p0*(1-p0)/n) = ')
    print('           sqrt({:.4f}*(1-{:.4f})/{:d}) = {:.4f}'.format(p0,p0,n,SE))
    z = (phat-p0)/SE;  # standardized statistic
    print('Standardized z = (phat-p0)/SE = ({:.4f}-{:.4f})/{:.4f}={:.4f}'
          .format(phat,p0,SE,z))
    pval2 = 2*(1-norm.cdf(np.abs(z)));  # 2 sided p-value
    print('P-values: 2-sided = {:.10f}, 1-sided = {:.10f}\\n'.format(pval2,pval2/2))

    # P-value graphical illustration FYI-----
    fig, axs = plt.subplots(1, 2, figsize=(10, 3), sharey=False)
    xv = np.arange(p0-4*SE, p0+4*SE, 0.001);  diff = np.abs(p0-phat)
    axs[0].plot(xv, norm.pdf(xv, p0, SE))
    xL=np.arange(p0-4*SE,p0-diff,0.001)
    axs[0].fill_between(xL,norm.pdf(xL,p0,SE),color='r')
    xR=np.arange(p0+diff,p0+4*SE,0.001)
    axs[0].fill_between(xR,norm.pdf(xR,p0,SE),color='b')
```

```

xv = np.arange(-4, 4, 0.001)
axs[1].plot(xv, norm.pdf(xv, 0, 1))
xL=np.arange(-4,-np.abs(z),0.001)
axs[1].fill_between(xL,norm.pdf(xL,0,1),color='r')
xR=np.arange(np.abs(z),4,0.001)
axs[1].fill_between(xR,norm.pdf(xR,0,1),color='b')
#-----

ConfLevel = np.array(ConfidenceLevels) # confidence level
alpha = 1-ConfLevel # error level corresponding to confidence level
print('Confidence Interval Approach: \n')
print('success/failure: n*phat = {:.3f}, n*(1-phat) = {:.3f} <=10?? '\
      .format(n*phat, n*(1-phat)))
SEci = np.sqrt(phat*(1-phat)/n); # standard error for CI
print('Standard Error = SEci =sqrt(phat*(1-phat)/n) =')
print('      sqrt({:.4f}*(1-{:.4f})/{:d}) = {:.4f} )'.\
      format(phat,phat,n,SEci))
zstar = norm.ppf(1-alpha/2); # z critical
MarginErr = zstar*SEci; # margin of error
CIL = phat - MarginErr; CIR = phat + MarginErr;
df = pd.DataFrame({'ConfLevel':ConfLevel,'zstar':zstar,'SEci':SEci, \
                    'MarginErr':MarginErr,'phat':phat,'CIL':CIL,'CIR':CIR});
pd.set_option("display.precision", 4); print(df, '\n')

```

[ ]: OneProportionTest(x=550,n=1200,p0=0.5,ConfidenceLevels=[0.9,0.95,0.99])

```

One Proportion Test function
Number of successes x = 550
Sample size n = 1200
Null hypothesis claimed proportion p0 = 0.5
Confidence Levels [0.9, 0.95, 0.99]

```

```

H0: p = p0
H1: p not p0 or one-sided test

```

```

sample proportion phat = 550.00/1200 = 0.4583
Hypothesis Testing Approach:
success/failure: n*p0 = 600.0000, n*(1-p0) = 600.0000 <=10??
Standard Error = SE =sqrt(p0*(1-p0)/n) =
      sqrt(0.5000*(1-0.5000)/1200) = 0.0144
Standardized z = (phat-p0)/SE = (0.4583-0.5000)/0.0144=-2.8868
P-values: 2-sided = 0.0038924171, 1-sided =0.0019462086

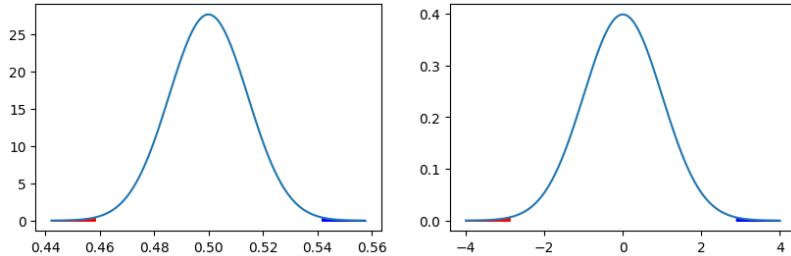
```

Confidence Interval Approach:

```

success/failure: n*phat = 550.000, n*(1-phat) = 650.000 <=10??
Standard Error = SEci =sqrt(phat*(1-phat)/n) =
      sqrt(0.4583*(1-0.4583)/1200 = 0.0144 )
ConfLevel   zstar   SEci   MarginErr   phat      CIL      CIR
0          0.90   1.6449  0.0144     0.0237  0.4583  0.4347  0.4820
1          0.95   1.9600  0.0144     0.0282  0.4583  0.4301  0.4865
2          0.99   2.5758  0.0144     0.0370  0.4583  0.4213  0.4954

```



### 5.3.1 More Proportion Hypothesis Tests Examples

#### Example

A Company claims that only about 9% of its widgets are defective. A random sample of 120 widgets is taken and 14 of them turn out to be defective. Test the company's claim at a 5% level.

**H0** :  $p = 0.09$  (company's claim - **assume true**).

**H1** :  $p \neq 0.09$  (not what they claim).

```
[ ]: OneProportionTest(x=14,n=120,p0=0.09,ConfidenceLevels=[0.9,0.95,0.99])
```

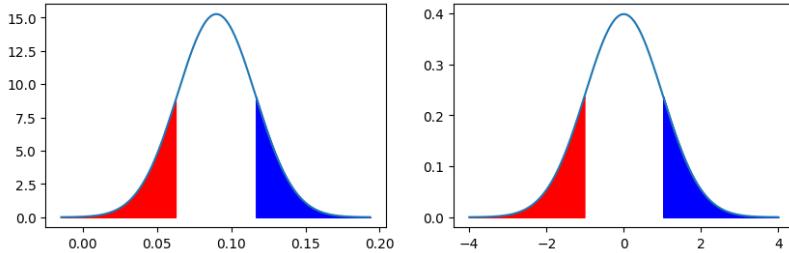
```
One Proportion Test function
Number of successes x = 14
Sample size n = 120
Null hypothesis claimed proportion p0 = 0.09
Confidence Levels [0.9, 0.95, 0.99]
```

```
H0: p = p0
H1: p not p0 or one-sided test
```

```
sample proportion phat = 14.00/120 = 0.1167
Hypothesis Testing Approach:
success/failure: n*p0 = 10.8000, n*(1-p0) = 109.2000 <=10??
Standard Error = SE =sqrt(p0*(1-p0)/n) =
sqrt(0.0900*(1-0.0900)/120) = 0.0261
Standardized z = (phat-p0)/SE = (0.1167-0.0900)/0.0261=1.0207
P-values: 2-sided = 0.3073751089, 1-sided =0.1536875545
```

Confidence Interval Approach:

```
success/failure: n*phat = 14.000, n*(1-phat) = 106.000 <=10??
Standard Error = SEci =sqrt(phat*(1-phat)/n) =
sqrt(0.1167*(1-0.1167)/120 = 0.0293 )
ConfLevel      zstar      SEci    MarginErr      phat        CIL        CIR
0            0.90    1.6449   0.0293      0.0482    0.1167    0.0685    0.1649
1            0.95    1.9600   0.0293      0.0574    0.1167    0.0592    0.1741
2            0.99    2.5758   0.0293      0.0755    0.1167    0.0412    0.1922
```



The hypothesis test tries to infer whether the sample proportion  $\hat{p} = 0.117$  is significantly different from  $H_0: p_0 = 0.09$ . This null value  $p_0$  is used to check the assumptions of the CLT. The sample of widgets is assumed to be a simple random sample, so the observations are *independent*. Also, the code above shows that  $n \cdot p_0 \geq 10$  and  $n \cdot (1 - p_0) \geq 10$ , so success/failure conditions are satisfied. Therefore, the sampling distribution of  $\hat{p}$  (**null distribution**) is normal with mean and standard deviation given by  $\mu = p_0$  and  $SE = \sqrt{\frac{p_0(1-p_0)}{n}}$  (see the code above).

The **p-value** is the probability to see the observed sampling proportion  $\hat{p}$  or something even more extreme under this bell-shaped normal null distribution. The z corresponding to the observed  $\hat{p}$  is:

$$z = \frac{\hat{p} - p_0}{SE} = \frac{0.117 - 0.09}{0.026} = 1.021$$

The alternative  $H_1 : p \neq p_0$  is two-sided, so both tails add up to p-value =  $0.3073751 > 0.05 = \alpha$ . Therefore, there is *not* enough evidence to reject the initial assumption  $H_0: p = p_0$  - no reason to reject the company's claim.

Also, from a confidence interval approach, we are 95% confident (for 5% significance error rate) that the true population proportion is between 5.92% and 17.41%, which contains the claimed  $H_0: p = 0.09$ . This confirms the conclusion *not* to reject  $H_0$ .

### Example

About 7% percent of the population of some country is left-handed. The rest are right-handed, and about 1% who are ambidextrous (no dominant hand). In a random sample of 300 children from a particular region, 32 are left-handed. Do these data provide evidence that the 7% value is inaccurate for children in this region? Use a stricter 1% level of significance.

**H0** :  $p = 0.07$  (standard percentage - **assume true**).

**H1** :  $p \neq 0.07$  (not standard percentage).

```
[ ]: OneProportionTest(x=32,n=300,p0=0.07,ConfidenceLevels=[0.9,0.95,0.99])
```

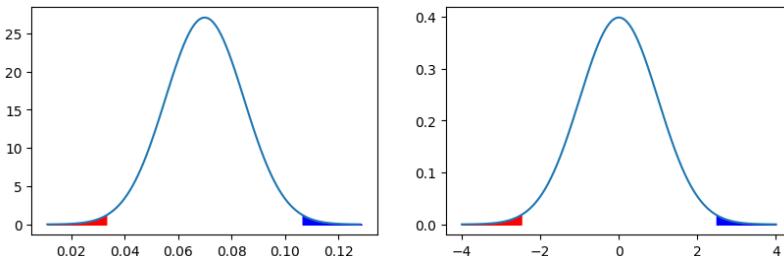
```
One Proportion Test function
Number of successes x = 32
Sample size n = 300
Null hypothesis claimed proportion p0 = 0.07
Confidence Levels [0.9, 0.95, 0.99]

H0: p = p0
H1: p not p0 or one-sided test

sample proportion phat = 32.00/300 = 0.1067
Hypothesis Testing Approach:
success/failure: n*p0 = 21.0000, n*(1-p0) = 279.0000 <=10??
Standard Error = SE =sqrt(p0*(1-p0)/n) =
sqrt(0.0700*(1-0.0700)/300) = 0.0147
Standardized z = (phat-p0)/SE = (0.1067-0.0700)/0.0147=2.4891
P-values: 2-sided = 0.0128068551, 1-sided =0.0064034276
```

Confidence Interval Approach:

```
success/failure: n*phat = 32.000, n*(1-phat) = 268.000 <=10??
Standard Error = SECi =sqrt(phat*(1-phat)/n) =
sqrt(0.1067*(1-0.1067)/300 = 0.0178 )
ConfLevel      zstar      SECi    MarginErr      phat       CIL       CIR
0            0.90    1.6449    0.0178      0.0293    0.1067    0.0774    0.1360
1            0.95    1.9600    0.0178      0.0349    0.1067    0.0717    0.1416
2            0.99    2.5758    0.0178      0.0459    0.1067    0.0608    0.1526
```



The hypothesis test tries to infer whether the sample proportion  $\hat{p} = 0.107$  is significantly different from  $H_0: p_0 = 0.07$ . This null value  $p_0$  is used to check the assumptions of the Central Limit Theorem (CLT). The sample of children is assumed to be a simple random sample, so the observations are *independent*. Also, the code above shows that  $n \cdot p_0 \geq 10$  and  $n \cdot (1 - p_0) \geq 10$ , so success/failure conditions are satisfied. Therefore, the sampling distribution of  $\hat{p}$  (**null distribution**) is normal with mean and standard deviation given by  $\mu = p_0$  and  $SE = \sqrt{\frac{p_0(1-p_0)}{n}}$  (see the code and the Figure above).

The **p-value** is the probability to see the observed sampling proportion  $\hat{p}$  or something even more extreme under this bell-shaped normal null distribution.

The z corresponding to the observed  $\hat{p}$  is:

$$z = \frac{\hat{p} - p_0}{SE} = \frac{0.107 - 0.07}{0.015} = 2.489$$

The alternative  $H1 : p \neq p_0$  is two-sided, so both tails add up to p-value  $= 0.0128069 > 0.01 = \alpha$ . Therefore, there is *not* enough evidence to reject the initial assumption  $H0: p = p_0$  - no reason to reject the null claim. Note, however, that if we used the usual  $\alpha = 0.05$  level, we would have rejected the  $H0$ .

Also, from a confidence interval approach, we are 99% confident (for 1% significance error rate) that the true population proportion is between 6.08% and 15.26%, which contains the claimed  $H0: p = 0.07$ . This confirms our conclusion not to reject  $H0$ . Note again that at 95% level, the CI is between 7.17% and 14.16% and does not contain  $H0: p = 0.07$  which would have resulted in the rejection of  $H0$ .

So far, only two-sided hypothesis tests have been considered. It is often not known in advance which way the data lands, so it is a safer, more conservative approach (medical studies are usually two-sided). A one-sided hypothesis may *overlook data supporting the opposite conclusion*, but one-sided tests are still quite common. Here is an example.

### Example

A School district claims that **at least** 80% of its students passed the SAT. The State education board needs to check their claim. In a random sample of 200 students, 149 passed. Test at 5% error level.

The Python function written above does not require any changes for one-sided tests since it always prints both two-sided and one-sided p-values (one-sided p-value is just half of the two-sided). However, the interpretation changes.

The "at least" in the problem formulation implies a **one-sided test** and the state board is trying to refute the school district's claim, so the alternative is to the left. Whether a one-sided test refutes or supports the claim is somewhat subtle, but important, decision.

**H0** :  $p = 0.80$  (district claim - **assume true**).

**H1** :  $p < 0.80$  (trying to refute their claim).

```
[1]: OneProportionTest(x=149,n=200,p0=0.80,ConfidenceLevels=[0.9,0.95,0.99])
```

```
One Proportion Test function
Number of successes x = 149
Sample size n = 200
Null hypothesis claimed proportion p0 = 0.8
```

Confidence Levels [0.9, 0.95, 0.99]

H0:  $p = p_0$

H1:  $p \neq p_0$  or one-sided test

sample proportion phat = 149.00/200 = 0.7450

Hypothesis Testing Approach:

success/failure:  $n \cdot p_0 = 160.0000$ ,  $n \cdot (1-p_0) = 40.0000 \leq 10??$

Standard Error = SE =  $\sqrt{p_0 \cdot (1-p_0) / n} =$

$\sqrt{0.8000 \cdot (1-0.8000) / 200} = 0.0283$

Standardized z =  $(\text{phat}-p_0)/\text{SE} = (0.7450-0.8000)/0.0283 = -1.9445$

P-values: 2-sided = 0.0518299272, 1-sided = 0.0259149636

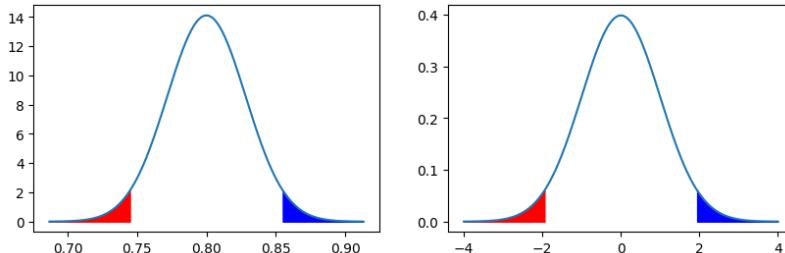
Confidence Interval Approach:

success/failure:  $n \cdot \text{phat} = 149.000$ ,  $n \cdot (1-\text{phat}) = 51.000 \leq 10??$

Standard Error = SEci =  $\sqrt{\text{phat} \cdot (1-\text{phat}) / n} =$

$\sqrt{0.7450 \cdot (1-0.7450) / 200} = 0.0308$ )

ConfLevel	zstar	SEci	MarginErr	phat	CIL	CIR
0	0.90	1.6449	0.0308	0.0507	0.745	0.6943
1	0.95	1.9600	0.0308	0.0604	0.745	0.6846
2	0.99	2.5758	0.0308	0.0794	0.745	0.6656



The hypothesis test tries to infer whether the sample proportion  $\hat{p} = 0.745$  is significantly **less** than H0:  $p_0 = 0.80$ . This null value  $p_0$  is used to check the assumptions of the CLT. The sample of students is assumed to be a simple random sample, so the observations are *independent*. Also, the code above shows that  $n \cdot p_0 \geq 10$  and  $n \cdot (1 - p_0) \geq 10$ , so success/failure conditions are satisfied. Therefore, the sampling distribution of  $\hat{p}$  (**null distribution**) is normal with mean and standard deviation given by  $\mu = p_0$  and  $SE = \sqrt{\frac{p_0(1-p_0)}{n}}$  (see the code above).

The **p-value** is the probability of seeing the observed sampling proportion  $\hat{p}$  or something even more extreme (to the left) under this bell-shaped normal null distribution.

The z corresponding to the observed  $\hat{p}$  is:

$$z = \frac{\hat{p} - p_0}{SE} = \frac{0.745 - 0.8}{0.028} = -1.944$$

The alternative  $H1: p < 0.80$  is one-sided, so only the left tail area gives p-value  $= 0.025915 < 0.05 = \alpha$ . Thus, the state officials can reject the district's claim. My code provides both two-tailed and one-tailed p-values. If a two-sided test were performed, then p-value  $= 0.0518299 > 0.05$ , so we would have failed to reject the  $H0$ .

Note that a one-sided test effectively doubles the chance of an error.

If  $\hat{p} < p_0$ , a one-sided alternative  $H1: p < p_0$  implies that any observation in the lower 5% tail of the null distribution rejects  $H0$ .

If  $\hat{p} > p_0$ , a one-sided alternative  $H1: p > p_0$  implies that any observation in the upper 5% tail of the null distribution rejects  $H0$ .

However, if  $H0$  is actually true, there is now a 10% chance of being in one of the two tails, not 5%.

The confidence interval method introduced before is always two-sided, therefore the confidence level has to be adjusted to  $1 - 2\alpha = 0.90$ , not  $1 - \alpha = 0.95$ . Therefore, we are 90% confident (10% significance error) that the interval (69.43%, 79.57%) contains true population proportion. This interval does not contain the claimed  $H0: p = 0.80$ , which confirms rejection of  $H0$ . Note that the 95% confidence interval (68.46%, 80.54%) contains the claimed  $H0: p = 0.8$ , so the conclusion would have been opposite. This underlines the difficulty of one-sided test presupposing left or right direction in addition to the aforementioned loss of accuracy.

Note that if, say, 165 students passed the test, then  $\hat{p} = 165/200 = 0.825 > 0.80$ . Thus,  $\hat{p}$  is already opposite to the alternative  $H1 : p < 0.80$ ; therefore it is pointless to continue with the calculation - the data supports  $H0$ . Only because the actual  $\hat{p} = 149/200 = 0.745 < 0.80$ , there was a point to continue.

Moreover, in case the sample proportion is below the claimed 80%, then the question becomes if it is significantly below. Assume, for example, that  $\hat{p} = \frac{155}{200} = 0.775 < 0.80$ , then the code and the Figure below give a very large p-value, for which we cannot reject  $H0$ . Therefore, not only should  $\hat{p}$  be below 0.80, it has to be sufficiently below to reject it.

```
[ ]: OneProportionTest(x=155,n=200,p0=0.80,ConfidenceLevels=[0.9,0.95,0.99])
```

```
One Proportion Test function
Number of successes x = 155
Sample size n = 200
Null hypothesis claimed proportion p0 = 0.8
Confidence Levels [0.9, 0.95, 0.99]
```

```
H0: p = p0
H1: p not p0 or one-sided test
```

```

sample proportion phat = 155.00/200 = 0.7750
Hypothesis Testing Approach:
success/failure: n*p0 = 160.0000, n*(1-p0) = 40.0000 <=10??
Standard Error = SE =sqrt(p0*(1-p0)/n) =
sqrt(0.8000*(1-0.8000)/200) = 0.0283
Standardized z = (phat-p0)/SE = (0.7750-0.8000)/0.0283=-0.8839
P-values: 2-sided = 0.3767591178, 1-sided =0.1883795589

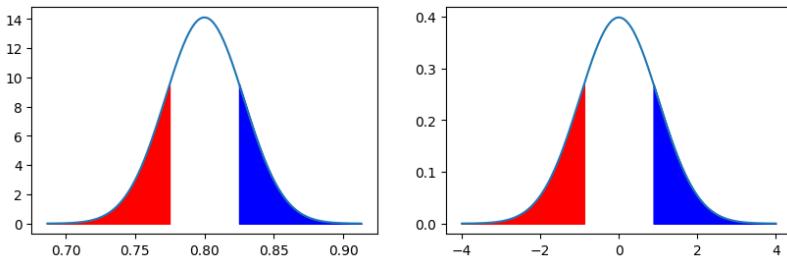
```

Confidence Interval Approach:

```

success/failure: n*phat = 155.000, n*(1-phat) = 45.000 <=10??
Standard Error = SEci =sqrt(phat*(1-phat)/n) =
sqrt(0.7750*(1-0.7750)/200 = 0.0295 )
ConfLevel      zstar      SEci   MarginErr    phat      CIL      CIR
0            0.90    1.6449  0.0295     0.0486  0.775  0.7264  0.8236
1            0.95    1.9600  0.0295     0.0579  0.775  0.7171  0.8329
2            0.99    2.5758  0.0295     0.0761  0.775  0.6989  0.8511

```



The next example is also one-sided illustrating substantiating rather than refuting the hypothesis.

### Example

Assume a random sample of 1000 of a top State University students is taken and 629 students are from this state. Can we conclude that **more than 60%** of students are in-state at a 5% level of significance?

**H0** :  $p = 0.60$  ( assume true).

**H1** :  $p > 0.60$

The test is one-sided again. However, this time, we are not refuting H0, but rather trying to substantiate it.

```
[ ]: OneProportionTest(x=629,n=1000,p0=0.60,ConfidenceLevels=[0.9,0.95,0.99])
```

```

One Proportion Test function
Number of successes x = 629
Sample size n = 1000
Null hypothesis claimed proportion p0 = 0.6
Confidence Levels [0.9, 0.95, 0.99]

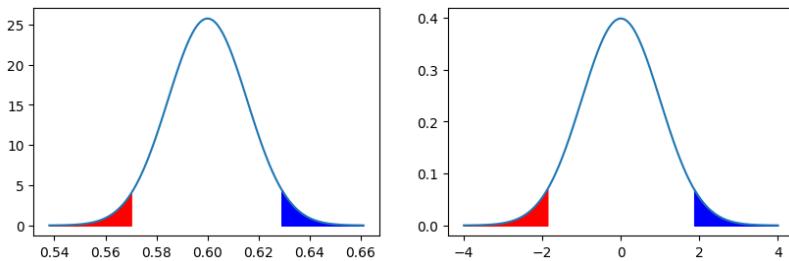
```

H0:  $p = p_0$   
 H1:  $p \neq p_0$  or one-sided test

```
sample proportion phat = 629.00/1000 = 0.6290
Hypothesis Testing Approach:
success/failure: n*p0 = 600.0000, n*(1-p0) = 400.0000 <=10??
Standard Error = SE =sqrt(p0*(1-p0)/n) =
sqrt(0.6000*(1-0.6000)/1000) = 0.0155
Standardized z = (phat-p0)/SE = (0.6290-0.6000)/0.0155=1.8719
P-values: 2-sided = 0.0612146350, 1-sided =0.0306073175
```

Confidence Interval Approach:

```
success/failure: n*phat = 629.000, n*(1-phat) = 371.000 <=10??
Standard Error = SEci =sqrt(phat*(1-phat)/n) =
sqrt(0.6290*(1-0.6290)/1000 = 0.0153 )
ConfLevel      zstar      SEci   MarginErr    phat      CIL      CIR
0            0.90  1.6449  0.0153     0.0251  0.629  0.6039  0.6541
1            0.95  1.9600  0.0153     0.0299  0.629  0.5991  0.6589
2            0.99  2.5758  0.0153     0.0393  0.629  0.5897  0.6683
```



The hypothesis test tries to infer whether the sample proportion  $\hat{p} = 0.629$  is significantly **greater** than  $H_0: p_0 = 0.60$ . The assumptions of the CLT hold as before. Therefore, the sampling distribution of  $\hat{p}$  (**null distribution**) is **normal** with mean and standard deviation given by  $\mu = p_0$  and  $SE = \sqrt{\frac{p_0(1-p_0)}{n}}$  (see the code).

The **p-value** is the probability to see the observed sampling proportion  $\hat{p}$  or something even more extreme under normal null distribution. The  $z$  corresponding to the observed  $\hat{p}$  is:

$$z = \frac{\hat{p} - p_0}{SE} = \frac{0.629 - 0.60}{0.0155} = 1.879$$

The alternative  $H_1: p > 0.60$  is one-sided, so only the right tail area shown in the Figure below gives you  $p\text{-value} = 0.030607 < 0.05 = \alpha$ . Therefore, there is enough evidence to reject the initial assumption. Thus, there are more than 60% of in-state students (the one-sided alternative claim has been

substantiated). Note that if the test was two-sided, the p-value would have been  $0.06121 > 0.05$  and we would have failed to reject the  $H_0$ .

As in the previous example with one-sided test, we have to use  $1 - 2\alpha = 0.90$  confidence level, not  $1 - \alpha = 0.95$ . Therefore, we are only 90% confident (for 10% significance error rate) that the true population proportion is between 60.39% and 65.41%, which does not contain the claimed  $H_0: p = 0.60$ . This confirms our conclusion to reject  $H_0$ . On the other hand, the 95% confidence interval between 59.91% and 65.89% contains the claimed  $H_0: p = 0.60$ , so it would have given us opposite conclusion from the hypothesis test result.

### Example

In a large survey, one year people were asked if they thought unemployment would increase, and 46% thought that it would increase. Next year, a small opinion poll was conducted, and 282 out of 649 said that they thought unemployment would increase. At the 10% level, is there enough evidence to show that the proportion of citizens who believe unemployment would increase is **less than** the proportion who felt it would increase in the earlier survey?

Note that in the code, we added an unusually low confidence level 0.8; we will need it later.

**H0** :  $p = 0.46$  (**assume true**).

**H1** :  $p < 0.46$ .

```
[ ]: OneProportionTest(x=282,n=649,p0=0.46,ConfidenceLevels=[0.8,0.9,0.95,0.99])
```

```
One Proportion Test function
Number of successes x = 282
Sample size n = 649
Null hypothesis claimed proportion p0 = 0.46
Confidence Levels [0.8, 0.9, 0.95, 0.99]

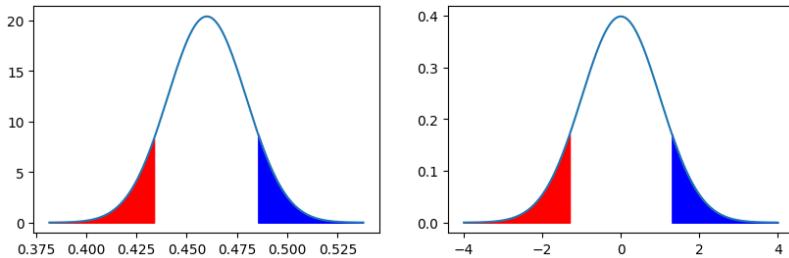
H0: p = p0
H1: p not p0 or one-sided test

sample proportion phat = 282.00/649 = 0.4345
Hypothesis Testing Approach:
success/failure: n*p0 = 298.5400, n*(1-p0) = 350.4600 <=10??
Standard Error = SE =sqrt(p0*(1-p0)/n) =
sqrt(0.4600*(1-0.4600)/649) = 0.0196
Standardized z = (phat-p0)/SE = (0.4345-0.4600)/0.0196=-1.3027
P-values: 2-sided = 0.1926844267, 1-sided =0.0963422133

Confidence Interval Approach:

success/failure: n*phat = 282.000, n*(1-phat) = 367.000 <=10??
Standard Error = SEci =sqrt(phat*(1-phat)/n) =
sqrt(0.4345*(1-0.4345)/649 = 0.0195 )
ConfLevel zstar SEci MarginErr phat CIL CIR
0 0.80 1.2816 0.0195 0.0249 0.4345 0.4096 0.4595
1 0.90 1.6449 0.0195 0.0320 0.4345 0.4025 0.4665
```

2	0.95	1.9600	0.0195	0.0381	0.4345	0.3964	0.4727
3	0.99	2.5758	0.0195	0.0501	0.4345	0.3844	0.4846



The hypothesis test tries to infer whether the sample proportion  $\hat{p} = 0.435$  is significantly **less** than  $H_0: p_0 = 0.46$ . The assumptions of the CLT hold as before. Therefore, the sampling distribution of  $\hat{p}$  (**null distribution**) is normal with mean and standard deviation given by  $\mu = p_0$  and  $SE = \sqrt{\frac{p_0(1-p_0)}{n}}$  (see the code).

The **p-value** is the probability to see the observed sampling proportion  $\hat{p}$  or something even more extreme under normal null distribution. The z corresponding to the observed  $\hat{p}$  is:

$$z = \frac{\hat{p} - p_0}{SE} = \frac{0.435 - 0.46}{0.020} = -1.303$$

The alternative  $H_1: p < 0.46$  is one-sided, so only the left tail area gives p-value  $= 0.963422 < 0.10 = \alpha$  as shown in the Figure below. Therefore, there is enough evidence to reject the initial assumption. Thus, there were significantly fewer people who thought that unemployment would increase. If we used more common  $\alpha = 0.05$ , we would have failed to reject the  $H_0$ . Note that my Python code provides both, one-tailed and two-tailed p-values. In this case, if we had done a two-sided test, the p-value would have been  $0.1926844 > 0.10$ , and we would have failed to reject the  $H_0$  as well.

As in the previous example with one-sided test, we have to use  $1 - 2\alpha = 0.80$  confidence level, not  $1 - \alpha = 0.90$ . Therefore, we are only 80% confident (20% significance error) that the true population proportion is between 40.96% and 45.95%, which does not contain the claimed  $H_0: p = 0.46$ . This confirms our conclusion to reject  $H_0$ . On the other hand, the 90% confidence interval between 40.25% and 46.65% contains the claimed  $H_0: p = 0.46$ , so it would have given us the opposite conclusion from the hypothesis test result.

### 5.3.2 Decision Errors and Choosing a Significance (Error) Level

Data varies from sample to sample, so hypothesis tests are not perfect - errors can be made as the table below illustrates.

Table	Fail to Reject H0	Reject H0
H0 is true	Correct	Type 1 error
H0 is false	Type 2 error	Correct

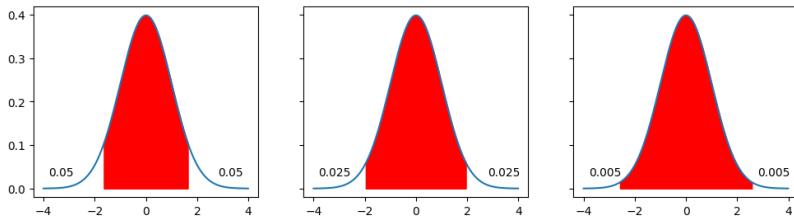
The rows show the actual truth, while the columns correspond to the decision made by the hypothesis test based on the data. Therefore, a Type 1 Error is rejecting the null hypothesis H0 when it is actually true. Its probability is significance level  $\alpha$  in the tails of the confidence interval regions shown again in the Figure below.

```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import norm; import pandas as pd
fig, axs = plt.subplots(1, 3, figsize=(12, 3), sharey=True)
xv = np.arange(-4, 4, 0.01)
ConfLevel = np.array([0.9, 0.95, 0.99])
alpha = 1-ConfLevel
zstar = norm.ppf(1-alpha/2);
df = pd.DataFrame({'ConfLevel':ConfLevel,'alpha':alpha,'zstar':zstar});
axs[0].plot(xv, norm.pdf(xv, 0, 1))
px=np.arange(-zstar[0],zstar[0],0.01)
axs[0].fill_between(px,norm.pdf(px,0,1),color='r')
axs[0].text(-3.8,0.03,"0.05"); axs[0].text(2.8,0.03,"0.05")

axs[1].plot(xv, norm.pdf(xv, 0, 1))
px=np.arange(-zstar[1],zstar[1],0.01)
axs[1].fill_between(px,norm.pdf(px,0,1),color='r')
axs[1].text(-3.8,0.03,"0.025"); axs[1].text(2.8,0.03,"0.025")

axs[2].plot(xv, norm.pdf(xv, 0, 1))
px=np.arange(-zstar[2],zstar[2],0.01)
axs[2].fill_between(px,norm.pdf(px,0,1),color='r')
axs[2].text(-3.8,0.03,"0.005"); axs[2].text(2.8,0.03,"0.005")
df
```

	ConfLevel	alpha	zstar
0	0.90	0.10	1.644854
1	0.95	0.05	1.959964
2	0.99	0.01	2.575829



A Type 2 Error is failing to reject the null hypothesis  $H_0$  when it is actually false (i.e., alternative  $H_1$  is really true). Type 2 error probability cannot be computed directly, as it assumes that the alternative  $H_1$  is true and requires the knowledge of the alternative true proportion. The Power of the test is the probability of the complement of the Type 2 error. It measures the probability of correctly rejecting  $H_0$  when it is really false. In this section we only introduce the idea, power computations are a bit more naturally introduced for means tests in the subsequent chapter.

Note that if you are rejecting the null  $H_0$ , you might have done the Type 1 error and if you are failing to reject  $H_0$ , you might have done the Type 2 error. It does not have to be the case, but it is a possibility. The Type 1 and Type 2 errors are in a trade-off relationship. If Type 1 error is reduced, it is harder to reject  $H_0$  when it is actually true, but it is also harder to reject  $H_0$  when it is actually false increasing Type 2 error, and vice-versa. If a sample size is increased, it would simultaneously reduce both types of errors, which increases overall accuracy.

In an analogy with the court system, a defendant is assumed innocent until proven guilty. Therefore:

$H_0$ : defendant is innocent

$H_1$ : defendant is guilty

Type 1 error is to find the defendant guilty when they are actually innocent.

Type 2 error is to find them innocent when they are actually guilty.

The null hypothesis must only be rejected if there is strong enough evidence against it. The most common choice is to use  $\alpha = 0.05 = 5\%$  significance error level (probability of type 1 error), so that if  $H_0$  is true, it is only rejected about 5% of the time.

In general, if Type 1 Error is more of a concern, stronger evidence for the alternative  $H_1$  is required and a smaller significance level (e.g., 0.01) should be chosen. Consider, for example, a new drug being tested against an established one:

$H_0$ : the new drug is no better than the standard (status quo)

$H_1$ : the new drug is better or worse than the standard

It is only worth to invest millions in a large-scale production if the evidence

in favor of the new drug is very strong.

If a Type 2 Error is more of a concern, then a higher significance level (0.10 say) might be chosen. For example, if there is no drug available at all, then perhaps more of a chance might be given to a new promising drug by increasing the significance level threshold to  $\alpha = 0.10$ . The choice of significance level  $\alpha$  is ultimately a cost-benefit analysis business decision.

### Statistical vs. Practical Significance

As the sample size  $n$  increases, the standard error  $SE = \sqrt{\frac{p(1-p)}{n}}$  decreases, which reduces the margin of error  $e = z^* \cdot SE$  leading to tighter confidence interval. Also,  $z = \frac{\hat{p} - p_0}{SE}$  increases leading to smaller p-values. Therefore, differences between sample  $\hat{p}$  and null  $p_0$  in the hypothesis testing are more likely to be statistically significant. In the early days of Statistics, samples were collected item-by-item and were rather small, so it was not an issue. Nowadays, samples are often automatically collected online, which can produce huge sample sizes. Then, even a minimal practically insignificant difference would be deemed statistically significant. For example, say a change in advertisement strategy may increase TV show viewership by 0.01% which is practically insignificant, but with millions in audience, it may be found to be statistically significant using hypothesis testing. There are specialized methods dealing with very large sample sizes, especially in genetics. In addition, these considerations lead to the next section on how big the sample size has to be to minimally achieve the desired accuracy.

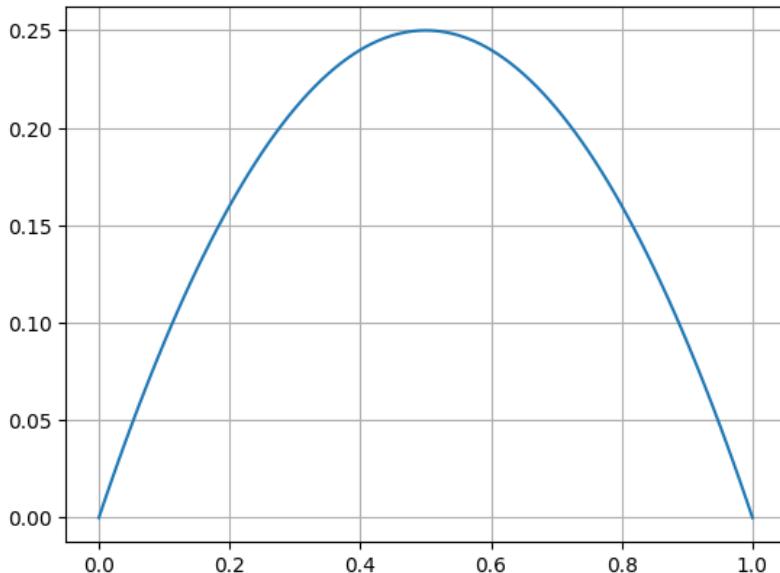
### 5.3.3 Sample Size for Proportion

In this section, we investigate how large should the sample size  $n$  be to ensure a given margin of error  $e$  (accuracy). First, solve the formula for the margin of error for  $n$ :

$$e = z^* \cdot SE = z^* \cdot \sqrt{\frac{p(1-p)}{n}} \Rightarrow e^2 = (z^*)^2 \cdot \frac{p(1-p)}{n} \Rightarrow n = (z^*)^2 \cdot \frac{p(1-p)}{e^2} \quad (5.3)$$

Note that we are trying to determine the sample size  $n$  *before* the experiment, so  $p$  or  $\hat{p}$  are *not* known. In case an estimate of  $p$  is available from a previous study, use it. If not,  $p(1-p) = p - p^2$  is a downward parabola maximized in the middle of  $[0, 1]$  interval at  $p = 0.5$  (see Figure below). Then  $p = 0.5$  is taken as a conservative worst case estimate.

```
[36]: import numpy as np; import matplotlib.pyplot as plt
pv = np.arange(0,1.01,0.01)
fv = pv*(1-pv)
plt.plot(pv,fv); plt.grid()
```



The sample size is directly proportional to  $(z^*)^2$ . Therefore, as the level of confidence increases (say 90% to 95%, to 99%),  $z^*$  increases, and the sample size  $n$  increases (see Figure below). In addition,  $n$  is inversely proportional to  $e^2$ , so as the margin of error  $e$  increases (lower accuracy)  $n$  will decrease.

### Example

How big of a sample is required for a pollster to measure a proportion of adults using the internet with a maximum error of 1% with 95% confidence?

(a) Assume we are given that a previous survey showed 70% of adults use the internet.

(b) Assume nothing is known about the proportion.

```
[ ]: from scipy.stats import norm
e = 0.01; alpha = 0.05;
zstar = norm.ppf(1-alpha/2);
print('e = {:.3f}, alpha = {:.3f}, zstar = {:.3f}'.format(e,alpha,zstar))
# (a)
p = 0.70
na = zstar**2*p*(1-p)/e**2;
print('For p = {:.3f} number needed for the study na = {:.3f}'.format(p,na))
# (b)
p = 0.50
nb = zstar**2*p*(1-p)/e**2;
print('For p = {:.3f} number needed for the study nb = {:.3f}'.format(p,nb))

e = 0.010, alpha = 0.050, zstar = 1.960
```

```
For p = 0.700 number needed for the study na = 8067.064
For p = 0.500 number needed for the study nb = 9603.647
```

Both sample sizes are rather large. The reason is that the given margin of error of 1% is too small, so it is increased to 2% and we obtain a 4-fold reduction in sample sizes.

```
[ ]: from scipy.stats import norm
e = 0.02; alpha = 0.05;
zstar = norm.ppf(1-alpha/2);
print('e = {:.3f}, alpha = {:.3f}, zstar = {:.3f}'.format(e,alpha,zstar))
# (a)
p = 0.70
na = zstar**2*p*(1-p)/e**2;
print('For p = {:.3f} number needed for the study na = {:.3f}'.format(p,na))
# (b)
p = 0.50
nb = zstar**2*p*(1-p)/e**2;
print('For p = {:.3f} number needed for the study nb = {:.3f}'.format(p,nb))
```

```
e = 0.020, alpha = 0.050, zstar = 1.960
For p = 0.700 number needed for the study na = 2016.766
For p = 0.500 number needed for the study nb = 2400.912
```

Let's illustrate the sample size needed for different confidence levels.

```
[ ]: from scipy.stats import norm; import numpy as np; import pandas as pd
e = 0.02;
ConfLevel = np.array([0.90,0.95,0.99]); # confidence levels
alpha = 1-ConfLevel; # corresponding error levels
zstar = norm.ppf(1-alpha/2);
print('e, alpha, zstar = ', e, alpha, zstar)
# (a)
p = 0.70
na = zstar**2*p*(1-p)/e**2;
df = pd.DataFrame({'ConfLevel':ConfLevel,'zstar':zstar,'na':na});
print('\nFor p = 0.7:\n',df)
# (b)
p = 0.50
nb = zstar**2*p*(1-p)/e**2;
df = pd.DataFrame({'ConfLevel':ConfLevel,'zstar':zstar,'nb':nb});
pd.set_option("display.precision", 4); print(df, '\n')
```

```
e, alpha, zstar =  0.02 [0.1  0.05  0.01] [1.64485363  1.95996398  2.5758293 ]
```

```
For p = 0.7:
    ConfLevel      zstar          na
0            0.90  1.6449  1420.4103
1            0.95  1.9600  2016.7659
2            0.99  2.5758  3483.3207
For p = 0.5:
    ConfLevel      zstar          nb
0            0.90  1.6449  1690.9647
1            0.95  1.9600  2400.9118
2            0.99  2.5758  4146.8104
```

As the confidence level increases,  $z^*$  increases and the required sample sizes increase.

### 5.3.4 One-Proportion Tests for Data Files

In this section, a one-proportion test is applied to a data file, which requires first determining each level counts as was done in [Chapter 2](#).

#### Example

In this example, we are going to investigate the file HELPrct.csv used already in [Chapter 2](#). It contains a plethora of information about 453 substance abusers. We are concentrating on the proportion of homeless vs. housed in this sample and want to test if it is 50/50.

**H0** :  $p = 0.5$  (**assume true**).

**H1** :  $p \neq 0.5$ .

We use `value_counts()` method to count subgroups. It is a method for this data type. Also, the `.bar()` method is used to produce a barplot.

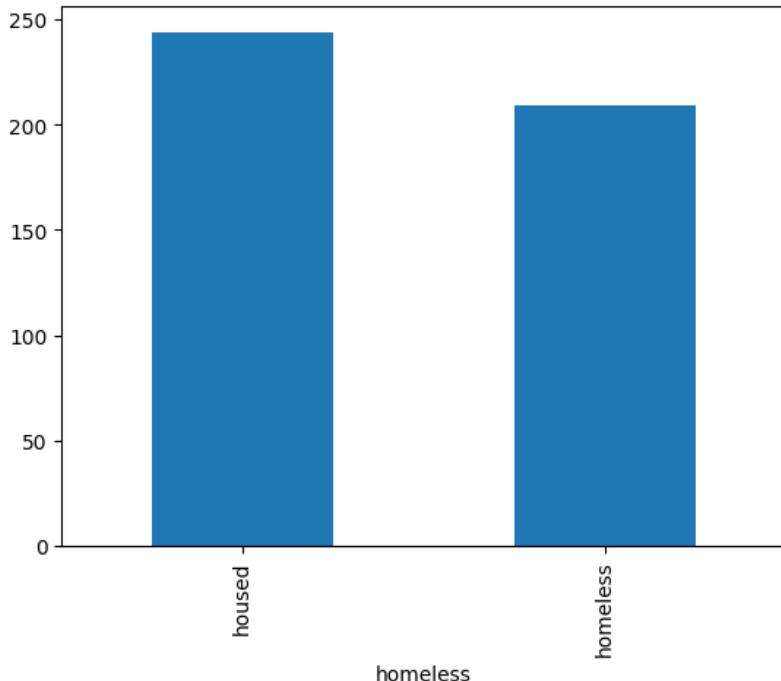
```
[39]: import pandas as pd; import numpy as np; import seaborn as sns
url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrct.csv"
mydata = pd.read_csv(url) # save as mydata file

mytable = mydata['homeless'].value_counts() #

print(mytable)
mytable.plot.bar()
```

```
homeless
housed      244
homeless     209
Name: count, dtype: int64
```

```
[39]: <Axes: xlabel='homeless'>
```



```
[40]: OneProportionTest(x=209,n=209+244,p0=0.50,ConfidenceLevels=[0.9,0.95,0.99])
```

One Proportion Test function  
Number of successes x = 209  
Sample size n = 453  
Null hypothesis claimed proportion p0 = 0.5  
Confidence Levels [0.9, 0.95, 0.99]

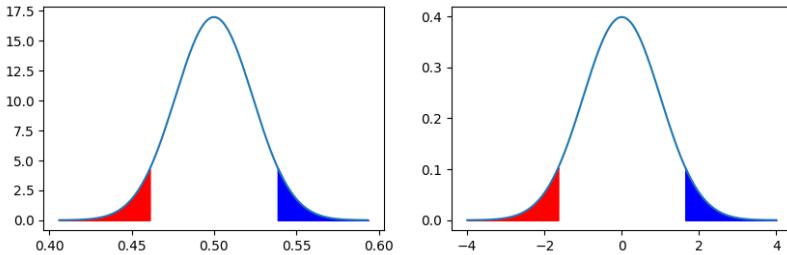
H0: p = p0  
H1: p not p0 or one-sided test

sample proportion phat = 209.00/453 = 0.4614  
Hypothesis Testing Approach:  
success/failure: n\*p0 = 226.5000, n\*(1-p0) = 226.5000 <=10??  
Standard Error = SE =sqrt(p0\*(1-p0)/n) =  
sqrt(0.5000\*(1-0.5000)/453) = 0.0235  
Standardized z = (phat-p0)/SE = (0.4614-0.5000)/0.0235=-1.6444  
P-values: 2-sided = 0.1000846363, 1-sided = 0.0500423181

Confidence Interval Approach:

success/failure: n\*phat = 209.000, n\*(1-phat) = 244.000 <=10??  
Standard Error = SEci =sqrt(phat\*(1-phat)/n) =  
sqrt(0.4614\*(1-0.4614)/453) = 0.0234  
ConfLevel zstar SEci MarginErr phat CIL CIR

0	0.90	1.6449	0.0234	0.0385	0.4614	0.4228	0.4999
1	0.95	1.9600	0.0234	0.0459	0.4614	0.4155	0.5073
2	0.99	2.5758	0.0234	0.0603	0.4614	0.4010	0.5217



The hypothesis test tries to infer whether the sample proportion  $\hat{p} = 0.461$  is significantly different from  $H_0: p_0 = 0.5$ . This null value  $p_0$  is used to check the assumptions of the CLT. The patients can be assumed *independent* from each other (based on the selection, simple random sample, etc.). Also, the code above shows that  $n \cdot p_0 \geq 10$  and  $n \cdot (1-p_0) \geq 10$ , so success/failure conditions are satisfied. Therefore, the sampling distribution of  $\hat{p}$  (**null distribution**) is normal with mean and standard deviation given by  $\mu = p_0$  and  $SE = \sqrt{\frac{p_0(1-p_0)}{n}}$  (see the code above).

The **p-value** is the probability of seeing the observed sampling proportion  $\hat{p}$  or something even more extreme under this bell-shaped normal null distribution. The z corresponding to the observed  $\hat{p}$  is:

$$z = \frac{\hat{p} - p_0}{SE} = \frac{0.461 - 0.5}{0.023} = -1.644$$

The alternative  $H_1 : p \neq p_0$  is two-sided, so both tails add up to p-value =  $0.1000846 > 0.05 = \alpha$  as shown in the Figure above. Therefore, there is *not* enough evidence to reject the initial assumption  $H_0: p = p_0$ . Thus, no reason to reject the 50/50 claim.

Using the confidence interval approach, we are 95% confident (for 5% significance error rate) that the true population proportion is between 41.55% and 50.73%, which contains the claimed  $H_0: p = 0.50$ . This confirms our conclusion not to reject  $H_0$ .

## 5.4 Difference of Two Proportions

Earlier in this chapter, one-sample proportions were considered. A more common situation, however, is to compare two proportions to each other. For example, a smartphone manufacturer may need to compare the proportions of defective chips from two chip manufacturers. A government oversight organization may be interested in comparing the success proportions of heart transplants at two hospitals. The Center for Disease Control may need to compare COVID-19 infection rates in two universities, etc.

### Central Limit Theorem for Two Proportions

Consider two simple random samples. Assume the following conditions:

1. The data are *independent within and between the two groups* (for example, two independent simple random samples).
2. *Success/failure conditions* hold for both groups:

$$n_1 p_1 \geq 10, n_1(1 - p_1) \geq 10, \text{ and } n_2 p_2 \geq 10, n_2(1 - p_2) \geq 10$$

then the difference of sample proportions  $\hat{p}_1 - \hat{p}_2$  follows **normal distribution** with the mean and standard error given by:

$$\mu = p_1 - p_2 \text{ and } SE = \sqrt{\frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2}}$$

respectively, where  $p_1$  and  $p_2$  represent population proportions, and  $n_1$  and  $n_2$  are sample sizes.

For the confidence interval approach, success/failure conditions and standard error computations directly use  $\hat{p}_1$  and  $\hat{p}_2$ :

1. Success/failure conditions:

$$n_1 \hat{p}_1 \geq 10, n_1(1 - \hat{p}_1) \geq 10, \text{ and } n_2 \hat{p}_2 \geq 10, n_2(1 - \hat{p}_2) \geq 10$$

$$2. SE = \sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}}$$

Then the margin of error  $e$  and confidence interval  $CI$  are

$$e = z^* \cdot SE = z^* \sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}} \quad CI = \hat{p}_1 - \hat{p}_2 \pm e$$

For the hypothesis testing approach for two proportions  $H_0: p_1 - p_2 = 0$  (equal population proportions); therefore, sample proportions are pooled as follows: Let  $\hat{p}_1 = \frac{x_1}{n_1}$  and  $\hat{p}_2 = \frac{x_2}{n_2}$  be the two-sample proportions, then the **pooled proportion** is:

$$p_{pool} = \frac{x_1+x_2}{n_1+n_2} = \frac{\frac{x_1}{n_1}n_1 + \frac{x_2}{n_2}n_2}{n_1+n_2} = \frac{\hat{p}_1n_1 + \hat{p}_2n_2}{n_1+n_2}$$

It is used in success/failure conditions and standard error computations:

- 1) Success/failure conditions:

$$n_1 p_{pool} \geq 10, n_1(1 - p_{pool}) \geq 10, \text{ and}$$

$$n_2 p_{pool} \geq 10, n_2(1 - p_{pool}) \geq 10$$

$$2) SE = \sqrt{\frac{p_{pool}(1-p_{pool})}{n_1} + \frac{p_{pool}(1-p_{pool})}{n_2}} = \sqrt{p_{pool}(1 - p_{pool})(\frac{1}{n_1} + \frac{1}{n_2})}$$

Then the test statistic is:

$$z = \frac{(\hat{p}_1 - \hat{p}_2) - (p_1 - p_2)}{SE} = \frac{(\hat{p}_1 - \hat{p}_2) - 0}{\sqrt{p_{pool}(1 - p_{pool})(\frac{1}{n_1} + \frac{1}{n_2})}}$$

Finally, the p-value is computed based on this standardized statistic  $z$  in the same way as for one sample.

### Example

The computer manufacturing company evaluates the quality of computer chips provided by companies A and B. In a random sample of manufacturer A, 23 out of 80 chips were defective, and for manufacturer B, 30 out of 64 were defective. Compare the proportions at the 5% level and make conclusions.

**H0** :  $p_1 - p_2 = 0$  (no difference in proportions - **assume true**).

**H1** :  $p_1 - p_2 \neq 0$  (there is a difference in proportions).

First, we create `TwoProportionTest` function to implement this method. The code is long, but it is all based on the above formulas plus a few technical details to print the steps and plot the figures.

```
[41]: def TwoProportionTest(x1,n1,x2,n2,CIs):
    import numpy as np; from scipy.stats import norm; import pandas as pd
    import matplotlib.pyplot as plt
    print('Two Proportions Test function')
    print('Group 1: number of yes x1 = {:d}, out of n1 = {:.4f}'.format(x1,n1))
    print('Group 2: number of yes x2 = {:d}, out of n2 = {:.4f}'.format(x2,n2))
    print('Confidence Levels',CIs, '\n')

    print('H0: p1-p2 = 0');
    print('H1: p1-p2 not 0 or one-sided test\n')

    ph1 = x1/n1; # sample proportion for 1st group
    ph2 = x2/n2; # sample proportion for 2nd group
    print('sample proportions ph1 = {:d}/{:d} = {:.4f}'.format(x1,n1,ph1))
    print('sample proportions ph2 = {:d}/{:d} = {:.4f}'.format(x2,n2,ph2))
    print('Hypothesis Testing Approach:')

    pL = (x1+x2)/(n1+n2);
    print('p pooled = pL = ({:d}+{:d})/({:d}+{:d})={:.4f}'.format(x1,x2,n1,n2,pL))
    print("success-failure conditions n1*pL,n1*(1-pL),n2*pL, n2*(1-pL) >=10??")
    print(' {:d},{:d},{:d},{:d}'.format(n1*pL,n1*(1-pL),n2*pL,n2*(1-pL)))
    SE = np.sqrt(pL*(1-pL)*(1/n1+1/n2)); SE # standard error
    print('Standard Error = SE =sqrt(pL*(1-pL)*(1/n1+1/n2)) = ')
    print(' = sqrt({:.4f}({1-{:d}}/{:d}+1/{:d})) = {:.4f}'.format(pL,n1,n2,SE))
    diffphats = ph1-ph2;
    print('difference phat1-phat2 = {:.4f}-{:.4f} = {:.4f}'.format(ph1,ph2,diffphats))
    z = (diffphats)/SE; # standardized statistic
    print('Standardized statistic = z = diffphats/SE = {:.4f}/{:.4f} = {:.4f}'
```

```

    .format(diffphats,SE,z))
pval2 = 2*(1-norm.cdf(np.abs(z))); # 2 sided p-value
print('P-values: 2-sided = {:.10f}, 1-sided = {:.10f}\n'.format(pval2,pval2/
→2))

# P-value graphical illustration FYI-----
fig, axs = plt.subplots(1, 2, figsize=(10, 3), sharey=False)
xv = np.arange(-4*SE, 4*SE, 0.001); diff = np.abs(diffphats)
axs[0].plot(xv, norm.pdf(xv, 0, SE))
xL=np.arange(-4*SE,-diff,0.001)
axs[0].fill_between(xL,norm.pdf(xL,0,SE),color='r')
xR=np.arange(diff,4*SE,0.001)
axs[0].fill_between(xR,norm.pdf(xR,0,SE),color='b')
xv = np.arange(-4, 4, 0.001)
axs[1].plot(xv, norm.pdf(xv, 0, 1))
xL=np.arange(-4,-np.abs(z),0.001)
axs[1].fill_between(xL,norm.pdf(xL,0,1),color='r')
xR=np.arange(np.abs(z),4,0.001)
axs[1].fill_between(xR,norm.pdf(xR,0,1),color='b')
#-----


print('Confidence Interval Approach:')
ConfLevel = np.array(CIs) # confidence level
alpha = 1-ConfLevel # error level corresponding to confidence level
print("success-failure conditions n1*ph1,n1*(1-ph1),n2*ph2, n2*(1-ph2)_
→>=10??")
print(' {:.4f},{:.4f},{:.4f},{:.4f}'.
→format(n1*ph1,n1*(1-ph1),n2*ph2,n2*(1-ph2)))
SEci = np.sqrt(ph1*(1-ph1)/n1+ph2*(1-ph2)/n2);
print('Standard Error = SEci =sqrt(ph1*(1-ph1)/n1+ph1*(1-ph2)/n2)) = ')
print(' = sqrt({:.4f}(1-{:.4f})/{:d}+{:.4f}(1-{:.4f})/{:d}) = {:.4f}'.
    .format(ph1,ph1,n1,ph2,ph2,n2,SEci))
zstar = norm.ppf(1-alpha/2); # z critical
MarginErr = zstar*SEci; # margin of error
CIL = diffphats - MarginErr; CIR = diffphats + MarginErr;
df = pd.DataFrame({'ConfLevel':ConfLevel,'zstar':zstar,'SEci':SEci, \
'MarginErr':MarginErr,'diffphats':diffphats,'CIL':\
→CIL,'CIR':CIR});
pd.set_option("display.precision", 4); print(df, '\n')

```

Let's call on the function above to solve the given problem.

[42]: TwoProportionTest(x1=23,n1=80,x2=30,n2=64,CIs=[0.90,0.95,0.99])

```

Two Proportions Test function
Group 1: number of yes x1 = 23, out of n1 = 80
Group 2: number of yes x2 = 30, out of n2 = 64
Confidence Levels [0.9, 0.95, 0.99]

```

```

H0: p1-p2 = 0
H1: p1-p2 not 0 or one-sided test

```

```

sample proportions ph1 = 23/80 = 0.2875
sample proportions ph2 = 30/64 = 0.4688
Hypothesis Testing Approach:
p pooled = pL = (23+30)/(80+64)=0.3681

```

```

success-failure conditions n1*pL,n1*(1-pL),n2*pL, n2*(1-pL) >=10??
29.4444,50.5556,23.5556,40.4444
Standard Error = SE =sqrt(pL*(1-pL)*(1/n1+1/n2)) =
= sqrt(0.3681(1-0.3681)(1/80+1/64)) = 0.0809
difference phat1-phat2 = 0.2875-0.4688 = -0.1813
Standardized statistic = z = diffphats/SE = -0.1813/0.0809 = -2.2410
P-values: 2-sided = 0.0250281627, 1-sided =0.0125140813

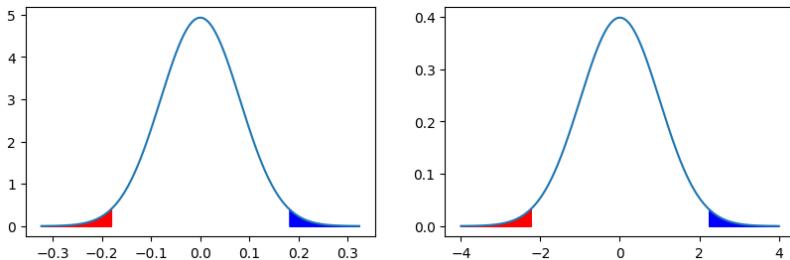
```

Confidence Interval Approach:

```

success-failure conditions n1*ph1,n1*(1-ph1),n2*ph2, n2*(1-ph2) >=10??
23.0000,57.0000,30.0000,34.0000
Standard Error = SEci =sqrt(ph1*(1-ph1)/n1+ph2*(1-ph2)/n2)) =
= sqrt(0.2875(1-0.2875)/80+0.4688(1-0.4688)/64) = 0.0803
ConfLevel      zstar      SEci   MarginErr   diffphats      CIL      CIR
0            0.90    1.6449  0.0803     0.1321    -0.1813 -0.3134 -0.0491
1            0.95    1.9600  0.0803     0.1574    -0.1813 -0.3387 -0.0238
2            0.99    2.5758  0.0803     0.2069    -0.1813 -0.3881  0.0256

```



The hypothesis test aims to infer whether the sample proportions difference  $\hat{p}_1 - \hat{p}_2 = 0.288 - 0.469 = -0.181$  is significantly different from the claimed null hypothesis difference of 0. The samples of chips are assumed to be simple random samples, so the observations are *independent* within groups and the samples are independent from each other. Also, the code above shows that:  $n_1 p_{pool} = 80 \cdot 0.368 = 29.444 \geq 10$ ,  $n_1(1 - p_{pool}) = 80 \cdot (1 - 0.368) = 50.556 \geq 10$ ,  $n_2 p_{pool} = 64 \cdot 0.368 = 23.556 \geq 10$ ,  $n_2(1 - p_{pool}) = 64 \cdot (1 - 0.368) = 40.444 \geq 10$  so the success/failure conditions are satisfied.

Therefore, based on CLT, the sampling distribution of  $\hat{p}_1 - \hat{p}_2$  (**null distribution**) is normal with mean and standard deviation given by:

$$\mu = 0 \text{ and } SE = \sqrt{p_{pool}(1 - p_{pool})(\frac{1}{n_1} + \frac{1}{n_2})} = \sqrt{0.368(1 - 0.368)(\frac{1}{80} + \frac{1}{64})} = 0.081 \text{ (see left plot of the Figure above).}$$

The **p-value** is the probability to see the observed sampling proportion difference  $\hat{p}_1 - \hat{p}_2 = -0.181$  or something even more extreme under the normal null distribution.

The  $z$  corresponding to the  $\hat{p}_1 - \hat{p}_2 = -0.181$  is

$$z = \frac{\hat{p}_1 - \hat{p}_2 - 0}{SE} = \frac{0.288 - 0.469}{0.081} = \frac{-0.181}{0.081} = -2.241$$

The alternative  $H1 : p_1 - p_2 \neq 0$  is two-sided, so both tails add up to p-value  $= 0.0250282 < 0.05 = \alpha$ , as shown in the Figure above (for the original proportions in the left Figure, and for the standardized z-scores in the right Figure). Thus, there is enough evidence to reject the initial assumption  $H0$ , i.e, rejecting the claim of equal proportion of defective chips produced by these two companies.

For the confidence interval computations, the success/failure conditions and standard error are computed using  $\hat{p}_1$  and  $\hat{p}_2$ :

$$\begin{aligned} n_1\hat{p}_1 &= 80 \cdot 0.288 = 23 \geq 10, \quad n_1(1 - \hat{p}_1) = 80 \cdot (1 - 0.288) = 57 \geq 10, \\ n_2\hat{p}_2 &= 64 \cdot 0.469 = 30 \geq 10, \quad n_2(1 - \hat{p}_2) = 64 \cdot (1 - 0.469) = 34 \geq 10 \implies \text{success/failure conditions are satisfied.} \end{aligned}$$

Standard error:

$$\begin{aligned} SE &= \sqrt{\frac{\hat{p}_1(1 - \hat{p}_1)}{n_1} + \frac{\hat{p}_2(1 - \hat{p}_2)}{n_2}} \\ &= \sqrt{\frac{0.288(1 - 0.288)}{80} + \frac{0.469(1 - 0.469)}{64}} = 0.0803 \end{aligned}$$

Therefore, say 95% confidence interval for the difference of population proportions  $p_1 - p_2$  is:

$$\begin{aligned} \hat{p}_1 - \hat{p}_2 \pm z^* \cdot SE &= 0.288 - 0.469 \pm 1.96 \cdot 0.0803 = \\ -0.181 \pm 0.157 &= (-0.3387, -0.0238) \end{aligned}$$

Therefore, we are 95% confident (5% significance error) that the true population proportion difference is between  $-33.87\%$  and  $-2.38\%$ , which does *not* contain  $H0 : p_1 - p_2 = 0$  once again confirming our conclusion to reject  $H0$ .

### Example

A total of 87 out of 100 students preparing with firm B's preparation guide passed the Medical admission test MCAT, while 91 out of 120 passed it with firm P's guide. Is there a significant difference at the 1% level? How about a 5% level?

**H0** :  $p_1 - p_2 = 0$  (no difference in proportions - **assume true**).

**H1** :  $p_1 - p_2 \neq 0$  (there is a difference in proportions).

[43]: `TwoProportionTest(x1=87, n1=100, x2=91, n2=120, CIs=[0.90, 0.95, 0.99])`

```
Two Proportions Test function
Group 1: number of yes x1 = 87, out of n1 = 100
Group 2: number of yes x2 = 91, out of n2 = 120
Confidence Levels [0.9, 0.95, 0.99]
```

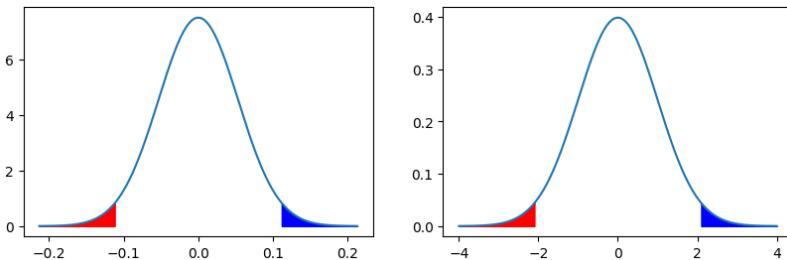
```

H0: p1-p2 = 0
H1: p1-p2 not 0 or one-sided test

sample proportions ph1 = 87/100 = 0.8700
sample proportions ph2 = 91/120 = 0.7583
Hypothesis Testing Approach:
p pooled = pL = (87+91)/(100+120)=0.8091
success-failure conditions n1*pL,n1*(1-pL),n2*pL, n2*(1-pL) >=10??
80.9091,19.0909,97.0909,22.9091
Standard Error = SE =sqrt(pL*(1-pL)*(1/n1+1/n2)) =
= sqrt(0.8091(1-0.8091)(1/100+1/120)) = 0.0532
difference phat1-phat2 = 0.8700-0.7583 = 0.1117
Standardized statistic = z = diffphats/SE = 0.1117/0.0532 = 2.0984
P-values: 2-sided = 0.0358687200, 1-sided =0.0179343600

Confidence Interval Approach:
success-failure conditions n1*ph1,n1*(1-ph1),n2*ph2, n2*(1-ph2) >=10??
87.0000,13.0000,91.0000,29.0000
Standard Error = SEci =sqrt(ph1*(1-ph1)/n1+ph2*(1-ph2)/n2)) =
= sqrt(0.8700(1-0.8700)/100+0.7583(1-0.7583)/120) = 0.0516
  ConfLevel      zstar      SEci   MarginErr   diffphats       CIL       CIR
0          0.90    1.6449    0.0516      0.0848      0.1117    0.0269    0.1965
1          0.95    1.9600    0.0516      0.1011      0.1117    0.0106    0.2127
2          0.99    2.5758    0.0516      0.1328      0.1117   -0.0211    0.2445

```



The hypothesis test tries to infer whether the sample proportions difference  $\hat{p}_1 - \hat{p}_2 = 0.87 - 0.758 = 0.112$  is significantly different from the claimed null hypothesis difference of 0.

The samples of students are assumed to be simple random samples, so the observations are *independent* within groups, and the samples of themselves are independent from each other. Also, the code above shows that:

$n_1 p_{pool}, n_1(1-p_{pool}), n_2 p_{pool}, n_2(1-p_{pool})$  are all at least 10, so success/failure conditions are satisfied.

Therefore, according to the CLT, the sampling distribution of  $\hat{p}_1 - \hat{p}_2$  (**null distribution**) is normal with mean:  $\mu = p_1 - p_2 = 0$  and standard deviation given  $SE = \sqrt{p_{pool}(1-p_{pool})(\frac{1}{n_1} + \frac{1}{n_2})}$  (see the code as well as the left plot

of the Figure above).

The **p-value** is the probability of seeing the observed sampling proportion difference or something even more extreme under our bell-shaped normal null distribution. The  $z$  corresponding to the  $\hat{p}_1 - \hat{p}_2$  is:

$$z = \frac{\hat{p}_1 - \hat{p}_2 - 0}{SE} = \frac{0.87 - 0.758}{0.053} = \frac{0.112}{0.053} = 2.098$$

The alternative  $H_1 : p_1 - p_2 \neq 0$  is two-sided, so both tails add up to p-value  $p\text{-value} = 0.0358687 > 0.01 = \alpha$ , as shown in the left and right Figures above. Thus, there is *not* enough evidence to reject the initial assumption  $H_0 : p_1 - p_2 = 0$  at the more strict 1% level. Therefore, we cannot reject the claim of an equal proportion of MCAT passing with two prep guides. However, if a 5% significance level was applied, then the same p-value  $< 0.05$  and we would have rejected the equality of population proportions.

In the confidence interval computations, the success/failure conditions and standard error are computed using  $\hat{p}_1$  and  $\hat{p}_2$ . The code above shows that  $n_1\hat{p}_1, n_1(1 - \hat{p}_1), n_2\hat{p}_2, n_2(1 - \hat{p}_2)$  are all 10 or more, so the success-failure conditions are satisfied.

Therefore the confidence interval for the difference of population proportions  $p_1 - p_2$  is:

$$\hat{p}_1 - \hat{p}_2 \pm z^* \cdot SE = 0.87 - 0.758 \pm 2.576 \cdot 0.052 = 0.112 \pm 0.132 = (-0.021, 0.244)$$

Therefore, we are 99% confident (1% significance error) that the true population proportion is between -2.11% and 24.45%, which contains the claimed  $H_0 : p_1 - p_2 = 0$  confirming *not* rejecting  $H_0$ . Once again the 95% CI (1.06%, 21.27%) does *not* contain 0, so we would have rejected  $H_0$ .

### Example

In this example, we look again at the substance abusers data set. We want to investigate if the proportions of people who got treatment are different in homeless vs. housed patients. Is there a significant difference at the 5% level? How about a 10% level?

**H0** :  $p_1 - p_2 = 0$  (no difference in proportions - **assume true**).

**H1** :  $p_1 - p_2 \neq 0$  (there is a difference in proportions).

```
[44]: import pandas as pd
url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrc.csv"
mydata = pd.read_csv(url)

# Note pd.crosstab() is used to cross-tabulate the counts
Om = pd.crosstab(mydata.homeless, mydata.satreat, margins=True);
print(Om)
```

satreat	no	yes	All
homeless			

homeless	141	68	209
housed	183	61	244
All	324	129	453

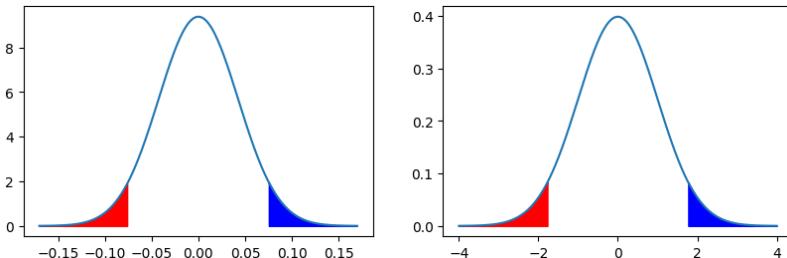
[45]: `TwoProportionTest(x1=68,n1=209,x2=61,n2=244,CIs=[0.90,0.95,0.99])`

```
Two Proportions Test function
Group 1: number of yes x1 = 68, out of n1 = 209
Group 2: number of yes x2 = 61, out of n2 = 244
Confidence Levels [0.9, 0.95, 0.99]

H0: p1-p2 = 0
H1: p1-p2 not 0 or one-sided test

sample proportions ph1 = 68/209 = 0.3254
sample proportions ph2 = 61/244 = 0.2500
Hypothesis Testing Approach:
p pooled = pL = (68+61)/(209+244)=0.2848
success-failure conditions n1*pL,n1*(1-pL),n2*pL, n2*(1-pL) >=10??
59.5166,149.4834,69.4834,174.5166
Standard Error = SE =sqrt(pL*(1-pL)*(1/n1+1/n2)) =
= sqrt(0.2848*(1-0.2848)*(1/209+1/244)) = 0.0425
difference phat1-phat2 = 0.3254-0.2500 = 0.0754
Standardized statistic = z = diffphats/SE = 0.0754/0.0425 = 1.7717
P-values: 2-sided = 0.0764485137, 1-sided =0.0382242569

Confidence Interval Approach:
success-failure conditions n1*ph1,n1*(1-ph1),n2*ph2, n2*(1-ph2) >=10??
68.0000,141.0000,61.0000,183.0000
Standard Error = SEci =sqrt(ph1*(1-ph1)/n1+ph1*(1-ph2)/n2)) =
= sqrt(0.3254*(1-0.3254)/209+0.2500*(1-0.2500)/244) = 0.0426
ConfLevel zstar SEci MarginErr diffphats CIL CIR
0 0.90 1.6449 0.0426 0.0701 0.0754 0.0052 0.1455
1 0.95 1.9600 0.0426 0.0836 0.0754 -0.0082 0.1589
2 0.99 2.5758 0.0426 0.1098 0.0754 -0.0345 0.1852
```



The hypothesis test tries to infer whether the sample proportions difference  $\hat{p}_1 - \hat{p}_2 = 0.325 - 0.25 = 0.075$  is significantly different from the claimed null hypothesis difference of 0.

The substance abuse patients in each group are assumed to be simple random samples, so the observations are *independent* within groups and the samples themselves are independent of each other. Also, the code above shows that:  $n_1 p_{pool}, n_1(1 - p_{pool}), n_2 p_{pool}, n_2(1 - p_{pool})$  are all at least 10, so success/failure conditions are satisfied. Therefore, according to the CLT the sampling distribution of  $\hat{p}_1 - \hat{p}_2$  (**null distribution**) is normal with mean  $\mu = p_1 - p_2 = 0$  and standard deviation given  $SE = \sqrt{p_{pool}(1 - p_{pool})(\frac{1}{n_1} + \frac{1}{n_2})}$  (see the calculations and the left plot of the Figure above).

The **p-value** is the probability of seeing the observed sampling proportion difference or something even more extreme under our bell-shaped normal null distribution. The  $z$  corresponding to the  $\hat{p}_1 - \hat{p}_2$  is:

$$z = \frac{\hat{p}_1 - \hat{p}_2 - 0}{SE} = \frac{0.325 - 0.25}{0.043} = \frac{0.075}{0.053} = 1.772$$

Given the two-sided nature of the alternative  $H1 : p_1 - p_2 \neq 0$ , we have to account for both left and right tails. The areas in these tails add up to the final p-value = 0.0764485 > 0.05 =  $\alpha$ , as shown in the left and right Figures above. Thus, we conclude that there is *not* enough evidence to reject the initial assumption  $H0 : p_1 - p_2 = 0$  at 5% level, i.e., *not* rejecting the claim of an equal proportion of patients getting treatment in homeless vs. housed population. Note, that if we apply a 10% significance level, then the same p-value < 0.10 and our conclusion is just the opposite - there is enough evidence to reject the equality of two population proportions.

For the confidence interval computations, the success/failure conditions and standard error are computed using  $\hat{p}_1$  and  $\hat{p}_2$ . The code above shows that  $n_1 \hat{p}_1, n_1(1 - \hat{p}_1), n_2 \hat{p}_2, n_2(1 - \hat{p}_2)$  are all 10 or more, so the success/failure conditions are satisfied. Therefore, the confidence interval for the difference of population proportions  $p_1 - p_2$  is:

$$\hat{p}_1 - \hat{p}_2 \pm z^* \cdot SE = 0.325 - 0.25 \pm z^* \cdot 0.0426 = 0.075 \pm z^* \cdot 0.0836$$

Thus, we are 95% confident (for 5% significance error rate) that the true population proportion is between -0.82% and 15.89%, which does contain the claimed  $H0 : p_1 - p_2 = 0$ . This confirms our conclusion *not* to reject H0 at 95% level. However, the 90% CI given by (0.52%, 14.55%) does *not* contain 0 and leads to the rejection of H0.

# 6

---

## Goodness of Fit and Contingency Tables

---

### 6.1 Goodness of Fit

The goodness of fit refers to a method for assessing a distribution of binned data. It is best illustrated with an example.

#### Example

A shop owner wants to compare the number of t-shirts of each size that were sold to the ordered proportions. Assume the sold (observed) numbers of t-shirts and expected proportions for each type are as given in the first two columns in the following table:

Table	Observed	Expected Proportions	Expected Values
Small	25	0.1	22.5
Medium	41	0.2	45
Large	91	0.4	90
X-Large	68	0.3	67.5

The total number of the shirts sold is  $Total = 25 + 41 + 91 + 68 = 225$ . If the sales follow the expected proportions, we would expect  $225 \cdot 0.1 = 22.5$  small shirts to be sold,  $225 \cdot 0.2 = 45$  medium, etc. (see the column of the Expected Values in the table above).

The numbers of shirts of each type (observed values) are quite close to the expected values in this case (not the same due to sampling variation). Are the differences large enough to indicate that the observed (sample) sales do not follow the expected (population) sales.

Let's formulate it as a hypothesis test:

$H_0$ : The observed sales follow the expected proportions or  $O \approx E$  (no bias, natural sampling fluctuation).

$H_1$ : The observed sales do NOT follow the expected proportions or  $O \neq E$ .

In the hypothesis tests for proportions, a test statistic was:

$$\frac{\text{point estimate} - \text{null estimate}}{\text{SE of point estimate}}$$

As always in hypothesis testing, the null hypothesis  $H_0$  is initially assumed true, so the expected counts -  $E$ -s are taken as null estimates. In Mathematical Statistics, it is shown that standard error  $SE = \sqrt{E}$ . Therefore, the standardized residuals for each category are

$$\frac{O_i - E_i}{\sqrt{E_i}}$$

where  $i$ -s refer to category (Small, Medium, etc.). As always, residuals have varying signs, so adding them would lead to cancellations; instead their squares are considered.

Let  $O_1, O_2, \dots, O_k$  be the observed counts in  $k$  groups, and  $E_1, E_2, \dots, E_k$  be the corresponding expected counts under a null hypothesis. Then provided that all  $E_i \geq 5$ , the **Chi-Squared test statistic**

$$\chi^2 = \frac{(O_1 - E_1)^2}{E_1} + \frac{(O_2 - E_2)^2}{E_2} + \dots + \frac{(O_k - E_k)^2}{E_k}$$

follows a chi-squared distribution with  $k - 1$  degrees of freedom. A larger difference between observed and expected values (against  $H_0$ ) leads to larger  $\chi^2$ , so the p-value is the upper tail area of this chi-squared distribution.

Unlike the normal distribution, the  $\chi^2$  is a family of distributions with different shapes based on the degree of freedom. It is skewed right, not symmetric as shown by the code and the resulting Figure below.

```
import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import chi2

# Parameters for the chi-squared distributions
df_values = [2, 5, 10] # Degrees of freedom
x = np.linspace(0, 20, 500) # x-values for the plots

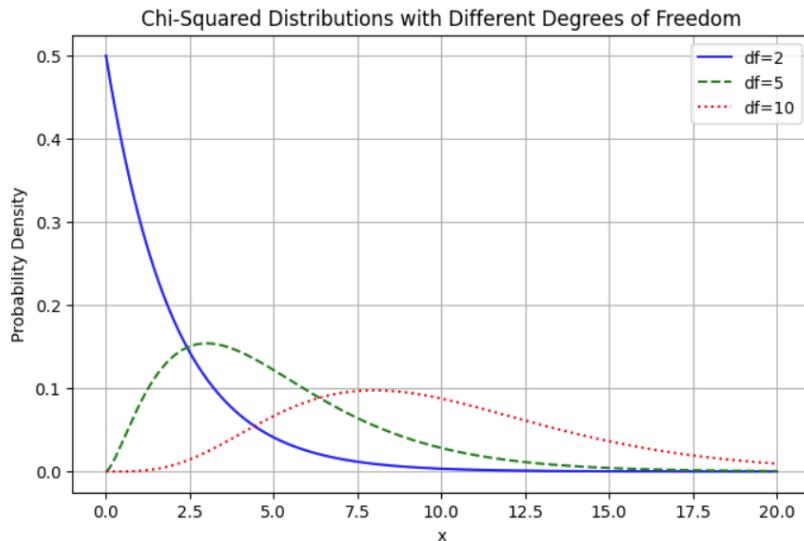
# Line styles and colors for the distributions
linestyles = ['-', '--', ':']; colors = ['blue', 'green', 'red']

# Plotting the chi-squared distributions
plt.figure(figsize=(8, 5))

for df, ls, color in zip(df_values, linestyles, colors):
    y = chi2.pdf(x, df) # Compute the PDF of the chi-squared distribution
    plt.plot(x, y, linestyle=ls, color=color, label=f"df={df}")
```

```
# Add title, labels, legend, and grid
plt.title("Chi-Squared Distributions with Different Degrees of Freedom")
plt.xlabel("x"); plt.ylabel("Probability Density")
plt.legend(); plt.grid(True); plt.show()
```

[ ]: <matplotlib.legend.Legend at 0x789fa9c3f1c0>



For our data:

$$\chi^2 = \frac{(O_1 - E_1)^2}{E_1} + \frac{(O_2 - E_2)^2}{E_2} + \dots + \frac{(O_k - E_k)^2}{E_k} =$$

$$\frac{(25 - 22.5)^2}{22.5} + \frac{(41 - 45)^2}{45} + \dots + \frac{(68 - 67.5)^2}{67.5} =$$

$$0.278 + 0.356 + \dots + 0.004 = 0.648$$

The function defined below computes the steps of the goodness of fit method and plots a chi-squared distribution with  $df = k - 1 = 4 - 1 = 3$  degrees of freedom with the shaded area corresponding to the p-value as shown in the Figure below.

```
[ ]: def GoodnessFit(Observed,BinNames,Pexpect):
    import numpy as np; import matplotlib.pyplot as plt; import pandas as pd
```

```

from scipy.stats import chisquare; from scipy.stats import chi2

O = np.array(Observed); P_exp = np.array(Pexpect); n = len(O)
total = np.sum(O);
print("Total Observed = {:d} + {:d} + ... {:d} = {:d}"
      .format(O[0], O[1], O[n-1], total))
E = total*P_exp;
print("Expected frequencies = E = total*P_exp ={:d}*{:.3f},{:.3f},...,{:.3f}]"
      .format(total,P_exp[0],P_exp[1],P_exp[n-1]))
print('      [{:.3f},{:.3f},...,{:.3f}]'.format(E[0],E[1],E[n-1]))
print('Residuals = (Observed - Expected)/sqrt(Expected) = ')
print('=({:d}-{:2f})/sqrt({:2f}), ... , ({:d}-{:2f})/sqrt({:2f})\n'
      .format(O[0],E[0],E[0],O[n-1],E[n-1],E[n-1]))
R = (O-E)/np.sqrt(E); R2 = R**2;

Frame = pd.DataFrame({'Observed': O, 'Expected P': P_exp, 'Expected': E,
                      'Residuals': R, 'Residuals**2': R2})
Frame.index = BinNames;
pd.set_option("display.precision", 4); print(Frame, '\n')

df = len(O)-1; print("df = len(O)-1 ={:d}={:d}" .format(len(O),df))
X2 = np.sum(R2);

print('Chi-Squared = sum( (O-E)**2/E )= sum( Residuals**2 ) = ')
print('=({:d}-{:2f})**2/{:2f} + ... + ({:d}-{:2f})**2/{:2f} = {:.3f}' 
      .format(O[0],E[0],E[0],O[n-1],E[n-1],X2))

pval = 1 - chi2.cdf(X2,df)
print('p-value = 1 - chi2.cdf(X2,df) = ', pval, '\n')

(X2, pval) = chisquare(f_obs=O,f_exp=E)
print("From chisquare function: X2 = ", round(X2,3), ' pval = ', pval)

x = np.arange(0, X2+10, 0.001) #x-axis 0 to 20 with .001 steps
P1 = plt.plot(x, chi2.pdf(x, df=df)) #plot Chi-square
xR=np.arange(X2,X2+10,0.001)
plt.fill_between(xR,chi2.pdf(xR, df=df),color='r') # filling for p-value
plt.show(P1)

Frame[['Observed','Expected']].plot(kind='bar') # observed barplot

```

```
[ ]: GoodnessFit(Observed=[25, 41, 91, 68],
                  BinNames=["Small","Medium","Large","X-Large"],
                  Pexpect=[0.1, 0.2, 0.4, 0.3])
```

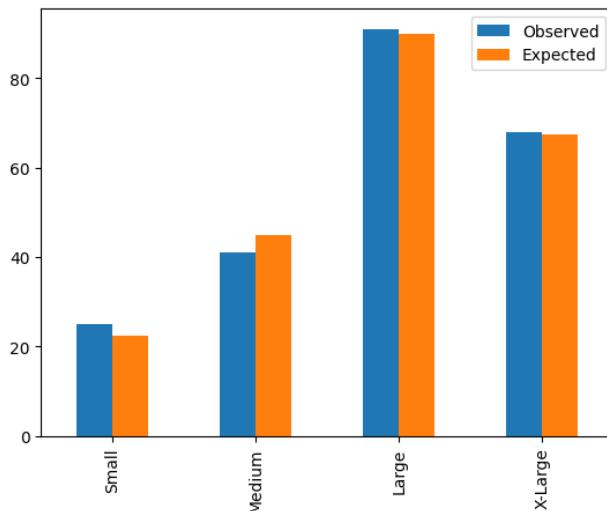
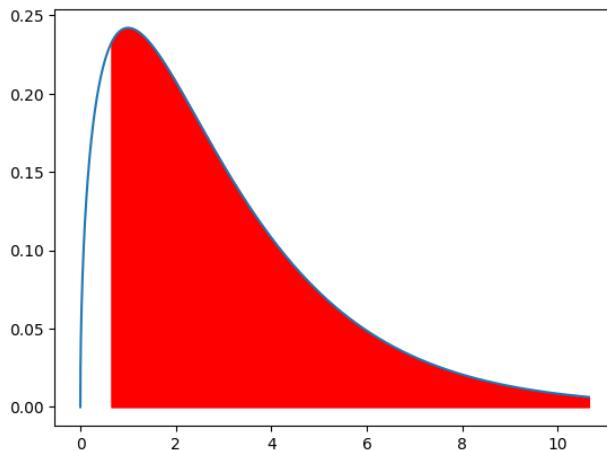
```
Total Observed = 25 + 41 + ... 68 = 225
Expected frequencies = E = total*P_exp =225*[0.100,0.200,...,0.300]=
[22.500,45.000,...,67.500]
Residuals = (Observed - Expected)/sqrt(Expected) =
=(25-22.50)/sqrt(22.50), ... , (68-67.50)/sqrt(67.50)
```

	Observed	Expected	P	Expected	Residuals	Residuals**2
Small	25	0.1		22.5	0.5270	0.2778
Medium	41	0.2		45.0	-0.5963	0.3556
Large	91	0.4		90.0	0.1054	0.0111

```
X-Large      68      0.3      67.5      0.0609      0.0037
```

```
df = len(O)-1 = 4-1 = 3
Chi-Squared = sum( (O-E)**2/E ) = sum( Residuals**2) =
=(25-22.50)**2/22.50 + ... + (68-67.50)**2/67.50 = 0.648
p-value = 1 - chi2.cdf(X2,df) = 0.8853267818237286
```

```
From chis-squared function: X2 = 0.648    pval = 0.8853267818237286
```



For our example, the observed  $O$ -s are very close to the expected  $E$ -s leading to very small  $\chi^2 = 0.648$ , which results in a very large p-value = 0.8853268 > 0.05 as shown in the chi-squared distribution Figure above. Therefore, there is not enough evidence to reject  $H_0$ , and the observed shirt sales follow the expected proportions. We can also see it in the side-by-side bar plot of the observed and expected counts above.

### Example

The data for a particular hospital's emergency room admissions over the previous month are given below. To properly allocate resources, investigate if the admissions are *uniformly* distributed by the day of the week.

$H_0$ : The observed values of the number of hospital admissions follow the expected proportions  $O \approx E$  (equally likely 7 days, so  $p_i = 1/7$ ).

$H_1$ : The observed values do NOT follow the expected proportions  $O \neq E$ .

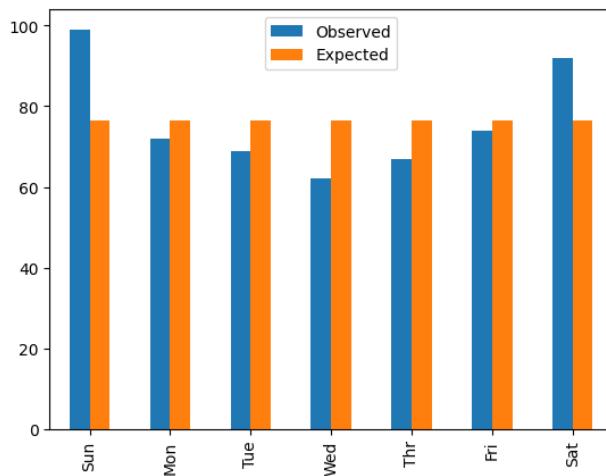
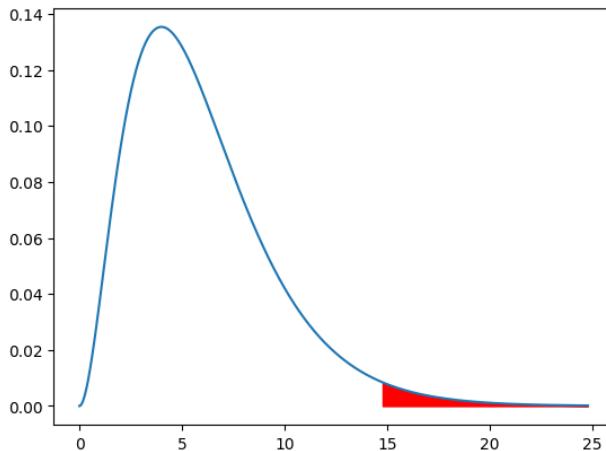
```
[ ]: GoodnessFit(Observed=[99,72,69,62,67,74,92],
                  BinNames=['Sun','Mon','Tue','Wed','Thr','Fri','Sat'],
                  Pexpect=[1/7,1/7,1/7,1/7,1/7,1/7,1/7])
```

```
Total Observed = 99 + 72 + ... 92 = 535
Expected frequencies = E = total*P_exp =535*[0.143,0.143,...,0.143] =
[76.429,76.429,...,76.429]
Residuals = (Observed - Expected)/sqrt(Expected) =
=(99-76.43)/sqrt(76.43), ... , (92-76.43)/sqrt(76.43)
```

	Observed	Expected	P	Expected	Residuals	Residuals**2
Sun	99	0.1429		76.4286	2.5819	6.6660
Mon	72	0.1429		76.4286	-0.5066	0.2566
Tue	69	0.1429		76.4286	-0.8497	0.7220
Wed	62	0.1429		76.4286	-1.6504	2.7239
Thr	67	0.1429		76.4286	-1.0785	1.1632
Fri	74	0.1429		76.4286	-0.2778	0.0772
Sat	92	0.1429		76.4286	1.7812	3.1725

```
df = len(O)-1 =7-1 = 6
Chi-Squared = sum( (O-E)**2/E )= sum( Residuals**2) =
=(99-76.43)**2/76.43 + ... + (92-76.43)**2/76.43 = 14.781
p-value = 1 - chisq.cdf(X2,df) = 0.022027570437209598
```

```
From chi-squared function: X2 = 14.781    pval = 0.022027570437209612
```



Adding the observed values, we obtain  $Total = 535$ . If the observed values follow the expected proportions,  $535 \cdot \frac{1}{7} \approx 76.429$  admissions are expected on Sun, the same on other days. All the Expected Values are given in the table above and *none is below 5*. The resulting test statistic  $\chi^2$  is:

$$\chi^2 = \frac{(O_1 - E_1)^2}{E_1} + \frac{(O_2 - E_2)^2}{E_2} + \dots + \frac{(O_k - E_k)^2}{E_k} =$$

$$\frac{(99 - 76.429)^2}{76.429} + \frac{(72 - 76.429)^2}{76.429} + \dots \frac{(92 - 76.429)^2}{76.429} =$$

$$6.666 + 0.257 + \dots + 3.172 = 14.781$$

The  $\chi^2$  degree of freedom is  $df = k - 1 = 7 - 1 = 6$ .

The  $\chi^2 = 14.781$  with the degree of freedom  $df = k - 1 = 7 - 1 = 6$  is sufficiently large to produce small p-value = 0.0220276 < 0.05 as shown in the chi-squared distribution Figure above. Therefore, there is enough evidence to reject H0, i.e., the observed calls on different days of the week are not uniformly distributed. We can see from the table that there are more emergency visits on weekends when regular doctors are not available. We can also see it in the side-by-side bar plot of the observed and expected counts above.

### Example

In the HELPrct file on substance abusers' health data, investigate if the substance variable is equally distributed between alcohol, cocaine, and heroin.

Note that we employ value\_counts() to get the counts for each substance level.

H0: The observed values of the substance abusers in each group follow the expected proportions  $O \approx E$  (equally likely).

H1: The observed values do NOT follow the expected proportions or  $O \neq E$ .

```
[ ]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrct.csv"
mydata = pd.read_csv(url) # save as mydata file
counts = mydata['substance'].value_counts(); counts
```

```
[ ]: substance
alcohol      177
cocaine     152
heroin      124
Name: count, dtype: int64
```

```
[ ]: GoodnessFit(Observed=[177,152,124],
                 BinNames=['alcohol','cocaine','heroin'],
                 Pexpect=[1/3,1/3,1/3])
```

```
Total Observed = 177 + 152 + ... 124 = 453
Expected frequencies = E = total*P_exp =453*[0.333,0.333,...,0.333]=
[151.000,151.000,...,151.000]
Residuals = (Observed - Expected)/sqrt(Expected) =
=(177-151.00)/sqrt(151.00), ... , (124-151.00)/sqrt(151.00)
```

Observed	Expected	P	Expected	Residuals	Residuals**2
----------	----------	---	----------	-----------	--------------

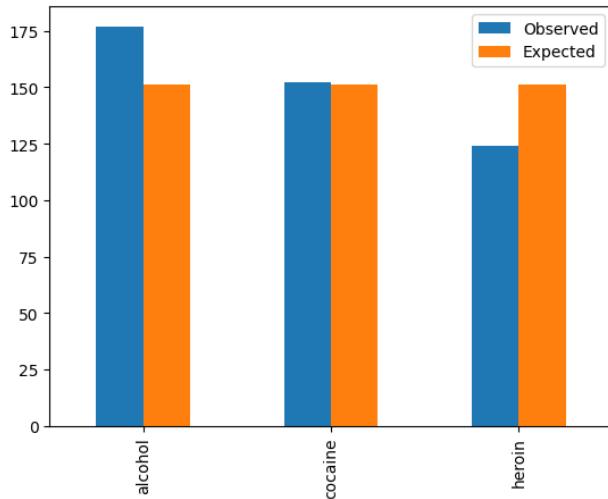
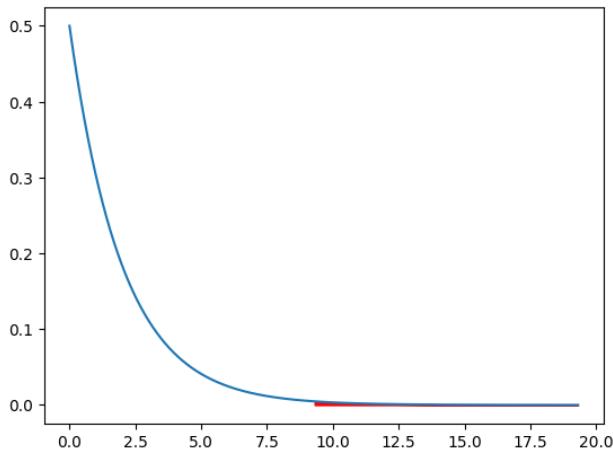
alcohol	177	0.3333	151.0	2.1158	4.4768
cocaine	152	0.3333	151.0	0.0814	0.0066
heroin	124	0.3333	151.0	-2.1972	4.8278

```

df = len(O)-1 = 3-1 = 2
Chi-Squared = sum( (O-E)**2/E )= sum( Residuals**2) =
=(177-151.00)**2/151.00 + ... + (124-151.00)**2/151.00 = 9.311
p-value = 1 - chi2.cdf(X2,df) = 0.009507929549892102

```

From chi-squared function: X2 = 9.311 pval = 0.009507929549892056



The observed values add up to  $Total = 453$ . If the observed values follow the expected proportions,  $453 \cdot \frac{1}{3} = 151$  should be the expected count for each substance (*none of them is below 5*). The resulting test statistic  $\chi^2$  is:

$$\chi^2 = \frac{(O_1 - E_1)^2}{E_1} + \frac{(O_2 - E_2)^2}{E_2} + \dots + \frac{(O_k - E_k)^2}{E_k} =$$

$$\frac{(177 - 151)^2}{151} + \frac{(152 - 151)^2}{151} + \frac{(124 - 151)^2}{151} =$$

$$4.477 + 0.007 + 4.828 = 9.311$$

The chi-squared distribution with  $df = k - 1 = 3 - 1 = 2$  degrees of freedom is shown in the Figure above. Here,  $\chi^2 = 9.311$  and the resulting p-value =  $0.0095 < 0.05$ , as shown in the chi-squared distribution Figure above. Thus, there is enough evidence to reject  $H_0$ ; the substances are not distributed equally. There are more alcoholics than any other kind. It can also be seen in the side-by-side bar plot of the observed and expected counts above.

### Example

An M&M candy pack is supposed to have 23% blue, 23% orange, 15% green, 15% yellow, 12% red, and 12% brown candies. A random sample of several packs is selected and different colors are counted. Do the observed counts follow the claimed proportions?

$H_0$ : The observed counts of colors follow the expected proportions  $O \approx E$ .  
 $H_1$ : The observed values do NOT follow the expected proportions or  $O \neq E$ .

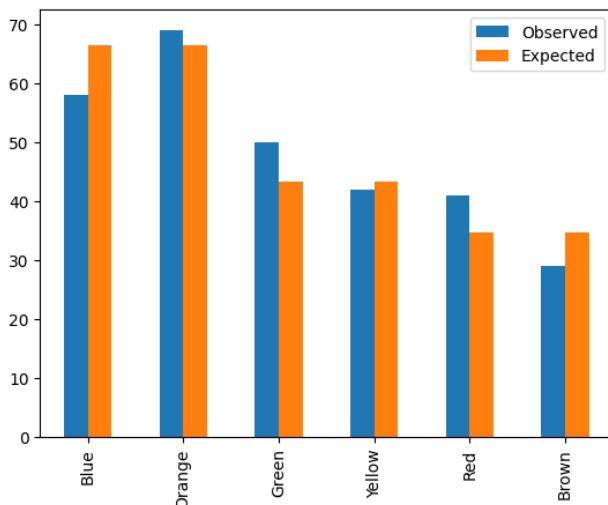
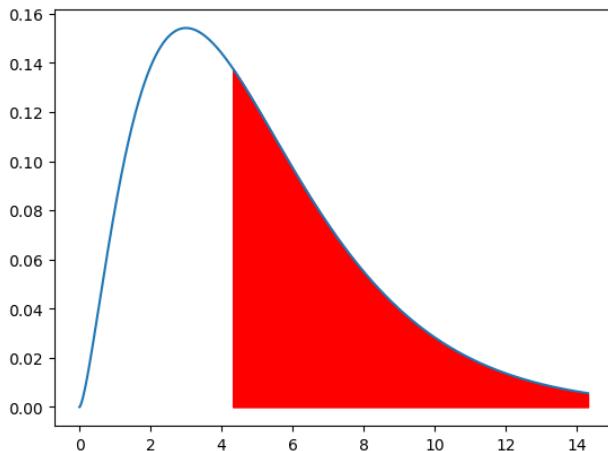
```
[ ]: GoodnessFit(Observed=[58, 69, 50, 42, 41, 29],
                  BinNames=["Blue", "Orange", "Green", "Yellow", "Red", "Brown"],
                  Pexpect=[0.23, 0.23, 0.15, 0.15, 0.12, 0.12])
```

```
Total Observed = 58 + 69 + ... 29 = 289
Expected frequencies = E = total*P_exp =289*[0.230,0.230,...,0.120]=
[66.470,66.470,...,34.680]
Residuals = (Observed - Expected)/sqrt(Expected) =
=(58-66.47)/sqrt(66.47), ..., (29-34.68)/sqrt(34.68)
```

	Observed	Expected	P	Expected	Residuals	Residuals**2
Blue	58	0.23		66.47	-1.0389	1.0793
Orange	69	0.23		66.47	0.3103	0.0963
Green	50	0.15		43.35	1.0100	1.0201
Yellow	42	0.15		43.35	-0.2050	0.0420
Red	41	0.12		34.68	1.0732	1.1517
Brown	29	0.12		34.68	-0.9645	0.9303

```
df = len(O)-1 =6-1 = 5
Chi-Squared = sum( (O-E)**2/E ) = sum( Residuals**2) =
=(58-66.47)**2/66.47 + ... + (29-34.68)**2/34.68 = 4.320
p-value = 1 - chi2.cdf(X2,df) = 0.5043501118366657
```

```
From chi-squared function: X2 = 4.32    pval = 0.5043501118366656
```



Observed values sum up to  $Total = 289$ . If they follow the expected proportions, there should be  $289 \cdot 0.23 \approx 66.47$  Blue, etc.. The Expected Values are in the table above, *none of them is below 5*. The resulting test statistic  $\chi^2$  is:

$$\chi^2 = \frac{(O_1 - E_1)^2}{E_1} + \frac{(O_2 - E_2)^2}{E_2} + \dots + \frac{(O_k - E_k)^2}{E_k} =$$

$$\frac{(58 - 66.47)^2}{66.47} + \frac{(69 - 66.47)^2}{66.47} + \dots + \frac{(29 - 34.68)^2}{34.68} =$$

$$1.079 + 0.096 + \dots + 0.93 = 4.32$$

The chi-squared distribution with  $df = k - 1 = 6 - 1 = 5$  degrees of freedom is shown above. The  $\chi^2 = 4.32$  is small and produces a very large p-value  $= 0.5044 > 0.05$  (very large right tail) as shown in the chi-squared distribution Figure above. Thus, there is NOT enough evidence to reject  $H_0$ , so the observed colors of candies follow the manufacturer's proportions claim. We can also see it in the side-by-side bar plot of the observed and expected counts above.

### Example

For a number of numerical data sets, the leading digit distribution is surprisingly heavily skewed right (Benford Law): leading 1 - 30% likely, leading 2 - 17.6%, ..., leading 9 < 5%. If they were uniformly distributed, each of the nine digits would occur about 11.1% of the time. The formula for the probability distribution of the first digit  $d = 1, 2, \dots, 9$  is given by:

$$P(d) = \log_{10}(d+1) - \log_{10}(d) = \log_{10}(1 + 1/d)$$

The observed counts of leading digits from a large financial document are given below. Do they follow Benford Law?

$H_0$ : The observed values of the leading digits follow the expected proportions  $O \approx E$ .

$H_1$ : The observed values do NOT follow the expected proportions or  $O \neq E$ .

```
[ ]: import numpy as np
nv = np.arange(1,10)
P_exp = np.array(np.log10(nv+1)-np.log10(nv)); P_exp
# log10(2)-log10(1), log10(3)-log10(2), log10(4)-log10(3)
```

```
[ ]: array([0.30103    , 0.17609126, 0.12493874, 0.09691001, 0.07918125,
       0.06694679, 0.05799195, 0.05115252, 0.04575749])
```

```
[ ]: GoodnessFit(Observed=[1110,601,462,345,260,223,205,180,156],
               BinNames=["1","2","3","4","5","6","7","8","9"],
               Pexpect=P_exp)
```

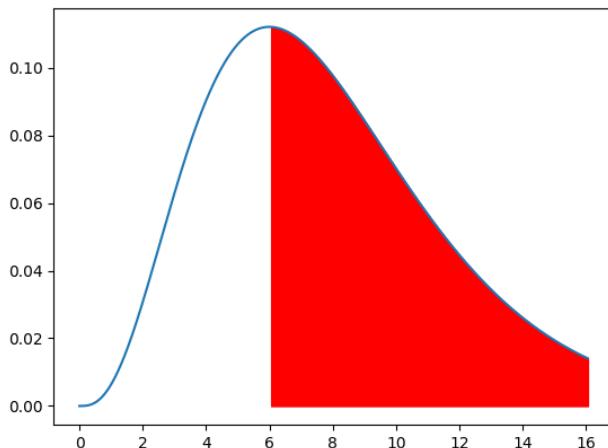
```
Total Observed = 1110 + 601 + ... 156 = 3542
Expected frequencies = E = total*P_exp =3542*[0.301,0.176,...,0.046]=
[1066.248,623.715,...,162.073]
```

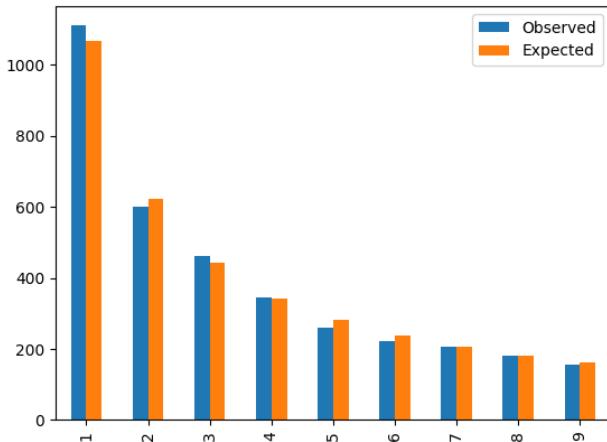
```
Residuals = (Observed - Expected)/sqrt(Expected) =
=(1110-1066.25)/sqrt(1066.25), ... , (156-162.07)/sqrt(162.07)
```

	Observed	Expected	P	Expected	Residuals	Residuals**2
1	1110	0.3010		1066.2482	1.3399	1.7953
2	601	0.1761		623.7152	-0.9095	0.8273
3	462	0.1249		442.5330	0.9254	0.8564
4	345	0.0969		343.2553	0.0942	0.0089
5	260	0.0792		280.4600	-1.2217	1.4926
6	223	0.0669		237.1255	-0.9173	0.8415
7	205	0.0580		205.4075	-0.0284	0.0008
8	180	0.0512		181.1822	-0.0878	0.0077
9	156	0.0458		162.0730	-0.4770	0.2276

```
df = len(O)-1 = 9-1 = 8
Chi-Squared = sum( (O-E)**2/E )= sum( Residuals**2) =
=(1110-1066.25)**2/1066.25 + ... + (156-162.07)**2/162.07 = 6.058
p-value = 1 - chi2.cdf(X2,df) = 0.6407462181172101
```

From chi-squared function: X2 = 6.058 pval = 0.6407462181172101





The observed counts add up to  $Total = 3542$ . If the observed counts follow the expected proportions,  $3542 \cdot 0.301 \approx 1066.248$  of the leading digits should be 1, etc. The Expected Values are shown in the table and the second Figure above, *none of them is below 5*. The resulting test statistic  $\chi^2$  is:

$$\begin{aligned}\chi^2 &= \frac{(O_1 - E_1)^2}{E_1} + \frac{(O_2 - E_2)^2}{E_2} + \dots + \frac{(O_k - E_k)^2}{E_k} = \\ &\frac{(1110 - 1066.248)^2}{1066.248} + \frac{(601 - 623.715)^2}{623.715} + \dots + \frac{(156 - 162.073)^2}{162.073} = \\ &1.795 + 0.827 + \dots + 0.228 = 6.058\end{aligned}$$

The  $\chi^2$  follows a chi-squared distribution with  $df = k - 1 = 9 - 1 = 8$  degrees of freedom shown in the Figure above. Here,  $\chi^2 = 6.058$  is quite small resulting in a very large p-value = 0.64 > 0.05 (right tail in the chi-squared plot in the Figure above). Therefore, there is NOT enough evidence to reject H0; the observed distribution of leading digits in this financial report follows Benford's law. We can also see it in the side-by-side bar plot of the observed and expected counts above.

[ ]:

[ ]:

---

## 6.2 Chi-Squared Test of Independence in a Two-Way Table

In this section two-way contingency tables of frequency counts for categorical data are studied to assess whether there is a dependence between these two variables.

### Example

A supplement company claims that their extract is effective in preventing common cold viruses. Healthy volunteers randomized into groups were given a placebo, low dose, or high dose of the supplement and exposed to a cold virus. The results are summarized in the Table below. Test the claim that getting a cold infection is independent of the treatment group (i.e., row and column variables are independent) at a 5% level.

	Placebo	Low Dose	High Dose	Sum
Infected	10	28	31	69
Not Infected	40	60	64	164
Sum	50	88	95	233

$H_0$ : Getting an infection is independent of the treatment (i.e., **row and column variables are independent**).

$H_1$ : **Dependent**.

The  $\chi^2$  statistic approach is used again, but what are the expected counts? As always, initially assume  $H_0$  is true - getting an infection is independent of the treatment group. Let's concentrate on a particular cell in the upper left corner:

$$P(\text{Infected and Placebo}) = [H_0 \text{ true} \implies \text{independent}] =$$

$$P(\text{Infected}) \cdot P(\text{Placebo}) = \frac{69}{233} \cdot \frac{50}{233}$$

Therefore, the expected count in this cell should be:

$$E = (\text{Grand Total}) \cdot (\text{Probability}) = 233 \cdot \frac{69}{233} \cdot \frac{50}{233} = \frac{69 \cdot 50}{233}$$

Therefore, the expected value in each cell is given by:

$$E = \frac{(\text{Row Total}) \cdot (\text{Column Total})}{\text{Grand Total}} \tag{6.1}$$

Therefore:

$$E_{1,1} = \frac{69 \cdot 50}{233} = 14.807$$

$$E_{1,2} = \frac{69 \cdot 88}{233} = 26.06$$

.....

$$E_{2,3} = \frac{164 \cdot 95}{233} = 66.87$$

Given the expected counts, the same  $\chi^2$  computations as in Goodness of Fit are performed, but the degree of freedom is:

$$df = (\text{number of rows} - 1)(\text{number of columns} - 1) = (2 - 1)(3 - 1) = 2$$

To illustrate this degree of freedom, let's consider the same contingency table as above, but specify only two cell entries. Then all other entries could be found by subtracting from the totals. For example, the entry for Infected and High Dose is  $69 - 10 - 28 = 31$ , or Not Infected and Placebo is  $50 - 10 = 40$ .

	Placebo	Low Dose	High Dose	Sum
Infected	10	28	?	69
Not Infected	?	?	?	164
Sum	50	88	95	233

The  $\chi^2$  statistic is

$$\chi^2 = \frac{(O_{1,1} - E_{1,1})^2}{E_{1,1}} + \frac{(O_{2,1} - E_{2,1})^2}{E_{2,1}} + \dots + \frac{(O_{2,3} - E_{2,3})^2}{E_{2,3}} =$$

$$\frac{(10 - 14.807)^2}{14.807} + \frac{(28 - 26.060)^2}{26.060} + \dots + \frac{(64 - 66.867)^2}{66.867} =$$

$$1.56 + 0.144 + \dots + 0.123 = 2.837$$

The  $\chi^2$  follows a chi-squared distribution with  $df = (2 - 1)(3 - 1) = 2$  degrees of freedom shown in the Figure below.  $\chi^2 = 2.837$  is very small and results in a very large p-value = 0.2420422 > 0.05 (the right tail in the Figure). Therefore, there is NOT enough evidence to reject H0, so getting infected is

independent of the treatment group which implies that the supplement is not effective. Note in the code below that the counts must be entered column-by-column, and row names are in the data frame index. The second Figure shows a stacked barplot of the data.

```
[ ]: def ChiSqIndependence(Observed):
    import numpy as np;   import matplotlib.pyplot as plt;
    import pandas as pd;  from scipy.stats import chi2_contingency;
    from scipy.stats import chi2

    O = Observed;  nr = Observed.shape[0];  nc = Observed.shape[1];

    Om = pd.DataFrame(O).copy()    # creating margins
    Om['Sum'] = O.sum(axis=1)
    rowsum = O.sum(axis=1)
    colsum = O.sum(axis=0)
    Om.loc[len(Om.index)] = colsum.tolist() + [O.sum().sum()]
    print(Om)
    print('nr, nc = ', nr,nc)

    chi_val, p_val, dof, E = chi2_contingency(O, correction=False)

    print('Expected in cell E00 = {:.2f}*{:.2f}/{:.2f} = {:.3f}'.
          format(rowsum[0],colsum[0],O.sum().sum(),E[0,0]))
    print('Expected in cell E01 = {:.2f}*{:.2f}/{:.2f} = {:.3f}'.
          format(rowsum[0],colsum[1],O.sum().sum(),E[0,1]))
    print('Expected in the last cell = {:.2f}*{:.2f}/{:.2f} = {:.3f}'.
          format(rowsum[nr-1],colsum[nc-1],O.sum().sum(),E[nr-1,nc-1]))

    print('Chi-Squared = sum( (O-E)**2/E ) = sum( Residuals**2 ) = ')
    print('=( {:.2f}- {:.2f})**2/{:.2f} + ... + ( {:.2f}- {:.2f})**2/{:.2f} = {:.3f}'.
          format(O.iloc[0,0],E[0,0],E[0,0],O.iloc[nr-1,nc-1],
                 E[nr-1,nc-1],E[nr-1,nc-1],chi_val))

    print('degree of freedom = df =(nr-1)(nc-1)= ( {:.2f}-1)( {:.2f}-1) = {:.2f}'.
          format(nr,nc,dof))
    pval = 1 - chi2.cdf(chi_val,df=dof)
    print('pval = 1 - chi2.cdf(X2,df) = ',pval,'\\n')

    print("From built-in chi-squared : X2 = ", round(chi_val,3), ' pval = ',pval)
    print("Expected: ")
    print(E)

    from scipy.stats import chi2
    x = np.arange(0, chi_val+10, 0.001)
    P1 = plt.plot(x, chi2.pdf(x, df=dof))
    xR=np.arange(chi_val,chi_val+10,0.001)
    plt.fill_between(xR,chi2.pdf(xR, df=dof),color='r') # filling for p-value
    plt.show(P1)
    O.plot.bar(stacked=True)

[ ]: import pandas as pd
O = pd.DataFrame({'Placebo':[10,40], 'LowDose':[28,60], 'HighDose':[31,64]},
                  index= ['Infected','NotInfected'])
```

```
ChiSqIndependence(0)
```

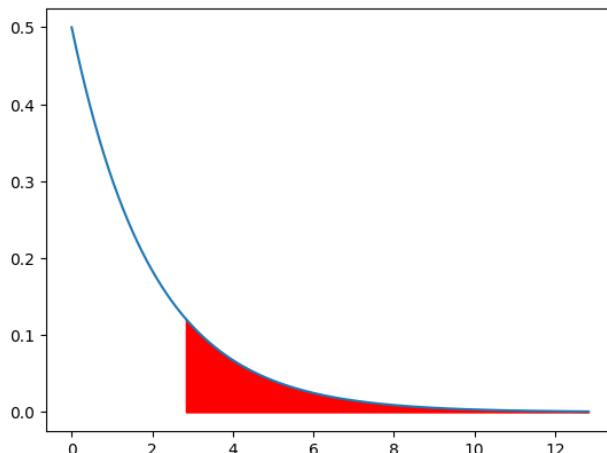
	Placebo	LowDose	HighDose	Sum
Infected	10	28	31	69
NotInfected	40	60	64	164
2	50	88	95	233

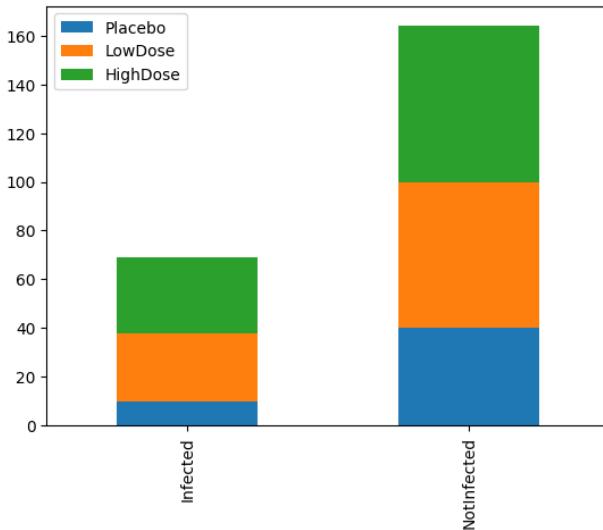
nr, nc = 2 3  
 Expected in cell E00 = 69\*50/233 = 14.807  
 Expected in cell E01 = 69\*88/233 = 26.060  
 Expected in the last cell = 164\*95/233 = 66.867  
 Chi-Squared = sum( (O-E)\*\*2/E )= sum( Residuals\*\*2) =  
 =(10-14.81)\*\*2/14.81 + ... + (64-66.87)\*\*2/66.87 = 2.837  
 degree of freedom = df =(nr-1)(nc-1)= (2-1)(3-1) = 2  
 pval = 1 - chi2.cdf(X2,df) = 0.24204221633323164

From built-in chi-squared : X2 = 2.837 pval = 0.24204221633323164

Expected:

```
[[14.80686695 26.06008584 28.13304721]
 [35.19313305 61.93991416 66.86695279]]
```





Note that when there are only two rows (as in this case) or only two columns, we can look at this problem from the proportion comparison point of view:

$H_0: p_1 = p_2 = p_3$  proportions of infected individuals in each treatment group are the same.

$H_1$ : At least one of the proportions is different.

As such, it is a generalization of the two-proportion test studied before. In fact, the proportions of infected individuals in our example are given below for Placebo, low dose, and high dose of the supplement, respectively.

```
[ ]: 0.iloc[0,:]/0.sum(axis=0)
```

```
[ ]: Placebo      0.200000
LowDose      0.318182
HighDose     0.326316
dtype: float64
```

```
[ ]:
```

### Example

A polling company conducted a study to determine the support for the National healthcare plan among randomly chosen respondents with different party affiliations. The data are shown in the code below. Test the claim that the response is independent of the party affiliation (i.e., row and column variables are independent).

$H_0$ : The response is independent of the party affiliation (i.e., row and column variables are independent)

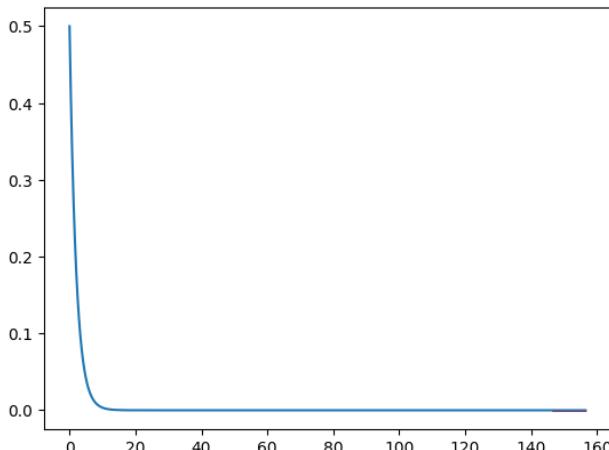
*H*1: Dependent.

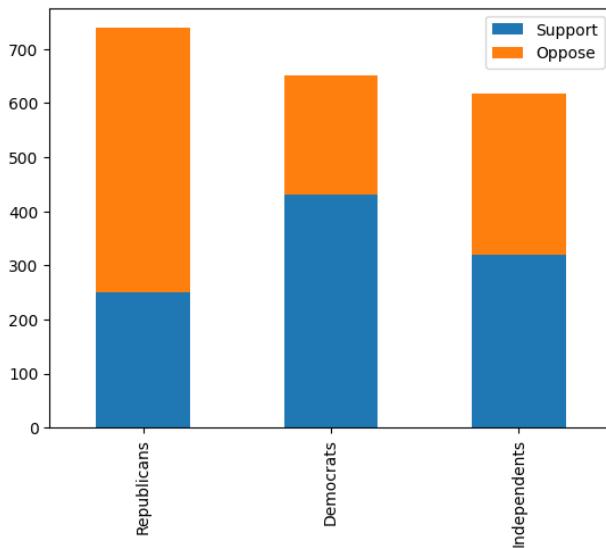
Let's start with the code and then explain the steps. Note that none of the expected counts are below 5, so the assumptions of the applicability of  $\chi^2$  test hold.

```
[ ]: import pandas as pd
O = pd.DataFrame({'Support':[250,431,320], 'Oppose':[489,220,298]}, 
                  index= ['Republicans', 'Democrats', 'Independents'])
ChiSqIndependence(0)

Support  Oppose  Sum
Republicans    250     489   739
Democrats      431     220   651
Independents   320     298   618
3            1001    1007  2008
nr, nc = 3 2
Expected in cell E00 = 739*1001/2008 = 368.396
Expected in cell E01 = 739*1007/2008 = 370.604
Expected in the last cell = 618*1007/2008 = 309.923
Chi-Squared = sum( (O-E)**2/E )= sum( Residuals**2) =
=(250-368.40)**2/368.40 + ... + (298-309.92)**2/309.92 = 146.450
degree of freedom = df =(nr-1)(nc-1)= (3-1)(2-1) = 2
pval = 1 - chi2.cdf(X2,df) = 0.0

From built-in chi-squared : X2 = 146.45    pval = 0.0
Expected:
[[368.39591633 370.60408367]
 [324.52739044 326.47260956]
 [308.07669323 309.92330677]]
```





The  $\chi^2 = 146.45$  is very large producing a very small p-value =  $1.5803313 \cdot 10^{-32} << 0.05$ , so there is more than enough evidence to reject  $H_0$  as shown in the chi-squared distribution Figure above. The response to the National plan is very much dependent on the party affiliation - Republicans mostly oppose any national plans, Democrats mostly support, and independents are about 50/50. We can also see it in the stacked bar plot above.

There are only two columns in this table, therefore the problem could be re-written from the proportion comparison point of view:

$H_0 : p_1 = p_2 = p_3$  proportions of support in each party are the same.

$H_1$ : At least one of the proportions is different.

The proportions of support in our example are given below for Republicans, Democrats, and Independents, respectively, and according to the problem results, they are significantly different.

```
[ ]: 0.iloc[:,0]/0.sum(axis=1)
```

```
[ ]: Republicans      0.338295
Democrats        0.662058
Independents     0.517799
dtype: float64
```

### Example

A marketing study of shopping habits by social class produced the data below. Investigate if the brand choice is independent of the social class (i.e., row and column variables are independent).

$H_0$ : The brand choice is independent of the social class (i.e., row and column variables are independent).

$H_1$ : Dependent.

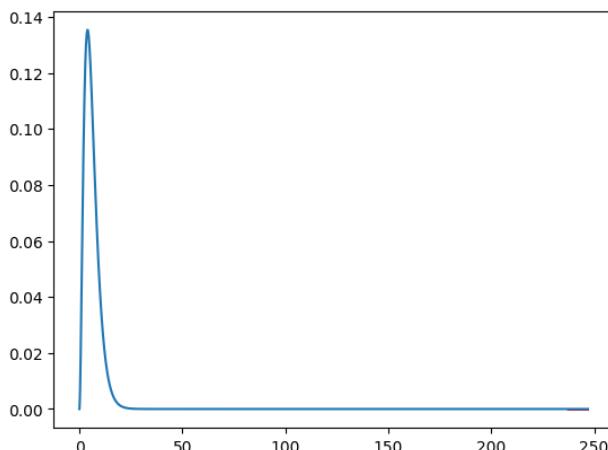
None of the expected counts below are less than 5, so  $\chi^2$  approach is applicable.

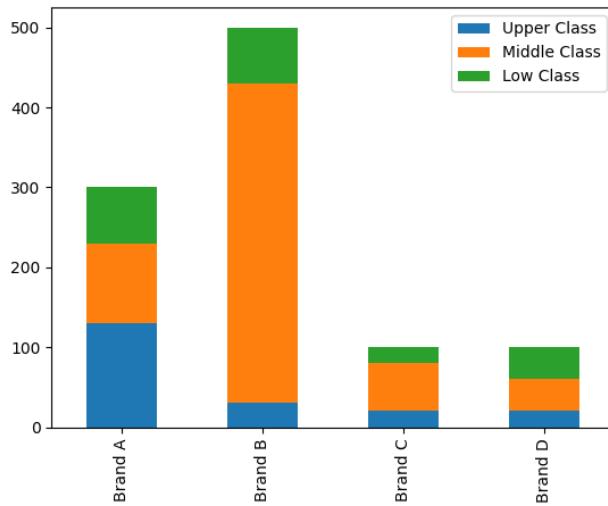
```
[ ]: import pandas as pd
O = pd.DataFrame({'Upper Class':[130,30,20,20], 'Middle Class':
→[100,400,60,40],
'Low Class':[70,70,20,40]}, 
index= ['Brand A','Brand B','Brand C','Brand D'])
ChiSqIndependence(0)
```

	Upper Class	Middle Class	Low Class	Sum
Brand A	130	100	70	300
Brand B	30	400	70	500
Brand C	20	60	20	100
Brand D	20	40	40	100
4	200	600	200	1000

nr, nc = 4 3  
Expected in cell E00 = 300\*200/1000 = 60.000  
Expected in cell E01 = 300\*600/1000 = 180.000  
Expected in the last cell = 100\*200/1000 = 20.000  
Chi-Squared = sum( (O-E)\*\*2/E )= sum( Residuals\*\*2) =  
=(130-60.00)\*\*2/60.00 + ... + (40-20.00)\*\*2/20.00 = 236.889  
degree of freedom = df =(nr-1)(nc-1)= (4-1)(3-1) = 6  
pval = 1 - chi2.cdf(X2,df) = 0.0

From built-in chi-squared : X2 = 236.889 pval = 0.0  
Expected:  
[[ 60. 180. 60.]  
 [100. 300. 100.]  
 [ 20. 60. 20.]  
 [ 20. 60. 20.]]





The  $\chi^2 = 236.89$  is very large producing essentially 0 p-value =  $2.591575 \cdot 10^{-49} << 0.05$ , as shown in the chi-squared distribution Figure above. There is more than enough evidence to reject  $H_0$  - the brand choice is very much dependent on social class. We can also see it in the stacked bar plot above. Note that in this case there are more than two row and column levels, so this problem cannot be thought of as a proportion test.

### Example

Consider the `HELPrc` data file again. Test the claim that the gender (`sex`) is independent of the preferred substance (`substance`) at a 5% level.

$H_0$ : The substance is independent of the gender (i.e., row and column variables are independent).

$H_1$ : Dependent.

First, since the data file is available, `pd.crosstab()` is used to tabulate `substance` and `sex` (the row and column names are automatically assigned). Its output is directly fed into my function. None of the expected counts are below 5, so the  $\chi^2$  test can be applied.

```
[ ]: import pandas as pd
url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrc.csv"
mydata = pd.read_csv(url)

0 = pd.crosstab(mydata.substance, mydata.sex, margins=False)
ChiSqIndependence(0)
```

```

sex      female   male   Sum
substance
alcohol      36     141    177
cocaine      41     111    152
heroin       30      94    124
3           107    346    453
nr, nc = 3 2
Expected in cell E00 = 177*107/453 = 41.808
Expected in cell E01 = 177*346/453 = 135.192
Expected in the last cell = 124*346/453 = 94.711
Chi-Squared = sum( (O-E)**2/E ) = sum( Residuals**2 ) =
=(36-41.81)**2/41.81 + ... + (94-94.71)**2/94.71 = 2.026
degree of freedom = df =(nr-1)(nc-1) = (3-1)(2-1) = 2
pval = 1 - chi2.cdf(X2,df) = 0.3630623650067597

```

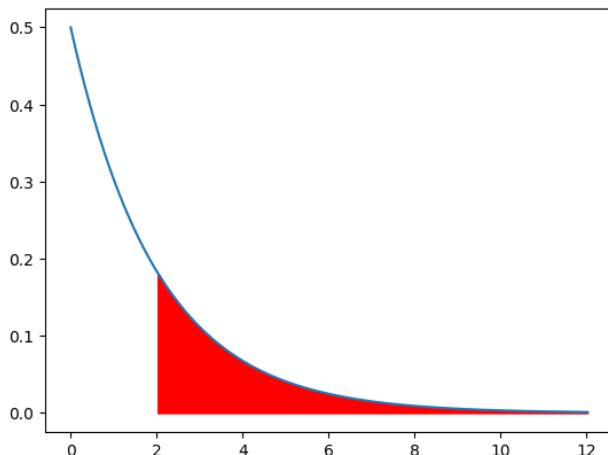
From built-in chi-squared : X2 = 2.026 pval = 0.3630623650067597

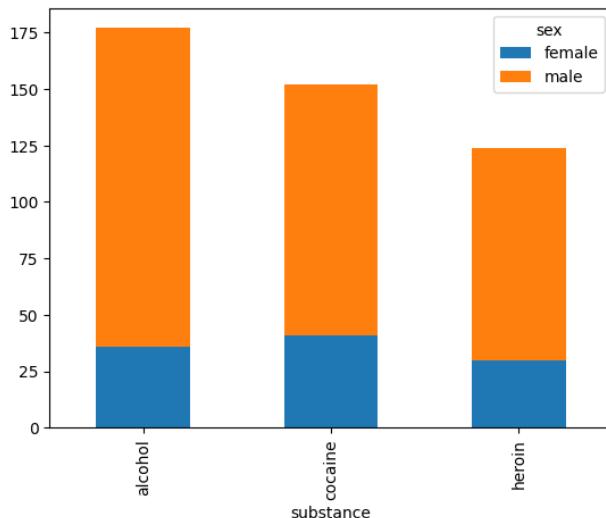
Expected:

```

[[ 41.80794702 135.19205298]
 [ 35.90286976 116.09713024]
 [ 29.28918322 94.71081678]]

```





$\chi^2 = 2.026$  is very small resulting in a large p-value = 0.363 > 0.05, as shown in the chi-squared distribution Figure above. Therefore, there is *not* enough evidence to reject H<sub>0</sub> - substance is independent of gender. The stacked bar plot above shows it as well.

With only two columns, the problem can be reformulated as a proportion test:  
 $H_0: p_1 = p_2 = p_3$  proportions of males in each substance.

$H_1:$  At least one of the proportions is different.

The observed proportions of males for each substance are:

```
[1]: 0.iloc[:,1]/0.sum(axis=1)
```

```
[2]: substance
alcohol      0.796610
cocaine     0.730263
heroin      0.758065
dtype: float64
```

Based on the results, these proportions are *not* significantly different.

### Example

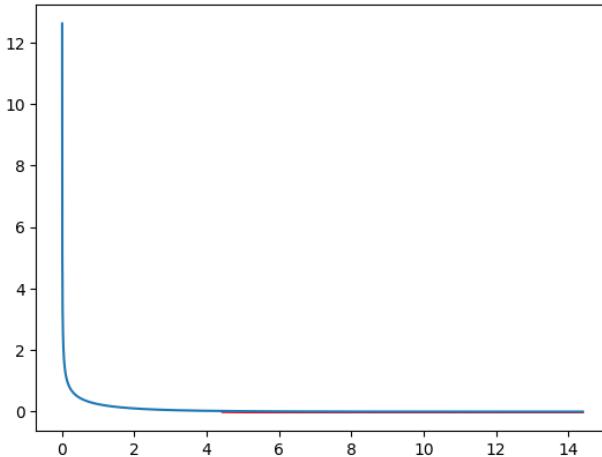
Let's come back to an example considered in the previous chapter. First, we repeat the main idea that 87 out of 100 students preparing with the Barron guide passed the Medical admission test MCAT, while 91 out of 120 passed it with the Princeton guide. Is there a significant difference at the 1% level? How about a 5% level? We recast this problem as a contingency table as follows: test the claim that the passing is independent of the type of review used (i.e.,

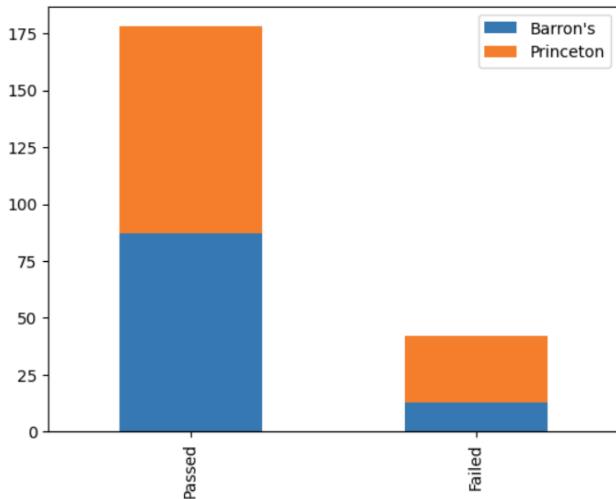
row and column variables are independent).

```
[ ]: import pandas as pd
O = pd.DataFrame({'Barrons':[87,13], 'Princeton':[91,29]}, 
                  index= ['Passed', 'Failed'])
ChiSqIndependence(0)
```

	Barrons	Princeton	Sum
Passed	87	91	178
Failed	13	29	42
2	100	120	220
nr, nc =	2 2		
Expected in cell E00 =	178*100/220 = 80.909		
Expected in cell E01 =	178*120/220 = 97.091		
Expected in the last cell =	42*120/220 = 22.909		
Chi-Squared = sum( (O-E)**2/E ) = sum( Residuals**2) =			
= (87-80.91)**2/80.91 + ... + (29-22.91)**2/22.91 = 4.403			
degree of freedom = df = (nr-1)(nc-1) = (2-1)(2-1) = 1			
pval = 1 - chi2.cdf(X2,df) = 0.03586871998573926			

```
From built-in chi-squared : X2 = 4.403    pval = 0.03586871998573926
Expected:
[[80.90909091 97.09090909]
 [19.09090909 22.90909091]]
```





The  $\chi^2 = 4.4033$  results in  $p\text{-value} = 0.0358 > 0.01$ , as shown in the chi-squared distribution Figure above. Therefore, there is *not* enough evidence to reject  $H_0$  at 0.01 level, so at this stricter level, the MCAT passing is independent of the preparation type. We can also see it in the stacked bar plot above. On the other hand, if  $\alpha = 0.05$  were used, we would have rejected  $H_0$ .

Comparing these results with the 2-proportions test in the previous chapter, we notice that the p-values are the same and  $z^2 = 2.098^2 = 4.4033 = \chi^2$ , which is true in general.



Taylor & Francis  
Taylor & Francis Group  
<http://taylorandfrancis.com>

# Inference for Means

---

In previous chapters, we concentrated on categorical variables, so we could only consider counts and proportions. In this chapter, we will look at numerical variables, so means tests can be considered.

---

## 7.1 One-Sample Mean Tests

Similar to modeling the sample proportion  $\hat{p}$  with normal distribution, the sample mean  $\bar{x}$  can also be modeled with a normal distribution as shown below.

### Central Limit Theorem for the Means

If a sample of size  $n$  of independent observations from a population with **any distribution** having mean  $\mu$  and standard deviation  $\sigma$  is sufficiently large, the sampling distribution of sample means  $\bar{x}$  will be nearly normal with

$$\text{Mean} = \mu \quad \text{Standard Error} = SE = \frac{\sigma}{\sqrt{n}} \quad (7.1)$$

Note, in addition, that *if the original population distribution is normal*, then this theorem holds for samples of **any size**.

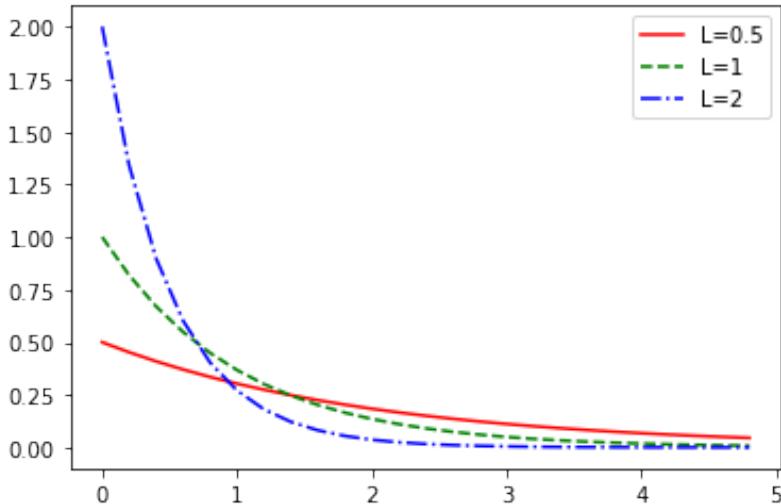
What is remarkable is that this is true for *any distribution*, the original distribution does *NOT have to be bell-shaped*. Proving this theorem, much like the Central Limit Theorem (CLT) for proportions, would require a more advanced course, but it can be illustrated with a simulation again. We use exponential distribution which is always skewed right as shown in the Figure below. A waiting time in line or lifetimes of some devices tend to have such skewed right distributions. We use it here purely for illustration purposes.

```
[ ]: from scipy.stats import expon; import pandas as pd;
import matplotlib.pyplot as plt; import numpy as np;
L = 0.5; # shape parameter lambda, scale = 1 / lambda = mean
xv = np.arange(0,5,0.2)
dv1 = expon.pdf(x=xv,scale=1/L)
plt.plot(xv,dv1,'r-',label='L=0.5')
```

```
L = 1;
dv2 = expon.pdf(x=xv,scale=1/L)
plt.plot(xv,dv2,'g--',label='L=1')

L = 2;
dv3 = expon.pdf(x=xv,scale=1/L)
plt.plot(xv,dv3,'b-.',label='L=2')
plt.legend()
```

[ ]: <matplotlib.legend.Legend at 0x7f3ee4e30280>



The code below simulates the distribution of sample means from an exponential distribution with a mean of  $1/2$ . The sample sizes are  $n = 100$ .

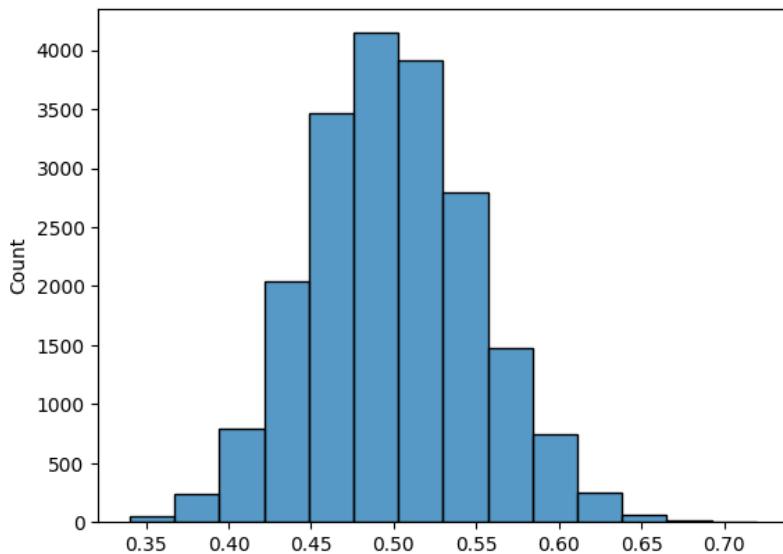
```
# Simulation of sample means sample size n = 100
import seaborn as sns; import numpy as np

L = 2;
n = 100; numsamples = 20000
xv = np.repeat(0.0,numsamples) # sample means
for j in range(numsamples):

    onesample = np.random.exponential(scale=1/L, size=n)
    xv[j] = np.mean(onesample)

sns.histplot(xv,bins=14) # create histogram of sample means
xvbar = np.mean(xv); # mean of sample means
xvsd1 = np.std(xv,ddof=1); # standard deviation of sample pmeans
print('xvbar = {:.6f}, mu = {:.1f}, xvsd1 = {:.6f}, sig/sqrt(n) = {:.6f}' )
```

```
.format(xvbar, 1/L, xvsd1, (1/L)/np.sqrt(n)))  
  
xvbar = 0.499573, mu = 0.5, xvsd1 = 0.050356, sig/sqrt(n) = 0.050000
```



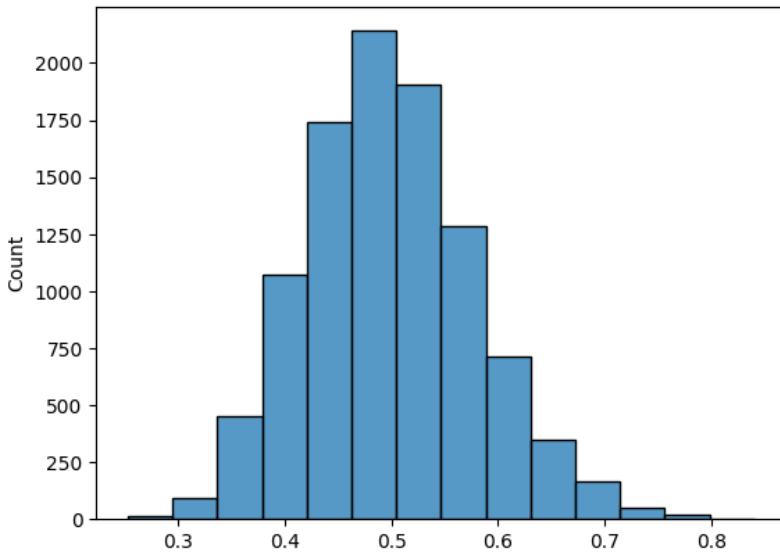
The distribution of sample means (**sampling distribution**) is symmetric and bell-shaped (*normal distribution*) as can be seen in the Figure above. The center (*mean*) of this distribution of sample means is very close to the original population mean  $\mu = \frac{1}{\lambda} = \frac{1}{2}$ . The standard deviation (**standard error**) *SE* is also very close to the value predicted by the CLT  $SE = \frac{\sigma}{\sqrt{n}} = \frac{\frac{1}{\lambda}}{\sqrt{n}} = \frac{0.5}{\sqrt{100}} = 0.05$ .

If we try to run it for a smaller sample size of  $n = 30$ , it still works. The mean is still 0.5, but the standard deviation increased.

```
[ ]: # Simulation of sample means sample size n = 30  
import seaborn as sns; import numpy as np  
  
L = 2;  
n = 40; numsamples = 10000  
xv = np.repeat(0.0,numsamples) # sample means  
for j in range(numsamples):  
  
    onesample = np.random.exponential(scale=1/L, size=n)  
    xv[j] = np.mean(onesample)  
  
sns.histplot(xv,bins=14) # create histogram of sample means  
xvbar = np.mean(xv); # mean of sample means
```

```
xvsd1 = np.std(xv,ddof=1); # standard deviation of sample pmeans
print('xvbar = {:.6f}, mu = {:.1f}, xvsd1 = {:.6f}, sig/sqrt(n) = {:.6f}'
      .format(xvbar, 1/L, xvsd1, (1/L)/np.sqrt(n)))
```

```
xvbar = 0.499090, mu = 0.5, xvsd1 = 0.079335, sig/sqrt(n) = 0.079057
```



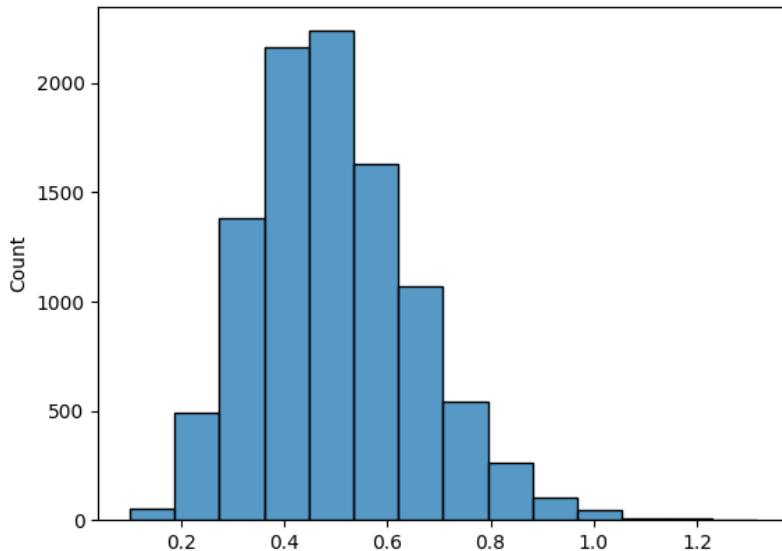
The simulation below shows a very small sample size  $n = 10$  and the distribution is skewed right and no longer bell-shaped as the Figure shows.

```
[ ]: # Simulation of sample means small sample size n = 10
import seaborn as sns; import numpy as np

L = 2;
n = 10; numsamples = 10000
xv = np.repeat(0.0,numsamples) # sample means
for j in range(numsamples):

    onessample = np.random.exponential(scale=1/L, size=n)
    xv[j] = np.mean(onessample)

sns.histplot(xv,bins=14) # create histogram of sample means
xvbar = np.mean(xv); # mean of sample means
xvsd1 = np.std(xv,ddof=1); # standard deviation of sample pmeans
print('xvbar = {:.6f}, mu = {:.1f}, xvsd1 = {:.6f}, sig/sqrt(n) = {:.6f}'
      .format(xvbar, 1/L, xvsd1, (1/L)/np.sqrt(n)))
```



```
xvbar = 0.498134, mu = 0.5, xvstd1 = 0.156169, sig/sqrt(n) = 0.158114
```

However, let's rerun the previous simulation with  $n = 10$  but sampling from a standard normal distribution. This results again in a normal sampling distribution shown in the Figure below, which illustrates the comment made at the end of the CLT that for a normal underlying distribution, the CLT holds for a sample of any size.

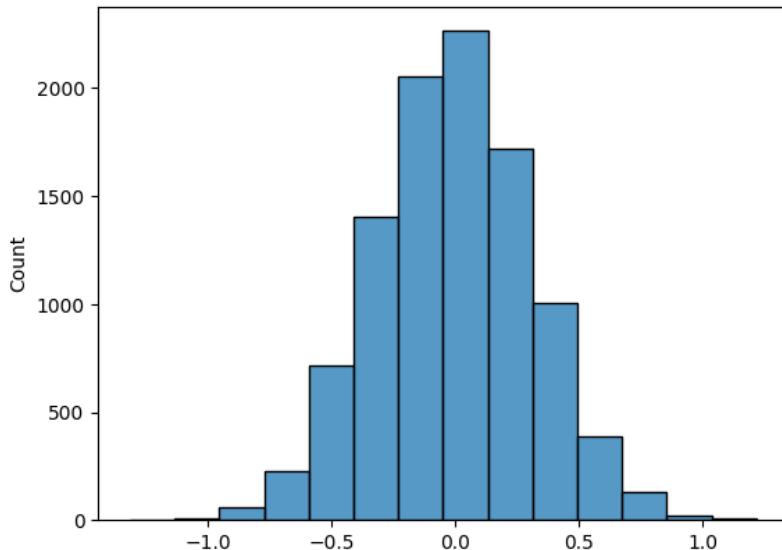
```
[ ]: # Simulation of sample means small sample size n = 10
import seaborn as sns; import numpy as np

n = 10; numsamples = 10000
xv = np.repeat(0.0,numsamples) # sample means
for j in range(numsamples):

    onesample = np.random.normal(size=n)
    xv[j] = np.mean(onesample)

sns.histplot(xv,bins=14) # create histogram of sample means
xvbar = np.mean(xv); # mean of sample means
xvstd1 = np.std(xv,ddof=1); # standard deviation of sample pmeans
print('xvbar = {:.6f}, mu = {:.1f}, xvstd1 = {:.6f}, sig/sqrt(n) = {:.6f}'.
      format(xvbar, 0, xvstd1, (1)/np.sqrt(n)))
```

```
xvbar = -0.008623, mu = 0.0, xvstd1 = 0.314911, sig/sqrt(n) = 0.316228
```



The CLT for sample means has an important complication compared to the CLT for sample proportions. For proportions, the standard error uses the same proportion  $p$ :  $SE = \sqrt{\frac{p(1-p)}{n}}$ , whereas for means, standard error  $SE = \frac{\sigma}{\sqrt{n}}$  is dependent on the population standard deviation,  $\sigma$  which is usually unknown and must be estimated. Any estimation is imperfect, so a more spread-out t-distribution is needed which will be introduced shortly.

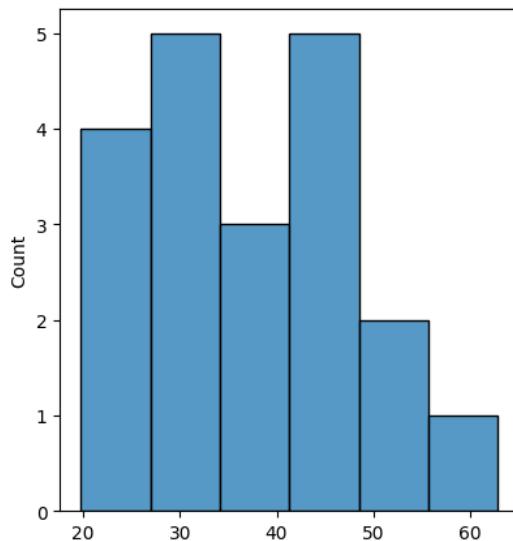
There are two conditions required to apply the Central Limit Theorem for a sample mean  $\bar{x}$ :

### 1. Independence

The sample observations must be independent. It is satisfied with a simple random of size less than 10% of the population, for example.

### 2. Normality

When a sample is **small**, the sample observations must come from a **normally** distributed population. The larger the sample gets, this condition can be **relaxed** more and more. This condition is admittedly more *vague*. For realistic data, especially with smaller sample sizes, there is no good way to check the normality condition. For example, the Figure below is a histogram of 20 randomly chosen numbers from a normal distribution, but it does not look bell-shaped at all.



Two **rules of thumb** are used instead:

- If the sample size  $n \leq 30$  and there are no clear outliers in the data, then it is assumed that the data come from a nearly normal distribution.
- If the sample size  $n > 30$  and there are no particularly extreme outliers, it is assumed the sampling distribution of  $\bar{x}$  is nearly normal (CLT theorem), even if the underlying individual distribution is not.

For example, consider the Figure below. The sample on the left has  $n = 20 < 30$  observations. It does not have any clear outliers (see the left Figure below). There is a peak in the histogram toward the right but it is not a clear outlier. Thus, the normality condition is met. On the other hand, the sample at the right has  $n = 50 > 30$  observations, but there are particularly extreme outliers, so the normality condition fails (see the right Figure below). If the actual data is available, there is a Shapiro test to evaluate the normality assumption (low p-value implies a failure of the null hypothesis of normality). The results correspond well with our visual cues. It should be noted though, that simulations show that the t-test procedure introduced in this chapter is **robust** - not heavily dependent on the normality assumption.

```
[ ]: import numpy as np; import seaborn as sns
import matplotlib.pyplot as plt; from scipy.stats import shapiro

x = [40.93, 40.94, 49.02, 23.93, 48.10, 45.56,
     38.55, 31.83, 23.12, 27.69, 27.34, 51.01,
     42.44, 52.21, 42.30, 36.55, 44.01, 30.51,
     46.71, 24.66, 70]
out1 = shapiro(x)
```

```

print('1st data pvalue = ', out1.pvalue)
fig, axs = plt.subplots(1, 2, figsize=(10,5))
sns.histplot(x,ax=axs[0])

x = np.random.normal(loc=40, scale=10, size=50);
x = np.append(x,[125,127]);
out1 = shapiro(x)
print('2nd data pvalue = ', out1.pvalue)
sns.histplot(x,ax=axs[1])

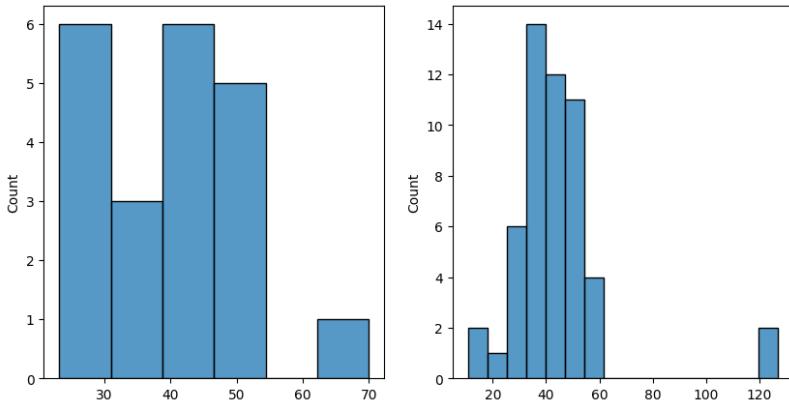
```

```

1st data pvalue =  0.24952353537082672
2nd data pvalue =  2.485375905081355e-09

```

[ ]: <Axes: ylabel='Count'>



When data is not directly available, we can try to assess the normality and skewness of the data based on experience. For example, consider the bank account distribution. The large majority of accounts would have moderate amounts, while a relatively tiny fraction has amassed millions and billions. Therefore, the distribution is extremely skewed. On the other hand, test scores tend to be bell-shaped, and we can use the CLT even for smaller sample sizes.

### 7.1.1 T-distribution

The Central Limit Theorem above had the true population standard deviation  $\sigma$  in the formula for standard error  $SE = \frac{\sigma}{\sqrt{n}}$ , which is usually unknown. For the distribution of sample proportions, the unknown true population

proportion  $p$  was approximated by the sample proportion  $\hat{p}$ :

$$SE = \sqrt{\frac{p(1-p)}{n}} \approx \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

Analogously, for sample means, the true population standard deviation  $\sigma$  is approximated by the sample standard deviation  $s$ :

$$SE = \frac{\sigma}{\sqrt{n}} \approx \frac{s}{\sqrt{n}}$$

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}}$$

The sample standard deviation  $s$  can estimate population value  $\sigma$  quite accurately for larger sample sizes  $n \geq 30$ . However, for a smaller sample  $n < 30$ , this estimate is less precise. In this case, W. Gosset proved that sample means distribution should be modeled by a slightly different **t-distribution**. It is also **symmetric and bell-shaped**, but it is **more spread out (has thicker tails)**. Therefore, observations are more likely to fall beyond 2 standard deviations away from the mean than under the normal distribution. Much like  $\chi^2$ , t-distribution is not one, but a family of distributions based on the

$$\text{degree of freedom} = df = n - 1 = (\text{number of observations}) - 1$$

Several t-distributions (with different degrees of freedom) and the standard normal distribution are shown in the Figure below. As the degree of freedom increases, the t-distributions converge to the standard normal. For  $df > 30$ , it is almost indistinguishable visually.

```
[ ]: from scipy.stats import norm, t; import pandas as pd;
import matplotlib.pyplot as plt; import numpy as np;

xv = np.arange(-6,6,0.01)
dv = t.pdf(x=xv,df = 2)
plt.plot(xv,dv,'r--',label='t, df=2')

dv = t.pdf(x=xv,df = 4)
plt.plot(xv,dv,'g--',label='t, df=4')

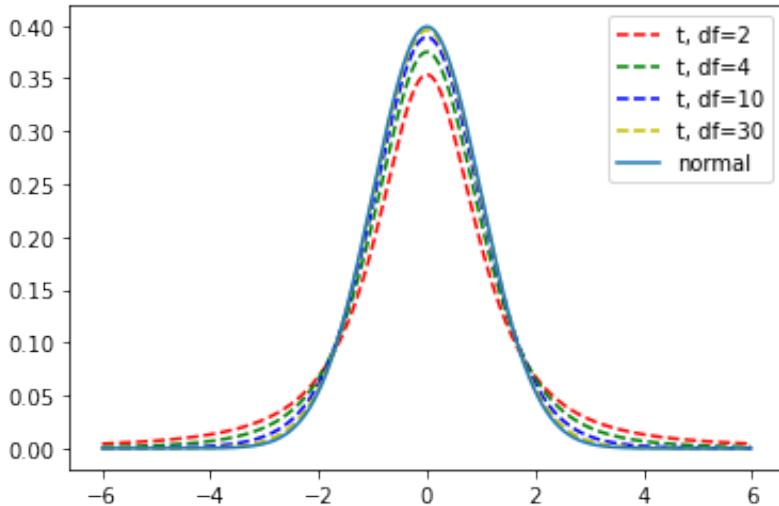
dv = t.pdf(x=xv,df = 10)
plt.plot(xv,dv,'b--',label='t, df=10')

dv = t.pdf(x=xv,df = 30)
plt.plot(xv,dv,'y--',label='t, df=30')

dv = norm.pdf(x=xv)
plt.plot(xv,dv,label='normal')

plt.legend()
```

```
[ ]: <matplotlib.legend.Legend at 0x7f5140ce4e50>
```



The Figure below shows our usual 90%, 95%, and 99% middle areas for confidence intervals on the t-distribution with  $df = n - 1 = 10 - 1 = 9$ . The overall shape is identical to the standard normal curve, but the critical values are larger and the areas are wider.

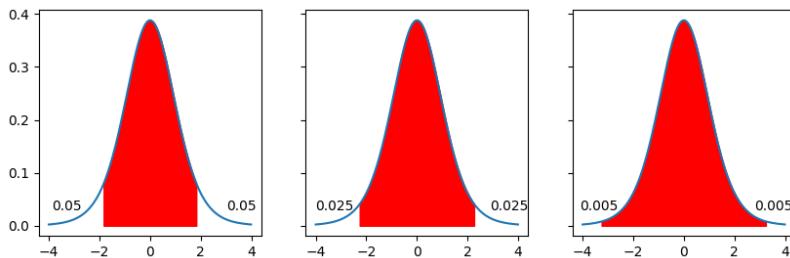
```
[ ]: import numpy as np; import matplotlib.pyplot as plt
from scipy.stats import t; import pandas as pd
fig, axs = plt.subplots(1, 3, figsize=(10, 3), sharey=True)
df1 = 9 # degree of freedom
xv = np.arange(-4, 4, 0.01)
ConfLevel = np.array([0.9, 0.95, 0.99])
alpha = 1-ConfLevel
tstar = t.ppf(1-alpha/2,df=df1);
df = pd.DataFrame({'ConfLevel':ConfLevel,'alpha':alpha,'tstar':tstar});
axs[0].plot(xv, t.pdf(xv, df=df1))
px=np.arange(-tstar[0],tstar[0],0.01)
axs[0].fill_between(px,t.pdf(px,df=df1),color='r')
axs[0].text(-3.9,0.03,"0.05"); axs[0].text(3.0,0.03,"0.05")

axs[1].plot(xv, t.pdf(xv,df=df1))
px=np.arange(-tstar[1],tstar[1],0.01)
axs[1].fill_between(px,t.pdf(px,df=df1),color='r')
axs[1].text(-4.0,0.03,"0.025"); axs[1].text(2.9,0.03,"0.025")

axs[2].plot(xv, t.pdf(xv,df=df1))
px=np.arange(-tstar[2],tstar[2],0.01)
axs[2].fill_between(px,t.pdf(px,df=df1),color='r')
axs[2].text(-4.1,0.03,"0.005"); axs[2].text(2.8,0.03,"0.005")
pd.set_option("display.precision", 4); print(df, '\n')
```

ConfLevel alpha tstar

0	0.90	0.10	1.8331
1	0.95	0.05	2.2622
2	0.99	0.01	3.2498



### 7.1.2 Confidence Interval for One-Sample Mean

In [Chapter 5](#) on proportions, a confidence interval formula for sample proportion was derived, which states that:

$$\text{point estimate} \pm z^* \times SE = \hat{p} \pm z^* \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

Analogously, the confidence interval for one-sample mean is constructed as:

$$\text{point estimate} \pm t_{df}^* \times SE = \bar{x} \pm t_{df}^* \frac{s}{\sqrt{n}}$$

where the margin of error is:

$$\text{Margin of error} = e = t_{df}^* \frac{s}{\sqrt{n}} \Rightarrow \quad (7.2)$$

$$CI = \bar{x} \pm e \quad (7.3)$$

#### Example

A random sample of 20 cars was tested for harmful emissions. The mean was 0.174 g/mi with a standard deviation of 0.016 g/mi (min 0.142, max 0.202). Construct 90%, 95%, and 99% confidence intervals. EPA requires that these emissions be no more than 0.165 g/mi. Is this requirement being met?

Since the observations are a simple random sample, independence holds. As the Python code below shows, the min and max observations are within 2.5 standard deviations of the mean, so no extreme outliers. Why 2.5? It is another rule of thumb. For a normal distribution, 95% of the data lies within about 2 standard deviations away from the mean. The t-distribution is somewhat more spread out, so 2.5 is used. Therefore, the normality assumption

seems reasonable in this case. The code below summarizes the steps needed to compute a confidence interval with a given set of significance levels.

```
[ ]: def OneMeanCI(xbar,s,n,ConfidenceLevels):
    import numpy as np; from scipy.stats import t; import pandas as pd
    import matplotlib.pyplot as plt
    print('One Mean Confidence Interval function')
    print('Sample mean xbar = ', xbar)
    print('Sample standard deviation = ', s)
    print('Sample size n = ', n)
    print('Confidence Levels',ConfidenceLevels,'\\n')

    ConfLevel = np.array(ConfidenceLevels) # confidence level
    alpha = 1-ConfLevel # error level corresponding to confidence level
    print('xbar +- 2.5*s bounds are {:.4f}, {:.4f}'.format(xbar-2.5*s,xbar+2.
    ↪5*s))
    SE = s/np.sqrt(n); # standard error for Confidence Interval (CI)
    print('Standard Error = SE = s/sqrt(n) = {:.3f}/sqrt({:d}) = {:.4f}'
    .format(s,n,SE))
    df1 = n-1; print('degree of freedom = df1 = {:d}-1 = {:d}\\n'.format(n,df1))
    tstar = t.ppf(1-alpha/2,df=df1); # t critical
    MarginErr = tstar*SE; # margin of error
    CIL = xbar - MarginErr; CIR = xbar + MarginErr;
    display = pd.DataFrame({'ConfLevel':ConfLevel,'tstar':tstar,'SE':SE, \
                           'MarginErr':MarginErr,'xbar':xbar,'CIL':CIL,'CIR':CIR});
    pd.set_option("display.precision", 4); print(display,'\\n')

    fig, axs = plt.subplots(1, 3, figsize=(10, 3), sharey=True)
    xv = np.arange(-4, 4, 0.01)
    axs[0].plot(xv, t.pdf(xv, df=df1))
    px=np.arange(-tstar[0],tstar[0],0.01)
    axs[0].fill_between(px,t.pdf(px,df=df1),color='r')
    axs[0].text(-3.9,0.03,"0.05"); axs[0].text(3.0,0.03,"0.05")

    axs[1].plot(xv, t.pdf(xv,df=df1))
    px=np.arange(-tstar[1],tstar[1],0.01)
    axs[1].fill_between(px,t.pdf(px,df=df1),color='r')
    axs[1].text(-4.0,0.03,"0.025"); axs[1].text(2.9,0.03,"0.025")

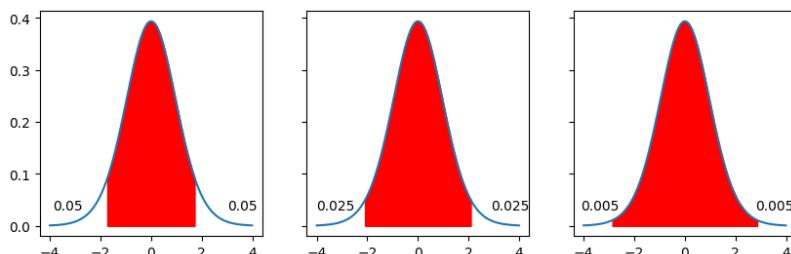
    axs[2].plot(xv, t.pdf(xv,df=df1))
    px=np.arange(-tstar[2],tstar[2],0.01)
    axs[2].fill_between(px,t.pdf(px,df=df1),color='r')
    axs[2].text(-4.1,0.03,"0.005"); axs[2].text(2.8,0.03,"0.005")
```

```
[ ]: OneMeanCI(xbar=0.174,s=0.016,n=20,ConfidenceLevels=[0.90,0.95,0.99])
```

```
One Mean Confidence Interval function
Sample mean xbar =  0.174
Sample standard deviation =  0.016
Sample size n =  20
Confidence Levels [0.9, 0.95, 0.99]

xbar +- 2.5*s bounds are 0.1340, 0.2140
Standard Error = SE = s/sqrt(n) = 0.016/sqrt(20) = 0.0036
degree of freedom = df1 = 20-1 = 19
```

	ConfLevel	tstar	SE	MarginErr	xbar	CIL	CIR
0	0.90	1.7291	0.0036	0.0062	0.174	0.1678	0.1802
1	0.95	2.0930	0.0036	0.0075	0.174	0.1665	0.1815
2	0.99	2.8609	0.0036	0.0102	0.174	0.1638	0.1842



The standard error is:

$$SE = \frac{0.016}{\sqrt{20}} = 0.00358$$

The margin of error is:

$$e = t^* \cdot SE = t^* \cdot 0.00358$$

Then the confidence interval for each level is obtained using:

$$\bar{x} \pm e = 0.174 \pm t^* \cdot 0.00358$$

In this way, we get all the standard-level confidence intervals in the Table above. In addition, the Figure above illustrates these intervals. We are 90% certain that the true mean population emission is between 0.168 and 0.18, 95% between 0.167 and 0.181, and 99% between 0.164 and 0.184. The confidence interval gets wider and wider as the level of confidence increases (same as for proportions). The most common 95% CI is completely above the EPA required value of 0.165, which implies that the population mean does not meet the pollution standards. Note, however, if we use the 99% level, the CI includes 0.165, so the confidence level is important. Still, to satisfy the EPA standard, we would want the confidence interval to be completely below 0.165.

### Example

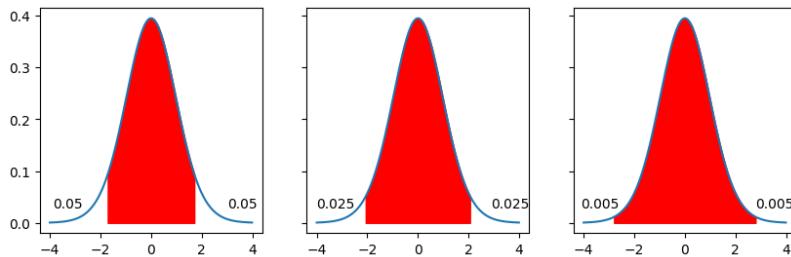
In a test of the effectiveness of garlic in lowering cholesterol levels, 25 randomly selected subjects regularly consumed raw garlic in fixed amounts. The changes in cholesterol levels have a mean of (-1.7) and a standard deviation of 4.2. The minimum value of this difference was -10.1 and the maximum was 7.9. Construct 90%, 95%, and 99% confidence intervals of the mean net change in LDL cholesterol after the treatment.

The observations are a simple random sample; therefore independence is reasonable. We can assume the data do not suggest any clear outliers, since all observations are within 2.5 standard deviations of the mean. Therefore, the normality assumption seems reasonable.

```
[ ]: OneMeanCI(xbar=-1.7,s=4.2,n=25,ConfidenceLevels=[0.90,0.95,0.99])
```

One Mean Confidence Interval function  
 Sample mean xbar = -1.7  
 Sample standard deviation = 4.2  
 Sample size n = 25  
 Confidence Levels [0.9, 0.95, 0.99]  
 xbar + -2.5\*s bounds are -12.2000, 8.8000  
 Standard Error = SE = s/sqrt(n) = 4.200/sqrt(25) = 0.8400  
 degree of freedom = df1 = 25-1 = 24

	ConfLevel	tstar	SE	MarginErr	xbar	CIL	CIR
0	0.90	1.7109	0.84	1.4371	-1.7	-3.1371	-0.2629
1	0.95	2.0639	0.84	1.7337	-1.7	-3.4337	0.0337
2	0.99	2.7969	0.84	2.3494	-1.7	-4.0494	0.6494



The standard error is:

$$SE = \frac{4.2}{\sqrt{25}} = 0.84$$

The margin of error is:

$$e = t^* \cdot SE = t^* \cdot 0.84$$

Then the confidence interval for each level is obtained using:

$$\bar{x} \pm e = -1.7 \pm t^* \cdot 0.84$$

We are 90% certain that the true mean LDL cholesterol change is between -3.137 and -0.263, 95% certain between -3.434 and 0.034, and 99% certain between 4.049 and 0.649. As the level of confidence increases, the interval gets wider and wider. The most common 95% CI contains 0 for the difference which implies that garlic is not effective. Note that if we use a 90% level, the CI is all negative which would imply an effective reduction in cholesterol, so once again, the choice of the confidence level can be quite important.

### 7.1.3 One-Sample Means t-tests

The procedure for one sample means hypothesis testing is built analogously to one sample proportions. The null  $H_0$  and alternative  $H_1$  hypotheses are set first. Then, assuming that  $H_0$  is true, the standardized test statistic  $t$

is computed, and the p-value area under appropriate t-distribution is found. The t-statistics formula is analogous to the z-statistic for proportions. For proportions:

$$H_0 : p = p_0 \quad SE = \sqrt{\frac{p_0(1 - p_0)}{n}}$$

$$z = \frac{(point\ estimate) - (null\ value)}{SE} = \frac{\hat{p} - p_0}{SE}$$

Analogously for one sample mean:

$$H_0 : \mu = \mu_0 \quad SE = \frac{s}{\sqrt{n}} \quad df = n - 1$$

$$t_{df} = \frac{(point\ estimate) - (null\ value)}{SE} = \frac{\bar{x} - \mu_0}{SE}$$

### Example

A consulting firm claims that its consultants earn on average \$150/hour. In a random sample of 24 consultants, the average rate was \$127/hour with standard deviation \$41/hour (min \$100, max \$180). Conduct a hypothesis test at 5% level.

**H0** :  $\mu = 150$  (assume true).

**H1** :  $\mu \neq 150$

The hypothesis test seeks to determine if the sample mean  $\bar{x} = 127$  is significantly different from the claimed null hypothesis  $\mu_0 = 150$ .

The code below implements the formulas above in one function. It also shows Cohen D's computation to be discussed later.

```
[ ]: def OneMeanTtest(xbar,s,n,mu0,ConfidenceLevels):
    import numpy as np; from scipy.stats import t; import pandas as pd
    import matplotlib.pyplot as plt
    print('One Mean T test detailed computation')
    print('Sample mean xbar = {:.4f}'.format(xbar))
    print('Sample standard deviation = {:.4f}'.format(s))
    print('Sample size n = {:d}'.format(n))
    print('Null hypothesis claimed mean mu0 = ', mu0)
    print('Confidence Levels',ConfidenceLevels,'\\n')

    print('H0: mu = mu0')
    print('H1: mu not mu0 or one-sided test\\n')

    print('xbar +- 2.5*s bounds are {:.4f}, {:.4f}'.format(xbar-2.5*s,xbar+2.
    ↪5*s))
    SE = s/np.sqrt(n); # standard error for Confidence Interval (CI)
    print('Standard Error = SE = s/sqrt(n) = {:.4f}/sqrt({:d}) = {:.4f}'.
    ↪format(s,n,SE))
    df1 = n-1; print('degree of freedom = df1 = {:d}-1 = {:d}\\n'.format(n,df1))
    meansdiff = xbar - mu0
    print("meansdiff = xbar - mu0 = {:.4f} - {:.4f} = {:.4f}"
```

```

        .format(xbar,mu0,meansdiff))
t1 = meansdiff/SE
print("test statistic = t1 = meansdiff/SE = {:.4f}/{:.4f} = {:.4f}"
      .format(meansdiff,SE,t1))
pval2 = 2*(1-t.cdf(np.abs(t1),df=df1));    # 2-sided p-value
print('P-values=2*(1-t.cdf(np.abs(t1),df=df1)), 2-sided and 1-sided =\n',u
→pval2, pval2/2,'\\n')

CohenD = meansdiff/s
print("Cohen D = meansdiff/s = {:.4f}/{:.4f} = {:.4f}\\n"
      .format(meansdiff,s,CohenD))

# P-value graphical illustration FYI-----
xv = np.arange(-4, 4, 0.001)
plt.plot(xv, t.pdf(xv,df=df1))
xL=np.arange(-4,-np.abs(t1),0.001)
plt.fill_between(xL,t.pdf(xL,df=df1),color='r')
xR=np.arange(np.abs(t1),4,0.001)
plt.fill_between(xR,t.pdf(xR,df=df1),color='b')
#-----

print('Confidence Interval (CI) approach')
ConfLevel = np.array(ConfidenceLevels) # confidence level
alpha = 1-ConfLevel # error level corresponding to confidence level
print('Critical t = tstar = t.ppf(1-alpha/2,df=df1)')
tstar = t.ppf(1-alpha/2,df=df1);      # tz critical
print('Margin of error = tstar*SE')
MarginErr = tstar*SE; # margin of error
print('CI = xbar +- MarginErr')
CIL = xbar - MarginErr; CIR = xbar + MarginErr;
display = pd.DataFrame({'ConfLevel':ConfLevel,'tstar':tstar,'SE':SE, \
                        'MarginErr':MarginErr,'xbar':xbar,'CIL':CIL,'CIR':CIR});
pd.set_option("display.precision", 4); print(display,'\\n')

```

[ ]: OneMeanTtest(xbar=127,s=41,n=24,mu0=150,ConfidenceLevels=[0.90,0.95,0.99])

```

One Mean T test detailed computation
Sample mean xbar = 127.0000
Sample standard deviation = 41.0000
Sample size n = 24
Null hypothesis claimed mean mu0 = 150
Confidence Levels [0.9, 0.95, 0.99]

```

```

H0: mu = mu0
H1: mu not mu0 or one-sided test

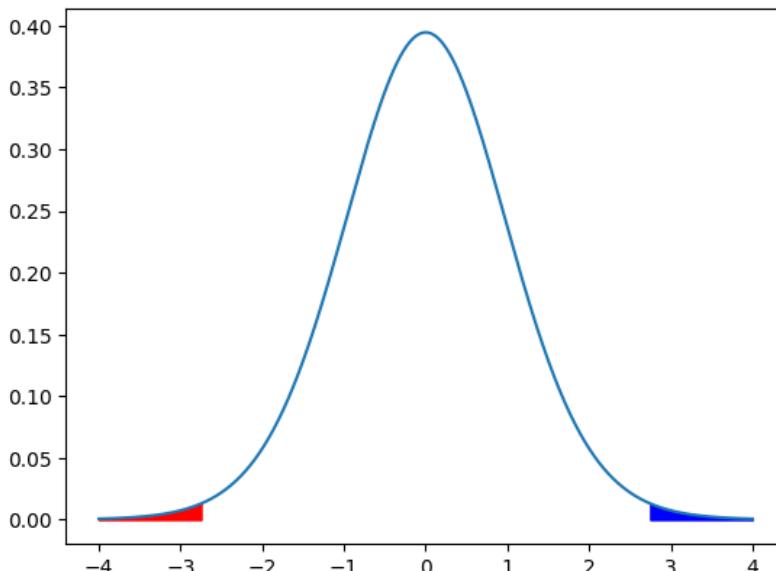
xbar +- 2.5*s bounds are 24.5000, 229.5000
Standard Error = SE = s/sqrt(n) = 41.0000/sqrt(24) = 8.3691
degree of freedom = df1 = 24-1 = 23

meansdiff = xbar - mu0 = 127.0000 - 150.0000 = -23.0000
test statistic = t1 = meansdiff/SE = -23.0000/8.3691 = -2.7482
P-values=2*(1-t.cdf(np.abs(t1),df=df1)), 2-sided and 1-sided =
0.011450910049694007 0.005725455024847004

```

```
Cohen D = meansdiff/s = -23.0000/41.0000 = -0.5610
```

```
Confidence Interval (CI) approach
Critical t = tstar = t.ppf(1-alpha/2,df=df1)
Margin of error = tstar*SE
CI = xbar +- MarginErr
    ConfLevel      tstar        SE  MarginErr   xbar       CIL       CIR
0          0.90  1.7139  8.3691    14.3435   127  112.6565  141.3435
1          0.95  2.0687  8.3691    17.3128   127  109.6872  144.3128
2          0.99  2.8073  8.3691    23.4948   127  103.5052  150.4948
```



The t-test assumptions:

1. Independence is assumed based on a simple random sample.
2. As the code shows, given min and max data values are within 2.5 standard deviations of the mean, so there are no extreme outliers, thus normality can be assumed, and a t-test can be applied. Note also as we mentioned before, the t-test procedure is robust, i.e., not heavily dependent on normality assumption in any case.

Thus, the sampling distribution of  $\frac{\bar{x} - \mu_0}{SE}$  (**null distribution**) is t-distribution, where  $SE = \frac{s}{\sqrt{n}} = \frac{41}{\sqrt{24}} = 8.369$ .

The **p-value** is the probability of seeing the observed sample mean  $\bar{x} = 127$  or something even more extreme under our t null distribution shown in the

Figure above.

$$t = \frac{\bar{x} - \mu_0}{SE} = \frac{127 - 150}{8.369} = -2.748$$

The alternative  $H1 : \mu \neq 150$  is two-sided; therefore both left and right tails must be accounted for. The areas under the tails add up to p-value =  $0.0114509 < 0.05 = \alpha$ , as shown in the Figure above. Thus, there is enough evidence to reject the initial assumption  $H0 : \mu = 150$ . Thus, the consultants do not make on average \$150/hour, it is significantly different. Note, however, that if we used the lower  $\alpha = 0.01$  level, we would *not* be able to reject the  $H0$ , so the choice of the significance level  $\alpha$  is very important.

Using the confidence interval approach, we are 95% confident (for a 5% significance error rate) that the true population mean is between 109.69 and 144.31. This interval does *not* contain the claimed  $H0 : \mu = 150$  confirming the rejection of  $H0$ . If the 99% level were used, the CI (103.51,150.49) would have contained 150 (just barely), resulting in failing to reject  $H0$ .

### Example

When 20 people used a popular diet for one year, their mean weight loss was 2.7 lb and the standard deviation was 6.1 lb (min -8, max 10). Use a 0.01 significance level to test the claim that the mean weight loss is not 0 lb. Based on these results, does the diet appear to be effective?

**H0 :  $\mu = 0$  (assume true)**

**H1 :  $\mu \neq 0$**

The hypothesis test seeks to determine if the sample mean  $\bar{x} = 2.7$  is significantly different from the claimed null hypothesis  $\mu_0 = 0$ .

```
[ ]: OneMeanTtest(xbar=2.7,s=6.1,n=20,mu0=0,ConfidenceLevels=[0.90,0.95,0.99])
```

```
One Mean T test detailed computation
Sample mean xbar = 2.7000
Sample standard deviation = 6.1000
Sample size n = 20
Null hypothesis claimed mean mu0 = 0
Confidence Levels [0.9, 0.95, 0.99]

H0: mu = mu0
H1: mu not mu0 or one-sided test

xbar +- 2.5*s bounds are -12.5500, 17.9500
Standard Error = SE = s/sqrt(n) = 6.1000/sqrt(20) = 1.3640
degree of freedom = df1 = 20-1 = 19

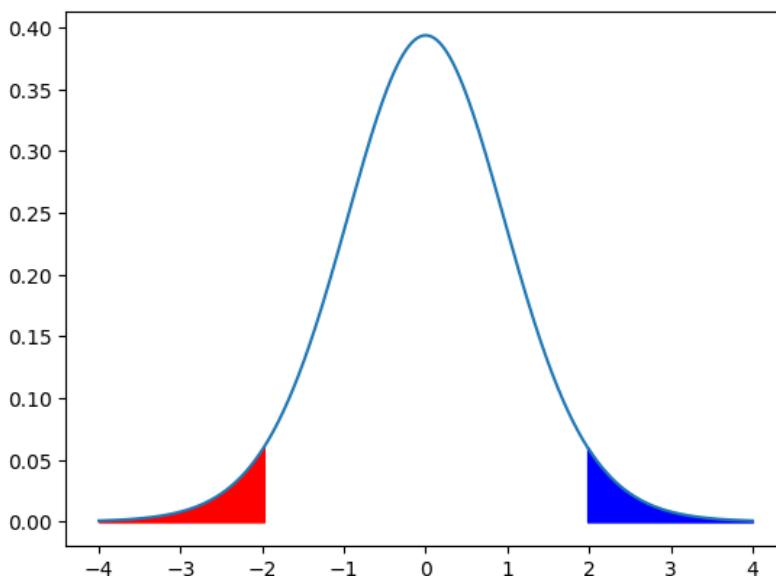
meansdiff = xbar - mu0 = 2.7000 - 0.0000 = 2.7000
test statistic = t1 = meansdiff/SE = 2.7000/1.3640 = 1.9795
P-values=2*(1-t.cdf(np.abs(t1),df=df1)), 2-sided and 1-sided =
0.06243778951725898 0.03121889475862949

Cohen D = meansdiff/s = 2.7000/6.1000 = 0.4426
```

```

Confidence Interval (CI) approach
Critical t = tstar = t.ppf(1-alpha/2,df=df1)
Margin of error = tstar*SE
CI = xbar +- MarginErr
    ConfLevel   tstar      SE  MarginErr   xbar      CIL      CIR
0          0.90  1.7291  1.364     2.3585   2.7  0.3415  5.0585
1          0.95  2.0930  1.364     2.8549   2.7 -0.1549  5.5549
2          0.99  2.8609  1.364     3.9023   2.7 -1.2023  6.6023

```



The assumptions for the t-test hold the same way as in the previous problem. A simple random sample ensures independence. The min/max of the data are within 2.5 standard deviations away from the mean, so no extreme outliers.

Therefore, the sampling distribution of  $\frac{\bar{x} - \mu_0}{SE}$  (**null distribution**) is t-distribution with  $SE = \frac{s}{\sqrt{n}} = \frac{6.1}{\sqrt{20}} = 1.364$ . The t-test procedure is generally robust to small departures from normality in any case.

The **p-value** is the probability of seeing the observed sample mean  $\bar{x} = 2.7$  or something even more extreme under our t null distribution shown in the Figure above.

$$t = \frac{\bar{x} - \mu_0}{SE} = \frac{2.7 - 0}{1.364} = 1.979$$

The alternative is two-sided  $H1 : \mu \neq 0$ , so both tails must be accounted for. The areas under the tails add up to p-value = 0.0624378 > 0.01 =  $\alpha$ , as shown in the Figure above. Thus, we conclude that there is *not* enough evidence to

reject the initial assumption  $H_0$ . Thus, the diet does not seem to be effective. The conclusion would also stand at a most common  $\alpha = 0.05$  (5%) level, but not at 10%.

Using the confidence interval approach, we are 99% confident (for 1% significance error rate) that the true population mean is between -1.2 and 6.6. This interval contains the claimed  $H_0 : \mu = 0$ , which confirms our conclusion *not* to reject  $H_0$ . The 95% CI also contains 0, but not 90% CI, which confirms our previous observations.

So far we have considered only a two-sided test. Let's investigate some one-sided tests adjusting to lower levels for confidence intervals similar to proportion tests.

### Example

A company claims to increase students' Math scores to **at least** 70 on the Regents exam. The consumer protection agency is testing their claim. A sample of 20 students is randomly selected and the mean is 65 with the standard deviation of 9. Test the claim at a 5% level.

Because of **at least**, the alternative is one-sided. Also, because the consumer protection agency is testing their claim, they are trying to refute it, so  $H_1$  is opposite to the company's claim.

**H0** :  $\mu = 70$  (**assume true**).

**H1** :  $\mu < 70$

The hypothesis test seeks to determine if the sample mean  $\bar{x} = 65$  is significantly **lower** than the claimed null hypothesis  $H_0$ .

```
[ ]: OneMeanTtest(xbar=65,s=9,n=20,mu0=70,ConfidenceLevels=[0.90,0.95,0.99])
```

```
One Mean T test detailed computation
Sample mean xbar = 65.0000
Sample standard deviation = 9.0000
Sample size n = 20
Null hypothesis claimed mean mu0 = 70
Confidence Levels [0.9, 0.95, 0.99]

H0: mu = mu0
H1: mu not mu0 or one-sided test

xbar +- 2.5*s bounds are 42.5000, 87.5000
Standard Error = SE = s/sqrt(n) = 9.0000/sqrt(20) = 2.0125
degree of freedom = df1 = 20-1 = 19

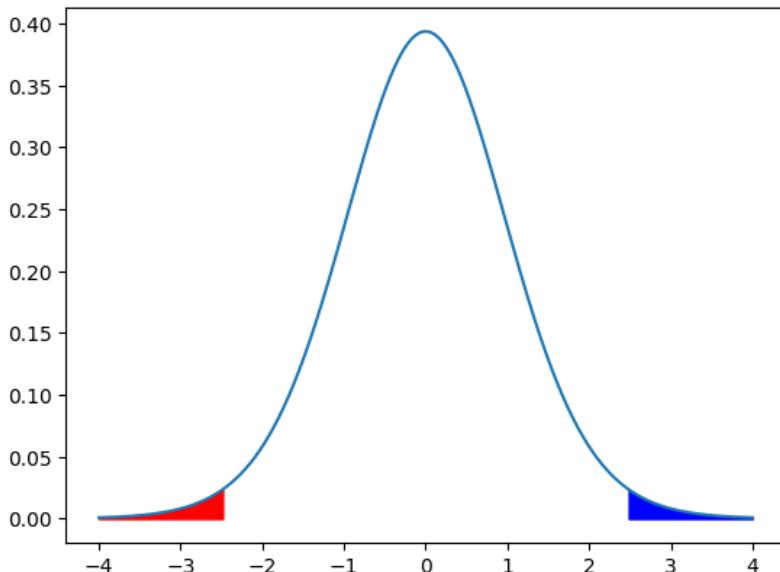
meansdiff = xbar - mu0 = 65.0000 - 70.0000 = -5.0000
test statistic = t1 = meansdiff/SE = -5.0000/2.0125 = -2.4845
P-values=2*(1-t.cdf(np.abs(t1),df=df1)), 2-sided and 1-sided =
0.02246083562915424 0.01123041781457712

Cohen D = meansdiff/s = -5.0000/9.0000 = -0.5556
```

```

Confidence Interval (CI) approach
Critical t = tstar = t.ppf(1-alpha/2,df=df1)
Margin of error = tstar*SE
CI = xbar +- MarginErr
    ConfLevel      tstar        SE  MarginErr   xbar       CIL       CIR
0          0.90  1.7291  2.0125     3.4798   65  61.5202  68.4798
1          0.95  2.0930  2.0125     4.2121   65  60.7879  69.2121
2          0.99  2.8609  2.0125     5.7575   65  59.2425  70.7575

```



The assumptions for the t-test hold. As always, a simple random sample is assumed which ensures independence. The min/max of the data are not given in this case, but test scores tend to be normally distributed.

Therefore, the sampling distribution of  $\frac{\bar{x} - \mu_0}{SE}$  (**null distribution**) is t-distribution with  $SE = \frac{s}{\sqrt{n}} = \frac{9}{\sqrt{20}} = 2.012$ . The t-test procedure is generally robust to small departures from normality in any case.

The **p-value** is the probability of seeing the observed sample mean  $\bar{x} = 70$  or something even more extreme under our null distribution is shown in the Figure.

$$t = \frac{\bar{x} - \mu_0}{SE} = \frac{65 - 70}{2.012} = -2.485$$

The alternative is one-sided to the left  $H_1 : \mu < 70$ , so only the left tail contributes. The area under this tail is the p-value =  $0.0112304 < 0.05 = \alpha$ , as shown in the Figure above. Therefore, there is enough evidence to reject the initial assumption  $H_0 : \mu = 65$ .

Once again, for the one-sided test, we have to use  $1 - 2\alpha = 1 - 2 \cdot 0.05 = 0.90$  confidence level. Therefore, we are 90% confident (10% significance error) that the true population mean is between 61.52 and 68.48, which does *not* contain the claimed  $H_0 : \mu = 70$  confirming our conclusion to reject  $H_0$ .

In an argument similar to the case of the proportion, any sample mean  $\bar{x}$  above 70 is already invalidating  $H_1 : \mu < 70$ , and there is no reason to continue to test such an alternative. The consumer protection agency has no case against the company. Moreover, even if the sample mean  $\bar{x}$  is below 70, it must be small enough to be significantly below 70 thus rejecting the  $H_0$ . In the case above it was, but the code below shows the same calculation for  $\bar{x} = 68$ , which is not low enough resulting in a p-value above 0.05 and failure to reject  $H_0$ , as shown in the Figure below.

```
[ ]: OneMeanTtest(xbar=68,s=9,n=20,mu0=70,ConfidenceLevels=[0.90,0.95,0.99])
```

```
One Mean T test detailed computation
Sample mean xbar = 68.0000
Sample standard deviation = 9.0000
Sample size n = 20
Null hypothesis claimed mean mu0 = 70
Confidence Levels [0.9, 0.95, 0.99]

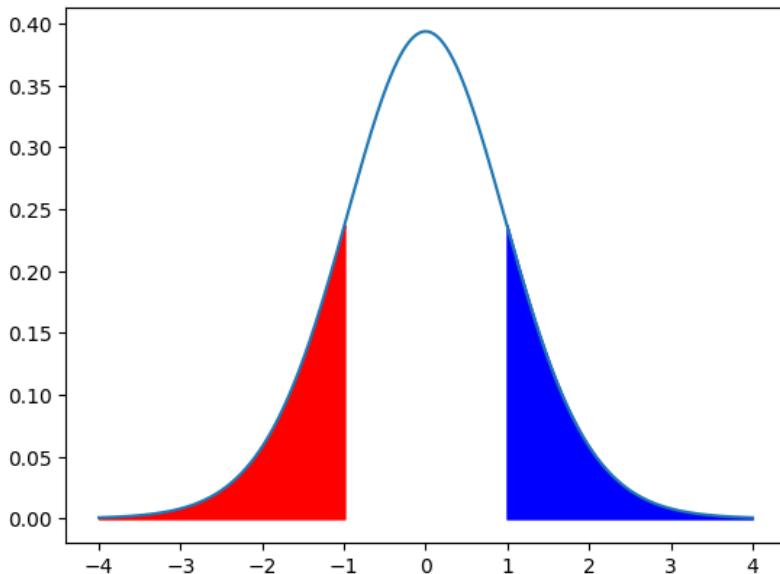
H0: mu = mu0
H1: mu not mu0 or one-sided test

xbar +- 2.5*s bounds are 45.5000, 90.5000
Standard Error = SE = s/sqrt(n) = 9.0000/sqrt(20) = 2.0125
degree of freedom = df1 = 20-1 = 19

meansdiff = xbar - mu0 = 68.0000 - 70.0000 = -2.0000
test statistic = t1 = meansdiff/SE = -2.0000/2.0125 = -0.9938
P-values=2*(1-t.cdf(np.abs(t1),df=df1)), 2-sided and 1-sided =
0.3328052573100222 0.1664026286550111

Cohen D = meansdiff/s = -2.0000/9.0000 = -0.2222

Confidence Interval (CI) approach
Critical t = tstar = t.ppf(1-alpha/2,df=df1)
Margin of error = tstar*SE
CI = xbar +- MarginErr
    ConfLevel      tstar        SE   MarginErr     xbar       CIL       CIR
0          0.90    1.7291    2.0125      3.4798     68  64.5202  71.4798
1          0.95    2.0930    2.0125      4.2121     68  63.7879  72.2121
2          0.99    2.8609    2.0125      5.7575     68  62.2425  73.7575
```



### Example

A water bottle company claims that there is at most 1 ppm (part per million) of arsenic in their bottled water. A consumer agency is set to test this claim. A random sample of 25 bottles was collected, and the mean concentration of arsenic was 1.09 with the standard deviation of 0.24 (min 0.9, max 1.37). Test the claim at 5%.

Because of the **at least** claim, the alternative is one-sided. Also, because the consumer protection agency is testing their claim, they are trying to refute it, so the alternative is opposite to the company's claim.

**H0** :  $\mu = 1$  (**assume true**).

**H1** :  $\mu > 1$

The hypothesis test seeks to determine if the sample mean  $\bar{x} = 1.09$  is significantly higher than the claimed null hypothesis H0.

```
[ ]: OneMeanTtest(xbar=1.09,s=0.24,n=25,mu0=1,ConfidenceLevels=[0.90,0.95,0.99])
```

```
One Mean T test detailed computation
Sample mean xbar = 1.0900
Sample standard deviation = 0.2400
Sample size n = 25
Null hypothesis claimed mean mu0 = 1
Confidence Levels [0.9, 0.95, 0.99]
```

```
H0: mu = mu0
```

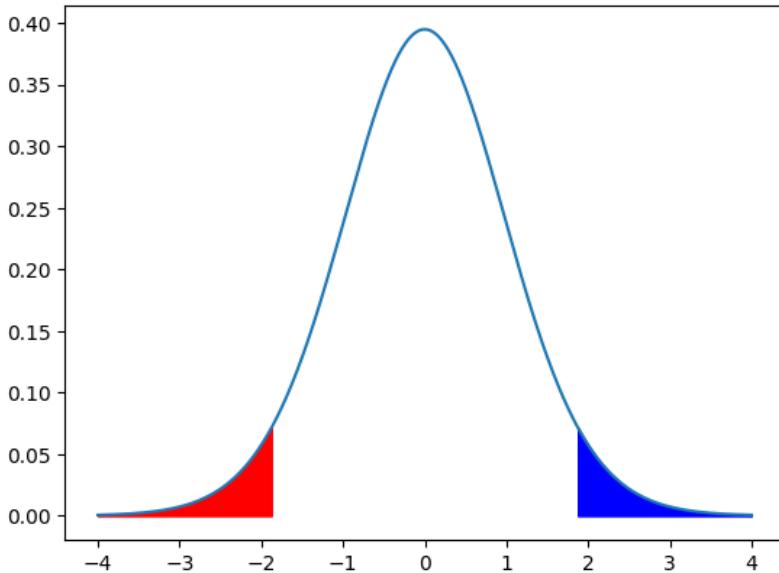
```
H1: mu not mu0 or one-sided test

xbar +- 2.5*s bounds are 0.4900, 1.6900
Standard Error = SE = s/sqrt(n) = 0.2400/sqrt(25) = 0.0480
degree of freedom = df1 = 25-1 = 24

meansdiff = xbar - mu0 = 1.0900 - 1.0000 = 0.0900
test statistic = t1 = meansdiff/SE = 0.0900/0.0480 = 1.8750
P-values=2*(1-t.cdf(np.abs(t1),df=df1)), 2-sided and 1-sided =
0.0730126385883189 0.036506319294415945

Cohen D = meansdiff/s = 0.0900/0.2400 = 0.3750

Confidence Interval (CI) approach
Critical t = tstar = t.ppf(1-alpha/2,df=df1)
Margin of error = tstar*SE
CI = xbar +- MarginErr
      ConfLevel    tstar      SE  MarginErr    xbar      CIL      CIR
0        0.90   1.7109  0.048      0.0821   1.09   1.0079   1.1721
1        0.95   2.0639  0.048      0.0991   1.09   0.9909   1.1891
2        0.99   2.7969  0.048      0.1343   1.09   0.9557   1.2243
```



The assumptions for the t-test hold - independent observations and min/max of the data are within 2.5 standard deviations away from the mean, so no extreme outliers.

Therefore, the sampling distribution of  $\frac{\bar{x} - \mu_0}{SE}$  (**null distribution**) is a t-distribution with  $SE = \frac{s}{\sqrt{n}} = \frac{0.24}{\sqrt{25}} = 0.048$ .

The **p-value** is the probability of seeing the observed sample mean  $\bar{x}$  or something even more extreme under our t null distribution shown in the Figure.

$$t = \frac{\bar{x} - \mu_0}{SE} = \frac{1.09 - 1}{0.048} = \frac{0.09}{0.048} = 1.875$$

The alternative is one-sided to the right  $H1 : \mu > 1$ , so only the right tail is included, which gives p-value = 0.0365063 < 0.05 =  $\alpha$ , as shown in the Figure above. Therefore, there is enough evidence to reject the initial assumption  $H0 : \mu = 1$ . Thus, the mean concentration of arsenic in the company's water bottles is significantly higher than 1. Note that if we used a two-sided test in this case, the p-value = 0.0730126 > 0.05 =  $\alpha$ , which would lead to failing to reject  $H0$ . Once again, it is important to follow the given hypothesis direction.

Yet again, for the one-sided test, the confidence level is adjusted to  $1 - 2\alpha = 1 - 2 \cdot 0.05 = 0.90$ . Thus, we are 90% confident (10% significance error) that the true population mean is between 1.01 and 1.17, which does NOT contain the claimed  $H0 : \mu = 1$ . This confirms our conclusion to reject  $H0$ . If we wrongly used 95% confidence interval of 0.99 to 1.19, the conclusion would have been the opposite.

### 7.1.4 One-Sample Means Tests on Data Files

In this section, we will work on actual data, not just data summaries (mean, standard deviations, etc.) as in previous examples.

Before we start with examples, let's remark that even though the t-test is quite robust to departures from normality, in more extreme cases, a non-parametric **Wilcoxon** signed rank test is an essential alternative. It uses data ranks rather than actual values. Ranks are resistant to outliers and skewness, making this test a better choice for small, skewed samples. It can be used, for example, to determine whether the center location of the sample (mean or median) is equal to the prescribed value. Generally, non-parametric procedures have less power to reject  $H0$  when it is false than the corresponding parametric procedure if normality assumption holds. When data is available, we will employ both t-test and Wilcoxon tests and compare the results.

#### Example

The greenhouse gas  $CO_2$  emissions is a very serious issue for the environment. Assume we are given that last year the  $CO_2$  emission in a country was 3.26 metric tons per capita. The data below gives a random sample of  $CO_2$  emissions this year. At the 1% level, is there enough evidence to show that the mean  $CO_2$  emissions level is lower this year?

The problem asks for lower emissions; therefore the alternative is one-sided:

**H0** :  $\mu = 3.26$  (**assume true**).

**H1** :  $\mu < 3.26$

The hypothesis test seeks to determine if the sample mean  $\bar{x} = 3.092$  is significantly lower than the claimed null hypothesis H0.

This time, the actual data are given, so we first consider the histogram and boxplot of the data in the code below as well as the stats package `texttstats.ttest_1samp()` function to perform the t-test. In addition, `scipy.stats.wilcoxon` function is used to run the Wilcoxon test on the data. Note that it takes the difference `mydata.x - 3.26` as the input. In both tests, we have to remember to specify `alternative = "less"` to compute the left-tail test; otherwise a two-sided alternative is done automatically. A step-by-step code is used as well.

```
[ ]: import pandas as pd; import numpy as np;
import matplotlib.pyplot as plt; import seaborn as sns
mydata = pd.DataFrame({'x':[3.2, 3.4, 3.2, 3.2, 2.8, 3.3, 2.9, 3.2,
                           2.6, 3.2, 2.9, 3.0, 3.1, 3.3]})

xbar1 = np.mean(mydata.x);
s1 = np.std(mydata.x,ddof=1); n1 = len(mydata.x);

fig, axs = plt.subplots(1, 3, figsize=(10, 3))
sns.histplot(data=mydata,x="x", ax=axs[0])
sns.boxplot(data=mydata,y="x", ax=axs[1])

from scipy import stats
print('Python Ttest_1sampResult built-in function')
print(stats.ttest_1samp(mydata.x, popmean=3.26, alternative='less'), '\n')

from scipy.stats import wilcoxon # Built-in function approach
print('Python non-parametric Wilcoxon built-in function')
print(wilcoxon(mydata.x-3.26, alternative='less'), '\n')# differences

OneMeanTtest(xbar=xbar1,s=s1,n=n1,mu0=3.26,
ConfidenceLevels=[0.90,0.95,0.98, 0.99])
```

Python Ttest\_1sampResult built-in function  
 TtestResult(statistic=-2.7983729360306717, pvalue=0.007538301009340417, df=13)

Python non-parametric Wilcoxon built-in function  
 WilcoxonResult(statistic=11.0, pvalue=0.00335693359375)

One Mean T test detailed computation  
 Sample mean xbar = 3.0929  
 Sample standard deviation = 0.2235  
 Sample size n = 14  
 Null hypothesis claimed mean mu0 = 3.26  
 Confidence Levels [0.9, 0.95, 0.98, 0.99]

H0: mu = mu0  
 H1: mu not mu0 or one-sided test

xbar +- 2.5\*s bounds are 2.5341, 3.6516

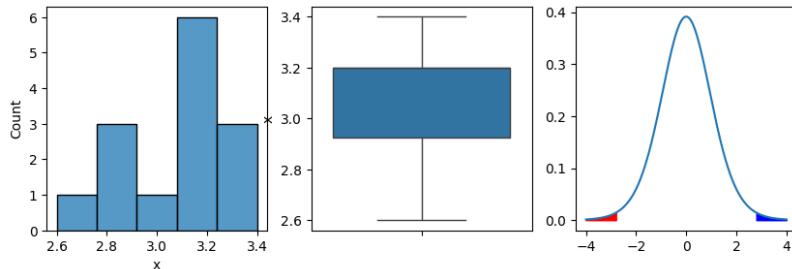
```
Standard Error = SE = s/sqrt(n) = 0.2235/sqrt(14) = 0.0597
degree of freedom = df1 = 14-1 = 13
```

```
meansdiff = xbar - mu0 = 3.0929 - 3.2600 = -0.1671
test statistic = t1 = meansdiff/SE = -0.1671/0.0597 = -2.7984
P-values=2*(1-t.cdf(np.abs(t1),df=df1)), 2-sided and 1-sided =
0.015076602018680774 0.007538301009340387
```

```
Cohen D = meansdiff/s = -0.1671/0.2235 = -0.7479
```

```
Confidence Interval (CI) approach
Critical t = tstar = t.ppf(1-alpha/2,df=df1)
Margin of error = tstar*SE
```

	ConfLevel	tstar	SE	MarginErr	xbar	CIL	CIR
0	0.90	1.7709	0.0597	0.1058	3.0929	2.9871	3.1986
1	0.95	2.1604	0.0597	0.1290	3.0929	2.9638	3.2219
2	0.98	2.6503	0.0597	0.1583	3.0929	2.9346	3.2512
3	0.99	3.0123	0.0597	0.1799	3.0929	2.9129	3.2728



As always, a simple random sample is assumed, which ensures independence. The histogram and boxplot above show no extreme outliers, so assumptions of normality are not violated. Thus, the assumptions for the t-test hold.

Therefore, the sampling distribution of  $\frac{\bar{x} - \mu_0}{SE}$  (**null distribution**) is t-distribution with  $SE = \frac{s}{\sqrt{n}} = \frac{0.223}{\sqrt{14}} = 0.06$ .

The **p-value** is the probability of seeing the observed sample mean  $\bar{x} = 3.093$  or something even more extreme under our t null distribution shown in the Figure.

$$t = \frac{\bar{x} - \mu_0}{SE} = \frac{3.093 - 3.26}{0.06} = -2.798$$

The alternative is one-sided to the left  $H1 : \mu < 3.26$ , so only the left tail area contributes to the p-value  $= 0.0075383 < 0.01 = \alpha$ . Therefore, there is enough evidence to reject  $H0: \mu = 3.26$ . Note that if we had used a two-sided test in this case, p-value  $= 0.0150766 > 0.01 = \alpha$ , so we would have failed to reject the  $H0$ . Note also that the non-parametric Wilcoxon test leads to the same conclusions.

Once again, for the one-sided test, we have to use  $1 - 2\alpha = 1 - 2 \cdot 0.01 = 0.98$  confidence level. Note that we added it to the list of confidence levels in the code above, as it is not the usual level. Thus, we are 98% confident (2% significance error) that the true population mean is between 2.93 and 3.25, which does *not* contain the claimed  $H_0 : \mu = 3.26$ . This confirms our conclusion to reject  $H_0$ . If we wrongly used a 99% confidence interval (2.91, 3.27), the opposite conclusion would have resulted.

### Example

A new experimental reading technique was tried on 16 elementary school children. Their reading scores are given below. Is there enough evidence to show that the mean score is **higher** than the State requirement of 65? Test at the 10% level.

Note that we are checking if the mean score is **higher** than the State requirement, so the alternative is one-sided. Also, once again we are *not* trying to refute the claim, just checking if it is higher, so the alternative is greater than 65.

**H0** :  $\mu = 65$  (**assume true**)

**H1** :  $\mu > 65$

The hypothesis test seeks to determine if the sample mean  $\bar{x} = 72.938$  is significantly higher than the claimed null hypothesis H0.

The actual data are given once again, so a histogram and a boxplot are used to check for the extreme skewness and/or outliers. Note again that the `alternative="greater"` must be used to ensure the correct one-sided test.

```
[ ]: import pandas as pd; import numpy as np;
import matplotlib.pyplot as plt; import seaborn as sns
mydata = pd.DataFrame({'x':
    [72,53,64,100,45,56,75,94,59,43,104,75,81,63,106,77]})
xbar1 = np.mean(mydata.x); # sample mean
s1 = np.std(mydata.x,ddof=1); # sample standard deviation
n1 = len(mydata.x); # sample size

fig, axs = plt.subplots(1, 3, figsize=(10, 3))
sns.histplot(data=mydata,x="x", ax=axs[0])
sns.boxplot(data=mydata,y="x", ax=axs[1])

from scipy import stats
print(stats.ttest_1samp(mydata.x, popmean=65, alternative='greater'),'\n')

from scipy.stats import wilcoxon # Built-in function approach
print('Python non-parametric Wilcoxon built-in function')
print(wilcoxon(mydata.x-65, alternative='greater'),'\n') # differences!

OneMeanTtest(xbar=xbar1,s=s1,n=n1,mu0=65,ConfidenceLevels=[0.80,0.90,0.95,0.
    ↪99])
```

```
TtestResult(statistic=1.5797600352187966, pvalue=0.06750718602179727, df=15)

Python non-parametric Wilcoxon built-in function
WilcoxonResult(statistic=93.5, pvalue=0.105712890625)

One Mean T test detailed computation
Sample mean xbar = 72.9375
Sample standard deviation = 20.0980
Sample size n = 16
Null hypothesis claimed mean mu0 = 65
Confidence Levels [0.8, 0.9, 0.95, 0.99]

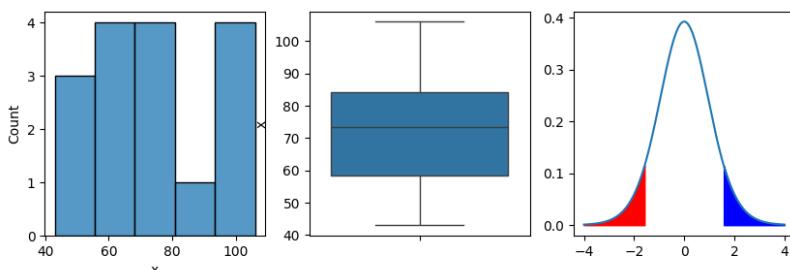
H0: mu = mu0
H1: mu not mu0 or one-sided test

xbar + 2.5*s bounds are 22.6925, 123.1825
Standard Error = SE = s/sqrt(n) = 20.0980/sqrt(16) = 5.0245
degree of freedom = df1 = 16-1 = 15

meansdiff = xbar - mu0 = 72.9375 - 65.0000 = 7.9375
test statistic = t1 = meansdiff/SE = 7.9375/5.0245 = 1.5798
P-values=2*(1-t.cdf(np.abs(t1),df=df1)), 2-sided and 1-sided =
0.13501437204359457 0.06750718602179728

Cohen D = meansdiff/s = 7.9375/20.0980 = 0.3949

Confidence Interval (CI) approach
Critical t = tstar = t.ppf(1-alpha/2,df=df1)
Margin of error = tstar*SE
CI = xbar +- MarginErr
    ConfLevel      tstar      SE  MarginErr      xbar       CIL       CIR
0        0.80    1.3406  5.0245     6.7359    72.9375   66.2016   79.6734
1        0.90    1.7531  5.0245     8.8082    72.9375   64.1293   81.7457
2        0.95    2.1314  5.0245    10.7095    72.9375   62.2280   83.6470
3        0.99    2.9467  5.0245    14.8058    72.9375   58.1317   87.7433
```



As always, a simple random sample is assumed, which ensures independence. The histogram and boxplot above show no extreme outliers, so assumptions of normality are not violated. Thus, the assumptions for the t-test hold (the

t-test is robust against small departures from normality in any case).

Therefore, according to the CLT, the sampling distribution of  $\frac{\bar{x} - \mu_0}{SE}$  (**null distribution**) is t-distribution, where  $SE = \frac{s}{\sqrt{n}} = \frac{20.098}{\sqrt{16}} = 5.024$ .

The **p-value** is the probability to see the observed sample mean  $\bar{x} = 72.938$  or something even more extreme under our t null distribution shown in the Figure above.

$$t = \frac{\bar{x} - \mu_0}{SE} = \frac{72.938 - 65}{5.024} = 1.58$$

The alternative is one-sided to the right  $H1 : \mu > 65$ , so only the right tail contributes. The area under this tail is the p-value  $= 0.0675072 < 0.10 = \alpha$ , so we conclude that there is enough evidence to reject the initial assumption  $H0 : \mu = 65$  at 10%. At the more usual 5% level, we would have failed to reject  $H0$ . Also, if we had used a two-sided test, p-value  $= 0.1350144 > 0.10 = \alpha$ , we would have failed to reject the  $H0$  as well. Note also that the non-parametric Wilcoxon test leads to the opposite conclusion - its p-value is slightly above  $\alpha = 0.10$ . Which one to follow here? The boxplot is rather symmetric; therefore t-test seems to be reliable. As we mentioned before, a non-parametric test tends to be less powerful in rejecting the null hypothesis.

Yet again, for the one-sided test, we have to use  $1 - 2\alpha = 1 - 2 \cdot 0.10 = 0.80$  confidence level since CI is two-sided. We have to actually add this level to the code. Thus, we are 80% confident (20% significance error) that the true population mean is between 66.2 and 79.67, which does *not* contain the claimed  $H0 : \mu = 65$ . This confirms our conclusion to reject  $H0$ . If we wrongly used 90% confidence interval 64.13 and 81.75, the conclusion would have been opposite.

### Example

Consider again the `HELPPrct.csv` file and concentrate on the depression `cesd` score. Let's say, for the general population, the mean score is around 30. Investigate if it is different for this sample of substance abusers. Use 5% level of significance.

**H0** :  $\mu = 30$  (**assume true**)

**H1** :  $\mu \neq 30$

The hypothesis test seeks to determine if the sample mean  $\bar{x} = 32.848$  is significantly different from the claimed null hypothesis  $H0$ .

The code is a bit different - `HELPPrct.csv` data file must be loaded and we reference variable `cesd`.

```
[ ]: import pandas as pd; import numpy as np;
import matplotlib.pyplot as plt; import seaborn as sns

url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPPrct.csv"
mydata = pd.read_csv(url) # save as mydata file
```

```

xbar1 = np.mean(mydata.cesd);      # sample mean
s1 = np.std(mydata.cesd,ddof=1);  # sample standard deviation
n1 = len(mydata.cesd);           # sample size

fig, axs = plt.subplots(1, 3, figsize=(10, 3))
sns.histplot(data=mydata,x="cesd", ax=axs[0])
sns.boxplot(data=mydata,y="cesd", ax=axs[1])

from scipy import stats
print('Python Ttest_1sampResult built-in function')
print(stats.ttest_1samp(mydata.cesd, popmean=30),'\n')

from scipy.stats import wilcoxon   # Built-in function approach
print('Python non-parametric Wilcoxon built-in function')
print(wilcoxon(mydata.cesd-30),'\n') # note differences!

OneMeanTtest(xbar=xbar1,s=s1,n=n1,mu0=30,ConfidenceLevels=[0.90,0.95,0.99])

```

```

Python Ttest_1sampResult built-in function
TtestResult(statistic=4.843156493027925, pvalue=1.7582140620493572e-06, df=452)

Python non-parametric Wilcoxon built-in function
WilcoxonResult(statistic=35171.5, pvalue=8.075719421734236e-07)

One Mean T test detailed computation
Sample mean xbar = 32.8477
Sample standard deviation = 12.5145
Sample size n = 453
Null hypothesis claimed mean mu0 = 30
Confidence Levels [0.9, 0.95, 0.99]

H0: mu = mu0
H1: mu not mu0 or one-sided test

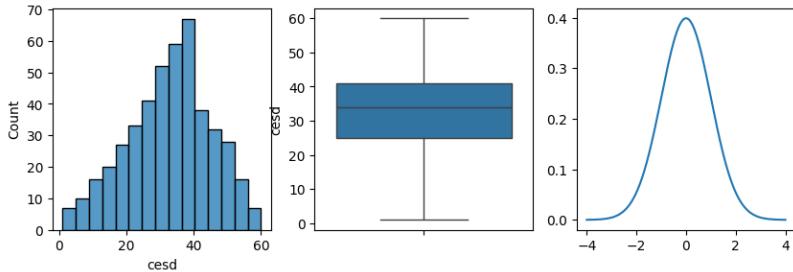
xbar +- 2.5*s bounds are 1.5615, 64.1338
Standard Error = SE = s/sqrt(n) = 12.5145/sqrt(453) = 0.5880
degree of freedom = df1 = 453-1 = 452

meansdiff = xbar - mu0 = 32.8477 - 30.0000 = 2.8477
test statistic = t1 = meansdiff/SE = 2.8477/0.5880 = 4.8432
P-values=2*(1-t.cdf(np.abs(t1),df=df1)), 2-sided and 1-sided =
1.758214062119734e-06 8.79107031059867e-07

Cohen D = meansdiff/s = 2.8477/12.5145 = 0.2276

Confidence Interval (CI) approach
Critical t = tstar = t.ppf(1-alpha/2,df=df1)
Margin of error = tstar*SE
CI = xbar +- MarginErr
    ConfLevel      tstar      SE  MarginErr      xbar       CIL       CIR
0        0.90    1.6482  0.588      0.9691    32.8477  31.8786  33.8168
1        0.95    1.9652  0.588      1.1555    32.8477  31.6922  34.0032
2        0.99    2.5867  0.588      1.5210    32.8477  31.3267  34.3686

```



We can assume these patients are a simple random sample of all such patients, so they are independent. The histogram and boxplot above are almost perfectly bell-shaped (normally distributed). Thus, the assumptions for the t-test hold well in this case (the t-test is robust against such departures anyway).

Therefore, according to the CLT, the sampling distribution of  $t = \frac{\bar{x} - \mu_0}{SE}$  (**null distribution**) is a t-distribution, where  $SE = \frac{s}{\sqrt{n}} = \frac{12.514}{\sqrt{453}} = 0.588$ .

The **p-value** is the probability of seeing the observed sample mean  $\bar{x} = 32.848$  or something even more extreme under our t null distribution shown in the Figure above.

$$t = \frac{\bar{x} - \mu_0}{SE} = \frac{32.848 - 30}{0.588} = 4.843$$

We are back to two-sided H1, so both left and right tails must be accounted for. These tails in the t-distribution are shown in the Figure above, but they are too small to see visually. The areas under the tails add up to p-value  $= 1.758 \cdot 10^{-6} < 0.05 = \alpha$ , so we conclude that there is enough evidence to reject the initial assumption H0. Thus, the mean population depression score for these substance abusers is not 30. Note also that the non-parametric Wilcoxon test leads to the same conclusions.

Using the confidence interval approach, we are 95% confident (for 5% significance error rate) that the true population mean is between 31.69 and 34, which does *not* contain the claimed  $H0 : \mu = 30$ . This confirms our conclusion to reject H0.

### Example

Consider the `HELPrc.csv` file again and concentrate on the mental health `mcs` score. From previous studies of substance abusers, the average `mcs` score is around 32. Investigate if it is different for this sample of substance abusers. Conduct a hypothesis test at a 5% level.

**H0** :  $\mu = 32$  (**assume true**)

**H1** :  $\mu \neq 32$

The hypothesis test seeks to determine if the sample mean  $\bar{x} = 31.677$  is significantly different from the claimed null hypothesis  $H_0$ .

```
[ ]: import pandas as pd; import numpy as np;
import matplotlib.pyplot as plt; import seaborn as sns

url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrc.csv"
mydata = pd.read_csv(url) # save as mydata file

xbar1 = np.mean(mydata.mcs); # sample mean
s1 = np.std(mydata.mcs,ddof=1); # sample standard deviation
n1 = len(mydata.mcs); # sample size

fig, axs = plt.subplots(1, 3, figsize=(10, 3))
sns.histplot(data=mydata,x="mcs", ax=axs[0])
sns.boxplot(data=mydata,y="mcs", ax=axs[1])

from scipy import stats
print('Python Ttest_1sampResult built in function')
print(stats.ttest_1samp(mydata.mcs, popmean=32),'\n')

from scipy.stats import wilcoxon # Built-in function approach
print('Python non-parametric Wilcoxon built-in function')
print(wilcoxon(mydata.mcs-32),'\n') # note differences!

OneMeanTtest(xbar=xbar1,s=s1,n=n1,mu0=32,ConfidenceLevels=[0.90,0.95,0.99])
```

Python Ttest\_1sampResult built in function  
TtestResult(statistic=-0.5359710658721357, pvalue=0.5922421917489025, df=452)

Python non-parametric Wilcoxon built-in function  
WilcoxonResult(statistic=47638.0, pvalue=0.17542737066915193)

One Mean T test detailed computation  
Sample mean xbar = 31.6767  
Sample standard deviation = 12.8393  
Sample size n = 453  
Null hypothesis claimed mean mu0 = 32  
Confidence Levels [0.9, 0.95, 0.99]

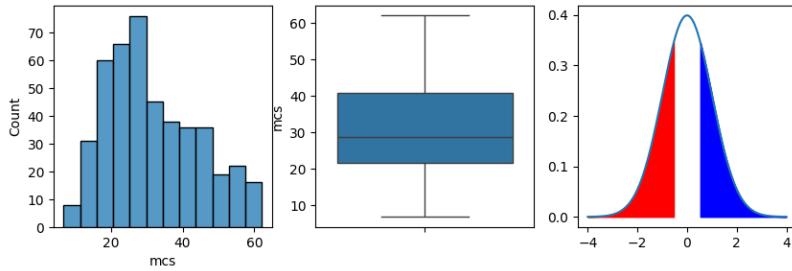
H0: mu = mu0  
H1: mu not mu0 or one-sided test  
  
xbar -+ 2.5\*s bounds are -0.4217, 63.7750  
Standard Error = SE = s/sqrt(n) = 12.8393/sqrt(453) = 0.6032  
degree of freedom = df1 = 453-1 = 452

meansdiff = xbar - mu0 = 31.6767 - 32.0000 = -0.3233  
test statistic = t1 = meansdiff/SE = -0.3233/0.6032 = -0.5360  
P-values=2\*(1-t.cdf(np.abs(t1),df=df1)), 2-sided and 1-sided =  
0.5922421917489022 0.2961210958744511

Cohen D = meansdiff/s = -0.3233/12.8393 = -0.0252

Confidence Interval (CI) approach  
Critical t = tstar = t.ppf(1-alpha/2,df=df1)

	Margin of error = tstar*SE						
	CI	xbar	MarginErr	xbar	CIL	CIR	
0	0.90	1.6482	0.6032	0.9943	31.6767	30.6824	32.6710
1	0.95	1.9652	0.6032	1.1855	31.6767	30.4912	32.8622
2	0.99	2.5867	0.6032	1.5604	31.6767	30.1162	33.2371



Same as in the previous problem, t-test assumptions hold. Even though the distribution of `mcs` scores is a bit skewed right, the sample is so large that the distribution of sample means is normal in any case by the Central Limit Theorem.

Therefore, the sampling distribution of  $\frac{\bar{x} - \mu_0}{SE}$  (**null distribution**) is a t-distribution, where  $SE = \frac{s}{\sqrt{n}} = \frac{12.839}{\sqrt{453}} = 0.603$ .

The **p-value** is the probability of seeing the observed sample mean  $\bar{x} = 31.677$  or something even more extreme under the t-distribution shown in the right plot of the Figure above.

$$t = \frac{\bar{x} - \mu_0}{SE} = \frac{31.677 - 32}{0.603} = -0.536$$

The alternative H1 is two-sided, so both left and right tails need to be accounted for. The areas under the tails add up to  $p\text{-value} = 0.5922422 > 0.05 = \alpha$ , so we conclude that there is *not* enough evidence to reject the initial assumption H0. Thus, this sample gives no reason to doubt that the mean population `mcs` mental score for these substance abusers is 32. Note also that the non-parametric Wilcoxon test leads to the same conclusions.

Also, we are 95% confident (5% significance error) that the true population mean is between 30.49 and 32.86, which contains the claimed  $H_0 : \mu = 32$ . This confirms our conclusion *not* to reject H0.

### 7.1.5 Choosing a Sample Size When Estimating a Mean.

Analogous to proportions, let's investigate how large a sample size  $n$  should be to achieve a certain desired accuracy (given the margin of error  $e$ ). First,

let's solve the formula of the margin of error for  $n$ :

$$\bar{x} \pm t^* \cdot SE \quad \text{Margin of error} = e = t^* \cdot SE = t^* \cdot \frac{s}{\sqrt{n}} \Rightarrow$$

$$\sqrt{n} = t^* \cdot \frac{s}{e} \Rightarrow n = \left( \frac{t^* s}{e} \right)^2$$

Unlike proportions computation, there is an additional complication, though. The critical  $t^*$  requires knowledge of the degree of freedom  $df = n - 1$  while  $n$  is being actually computed. In practice,  $t^*$  is replaced with  $z^*$  from the standard normal distribution. Strictly speaking, this substitution is really valid only for  $n \geq 30$  when t-distribution is almost indistinguishable from the standard normal.

Thus, we obtain an estimate:

$$n = \left( \frac{z^* s}{e} \right)^2 \quad (7.4)$$

### Example

What sample size is needed to estimate the mean height for a population of adults in the United States? Assume that we want to be 95% confident that the sample mean is within 0.2 in of the population mean with a standard deviation of 3 in.

```
[ ]: from scipy.stats import norm
e = 0.2; s = 3; alpha = 0.05;
zstar = norm.ppf(1-alpha/2);
print('e = {:.4f}, alpha = {:.2f}, zstar = {:.4f}'.format(e,alpha,zstar))
n = (zstar*s/e)**2;
print('The number of people needed for the study = n = {:.2f}'.format(n))
```

```
e = 0.2000, alpha = 0.05, zstar = 1.9600
The number of people needed for the study = n = 864.33
```

As always, the estimate is rounded up, so you would need 865 subjects in the study to achieve the desired accuracy.

The estimate's dependence on  $z^*$  and  $e$  is the same as for the sample size in the proportion test -  $n = \frac{(z^*)^2 p(1-p)}{e^2}$ . First, it is directly proportional to  $(z^*)^2$ , so as the level of confidence increases  $0.90 \rightarrow 0.95 \rightarrow 0.99$ ,  $z^*$  increases, and the resulting  $n$  increases. Conversely, as the margin of error  $e$  increases (lower accuracy),  $n$  decreases. Let's illustrate it with a modification of the above example to compute  $n$  for several confidence levels and several margins of error.

```
[ ]: from scipy.stats import norm; import numpy as np;
e = 0.2; s = 3;
```

```
ConfLevel = np.array([0.90,0.95,0.99]); # confidence levels
alpha = 1-ConfLevel; # corresponding error levels
zstar = norm.ppf(1-alpha/2);
# print('e, alpha, zstar = ', e, alpha, zstar)
n = (zstar*s/e)**2;
df = pd.DataFrame({'ConfLevel':ConfLevel,'alpha':alpha,'zstar':zstar,'n':n});
pd.set_option("display.precision", 4); print(df, '\n')
```

	ConfLevel	alpha	zstar	n
0	0.90	0.10	1.6449	608.7473
1	0.95	0.05	1.9600	864.3282
2	0.99	0.01	2.5758	1492.8517

```
[ ]: from scipy.stats import norm; import numpy as np;
e = np.array([0.1,0.2,0.3]); s= 3;
ConfLevel = 0.95; # confidence levels
alpha = 1-ConfLevel; # corresponding error levels
zstar = norm.ppf(1-alpha/2);
# print('e, alpha, zstar = ', e, alpha, zstar)
n = (zstar*s/e)**2;
df = pd.DataFrame({'ConfLevel':ConfLevel,'zstar':zstar,'e':e,'n':n});
pd.set_option("display.precision", 4); print(df, '\n')
```

	ConfLevel	zstar	e	n
0	0.95	1.96	0.1	3457.3129
1	0.95	1.96	0.2	864.3282
2	0.95	1.96	0.3	384.1459

### 7.1.6 Effect Size and Power

Although the hypothesis testing approach is used very widely, there are serious issues with it. One is that a lot of attention is given to obtaining a p-value below 0.05, rather than focusing on the strength of the effect in the population. The p-value is directly related to sample size. As the sample size increases, the standard error  $SE = \frac{s}{\sqrt{n}}$  decreases and  $t = \frac{\bar{x} - \mu_0}{SE}$  increases, therefore p-value (area beyond  $t$ ) decreases. Thus, it is more likely to reject a null hypothesis for large samples. Therefore, we should focus not only on the p-value but also on the actual strength of the effect that we are investigating - **effect size**. More precisely, it is a standardized difference between observed and claimed value or between two or more groups.

For one-sample means test studied in the previous sections, Cohen defined effect size to be:

$$d = \frac{\bar{x} - \mu_0}{s}$$

which is essentially the  $t$  or  $z$  of  $\bar{x}$  based on the original standard deviation  $s$ , not standard error  $SE = \frac{s}{\sqrt{n}}$ . He also introduced guidelines on what constitutes a small, medium or large effect: small ( $d = 0.2$ ), medium ( $d = 0.5$ ), and large ( $d = 0.8$ ).

Another criticism is that too much focus is on the probability of making a Type I error (rejecting H<sub>0</sub> when it is actually true) and not enough attention is given to the Type II error (not rejecting H<sub>0</sub> when it is actually false). Ideally, we need a method sensitive enough to detect real effects in the underlying population which is measured by the **statistical power (probability to reject H<sub>0</sub> when it is actually false - complement of Type II error)**. A reasonable level of power is 0.80-0.90 which translates into 80%-90% probability of finding the effect if it exists in the population. Power is influenced by a number of factors:

1. Sample size - a larger sample increases the power.
2. Effect size - a bigger effect increases the power (big effect is easier to detect).
3. The criterion for significance error  $\alpha$  (Type I error probability) is usually set at 0.05. Reducing  $\alpha$  to 0.01, say, will make it harder to reject H<sub>0</sub> when it is true, but also when it is false and power decreases. Increasing  $\alpha$  to 0.10, say, will make it easier to reject H<sub>0</sub> when it is true, but also when it is false and the power increases.
4. Whether you have a one- or two-tailed hypothesis, one-tailed statistical tests tend to be more powerful than two-tailed tests.

If we failed to reject the null hypothesis, it can be argued that there is no effect to detect in the population. If, however, the study was underpowered (e.g., small sample), then we don't know if the null was accepted correctly or whether it was a Type II error.

The code below shows that given any three of the sample size -  $n$ , probability of Type I error -  $\alpha$ , effect size -  $d$ , and power -  $power$ , the fourth one can be found with `TTestPower().solve_power()` function.

```
[ ]: from statsmodels.stats.power import TTestPower
n1 = 40;   print('Given sample size = n1 = ', n1)
d1 = 0.5;  print('Given Cohen d effect size = difference/s = ', d1)
alpha1 = 0.05;  print('Significance level = alpha = ', alpha1)
power1 = TTestPower().solve_power(nobs=n1, effect_size = d1, \
                                  power = None, alpha = alpha1)
print('For the values above, power = {:.4f}\n'.format(power1))

power1 = 0.90;  print('Given power = ', power1)
d1 = 0.5;  print('Given Cohen d effect size = difference/s = ', d1)
alpha1 = 0.05;  print('Significance level = alpha = ', alpha1)
n1 = TTestPower().solve_power(nobs=None, effect_size = d1, \
                               power = power1, alpha = alpha1)
print('Needed sample size = n1 = {:.4f}\n'.format(n1))

power1 = 0.90;  print('Given power = ', power1)
n1 = 40;   print('Given sample size = n1 = ', n1)
alpha1 = 0.05;  print('Significance level = alpha = ', alpha1)
d1 = TTestPower().solve_power(nobs=n1, effect_size = None, \
                               power = power1, alpha = alpha1)
print('The detectable effect size = d1 = {:.4f}\n'.format(d1))
```

```

power1 = 0.90;  print('Given power = ', power1)
n1 = 40;      print('Given sample size = n1 = ', n1)
d1 = 0.5;      print('Given Cohen d effect size = difference/s = ', d1)
alpha1 = TTestPower().solve_power(nobs=n1, effect_size = d1, \
                                  power = power1, alpha = None)
print('Probability of Type 1 error = alpha1 = {:.4f}\n'.format(alpha1))

```

Given sample size = n1 = 40  
 Given Cohen d effect size = difference/s = 0.5  
 Significance level = alpha = 0.05  
 For the values above, power = 0.8694

Given power = 0.9  
 Given Cohen d effect size = difference/s = 0.5  
 Significance level = alpha = 0.05  
 Needed sample size = n1 = 43.9955

Given power = 0.9  
 Given sample size = n1 = 40  
 Significance level = alpha = 0.05  
 The detectable effect size = d1 = 0.5256

Given power = 0.9  
 Given sample size = n1 = 40  
 Given Cohen d effect size = difference/s = 0.5  
 Probability of Type 1 error = alpha1 = 0.0698

It is more illuminating to plot power curves which show how the change in effect size and sample size impact the power of the statistical test. The Figure below shows how the power of the test increases with a larger effect size (it is easier to detect a larger effect size). It also increases much steeper for larger sample sizes.

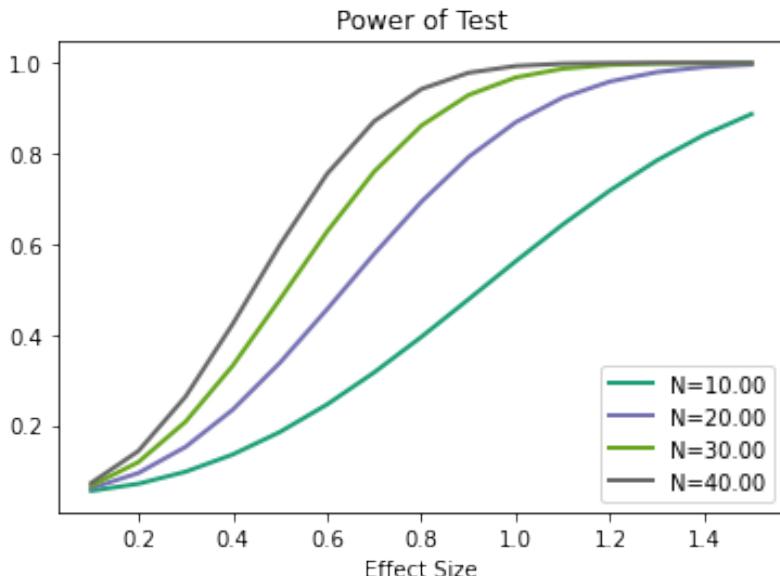
```

[ ]: import numpy as np
      import matplotlib.pyplot as plt
      from statsmodels.stats.power import TTestIndPower

      # power analysis varying parameters
      effect_sizes = np.arange(0.1,1.6,0.1)
      sample_sizes = np.array([10,20,30,40])

      # plot power curves
      TTestIndPower().plot_power(dep_var='effect_size', nobs=sample_sizes,
                                 effect_size=effect_sizes, alpha=0.05)
      plt.show()

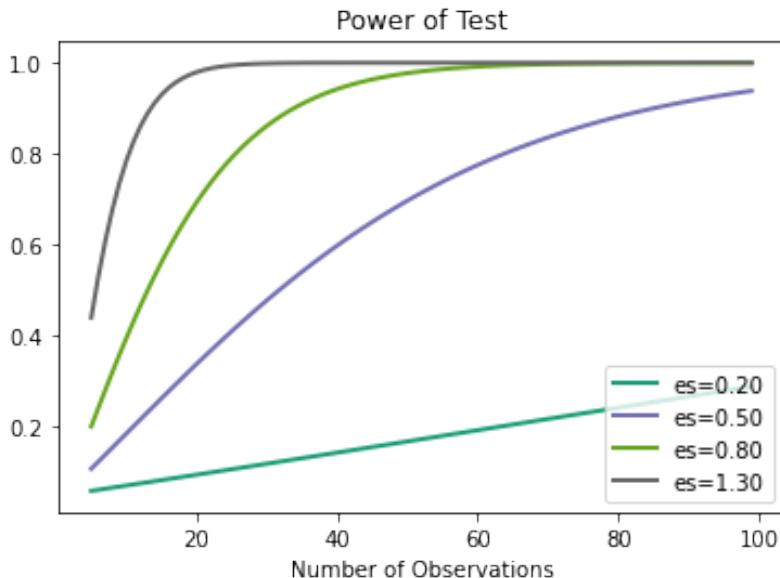
```



Analogously, the Figure below shows how the power of the test increases with the sample size. Also it is steeper for the larger effect size.

```
[ ]: # power analysis varying parameters
effect_sizes = np.array([0.2, 0.5, 0.8, 1.3])
sample_sizes = np.array(range(5, 100))

# plot power curves
TTestIndPower().plot_power(dep_var='nobs', nobs=sample_sizes,
                           effect_size=effect_sizes, alpha=0.05)
plt.show()
```



## 7.2 Means Tests for Paired Data Samples

Two samples are **paired (dependent)** if each value in one sample is paired with a corresponding value in the other sample. The typical example of paired samples is two weight measurements from the same subjects before and after the diet and exercise treatment. Another paired samples example is provided by comparing prices for the college textbooks at the Campus store and online. In some cases, the same study can be done in both paired and unpaired fashion. For example, consider a study of the effect of alcohol on people's reaction time. For a *paired* approach, we could have the same subject take the reaction time test before and after drinking alcohol. On the other hand, we could have randomly assigned all subjects to two *independent* groups. One group would drink a real alcohol, while another one a non-alcoholic drink with similar taste. In principle, if you have a choice, a paired design is much better as the test is done on the same person before and after which removes confounding effects due to unintended bias in two groups. Obviously, in many cases, paired design is just not possible, say, comparing test scores for two different socioeconomic groups. We will consider independent samples tests in the next section.

The key observation is that because the data are paired, we can construct the **differences**. Therefore, we get just **one set of differences** and we are back to one sample t-test derived in the previous section with the same exact code!

$$H_0 : \mu_d = 0$$

$$H_1 : \mu_d \neq 0 \quad \text{or} \quad < >$$

$$SE = \frac{s_d}{\sqrt{n}} \quad df = n - 1 \quad t_{df} = \frac{\bar{d} - 0}{SE}$$

where  $\bar{d}$  is mean of sample differences,  $n$  is number of pairs, and  $s_d$  is their standard deviation.

### Example

In a test of the effectiveness of a new exercise system, 16 randomly selected subjects were observed following this system. The weight changes (before-after) have a mean of 2.7 lb and a standard deviation of 5.5 lb. The minimum value of this difference was -3.9 and the maximum was 9.1. Is there a difference at the 5% level?

### Solution:

This is a classical example of paired before and after data. In fact, we are already given summaries in terms of *differences*. The only change to the code used for one sample means tests is to set the null hypothesis  $H_0$  to the true population mean difference of 0 implying no difference.

$$H_0 : \mu_d = 0 \text{ (assume true)}$$

$$H_1 : \mu_d \neq 0$$

The hypothesis test seeks to determine if the sample mean of 2.7 is significantly different from the claimed null hypothesis 0, i.e., whether there is a significant difference before and after the exercise course.

```
[ ]: OneMeanTtest(xbar=2.7,s=5.5,n=16,mu0=0,ConfidenceLevels=[0.90,0.95,0.99])
```

```
One Mean T test detailed computation
Sample mean xbar = 2.7000
Sample standard deviation = 5.5000
Sample size n = 16
Null hypothesis claimed mean mu0 = 0
Confidence Levels [0.9, 0.95, 0.99]
```

```
H0: mu = mu0
H1: mu not mu0 or one-sided test
```

```
xbar +- 2.5*s bounds are -11.0500, 16.4500
Standard Error = SE = s/sqrt(n) = 5.5000/sqrt(16) = 1.3750
degree of freedom = df1 = 16-1 = 15
```

```
meansdiff = xbar - mu0 = 2.7000 - 0.0000 = 2.7000
test statistic = t1 = meansdiff/SE = 2.7000/1.3750 = 1.9636
```

```
P-values=2*(1-t.cdf(np.abs(t1),df=df1)), 2-sided and 1-sided =
0.06838512873304659 0.03419256436652329
```

```
Cohen D = meansdiff/s = 2.7000/5.5000 = 0.4909
```

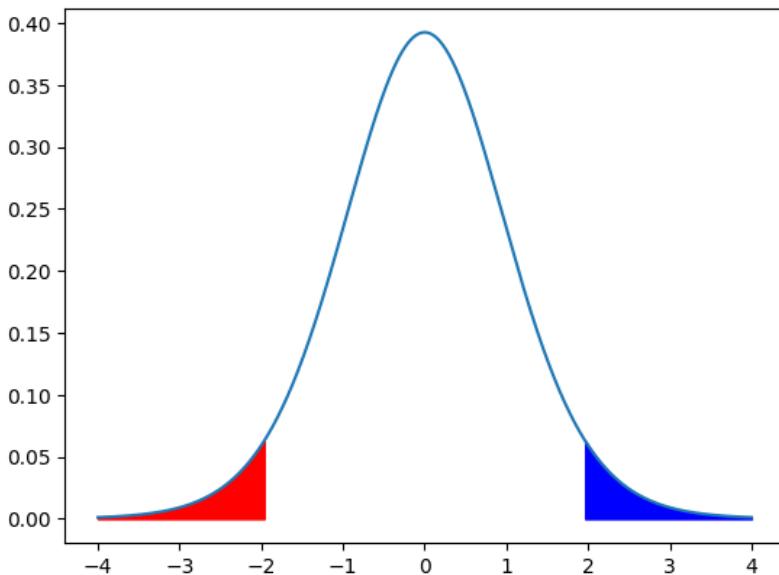
Confidence Interval (CI) approach

Critical t = tstar = t.ppf(1-alpha/2,df=df1)

Margin of error = tstar\*SE

CI = xbar +- MarginErr

	ConfLevel	tstar	SE	MarginErr	xbar	CIL	CIR
0	0.90	1.7531	1.375	2.4104	2.7	0.2896	5.1104
1	0.95	2.1314	1.375	2.9307	2.7	-0.2307	5.6307
2	0.99	2.9467	1.375	4.0517	2.7	-1.3517	6.7517



The assumptions for the t-test hold the same way for differences. A simple random sample of patients ensures independence. The min/max of the differences data are within 2.5 standard deviations away from the mean, so no extreme outliers.

Therefore, the sampling distribution of  $\frac{\bar{x} - \mu_0}{SE}$  (**null distribution**) is a t-distribution with  $SE = \frac{s}{\sqrt{n}} = \frac{5.5}{\sqrt{16}} = 1.375$ .

The **p-value** is the probability of seeing the observed sample differences mean  $\bar{x} = 2.7$  or something even more extreme under the t null distribution shown in the Figure above.

$$t = \frac{\bar{x} - \mu_0}{SE} = \frac{2.7 - 0}{1.375} = 1.964$$

The alternative is two-sided  $H_1 : \mu \neq 0$ , so both tails must be accounted for. The areas under the tails add up to p-value = 0.0684 > 0.05 =  $\alpha$ , so we conclude that there is *not* enough evidence to reject the initial assumption  $H_0 : \mu = 0$ . Thus, the exercise system does not seem to be effective.

Using a confidence interval approach, we are 95% confident (5% significance error) that the true population mean of differences is between -0.231 and 5.631, which contains the claimed  $H_0 : \mu = 0$ . This confirms our conclusion *not* to reject  $H_0$ .

### Example

Consider the data set textbooks available from the `openintro` library. It has prices for 73 textbooks at the UCLA bookstore and Amazon. Test whether there is a difference at the 1% level.

This is another classical example of paired data - each book has a different price from two different sellers. This time, we have the actual data set, thus we can explore the boxplot and the histogram of the differences in the Figure below. The mean, standard deviation, and sample size of the differences have to be computed before my step-by-step function can be applied.

**H0** :  $\mu = 0$  (assume true)

**H1** :  $\mu \neq 0$

The hypothesis test seeks to determine if the sample mean  $\bar{x} = 12.762$  is significantly different from the claimed null hypothesis 0, i.e., whether there is a significant difference between the prices for the same textbooks at the UCLA bookstore and at Amazon.

```
[ ]: import pandas as pd; import numpy as np;
import matplotlib.pyplot as plt; import seaborn as sns

url="https://raw.githubusercontent.com/leonkag/Statistics0/main/
    ↪UCLAtextbooks.csv"
mydata = pd.read_csv(url)      # save as mydata file
mydata
```

	dept_abbr	course	isbn	ucla_new	amaz_new	more	diff
0	Am Ind	C170	978-0803272620	27.67	27.95	Y	-0.28
1	Anthro	9	978-0030119194	40.59	31.14	Y	9.45
2	Anthro	135T	978-0300080643	31.68	32.00	Y	-0.32
3	Anthro	191HB	978-0226206813	16.00	11.52	Y	4.48
4	Art His	M102K	978-0892365999	18.95	14.21	Y	4.74
..	...	...	...	...	...	...	...
68	Wld Art	C180	978-1412923484	48.46	39.34	N	9.12
69	Wld Art	C409A	978-0195390827	39.55	26.37	N	13.18
70	Wld Art	495	978-0820474151	29.65	24.19	Y	5.46
71	Wom Std	M144	978-1570755637	23.76	18.72	Y	5.04
72	Wom Std	285	978-0822341147	27.70	18.22	Y	9.48

[73 rows x 7 columns]

```
[ ]: mydata['difference'] = mydata['ucla_new'] - mydata['amaz_new']

fig, axs = plt.subplots(1, 3, figsize=(10, 3))
sns.histplot(data=mydata,x="difference", ax=axs[0])
sns.boxplot(data=mydata[["ucla_new","amaz_new"]], ax=axs[1])

from scipy import stats    # Built-in function approach
print('Python Ttest_1sampResult built in function')
print(stats.ttest_rel(mydata.ucla_new, mydata.amaz_new), '\n')

from scipy.stats import wilcoxon    # Built-in function approach
print('Python non-parametric Wilcoxon built in function')
print(wilcoxon(mydata.ucla_new, mydata.amaz_new), '\n')

# My function detailed computations
xbar1 = np.mean(mydata.difference);
s1 = np.std(mydata.difference,ddof=1);
n1 = len(mydata.difference);
OneMeanTtest(xbar=xbar1,s=s1,n=n1,mu0=0,ConfidenceLevels=[0.90,0.95,0.99])
```

Python Ttest\_1sampResult built in function  
TtestResult(statistic=7.648771112479753, pvalue=6.927581126065491e-11, df=72)

Python non-parametric Wilcoxon built in function  
WilcoxonResult(statistic=99.0, pvalue=9.216616365779943e-12)

One Mean T test detailed computation  
Sample mean xbar = 12.7616  
Sample standard deviation = 14.2553  
Sample size n = 73  
Null hypothesis claimed mean mu0 = 0  
Confidence Levels [0.9, 0.95, 0.99]

H0: mu = mu0  
H1: mu not mu0 or one-sided test

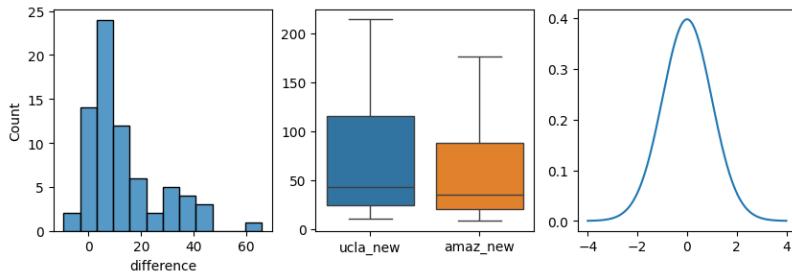
xbar +- 2.5\*s bounds are -22.8766, 48.3999  
Standard Error = SE = s/sqrt(n) = 14.2553/sqrt(73) = 1.6685  
degree of freedom = df1 = 73-1 = 72

meansdiff = xbar - mu0 = 12.7616 - 0.0000 = 12.7616  
test statistic = t1 = meansdiff/SE = 12.7616/1.6685 = 7.6488  
P-values=2\*(1-t.cdf(np.abs(t1),df=df1)), 2-sided and 1-sided =  
6.927569629056052e-11 3.463784814528026e-11

Cohen D = meansdiff/s = 12.7616/14.2553 = 0.8952

Confidence Interval (CI) approach  
Critical t = tstar = t.ppf(1-alpha/2,df=df1)  
Margin of error = tstar\*SE  
CI = xbar +- MarginErr  

	ConfLevel	tstar	SE	MarginErr	xbar	CIL	CIR
0	0.90	1.6663	1.6685	2.7801	12.7616	9.9815	15.5418
1	0.95	1.9935	1.6685	3.3260	12.7616	9.4356	16.0877
2	0.99	2.6459	1.6685	4.4145	12.7616	8.3472	17.1761



As always, a simple random sample is assumed, which ensures independence. The histogram and boxplot above show that the data is skewed right, but the sample size  $n = 73 > 30$ , so the CLT guarantees the normality of the sampling distribution, and the t-test is robust to departures from normality in any case. We run the non-parametric Wilcoxon test as well for comparison - it reaches the same conclusion.

Therefore, the sampling null distribution of  $\frac{\bar{x} - \mu_0}{SE}$  is approximately a t-distribution (although standard normal could have been used for such a large sample as well), where  $SE = \frac{s}{\sqrt{n}} = \frac{14.255}{\sqrt{73}} = 1.668$ .

The **p-value** is the probability of seeing the observed sample mean  $\bar{x}$  or something even more extreme under our t null distribution shown in the Figure above.

$$t = \frac{\bar{x} - \mu_0}{SE} = \frac{12.762 - 0}{1.668} = 7.649$$

The alternative is two-sided  $H1 : \mu \neq 0$ , so both tails must be accounted for. The areas under the tails add up to p-value =  $6.93 \cdot 10^{-11} < 0.01 = \alpha$ , so we conclude that there is enough evidence to reject the initial assumption  $H0 : \mu = 0$ . Thus, there is a difference in book prices.

Using the confidence interval approach, we are 99% confident (1% significance error) that the true population mean of differences is between 8.35 and 17.18, which does *not* contain the claimed  $H0 : \mu = 0$ . This confirms our conclusion to reject  $H0$ .

### Example

To determine if a massage is effective for treating muscle pain, a pilot study was conducted where a certified therapist treated randomly chosen patients. Pain level was measured immediately before and after the treatment. The data is given in the code below. Do the data show that the treatment reduces pain? Test at the 5% level.

First, the boxplot and histogram of the differences (before - after) are explored in the Figure below. Because it asks for the *reduction of pain*, it implies that

the average pain scores “before” are larger than “after”; therefore it is a **one-sided test** to the right:

**H0** :  $\mu_d = 0$  (**assume true**)

**H1** :  $\mu_d > 0$  (**note it is a one-sided test**)

The hypothesis test determines whether the sample mean  $\bar{x} = 1.65$  is significantly *larger* than the claimed null hypothesis H0:  $\mu_d = 0$ , which implies a significant reduction in pain from before to after the treatment. Note `alternative = "greater"` must be used in built-in tests.

```
[ ]: import pandas as pd; import numpy as np;
import matplotlib.pyplot as plt; import seaborn as sns

mydata = pd.DataFrame({'before':[2, 3, 8, 2, 3, 3, 2, 6, 2, 5, 5, 3, 3, 3, 3,
    ↪7, 5, 1, 3, 9, 1],
    'after' :[0, 2, 6, 0, 0, 3, 3, 4, 1, 5, 2, 3, 0, 3, 3,
    ↪3, 0, 2, 1, 4, 1]})

mydata['difference'] = mydata['before'] - mydata['after']

fig, axs = plt.subplots(1, 3, figsize=(10, 3))
sns.histplot(data=mydata,x="difference", ax=axs[0])
sns.boxplot(data=mydata[["before","after"]], ax=axs[1])

from scipy import stats      # Built-in function approach
print('Python Ttest_1sampResult built in function')
print(stats.ttest_rel(mydata.before, mydata.after,
    ↪alternative='greater'), '\n')

from scipy.stats import wilcoxon    # Built-in function approach
print('Python non-parametric Wilcoxon built in function')
print(wilcoxon(mydata.before, mydata.after, alternative='greater'), '\n')

# My function detailed computations
xbar1 = np.mean(mydata.difference);
s1 = np.std(mydata.difference,ddof=1);
n1 = len(mydata.difference);
OneMeanTtest(xbar=xbar1,s=s1,n=n1,mu0=0,ConfidenceLevels=[0.90,0.95,0.99])
```

Python Ttest\_1sampResult built in function  
 TtestResult(statistic=4.066886812455412, pvalue=0.000328820598623293, df=19)

Python non-parametric Wilcoxon built in function  
 WilcoxonResult(statistic=115.0, pvalue=0.0008274046414323649)

One Mean T test detailed computation  
 Sample mean xbar = 1.6500  
 Sample standard deviation = 1.8144  
 Sample size n = 20  
 Null hypothesis claimed mean mu0 = 0  
 Confidence Levels [0.9, 0.95, 0.99]

H0: mu = mu0  
 H1: mu not mu0 or one-sided test

```

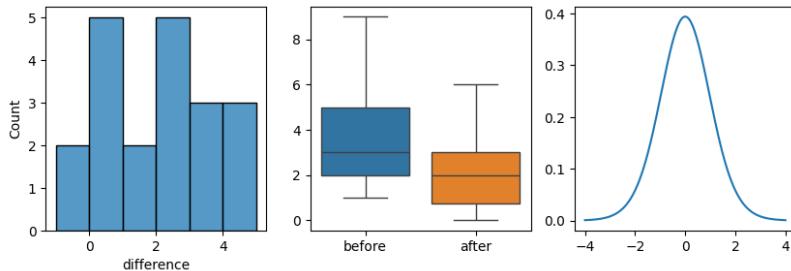
xbar +- 2.5*s bounds are -2.8860, 6.1860
Standard Error = SE = s/sqrt(n) = 1.8144/sqrt(20) = 0.4057
degree of freedom = df1 = 20-1 = 19

meansdiff = xbar - mu0 = 1.6500 - 0.0000 = 1.6500
test statistic = t1 = meansdiff/SE = 1.6500/0.4057 = 4.0669
P-values=2*(1-t.cdf(np.abs(t1),df=df1)), 2-sided and 1-sided =
0.000657641197246539 0.0003288205986232695

Cohen D = meansdiff/s = 1.6500/1.8144 = 0.9094

Confidence Interval (CI) approach
Critical t = tstar = t.ppf(1-alpha/2,df=df1)
Margin of error = tstar*SE
CI = xbar +- MarginErr
    ConfLevel      tstar          SE   MarginErr     xbar       CIL       CIR
0            0.90    1.7291    0.4057      0.7015    1.65  0.9485  2.3515
1            0.95    2.0930    0.4057      0.8492    1.65  0.8008  2.4992
2            0.99    2.8609    0.4057      1.1607    1.65  0.4893  2.8107

```



A simple random sample ensures independence. The histogram and boxplot above show no extreme outliers, so assumptions of normality are not violated. Thus, the assumptions for the t-test hold.

Therefore, the sampling distribution of  $\frac{\bar{x} - \mu_0}{SE}$  (**null distribution**) is t-distribution, where  $SE = \frac{s}{\sqrt{n}} = \frac{1.814}{\sqrt{20}} = 0.406$ .

The **p-value** is the probability of seeing the observed sample mean  $\bar{x}$  or something even more extreme under our t null distribution shown in the Figure above.

$$t = \frac{\bar{x} - \mu_0}{SE} = \frac{1.65 - 0}{0.406} = 4.067$$

The alternative is one-sided to the right  $H1 : \mu > 0$ , so we have to account only for the right tail area, which gives p-value =  $3.29 \cdot 10^{-4} = 0.000329 < 0.05 = \alpha$ . Therefore, there is enough evidence to reject the initial assumption  $H0: \mu = 0$  at 0.05 level (or at 0.01 as well). Thus, there is a significant reduction in pain.

For a one-sided test, we have to use  $1 - 2\alpha = 1 - 2 \cdot 0.05 = 0.90$  confidence level since CI is two-sided. Therefore, we are 90% confident (for 10% significance error rate) that the true population mean is between 0.95 and 2.35, which does NOT contain the claimed  $H_0 : \mu = 0$ . This confirms our conclusion to reject  $H_0$ .

### Example

A small sample of students is tested before and after they use the tutoring center. Test if their grades improved at the 1% level.

Once again, we have the actual data set; thus we can explore the boxplots and the histograms of the differences in the Figure below. Note also that the statement of the improvement of grades means that the average scores “before” are smaller than “after”; therefore it is a *one-sided test to the left*.

**H0** :  $\mu = 0$  (assume true).

**H1** :  $\mu < 0$  (one-sided test).

The hypothesis test seeks to determine if the mean of the differences  $\bar{x} = -2.167$  is significantly *smaller* than the claimed null hypothesis  $\mu_0 = 0$ , i.e., whether there is a significant improvement in the test scores due to tutoring.

```
[ ]: import pandas as pd; import numpy as np;
import matplotlib.pyplot as plt; import seaborn as sns

mydata = pd.DataFrame({'before':[74, 89, 75, 83, 92, 91],
                       'after' :[78, 91, 77, 81, 96, 94]})

mydata['difference'] = mydata['before'] - mydata['after']

fig, axs = plt.subplots(1, 3, figsize=(10, 3))
sns.histplot(data=mydata,x="difference", ax=axs[0])
sns.boxplot(data=mydata[["before","after"]], ax=axs[1])

from scipy import stats      # Built-in function approach
print('Python Ttest_1sampResult built in function')
print(stats.ttest_rel(mydata.before, mydata.after, alternative='less'), '\n')

from scipy.stats import wilcoxon    # Built-in function approach
print('Python non-parametric Wilcoxon built in function')
print(wilcoxon(mydata.before, mydata.after, alternative='less'), '\n')

# My function detailed computations
xbar1 = np.mean(mydata.difference);
s1 = np.std(mydata.difference,ddof=1);
n1 = len(mydata.difference);
OneMeanTtest(xbar=xbar1,s=s1,n=n1,mu0=0,ConfidenceLevels=[0.90,0.95,0.98,0.
→99])
```

Python Ttest\_1sampResult built-in function  
TtestResult(statistic=-2.381415742703763, pvalue=0.03152714133931948, df=5)

Python non-parametric Wilcoxon built-in function  
WilcoxonResult(statistic=2.0, pvalue=0.046875)

```
One Mean T test detailed computation
Sample mean xbar = -2.1667
Sample standard deviation = 2.2286
Sample size n = 6
Null hypothesis claimed mean mu0 = 0
Confidence Levels [0.9, 0.95, 0.98, 0.99]
```

```
H0: mu = mu0
H1: mu not mu0 or one-sided test
```

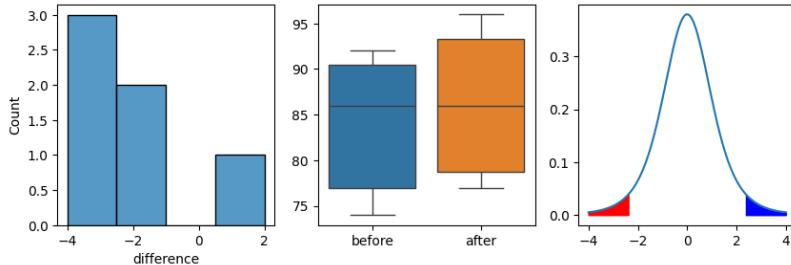
```
xbar +- 2.5*s bounds are -7.7382, 3.4048
Standard Error = SE = s/sqrt(n) = 2.2286/sqrt(6) = 0.9098
degree of freedom = df1 = 6-1 = 5
```

```
meansdiff = xbar - mu0 = -2.1667 - 0.0000 = -2.1667
test statistic = t1 = meansdiff/SE = -2.1667/0.9098 = -2.3814
P-values=2*(1-t.cdf(np.abs(t1),df=df1)), 2-sided and 1-sided =
0.06305428267863888 0.03152714133931944
```

```
Cohen D = meansdiff/s = -2.1667/2.2286 = -0.9722
```

```
Confidence Interval (CI) approach
Critical t = tstar = t.ppf(1-alpha/2,df=df1)
Margin of error = tstar*SE
CI = xbar +- MarginErr
```

	ConfLevel	tstar	SE	MarginErr	xbar	CIL	CIR
0	0.90	2.0150	0.9098	1.8333	-2.1667	-4.0000	-0.3333
1	0.95	2.5706	0.9098	2.3388	-2.1667	-4.5054	0.1721
2	0.98	3.3649	0.9098	3.0615	-2.1667	-5.2282	0.8948
3	0.99	4.0321	0.9098	3.6685	-2.1667	-5.8352	1.5019



A simple random sample ensures independence. The data set is so small in this problem that we cannot really confirm normality from the histogram and/or boxplot; however, test scores tend to be bell-shaped, so it is reasonable to assume that the assumptions for the t-test hold.

Therefore, the sampling distribution of  $\frac{\bar{x} - \mu_0}{SE}$  (**null distribution**) is t-distribution, where  $SE = \frac{s}{\sqrt{n}} = \frac{2.229}{\sqrt{6}} = 0.91$ .

The **p-value** is the probability of seeing the observed sample mean  $\bar{x} = -2.167$  or something even more extreme under our t null distribution shown in the Figure above.

$$t = \frac{\bar{x} - \mu_0}{SE} = \frac{-2.167 - 0}{0.91} = -2.381$$

The alternative is one-sided to the left  $H1 : \mu < 0$ , so we have to account only for the left tail area, which produces p-value = 0.0315271 > 0.01 =  $\alpha$ , so there is *not* enough evidence to reject the initial assumption  $H0: \mu = 0$ . Thus, there is *no* significant improvement due to tutoring at this strict level. If the usual  $\alpha = 0.05$  was used, the  $H0$  would have been rejected. Thus, it is very important to be mindful of the level of significance  $\alpha$ .

For a one-sided test, we have to use  $1 - 2\alpha = 1 - 2 \cdot 0.01 = 0.98$  confidence level since CI is two-sided (0.98 was added to the list of CI levels). Thus, we are 98% confident (2% significance error rate) that the true population mean is between -5.228 and 0.895, which contains the claimed  $H0 : \mu = 0$ . This confirms our conclusion not to reject  $H0$ .

If  $\alpha = 0.05$  was used,  $1 - 2\alpha = 1 - 2 \cdot 0.05 = 0.90$  confidence interval  $(-4.000, -0.333)$  does not contain 0 and would have resulted in the rejection of  $H0$ .

### *Power of Paired Means Test*

For a paired means test, the **effect size** is a standardized difference between the paired groups:

$$D = \frac{\bar{d}}{s_d}$$

where  $\bar{d}$  is the mean of the paired differences and  $s_d$  is their standard deviation. Cohen's guidelines are similar: small  $D = 0.2$ , medium  $D = 0.5$ , and large  $D = 0.8$ . Also,  $n$  is now the number of observations per sample - the number of pairs. Otherwise, the computations and power graphs are the same as for a one-sample test.

## 7.3 Means Tests for Two Independent Samples

Finally, in this longer chapter consider two **independent (not paired)** samples. Their sample means are  $\bar{x}_1$  and  $\bar{x}_2$ , sample standard deviations are  $s_1$  and  $s_2$ , and sample sizes are  $n_1$  and  $n_2$  (sample sizes do *not have to be equal*).

The conditions to use t-distribution for the difference of independent means are given below:

1. **Independence** between and within the groups. The two simple random samples would ensure independence within each group, and the samples have

to be independent from each other (not paired).

2. **Normality** - the outlier's rules of thumb are checked for each group separately.

Hypothesis test:

$$H_0: \mu_1 = \mu_2 \iff \mu_1 - \mu_2 = 0$$

$$H_1: \mu_1 \neq \mu_2 \iff \mu_1 - \mu_2 \neq 0$$

The standard error for the difference of means is computed differently depending on the variances of the groups:

1. **Pooled Variance** - population variances are assumed to be equal:  $\sigma_1^2 = \sigma_2^2$ . If the original data are available, a Levene's test for equality of variances can assess it. If not, equality is assumed if the ratio of the larger sample standard deviation to the smaller is less than 2 (rule of thumb).

$$\text{Pooled Variance} = s_p^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}$$

$$\text{Standard Error} = SE = \sqrt{s_p^2 \left( \frac{1}{n_1} + \frac{1}{n_2} \right)}$$

$$\text{Test Statistic} = t_{df} = \frac{\text{point estimate} - \text{null value}}{SE} = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{s_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

$$\text{Degree of Freedom} = df = n_1 + n_2 - 2$$

$$\text{Margin of error} = e = t_{df}^* SE \quad CI = (\bar{x}_1 - \bar{x}_2) \pm e$$

## 2. Unpooled Variance

In case the population variances cannot be assumed equal, there is no exact solution, but Welch's approximation is used:

$$\text{Standard Error} = SE = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

$$\text{Test Statistic} = t_{df} = \frac{\text{point estimate} - \text{null value}}{SE} = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

$$\text{Degree of Freedom} = df = \frac{\left( \frac{s_1^2}{n_1} + \frac{s_2^2}{n_2} \right)^2}{\frac{\left( \frac{s_1^2}{n_1} \right)^2}{n_1 - 1} + \frac{\left( \frac{s_2^2}{n_2} \right)^2}{n_2 - 1}}$$

$$\text{Margin of error} = e = t_{df}^* SE \quad CI = (\bar{x}_1 - \bar{x}_2) \pm e$$

The non-parametric analog of these t-tests is the **Mann–Whitney** test. Like other non-parametric tests, it uses the data ranks to compare the two groups rather than actual data, so it is not sensitive to skew and outliers. It is included in the code when the data is available.

### Example

Compare the test scores of independent samples of men and women on a standardized accounting test. We take a random sample of 31 people (16 men and 15 women). Sample means are 80 and 84, and standard deviations are 12 and 14, for men and women, respectively. Note that women outperform men by 4 points, but that might be a sampling error. We would like to test whether this difference is significant at the 5% level.

$$H_0 : \mu_1 = \mu_2 \iff \mu_1 - \mu_2 = 0 \text{ (assumed true)}$$

$$H_1 : \mu_1 \neq \mu_2 \iff \mu_1 - \mu_2 \neq 0$$

The function below is rather long, but it just implements the formulas outlined above.

```
[ ]: def TwoMeansTtests2(x1,s1,n1,x2,s2,n2,ConfLevels):
    import numpy as np; from scipy.stats import t; import pandas as pd
    import matplotlib.pyplot as plt
    print('Two Independent Means T test detailed computation')
    print('Sample means, standard deviation, and sample size are:')
    print('1st Sample: x1 = {:.3f}, s1 = {:.3f}, n1 = {:d}'
          .format(x1,s1,n1))
    print('1st Sample: x2 = {:.3f}, s2 = {:.3f}, n2 = {:d}'
          .format(x2,s2,n2))
    print('Confidence Levels',ConfLevels,'\\n')
    ConfLevels = np.array(ConfLevels)

    print('H0: mu1-mu2 = 0')
    print('H1: mu1-mu2 not 0 or one-sided test\\n')

    print('1st sample xbar +- 2.5*s: ({:.4f}, {:.4f})'.
          format(x1 - 2.5*s1,x1 + 2.5*s1))
    print('2nd sample xbar +- 2.5*s: ({:.4f}, {:.4f})\\n'.
          format(x2 - 2.5*s2,x2 + 2.5*s2))

    sdratio = np.maximum(s1/s2,s2/s1)
    print("standard deviations ratios <2? = {:.4f}\\n".format(sdratio))

    meansdiff = x1-x2
    print('meansdiff = x1-x2 = {:.4f} - {:.4f} = {:.4f}\\n'.
          format(x1,x2,meansdiff))

    print("====")
    print("1st Pooled approach: u"
         "-----")
```

```

sp2 = ((n1-1)*s1**2 + (n2-1)*s2**2)/(n1+n2-2)
print("sp2 = ((n1-1)*s1**2 + (n2-1)*s2**2)/(n1+n2-2) = ")
print('      ({:.d}-1){:.3f}**2 + ({:.d}-1){:.3f}**2)/({:.d}+{:.d}-2) = {:.3f}'.
      .format(n1,s1,n2,s2,n1,n2,sp2))
SE = np.sqrt(sp2*(1/n1+1/n2))
print("SE = np.sqrt(sp*(1/n1+1/n2)) = np.sqrt({:.3f}*(1/{:.d}+1/{:.d}) = {:.3f}".
      .format(sp2,n1,n2,SE))
df1 = n1+n2-2
print("degree of freedom = df = n1+n2-2 = {:.d}+{:.d}-2 = {:.d}".
      .format(n1,n2,df1))
t1 = meansdiff/SE
print('t = meansdiff/SE = {:.3f}/{:.3f} = {:.3f}'.
      .format(meansdiff,SE,t1))
pval2 = 2*(1 - t.cdf(x=abs(t1), df=df1))
print('pval2 = 2*(1 - spst.t.cdf(x=abs(t1), df=df1))')
print("pval two and one tailed : ", pval2, pval2/2, '\n')

print("Compute confidence interval for the difference of means:")
alpha = 1-ConfLevels;
tstar = t.ppf(q=1-alpha/2,df=df1)
print("tstar = t.ppf(q=1-alpha/2,df=df1)")
MarginErr = tstar*SE; # margin of error
print("MarginErr = tstar*SE")
CIL = meansdiff - MarginErr; CIR = meansdiff + MarginErr;
print("CIL = meansdiff - MarginErr; CIR = meansdiff + MarginErr; ")
display = pd.DataFrame({'ConfLevels':ConfLevels,'tstar':tstar,
                        'SE':SE, 'MarginErr':MarginErr,'x1-x2':meansdiff,
                        'CIL':CIL,'CIR':CIR});
pd.set_option("display.precision", 4); print(display, '\n')

print("=====")

print("2nd UNpooled approach (Welch Approximation):-----")
SE = np.sqrt(s1**2/n1 + s2**2/n2)
print("SE = np.sqrt(s1^2/n1 + s2^2/n2) = ")
print("  sqrt({:.3f}**2/{:.d} + {:.3f}**2/{:.d}) = {:.3f}".
      .format(s1,n1,s2,n2,SE))
df1 = (s1**2/n1+s2**2/n2)**2/
      ((s1**2/n1)**2/(n1-1)+(s2**2/n2)**2/(n2-1));
print("degree of freedom = df = ",df1)
t1 = meansdiff/SE
print('t = meansdiff/SE = {:.3f}/{:.3f} = {:.3f}'.
      .format(meansdiff,SE,t1))
pval2 = 2*(1 - t.cdf(x=abs(t1), df=df1))
print('pval2 = 2*(1 - spst.t.cdf(x=abs(t1), df=df1))')
print("pval two and one tailed : ", pval2, pval2/2, '\n')

# P-value graphical illustration FYI-----
xv = np.arange(-4, 4, 0.001)
plt.plot(xv, t.pdf(xv,df=df1))
xL=np.arange(-4,-np.abs(t1),0.001)
plt.fill_between(xL,t.pdf(xL,df=df1),color='r')
xR=np.arange(np.abs(t1),4,0.001)
plt.fill_between(xR,t.pdf(xR,df=df1),color='b')
# -----

```

```

print("Compute confidence interval for the difference of means:")
alpha = 1-ConfLevels
tstar = t.ppf(q=1-alpha/2,df=df1)
print("tstar = t.ppf(q=1-alpha/2,df=df1)")
MarginErr = tstar*SE; # margin of error
print("MarginErr = tstar*SE")
CIL = meansdiff - MarginErr; CIR = meansdiff + MarginErr;
print("CIL = meansdiff - MarginErr; CIR = meansdiff + MarginErr; ")
df = pd.DataFrame({'ConfLevels':ConfLevels,'tstar':tstar,'SE':SE, \
                    'MarginErr':MarginErr,'x1-x2':meansdiff,'CIL':CIL,'CIR':CIR});
pd.set_option("display.precision", 4); print(df, '\n')

print("=====")

print("Compute Cohen's d:")
averagesd = (s1+s2)/2; d = meansdiff/averagesd
print('averagesd = (s1+s2)/2 = {:.3f}+{:.3f})/2 = {:.3f}' \
      .format(s1,s2,averagesd))
print("Cohen's d = meansdiff/averagesd = {:.3f}/{:.3f})/2 = {:.3f}\n" \
      .format(meansdiff,averagesd,d))

```

[ ]: TwoMeansTtests2(x1=80,s1=12,n1=16,  
       x2=84,s2=14,n2=15,ConfLevels=[0.90,0.95,0.99])

Two Independent Means T test detailed computation  
 Sample means, standard deviation, and sample size are:  
 1st Sample: x1 = 80.000, s1 = 12.000, n1 = 16  
 1st Sample: x2 = 84.000, s2 = 14.000, n2 = 15  
 Confidence Levels [0.9, 0.95, 0.99]

H0: mu1-mu2 = 0  
 H1: mu1-mu2 not 0 or one-sided test

1st sample xbar +- 2.5\*s: (50.0000, 110.0000)  
 2nd sample xbar +- 2.5\*s: (49.0000, 119.0000)

standard deviations ratios <2? = 1.1667

meansdiff = x1-x2 = 80.0000 - 84.0000 = -4.0000

=====

1st Pooled approach: -----  
 sp2 = ((n1-1)\*s1\*\*2 + (n2-1)\*s2\*\*2)/(n1+n2-2) =  
       (16-1)12.000\*\*2 + (15-1)14.000\*\*2)/(16+15-2) = 169.103  
 SE = np.sqrt(sp2\*(1/n1+1/n2)) = np.sqrt(169.103\*(1/16+1/15)) = 4.674  
 degree of freedom = df = n1+n2-2 = 16+15-2 = 29  
 t = meansdiff/SE = -4.000/4.674 = -0.856  
 pval2 = 2\*(1 - spst.t.cdf(x=abs(t1), df=df1))  
 pval two and one tailed : 0.39908551964946626 0.19954275982473313

Compute confidence interval for the difference of means:  
 tstar = t.ppf(q=1-alpha/2,df=df1)  
 MarginErr = tstar\*SE  
 CIL = meansdiff - MarginErr; CIR = meansdiff + MarginErr;

	ConflLevels	tstar	SE	MarginErr	x1-x2	CIL	CIR
0	0.90	1.6991	4.6736	7.9410	-4	-11.9410	3.9410
1	0.95	2.0452	4.6736	9.5586	-4	-13.5586	5.5586
2	0.99	2.7564	4.6736	12.8822	-4	-16.8822	8.8822

=====

2nd UNpooled approach (Welch Approximation):-----

```
SE = np.sqrt(s1^2/n1 + s2^2/n2) =
sqrt(12.000**2/16 + 14.000**2/15) = 4.698
degree of freedom = df = 27.67390755241222
t = meansdiff/SE = -4.000/4.698 = -0.852
pval2 = 2*(1 - spst.t.cdf(x=abs(t1), df=df1))
pval two and one tailed : 0.4017926906364071 0.20089634531820355
```

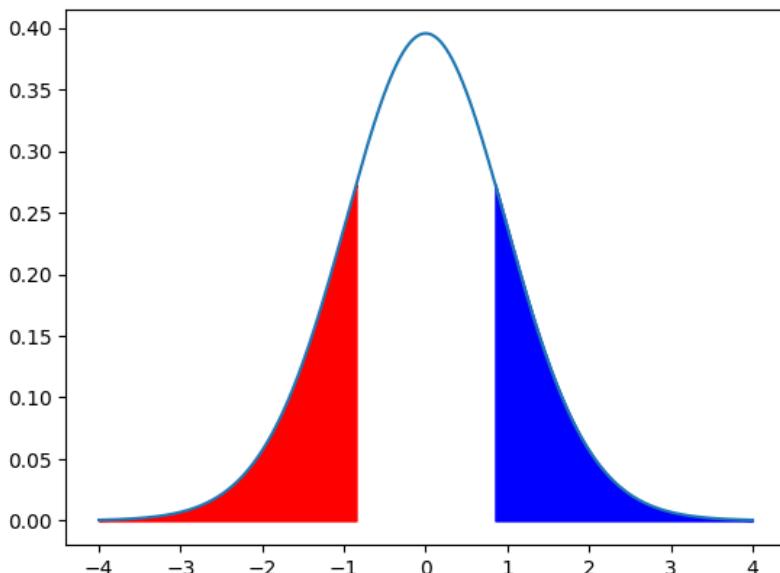
Compute confidence interval for the difference of means:

```
tstar = t.ppf(q=1-alpha/2, df=df1)
MarginErr = tstar*SE
CIL = meansdiff - MarginErr; CIR = meansdiff + MarginErr;
ConfLevels tstar SE MarginErr x1-x2 CIL CIR
0 0.90 1.7018 4.6975 7.9943 -4 -11.9943 3.9943
1 0.95 2.0495 4.6975 9.6275 -4 -13.6275 5.6275
2 0.99 2.7656 4.6975 12.9915 -4 -16.9915 8.9915
```

=====

Compute Cohen's d:

```
averagesd = (s1+s2)/2 = (12.000+14.000)/2 = 13.000
Cohen's d = meansdiff/averagesd = (-4.000/13.000)/2 = -0.308
```



The groups of men and women are independent random samples, so the independence within and between the groups should hold. We are not given min and max values to check the rule of thumb for each group, but test scores tend to be bell-shaped, so the normality could be assumed (the t-test is robust to departures from normality in any case).

The alternative  $H_1 : \mu_1 - \mu_2 \neq 0$  is two-sided, so both left and right tails must be accounted for. The **p-value** is the probability of seeing the observed difference of sample means or something even more extreme and is the sum of two tails shown in the Figure above.

In this case, the standard deviations are quite close, so the pooled approach can be used, but the results for the pooled and unpooled approaches are almost the same anyway.

The p-value  $> 0.05 = \alpha$ , so we conclude that there is *not* enough evidence to reject the initial assumption  $H_0: \mu_1 - \mu_2 = 0$ . Thus, there is no significant difference between men's and women's test scores. Also, all 95% confidence intervals contain the claimed  $H_0: \mu_1 - \mu_2 = 0$ , which confirms our conclusion *not* to reject  $H_0$ .

### Example

In this example we are given data summaries on two companies' daily pay: sample sizes 20 and 25, sample means 240 and 220, and standard deviations 26 and 12, respectively. Min and max for the 1st group are 200 and 275, and for the 2nd group 200 and 245. We would like to determine whether the difference in daily pay between the two companies is statistically significant at a 1% level.

Note that because we were told to use 1% significance level,  $\alpha$  has to be set to 0.01 (confidence level 0.99).

$$H_0: \mu_1 = \mu_2 \iff \mu_1 - \mu_2 = 0 \text{ (assumed true)}$$

$$H_1: \mu_1 \neq \mu_2 \iff \mu_1 - \mu_2 \neq 0$$

```
[ ]: TwoMeansTtests2(x1=240,s1=26,n1=20,
                      x2=220,s2=12,n2=25,ConfLevels=[0.90,0.95,0.99])
```

```
Two Independent Means T test detailed computation
Sample means, standard deviation, and sample size are:
1st Sample: x1 = 240.000, s1 = 26.000, n1 = 20
1st Sample: x2 = 220.000, s2 = 12.000, n2 = 25
Confidence Levels [0.9, 0.95, 0.99]
```

```
H0: mu1-mu2 = 0
H1: mu1-mu2 not 0 or one-sided test
```

```
1st sample xbar +- 2.5*s: (175.0000, 305.0000)
2nd sample xbar +- 2.5*s: (190.0000, 250.0000)
```

```
standard deviations ratios <2? = 2.1667
```

```

meansdiff = x1-x2 = 240.0000 - 220.0000 = 20.0000
=====
1st Pooled approach: -----
sp2 = ((n1-1)*s1**2 + (n2-1)*s2**2)/(n1+n2-2) =
(20-1)26.000**2 + (25-1)12.000**2)/(20+25-2) = 379.070
SE = np.sqrt(sp2*(1/n1+1/n2)) = np.sqrt(379.070*(1/20+1/25)) = 5.841
degree of freedom = df = n1+n2-2 = 20+25-2 = 43
t = meansdiff/SE = 20.000/5.841 = 3.424
pval2 = 2*(1 - spst.t.cdf(x=abs(t1), df=df1))
pval two and one tailed : 0.001366515043855987 0.0006832575219279935

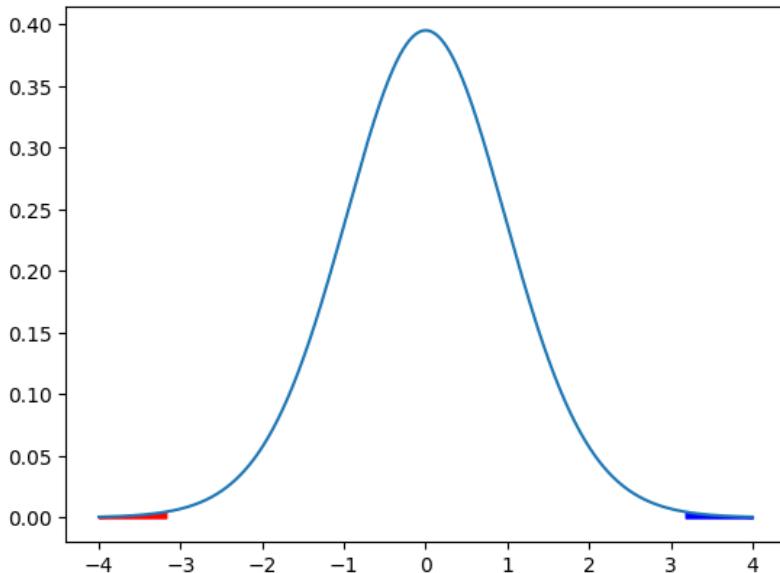
Compute confidence interval for the difference of means:
tstar = t.ppf(q=1-alpha/2,df=df1)
MarginErr = tstar*SE
CIL = meansdiff - MarginErr; CIR = meansdiff + MarginErr;
ConfLevels   tstar      SE  MarginErr  x1-x2    CIL      CIR
0            0.90    1.6811  5.8409    9.8190    20  10.1810  29.8190
1            0.95    2.0167  5.8409   11.7793    20   8.2207  31.7793
2            0.99    2.6951  5.8409   15.7419    20   4.2581  35.7419
=====

2nd UNpooled approach (Welch Approximation):-----
SE = np.sqrt(s1^2/n1 + s2^2/n2) =
sqrt(26.000**2/20 + 12.000**2/25) = 6.290
degree of freedom = df = 25.442573732854534
t = meansdiff/SE = 20.000/6.290 = 3.180
pval2 = 2*(1 - spst.t.cdf(x=abs(t1), df=df1))
pval two and one tailed : 0.0038511672711767364 0.0019255836355883682

Compute confidence interval for the difference of means:
tstar = t.ppf(q=1-alpha/2,df=df1)
MarginErr = tstar*SE
CIL = meansdiff - MarginErr; CIR = meansdiff + MarginErr;
ConfLevels   tstar      SE  MarginErr  x1-x2    CIL      CIR
0            0.90    1.7070  6.2897   10.7365    20  9.2635  30.7365
1            0.95    2.0577  6.2897   12.9424    20  7.0576  32.9424
2            0.99    2.7835  6.2897   17.5072    20  2.4928  37.5072
=====

Compute Cohen's d:
averagesd = (s1+s2)/2 = (26.000+12.000)/2 = 19.000
Cohen's d = meansdiff/averagesd = (20.000/19.000)/2 = 1.053

```



The groups of employees from two firms are assumed to be independent random samples, so the independence within and between the groups holds. The min and max for each group are within  $\bar{x} \pm 2.5s$  bounds, so the observations are not too extreme and normality could be assumed (t-test is robust to departures from normality in any case).

The alternative  $H_1 : \mu_1 - \mu_2 \neq 0$  is two-sided, so both left and right tails must be accounted for. The **p-value** is the probability of seeing the observed difference of sample means or something even more extreme and is the sum of two tails shown in the Figure above.

This time, the standard deviations ratio is more than 2 and the pooled approach should not be used. We can also see in the final output that its results are quite different. The conclusions are still the same though. The  $pvalue < 0.01 = \alpha$ , so we conclude that there is enough evidence to reject the initial assumption  $H_0: \mu_1 - \mu_2 = 0$ . Thus, there is significant difference between groups. Also, 99% confidence interval does NOT contain the claimed 0, which confirms our conclusion to reject  $H_0$ .

### Example

Consider again the HELPrct file. Compare mean cesd depression scores by gender. We would like to test whether there is a significant difference at the 1% level.

This time we have an actual data file, so in the Figure below we explore histogram and boxplot separated by gender. Also, Levene's test is used to assess equality of group variances.

$$H_0 : \mu_1 = \mu_2 \iff \mu_1 - \mu_2 = 0 \text{ (assumed true)}$$

$$H_1 : \mu_1 \neq \mu_2 \iff \mu_1 - \mu_2 \neq 0$$

```
[ ]: import pandas as pd; import numpy as np; from scipy import stats
import matplotlib.pyplot as plt; import seaborn as sns

url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrc.csv"
mydata = pd.read_csv(url) # save as mydata file
mydata[['cesd','sex']].head()
```

```
[ ]:      cesd      sex
0       49    male
1       30    male
2       39    male
3       15  female
4       39    male
```

```
[ ]: data1 = mydata.loc[mydata['sex']=='female']['cesd'] # 1st variable
data2 = mydata.loc[mydata['sex']=='male']['cesd'] # 2nd variable

print("Results from Levene's test to check equality of variance assumption:")
print(stats.levene(data1, data2), '\n')

print("Independent Samples T-test built in function (assume equal variances):
→")
print(stats.ttest_ind(data1, data2, equal_var=True), '\n')
print("Independent Samples T-test built in function (NOT assume equal_
→variances)")
print(stats.ttest_ind(data1, data2, equal_var=False), '\n')

print("Results from built in Non-parametric Mann Whitney function:")
print(stats.mannwhitneyu(data1, data2), '\n')

fig, axs = plt.subplots(1, 2, figsize=(7, 3))
sns.histplot(data=mydata, x="cesd", hue="sex", ax=axs[0])
sns.boxplot(data=mydata, x="cesd", y="sex", ax=axs[1])
```

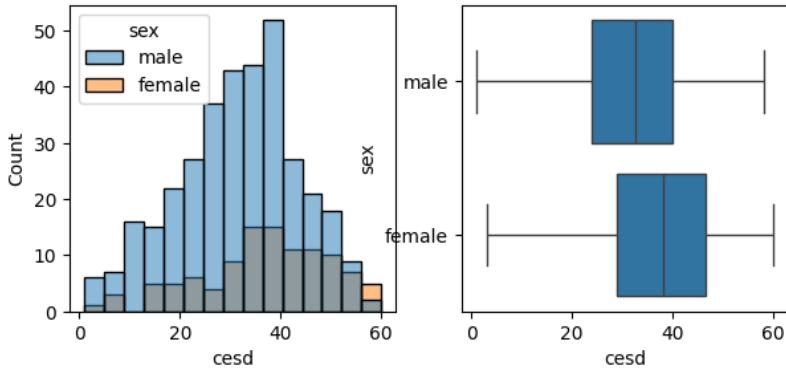
Results from Levene's test to check equality of variance assumption:  
LeveneResult(statistic=0.37376603677822684, pvalue=0.5412666039580007)

Independent Samples T-test built in function (assume equal variances):  
TtestResult(statistic=3.8800707068561255, pvalue=0.00011999508048677403,  
df=451.0)

Independent Samples T-test built in function (NOT assume equal variances)  
TtestResult(statistic=3.733669101195794, pvalue=0.00025872584417343473,  
df=166.59197910731234)

Results from built in Non-parametric Mann Whitney function:  
MannwhitneyuResult(statistic=23105.0, pvalue=0.00010334835267409345)

```
[ ]: <Axes: xlabel='cesd', ylabel='sex'>
```



```
[ ]: x1 = np.mean(data1); s1 = np.std(data1,ddof=1); n1 = len(data1);
x2 = np.mean(data2); s2 = np.std(data2,ddof=1); n2 = len(data2);
TwoMeansTtests2(x1=x1,s1=s1,n1=n1,
                 x2=x2,s2=s2,n2=n2,ConfLevels=[0.90,0.95,0.99])
```

Two Independent Means T test detailed computation  
 Sample means, standard deviation, and sample size are:  
 1st Sample:  $x_1 = 36.888$ ,  $s_1 = 13.018$ ,  $n_1 = 107$   
 1st Sample:  $x_2 = 31.598$ ,  $s_2 = 12.103$ ,  $n_2 = 346$   
 Confidence Levels [0.9, 0.95, 0.99]

H0:  $\mu_1 - \mu_2 = 0$

H1:  $\mu_1 - \mu_2 \neq 0$  or one-sided test

1st sample xbar + 2.5\*s: (4.3437, 69.4320)  
 2nd sample xbar + 2.5\*s: (1.3400, 61.8566)

standard deviations ratios <2? = 1.0755

meansdiff =  $x_1 - x_2 = 36.8879 - 31.5983 = 5.2896$

=====

1st Pooled approach: -----

$sp^2 = ((n_1-1)*s_1^2 + (n_2-1)*s_2^2)/(n_1+n_2-2) = (107-1)13.018^2 + (346-1)12.103^2/(107+346-2) = 151.889$

$SE = \sqrt{sp^2/(n_1+n_2)} = \sqrt{151.889/(107+346)} = 1.363$

degree of freedom = df =  $n_1+n_2-2 = 107+346-2 = 451$

t = meansdiff/SE =  $5.290/1.363 = 3.880$

pval2 =  $2*(1 - spst.t.cdf(x=abs(t1), df=df1))$

pval two and one tailed : 0.00011999508048665675 5.999754024332837e-05

Compute confidence interval for the difference of means:

tstar = t.ppf(q=1-alpha/2,df=df1)

MarginErr = tstar\*SE

CIL = meansdiff - MarginErr; CIR = meansdiff + MarginErr;

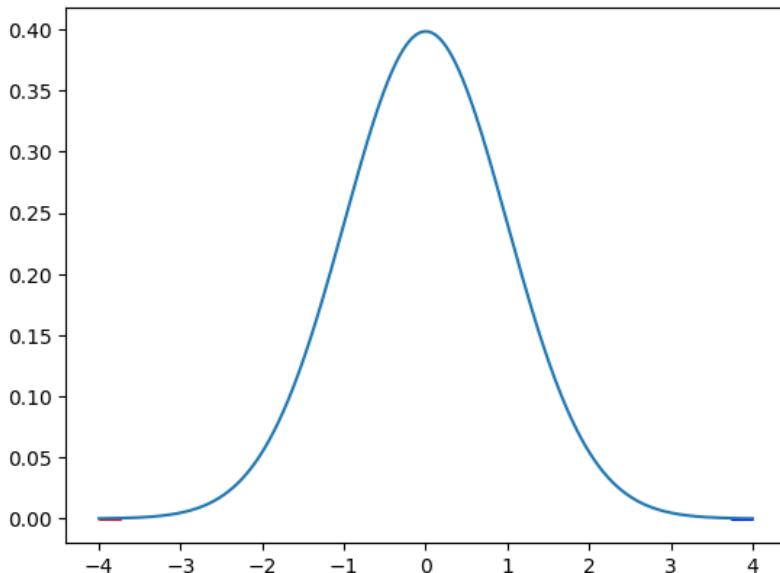
```

ConfLevels   tstar      SE  MarginErr    x1-x2      CIL      CIR
0           0.90  1.6482  1.3633     2.2470  5.2896  3.0426  7.5366
1           0.95  1.9652  1.3633     2.6792  5.2896  2.6104  7.9687
2           0.99  2.5868  1.3633     3.5265  5.2896  1.7631  8.8161
=====
2nd UNpooled approach (Welch Approximation):
SE = np.sqrt(s1^2/n1 + s2^2/n2) =
sqrt(13.018**2/107 + 12.103**2/346) = 1.417
degree of freedom = df = 166.5919791073123
t = meansdiff/SE = 5.290/1.417 = 3.734
pval2 = 2*(1 - spst.t.cdf(x=abs(t1), df=df1))
pval two and one tailed : 0.000258725844173302 0.000129362922086651

Compute confidence interval for the difference of means:
tstar = t.ppf(q=1-alpha/2,df=df1)
MarginErr = tstar*SE
CIL = meansdiff - MarginErr;    CIR = meansdiff + MarginErr;
ConfLevels   tstar      SE  MarginErr    x1-x2      CIL      CIR
0           0.90  1.6541  1.4167     2.3433  5.2896  2.9462  7.6329
1           0.95  1.9743  1.4167     2.7971  5.2896  2.4925  8.0866
2           0.99  2.6057  1.4167     3.6915  5.2896  1.5981  8.9811
=====

Compute Cohen's d:
averagesd = (s1+s2)/2 = (13.018+12.103)/2 = 12.560
Cohen's d = meansdiff/averagesd = (5.290/12.560)/2 = 0.421

```



Female and male substance abusers can be assumed to be independent random samples, so the independence within and between the groups holds. The Figure above shows histograms and boxplots for males and females. The graphs are not particularly skewed, with no extreme outliers, so we can assume normality. The sample sizes are so large that CLT guarantees the normality of sample means in any case. We also look at the non-parametric test just to compare.

Because Levene's test shows a p-value much higher than 0.05, equal variance can be assumed, and the pooled approach is justified. The pooled and unpooled approaches give the same answer anyway.

The  $pvalue < 0.01 = \alpha$ , so we conclude that there is enough evidence to reject the initial assumption  $H_0: \mu_1 - \mu_2 = 0$ . There is a significant difference between the depression scores by gender. Also, 99% confidence interval does *not* contain the claimed 0, which confirms our conclusion to reject  $H_0$ .

### Example

In this example, we would like to determine if there is a significant difference in the average prices of concert tickets in Kansas City and Salt Lake City. We sample 11 ticket stubs from Kansas City and 10 from Salt Lake City. Test the claim using a 0.05 level of significance. Note that Welch approximation is used!

This time, the data set is small and is best entered on the fly into `pd.DataFrame`. The Figure below shows histograms and boxplots separated by city. Levene's test is used to assess equality of group variances. Note how the data is entered in a long format: all prices are entered in the variable `price`, and the categorical variable `city` identifies which city it is from.

$$H_0: \mu_1 = \mu_2 \iff \mu_1 - \mu_2 = 0 \text{ (assumed true)}$$

$$H_1: \mu_1 \neq \mu_2 \iff \mu_1 - \mu_2 \neq 0$$

```
[1]: import pandas as pd; import numpy as np; from scipy import stats
import matplotlib.pyplot as plt; import seaborn as sns

mydata = pd.DataFrame({'price':[7, 8, 9, 6, 7, 7, 8, 9, 7, 7, 6, 13, 11, 16,
                                9, 14, 10, 9, 11, 8, 14],
                       'city': ['KC', 'KC', 'KC', 'KC', 'KC', 'KC', 'KC', 'KC', 'KC',
                                'KC', 'KC', 'SL', 'SL', 'SL', 'SL', 'SL', 'SL',
                                'SL', 'SL', 'SL']})
```

```
[2]: data1 = mydata.loc[mydata['city']=='KC']['price']      # 1st variable
data2 = mydata.loc[mydata['city']=='SL']['price']      # 2nd variable

print("Results from Levene's test to check equality of variance assumption:")
print(stats.levene(data1, data2), '\n')

print("Independent Samples T-test built in function (NOT assume equal
      variances)")
print(stats.ttest_ind(data1, data2, equal_var=False), '\n')
```

```

print("Results from built in Non-parametric Mann Whitney function:")
print(stats.mannwhitneyu(data1, data2), '\n')

fig, axs = plt.subplots(1, 2, figsize=(7, 3))
sns.histplot(data=mydata,x="price",hue="city", ax=axs[0])
sns.boxplot(data=mydata,x="price",hue="city", ax=axs[1])

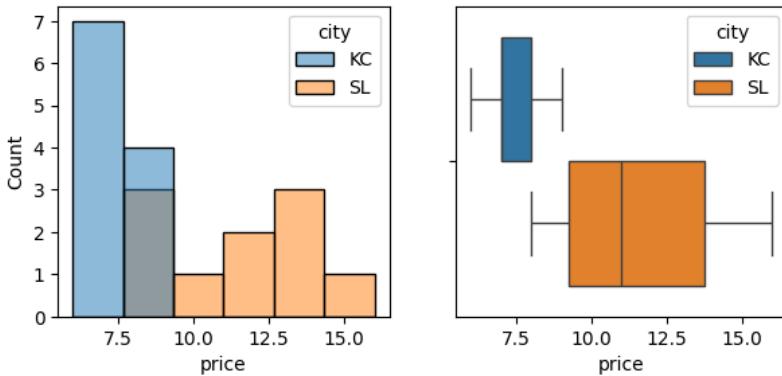
```

Results from Levene's test to check equality of variance assumption:  
LeveneResult(statistic=6.9249668313112425, pvalue=0.01643892458076757)

Independent Samples T-test built in function (NOT assume equal variances)  
TtestResult(statistic=-4.65285335326208, pvalue=0.0006297276542999839,  
df=11.459853276011748)

Results from built in Non-parametric Mann Whitney function:  
MannwhitneyuResult(statistic=5.0, pvalue=0.0004181928783958954)

[ ]: <Axes: xlabel='price'>



[ ]: x1 = np.mean(data1); s1 = np.std(data1,ddof=1); n1 = len(data1);
x2 = np.mean(data2); s2 = np.std(data2,ddof=1); n2 = len(data2);
TwoMeansTtests2(x1=x1,s1=s1,n1=n1,
x2=x2,s2=s2,n2=n2,ConfLevels=[0.90,0.95,0.99])

Two Independent Means T test detailed computation  
Sample means, standard deviation, and sample size are:  
1st Sample: x1 = 7.364, s1 = 1.027, n1 = 11  
1st Sample: x2 = 11.500, s2 = 2.635, n2 = 10  
Confidence Levels [0.9, 0.95, 0.99]

H0:  $\mu_1 - \mu_2 = 0$   
H1:  $\mu_1 - \mu_2 \neq 0$  or one-sided test

1st sample xbar +- 2.5\*s: (4.7964, 9.9309)

```

2nd sample xbar -+ 2.5*s: (4.9119, 18.0881)

standard deviations ratios <2? = 2.5662

meansdiff = x1-x2 = 7.3636 - 11.5000 = -4.1364

=====
1st Pooled approach: -----
sp2 = ((n1-1)*s1**2 + (n2-1)*s2**2)/(n1+n2-2) =
    (11-1)1.027**2 + (10-1)2.635**2)/(11+10-2) = 3.844
SE = np.sqrt(sp2*(1/n1+1/n2)) = np.sqrt(3.844*(1/11+1/10)) = 0.857
degree of freedom = df = n1+n2-2 = 11+10-2 = 19
t = meansdiff/SE = -4.136/0.857 = -4.828
pval2 = 2*(1 - spst.t.cdf(x=abs(t1), df=df1))
pval two and one tailed : 0.00011681853580802759 5.8409267904013795e-05

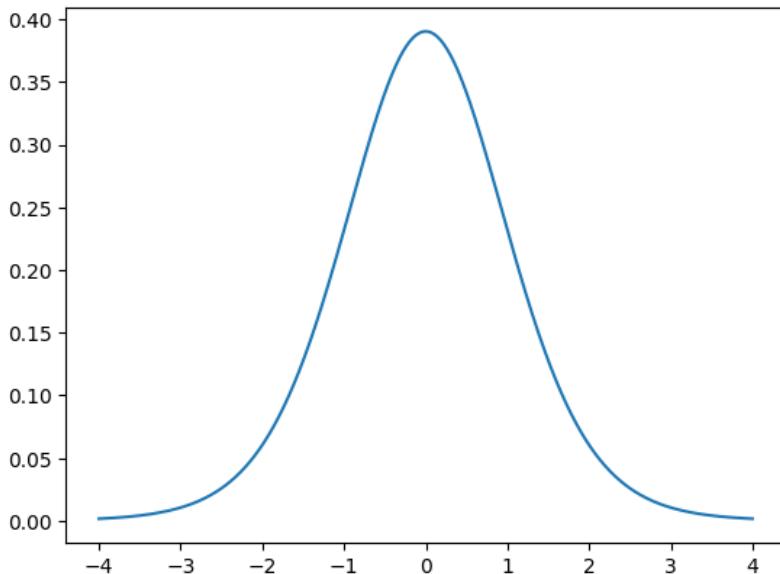
Compute confidence interval for the difference of means:
tstar = t.ppf(q=1-alpha/2,df=df1)
MarginErr = tstar*SE
CIL = meansdiff - MarginErr; CIR = meansdiff + MarginErr;
    ConfLevels   tstar      SE  MarginErr   x1-x2      CIL      CIR
0        0.90  1.7291  0.8567      1.4814 -4.1364 -5.6177 -2.6550
1        0.95  2.0930  0.8567      1.7931 -4.1364 -5.9295 -2.3433
2        0.99  2.8609  0.8567      2.4510 -4.1364 -6.5874 -1.6854

=====
2nd UNpooled approach (Welch Approximation):-----
SE = np.sqrt(s1^2/n1 + s2^2/n2) =
    sqrt(1.027**2/11 + 2.635**2/10) = 0.889
degree of freedom = df = 11.459853276011748
t = meansdiff/SE = -4.136/0.889 = -4.653
pval2 = 2*(1 - spst.t.cdf(x=abs(t1), df=df1))
pval two and one tailed : 0.0006297276542999164 0.0003148638271499582

Compute confidence interval for the difference of means:
tstar = t.ppf(q=1-alpha/2,df=df1)
MarginErr = tstar*SE
CIL = meansdiff - MarginErr; CIR = meansdiff + MarginErr;
    ConfLevels   tstar      SE  MarginErr   x1-x2      CIL      CIR
0        0.90  1.7893  0.889      1.5907 -4.1364 -5.7271 -2.5457
1        0.95  2.1903  0.889      1.9471 -4.1364 -6.0835 -2.1892
2        0.99  3.0809  0.889      2.7389 -4.1364 -6.8753 -1.3974

=====
Compute Cohen's d:
averagesd = (s1+s2)/2 = (1.027+2.635)/2 = 1.831
Cohen's d = meansdiff/averagesd = (-4.136/1.831)/2 = -2.259

```



The two cities' ticket prices are independent random samples, so independence within and between the groups is maintained. The Figure above shows a histogram and boxplot by group. The graphs are not extremely skewed, with no extreme outliers. We also looked at the non-parametric test to compare.

Because Levene's test shows a p-value smaller than 0.05, equal variance cannot be assumed, and a pooled approach should not be used.

Welch approximation gives two-sided  $pvalue < 0.01 = \alpha$ , so we conclude that there is enough evidence to reject the initial assumption  $H_0: \mu_1 - \mu_2 = 0$ . There is a significant difference between the prices in the two cities. Also, 99% confidence interval does *not* contain the claimed 0, which confirms our conclusion to reject  $H_0$ .



Taylor & Francis  
Taylor & Francis Group  
<http://taylorandfrancis.com>

# 8

---

## Correlation and Regression

---

In this chapter, we introduce methods for determining whether there is a linear correlation, or association, between two numerical variables. In the case it exists, we learn how to derive a linear model and how to use it for prediction, which is very important in applications of many different fields from Business and Finance to Biology and Medicine. For example, based on income, consumer debt, and several other predictors about a person, we might be interested in predicting a person's credit rating. Or we might be interested in predicting growth in sales of a company based on current economic conditions and its previous year's performance. In Medicine, it would be important to predict patients' blood pressure based on health measurements and the dosage of a new drug. In fairness, the more interesting examples above have several predictors, which is called multiple regression and is taken up in detail in later chapters. In this chapter, we study correlation in general, but the linear models are limited to a single predictor.

---

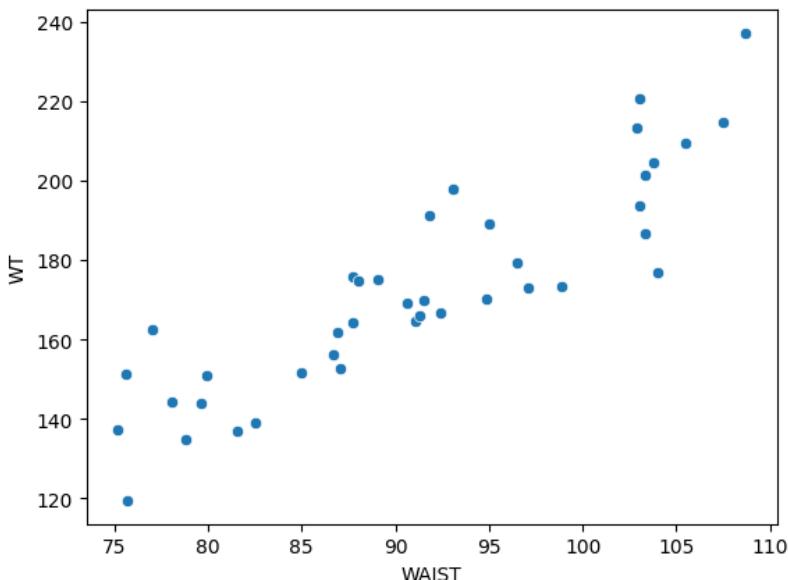
### 8.1 Correlation

As an example, let's return to the file `MHEALTH.csv` mentioned before in [chapters 1](#) and [2](#). It contains many health measurements for 40 men. Let's investigate weight (WT) dependence on the waist size (WAIST). Whenever you plan a correlation/regression study, you *must always start with a scatterplot* as shown in the Figure below.

```
[ ]: import pandas as pd; import numpy as np
import seaborn as sns; import matplotlib.pyplot as plt

url="https://raw.githubusercontent.com/leonkag/Statistics0/main/MHEALTHs.csv"
mydata = pd.read_csv(url)    # save as mydata file
# mydata.head(10)
sns.scatterplot(data=mydata, x="WAIST", y="WT")
```

```
[ ]: <Axes: xlabel='WAIST', ylabel='WT'>
```



There is a fair amount of variation, but there seems to be an overall linear trend with a positive slope. Not surprisingly, a man with a larger waist size has a higher weight. The numerical measure of the strength of this *linear trend* is given by the **Pearson correlation coefficient**:

$$r = \frac{1}{n-1} \sum_{i=1}^n (z_x)_i \cdot (z_y)_i = \frac{1}{n-1} \sum_{i=1}^n \frac{x_i - \bar{x}}{s_x} \cdot \frac{y_i - \bar{y}}{s_y} \quad (8.1)$$

where  $n$  is the number of data point pairs. The `scipy.stats` library has an effective function to perform this rather tedious computation.

```
[ ]: from scipy.stats import pearsonr
r = pearsonr(mydata.WAIST,mydata.WT);
print("Pearson correlation coefficient r and corresponding p-value:")
print(r)
```

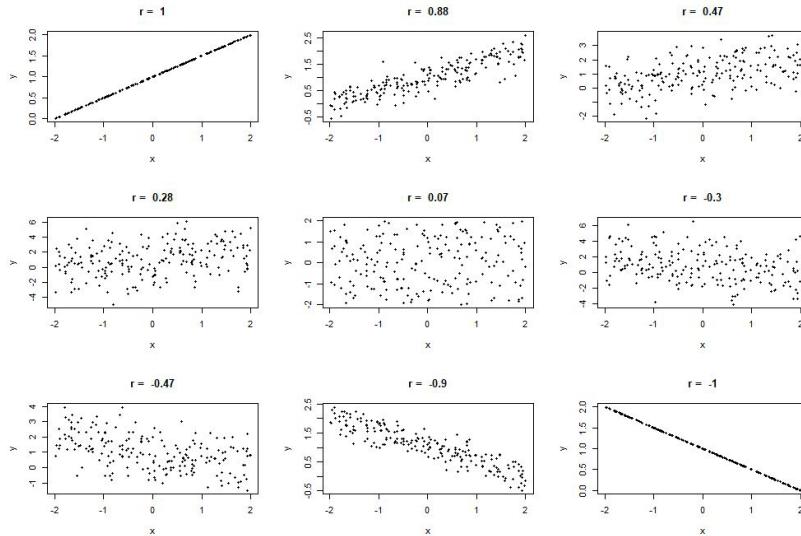
```
Pearson correlation coefficient r and corresponding p-value:
PearsonRResult(statistic=0.8889256283970652, pvalue=1.8716847367134364e-14)
```

Note that because the standardized z-scores are used to define  $r$ , it is a dimensionless number that does not depend on units of measurement of either variable. For example, assume that  $x$  is the square footage ( $ft^2$ ) of an apartment, and  $y$  is its price in dollars. If we change  $x$  to square meters and/or  $y$  to thousands of dollars, the correlation coefficients would not change. Also,  $x$  and  $y$  come symmetrically into the  $r$  formula, so they can be interchanged.

The correlation coefficient is always between  $-1 \leq r \leq 1$  and the Figure below illustrates different sample scatterplots and their correlations.

```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main/
→CorrRcompare.JPG'
page = requests.get(url); Image.open(BytesIO(page.content))
```

[ ]:



A set of heuristic bounds for the correlation coefficient could be defined as follows:

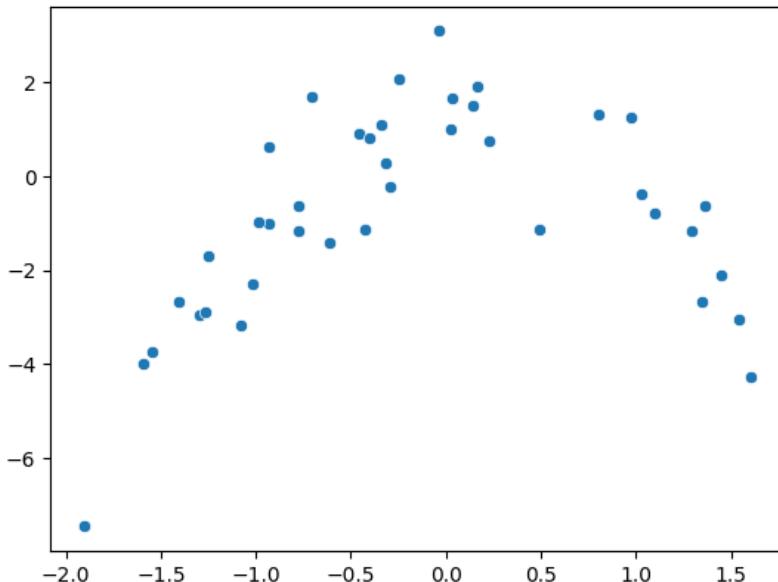
1.  $0 \leq |r| \leq 0.3$  - no correlation to weak correlation
2.  $0.3 \leq |r| \leq 0.6$  - weak correlation to moderate correlation
3.  $0.6 \leq |r| \leq 1$  - moderate correlation to strong correlation.

These bounds are imprecise and very much depend on the field of study. In a science or an engineering experiment, when the variables are related by a Physics law, a correlation must be very high, close to -1 or 1. On the other hand, in business, economics, medicine, etc. with all the variability in human behavior, even a correlation of 0.6 could be considered strong. In the figure above, top to bottom, left to right, the correlation coefficient decreases from perfect positive to strong, moderate, weak, no correlation, and similarly for negative correlations. Note that Pearson correlation coefficient  $r$  only quantifies **linear** correlation. It **can't** describe any **non-linear** relationship. The  $r$  could still be computed, but it does not make much sense as shown in the Figure below.

```
[ ]: n1 = 40; a = 1; b= 0.5; c = -2; sig = 1;
x = np.random.uniform(low=-2, high=1.75, size=n1)
y = a+b*x+c*x**2+ np.random.normal(loc=0.0, scale=1.0, size=n1)
from scipy.stats import pearsonr
r = pearsonr(x,y);
print("Pearson correlation coefficient r and corresponding p-value:")
print(r)
sns.scatterplot(x=x,y=y)
```

Pearson correlation coefficient r and corresponding p-value:  
PearsonRResult(statistic=0.2536672308873064, pvalue=0.11424060228942981)

```
[ ]: <Axes: >
```



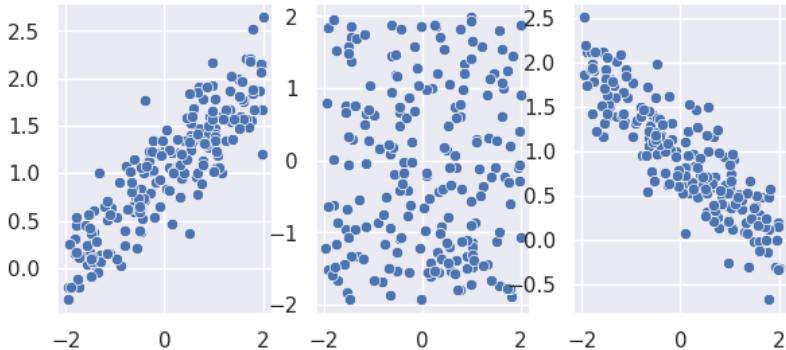
The formula for the correlation coefficient  $r = \frac{1}{n-1} \sum_{i=1}^n (z_x)_i \cdot (z_y)_i$ , and the Figure below can help understand the sign and magnitude of the correlation.

```
[ ]: n1 = 200; a = 1; b= 0.5; sig = 0.3;
x = np.random.uniform(low=-2, high=2, size=n1)
y = a + b*x + np.random.normal(loc=0.0, scale=sig, size=n1)
from scipy.stats import pearsonr
r = pearsonr(x,y);
print("Pearson correlation coefficient r and corresponding p-value:")
print(r)
fig, axs = plt.subplots(ncols=3, figsize=(7,3))
sns.scatterplot(x=x,y=y, ax = axs[0])
y = np.random.uniform(low=-2, high=2, size=n1)
```

```
r = pearsonr(x,y);
print("Pearson correlation coefficient r and corresponding p-value:")
print(r)
sns.scatterplot(x=x,y=y, ax = axs[1])
y = a - b*x + np.random.normal(loc=0.0, scale=sig, size=n1)
r = pearsonr(x,y);
print("Pearson correlation coefficient r and corresponding p-value:")
print(r)
sns.scatterplot(x=x,y=y, ax = axs[2])
```

Pearson correlation coefficient r and corresponding p-value:  
PearsonRResult(statistic=0.8813611654172478, pvalue=2.1305371897896783e-66)  
Pearson correlation coefficient r and corresponding p-value:  
PearsonRResult(statistic=-0.002099667931854406, pvalue=0.9764596789551836)  
Pearson correlation coefficient r and corresponding p-value:  
PearsonRResult(statistic=-0.8822726517735485, pvalue=1.0405663967474857e-66)

[ ]: <Axes: >



For the positive slope (uphill) line,  $z_x$  and  $z_y$  are either both  $> 0$  or both  $< 0$ ; therefore the  $\sum z_x \cdot z_y$  is positive and large. Analogously, for the negative slope (downhill) line,  $z_x$  and  $z_y$  have opposite signs, so  $\sum z_x \cdot z_y$  is negative and large. For the points in the middle plot, which are all over the place,  $z_x \cdot z_y$  have different signs and cancel each other out, so  $\sum z_x \cdot z_y$  is close to 0. This sum is divided by  $n - 1$  to average, which leads to bounds  $-1 \leq r \leq 1$ .

The sample correlation coefficient  $r$  is an approximation of the true population correlation coefficient  $\rho$ . The hypothesis test for  $\rho$  is:

H<sub>0</sub>:  $\rho = 0$  - no linear correlation

H<sub>1</sub>:  $\rho \neq 0$  - there is a linear correlation

The hypothesis is tested with t-distribution with  $df = n - 2$  degrees of freedom:

$$SE = \sqrt{\frac{1 - r^2}{n - 2}} \quad (8.2)$$

$$t = \frac{(point\ estimate) - (null\ value)}{SE} = \frac{r - 0}{SE} = \frac{r}{SE} \quad (8.3)$$

Note that the Pearson's r hypothesis test is a parametric test, and the data must satisfy the usual assumptions that both variables follow normal distributions. At least no outliers should dramatically change r. Such outliers are influential observations, which will be discussed later in this chapter.

For example, for WT and WAIST size in the MHEALTH data file, the above formulas give:

```
[ ]: from scipy import stats
r = stats.pearsonr(mydata['WAIST'],mydata['WT']);
print("Pearson correlation coefficient r and corresponding p-value:")
print(r)
# step by step calculation of r to confirm
x = mydata['WAIST']; y = mydata['WT'];
xbar = np.mean(x); ybar = np.mean(y);
sx = np.std(x,ddof=1); sy = np.std(y,ddof=1); n = len(x)
print('xbar = {:.3f}, ybar = {:.3f}, sx = {:.3f}, sy = {:.3f}, n = {:d}'
      .format(xbar, ybar, sx, sy, n))
zx = (x-xbar)/sx; zy = (y-ybar)/sy; r = sum(zx*zy)/(n-1);
SE = np.sqrt((1-r**2)/(n-2)); t1 = r/SE;
from scipy.stats import t
pval2 = 2*(1 - t.cdf(np.abs(t1), df=n-2));
print('r = {:.3f}, SE = {:.3f}, t = {:.3f}, pval2 = {:.3e}'
      .format(r, SE, t1, pval2))
```

```
Pearson correlation coefficient r and corresponding p-value:
PearsonRResult(statistic=0.8889256283970652, pvalue=1.8716847367134364e-14)
xbar = 91.285, ybar = 172.550, sx = 9.862, sy = 26.327, n = 40
r = 0.889, SE = 0.074, t = 11.963, pval2 = 1.865e-14
```

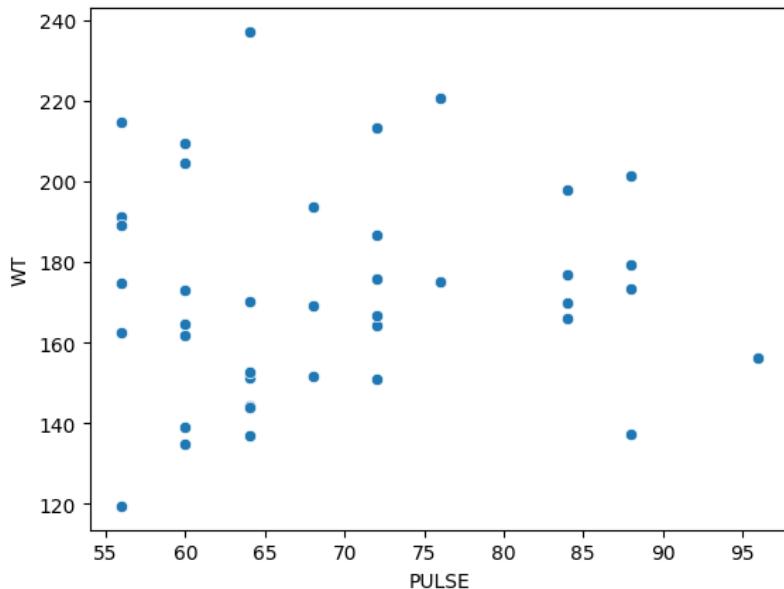
Therefore, WAIST and WT are strongly positively linearly correlated. Also, p-value < 0.05, so we can reject H0:  $\rho = 0$  (no linear correlation) to claim that this correlation is significant.

On the other hand, for WT and PULSE, there is practically no correlation and p-value > 0.05 (non-significant):

```
[ ]: print("Pearson correlation coefficient between PULSE and WT: ")
print(pearsonr(mydata['PULSE'],mydata['WT']),'\n')
sns.scatterplot(data=mydata, x="PULSE", y="WT")
```

```
Pearson correlation coefficient between PULSE and WT:
PearsonRResult(statistic=0.055829081775840124, pvalue=0.7322270590212745)
```

```
[ ]: <Axes: xlabel='PULSE', ylabel='WT'>
```



Note that the strength of the correlation is given by the magnitude of  $r$ , while the significance is determined by the p-value of the corresponding t-test. They are related, but not the same. Say, for a large data set, even a weak correlation coefficient can be statistically significant.

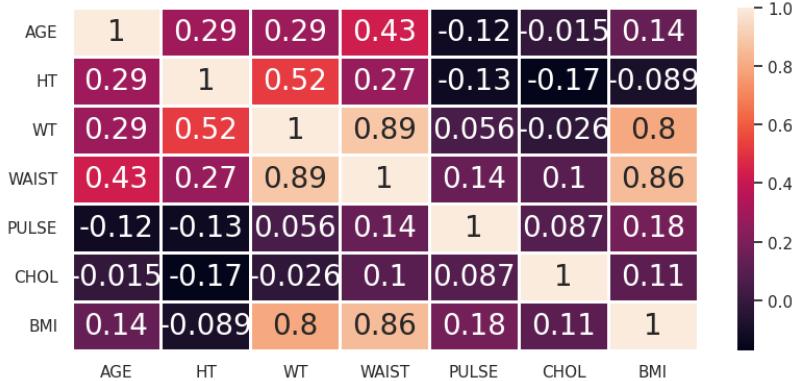
Many possible pairwise correlation coefficients exist in the case of several interrelated numerical variables. The code below gives an efficient way to obtain all the pairwise correlations at once and have their visual representation.

```
[ ]: r = mydata[['AGE','HT','WT','WAIST','PULSE','CHOL','BMI']].corr(method='pearson'); r
```

	AGE	HT	WT	WAIST	PULSE	CHOL	BMI
AGE	1.000000	0.288736	0.285481	0.428930	-0.121024	-0.015371	0.137197
HT	0.288736	1.000000	0.522246	0.268548	-0.132260	-0.171151	-0.089097
WT	0.285481	0.522246	1.000000	0.888926	0.055829	-0.025909	0.799659
WAIST	0.428930	0.268548	0.888926	1.000000	0.141870	0.102332	0.855092
PULSE	-0.121024	-0.132260	0.055829	0.141870	1.000000	0.087083	0.180831
CHOL	-0.015371	-0.171151	-0.025909	0.102332	0.087083	1.000000	0.110292
BMI	0.137197	-0.089097	0.799659	0.855092	0.180831	0.110292	1.000000

```
[ ]: sns.set(rc={'figure.figsize':(9.7,4.27)})  
sns.heatmap(r,linewidth=1,annot=True,annot_kws={'size':20})
```

[ ]: <Axes:



Therefore, WT is weakly correlated with AGE, not correlated with CHOL (cholesterol), moderately correlated with HT (height), and strongly positively correlated with BMI. We can check individual significance with p-value, for example for WT and HT:

```
[1]: pearsonr(mydata['HT'],mydata['WT'])

[2]: PearsonRResult(statistic=0.5222462687506603, pvalue=0.000547057356449642)
```

Moreover, the square of  $r - r^2$  is called the **coefficient of determination** and has an important meaning. It is the proportion of the variation in the response variable  $y$  that is explained by the linear relationship between  $x$  and  $y$ .

```
[3]: r, pval = pearsonr(mydata['HT'],mydata['WT'])
print('r = {:.3f}, pval = {:.10f}, r-squared = {:.3f}'.format(r, pval, r**2))
```

$r = 0.522, pval = 0.0005470574, r-squared = 0.273$

For example, for the relationship between HT and WT,  $r^2 = 0.522^2 = 0.273$ . Therefore 27.3% of variation in WT can be explained by the linear relationship with HT. On the other hand, for the relationship between WT and PULSE,  $r^2 = 0.056^2 = 0.0031$ . Therefore, only 0.31% of variation in WT can be explained with the linear relationship with PULSE (no relationship).

To repeat the point that was made in the beginning, **correlation does NOT imply causation**. Only a randomized experiment can be used to assess the causality!

### 8.1.1 Spearman Correlation

In the case when the samples are small and/or skewed, the normality cannot be assumed, and non-parametric tests that make no such assumptions are in order. Spearman rank-order correlation can evaluate the monotonic relationship between two continuous or ordinal variables. In such a relationship, the variables tend to change together, but not necessarily at a constant rate. Much like other non-parametric tests (Wilcoxon, Mann-Whitney, etc.), the Spearman correlation coefficient is based on the ranked values rather than the raw data. As a ranking, it can also be naturally used for ordinal variables like the Lekert Scale (1 - Strongly Disagree, 2 - Disagree, 3 - Neutral, 4 - Agree, 5 - Strongly Agree).

As an example, consider a data file on a possible correlation between a number of medications a person took in the past week (discrete counting variable, not normally distributed) and the score on a standard memory recall test. We expect that the greater the number of medications, the worse the patients' memory will be; therefore a one-tailed test is in order. The data set is small.

```
[ ]: mydata = pd.DataFrame({'NumberMeds':[3,4,1,0,3,1,1,3,5,7,3,2,2,0],
                           'TestScore':
                           →[16,19,12,14,16,16,11,11,10,11,13,16,20,14]})

from scipy import stats
r, pval = stats.spearmanr(mydata['NumberMeds'],mydata['TestScore'])
print('r = {:.3f}, pval = {:.10f}, pval/2 = {:.10f}, r-squared = {:.3f}'.
      format(r, pval, pval/2, r**2))
r = mydata.corr(method='spearman'); r
```

```
r = -0.163, pval = 0.5784273124, pval/2 = 0.2892136562, r-squared = 0.026
```

```
[ ]:          NumberMeds  TestScore
NumberMeds      1.000000   -0.162685
TestScore      -0.162685    1.000000
```

The correlation coefficient is negative (as predicted), but it is weak and not statistically significant at any acceptable level of significance. Note that we have to divide the p-value which we obtained by 2 (one-sided test), but it is still way more than 0.05. Thus, the number of medications was not significantly related to memory performance.

Kendall's rank correlation tests the similarities in the ordering of data ranked by quantities. Pearson and Spearman use the individual observations as the basis of the correlation, while Kendall's correlation coefficient uses pairs of observations. It determines the strength of association based on the pattern of concordance and discordance between the pairs. For the medication problem above, the conclusions are the same.

```
[ ]: r, pval = stats.kendalltau(mydata['NumberMeds'],mydata['TestScore'])
print('r = {:.3f}, pval = {:.10f}, pval/2 = {:.10f}, r-squared = {:.3f}'.
      format(r, pval, pval/2, r**2))
```

```
r = mydata.corr(method='kendall'); r
```

```
r = -0.099, pval = 0.6481060155, pval/2 = 0.3240530077, r-squared = 0.010
```

```
[ ]:      NumberMeds  TestScore
NumberMeds      1.000000   -0.099381
TestScore      -0.099381    1.000000
```

### 8.1.2 Partial Correlation

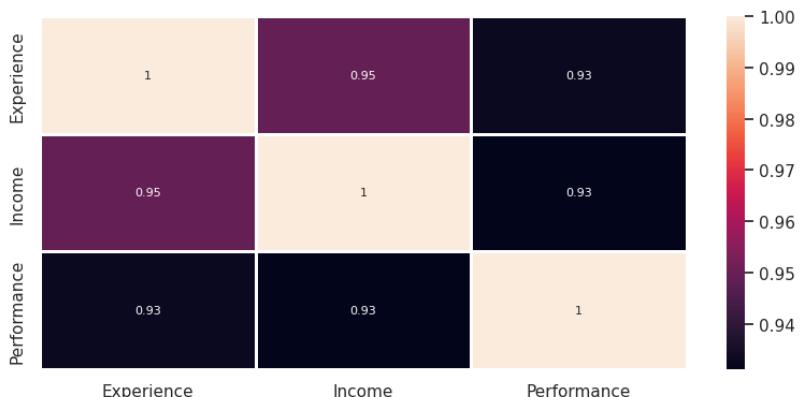
Partial correlation is the correlation between two variables while controlling for a third (or more) variable. Consider an example data set of `Income` and `Performance` scores at a job together with `Experience` in years. As we can see in the correlation matrix Figure below, the correlations computed without partialling anything out are very strong between all three variables.

```
[ ]: mydata = pd.DataFrame({'Experience':[1,1,2,2,4,4,7,7,10,10,12,12,12,20],
                           'Income':[50,51,55,53,61,62,70,74,89,90,91,90,90,98],
                           'Performance':
                           [16,19,21,24,20,21,25,24,28,30,31,32,28,35]})
r = mydata.corr(method='pearson'); r
```

```
[ ]:      Experience     Income     Performance
Experience  1.000000  0.949088  0.933057
Income      0.949088  1.000000  0.931230
Performance  0.933057  0.931230  1.000000
```

```
[ ]: sns.heatmap(r,linewidth=1,annot=True,annot_kws={'size':8})
```

```
[ ]: <Axes: >
```



```
[ ]: pip install pingouin --quiet pygam # NOTE that this is an external package which requires a specialized installation

[ ]: from pingouin import partial_corr
partial_corr(data=mydata, x='Income', y='Performance', covar='Experience')

[ ]:      n          r        CI95%       p-val
pearson  14  0.403083  [-0.19, 0.78]  0.172036
```

In the above output, the years of `Experience` have been covaried (partialled out). The correlation between `Income` and `Performance` score has been reduced considerably to less than half of the original one and the `p-value`  $> 0.05$ , so it is not significant anymore. It removed that part of the correlation between `Income` and `Performance` score which was due to years of `Experience`, and there is not much correlation left.

```
[ ]: partial_corr(data=mydata, x='Income', y='Experience', covar='Performance')

[ ]:      n          r        CI95%       p-val
pearson  14  0.611744  [0.09, 0.87]  0.026291
```

The correlation above has been reduced to a moderate 0.61, but it is still significant.

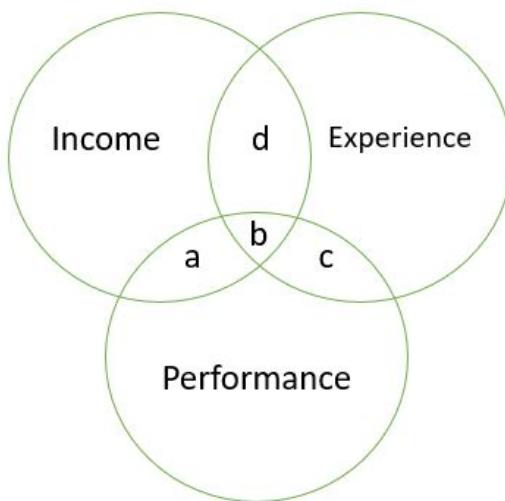
```
[ ]: partial_corr(data=mydata, x='Performance', y='Experience', covar='Income')

[ ]:      n          r        CI95%       p-val
pearson  14  0.428899  [-0.16, 0.79]  0.143632
```

The correlation above is not significant anymore.

```
[ ]: import requests; from io import BytesIO; from PIL import Image
url = 'https://raw.githubusercontent.com/leonkag/Statistics0/main/PartialCorrDiagram.JPG'
page = requests.get(url); Image.open(BytesIO(page.content))

[ ]:
```



We can illustrate this idea using the diagram above showing overlapping circles for each of our three variables. It is not mathematically accurate; the circles are exaggerated to understand the idea. The shared variance between **Income** and **Performance** is represented by the areas **a** and **b**. The **Experience** is related (shares variance with) to both **Income** and **Performance** with areas **c**, **b**, and **d**, respectively. On the other hand, area **a** shows the unique variance of **Income** and **Performance**. Area **b** represents the part of the relationship between **Income** and **Performance** which is due to **Experience**. If we remove the influence of **Experience** (area **b**), the correlation between **Income** and **Performance** is considerably reduced (area **a** only).

A medical example of partial correlation might be a relationship between symptoms and quality of life, with depression partialled out (covaried). If the correlation between symptoms and quality of life is considerably reduced after partialling out depression, it can be concluded that a part of the relationship between symptoms and quality of life is due to depression.

## 8.2 Least Squares Regression Line

Given that there is a strong correlation between **WT** and **WAIST**, the observations follow a linear model. What is the equation of the best line? We denote it as:

$$\hat{y} = E(Y|X = x) = b_0 + b_1x \quad (8.4)$$

To find the best line, a **residual** for each data point  $i = 1\dots n$  is defined as a vertical distance between the actual y-value  $y_i$  of the data point and its prediction by the line  $\hat{y}_i$ :

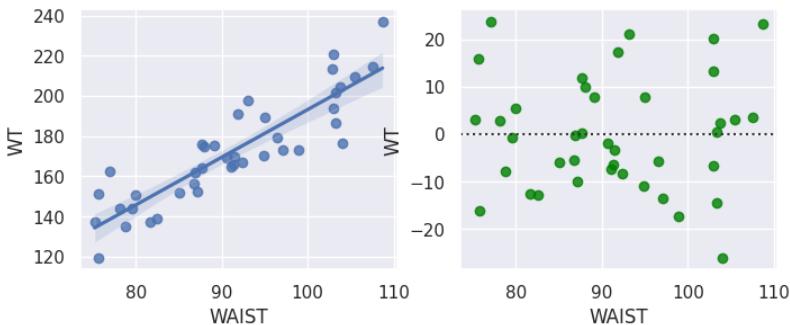
$$e_i = y_i - \hat{y}_i = y_i - (b_0 + b_1 \cdot x_i) \quad (8.5)$$

The Figure below shows data with the best regression line on the left side and residuals on the right side using `sns.regplot()` and `sns.residplot()`, respectively.

```
[ ]: import pandas as pd; import numpy as np; import seaborn as sns
import matplotlib.pyplot as plt

url="https://raw.githubusercontent.com/leonkag/Statistics0/main/MHEALTH.csv"
mydata = pd.read_csv(url) # save as mydata file
fig, axs = plt.subplots(ncols=2); fig.set_figheight(3); fig.set_figwidth(8)
sns.regplot(data=mydata,x="WAIST", y="WT",ax=axs[0]);
sns.residplot(data=mydata,x="WAIST", y="WT",color='green',ax=axs[1])
```

[ ]: <Axes: xlabel='WAIST', ylabel='WT'>



The residuals seem to be randomly positive and negative around  $y = 0$ , so the line fits the data well. The positive and negative residuals would cancel each other in a sum. Instead, a sum of squared residuals:  $e_1^2 + e_2^2 + \dots + e_n^2$  is minimized (**least squares regression line**). The idea is analogous to defining the standard deviation using the sum of squared deviations from the mean  $s = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n-1}}$ . The minimization is done using Multivariable Calculus. The slope is:

$$b_1 = \frac{s_y \cdot r}{s_x} \quad (8.6)$$

where  $r$  is the correlation coefficient, and  $s_x$  and  $s_y$  are standard deviations. The proportionality and positivity of standard deviations implies that the sign

of the slope of the line is the same as correlation coefficient  $r$  sign. Also, it is shown that means point  $(\bar{x}, \bar{y})$  is always on the regression line. Therefore, the point-slope form of the equation of regression line is:

$$y - \bar{y} = b_1(x - \bar{x})$$

$$y = \bar{y} - b_1\bar{x} + b_1x$$

Therefore, the y-intercept is

$$b_0 = \bar{y} - b_1\bar{x} \quad (8.7)$$

Let's illustrate these formulas:

```
[ ]: x = mydata['WAIST']; y = mydata['WT'];
xbar = np.mean(x); sx = np.std(x,ddof=1);
ybar = np.mean(y); sy = np.std(y,ddof=1);
r, pval = pearsonr(x,y); b1 = sy*r/sx; b0 = ybar-b1*xbar;
print('y-intercept b0 = {:.4f} and slope b1 = {:.4f}'.format(b0,b1))
```

y-intercept b0 = -44.0759 and slope b1 = 2.3731

Based on the code above, the y-intercept is  $b_0 = -44.08$ , and the slope is  $m = b_1 = 2.37$ . Therefore,

$$\hat{y} = b_0 + b_1x = -44.08 + 2.37 \cdot x$$

or in terms of the original variables

$$\widehat{WT} = -44.08 + 2.37 \cdot WAIST$$

A slope is a rise over run; therefore here for each 1 cm increase in WAIST, you are expected to gain **on average** 2.37 lb of WT. Note that for the linear model, it matters that **explanatory (independent) variable** WAIST explains **response (dependent) variable** WT, while for correlation it did not matter.

Let's find residual for a given man with WAIST size of 95 cm and weight of 181 lb - (data point (95, 181)).

$$\hat{y} = -44.08 + 2.37 \cdot 95 \approx 181.37$$

Therefore, the residual is

$$e = y - \hat{y} = 181 - 181.37 = -0.37 < 0$$

Now, for a different man with WAIST size of 75 cm and weight of 170, the residual is positive:

$$\hat{y} = -44.08 + 2.37 \cdot 75 \approx 133.90$$

$$e = y - \hat{y} = 170 - 133.90 = 36.1 > 0$$

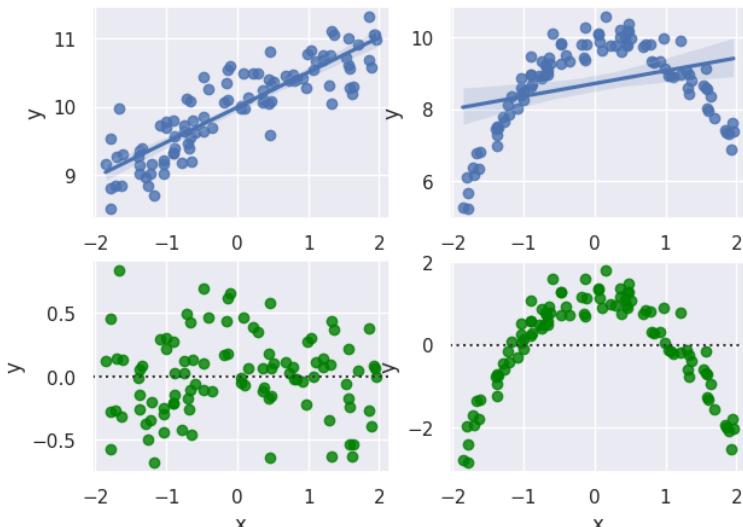
Generally, if a prediction is below the actual data point (*underestimate*), then  $e = y - \hat{y} > 0$  - positive residual. On the other hand, if a prediction is above the actual data point (*overestimate*), then  $e = y - \hat{y} < 0$  - negative residuals.

### 8.2.1 Assumptions for the Least Squares Model

1. **Linearity.** The scatterplot must have a linear trend, and the corresponding residual plot should be random (no regular pattern). The simulation below shows such a linear trend in the data with a corresponding random residual plot without any pattern (left side of the Figure below), as well as a non-linear pattern in the data and residuals (right side of the Figure below).

```
[ ]: n1 = 100; a = 10; b= 0.5; sig = 0.3;
x = np.random.uniform(low=-2, high=2, size=n1)
y = a + b*x + np.random.normal(loc=0.0, scale=sig, size=n1)
df = pd.DataFrame({'x':x, 'y':y})
fig, axs = plt.subplots(2, 2, figsize=(7,5))
sns.regplot(data=df,x='x',y='y',ax=axs[0,0]);
sns.residplot(data=df,x='x',y='y',color='green',ax=axs[1,0])
y = a + b*x - x**2 + np.random.normal(loc=0.0, scale=sig, size=n1)
df = pd.DataFrame({'x':x, 'y':y})
sns.regplot(data=df,x='x',y='y',ax=axs[0,1]);
sns.residplot(data=df,x='x',y='y',color='green',ax=axs[1,1])
```

[ ]: <Axes: xlabel='x', ylabel='y'>



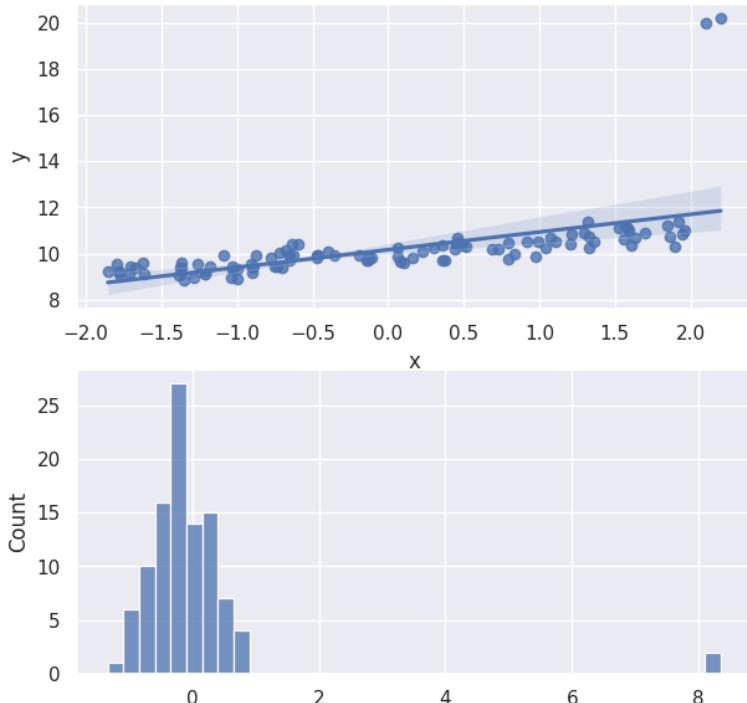
2. **Normal Residuals.** The residuals must be approximately normally distributed. Usually, it is very influential outliers that fail this condition and may completely invalidate the linear model. We discuss the classification of leverage and the influence of such outliers in more detail shortly. The Figure below shows the data with dramatic outliers which destroy the cor-

rect linear trend in the regression line and show up as extreme right skew outliers in the residual histogram below.

```
[ ]: x2 = np.append(x, [2.1,2.2])
y2 = a + b*x2 + np.random.normal(loc=0.0, scale=sig, size=n1+2);
y2[n1]=20; y2[n1+1]=20.2;
df = pd.DataFrame({'x':x2, 'y':y2})
fig, axs = plt.subplots(nrows=2); fig.set_figheight(7);
fig.set_figwidth(7)
sns.regplot(data=df,x='x',y='y',ax=axs[0]);

import statsmodels.api as sm
from statsmodels.formula.api import ols
# fit simple linear regression model
linear_model = ols('y ~ x', data=df).fit()
sns.histplot(linear_model.resid,ax=axs[1])
```

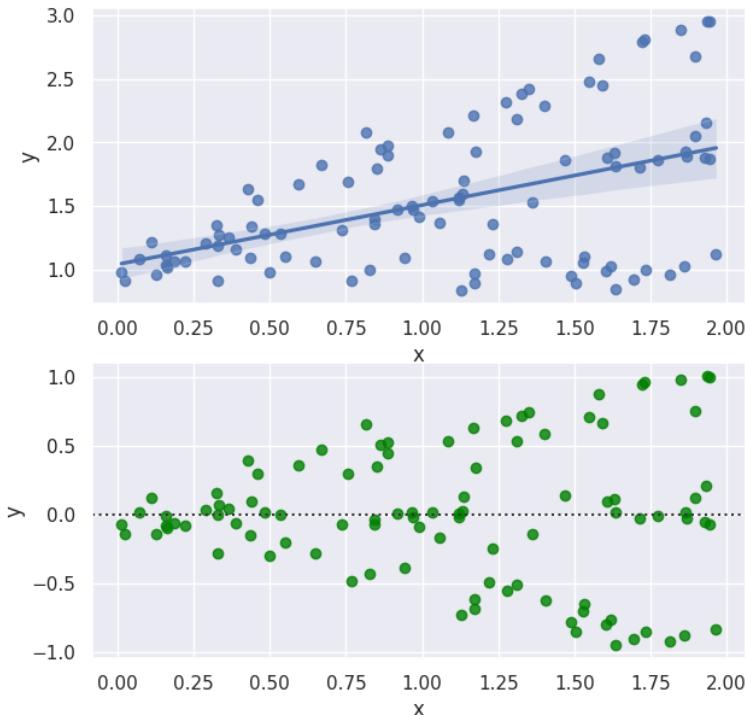
```
[ ]: <Axes: ylabel='Count'>
```



3. **Constant Variability.** The variability of observations about the least squares line should remain approximately the same. The Figure below shows the data where the variability increases (**funnel pattern**).

```
[ ]: n1 = 100; a = 1; b= 0.5; sig = 1
x2 = np.random.uniform(low=0, high=2, size=n1)
y2 = a+ b*x2 + np.random.choice([-1,0,1], size=n1, \
    replace=True)*b*x2 + np.random.normal(loc=0.0, scale=0.1, size=n1)
df = pd.DataFrame({'x':x2,'y':y2})
fig, axs = plt.subplots(nrows=2); fig.set_figheight(7);
fig.set_figwidth(7)
sns.regplot(data=df,x='x',y='y',ax=axs[0]);
sns.residplot(data=df,x='x',y='y',color='green',ax=axs[1])
```

[ ]: <Axes: xlabel='x', ylabel='y'>

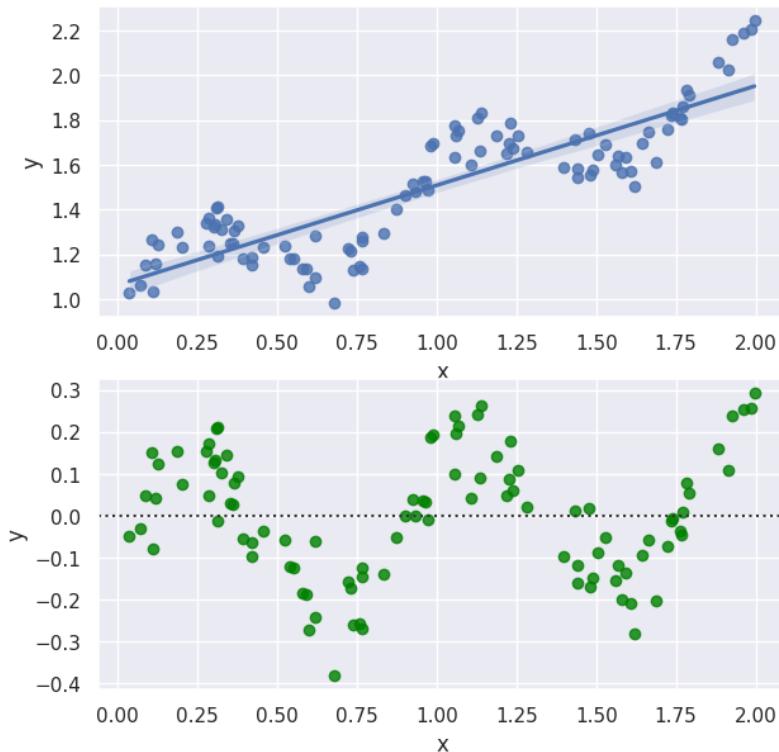


4. **Independent Observations.** The observations must be independent - one does not affect the other. The least squares line model cannot explain the time series data (sequential observations in time such as gasoline prices over several years or stock prices variations in time). There are specialized time series analysis techniques for such data. The Figure below shows the data with oscillating time dependence (seasonal changes on top of linear trend). In addition, the Durbin-Watson test can be used to check for autocorrelation. It reports a test statistic with a range from 0 to 4, where the value of 2 indicates that there is no autocorrelation. The (0,2) range

corresponds to positive autocorrelation (common in time series data), and the (2,4) range indicates negative autocorrelation (less common in time series data).

```
[ ]: n1 = 100; a = 1; b = 0.5; sig = 1
x2 = np.random.uniform(low=0, high=2, size=n1)
y2 = a + b*x2 + b/3*np.sin(7*x2) +
     np.random.normal(loc=0.0, scale=0.07, size=n1)
df = pd.DataFrame({'x':x2, 'y':y2})
fig, axs = plt.subplots(nrows=2);
fig.set_figheight(7); fig.set_figwidth(7)
sns.regplot(data=df,x='x',y='y',ax=axs[0]);
sns.residplot(data=df,x='x',y='y',color='green',ax=axs[1])
```

[ ]: <Axes: xlabel='x', ylabel='y'>



### 8.2.2 Statistical Analysis of Regression Coefficients

Assuming the data set satisfies all the assumptions of the linear regression above, we can analyze the goodness of fit of the linear model in several ways:

correlation, slope, ANOVA, etc. The correlation analysis was already done in the previous section.

The slope hypothesis test:

$H_0 : \beta_1 = 0$  - zero slope, no linear relationship

$H_1 : \beta_1 \neq 0$  - non-zero slope, there is a linear relationship

The standard error of the slope is:

$$SE(b_1) = \sqrt{\frac{1}{n-2} \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (x_i - \bar{x})^2}} \quad (8.8)$$

The regression summaries for WT vs. WAIST are given in the extensive tables below and the Figure shows scatterplot with the best fit line.

```
[ ]: # import packages and libraries
import numpy as np; import pandas as pd
import matplotlib.pyplot as plt; import seaborn as sns
import statsmodels.api as sm; from statsmodels.formula.api import ols

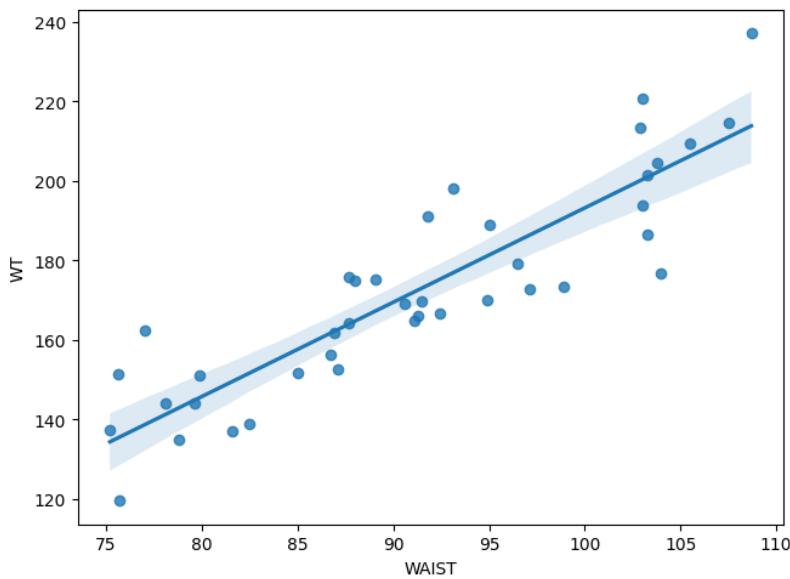
url="https://raw.githubusercontent.com/leonkag/Statistics0/main/MHEALTH.csv"
mydata = pd.read_csv(url)    # save as mydata file
# mydata.head(10)
fig, ax = plt.subplots(); fig.set_size_inches(7.5, 5.5)
sns.regplot(data=mydata, x="WAIST", y="WT")

# fit simple linear regression model
linear_model = ols('WT ~ WAIST', data=mydata).fit()
print(linear_model.summary())
```

OLS Regression Results						
Dep. Variable:	WT	R-squared:				0.790
Model:	OLS	Adj. R-squared:				0.785
Method:	Least Squares	F-statistic:				143.1
Date:	Fri, 19 Apr 2024	Prob (F-statistic):				1.87e-14
Time:	18:20:36	Log-Likelihood:				-155.84
No. Observations:	40	AIC:				315.7
Df Residuals:	38	BIC:				319.1
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
Intercept	-44.0759	18.211	-2.420	0.020	-80.941	-7.210
WAIST	2.3731	0.198	11.963	0.000	1.972	2.775
Omnibus:		0.865	Durbin-Watson:			2.231
Prob(Omnibus):		0.649	Jarque-Bera (JB):			0.930
Skew:		0.263	Prob(JB):			0.628
Kurtosis:		2.469	Cond. No.			866.

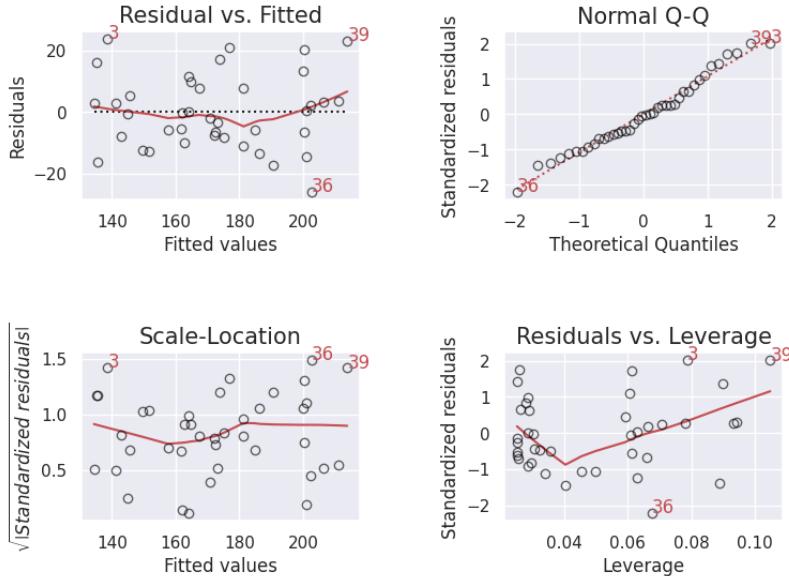
Notes:

```
[1] Standard Errors assume that the covariance matrix of the errors is ✉  
→correctly  
specified.
```



```
[ ]: !pip install --quiet lmdiag #
```

```
[ ]: plt.figure(figsize=(7.5,5.5))  
import lmdiag  
lmdiag.plot(linear_model);
```



The diagnostic plots above provide extremely important data summaries. The top-left one is Residuals vs. Fitted values. This plot checks if residuals have non-linear patterns. If you find equally spread residuals around a horizontal line without distinct patterns (as in this case), that is a good indication we don't have a non-linear relationship. The top-right plot shows a quantile-quantile (QQ) plot of standardized residuals vs. the standard normal distribution quantiles. If the residuals approximately follow a straight line (as in this case) the normality assumption of residuals holds. The bottom-left figure is the Scale-Location plot. It checks if residuals are spread equally along the ranges of predictors - assumption of equal variance (homoscedasticity). The top-left figure shows this too, but this one shows the possible funnel pattern more clearly. This example shows the line with equally (randomly) spread points, so the homoscedasticity holds. The bottom-right figure shows Residuals vs. Leverage. We discuss it in more detail in the next section.

The “Coef” part of the table gives the y-intercept, slope, and the corresponding statistical analysis. For the slope:

$$t = \frac{(point\ estimate) - (null\ value)}{SE} = \frac{(b_1 - 0)}{SE(b_1)} \approx \frac{2.37}{0.2} \approx 11.96 \quad (8.9)$$

The t-distribution for the slope with  $df = n - 2 = 40 - 2 = 38$  degrees of freedom produced extremely low **p-value** =  $1.87 \cdot 10^{-14} < 0.05$ , so null

hypothesis of zero slope  $\beta_1 = 0$  can be rejected (significant linear model). Note also, that the t-test and p-value are exactly the same as for the correlation hypothesis. It is not an accident because for simple one-variable regression  $b_1 = \frac{s_{y|r}}{s_x}$ , i.e., slope is proportional to the correlation coefficient. In addition, slope standard error can be used to define the confidence interval:

$$(point\ estimate) \pm t_{df} \cdot SE = b_1 \pm t_{n-2} \cdot SE(b_1) =$$

$$2.37 \pm 2.024 \cdot 0.2 = (1.97, 2.77)$$

This confidence interval does not contain 0 which proves, yet again, that the slope is not 0 and the linear model is valid.

The code output also shows an analogous statistical analysis for the y-intercept  $H_0 : \beta_0 = 0$ , which is rarely used (unless we look for direct proportionality). It is the output of the linear model when input is zero, but  $x = 0$  is often impractical. For example, in our WT vs. WAIST example, WAIST = 0 is not practical and resulting  $b_0 = -44.08$  correspond to negative weight which does not make sense either. We will see some other examples in future sections where  $b_0$  has a practical meaning.

### 8.2.3 Leverage and Influence

Generally, outliers in regression are observations that fall far from the regression line. However, the leverage and influence of such outliers must be investigated.

**1. Leverage.** Points that fall horizontally far away from the center (mean) of the data cloud pull harder on the line - high-leverage points, so leverage measures the extremity of the predictors. A high-leverage point is an observation with x-values extreme enough to *potentially* significantly change the slopes or trends in the fitted model. An outlier can have a high or low leverage.

The formula for leverage is

$$h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{s_x^2}$$

where  $n$  is the sample size of  $(x, y)$  pairs,  $x_i$  is the x-coordinate of a particular point,  $\bar{x}$  is the mean of all  $x$ 's, and  $s_x^2$  is the variance of  $x$ 's.

In general  $1/n \leq h_i \leq 1$ . If there are  $p$  independent variables in the model, the mean value for leverage is  $(p+1)/n$  (for simple regression  $p=1$ , so it is  $2/n$ ). A rule of thumb (Steven's) is that values that are 3 times this mean value are considered large.

**2. Influence.** An observation with high leverage that significantly affects the regression line is influential. The standard measure of influence is Cook's

distance, which estimates the magnitude of the effect of deleting the  $i$ -th observation from the model. Cook's distance is defined in a more general context of multiple regression with  $p$  explanatory variables (predictors), in our case  $p = 1$ .

Let:

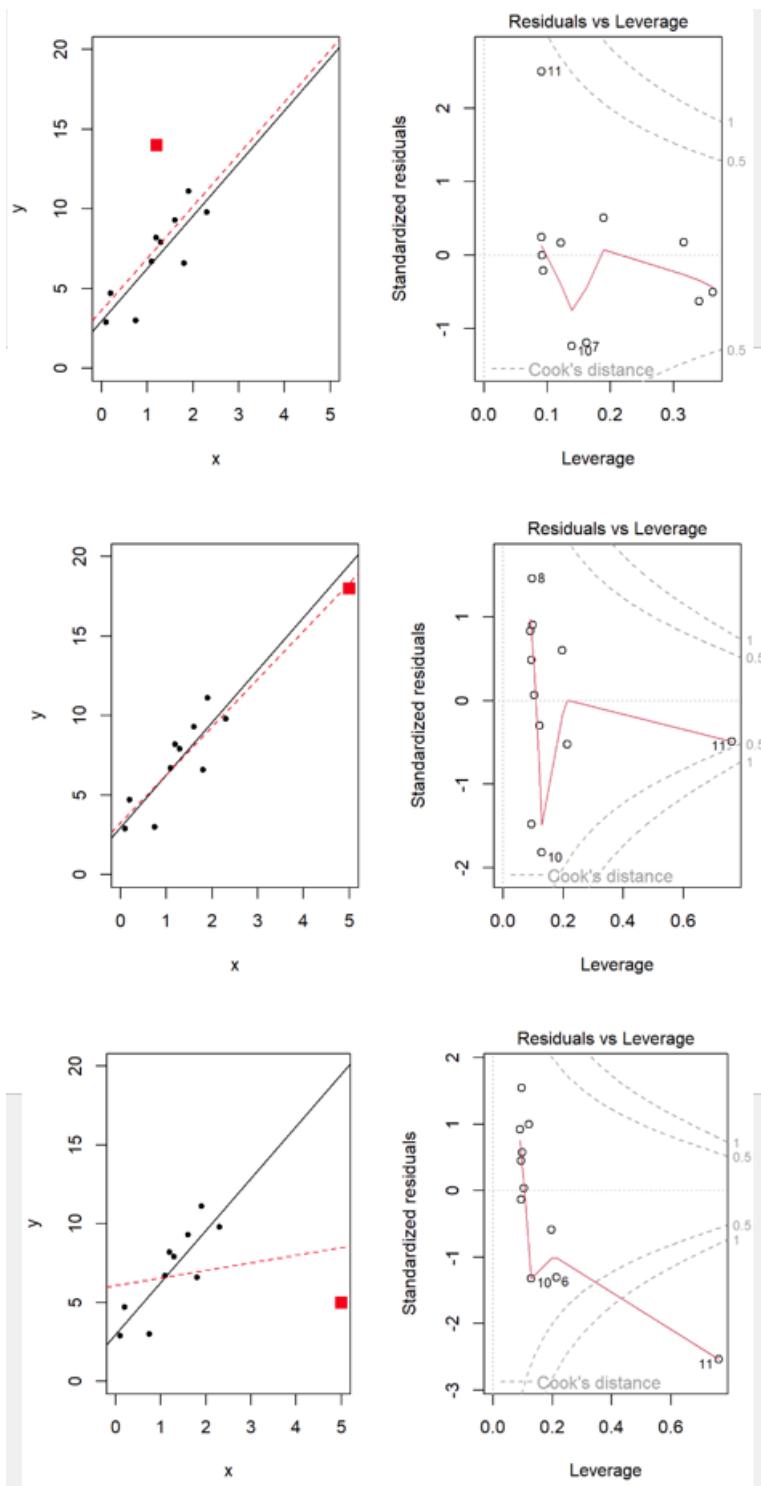
$\hat{y}_j$  = predicted mean response of observation  $j$  for the model fitted with all  $n$  observations.

$\hat{y}_j^{(-i)}$  = same response fitted without the  $i$ -th observation.

$s_e$  = residual standard error. Then Cook's distance is:

$$D_i = \frac{\sum_{j=1}^n (\hat{y}_j - \hat{y}_j^{(-i)})^2}{(p+1)s_e^2} \quad (8.10)$$

The Residuals vs. Leverage diagnostic plot with Cook's distances shown as dashed level curves with levels 0.5, 1 etc. illustrates the interplay between leverage and influence. The Figure below shows the effect of adding one point (big square) to the original data on the scatterplot and Residuals vs. Leverage plots. Highly influential points cross Cook's level curves as shown in the bottom-right plot below.



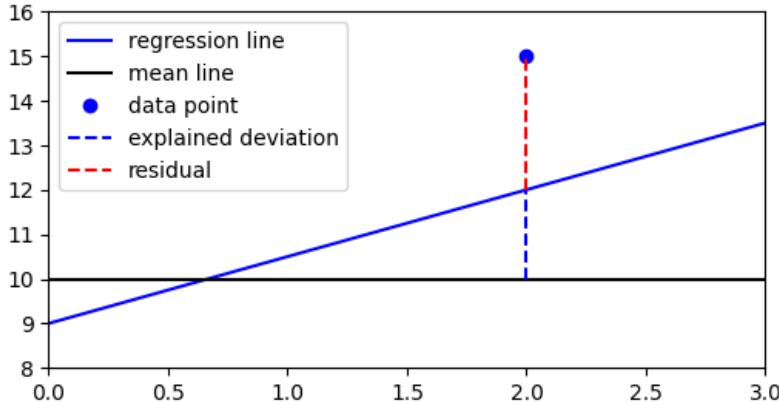
In the top plots of the above Figure, the additional point has low leverage since its predictor value is close to the  $\bar{x}$ . Its inclusion only slightly modifies the y-intercept of the original line, not slope, so it has very low influence, as can be seen in top-right graph with none of the points crossing Cook's distance level 0.5. In the middle plots, the additional point is far apart from the original observations' mean, so it has high leverage. However, it conforms very well to the regression line model fitted without it. Its inclusion changes the regression line very slightly, so the influence is still low, so it does not cross Cook's distance level 0.5. Lastly, in the bottom plots, the additional point is very far away from the original observations and has high leverage. This time, however, its inclusion significantly alters the regression line by turning it clockwise, so it has very high influence and it crosses Cook's distance level curves 0.5 and 1.

#### 8.2.4 Regression ANOVA

Next, we introduce an Analysis of Variance (ANOVA) for regression. It is studied in much more detail in the context of comparing several means, but it can illuminate the goodness of fit of the regression line as well. Assume there is a set of observations (not shown in the Figure below) that produces the regression line  $\hat{y} = 9 + 1.5x$ . Concentrate on just one data point  $(2, 15)$  to keep the figure clear. Also, the overall mean of the response variable is  $\bar{y} = 10$ .

```
[ ]: import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(6,3))
ax.axline((0, 9), slope=1.5, color='blue', label='regression line')
ax.set_xlim(0, 3); ax.set_ylim(8, 16)
ax.axline((0, 10), slope=0, color='black', label='mean line')
ax.plot(2, 15, 'bo', label='data point')
plt.plot([2,2], [10,12], 'blue', linestyle="--", label='explained deviation')
plt.plot([2,2], [12,15], 'red', linestyle="--", label='residual')
ax.legend()
```

```
[ ]: <matplotlib.legend.Legend at 0x7e0530f2cdf0>
```



The total vertical deviation of data point  $(2, 15)$  from overall response mean is  $y_i - \bar{y} = 15 - 10 = 5$ . It can be broken into the sum of two deviations:

$$(total\ deviation) = (explained\ deviation) + (unexplained\ deviation\ (residual))$$

$$\begin{aligned} y_i - \bar{y} &= (\hat{y}_i - \bar{y}) + (y_i - \hat{y}_i) \\ 5 &= 2 + 3 \end{aligned}$$

As always, deviations are both positive and negative and would cancel each other out in a sum, so squared deviations are considered. The total sum of squares can be broken into:

$$SST(\text{total}) = SSR(\text{explained or regression}) + SSE(\text{residual or error})$$

$$\sum_{i=1}^n (y_i - \bar{y})^2 = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2 + \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

The ANOVA results are best represented in a table below that shows degrees of freedom, sums of squares, averaged squared deviations (sum of squares divided by the corresponding degrees of freedom), F ratio of averaged deviations, and the corresponding p-value.

```
[ ]: sm.stats.anova_lm(linear_model)
```

	df	sum_sq	mean_sq	F	PR(>F)
WAIST	1.0	21360.114243	21360.114243	143.115189	1.871685e-14
Residual	38.0	5671.545757	149.251204	NaN	NaN

For **simple regression** with only one explanatory variable, the F-test ( $F = t^2$  i.e.,  $143.12 \approx 11.96^2$ ) and the p-value is the same as for the slope hypothesis

t-test. Thus, this confirms, yet again, the validity of the linear model. This is not true for multiple regression where F-test provides overall significance and each variable has its own t-test. Moreover, the averaged residual sum of squares  $MME = \frac{1}{df} \sum_{i=1}^n (y_i - \hat{y}_i)^2$  in the ANOVA can be used to calculate the

$$\text{Residual standard error} = \sqrt{MME} = \sqrt{\frac{1}{df} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (8.11)$$

For WT vs. WAIST, the residual standard error is  $\sqrt{149.30} = 12.22$ .

When correlation was introduced, the **coefficient of determination**  $r^2$  was defined, which measures the proportion of the variation in response variable that is explained by the linear model. More precisely:

$$r^2 = \frac{SSR}{SST} = \frac{21360.1}{21360.1 + 5671} = 0.79019 \quad (8.12)$$

which is the ratio of the explained variation in ANOVA to the total variation. It is exactly the  $r^2$  that we obtained before.

### 8.2.5 Linear Model Prediction

If a linear model is appropriate for the data, it can be used to predict the response for a given explanatory variable. However, it is only a point estimate, while a confidence interval is needed. There are actually two different kinds of confidence intervals for the prediction.

#### Confidence interval for the individual response value.

First, suppose we consider a particular man (John) with a WAIST size of 81 cm. For our WT vs. WAIST model, the WT predicted by the least squares line is simply found by plugging 81 into the regression line equation:

$$y^* = b_0 + b_1 \cdot x^* = -44.08 + 2.37 \cdot 81 = 148.14$$

The regression model is based on a particular sample of  $n = 40$  men and would be a bit different for a different sample. The sample variation can be quantified via a confidence interval.

$$(\text{point estimate}) \pm t_{df} \cdot SE = y^* \pm t_{df} \cdot SE \quad (8.13)$$

The standard error formula is given by

$$SE = \sqrt{s_e^2 + \frac{s_e^2}{n} + SE(b_1)^2(x^* - \bar{x})^2} \quad (8.14)$$

where  $s_e$  is residual standard error and  $SE(b_1)$  - slope standard error. The  $s_e^2$  term accounts for variation among all of the residuals, while  $\frac{s_e^2}{n}$  accounts

for variation in the particular residual for  $y^*$ . Any linear regression model is anchored at the means point  $(\bar{x}, \bar{y})$ . As the given  $x^*$  gets further away from the mean  $\bar{x}$ , the term  $SE(b_1)^2(x^* - \bar{x})^2$  becomes larger, and the standard error increases producing a wider confidence interval. For our example:

```
[ ]: xs = 81; b = linear_model.params; ys = b[0] + b[1]*xs;
se = np.sqrt(linear_model.scale); SE1 = linear_model.bse[['WAIST']];
n = mydata.shape[0]; xbar = np.mean(mydata['WAIST']);
print('xs = {:.3f}, ys = {:.3f}, se = {:.3f}, SE1 = {:.3f}, \
n = {:.d}, xbar = {:.3f}'.format(xs,ys,se,SE1,n,xbar))
SEobs = np.sqrt(se**2 + se**2/n + SE1**2*(xs-xbar)**2);
from scipy.stats import t; tf = t.ppf(0.975,df=n-2);
print('SEobs = {:.3f}, tf = {:.3f}'.format(SEobs, tf))
CIobsL = ys-tf*SEobs; CIobsR = ys+tf*SEobs;
print('Confidence interval for the observation:')
print('(CIobsL, CIobsR) = {:.3f},{:.3f})'.format(CIobsL,CIobsR))
```

```
xs = 81.000, ys = 148.143, se = 12.217, SE1 = 0.198, n = 40, xbar = 91.285
SEobs = 12.536, tf = 2.024
Confidence interval for the observation:
(CIobsL, CIobsR) = (122.766,173.520)
```

$$SE = \sqrt{12.22^2 + \frac{12.22^2}{40} + 0.198^2(81 - 91.28)^2} = 12.54$$

$$CI = 148.14 \pm 2.024 \cdot 12.54 = (122.77, 173.52)$$

### Confidence interval for the mean response value.

The mean values are more predictable than individual measurements (based on the Central Limit Theorem), so in predicting the *mean WT* of men with WAIST 81cm, we can expect a smaller standard error SE:

$$SE = \sqrt{\frac{s_e^2}{n} + SE(b_1)^2(x^* - \bar{x})^2} \quad (8.15)$$

It is almost the same formula as for the individual prediction, except it does not have the term  $s_e^2$ , which accounts for variation among all of the residuals.

```
[ ]: SEmean = np.sqrt(se**2/n + SE1**2*(xs-xbar)**2);
print('SEmean = {:.3f}, tf = {:.3f}'.format(SEmean, tf))
CImeanL = ys-tf*SEmean; CImeanR = ys+tf*SEmean;
print('Confidence interval for the mean (regression line):')
print('(CImeanL, CImeanR) = {:.3f},{:.3f})'.format(CImeanL,CImeanR))
```

```
SEmean = 2.810, tf = 2.024
Confidence interval for the mean (regression line):
(CImeanL, CImeanR) = (142.455,153.831)
```

Both, the individual prediction confidence interval and the mean prediction confidence interval are really computed by a single command:

```
[ ]: predictions = linear_model.get_prediction(pd.DataFrame({'WAIST': [xs]}))
predictions.summary_frame(alpha=0.05)

[ ]:      mean    mean_se  mean_ci_lower  mean_ci_upper  obs_ci_lower \
0  148.142956  2.809571     142.455278     153.830635     122.765668

      obs_ci_upper
0      173.520245
```

In fact, we can compute prediction intervals for several values at once as shown in the code below:

```
[ ]: xs = np.linspace(80, 95, num=16)
predictions = linear_model.get_prediction(pd.DataFrame({'WAIST': xs}))
df = pd.DataFrame(predictions.summary_frame(alpha=0.05))
df.insert(loc=0, column='xs', value=xs); df
```

	xs	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	80.0	145.769885	2.956763	139.784230	151.755539	120.324156	171.215614
1	81.0	148.142956	2.809571	142.455278	153.830635	122.765668	173.520245
2	82.0	150.516028	2.669012	145.112896	155.919160	125.200995	175.831061
3	83.0	152.889100	2.536190	147.754852	158.023348	127.630091	178.148110
4	84.0	155.262172	2.412383	150.378558	160.145786	130.052913	180.644272
5	85.0	157.635244	2.299048	152.981064	162.289423	132.469425	183.226314
6	86.0	160.008316	2.197806	155.559090	164.457541	134.879593	185.801355
7	87.0	162.381387	2.110398	158.109110	166.653665	137.283391	188.015437
8	88.0	164.754459	2.038604	160.627521	168.881397	139.680794	190.362562
9	89.0	167.127531	1.984120	163.110890	171.144171	142.071784	192.711663
10	90.0	169.500603	1.948398	165.556277	173.444928	144.456347	195.050734
11	91.0	171.873675	1.932479	167.961575	175.785774	146.834473	197.321805
12	92.0	174.246746	1.936852	170.325794	178.167699	149.206161	199.608876
13	93.0	176.619818	1.961381	172.649210	180.590426	151.571409	201.979447
14	94.0	178.992890	2.005326	174.933320	183.052460	153.930225	204.473508
15	95.0	181.365962	2.067449	177.180629	185.551294	156.282619	206.449304

The Figure below shows 95% confidence intervals for the mean prediction around the regression line and the individual prediction (much wider). As the

point moves further away from the mean  $\bar{x}$ , the confidence intervals get wider and wider.

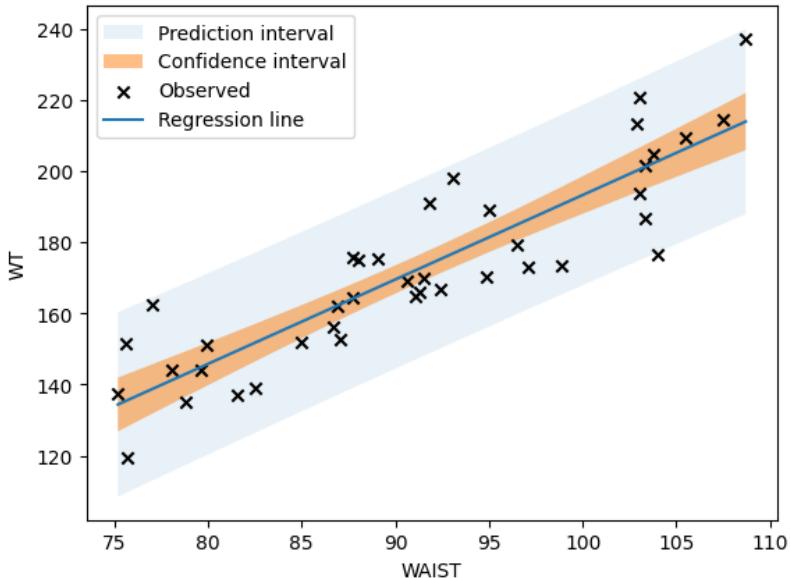
```
[ ]: alpha = 0.05
predictions = linear_model.get_prediction(mydata).summary_frame(alpha)
print("Predictions based on the model: \n\n:")
print(predictions.head())

plt.fill_between(mydata['WAIST'], predictions['obs_ci_lower'],
                 predictions['obs_ci_upper'], alpha=.1, label='Prediction interval')
plt.fill_between(mydata['WAIST'], predictions['mean_ci_lower'],
                 predictions['mean_ci_upper'], alpha=.5, label='Confidence interval')
plt.scatter(mydata['WAIST'], mydata['WT'], label='Observed',
            marker='x', color='black')
plt.plot(mydata['WAIST'], predictions['mean'], label='Regression line')
plt.xlabel('WAIST'); plt.ylabel('WT'); plt.legend()
plt.show()
```

Predictions based on the model:

```
:
      mean   mean_se  mean_ci_lower  mean_ci_upper  obs_ci_lower \
0  134.379140  3.729878     126.828396     141.929884    108.520470
1  135.328369  3.662232     127.914568     142.742170    109.509354
2  135.565676  3.645394     128.185962     142.945390    109.756429
3  138.650669  3.429421     131.708169     145.593169    112.963019
4  141.261048  3.251448     134.678836     147.843260    115.668421

      obs_ci_upper
0      160.237810
1      161.147383
2      161.374923
3      164.338319
4      166.853676
```



### 8.3 Linear Model Examples

#### Example

Investigate correlation and regression for the data below on Longevity and Smoking. Predict several typical values.

```
[ ]: # import packages and libraries
import numpy as np; import pandas as pd
import matplotlib.pyplot as plt; import seaborn as sns;
import statsmodels.api as sm; from scipy.stats import pearsonr
from statsmodels.formula.api import ols

url="https://raw.githubusercontent.com/leonkag/Statistics0/main/LongevSmoke.
↪csv"
mydata = pd.read_csv(url)    # save as mydata file
print(mydata.head(10))
sns.regplot(data=mydata,x='Packs',y='Longevity')

r = pearsonr(mydata['Packs'],mydata['Longevity']);
print("\nPearson corelation: ")
print(r, '\n')

# fit simple linear regression model
```

```
linear_model = ols('Longevity ~ Packs', data=mydata).fit()
print(linear_model.summary())
print('\nHigher accuracy printout of p-values: \n', linear_model.pvalues, '\n')
```

	Packs	Longevity
0	0	80
1	0	70
2	0	75
3	0	77
4	1	72
5	1	70
6	1	69
7	2	68
8	2	65
9	2	70

Pearson corelation:

```
PearsonRResult(statistic=-0.8882477180477968, pvalue=4.323033187879979e-06)
```

#### OLS Regression Results

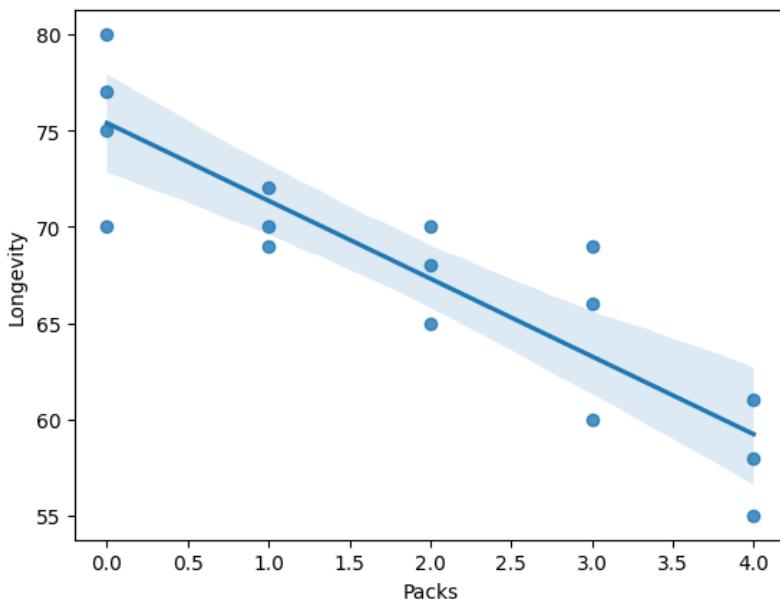
```
=====
Dep. Variable: Longevity R-squared: 0.789
Model: OLS Adj. R-squared: 0.774
Method: Least Squares F-statistic: 52.35
Date: Fri, 19 Apr 2024 Prob (F-statistic): 4.32e-06
Time: 19:49:34 Log-Likelihood: -40.467
No. Observations: 16 AIC: 84.93
Df Residuals: 14 BIC: 86.48
Df Model: 1
Covariance Type: nonrobust
=====
      coef    std err        t   P>|t|    [0.025    0.975]
-----
Intercept  75.3889     1.325    56.915  0.000    72.548    78.230
Packs      -4.0407     0.558    -7.235  0.000    -5.239    -2.843
=====
Omnibus:            0.268 Durbin-Watson: 1.911
Prob(Omnibus):      0.875 Jarque-Bera (JB): 0.439
Skew:                 0.090 Prob(JB): 0.803
Kurtosis:             2.209 Cond. No. 4.33
=====
```

Notes:

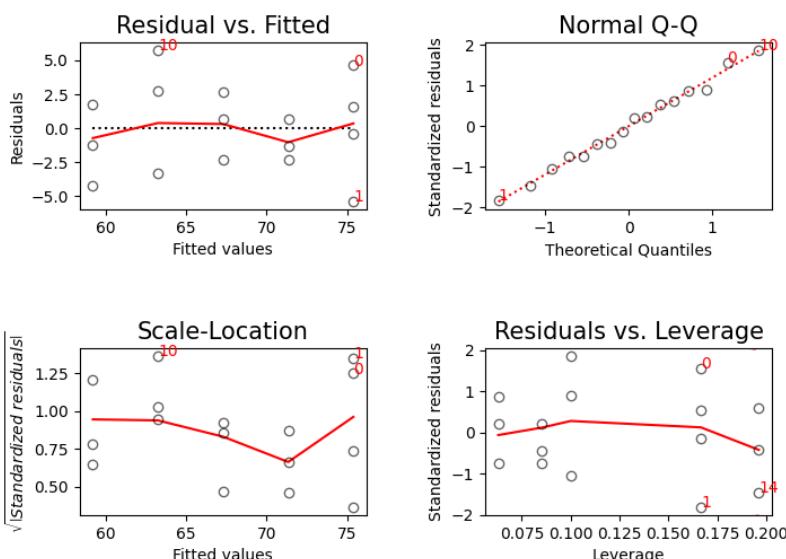
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
p-values:
Intercept 5.735547e-18
Packs      4.323033e-06
dtype: float64
```

```
/usr/local/lib/python3.10/dist-packages/scipy/stats/_stats_py.py:1806:
UserWarning: kurtosistest only valid for n>=20 ... continuing anyway, n=16
  warnings.warn("kurtosistest only valid for n>=20 ... continuing ")
```



```
[ ]: plt.figure(figsize=(7,5))
import lmdiag
lmdiag.plot(linear_model);
```



Based on the diagnostic plots and the data with regression line Figure above, we conclude the following. The assumptions of regression: a linear trend (no pattern of residuals), normality of residuals (QQ plot), constant variability (no funnel pattern), independent observations (no time series, Durbin-Watson is close to 2) are all satisfied. Residuals vs. Leverage does not have any points over Cook's distance level curves. Thus, the linear model is applicable.

The correlation coefficient  $r = -0.888$  indicates a strong negative correlation. Correlation hypothesis test:

$H_0: \rho = 0$  no linear correlation

$H_1: \rho \neq 0$  there is linear correlation.

The t-test gives  $t = -7.24$  and corresponding p-value  $= 4.3230332 \cdot 10^{-6} < 0.05$ , therefore  $H_0$  is rejected, and there is a significant linear correlation.

$r^2 = 0.789$ ; therefore 78.9% of variation in longevity is explained by this linear model (coefficient of determination).

Slope  $b_1 = -4.04$ ; therefore for each extra pack smoked per day, on average, the longevity decreases by 4.04 years. The hypothesis test for the slope:

$H_0: \beta_1 = 0$  zero slope, no linear relationship

$H_1: \beta_1 \neq 0$  non-zero slope, there is a linear relationship

The same t-test  $t = -7.24$  and p-value  $= 4.3230332 \cdot 10^{-6} < 0.05$  lead to rejection of  $H_0$ ; so there is a significant linear relationship.

The 95% confidence interval for the slope  $(-5.24, -2.84)$  doesn't contain 0, which is yet another indication of the non-zero slope.

The y-intercept  $b_0 = 75.39$  has practical meaning, unlike most other problems. It is the expected value of longevity for a non-smoker (packs smoked per day equal to 0).

Thus, it has been shown in several ways that the linear model  $\hat{y} = 75.39 - 4.04 \cdot x$  or  $\widehat{\text{Longevity}} = 75.39 - 4.04 \cdot \text{Packs}$  is appropriate for these data, so it can be used for prediction. The code below gives the predictions fit as well as individual prediction confidence intervals and tighter mean confidence intervals.

```
[ ]: xs = np.linspace(0, 7, num=8)
predictions = linear_model.get_prediction(pd.DataFrame({'Packs': xs}))
df = pd.DataFrame(predictions.summary_frame(alpha=0.05))
df.insert(loc=0, column='xs', value=xs); df
```

	xs	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	0.0	75.388889	1.324591	72.547924	78.229854	67.872401	76.000000
1	1.0	71.348148	0.946977	69.317084	73.379212	64.098892	70.000000
2	2.0	67.307407	0.814142	65.561247	69.053568	60.132759	69.000000
3	3.0	63.266667	1.026024	61.066064	65.467269	55.968095	63.000000
4	4.0	59.225926	1.437519	56.142755	62.309097	51.614589	59.000000
5	5.0	55.185185	1.924586	51.057358	59.313012	47.094113	55.000000
6	6.0	51.144444	2.442425	45.905963	56.382926	42.434210	51.000000

```

    7   7.0  47.103704  2.975010      40.722941      53.484466      37.662272

  obs_ci_upper
0     82.905376
1     78.597405
2     74.482056
3     70.565239
4     66.837262
5     63.276258
6     59.854679
7     56.545135

```

We can also compute a residual for a specific value slightly outside of the data set (reasonable extrapolation) as well as the typical residual error (Residual standard error  $s_e$ ).

```
[ ]: xs = 5; yexact = 60; b = linear_model.params; ys = b[0] + b[1]*xs;
print('xs = {:.3f}, ys = {:.3f}, yexact = {:.3f}, residual = {:.3f}'.format(xs,ys,yexact,yexact-ys))
se = np.sqrt(linear_model.scale);
print('Residual standard error = se = {:.3f}'.format(se))

xs = 5.000, ys = 55.185, yexact = 60.000, residual = 4.815
Residual standard error = se = 3.245
```

### Example

Consider correlation and regression for the WT vs. CHOL level in the MHEALTH file. If the linear model is good, predict several typical values.

```
[ ]: # import packages and libraries
import numpy as np; import pandas as pd
import matplotlib.pyplot as plt; import seaborn as sns;
import statsmodels.api as sm; from scipy.stats import pearsonr
from statsmodels.formula.api import ols

url="https://raw.githubusercontent.com/leonkag/Statistics0/main/MHEALTH.csv"
mydata = pd.read_csv(url)    # save as mydata file
sns.regplot(data=mydata,x='CHOL',y='WT')

r = pearsonr(mydata['CHOL'],mydata['WT']);
print("\nPearson corelation:")
print(r,'\n')

# fit simple linear regression model
linear_model = ols('WT ~ CHOL', data=mydata).fit()
print(linear_model.summary())
print('\nHigher accuracy printout of p-values: \n',linear_model.pvalues,'\\n')
```

```
Pearson corelation:
PearsonRResult(statistic=-0.025909074093860555, pvalue=0.8739101443884549)
```

```
OLS Regression Results
=====

```

```

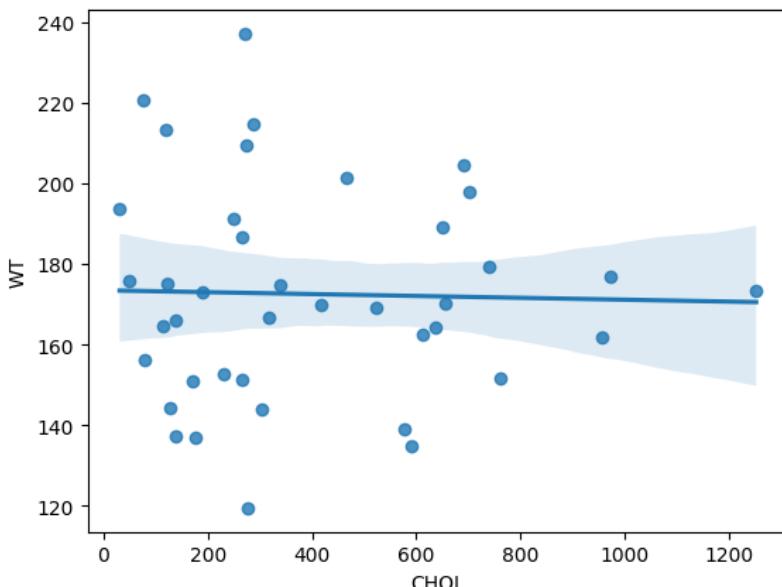
Dep. Variable: WT R-squared: 0.001
Model: OLS Adj. R-squared: -0.026
Method: Least Squares F-statistic: 0.02553
Date: Fri, 19 Apr 2024 Prob (F-statistic): 0.874
Time: 19:50:06 Log-Likelihood: -187.06
No. Observations: 40 AIC: 378.1
Df Residuals: 38 BIC: 381.5
Df Model: 1
Covariance Type: nonrobust
=====
      coef    std err        t   P>|t|    [0.025    0.975]
-----
Intercept  173.4719     7.146    24.274    0.000  159.005  187.939
CHOL       -0.0023     0.015    -0.160    0.874   -0.032    0.027
=====
Omnibus:          0.927 Durbin-Watson: 0.499
Prob(Omnibus):  0.629 Jarque-Bera (JB): 0.902
Skew:            0.332 Prob(JB): 0.637
Kurtosis:        2.684 Cond. No. 830.
=====
```

## Notes:

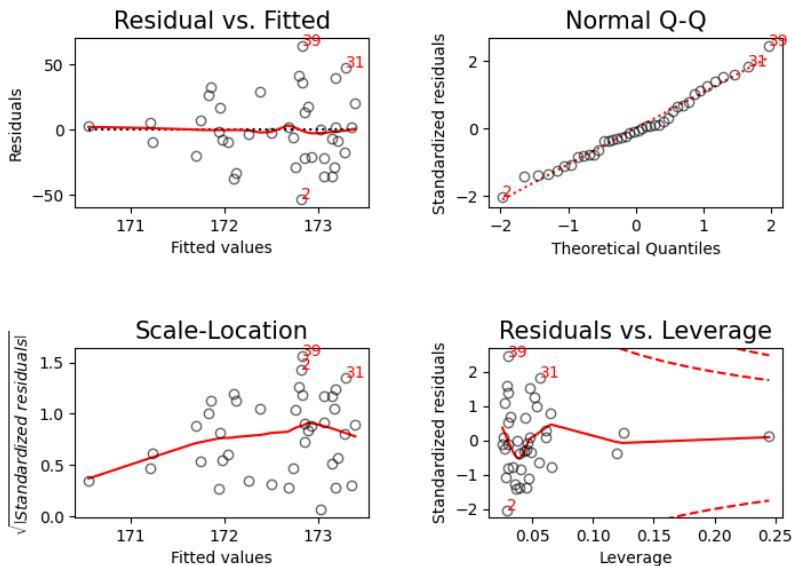
[1] Standard Errors assume that the covariance matrix of the errors is incorrectly  
specified.

```
p-values::
```

Intercept	9.702267e-25
CHOL	8.739101e-01
dtype:	float64



```
[ ]: plt.figure(figsize=(7,5))
import lmdiag
lmdiag.plot(linear_model);
```



Most importantly, there is *NO* linear trend as shown in the data with regression line Figure above. Also, based on the diagnostic plots above ,the normality of residuals (QQ plot) is approximately satisfied. The constant variability seems to fail; there is a funnel pattern. Independent observations (no time series, although Durbin-Watson is a bit below 2 indicating some autocorrelation) seem to be satisfied. The residuals vs. Leverage plot also shows no extreme behavior. With no linear trend, there is no reason to continue; however, we pursue to see how each test fails in detail.

Correlation coefficient  $r = -0.026$  implies almost no correlation. Correlation hypothesis test:

H0:  $\rho = 0$  no linear correlation

H1:  $\rho \neq 0$  there is a linear correlation.

The t-test gives  $t = -0.16$  and corresponding  $pvalue = 0.8739101 > 0.05$ , so H0 cannot be rejected, and there is no significant linear correlation.

The coefficient of determination  $r^2 = 0.00067$ ; therefore only 0.067% of variation in WT is explained by this linear relationship with CHOL (coefficient of determination).

The hypothesis test for the slope:

H0:  $\beta_1 = 0$  zero slope, no linear relationship

H1:  $\beta_1 \neq 0$  non-zero slope, there is a linear relationship.

The same t-test  $t = -0.16$  and corresponding  $pvalue = 0.8739101 > 0.05$  tell us again not to reject H0; so there is no significant linear relationship. In addition, the ANOVA test gives  $F = 0.026$  with the same p-value, confirming our conclusion yet again. The 95% confidence interval for the slope is  $(-0.03, 0.03)$ , which contains 0 - yet another indication of a non-significant slope.

Because this linear model is not appropriate, it should *NOT be used for prediction*.

### Example

Investigate correlation and regression for the HELPrct substance abusers data on depression score `cesd` vs. mental health score `mcs`. Predict several typical values.

```
[ ]: # import packages and libraries
import numpy as np; import pandas as pd
import matplotlib.pyplot as plt; import seaborn as sns;
import statsmodels.api as sm; from scipy.stats import pearsonr
from statsmodels.formula.api import ols

url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrct.csv"
mydata = pd.read_csv(url)    # save as mydata file
# print(mydata.head(10))
sns.regplot(data=mydata,x='mcs',y='cesd')

r = pearsonr(mydata['mcs'],mydata['cesd']);
print("\nPearson corelation:")
print(r, '\n')

# fit simple linear regression model
linear_model = ols('cesd ~ mcs', data=mydata).fit()
print(linear_model.summary())
print('\nHigher accuracy printout of p-values: \n',linear_model.pvalues, '\n')
```

Pearson corelation:

PearsonRResult(statistic=-0.6819239139610712, pvalue=3.0189415357078893e-63)

#### OLS Regression Results

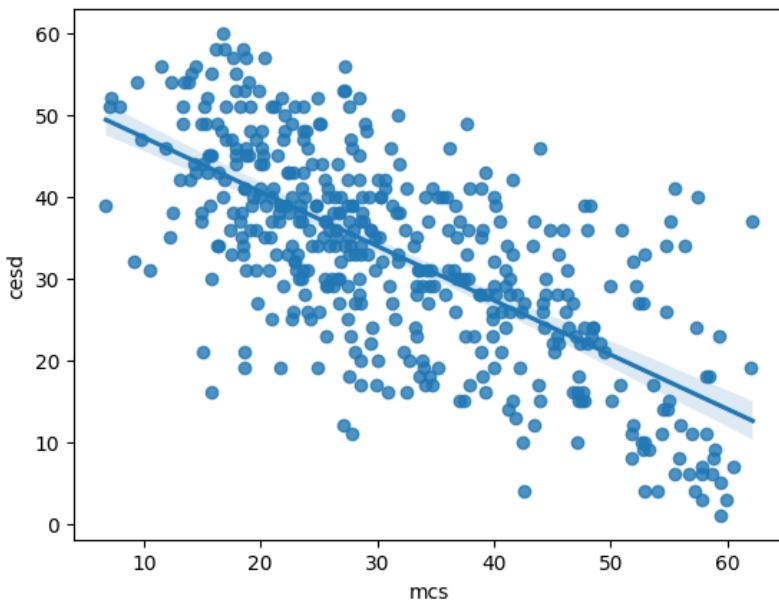
```
=====
Dep. Variable:          cesd    R-squared:       0.465
Model:                 OLS     Adj. R-squared:   0.464
Method:                Least Squares F-statistic:    392.0
Date:      Fri, 19 Apr 2024 Prob (F-statistic): 3.02e-63
Time:      19:50:29   Log-Likelihood:   -1645.3
No. Observations:      453    AIC:             3295.
Df Residuals:          451    BIC:             3303.
Df Model:                  1
Covariance Type:        nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
<hr/>						
Intercept	53.9022	1.147	46.984	0.000	51.648	56.157
mcs	-0.6647	0.034	-19.800	0.000	-0.731	-0.599
<hr/>						
Omnibus:		0.768	Durbin-Watson:		2.001	
Prob(Omnibus):		0.681	Jarque-Bera (JB):		0.847	
Skew:		-0.025	Prob(JB):		0.655	
Kurtosis:		2.794	Cond. No.		91.1	
<hr/>						

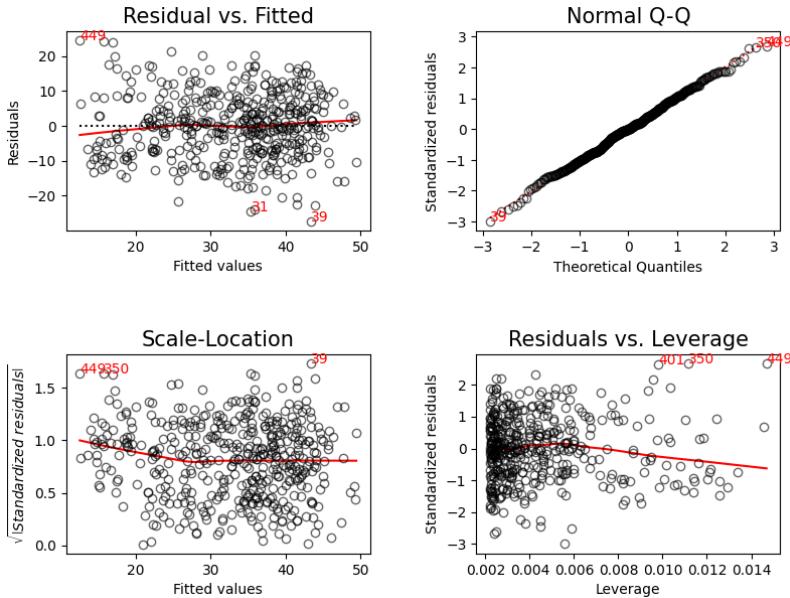
Notes:

[1] Standard Errors assume that the covariance matrix of the errors is incorrectly specified.

```
p-values::  
  Intercept    7.491255e-176  
  mcs          3.018942e-63  
  dtype: float64
```



```
[ ]: plt.figure(figsize=(8,6))  
import lmdiag  
lmdiag.plot(linear_model);
```



Based on the diagnostic plots and the data with regression line Figure above, the following conclusions can be made. The assumptions of regression: linear trend (no pattern of residuals), normality of residuals (QQ plot), constant variability (no funnel pattern), and independent observations (no time series) are all satisfied. In the Residuals vs. Leverage plot, none of the points go over Cook's distance level curves (they are not even visible); so none of the observations are too influential. Thus, the linear model is applicable.

Pearson Correlation coefficient  $r = -0.682$  corresponds to a moderate negative correlation.

Correlation hypothesis test:

H<sub>0</sub>:  $\rho = 0$  - no linear correlation

H<sub>1</sub>:  $\rho \neq 0$  - there is linear correlation

The t-test produces  $t = -19.8$  and the corresponding  $pvalue \approx 0 < 0.05$ ; therefore, H<sub>0</sub> can be rejected, i.e., there is a significant linear correlation.

The coefficient of variation is  $r^2 = 0.465$ ; therefore, 46.5% of variation in `cesd` is explained by this linear relationship with `mcs` (coefficient of determination).

Slope  $b_1 = -0.665$ , so for each extra point of `mcs` score, the depression score `cesd` decreases by 0.665 *on average*. The hypothesis test for the slope:

H<sub>0</sub>:  $\beta_1 = 0$  - zero slope, no linear relationship

H<sub>1</sub>:  $\beta_1 \neq 0$  - non-zero slope, there is a linear relationship.

The same t-test  $t = -19.8$  and  $pvalue \approx 0 < 0.05$  lead to the rejection of H<sub>0</sub>, i.e., there is a significant linear relationship.

The 95% confidence interval for the slope  $(-0.73, -0.6)$  doesn't contain 0, which is yet another indication of the non-zero slope.

Having confirmed in several ways that the linear model  $\hat{y} = 53.90 - 0.665 \cdot x$  is appropriate, let's predict with it:

```
[ ]: xs = np.linspace(10, 70, num=7)
predictions = linear_model.get_prediction(pd.DataFrame({'mcs':xs}))
df = pd.DataFrame(predictions.summary_frame(alpha=0.05))
df.insert(loc=0, column='xs', value=xs); df
```

	xs	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	10.0	47.255498	0.845510	45.593869	48.917127	29.170530	65.340466
1	20.0	40.608808	0.582251	39.464547	41.753070	22.564019	58.653597
2	30.0	33.962118	0.434203	33.108806	34.815430	15.933441	51.990796
3	40.0	27.315428	0.513260	26.306751	28.324106	9.278730	45.352127
4	50.0	20.668738	0.750817	19.193204	22.144273	2.599918	38.737559
5	60.0	14.022048	1.043746	11.970840	16.073257	-4.102866	32.146963
6	70.0	7.375358	1.356640	4.709239	10.041478	-10.829401	25.580118

The output above shows both tighter mean predictions and wider individual predictions.

We can also compute a residual for a specific value slightly outside of the data set (reasonable extrapolation). The typical residual error using this linear model is given by Residual standard error  $s_e$  computed below.

```
[ ]: xs = 70; yexact = 12; b = linear_model.params; ys = b[0] + b[1]*xs;
print('xs = {:.3f}, ys = {:.3f}, yexact = {:.3f}, residual = {:.3f}'.
      format(xs,ys,yexact,yexact-ys))
se = np.sqrt(linear_model.scale);
print('Residual standard error = se = {:.3f}'.format(se))
```

```
xs = 70.000, ys = 7.375, yexact = 12.000, residual = 4.625
Residual standard error = se = 9.164
```

## Example

Investigate the correlation and regression of a particular stock return vs. the S&P500 stock average. Predict several typical values.

```
[ ]: # import packages and libraries
import numpy as np; import pandas as pd
import matplotlib.pyplot as plt; import seaborn as sns;
import statsmodels.api as sm; from scipy.stats import pearsonr
from statsmodels.formula.api import ols
```

```

mydata = pd.DataFrame({'ReturnsABC':[0.11, 0.06, -0.08, 0.12, 0.07, 0.08,
                                     -0.10, 0.09, 0.06, -0.08, 0.04, 0.11,
                                     -0.12, -0.02, 0.03, -0.06, -0.10, -0.
                                     ↪01],
                       'Returns500':[0.20, 0.18, -0.14, 0.18, 0.13, 0.12,
                                     -0.20, 0.14, 0.13, -0.17, 0.04, 0.14,
                                     -0.15, 0.05, 0.04, -0.07, -0.08, -0.
                                     ↪02]})

#mydata = pd.read_csv(url)    # save as mydata file
print(mydata.head(10))
sns.regplot(data=mydata,x='Returns500',y='ReturnsABC')

r = pearsonr(mydata['ReturnsABC'],mydata['Returns500']);
print("\nPearson corelation:")
print(r, '\n')

# fit simple linear regression model
linear_model = ols('ReturnsABC~Returns500', data=mydata).fit()
print(linear_model.summary())
print('\nHigher accuracy printout of p-values: \n',linear_model.pvalues, '\n')

```

	ReturnsABC	Returns500
0	0.11	0.20
1	0.06	0.18
2	-0.08	-0.14
3	0.12	0.18
4	0.07	0.13
5	0.08	0.12
6	-0.10	-0.20
7	0.09	0.14
8	0.06	0.13
9	-0.08	-0.17

Pearson corelation:  
PearsonRResult(statistic=0.9535123202590954, pvalue=9.475962314785722e-10)

```
/usr/local/lib/python3.10/dist-packages/scipy/stats/_stats_py.py:1806:
UserWarning: kurtosistest only valid for n>=20 ... continuing anyway, n=18
    warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

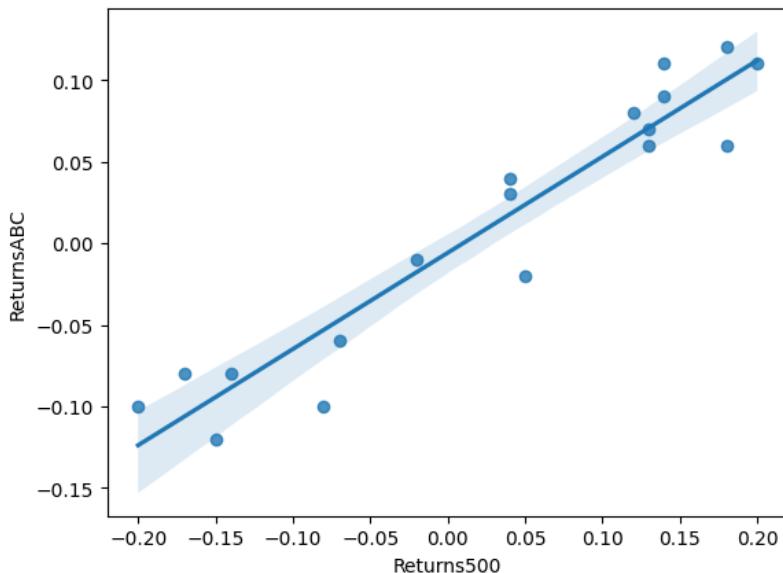
OLS Regression Results						
Dep. Variable:		ReturnsABC	R-squared:	0.909		
Model:		OLS	Adj. R-squared:	0.904		
Method:		Least Squares	F-statistic:	160.2		
Date:	Fri, 19 Apr 2024	Prob (F-statistic):	9.48e-10			
Time:	19:50:57	Log-Likelihood:	41.390			
No. Observations:	18	AIC:	-78.78			
Df Residuals:	16	BIC:	-77.00			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]

```
Intercept      -0.0059      0.006     -0.952      0.355      -0.019      0.007
Returns500      0.5894      0.047     12.656      0.000      0.491      0.688
=====
Omnibus:            2.225   Durbin-Watson:          1.754
Prob(Omnibus):    0.329   Jarque-Bera (JB):       1.801
Skew:              -0.679   Prob(JB):             0.406
Kurtosis:           2.254   Cond. No.            7.68
=====
```

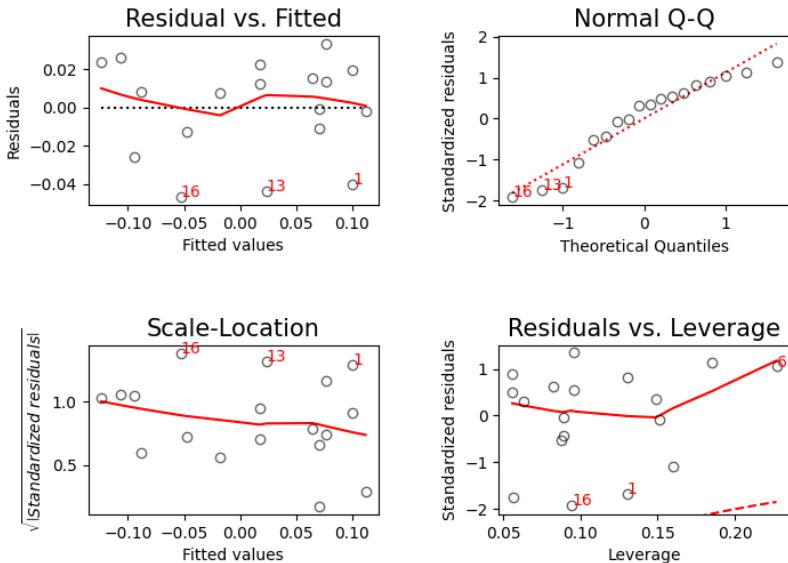
Notes:

[1] Standard Errors assume that the covariance matrix of the errors is  $\square$   
correctly  
specified.

p-values::  
Intercept 3.552459e-01  
Returns500 9.475962e-10  
dtype: float64



```
[ ]: plt.figure(figsize=(7,5))
import lmdiag
lmdiag.plot(linear_model);
```



Based on the diagnostic plots and the data with regression line Figure above, the following conclusions can be made. The assumptions of regression: linear trend in the data (no pattern of residuals), normality of residuals (QQ plot), constant variability (no funnel pattern), independent observations (no time series) are satisfied. In the Residuals vs. Leverage plot, none of the observations cross over Cook's distance level curves. Thus, we can apply the linear model to this data.

Pearson Correlation coefficient  $r = 0.954$  indicates a strong positive correlation.

Correlation hypothesis test:

H<sub>0</sub>:  $\rho = 0$  - no linear correlation

H<sub>1</sub>:  $\rho \neq 0$  - there is a linear correlation.

The t-test produces  $t = 12.66$  and corresponding  $pvalue \approx 0 < 0.05$ ; therefore, H<sub>0</sub> can be rejected and there is a significant linear correlation.

The coefficient of determination is  $r^2 = 0.909$ , so 90.9% of the variation in returns of the stock ABC is explained by this linear relationship with returns of S&P500.

Slope  $b_1 = 0.59$ , so for each extra point of return of S&P500, the returns of stock ABC increase by 0.59 on average.

The hypothesis test for the slope:

H<sub>0</sub>:  $\beta_1 = 0$  - zero slope, no linear relationship

H<sub>1</sub>:  $\beta_1 \neq 0$  - non-zero slope, there is a linear relationship

The t-test is same  $t = 12.66$  and  $pvalue \approx 0 < 0.05$ , so H<sub>0</sub> can be rejected,

and there is a significant linear relationship.

The 95% confidence interval for the slope (0.49, 0.69) doesn't contain 0, which is yet another indication of the non-zero slope.

Having confirmed in several ways that the linear model  $\hat{y} = -0.0059 + 0.5894 \cdot x$  fits data well, use it for prediction.

```
[ ]: xs = np.linspace(-0.3, 0.3, num=7)
predictions = linear_model.get_prediction(pd.DataFrame({'Returns500':xs}))
df = pd.DataFrame(predictions.summary_frame(alpha=0.05))
df.insert(loc=0, column='xs', value=xs); df
```

	xs	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	-0.3	-0.182752	0.016476	-0.217679	-0.147825	-0.247548	-0.117955
1	-0.2	-0.123807	0.012266	-0.149810	-0.097804	-0.184263	-0.063351
2	-0.1	-0.064862	0.008536	-0.082957	-0.046768	-0.122361	-0.007363
3	0.0	-0.005917	0.006216	-0.019094	0.007259	-0.062063	0.050228
4	0.1	0.053027	0.006913	0.038372	0.067683	-0.003483	0.109538
5	0.2	0.111972	0.010017	0.090738	0.133206	0.053410	0.170535
6	0.3	0.170917	0.014009	0.141219	0.200615	0.108783	0.233051

In the code below, we investigate a residual for a particular value. Also, the typical residual error is given by Residual standard error  $s_e$ .

```
[ ]: xs = 0.3; yexact = 0.15; b = linear_model.params; ys = b[0] + b[1]*xs;
print('xs = {:.3f}, ys = {:.3f}, yexact = {:.3f}, residual = {:.3f}'.format(xs,ys,yexact,yexact-ys))
se = np.sqrt(linear_model.scale);
print('Residual standard error = se = {:.3f}'.format(se))
```

```
xs = 0.300, ys = 0.171, yexact = 0.150, residual = -0.021
Residual standard error = se = 0.026
```

For the standardized stock returns in this problem, the slope of the linear model  $b_1 = 0.59$  gives **beta of a stock**, which measures the volatility of the stock relative to the stock market as a whole (represented via S&P500). Thus, this ABC stock is about 60% as volatile as the market.

## 8.4 Linearized Models - Exponential and Logarithmic Transformations

Some non-linear relationships can be linearized with appropriate transformations.

### 8.4.1 Exponential Regression

Suppose the relationship between two variables is best described by an exponential function of the form

$$y = a \cdot e^{bx} \quad (8.16)$$

where  $a$  and  $b$  are parameters to be found.

Apply  $\ln()$  (natural log) to both sides:

$$\ln(y) = \ln(a \cdot e^{bx}) = \ln(a) + \ln(e^{bx}) = \ln(a) + bx \quad (8.17)$$

Therefore  $\ln(y)$  is linear in  $x$ .

As an example, consider Moore's law stating that the number of transistors per chip would double every 18 months, which results in an exponential model. The code below uses real data from 1975 to 1995 and compares linear and exponential fits.

```
[ ]: import numpy as np; import pandas as pd
import matplotlib.pyplot as plt; import seaborn as sns;
import statsmodels.api as sm; from scipy.stats import pearsonr
from statsmodels.formula.api import ols

mydata = pd.DataFrame({'year': [1975,1978,1982,1985,1989,1993,1995],
                      'x':[0,3,7,10,14,18,20],
                      'y':[4500,29000,90000,229000,1200000,3100000,5500000]})

mydata.head()
```

```
[ ]:      year     x      y
0   1975     0    4500
1   1978     3   29000
2   1982     7   90000
3   1985    10  229000
4   1989    14 1200000
```

```
[ ]: sns.regplot(data=mydata,x='x',y='y')

r = pearsonr(mydata['x'],mydata['y']);
print("Pearson correlation : ")
print(r)

# fit simple linear regression model
linear_model = ols('y ~ x', data=mydata).fit()
```

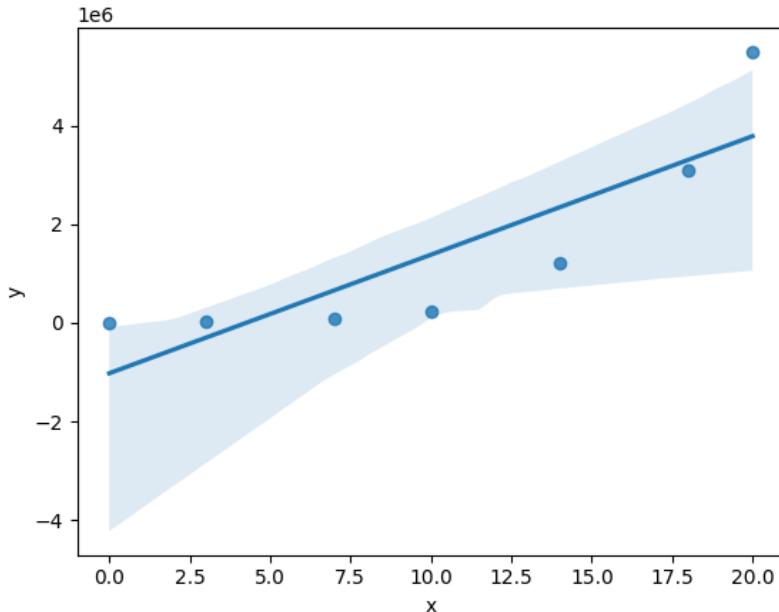
```
print(linear_model.summary())
print('\nHigher accuracy printout of p-values: \n',linear_model.pvalues,'\\n')
```

```
Pearson correlation :
PearsonRResult(statistic=0.8562985042360467, pvalue=0.01389401025685245)
OLS Regression Results
=====
Dep. Variable:                      y      R-squared:                 0.733
Model:                            OLS      Adj. R-squared:            0.680
Method:                           Least Squares      F-statistic:               13.74
Date:                          Fri, 19 Apr 2024      Prob (F-statistic):        0.0139
Time:                            20:24:39      Log-Likelihood:           -106.69
No. Observations:                  7      AIC:                     217.4
Df Residuals:                      5      BIC:                     217.3
Df Model:                           1
Covariance Type:                nonrobust
=====
            coef    std err          t      P>|t|      [0.025      0.975]
-----
Intercept   -1.024e+06   8.05e+05     -1.272      0.259     -3.09e+06    1.05e+06
x            2.406e+05   6.49e+04      3.707      0.014      7.38e+04    4.07e+05
=====
Omnibus:                      nan      Durbin-Watson:             0.872
Prob(Omnibus):                  nan      Jarque-Bera (JB):         0.583
Skew:                           0.409      Prob(JB):                  0.747
Kurtosis:                        1.846      Cond. No.                   22.3
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is incorrectly specified.

```
p-values::
Intercept    0.259373
x            0.013894
dtype: float64
```



The linear fit is clearly very poor as shown in the Figure above; still, it is statistically significant with the slope's p-value below 0.05. Therefore a scatterplot must always be examined.

Now, consider the exponential fit with  $\ln(y)$ .

```
[ ]: mydata['logy'] = np.log(mydata['y']) # adding logy variable to data frame
sns.regplot(data=mydata,x='x',y='logy')

r = pearsonr(mydata['x'],mydata['logy']);
print("Pearson correlation = ", r)

# fit simple linear regression model
linear_model = ols('logy ~ x', data=mydata).fit()
print(linear_model.summary())
print('\nHigher accuracy printout of p-values: \n',linear_model.pvalues, '\n')

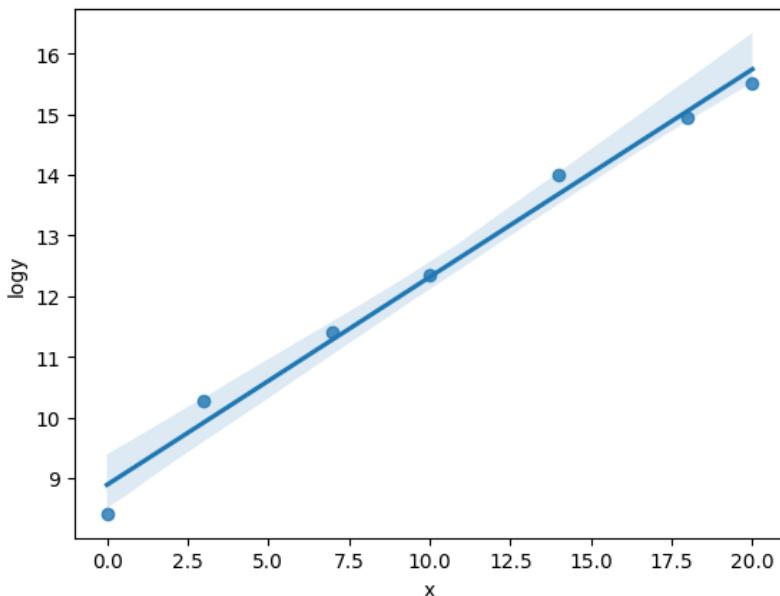
Pearson correlation = PearsonRResult(statistic=0.9933910639496113,
pvalue=6.795843315161991e-06)
                OLS Regression Results
=====
Dep. Variable:          logy    R-squared:       0.987
Model:                 OLS    Adj. R-squared:   0.984
Method:                Least Squares    F-statistic:     374.5
Date:      Fri, 19 Apr 2024    Prob (F-statistic):  6.80e-06
Time:      20:24:43    Log-Likelihood:   -0.89576
No. Observations:      7    AIC:             5.792
```

```
Df Residuals: 5 BIC: 5.683
Df Model: 1
Covariance Type: nonrobust
=====
            coef    std err      t    P>|t|   [0.025    0.975]
Intercept  8.8884    0.220    40.434    0.000    8.323    9.453
x          0.3428    0.018    19.353    0.000    0.297    0.388
=====
Omnibus:           nan Durbin-Watson: 1.955
Prob(Omnibus):    nan Jarque-Bera (JB): 0.394
Skew:             -0.298 Prob(JB): 0.821
Kurtosis:         2.003 Cond. No. 22.3
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is u  
→correctly  
specified.

```
p-values::
Intercept 1.744644e-07
x          6.795843e-06
dtype: float64
```



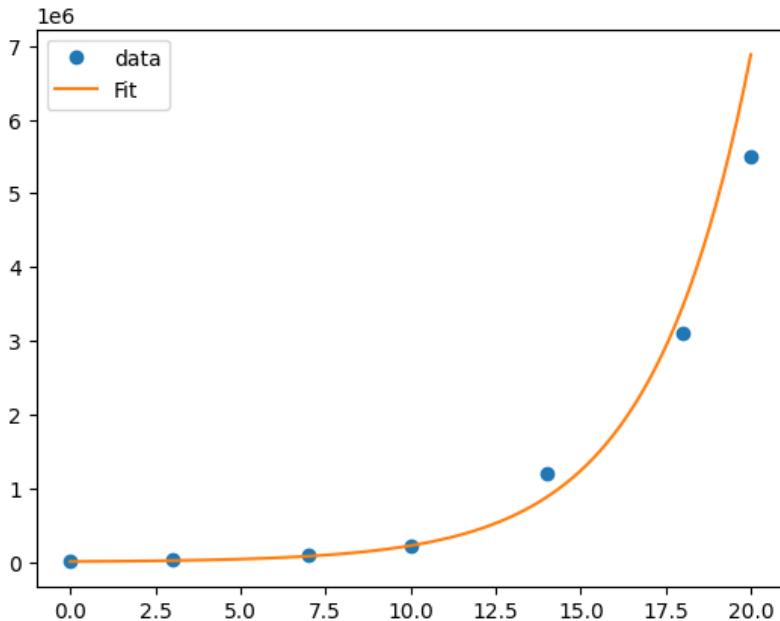
The exponential fit in  $\ln(y)$  vs  $x$  coordinates is very good as shown in the Figure above.

Once the linear model is fit, we can express the model in original coordinates and obtain an exponential fit graph in regular x-y coordinates as shown in the Figure below:

```
[ ]: a = np.exp(linear_model.params.values[[0]]);
b = linear_model.params.values[[1]];
# fit function
def f(x, a, b):
    return a*np.exp(b*x)

plt.plot(mydata['x'], mydata['y'], 'o', label='data')
xv = np.linspace(start=0, stop=20, num=100)
plt.plot(xv, f(xv, a, b), '-', label='Fit')
plt.legend()
```

[ ]: <matplotlib.legend.Legend at 0x7e050ea47820>



The doubling time  $h$  can be found by equating

$$e^{bh} = 2 \implies bh = \ln(2) \implies h = \frac{\ln(2)}{b}$$

Therefore, the doubling time is given by:

```
[ ]: doubling_time = np.log(2)/b;
print(doubling_time)
```

```
[2.02195535]
```

It is only a bit higher than Moore's original prediction of 18 months.

### 8.4.2 Logarithmic Regression

Another common non-linear model that linearizes well is a power model:

$$y = a \cdot x^b \quad (8.18)$$

Apply natural log to both sides of this equation and obtain:

$$\ln(y) = \ln(a \cdot x^b) = \ln(a) + \ln(x^b) = \ln(a) + b \ln(x) \quad (8.19)$$

Therefore  $\ln(y)$  is a linear function of  $\ln(x)$ .

For example, the table below shows the mean distance from the Sun ( $x$ ) in astronomical units (Earth distance is 1) and period ( $y$ ) in Earth years. The code below fits power model to the data and compares it to the linear model.

```
[ ]: import numpy as np; import pandas as pd
import matplotlib.pyplot as plt; import seaborn as sns;
import statsmodels.api as sm; from scipy.stats import pearsonr
from statsmodels.formula.api import ols

mydata = pd.DataFrame({'Planet':
    ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn'],
    'x':[0.39, 0.72, 1.00, 1.52, 5.20, 9.54],
    'y':[0.24, 0.62, 1.00, 1.89, 11.86, 29.46]})

mydata
```

Planet	x	y
Mercury	0.39	0.24
Venus	0.72	0.62
Earth	1.00	1.00
Mars	1.52	1.89
Jupiter	5.20	11.86
Saturn	9.54	29.46

```
[ ]: sns.regplot(data=mydata,x='x',y='y')

r = pearsonr(mydata['x'],mydata['y']);
print("Pearson correlation: ")
print(r,'\\n')

# fit simple linear regression model
linear_model = ols('y ~ x', data=mydata).fit()
print(linear_model.summary())
print('\\nHigher accuracy printout of p-values: \\n',linear_model.pvalues,'\\n')
```

Pearson correlation:  
PearsonRResult(statistic=0.9924524861041891, pvalue=8.523247707493205e-05)

```

OLS Regression Results
=====
Dep. Variable:                      y      R-squared:       0.985
Model:                            OLS      Adj. R-squared:   0.981
Method:                           Least Squares      F-statistic:     262.0
Date:                     Fri, 19 Apr 2024      Prob (F-statistic): 8.52e-05
Time:                         20:26:42      Log-Likelihood:  -10.091
No. Observations:                  6      AIC:             24.18
Df Residuals:                      4      BIC:            23.77
Df Model:                           1
Covariance Type:            nonrobust
=====

      coef    std err        t      P>|t|      [0.025      0.975]
-----
Intercept    -2.2213     0.886     -2.508     0.066     -4.681     0.238
x             3.1790     0.196     16.186     0.000      2.634     3.724
=====

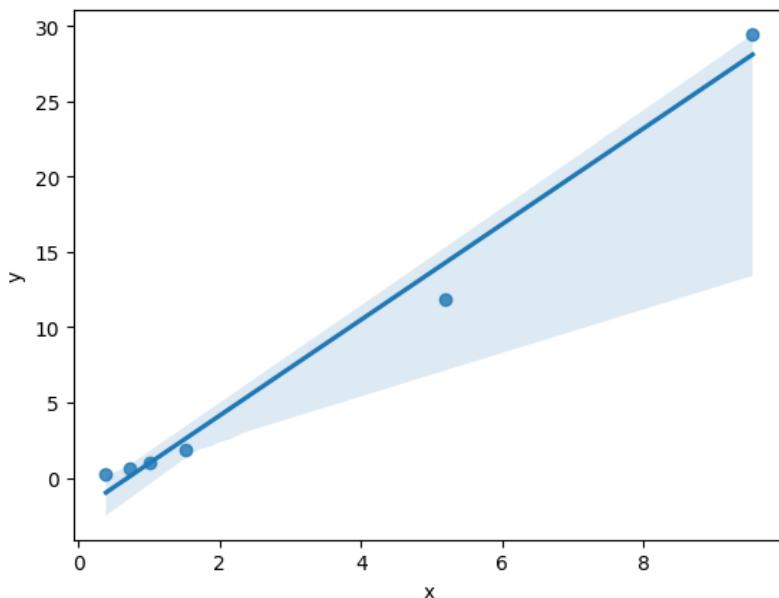
Omnibus:                      nan      Durbin-Watson:     1.846
Prob(Omnibus):                 nan      Jarque-Bera (JB):  0.722
Skew:                          -0.803     Prob(JB):        0.697
Kurtosis:                      2.442     Cond. No.       6.29
=====
```

## Notes:

[1] Standard Errors assume that the covariance matrix of the errors is incorrectly  
specified.

```

p-values::
Intercept    0.066209
x            0.000085
dtype: float64
```



The linear fit is very poor again (see Figure above) but still statistically significant with a p-value well below 0.05. As always, a scatterplot must be investigated.

```
[ ]: mydata['logx'] = np.log(mydata['x'])
mydata['logy'] = np.log(mydata['y'])
sns.regplot(data=mydata,x='logx',y='logy')

r = pearsonr(mydata['logx'],mydata['logy']);
print("Pearson correlation: ")
print(r, '\n')

# fit simple linear regression model
linear_model = ols('logy ~ logx', data=mydata).fit()
print(linear_model.summary())
print('\nHigher accuracy printout of p-values: \n',linear_model.pvalues, '\n')
```

```
Pearson correlation:
PearsonRResult(statistic=0.9999855135331572, pvalue=3.147850623314139e-10)
```

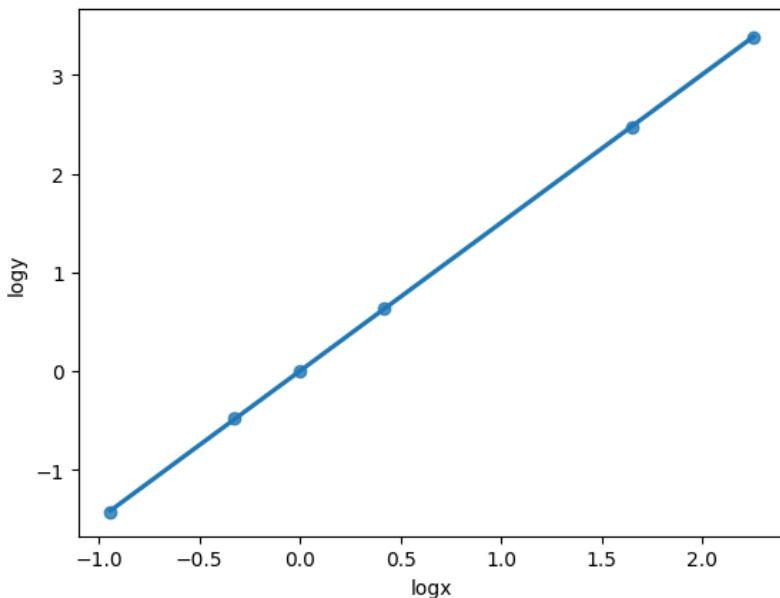
OLS Regression Results			
Dep. Variable:	logy	R-squared:	1.000
Model:	OLS	Adj. R-squared:	1.000
Method:	Least Squares	F-statistic:	1.381e+05
Date:	Fri, 19 Apr 2024	Prob (F-statistic):	3.15e-10
Time:	20:26:45	Log-Likelihood:	19.759
No. Observations:	6	AIC:	-35.52

```
Df Residuals:                 4     BIC:                  -35.93
Df Model:                      1
Covariance Type:      nonrobust
=====
            coef    std err        t     P>|t|    [0.025    0.975]
Intercept    0.0009    0.005     0.175    0.870   -0.013    0.015
logx         1.5011    0.004    371.560    0.000    1.490    1.512
=====
Omnibus:                     nan   Durbin-Watson:       2.505
Prob(Omnibus):                nan   Jarque-Bera (JB):  0.138
Skew:                         0.025   Prob(JB):        0.933
Kurtosis:                      2.259   Cond. No.        1.63
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is u  
↳correctly  
specified.

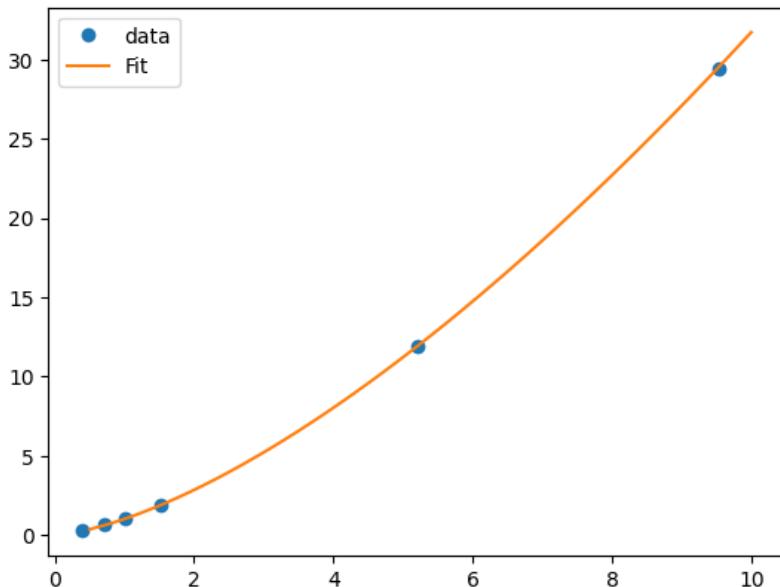
```
p-values::
Intercept    8.697703e-01
logx        3.147851e-10
dtype: float64
```



```
[ ]: a = np.exp(linear_model.params.values[[0]]);
b = linear_model.params.values[[1]];
# fit function
def f(x, a, b):
    return a*x**b

plt.plot(mydata['x'], mydata['y'], 'o', label='data')
xv = np.linspace(start=0.5, stop =10, num=100)
plt.plot(xv, f(xv, a, b), '--', label='Fit')
plt.legend()
```

[ ]: <matplotlib.legend.Legend at 0x7e050faca950>



The power function fit is very good as shown in the Figure above. It gives

$$\text{period } (y) = (\text{distance } (x))^{1.5}$$

which is exactly Kepler's law.

### 8.4.3 S-Shaped (Logistic) Curve

Moore's law in previous sections corresponds to the unabated exponential growth of the number of transistors in computer chips. However, there is a limit to how many transistors can be physically fit in a chip. In a more realistic growth model, the initial exponential growth tapers off. For example, the population of bacteria, the spread of viruses or rumors, the proliferation

of global markets, and acceptance of new reform all follow this paradigm. The logistic equation (S-shaped curve) describes such data well:

$$y = \frac{L}{1 + e^{a+bx}} \quad (8.20)$$

As  $x$  increases,  $y$  is initially increasing exponentially and concave up, but later tapers off to a concave down curve converging to  $L$  that is called the **population carrying capacity** (estimated graphically). To linearize this equation

$$\frac{1}{y} = \frac{1 + e^{a+bx}}{L} \implies \frac{L}{y} - 1 = e^{a+bx} \implies \frac{L-y}{y} = e^{a+bx} \implies \quad (8.21)$$

$$\ln\left(\frac{L-y}{y}\right) = a + bx \quad (8.22)$$

Therefore  $\ln\left(\frac{L-y}{y}\right)$  is linear with  $x$ .

Consider the growth of bacteria in a limited environment (like a petri dish). The initial exponential growth cannot be sustained. Limited resources and the buildup of toxins decrease the rate of growth. The curve begins to concave up, inflects to concave down, and approaches an asymptotic limit, which looks like an **S-shape**. In these data, the population is converging to about 3000; therefore, the carrying capacity is taken to be  $L = 3000$ . In the code, we simulated the data adding normal variation to the logistic model.

```
[1]: import numpy as np; import pandas as pd
import matplotlib.pyplot as plt; import seaborn as sns;
import statsmodels.api as sm; from scipy.stats import pearsonr
from statsmodels.formula.api import ols

L = 3000; a = 3; b=-0.1
x = np.arange(0,100,5);
n = len(x)
y = L/(1+np.exp(a+b*x)) + 3*np.random.normal(0,1,n)
mydata = pd.DataFrame({'x':x,'y':y})
pd.set_option("display.precision", 4);
print(mydata.head(),'\n')
```

	x	y
0	0	146.7065
1	5	227.8411
2	10	360.8243
3	15	554.7068
4	20	810.6091

```
[2]: sns.regplot(data=mydata,x='x',y='y')
r = pearsonr(mydata['x'],mydata['y']);
```

```

print("Pearson correlation = ")
print(r)

# fit simple linear regression model
linear_model = ols('y ~ x', data=mydata).fit()
print(linear_model.summary())
print('\nHigher accuracy printout of p-values: \n',linear_model.pvalues,\n')

```

```

Pearson correlation =
PearsonRResult(statistic=0.9389555271027002, pvalue=8.947154877473107e-10)
OLS Regression Results
=====
Dep. Variable:                      y      R-squared:                 0.882
Model:                            OLS      Adj. R-squared:            0.875
Method:                           Least Squares      F-statistic:             134.1
Date:        Sun, 21 Apr 2024      Prob (F-statistic):       8.95e-10
Time:          19:39:32      Log-Likelihood:           -146.29
No. Observations:                  20      AIC:                   296.6
Df Residuals:                     18      BIC:                   298.6
Df Model:                          1
Covariance Type:                nonrobust
=====
      coef    std err        t      P>|t|      [0.025      0.975]
-----
Intercept    379.9340     165.107      2.301      0.034     33.057    726.811
x            34.4061      2.971     11.579      0.000     28.163     40.649
=====
Omnibus:                   3.951      Durbin-Watson:            0.121
Prob(Omnibus):              0.139      Jarque-Bera (JB):         1.395
Skew:                      0.057      Prob(JB):               0.498
Kurtosis:                   1.711      Cond. No.                107.
=====
```

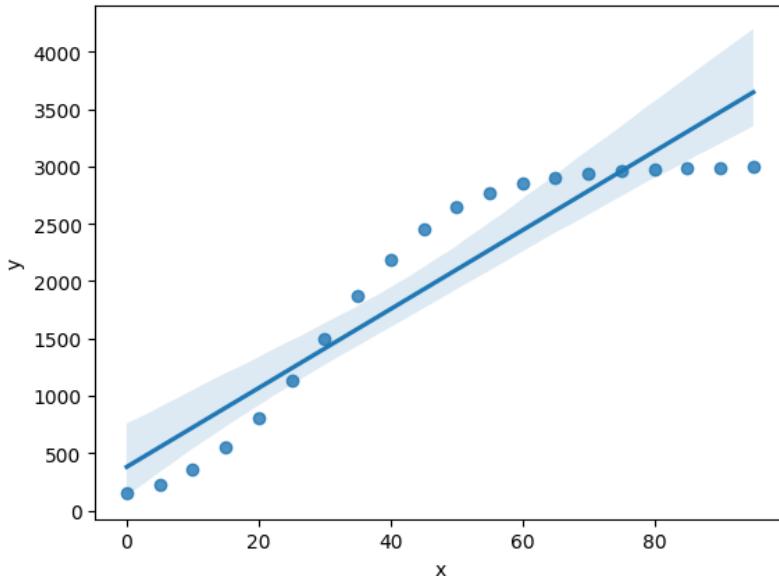
## Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```

p-values::
Intercept    3.3551e-02
x            8.9472e-10
dtype: float64

```



As before, the linear fit is very poor but statistically significant with a p-value well below 0.05; only the scatterplot above clarifies it. The logistic function fit is very good as shown below.

```
[4]: mydata['logLy'] = np.log(np.abs(L-mydata['y'])/mydata['y'])
sns.regplot(data=mydata,x='x',y='logLy')

r = pearsonr(mydata['x'],mydata['logLy']);
print("Pearson correlation = ")
print(r,'\n')

# fit simple linear regression model
linear_model = ols('logLy ~ x', data=mydata).fit()
print(linear_model.summary())
print('\nHigher accuracy printout of p-values: \n',linear_model.pvalues,'\\n')
```

```
Pearson correlation =
PearsonRResult(statistic=-0.9930846432128577, pvalue=3.3500306084557878e-18)
```

#### OLS Regression Results

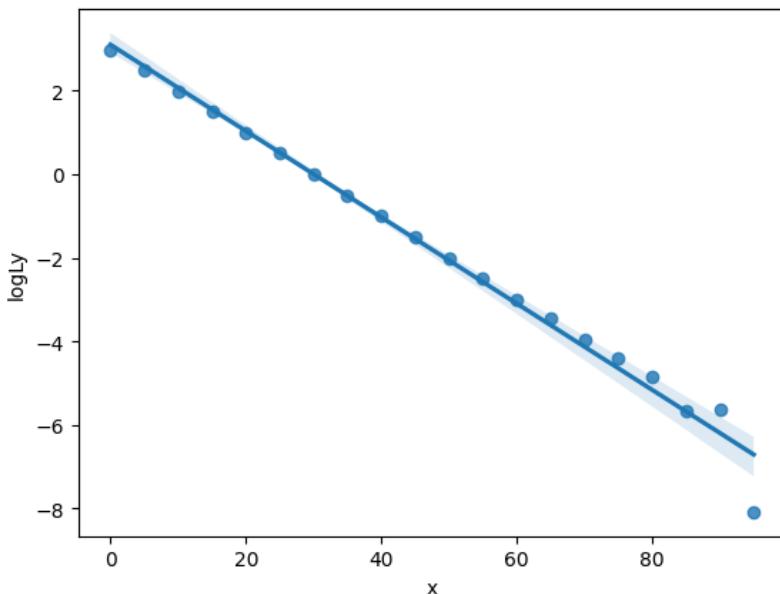
Dep. Variable:	logLy	R-squared:	0.986
Model:	OLS	Adj. R-squared:	0.985
Method:	Least Squares	F-statistic:	1288.
Date:	Sun, 21 Apr 2024	Prob (F-statistic):	3.35e-18
Time:	19:40:15	Log-Likelihood:	-7.4936
No. Observations:	20	AIC:	18.99
Df Residuals:	18	BIC:	20.98

```
Df Model: 1
Covariance Type: nonrobust
=====
            coef    std err      t    P>|t|    [0.025    0.975]
Intercept  3.0991   0.160   19.384   0.000    2.763    3.435
x         -0.1033   0.003  -35.888   0.000   -0.109   -0.097
=====
Omnibus:      35.230  Durbin-Watson:      1.679
Prob(Omnibus): 0.000  Jarque-Bera (JB):  94.154
Skew:        -2.738  Prob(JB):       3.59e-21
Kurtosis:     12.110  Cond. No.        107.
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is  $\square$   
 $\hookrightarrow$  correctly specified.

```
p-values:
Intercept  1.6513e-13
x          3.3500e-18
dtype: float64
```



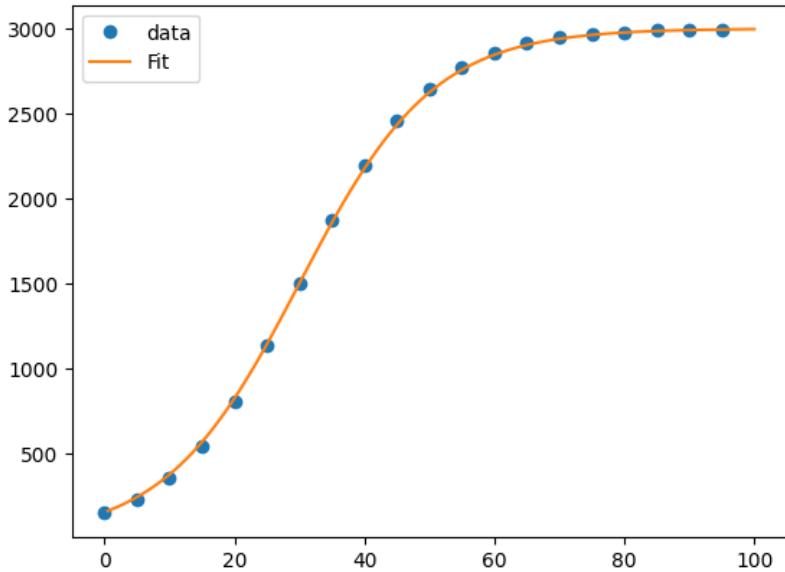
```
[ ]: a = linear_model.params.values[[0]];
b = linear_model.params.values[[1]];
# fit function
```

```

def f(x, a, b, L):
    return L/(1+np.exp(a+b*x))

plt.plot(mydata['x'], mydata['y'], 'o', label='data')
xv = np.linspace(start=0.5, stop =100, num=100)
plt.plot(xv, f(xv, a, b, L), '--', label='Fit')
plt.legend()

```



The Figures above show very good fit, both in the transformed (linearized) and the original coordinates.

## 8.5 Categorical Predictors

So far, both explanatory (x) and response variables (y) were numerical. However, the linear regression method can be applied to categorical explanatory variables, but the response variable should still be numerical. The categorical response variable requires a special approach of logistic regression considered in 2nd part of the book.

## Example

Investigate linear regression model of cesd by sex in the HELPrct data file.

```
[5]: # import packages and libraries
import numpy as np; import pandas as pd
import matplotlib.pyplot as plt; import seaborn as sns;
import statsmodels.api as sm; from scipy.stats import pearsonr
from statsmodels.formula.api import ols

url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrct.csv"
mydata = pd.read_csv(url)    # save as mydata file
# print(mydata.head(10))
sns.boxplot(data=mydata,x='sex',y='cesd')

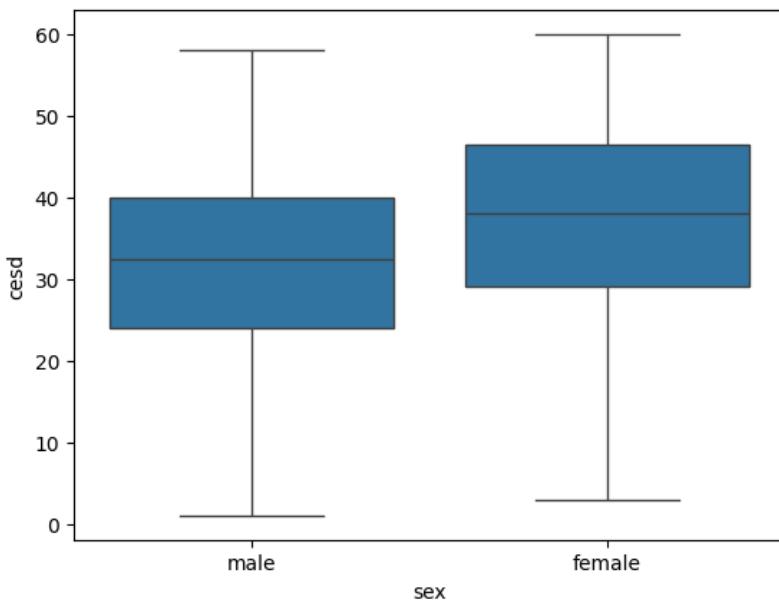
# fit simple linear regression model
linear_model = ols('cesd ~ sex', data=mydata).fit()
print(linear_model.summary())
print('\nHigher accuracy printout of p-values: \n',linear_model.pvalues,'\\n')
```

OLS Regression Results							
		cesd	R-squared:	0.032			
Dep. Variable:		OLS	Adj. R-squared:	0.030			
Model:		Least Squares	F-statistic:	15.05			
Date:	Sun, 21 Apr 2024		Prob (F-statistic):	0.000120			
Time:	19:41:58		Log-Likelihood:	-1779.5			
No. Observations:	453		AIC:	3563.			
Df Residuals:	451		BIC:	3571.			
Df Model:	1						
Covariance Type:	nonrobust						
	coef	std err	t	P> t	[0.025	0.975]	
Intercept	36.8879	1.191	30.961	0.000	34.546	39.229	
sex[T.male]	-5.2896	1.363	-3.880	0.000	-7.969	-2.610	
Omnibus:		11.117	Durbin-Watson:		2.114		
Prob(Omnibus):		0.004	Jarque-Bera (JB):		10.248		
Skew:		-0.314	Prob(JB):		0.00595		
Kurtosis:		2.615	Cond. No.		3.90		

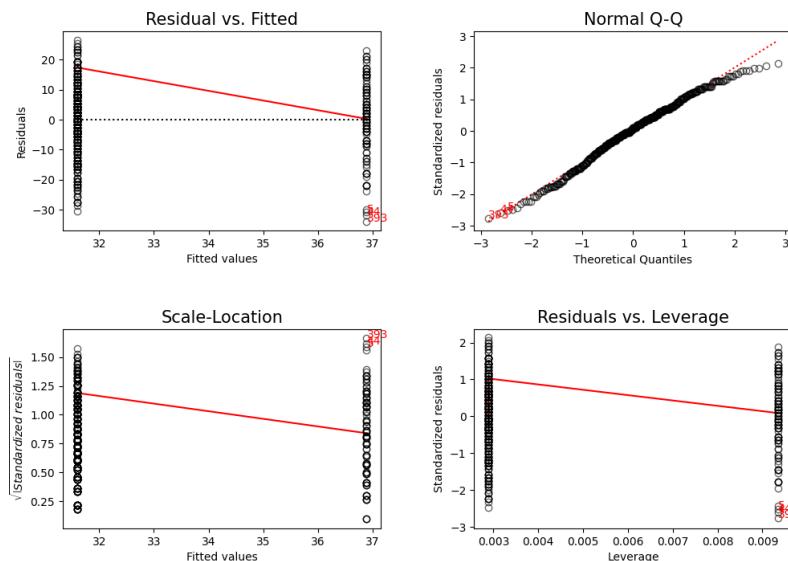
Notes:

[1] Standard Errors assume that the covariance matrix of the errors is  $\rightarrow$  correctly specified.

```
p-values::
Intercept      1.1363e-113
sex[T.male]    1.2000e-04
dtype: float64
```



```
[ ]: plt.figure(figsize=(10,7))
import lmdiag
lmdiag.plot(linear_model);
```



The Figures above show a boxplot rather than a scatterplot because the data is categorical. There are only two levels of the predictor variable `sex`, so the residual graph is quite different, but at least the variability is about the same for both groups. The QQ-plot is still not violating normality. It is not a time series and the Residuals vs. Leverage plot also shows no extreme behavior. Thus, the model can be applied to this data. Note that the correlation between numerical variable `cesd` and categorical variable `sex` cannot be considered.

The **binary yes/no categorical variable** in regression is coded as **1/0 (indicator variable)**. For our data, the base=0 level is female (F comes before M alphabetically), but this base level can be reset if needed. Our model  $\hat{y} = b_0 + b_1 \cdot x = 36.89 - 5.29 \cdot x$  can now be understood as follows: For females,  $x = \text{sex} = 0$ ; therefore, the **average** predicted `cesd` score is  $\hat{y} = b_0 = 36.89$ . For males,  $x = \text{sex} = 1$ ; therefore, the **average** predicted `cesd` score for males is  $\hat{y} = b_0 + b_1 \cdot 1 = b_0 + b_1 = 36.89 - 5.29 = 31.6$ . Thus, for binary (yes/no) categorical variable, the slope  $b_1$  is the average change in the response variable between the two categories. The code below confirms this by computing the means by sex (male/female).

```
[6]: print("Grouped means: \n", mydata.groupby(['sex'])['cesd'].mean())
```

```
Grouped means:
   sex
female    36.8879
male      31.5983
Name: cesd, dtype: float64
```

The hypothesis test for the slope:

$H_0: \beta_1 = 0$  - zero slope

$H_1: \beta_1 \neq 0$  - non-zero slope

The t-test  $t = -3.88$  and  $pvalue = 0.00012 < 0.05$  imply that  $H_0$  is rejected and there is a significant slope.

The 95% confidence interval for the slope is  $(-7.97, -2.61)$  which does not contain 0 - yet another indication of the non-zero slope.

Let's run the t-test for the comparison of the means with equal variance assumption:

```
[7]: from scipy import stats
stats.ttest_ind(mydata[mydata['sex']=='female']['cesd'],
                mydata[mydata['sex']=='male']['cesd'], equal_var=True)
```

```
[7]: TtestResult(statistic=3.8800707068561255, pvalue=0.00011999508048677403,
df=451.0)
```

The t-test  $t = -3.88$  and  $pvalue = 0.00012 < 0.05$  are the same. The difference in sample means  $\bar{x}_{\text{male}} - \bar{x}_{\text{female}} = 31.6 - 36.89 = -5.29$  is the slope of the

regression line  $b_1$ . Therefore, linear regression for binary categorical variables is equivalent to a t-test with the assumption of equal variance (regression assumes constant variance).

Having confirmed in several ways that the linear model  $\hat{y} = 36.89 - 5.29 \cdot x$  is a good fit for the data, use it for prediction. There are only two values of the explanatory variable to predict - female/male; so it just gives average values of the cesd score by gender (`sex`).

```
[9]: xs = ['female', 'male']
predictions = linear_model.get_prediction(pd.DataFrame({'sex':xs}))
df = pd.DataFrame(predictions.summary_frame(alpha=0.05))
df.insert(loc=0, column='xs', value=xs);
pd.set_option("display.precision", 2); print(df, '\n')
```

	xs	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	female	36.89	1.19	34.55	39.23	12.55	61.22
1	male	31.60	0.66	30.30	32.90	7.34	55.85

The typical residual error using this linear model is given by Residual standard error  $s_e$  computed below.

```
[11]: se = np.sqrt(linear_model.scale);
print('Residual standard error = se = {:.4f}'.format(se))
```

```
Residual standard error = se = 12.3243
```

Next, consider a categorical variable with more than two levels.

### Example

Consider the linear regression model for `cesd` by `substance` in the `HELPrc` data file.

```
[12]: # import packages and libraries
import numpy as np; import pandas as pd
import matplotlib.pyplot as plt; import seaborn as sns;
import statsmodels.api as sm; from scipy.stats import pearsonr
from statsmodels.formula.api import ols

url="https://raw.githubusercontent.com/leonkag/Statistics0/main/HELPrc.csv"
mydata = pd.read_csv(url)    # save as mydata file
# print(mydata.head(10))
sns.boxplot(data=mydata,x='substance',y='cesd')

# fit simple linear regression model
linear_model = ols('cesd ~ substance', data=mydata).fit()
print(linear_model.summary())
print('\nHigher accuracy printout of p-values: \n',linear_model.pvalues, '\n')
```

```

OLS Regression Results
=====
Dep. Variable:      cesd    R-squared:       0.038
Model:              OLS     Adj. R-squared:   0.034
Method:             Least Squares F-statistic:    8.936
Date:              Sun, 21 Apr 2024 Prob (F-statistic): 0.000156
Time:              19:43:10   Log-Likelihood: -1778.1
No. Observations:  453    AIC:            3562.
Df Residuals:      450    BIC:            3575.
Df Model:          2
Covariance Type:   nonrobust
=====

      coef      std err      t      P>|t|      [0.025
0.975]
-----
Intercept        34.3729    0.925    37.178    0.000    32.556
36.190
substance[T.cocaine] -4.9518    1.360    -3.640    0.000    -7.625
-2.279
substance[T.heroin]  0.4981    1.440     0.346    0.730    -2.333
3.329
-----
Omnibus:           8.742    Durbin-Watson:    2.091
Prob(Omnibus):    0.013    Jarque-Bera (JB):  6.623
Skew:              -0.189    Prob(JB):       0.0365
Kurtosis:          2.544    Cond. No.       3.51
=====
```

## Notes:

[1] Standard Errors assume that the covariance matrix of the errors is incorrectly  
specified.

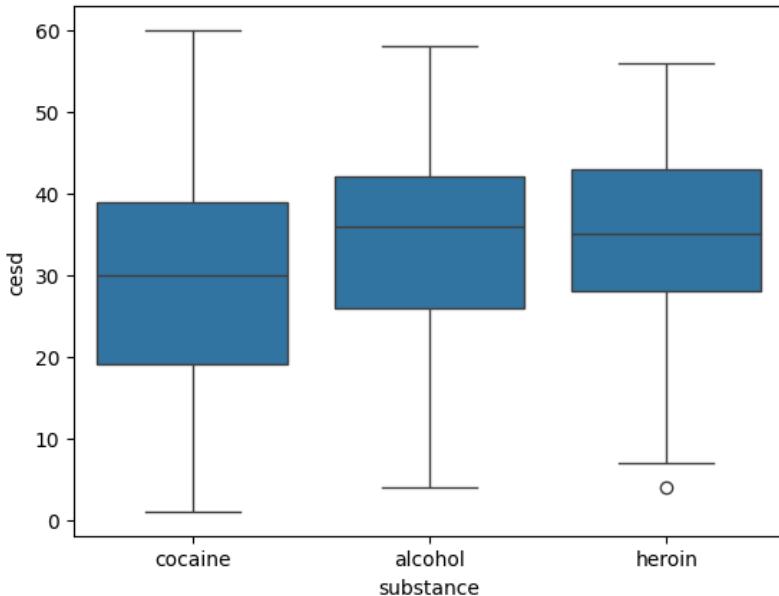
```
p-values::  

  Intercept      2.75e-139  

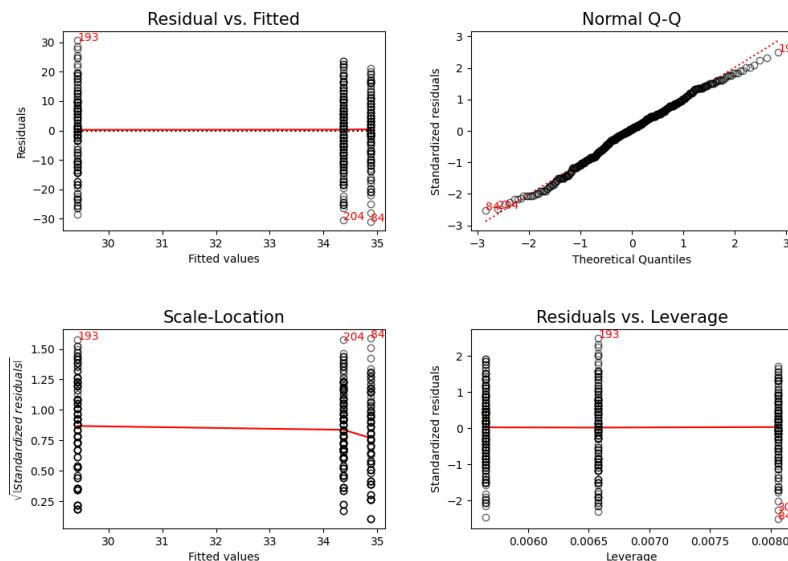
  substance[T.cocaine] 3.04e-04  

  substance[T.heroin]   7.30e-01  

  dtype: float64
```



```
[ ]: plt.figure(figsize=(10,7))
import lmdiag
lmdiag.plot(linear_model);
```



The Figures above show boxplots of the three substances and the diagnostic plots of the model. The substance variable has three levels: alcohol, cocaine, and heroin, so two indicator variables are needed. Alcohol is the base level alphabetically. The cocaine and heroin differences are given by two indicator variables:

$$\widehat{cesd} = b_0 + b_{1c} \cdot \text{substance}_{\text{cocaine}} + b_{1h} \cdot \text{substance}_{\text{heroin}}$$

$$\widehat{cesd} = 34.37 - 4.95 \cdot \text{substance}_{\text{cocaine}} + 0.50 \cdot \text{substance}_{\text{heroin}}$$

When a categorical variable that has  $k$  levels,  $k - 1$  indicator variables are needed. There are  $k - 1$  slopes relative to the reference level.

For example, if we consider an alcoholic, then the indicators for cocaine and heroin are both zero:  $\text{substance}_{\text{cocaine}} = \text{substance}_{\text{heroin}} = 0$ , so

$$\widehat{cesd} = b_0 + b_{1c} \cdot 0 + b_{1h} \cdot 0 = b_0 = 34.37$$

which is the **average** cesd depression score for alcoholics.

For cocaine abusers:

$$\widehat{cesd} = b_0 + b_{1c} \cdot 1 + b_{1h} \cdot 0 = b_0 + b_{1c} = 34.37 - 4.95 = 29.42$$

For heroin abusers:

$$\widehat{cesd} = b_0 + b_{1c} \cdot 0 + b_{1h} \cdot 1 = b_0 + b_{1h} = 34.37 + 0.5 = 34.87$$

The depression score means grouped by substance produce exactly the same results:

```
[13]: print("Grouped means: \n", mydata.groupby(['substance'])['cesd'].mean())
```

```
Grouped means:
substance
alcohol    34.37
cocaine   29.42
heroin    34.87
Name: cesd, dtype: float64
```

Therefore, the slopes  $b_{1c} = -4.95$  and  $b_{1h} = 0.5$  give relative mean differences compared to the reference level of alcohol. The same problem is more naturally considered using the Analysis of Variance (ANOVA) in a separate chapter.

In addition, the linear model output has separate hypothesis tests for each slope.

For cocaine vs. alcohol:

H0:  $\beta_{1c} = 0$  - zero slope for cocaine

H1:  $\beta_{1c} \neq 0$  - non-zero slope

The t-test  $t = -3.64$  and p-value =  $0.0003 < 0.05$ , so H0 is rejected and there is a significant slope (difference) in average **cesd** scores for cocaine vs alcohol.

For heroin vs. alcohol:

H0:  $\beta_{1h} = 0$  -zero slope for heroine

H1:  $\beta_{1h} \neq 0$  - non-zero slope

The t-test  $t = 0.35$  and p-value =  $0.73 > 0.05$ , H0 is not rejected and there is not a significant slope (difference) in average **cesd** scores for heroin vs. alcohol.

---

## Bibliography

---

- [1] S. Guido A. Muller. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reily Media, 2016. The book is an introduction to more advanced concepts of Machine Learning.
- [2] Martha A. Allen. *Understanding Basic Statistics*. Cengage Learning, 7th edition, 2022. Includes examples using software such as Excel.
- [3] Bruce A. Craig. *A First Course in Statistics*. Pearson, 12th edition, 2017. Uses software tools like SPSS for data analysis.
- [4] D.J. Dennis. *Applied Univariate, Bivariate, and Multivariate Statistics Using Python: A Beginner's Guide to Advanced Data Analysis*. Wiley, 2021. The book is geared toward Psychology and Social Science students. uses Python.
- [5] David M. Diez, Christopher D. Barr, and Mine Çetinkaya Rundel. *OpenIntro Statistics*. OpenIntro, Inc., 4th edition, 2019. Available online.
- [6] Julian J. Faraway. *Linear Models with R*. Chapman & Hall/CRC, Boca Raton, 2005. Designed for students with a background in statistics, focusing on linear models using R.
- [7] Ronald A. Fisher. *Statistical Methods for the Social Sciences*. Pearson, 4th edition, 2018. Features applications and exercises using various software tools.
- [8] J. Taylor G. James. D. Witten. T. Hastie, R. Tibshirani. *An Introduction to Statistical Learning with Applications in Python*. Springer, 2023. Advanced textbook using Python, assumes familiarity with basic Statistics.
- [9] T. Haslwanter. *An Introduction to Statistics with Python*. Springer, 2022. The book assumes familiarity with Statistics and introduces a number of Python tools.
- [10] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R*. Springer, New York, 2013. Covers machine learning and statistical methods using R, suitable for students who have already taken an introductory statistics course.

- [11] G. Jay Kerns. *Introduction to Probability and Statistics Using R*. 2010. Open-source book that provides a comprehensive overview of probability and statistics using R, suitable for students with prior knowledge of statistics.
- [12] Robert L. Merriam and Paul D. Berger. *Introductory Statistics with R*. Wiley, 2nd edition, 2021. Focuses on statistical analysis using R.
- [13] David S. Moore and George P. McCabe. *Introduction to the Practice of Statistics*. W. H. Freeman, 9th edition, 2016.
- [14] Michael J. Sullivan. *Fundamentals of Statistics*. Pearson, 5th edition, 2018. Includes examples and exercises using statistical software.
- [15] Mario F. Triola. *Elementary Statistics*. Pearson, 13th edition, 2018. Uses software tools like Excel and TI84 for data analysis.
- [16] Maria A. Trujillo and Kevin C. Decker. *Introduction to Statistics: An Applications Approach*. McGraw-Hill, 4th edition, 2015. Incorporates software such as Excel and Minitab.
- [17] W.N. Venables and B.D. Ripley. *Modern Applied Statistics with S*. Springer, New York, 4th edition, 2002. Assumes prior knowledge of statistics and focuses on statistics using S-Plus and R.
- [18] Neil A. Weiss. *Introductory Statistics*. Pearson, 9th edition, 2016. Includes applications with software such as Excel and Minitab.
- [19] Hadley Wickham and Garrett Grolemund. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media, Sebastopol, 2016. A practical guide for data analysis using R, assuming a basic understanding of statistics.
- [20] Simon N. Wood. *Generalized Additive Models: An Introduction with R*. Chapman & Hall/CRC, Boca Raton, 2nd edition, 2017. Advanced text for those familiar with statistics, focusing on generalized additive models using R.

---

# Index

---

- 2 standard deviations rule, 83  
95% confidence interval, 105  
addition rule, 37  
alternative hypothesis, 113  
autocorrelation, 253  
barplot, 26  
Bayes Theorem, 59  
bell-shaped, 16, 67  
Benford's law, 40  
Bernoulli random variable, 86  
bias, 8  
bimodal, 17  
binomial distribution, 86  
binomial formula, 89  
bins, 14  
birthday-problem, 44  
boxplot, 22  
categorical variables, 2  
census, 8, 99  
Central Limit Theorem, 74  
Central Limit Theorem for means, 171  
Central Limit Theorem for Proportions, 103  
Chebyshev Theorem, 20  
chi-squared test of independence, 157  
coefficient of determination, 244, 263  
coefficient of variation, 21  
Cohen D, 185  
combinations, 88  
complement, 41  
conditional probability, 45, 48  
confidence interval, 104  
confidence interval for prediction, 263  
confounding variable, 7  
contingency table, 47  
contingency tables, 28  
continuity correction, 97  
continuous, 3  
continuous random variable, 65  
control group, 1  
correlation, 237  
correlation t-test, 242  
CPS85, 2  
critical  $z^*$ , 109  
data format, 2  
data frame, 2  
data ranks, 195  
degree of freedom, 18  
dependent variable, 250  
descriptive statistics, 9, 11  
discrete, 3  
discrete random variable, 65  
disjoint, 37  
double blinding, 9  
Durbin-Watson test, 253  
effect size, 206  
effect size of paired t-test, 220  
efficacy, 1  
event, 35  
expected counts, 143  
expected proportions, 143  
expected value, 63  
experiments, 7  
explained deviation, 262  
explanatory variable, 250  
exponential regression, 282  
false negative, 51  
false positive, 51  
finite sample space, 38

- first quartile, 22
- fluctuations, 1
- frequency, 14
- funnel pattern, 252
- general addition rule, 38
- goodness of fit, 143
- HELPrc, 2
- histogram, 14
- hypothesis testing, 112
- independent, 42
- independent variable, 250
- inferential statistics, 9, 99
- influence, 258
- interquartile range, 23
- joint probabilities, 47
- least squares regression line, 249
- left tail, 74
- left-skewed shape, 15
- levels, 4
- leverage, 258
- linear model, 248
- linear trend, 238
- Mann–Whitney test, 222
- margin of error, 109
- marginal probabilities, 47
- mean, 11
- median, 22
- MHEALTH, 3
- mode, 17
- multimodal, 17
- multiplication rule, 42
- mutually exclusive, 37
- negative predictive value, 53
- nominal, 4
- non-parametric test, 195
- non-response rate, 8
- normal distribution, 67
- null hypothesis, 113
- numpy, 11
- observational study, 7
- observed counts, 143
- one-sample means t-test, 184
- one-sample means test, 171
- one-sided means t-test, 190
- one-sided test, 119
- ordinal, 4
- outlier, 23
- overestimate, 250
- paired (dependent), 210
- pandas library, 2
- parameter, 8
- partial correlation, 246
- Pearson correlation coefficient, 238
- percentile, 24
- permutations, 87
- pie chart, 28
- placebo effects, 9
- point estimate, 13, 99
- polls, 1
- pooled proportion, 134
- pooled variance, 221
- population, 7
- population mean, 13
- population variance, 18
- positive predictive value, 52
- power, 207
- practical significance, 128
- predictor variable, 5
- prevalence, 55
- probability, 35
- prospective studies, 7
- quantile, 72
- quantitative variables, 2
- random processes, 35
- randomized block design, 9
- randomized experimental design, 8
- regression ANOVA, 261
- regression categorical predictor, 296
- residual, 249
- residual standard error, 263
- response variable, 5, 250
- retrospective studies, 7

- right tail, 75
- right-skewed shape, 15
- robust statistics, 25
- rule of thumb, 177
- sample, 7
- sample mean, 76
- sample proportion, 99
- sample size for mean, 204
- sample size mean, 204
- sample size proportion, 128
- sample space, 35
- sample standard deviation, 18
- sample variance, 18
- sampling distribution, 100, 173
- sampling error, 99
- scatterplot, 4, 237
- seaborn, 14
- sensitivity, 53
- shape, 15
- simple random sample, 8
- Spearman correlation, 245
- specificity, 53
- standard deviation, 18
- standard error, 100, 173
- standard normal distribution, 68
- statistic, 8
- statistically significant, 1
- statistics, 1
- symmetric, 16
- t-distribution, 179
- third quartile, 22
- time series, 253
- trade-off relationship, 109
- treatment group, 1
- tree diagrams, 57
- trimmed mean, 12
- true negative, 51
- true positive, 51
- two proportions test, 134
- two-sided, 113
- two-way contingency table, 157
- type 1 error, 126
- type 2 error, 127
- unbiased, 8
- underestimate, 250
- unexplained deviation, 262
- uniform distribution, 39, 66
- unimodal, 17
- upper whisker, 23
- value counts(), 26
- variability, 18
- volatility, 18
- voluntary response, 8
- Wilcoxon signed rank test, 195
- with replacement, 46
- without replacement, 46
- Z-score, 69