



PRATHYUSHA

ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

REGULATION 2021

II YEAR - III SEMESTER

CS3352 – FOUNDATIONS OF DATA SCIENCE

SYLLABUS

CS3352

FOUNDATIONS OF DATA SCIENCE

COURSE OBJECTIVES:

- To understand the data science fundamentals and process.
- To learn to describe the data for the data science process.
- To learn to describe the relationship between data.
- To utilize the Python libraries for Data Wrangling.
- To present and interpret data using visualization libraries in Python

UNIT I INTRODUCTION

Data Science: Benefits and uses – facets of data - Data Science Process: Overview – Defining research goals – Retrieving data – Data preparation - Exploratory Data analysis – build the model– presenting findings and building applications - Data Mining - Data Warehousing – Basic Statistical descriptions of Data

UNIT II DESCRIBING DATA

Types of Data - Types of Variables -Describing Data with Tables and Graphs –Describing Data with Averages - Describing Variability - Normal Distributions and Standard (z) Scores

UNIT III DESCRIBING RELATIONSHIPS

Correlation –Scatter plots –correlation coefficient for quantitative data –computational formula for correlation coefficient – Regression –regression line –least squares regression line – Standard error of estimate – interpretation of r^2 –multiple regression equations –regression towards the mean

UNIT IV PYTHON LIBRARIES FOR DATA WRANGLING

Basics of Numpy arrays –aggregations –computations on arrays –comparisons, masks, boolean logic – fancy indexing – structured arrays – Data manipulation with Pandas – data indexing and selection – operating on data – missing data – Hierarchical indexing – combining datasets – aggregation and grouping – pivot tables

UNIT V DATA VISUALIZATION

Importing Matplotlib – Line plots – Scatter plots – visualizing errors – density and contour plots – Histograms – legends – colors – subplots – text and annotation – customization – three dimensional plotting - Geographic Data with Basemap - Visualization with Seaborn.

CS3352 – Foundations of Data Science

UNIT I INTRODUCTION

Data Science: Benefits and uses – facets of data - Data Science Process: Overview – Defining research goals – Retrieving data – Data preparation - Exploratory Data analysis – build the model– presenting findings and building applications - Data Mining - Data Warehousing – Basic Statistical descriptions of Data

Big Data:

Big data is a blanket term for any collection of data sets so large or complex that it becomes difficult to process them using traditional data management techniques such as for example, the RDBMS.

I. Data Science:

- Data science involves using methods to analyze massive amounts of data and extract the knowledge it contains.
- The characteristics of big data are often referred to as the three Vs:
 - Volume—How much data is there?
 - Variety—How diverse are different types of data?
 - Velocity—At what speed is new data generated?
- Fourth V:
- Veracity: How accurate is the data?
- Data science is an evolutionary extension of statistics capable of dealing with the massive amounts of data produced today.
- Data scientist apart from a statistician are the ability to work with big data and experience in machine learning, computing, and algorithm building. Tools Hadoop, Pig, Spark, R, Python, and Java, among others.

II. Benefits and uses of data science and big data

- Data science and big data are used almost everywhere in both commercial and non-commercial settings.
- Commercial companies in almost every industry use data science and big data to gain insights into their customers, processes, staff, completion, and products.
- Many companies use data science to offer customers a better user experience.
 - Eg: Google AdSense, which collects data from internet users so relevant commercial messages can be matched to the person browsing the internet
 - MaxPoint - example of real-time personalized advertising.
- Human resource professionals:
 - people analytics and text mining to screen candidates,
 - monitor the mood of employees, and
 - study informal networks among coworkers
- Financial institutions use data science:
 - to predict stock markets, determine the risk of lending money, and
 - learn how to attract new clients for their services
- Governmental organizations:
 - internal data scientists to discover valuable information,
 - share their data with the public

- Eg: Data.gov is but one example; it's the home of the US Government's open data.
- organizations collected 5 billion data records from widespread applications such as Google Maps, Angry Birds, email, and text messages, among many other data sources.
- Nongovernmental organizations:
 - World Wildlife Fund (WWF), for instance, employs data scientists to increase the effectiveness of their fundraising efforts.
 - Eg: DataKind is one such data scientist group that devotes its time to the benefit of mankind.
- Universities:
 - Use data science in their research but also to enhance the study experience of their students.
 - massive open online courses (MOOC) produces a lot of data, which allows universities to study how this type of learning can complement traditional classes.
 - Eg: Coursera, Udacity, and edX

III. Facets of data:

The main categories of data are these:

- Structured
- Unstructured
- Natural language
- Machine-generated
- Graph-based
- Audio, video, and images
- Streaming

Structured data:

- Structured data is data that depends on a data model and resides in a fixed field
- within a record.
- Easy to store structured data in tables within databases or Excel files or Structured Query Language.

Indicator ID	Dimension List	Timeframe	Numeric Value	Missing Value Flag	Confidence Int
2	Total (Age-adjusted)	2008	74.6%		73.8%
3	Aged 18-44 years	2008	59.4%		58.0%
4	Aged 18-24 years	2008	37.4%		34.6%
5	Aged 25-44 years	2008	66.9%		65.5%
6	Aged 45-64 years	2008	88.6%		87.7%
7	Aged 45-54 years	2008	86.3%		85.1%
8	Aged 55-64 years	2008	91.5%		90.4%
9	Aged 65 years and over	2008	94.6%		93.8%
10	Aged 65-74 years	2008	93.6%		92.4%
11	Aged 75-84 years	2008	95.6%		94.4%
12	Aged 85 years and over	2008	96.0%		94.0%
13	Male (Age-adjusted)	2008	72.2%		71.1%
14	Female (Age-adjusted)	2008	76.8%		75.9%
15	White only (Age-adjusted)	2008	73.8%		72.9%
16	Black or African American only (Age-adjusted)	2008	77.0%		75.0%
17	American Indian or Alaska Native only (Age-adjusted)	2008	66.5%		57.1%
18	Asian only (Age-adjusted)	2008	80.5%		77.7%
19	Native Hawaiian or Other Pacific Islander only (Age-adjusted)	2008	DSU		
20	2 or more races (Age-adjusted)	2008	75.6%		69.6%

Figure 1.1 An Excel table is an example of structured data.

Unstructured data:

- Unstructured data is data that isn't easy to fit into a data model
- The content is context-specific or varying.
- Eg: E-mail
- Email contains structured elements such as the sender, title, and body text

- Eg: It's a challenge to find the number of people who have written an email complaint about a specific employee because so many ways exist to refer to a person.
- The thousands of different languages and dialects.

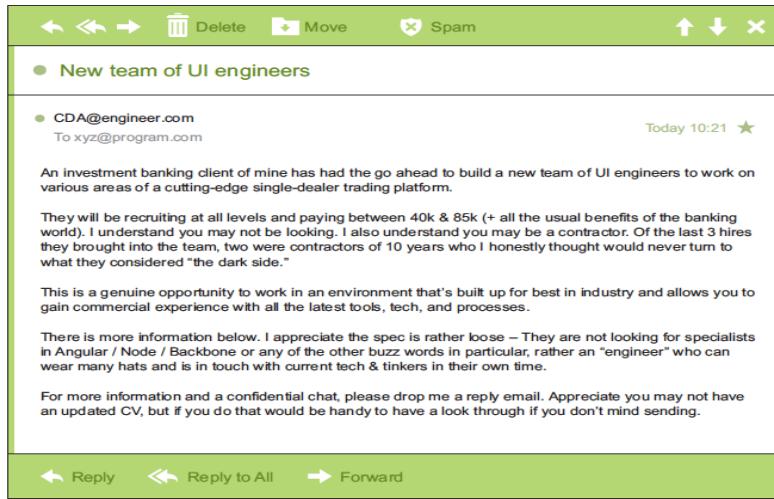


Figure 1.2 Email is simultaneously an example of unstructured data and natural language data.

Natural language:

- A human-written email is also a perfect example of natural language data.
- Natural language is a special type of unstructured data;
- It's challenging to process because it requires knowledge of specific data science techniques and linguistics.
- Topics in NLP: entity recognition, topic recognition, summarization, text completion, and sentiment analysis.
- Human language is ambiguous in nature.

Machine-generated data:

- Machine-generated data is information that's automatically created by a computer, process, application, or other machines without human intervention.
- Machine-generated data is becoming a major data resource.
- Eg: Wikibon has forecast that the market value of the industrial Internet will be approximately \$540 billion in 2020.
- International Data Corporation has estimated there will be 26 times more connected things than people in 2020.
- This network is commonly referred to as the internet of things.
- Examples of machine data are web server logs, call detail records, network event logs, and telemetry.

```

CSI:PERF:TXCOMMIT;313236
2014-11-28 11:36:13, Info
69), objectname [6]"(null)"
2014-11-28 11:36:13, Info
result 0x00000000, handle 0x4e54
2014-11-28 11:36:13, Info
Beginning NT transaction commit...
2014-11-28 11:36:13, Info
trace:
CSI:PERF:TXCOMMIT;273983
2014-11-28 11:36:13, Info
70), objectname [6]"(null)"
2014-11-28 11:36:13, Info
result 0x00000000, handle 0x4e5c
2014-11-28 11:36:13, Info
Beginning NT transaction commit...
2014-11-28 11:36:14, Info
trace:
CSI:PERF:TXCOMMIT;386259
2014-11-28 11:36:14, Info
71), objectname [6]"(null)"
2014-11-28 11:36:14, Info
result 0x00000000, handle 0x4e5c
2014-11-28 11:36:14, Info
Beginning NT transaction commit...
2014-11-28 11:36:14, Info
trace:
CSI:PERF:TXCOMMIT;375581

```

Figure 1.3 Example of machine-generated data

Graph-based or network data:

- “Graph” in this case points to mathematical graph theory. In graph theory, a graph is a mathematical structure to model pair-wise relationships between objects.
- Graph or network data is, in short, data that focuses on the relationship or adjacency of objects.
- The graph structures use nodes, edges, and properties to represent and store graphical data.
- Graph-based data is a natural way to represent social networks, and its structure allows you to calculate the shortest path between two people.
- Graph-based data can be found on many social media websites.
- Eg: LinkedIn, Twitter, movie interests on Netflix
- Graph databases are used to store graph-based data and are queried with specialized query languages such as SPARQL.

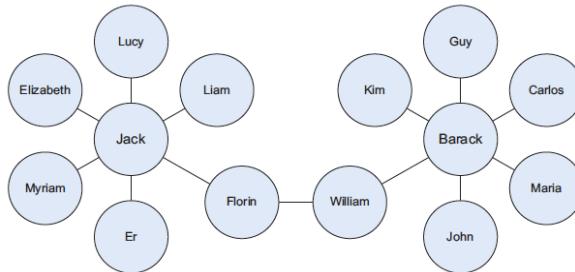


Figure 1.4 Friends in a social network are an example of graph-based data.

Audio, image, and video:

- Audio, image, and video are data types that pose specific challenges to a data scientist.
- Recognizing objects in pictures, turn out to be challenging for computers.
- Major League Baseball Advanced Media - video capture to approximately 7 TB per game for the purpose of live, in-game analytics.
- High-speed cameras at stadiums will capture ball and athlete movements to calculate in real time.
- DeepMind succeeded at creating an algorithm that's capable of learning how to play video games.
- This algorithm takes the video screen as input and learns to interpret everything via a complex process of deep learning.
- Google – Artificial Intelligence Development plans

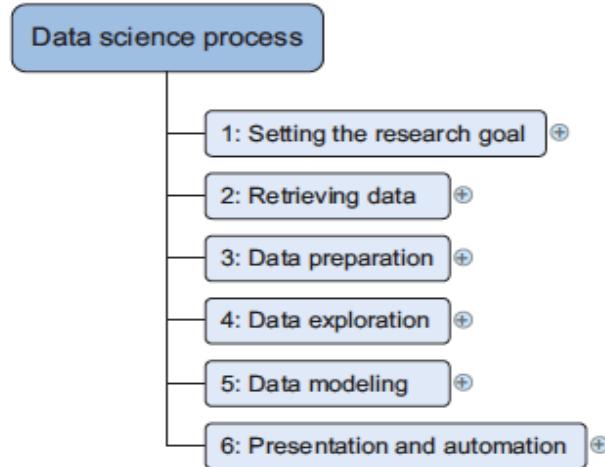
Streaming data:

- The data flows into the system when an event happens instead of being loaded into a data store in a batch.
- Examples are the “What’s trending” on Twitter, live sporting or music events, and the stock market.

The data science process:

- The data science process typically consists of six steps:
 - Setting the research goal
 - Retrieving data
 - Data preparation

- Data exploration
- Data modeling or model building
- Presentation and automation



The data science process

IV. Overview of the data science process:

- A structured data science approach helps you maximize your chances of success in a data science project at the lowest cost.
- The first step of this process is setting a research goal.
- The main purpose here is to make sure all the stakeholders understand the what, how, and why of the project.
- Draw the result in a project charter.

Step 1: Defining research goals and creating a project charter

- A project starts by understanding your project's what, why, and how.
- The outcome should be a clear research goal, a good understanding of the context, well-defined deliverables, and a plan of action with a timetable.
- The information is then best placed in a project charter.

Spend time understanding the goals and context of your research:

- An essential outcome is the research goal that states the purpose of your assignment
- in a clear and focused manner.
- Understanding the business goals and context is critical for project success.
- To asking questions and devising examples:
 - for business expectations,
 - how your research is going to change the business, and
 - understand how they'll use your results

Create a project charter:

- The formal agreement on the deliverables.
- All this information is best collected in a project charter.
- A project charter requires teamwork, and your input covers at least the following:
 - A clear research goal
 - The project mission and context

- How you're going to perform your analysis
- What resources you expect to use
- Proof that it's an achievable project, or proof of concepts
- Deliverables and a measure of success
- A timeline

V. Step 2: Retrieving data

- The next step in data science is to retrieve the required data.
- Sometimes we need to go into the field and design a data collection process.
- Many companies will have already collected and stored the data.
- That also can be bought from third parties.
- look outside your organization for data - high-quality data freely available for
- public and commercial use.
- Data can be stored in many forms, ranging from simple text files to tables in a database.

Start with data stored within the company

- To assess the relevance and quality of the data that's readily available within the company.
- Company data - data can be stored in official data repositories such as databases, data marts, data warehouses, and data lakes maintained by a team of IT professionals.
- Data mart: A data mart is a subset of the data warehouse and will be serving a specific business unit.
- Data lakes: Data lakes contain data in its natural or raw format.
- Challenge: As companies grow, their data becomes scattered around many places.
- Knowledge of the data may be dispersed as people change positions and leave the company.
- Chinese Walls: These policies translate into physical and digital barriers called Chinese walls. These "walls" are mandatory and well-regulated for customer data.

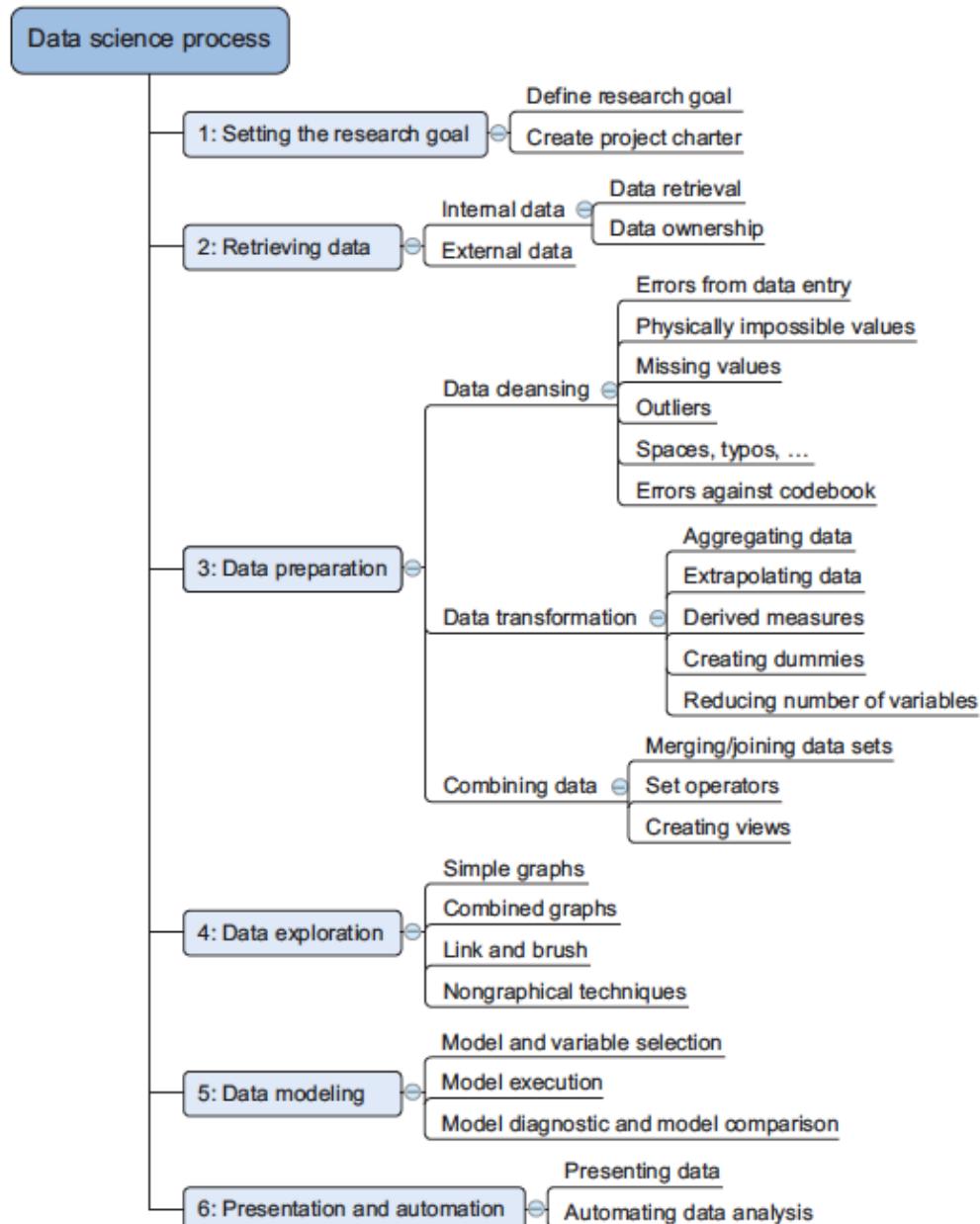


Figure 2.1 The six steps of the data science process

Don't be afraid to shop around:

- Many companies specialize in collecting valuable information.
- Nielsen and GFK - retail industry.
- Data as Service - Twitter, LinkedIn, and Facebook.

Do data quality checks now to prevent problems later:

- Data Correction and cleansing.
- Data retrieval - to see if the data is equal to the data in the source document and if you have the right data types.
- Discover outliers in the exploratory phase, they can point to a data entry error.

Table 2.1 A list of open-data providers that should get you started

Open data site	Description
Data.gov	The home of the US Government's open data
https://open-data.europa.eu/	The home of the European Commission's open data
Freebase.org	An open database that retrieves its information from sites like Wikipedia, MusicBrains, and the SEC archive
Data.worldbank.org	Open data initiative from the World Bank
Aiddata.org	Open data for international development
Open.fda.gov	Open data from the US Food and Drug Administration

VI. Step 3: Cleansing, integrating, and transforming data

The model needs the data in a specific format, so data transformation will be the step. It's a good habit to correct data errors as early on in the process as possible.

Cleansing data:

Data cleansing is a subprocess of the data science process.

It focuses on removing errors in the data.

Then the data becomes a true and consistent representation of the processes.

Types of errors:

Interpretation error - a person's age is greater than 300 years

Inconsistencies - class of errors is putting "Female" in one table and "F" in another when they represent the same thing.

Table 2.2 An overview of common errors

General solution	
Try to fix the problem early in the data acquisition chain or else fix it in the program.	
Error description	Possible solution
<i>Errors pointing to false values within one data set</i>	
Mistakes during data entry	Manual overrules
Redundant white space	Use string functions
Impossible values	Manual overrules
Missing values	Remove observation or value
Outliers	Validate and, if erroneous, treat as missing value (remove or insert)
<i>Errors pointing to inconsistencies between data sets</i>	
Deviations from a code book	Match on keys or else use manual overrules
Different units of measurement	Recalculate
Different levels of aggregation	Bring to same level of measurement by aggregation or extrapolation

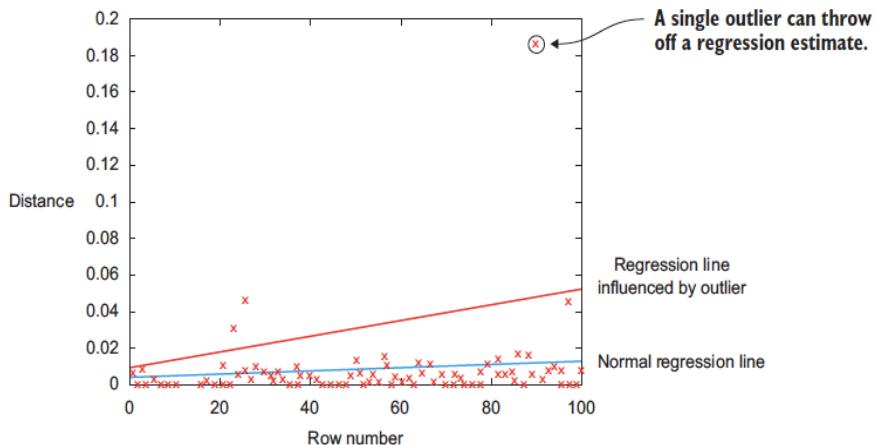


Figure 2.5 The encircled point influences the model heavily and is worth investigating because it can point to a region where you don't have enough data or might indicate an error in the data, but it also can be a valid data point.

DATA ENTRY ERRORS:

- Data collection and data entry are error-prone processes.
- Errors can arise from human sloppiness, whereas others are due to machine or hardware failure.
- Eg: transmission errors

Table 2.3 Detecting outliers on simple variables with a frequency table

Value	Count
Good	1598647
Bad	1354468
Godo	15
Bade	1

```
if x == "God0" :
    x = "Good"
if x == "Bade" :
    x = "Bad"
```

REDUNDANT WHITESPACE:

- Whitespaces tend to be hard to detect but cause errors like other redundant characters.
- Eg: a mismatch of keys such as “FR” – “FR”
- Fixing redundant whitespaces - Python can use the strip() function to remove leading and trailing spaces.

FIXING CAPITAL LETTER MISMATCHES:

- Capital letter mismatches - distinction between “Brazil” and “brazil”
- strings in lowercase, such as .lower() in Python. “Brazil”.lower() == “brazil”.lower() should result in true.

IMPOSSIBLE VALUES AND SANITY CHECKS:

- Sanity checks are another valuable type of data check.
- Check the value against physically or theoretically impossible values : such as people taller than 3 meters or someone with an age of 299 years.

```
check = 0 <= age <= 120
```

OUTLIERS

- An outlier is an observation that seems to be distant from other observations.

- The normal distribution, or Gaussian distribution, is the most common distribution in natural sciences.

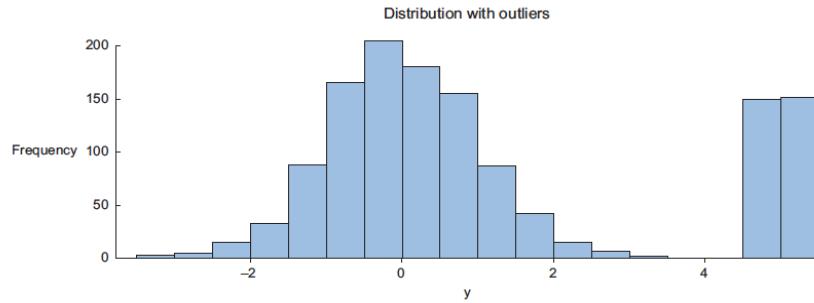


Figure 2.6 Distribution plots are helpful in detecting outliers and helping you understand the variable.

The high values in the bottom graph can point to outliers when assuming a normal distribution.

DEALING WITH MISSING VALUES:

Table 2.4 An overview of techniques to handle missing data

Technique	Advantage	Disadvantage
Omit the values	Easy to perform	You lose the information from an observation
Set value to null	Easy to perform	Not every modeling technique and/or implementation can handle null values
Impute a static value such as 0 or the mean	Easy to perform You don't lose information from the other variables in the observation	Can lead to false estimations from a model
Impute a value from an estimated or theoretical distribution	Does not disturb the model as much	Harder to execute You make data assumptions
Modeling the value (nondependent)	Does not disturb the model too much	Can lead to too much confidence in the model Can artificially raise dependence among the variables Harder to execute You make data assumptions

DEVIATIONS FROM A CODE BOOK:

- Detecting errors in larger data sets against a code book or against standardized values
- can be done with the help of set operations.
- A code book is a description of your data form of metadata.

DIFFERENT UNITS OF MEASUREMENT

- When integrating two data sets, we have to pay attention to their respective units of measurement.
- Eg: Data sets can contain prices per gallon and others can contain prices per liter.

DIFFERENT LEVELS OF AGGREGATION

- Having different levels of aggregation is similar to having different types of measurement.
- Eg: A data set containing data per week versus one containing data per work week.

Correct errors as early as possible:

- A good practice is to mediate data errors as early as possible in the data collection chain and to fix as little as possible.
- The data collection process is error-prone, and in a big organization, it involves many steps and teams.
- Data should be cleansed when acquired for many reasons:
- Not everyone spots the data anomalies
- If errors are not corrected early on in the process, the cleansing will have to be done.
- Data errors may point to a business process that isn't working as designed.
- Data errors may point to defective equipment, etc.,
- Data errors can point to bugs in software or in the integration of software.
- Data manipulation doesn't end with correcting mistakes; still need to combine your incoming data.

Combining data from different data sources:

- Data varies in size, type, and structure, ranging from databases and Excel files to text documents.

THE DIFFERENT WAYS OF COMBINING DATA:

- Two operations to combine information from different data.
- joining: enriching an observation from one table with information from another table.
- The second operation is appending or stacking: adding the observations of one table to those of another table.

JOINING TABLES

- Joining tables allows you to combine the information of one observation found in one table with the information that you find in another table

CHAPTER 2 *The data science process*

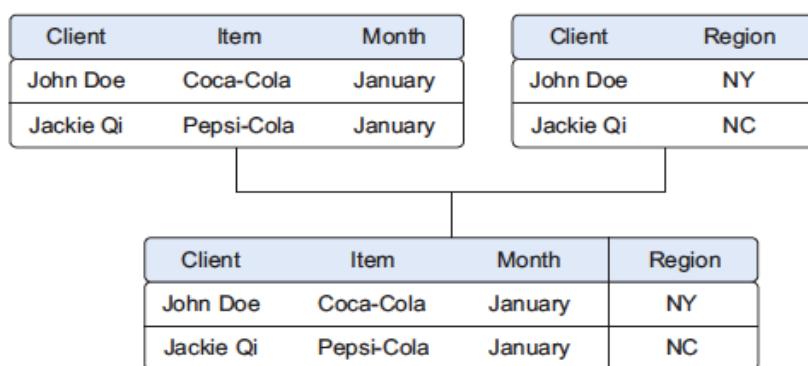


Figure 2.7 Joining two tables on the Item and Region keys

- When these keys also uniquely define the records in the table they are called primary keys.

APPENDING TABLES

- Appending or stacking tables is effectively adding observations from one table to another table.

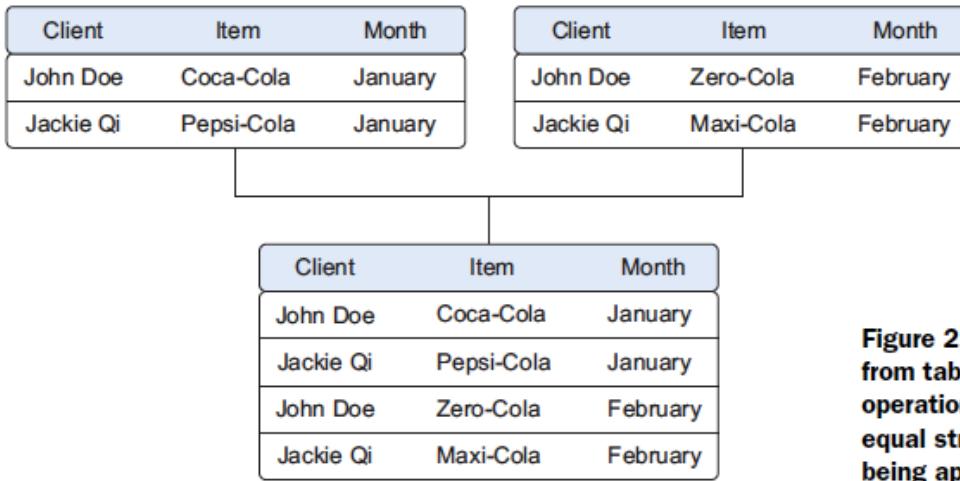


Figure 2.8 Appending data from tables is a common operation but requires an equal structure in the tables being appended.

USING VIEWS TO SIMULATE DATA JOINS AND APPENDS

- To avoid duplication of data, we can virtually combine data with views.
- How the sales data from the different months is combined virtually into a yearly sales table instead of duplicating the data?
- A table join is only performed once, the join that creates the view is recreated every time it's queried.

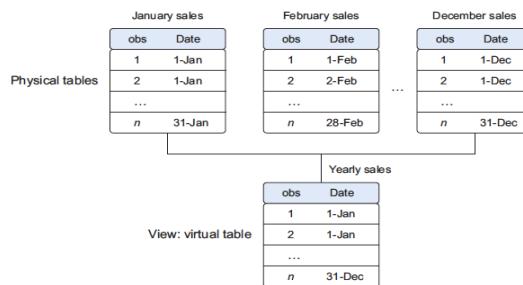


Figure 2.9 A view helps you combine data without replication.

ENRICHING AGGREGATED MEASURES

- Data enrichment can also be done by adding calculated information to the table.
- Eg: such as the total number of sales or what percentage of total stock has been sold in a certain region.

Product class	Product	Sales in \$	Sales t-1	Growth	Sales by	Rank sales
			in \$		product class	
A	B	X	Y	(X-Y) / Y	AX	NX
Sport	Sport 1	95	98	-3.06%	215	2
Sport	Sport 2	120	132	-9.09%	215	1
Shoes	Shoes 1	10	6	66.67%	10	3

Figure 2.10 Growth, sales by product class, and rank sales are examples of derived and aggregate measures.

TRANSFORMING DATA

- Relationships between an input variable and an output variable aren't always linear.
- Take, for instance, a relationship of the form $y = ae^{bx}$.
- Taking the log of the independent variables simplifies the estimation problem.

x	1	2	3	4	5	6	7	8	9	10
log(x)	0.00	0.43	0.68	0.86	1.00	1.11	1.21	1.29	1.37	1.43
y	0.00	0.44	0.69	0.87	1.02	1.11	1.24	1.32	1.38	1.46

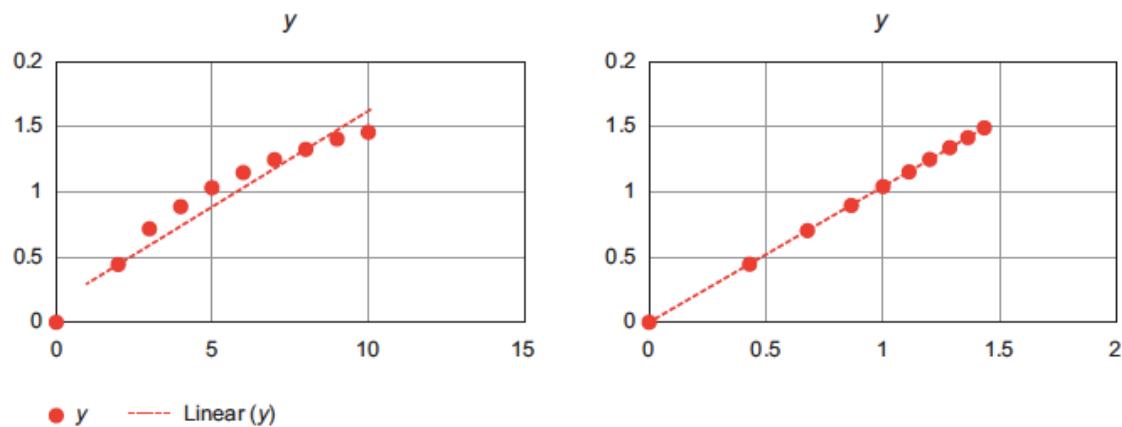


Figure 2.11 Transforming x to log x makes the relationship between x and y linear (right), compared with the non-log x (left).

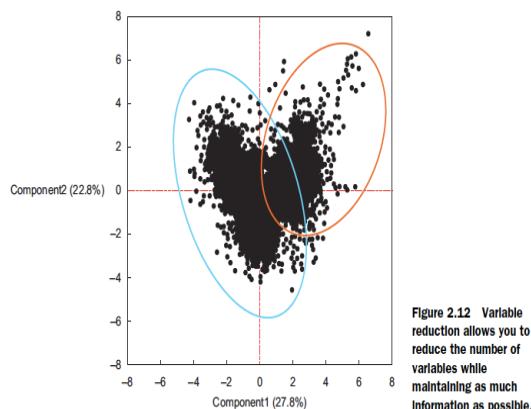
REDUCING THE NUMBER OF VARIABLES

- We have too many variables and need to reduce the number because they don't add new information to the model.
- Having too many variables in your model makes the model difficult to handle, and certain techniques don't perform well when you overload them with too many input variables.

Euclidean distance

Euclidean distance or “ordinary” distance is an extension to one of the first things anyone learns in mathematics about triangles (trigonometry): Pythagoras’s leg theorem. If you know the length of the two sides next to the 90° angle of a right-angled triangle you can easily derive the length of the remaining side (hypotenuse). The formula for this is $\text{hypotenuse} = \sqrt{(\text{side1} + \text{side2})^2}$. The Euclidean distance between two points in a two-dimensional plane is calculated using a similar formula: $\text{distance} = \sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2)}$. If you want to expand this distance calculation to more dimensions, add the coordinates of the point within those higher dimensions to the formula. For three dimensions we get $\text{distance} = \sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2)}$.

- Data scientists use special methods to reduce the number of variables but retain the maximum amount of data.
- Reducing the number of variables makes it easier to understand the key values.
- These variables, called “component1” and “component2,” are both combinations of the original variables.
- They’re the principal components of the underlying data structure.



TURNING VARIABLES INTO DUMMIES

- Variables can be turned into dummy variables.
- Dummy variables can only take two values: true(1) or false(0).
- They’re used to indicate the absence of a categorical effect that may explain the observation.
- An example is turning one column named Weekdays into the columns Monday through Sunday.
- We use an indicator to show if the observation was on a Monday; you put 1 on Monday and 0 elsewhere.

The diagram illustrates the process of creating dummy variables. It starts with a table of raw data:

Customer	Year	Gender	Sales
1	2015	F	10
2	2015	M	8
1	2016	F	11
3	2016	M	12
4	2017	F	14
3	2017	M	13

Below this, a decision tree splits the 'Gender' column into two binary variables: 'Male' (labeled 'M') and 'Female' (labeled 'F'). This leads to a second table of transformed data:

Customer	Year	Sales	Male	Female
1	2015	10	0	1
1	2016	11	0	1
2	2015	8	1	0
3	2016	12	1	0
3	2017	13	1	0
4	2017	14	0	1

Figure 2.13 Turning variables into dummies is a data transformation that breaks a variable that has multiple classes into multiple variables, each having only two possible values: 0 or 1.

VII. Step 4: Exploratory data analysis

Information becomes much easier to grasp when shown in a picture.
The graphical techniques to gain an understanding of your data and the interactions between variables.

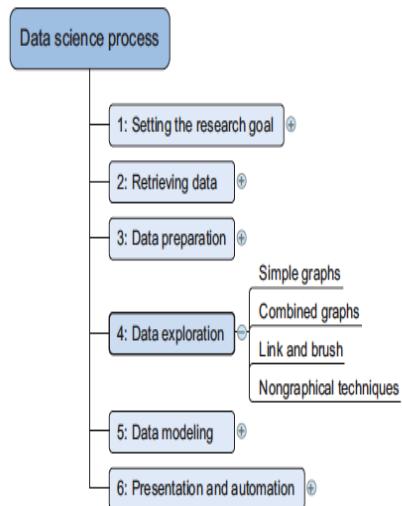


Figure 2.14 Step 4:
Data exploration

visualization techniques : simple line graphs or histograms

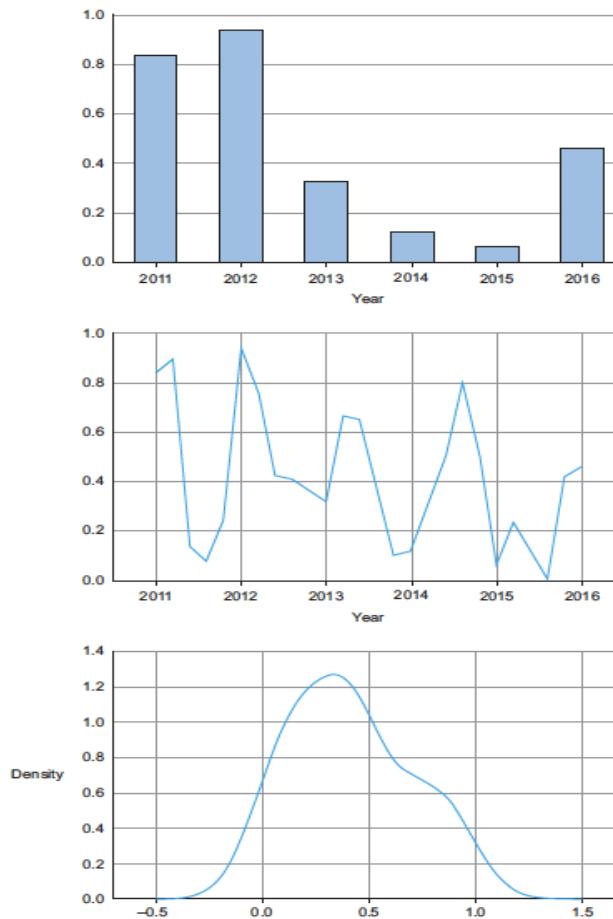


Figure 2.15 From top to bottom, a bar chart, a line plot, and a distribution are some of the graphs used in exploratory analysis.

Brushing and Linking:

- With brushing and linking we combine and link different graphs and tables or views so changes in one graph are automatically transferred to the other graphs.

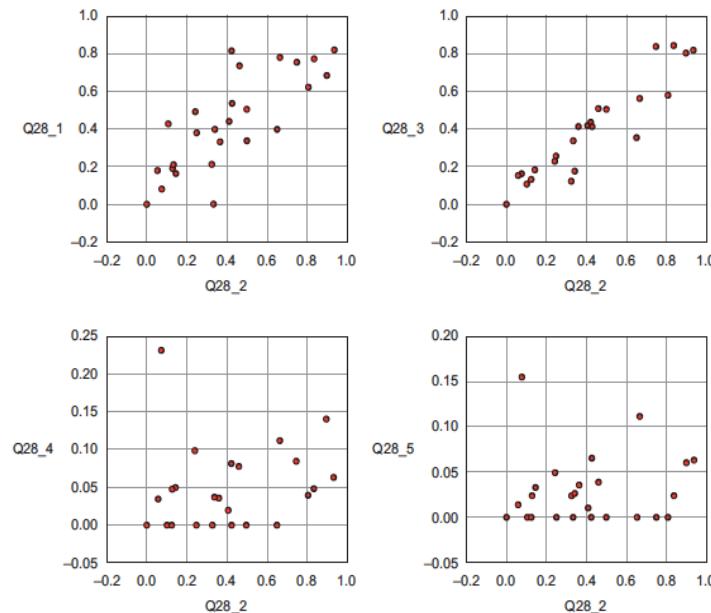


Figure 2.16 Drawing multiple plots together can help you understand the structure of your data over multiple variables.

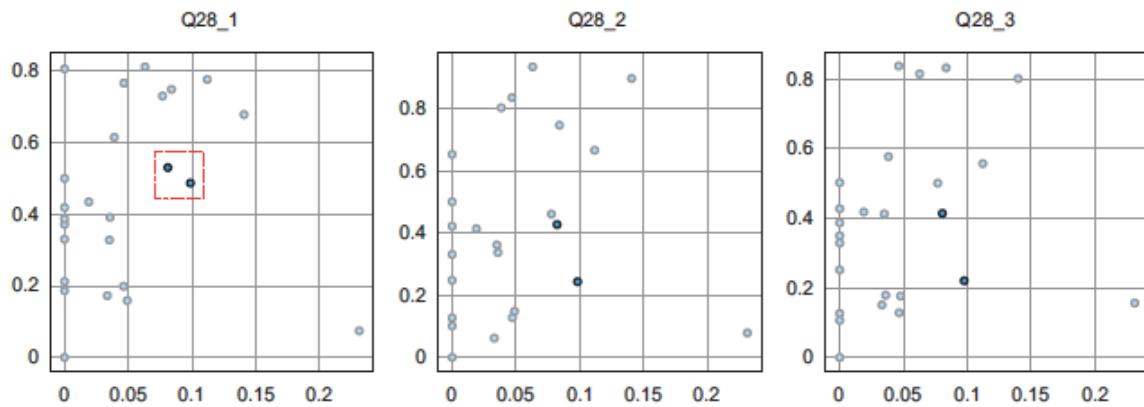
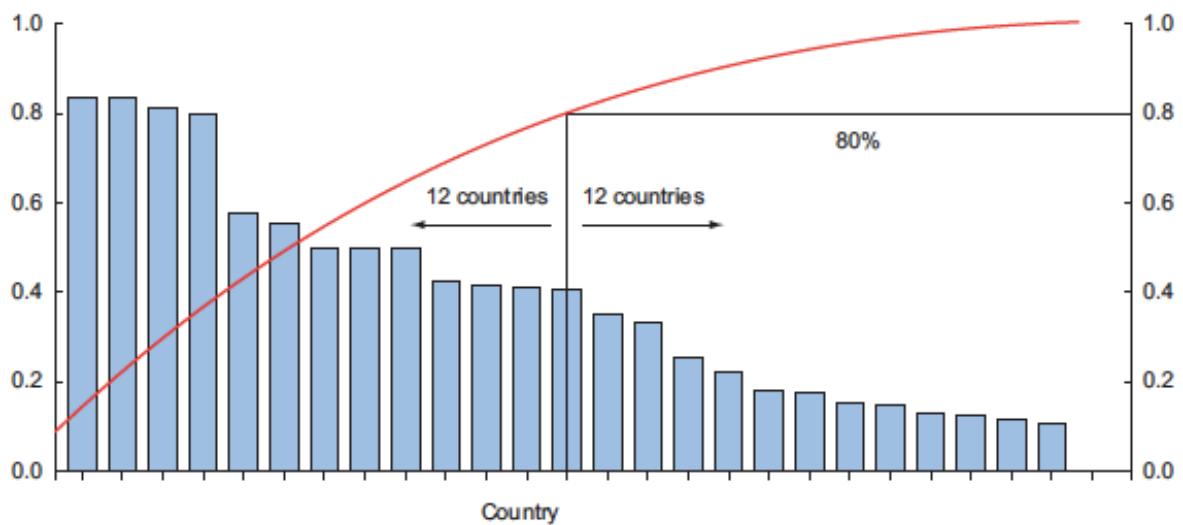


Figure 2.18 Link and brush allows you to select observations in one plot and highlight the same observations in the other plots.

Pareto Diagram:

- A Pareto diagram is a combination of the values and a cumulative distribution.
- It's easy to see from this diagram that the first 50% of the countries contain slightly less than 80% of the total amount.
- If this graph represented customer buying power and we sell expensive products, we probably don't need to spend our marketing budget in every country; we could start with the first 50%.



- In a histogram a variable is cut into discrete categories and the number of occurrences in each category are summed up and shown in the graph.
- The boxplot, doesn't show how many observations are present but does offer an impression of the distribution within categories.
- It can show the maximum, minimum, median, and other characterizing measures at the same time.

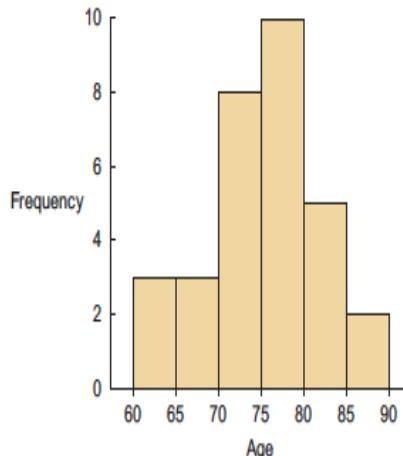


Figure 2.19 Example histogram:
the number of people in the age-
groups of 5-year intervals

Tabulation, clustering, and other modeling techniques can also be a part of exploratory analysis.

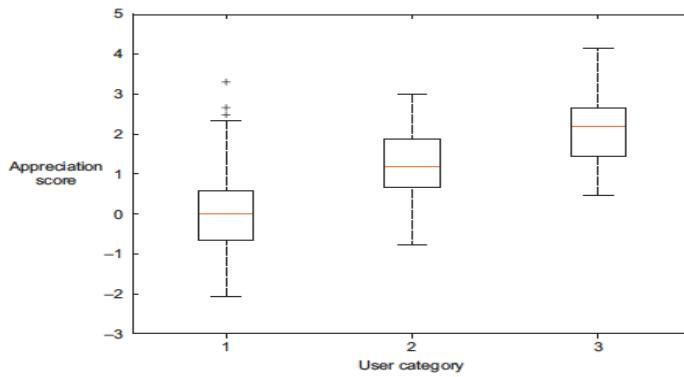


Figure 2.20 Example boxplot: each user category has a distribution of the appreciation each has for a certain picture on a photography website.

VIII. Step 5: Build the models

- With clean data in place and a good understanding of the content, we're ready to build models with the goal of making better predictions, classifying objects, or gaining an understanding of the system that we're modeling.
- The techniques we'll use now are borrowed from the field of machine learning, data mining, and/or statistics.
- Building a model is an iterative process.
- The way we build our model depends on whether we go with classic statistics or the recent machine learning
- and the type of technique we want to use.
- Either way, most models consist of the following main steps:
 - 1 Selection of a modeling technique and variables to enter in the model
 - 2 Execution of the model
 - 3 Diagnosis and model comparison

Model and variable selection

We need to select the variables you want to include in your model and a modeling technique.

We'll need to consider model performance and whether our project meets all the requirements to use your model, as well as other factors:

- Must the model be moved to a production environment and, if so, would it be easy to implement?
- How difficult is the maintenance on the model: how long will it remain relevant if left untouched?
- Does the model need to be easy to explain?

Model execution:

- The most programming languages, such as Python, already have libraries such as StatsModels or Scikit-learn.
- These packages use several of the most popular techniques.
- Coding a model is a nontrivial task in most cases, so having these libraries available can speed up the process.

Listing 2.1 Executing a linear prediction model on semi-random data

```
import statsmodels.api as sm  
import numpy as np  
predictors = np.random.random(1000).reshape(500,2)  
target = predictors.dot(np.array([0.4, 0.6])) + np.random.random(500)  
lmRegModel = sm.OLS(target,predictors)  
result = lmRegModel.fit()  
result.summary()
```

Imports required Python modules.

Creates random data for predictors (x-values) and semi-random data for the target (y-values) of the model. We use predictors as input to create the target so we infer a correlation here.

Fits linear regression on data.

Shows model fit statistics.

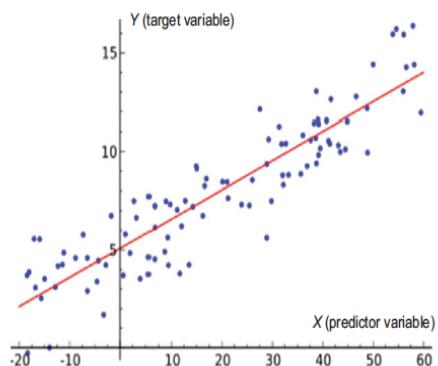


Figure 2.22 Linear regression tries to fit a line while minimizing the distance to each point

Model fit—For this the R-squared or adjusted R-squared is used.

- This measure is an indication of the amount of variation in the data that gets captured by the model.
- The difference between the adjusted R-squared and the R-squared is minimal here because the adjusted one is the normal one + a penalty for model complexity.
- A model gets complex when many variables or features are introduced.

Predictor variables have a coefficient—For a linear model this is easy to interpret.

- In our example if you add "1" to x1, it will change y by "0.7658".
- It's easy to see how finding a good predictor can be your route.

In [45]: `metrics.confusion_matrix(target,prediction)`

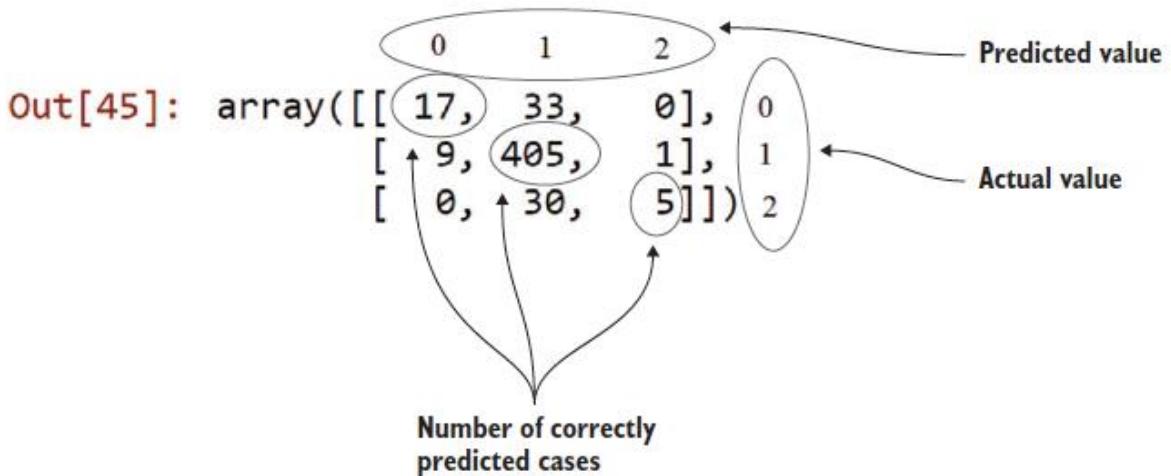


Figure 2.25 Confusion matrix: It shows how many cases were correctly classified and incorrectly classified by comparing the prediction with the real values. Remark: the classes (0,1,2) were added in the figure for clarification.

Model diagnostics and model comparison

Working with a holdout sample helps you pick the best-performing model.

A holdout sample is a part of the data you leave out of the model building so it can be used to evaluate the model afterward. The principle here is simple: the model should work on unseen data.

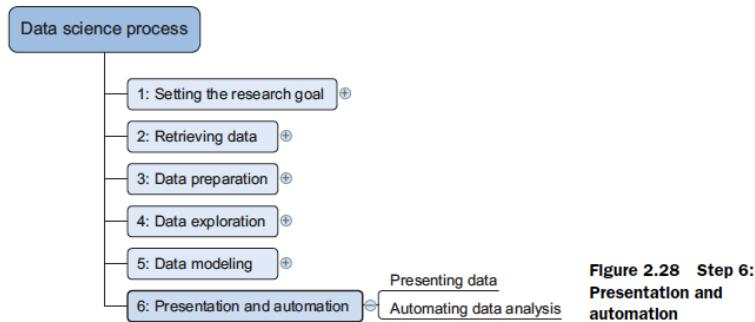
Mean square error is a simple measure: check for every prediction how far it was from the truth, square this error, and add up the error of every prediction.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2$$

Figure 2.26 Formula for mean square error

	<i>n</i>	Size	Price	Predicted model 1	Predicted model 2	Error model 1	Error model 2
80% train	1	10	3				
	2	15	5				
	3	18	6				
	4	14	5				
					
	800	9	3				
	801	12	4	12	10	0	2
	802	13	4	12	10	1	3
	...						
	999	21	7	21	10	0	11
20% test	1000	10	4	12	10	-2	0
				Total	5861	110225	

Figure 2.27 A holdout sample helps you compare models and ensures that you can generalize results to data that the model has not yet seen.



- Some work need to repeat it over and over again because they value the predictions of our models or the insights that you produced.
- For this reason, we need to automate your models.
- This doesn't always mean that we have to redo all of your analysis all the time.
- Sometimes it's sufficient that we implement only the model scoring; other times we might build an application
- that automatically updates reports, Excel spreadsheets, or PowerPoint presentations.

X. Data Mining:

- Data mining turns a large collection of data into knowledge.
- A search engine (e.g., Google) receives hundreds of millions of queries every day.
- Each query can be viewed as a transaction where the user describes her or his information need.
- For example, Google's Flu Trends uses specific search terms as indicators of flu activity.
- It found a close relationship between the number of people who search for flu-related information and the number of people who actually have flu symptoms.

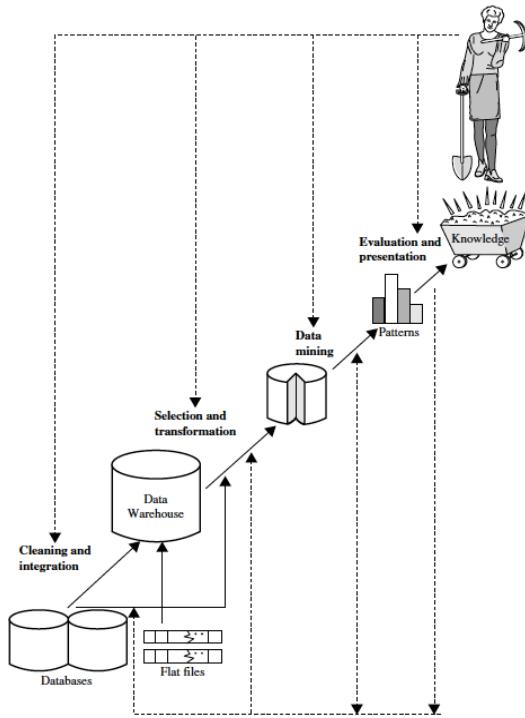


Figure 1.4 Data mining as a step in the process of knowledge discovery.

- In summary, the abundance of data, coupled with the need for powerful data analysis tools, has been described as a data rich but information-poor situation (Figure 1.2).
- The fast-growing, tremendous amount of data, collected and stored in large and numerous data repositories, has far exceeded our human ability for comprehension without powerful tools.
- As a result, data collected in large data repositories become “data tombs”—data archives that are seldom visited.
- Unfortunately, however, the manual knowledge input procedure is prone to biases and errors and is extremely costly and time consuming.
- The widening gap between data and information calls for the systematic development of data mining tools that can turn data tombs into “golden nuggets” of knowledge.
- other terms have a similar meaning to data mining—for example, knowledge mining from data, knowledge extraction, data/pattern analysis, data archaeology, and data dredging.

Many people treat data mining as a synonym for another popularly used term, knowledge discovery from data, or KDD, while others view data mining as merely an essential step in the process of knowledge discovery. The knowledge discovery process is shown in Figure 1.4 as an iterative sequence of the following steps:

1. Data cleaning (to remove noise and inconsistent data)
2. Data integration (where multiple data sources may be combined)
3. Data selection (where data relevant to the analysis task are retrieved from the database)
4. Data transformation (where data are transformed and consolidated into forms appropriate for mining by performing summary or aggregation operations)
5. Data mining (an essential process where intelligent methods are applied to extract data patterns)
6. Pattern evaluation (to identify the truly interesting patterns representing knowledge based on interestingness measures)

7. Knowledge presentation (where visualization and knowledge representation techniques are used to present mined knowledge to users)

XI. DataWarehouses

- A data warehouse is a repository of information collected from multiple sources, stored under a unified schema, and usually residing at a single site.
- Data warehouses are constructed via a process of data cleaning, data integration, data transformation, data loading, and periodic data refreshing.

Eg: All Electronics

- To facilitate decision making, the data in a data warehouse are organized around major subjects (e.g., customer, item, supplier, and activity).
- The data are stored to provide information from a historical perspective, such as in the past 6 to 12 months, and are typically summarized.
- For example, rather than storing the details of each sales transaction, the data warehouse may store a summary of the transactions per item type for each store or, summarized to a higher level, for each sales region.
- A data warehouse is usually modeled by a multidimensional data structure, called a data cube, in which each dimension corresponds to an attribute or a set of attributes in the schema, and each cell stores the value of some aggregate measure such as count.

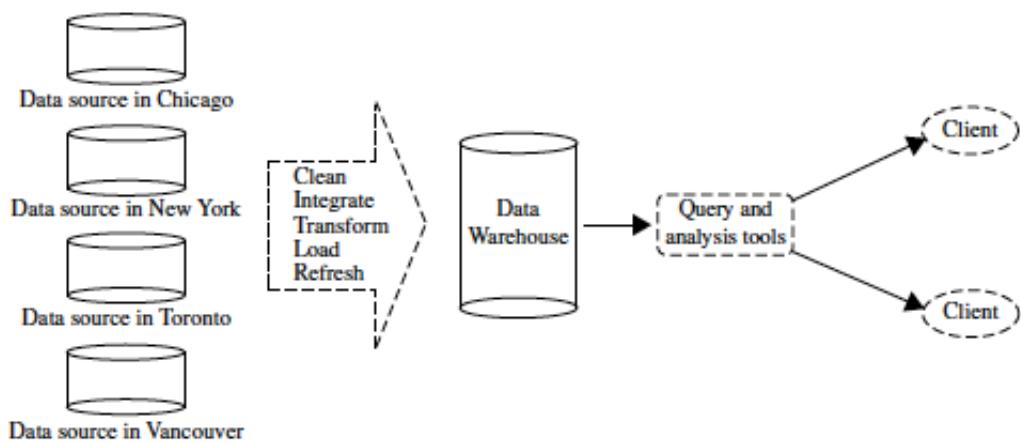


Figure 1.6 Typical framework of a data warehouse for *AllElectronics*.

- By providing multidimensional data views and the precomputation of summarized data, data warehouse systems can provide inherent support for OLAP.
- Online analytical processing operations make use of background knowledge regarding the domain of the data being studied to allow the presentation of data at different levels of abstraction.
- Such operations accommodate different user viewpoints.
- Examples of OLAP operations include drill-down and roll-up, which allow the user to view the data at differing degrees of summarization, as illustrated.

<i>trans_ID</i>	<i>list_of_item_IDs</i>
T100	I1, I3, I8, I16
T200	I2, I8
...	...

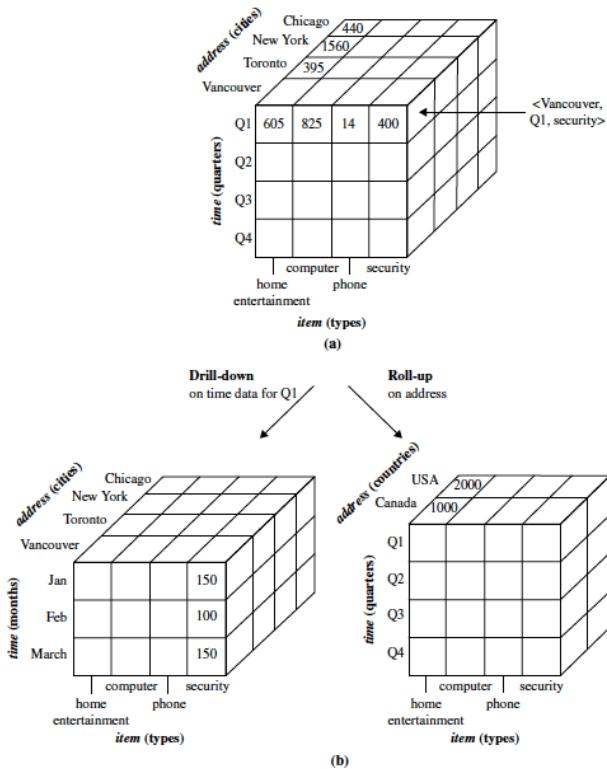


Figure 1.7 A multidimensional data cube, commonly used for data warehousing, (a) showing summarized data for *AllElectronics* and (b) showing summarized data resulting from drill-down and roll-up operations on the cube in (a). For improved readability, only some of the cube cell values are shown.

XII. Basic Statistical Descriptions of Data: Measuring the Central Tendency: Mean, Median, and Mode

- Suppose that we have some attribute X, like salary, which has been recorded for a set of objects.
- Let x_1, x_2, \dots, x_N be the set of N observed values or observations for X.
- Here, these values may also be referred to as the data set (for X).
- Measures of central tendency include the mean, median, mode, and midrange.
- The most common and effective numeric measure of the “center” of a set of data is the (arithmetic)mean.
- Let x_1, x_2, \dots, x_N be a set of N values or observations, such as for some numeric attribute X, like salary.
- The mean of this set of values is

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

Example 2.6 Mean. Suppose we have the following values for *salary* (in thousands of dollars), shown in increasing order: 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110. Using Eq. (2.1), we have

$$\bar{x} = \frac{30 + 36 + 47 + 50 + 52 + 52 + 56 + 60 + 63 + 70 + 70 + 110}{12}$$

$$= \frac{696}{12} = 58.$$

Thus, the mean salary is \$58,000. ■

- Sometimes, each value x_i in a set may be associated with a weight w_i for $i \in 1, \dots, N$.
- The weights reflect the significance, importance, or occurrence frequency attached to their respective values. In this case, we can compute. This is called the weighted arithmetic mean or the weighted average.

$$\bar{x} = \frac{\sum_{i=1}^N w_i x_i}{\sum_{i=1}^N w_i} = \frac{w_1 x_1 + w_2 x_2 + \dots + w_N x_N}{w_1 + w_2 + \dots + w_N}.$$

- To offset the effect caused by a small number of extreme values, we can instead use the trimmed mean, which is the mean obtained after chopping off values at the high and low extremes.
- For example, we can sort the values observed for salary and remove the top and bottom 2% before computing the mean.
- We should avoid trimming too large a portion (such as 20%) at both ends, as this can result in the loss of valuable information.
- For skewed data, a better measure of the center of data is the median, which is the middle value in a set of ordered data values.
- It is the value that separates the higher half of a data set from the lower half.

Median:

- The median generally applies to numeric data; however, we may extend the concept to ordinal data.
- Suppose that a given data set of N values for an attribute X is sorted in increasing order.
- If N is odd, then the median is the middle value of the ordered set. If N is even, then the median is not unique; it is the two middlemost values and any value in between.
- The median is expensive to compute when we have a large number of observations.
- For numeric attributes, however, we can easily approximate the value.

$$\text{median} = L_1 + \left(\frac{N/2 - (\sum \text{freq})_1}{\text{freq}_{\text{median}}} \right) \text{width},$$

Mode:

- The mode is another measure of central tendency.

- The mode for a set of data is the value that occurs most frequently in the set.
- Therefore, it can be determined for qualitative and quantitative attributes.
- It is possible for the greatest frequency to correspond to several different values, which results in more than one mode. Data sets with one, two, or three modes are respectively called unimodal, bimodal, and trimodal. In general, a data set with two or more modes is multimodal.

Example 2.8 Mode. The data from Example 2.6 are bimodal. The two modes are \$52,000 and \$70,000. ■

For unimodal numeric data that are moderately skewed (asymmetrical), we have the following empirical relation:

$$\text{mean} - \text{mode} \approx 3 \times (\text{mean} - \text{median}). \quad (2.4)$$

Mid Range:

- The midrange can also be used to assess the central tendency of a numeric data set.
- It is the average of the largest and smallest values in the set.
- This measure is easy to compute using the SQL aggregate functions, `max()` and `min()`.

Example 2.9 Midrange. The midrange of the data of Example 2.6 is $\frac{30,000+110,000}{2} = \$70,000$. ■

- In a unimodal frequency curve with perfect symmetric data distribution, the mean, median, and mode are all at the same center value.
- Data in most real applications are not symmetric.
- They may instead be either positively skewed, where the mode occurs at a value that is smaller than the median or negatively skewed, where the mode occurs at a value greater than the Median.

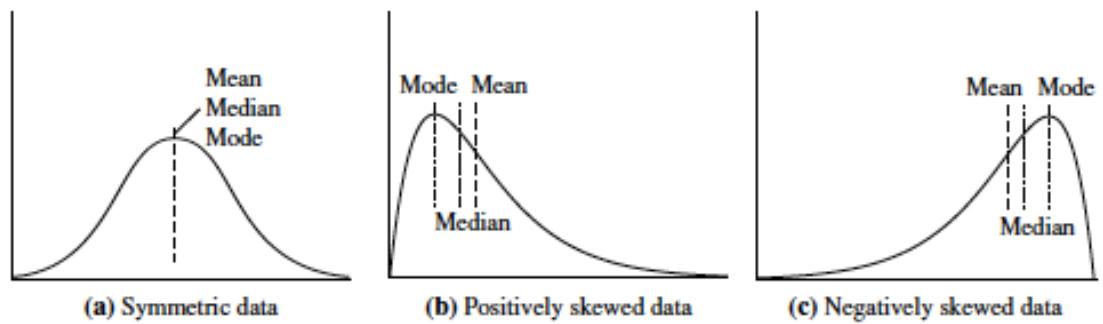


Figure 2.1 Mean, median, and mode of symmetric versus positively and negatively skewed data.

Measuring the Dispersion of Data: Range, Quartiles, Variance, Standard Deviation, and Interquartile Range

- Let x_1, x_2, \dots, x_N be a set of observations for some numeric attribute, X. The range of the set is the difference between the largest (`max()`) and smallest (`min()`) values.
- Suppose that the data for attribute X are sorted in increasing numeric order.

- Imagine that we can pick certain data points so as to split the data distribution into equal-size consecutive sets, as in Figure 2.2.
- These data points are called quantiles.
- Quantiles are points taken at regular intervals of a data distribution, dividing it into essentially equal-size consecutive sets.
- The 2-quantile is the data point dividing the lower and upper halves of the data distribution.
- It corresponds to the median.
- The 4-quantiles are the three data points that split the data distribution into four equal parts; each part represents one-fourth of the data distribution. They are more commonly referred to as quartiles.
- The 100-quantiles are more commonly referred to as percentiles; they divide the data distribution into 100 equal-sized consecutive sets.
- The median, quartiles, and percentiles are the most widely used forms of quantiles.
- The distance between the first and third quartiles is a simple measure of spread that gives the range covered by the middle half of the data. This distance is called the interquartile range (IQR) and is defined as

$$IQR = Q3 - Q1.$$

Five-Number Summary, Boxplots, and Outliers:

- A common rule of thumb for identifying suspected outliers is to single out values falling at least 1.5 IQR above the third quartile or below the first quartile.
- Because Q1, the median, and Q3 together contain no information about the endpoints (e.g., tails) of the data, a fuller summary of the shape of a distribution can be obtained by providing the lowest and highest data values as well. This is known as the five-number summary.
- The five-number summary of a distribution consists of the median (Q2), the quartiles Q1 and Q3, and the smallest and largest individual observations, written in the order of Minimum, Q1, Median, Q3, Maximum.
- Boxplots are a popular way of visualizing a distribution.
- A boxplot incorporates the five-number summary as follows:
- Typically, the ends of the box are at the quartiles so that the box length is the interquartile range.
- The median is marked by a line within the box.
- Two lines (called whiskers) outside the box extend to the smallest (Minimum) and largest (Maximum) observations.

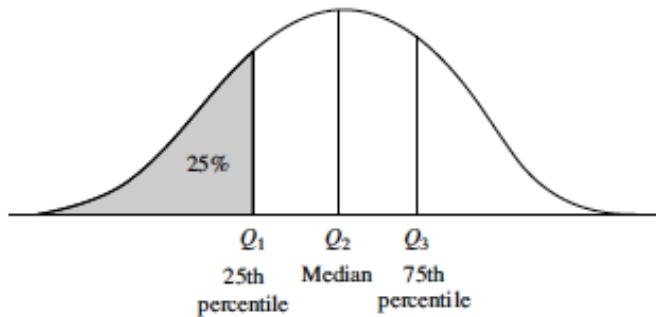


Figure 2.2 A plot of the data distribution for some attribute X . The quantiles plotted are quartiles. The three quartiles divide the distribution into four equal-size consecutive subsets. The second quartile corresponds to the median.

Boxplot: Figure shows boxplots for unit price data for items sold at four branches of AllElectronics during a given time period.

For branch 1, we see that the median price of items sold is \$80, Q1 is \$60, and Q3 is \$100. Notice that two outlying observations for this branch were plotted individually, as their values of 175 and 202 are more than 1.5 times the IQR here of 40.

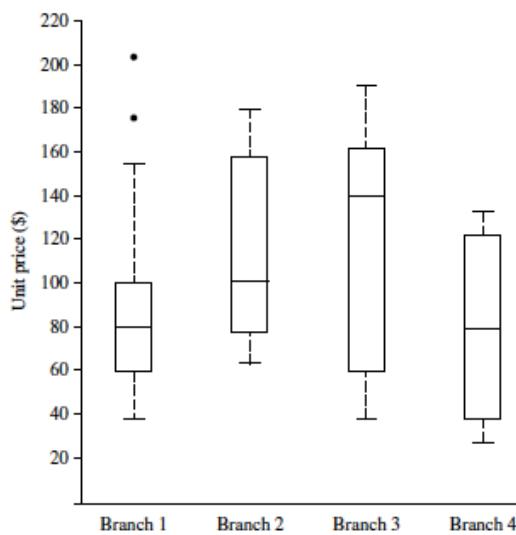


Figure 2.3 Boxplot for the unit price data for items sold at four branches of AllElectronics during a given time period.

Variance and Standard Deviation

- Variance and standard deviation are measures of data dispersion.
- They indicate how spread out a data distribution is.
- A low standard deviation means that the data observations tend to be very close to the mean, while a high standard deviation indicates that the data are spread out over a large range of values.

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2 = \left(\frac{1}{N} \sum_{i=1}^N x_i^2 \right) - \bar{x}^2,$$

The standard deviation, σ , of the observations is the square root of the variance, σ^2 .

Graphic Displays of Basic Statistical Descriptions of Data:

Quantile Plot

A quantile plot is a simple and effective way to have a first look at a univariate data distribution. First, it displays all of the data for the given attribute. Second, it plots quantile information.

Quantile-Quantile Plot

A quantile-quantile plot, or q-q plot, graphs the quantiles of one univariate distribution against the corresponding quantiles of another.

It is a powerful visualization tool in that it allows the user to view whether there is a shift in going from one distribution to another.

Histograms

Histograms (or frequency histograms) are at least a century old and are widely used. “Histos” means pole or mast, and “gram” means chart, so a histogram is a chart of poles. Plotting histograms is a graphical method for summarizing the distribution of a given attribute, X.

Scatter Plots and Data Correlation

- A scatter plot is one of the most effective graphical methods for determining if there appears to be a relationship, pattern, or trend between two numeric attributes.
- Each pair of values is treated as a pair of coordinates and plotted as points in the plane.
- The scatter plot is a useful method for providing a first look at bivariate data to see
 - clusters of points and outliers,
 - or to explore the possibility of correlation relationships.
- Two attributes, X, and Y, are correlated if one attribute implies the other.
- Correlations can be positive, negative, or null (uncorrelated).
- If the pattern of plotted points slopes from upper left to lower right, X's values increase as Y's values decrease, suggesting a negative correlation.
- A line of best fit can be drawn to study the correlation between the variables.

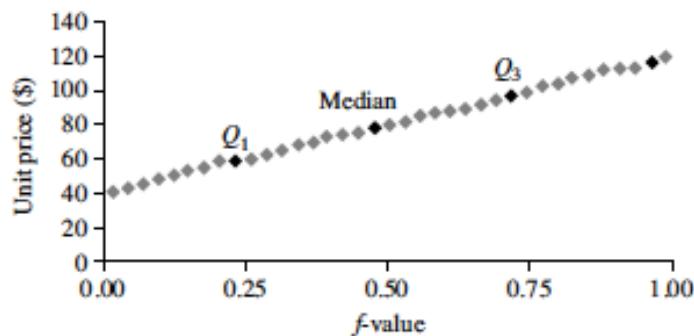


Figure 2.4 A quantile plot for the unit price data of Table 2.1.

Table 2.1 A Set of Unit Price Data for Items Sold at a Branch of *AllElectronics*

Unit price (\$)	Count of items sold
40	275
43	300
47	250
—	—
74	360
75	515
78	540
—	—
115	320
117	270
120	350

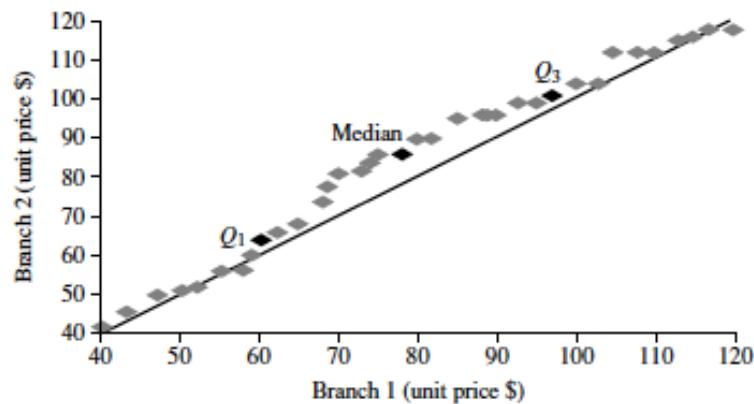


Figure 2.5 A q-q plot for unit price data from two *AllElectronics* branches.

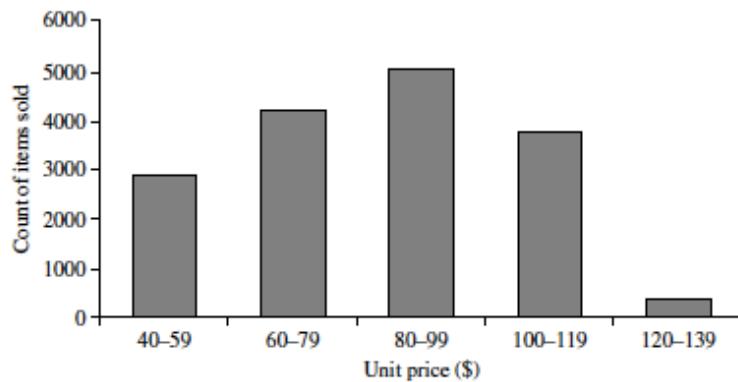


Figure 2.6 A histogram for the Table 2.1 data set.

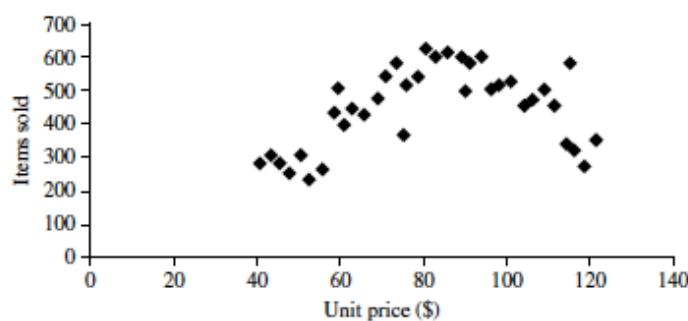


Figure 2.7 A scatter plot for the Table 2.1 data set.

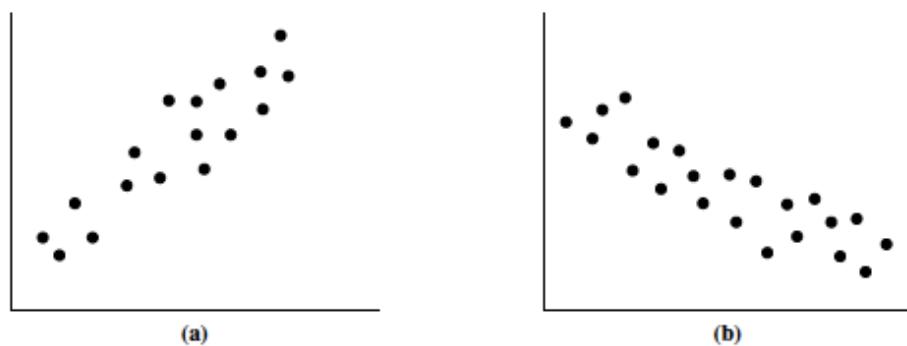


Figure 2.8 Scatter plots can be used to find (a) positive or (b) negative correlations between attributes.



Figure 2.9 Three cases where there is no observed correlation between the two plotted attributes in each of the data sets.

UNIT II

DESCRIBING DATA

Types of Data - Types of Variables -Describing Data with Tables and Graphs –Describing Data with Averages - Describing Variability - Normal Distributions and Standard (z) Scores

WHAT IS STATISTICS?

Statistics exists because of the prevalence of variability in the real world.

Descriptive Statistics:

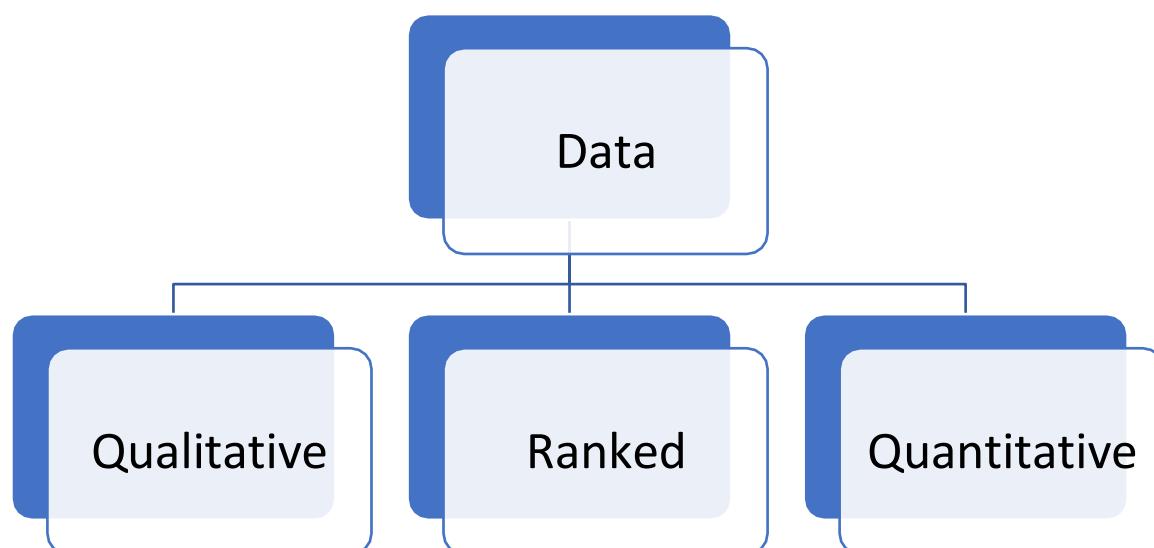
- In its simplest form, known as descriptive statistics, statistics provides us with tools—tables,
- graphs, averages, ranges, correlations—for organizing and summarizing the inevitable
- variability in collections of actual observations or scores.
- Eg: A tabular listing, ranked from most to least, A graph showing the annual change in global temperature during the last 30 years

Inferential Statistics:

- Statistics also provides tools—a variety of tests and estimates—for generalizing beyond collections of actual observations.
- This more advanced area is known as inferential statistics.
- Eg: An assertion about the relationship between job satisfaction and overall happiness

I. **THREE TYPES OF DATA:**

- Data is a collection of actual observations or scores in a survey or an experiment.
- The precise form of statistical analysis often depends on whether data are qualitative, ranked, or quantitative.



- Qualitative Data: Qualitative data consist of words (Yes or No), letters (Y or N), or numerical codes (0 or 1) that represent a class or category.
- Ranked data consist of numbers (1st, 2nd, . . . 40th place) that represent relative standing within a group.
- Quantitative data consists of numbers (weights of 238, 170, . . . 185 lbs) that represent an amount or a count.

Quantitative Data:

Table 1.1 QUANTITATIVE DATA: WEIGHTS (IN POUNDS) OF MALE STATISTICS STUDENTS							
160	168	133	170	150	165	158	165
193	169	245	160	152	190	179	157
226	160	170	180	150	156	190	156
157	163	152	158	225	135	165	135
180	172	160	170	145	185	152	
205	151	220	166	152	159	156	
165	157	190	206	172	175	154	

Quantitative Data

- The weights reported by 53 male students in Table 1.1 are quantitative data, since any single observation, such as 160 lbs, represents an amount of weight.

	Data	RData	var	
1	87.00	14.000		
2	26.00	3.000		
3	54.00	8.500		
4	39.00	6.000		
5	67.00	11.000		
6	12.00	1.000		
7	28.00	4.500		
8	98.00	15.000		
9	54.00	8.500		
10	68.00	12.000		
11	23.00	2.000		
12	64.00	10.000		
13	28.00	4.500		
14	43.00	7.000		
15	77.00	13.000		

Ranked Data

- The ranked data in order from 1 to 15 depending on the data available in the list.

Table 1.2 QUALITATIVE DATA: "DO YOU HAVE A FACEBOOK PROFILE?" YES (Y) OR NO (N) REPLIES OF STATISTICS STUDENTS							
Y	Y	Y	N	N	Y	Y	Y
Y	Y	Y	N	N	Y	Y	Y
N	Y	N	Y	Y	Y	Y	Y
Y	Y	N	Y	N	Y	N	Y
Y	N	Y	N	N	Y	Y	Y
Y	Y	N	Y	Y	Y	Y	Y
N	N	N	N	Y	N	N	Y
Y	Y	Y	Y	Y	N	Y	N
Y	Y	Y	Y	N	N	Y	Y
N	Y	N	N	Y	Y	Y	Y
Y	Y	N	N	Y	Y	Y	Y

Qualitative Data

The Y and N replies of students in Table 1.2 are qualitative data, since any single observation is a letter that represents a class of replies.

II. TYPES OF VARIABLES

- A variable is a characteristic or property that can take on different values.

Discrete and Continuous Variables

- Quantitative variables can be further distinguished in terms of whether they are discrete or continuous.
- A discrete variable consists of isolated numbers separated by gaps.
- Examples include most counts, such as the number of children in a family; the number of foreign countries you have visited; and the current size of the U.S. population.
- A continuous variable consists of numbers whose values, at least in theory, have no restrictions.
- Examples include amounts, such as weights of male statistics students; durations, and standardized test scores, such as those on the Scholastic Aptitude Test (SAT).

Approximate Numbers

- In theory, values for continuous variables can be carried out infinitely far.
- Eg: Someone's weight, in pounds, might be 140.01438, and so on, to infinity!
- Practical considerations require that values for continuous variables be rounded off.
- Whenever values are rounded off, as is always the case with actual values for continuous variables, the resulting numbers are approximate, never exact.
- For example, the weights of the to the nearest pound.
- A student whose weight is listed as 150 lbs could actually weigh between 149.5 and 150.5 lbs.

Independent and Dependent Variables

- The most studies raise questions about the presence or absence of a relationship between two (or more) variables.
- Eg: For example, a psychologist might wish to investigate whether couples who undergo special training in “active listening” tend to have fewer communication breakdowns than do couples who undergo no special training.
- An experiment is a study in which the investigator decides who receives the special treatment.

Dependent Variable

- When a variable is believed to have been influenced by the independent variable, it is called a dependent variable.
- In an experimental setting, the dependent variable is measured, counted, or recorded by the investigator.
- Unlike the independent variable, the dependent variable isn’t manipulated by the investigator.
- Instead, it represents an outcome: the data produced by the experiment.
- Eg: To test whether training influences communication, the psychologist counts the number of communication breakdowns between each couple

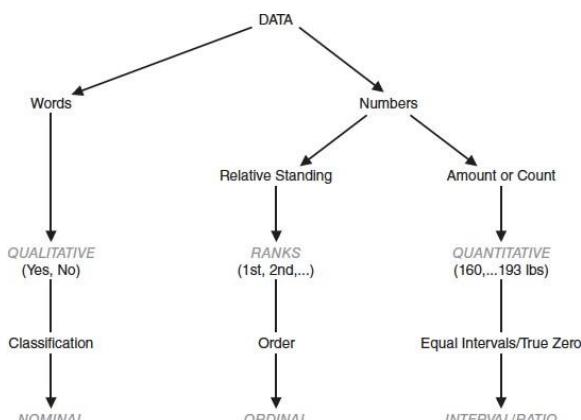


FIGURE 1.2
Overview: types of data and levels of measurement.

Observational Studies

- Instead of undertaking an experiment, an investigator might simply observe the relation between two variables. For example, a sociologist might collect paired measures of poverty level and crime rate for each individual in some group.
- Such studies are often referred to as observational studies.
- An observational study focuses on detecting relationships between variables not manipulated by the investigator, and it yields less clear-cut conclusions about cause-effect relationships than does an experiment.

Confounding Variable

- Whenever groups differ not just because of the independent variable but also because some uncontrolled variable co-varies with the independent variable, any conclusion about a cause-effect relationship is suspect.
- A difference between groups might be due not to the independent variable but to a confounding variable.
- For instance, couples willing to devote extra effort to special training might already possess a deeper commitment that co-varies with more active-listening skills.
- An uncontrolled variable that compromises the interpretation of a study is known as a confounding variable.

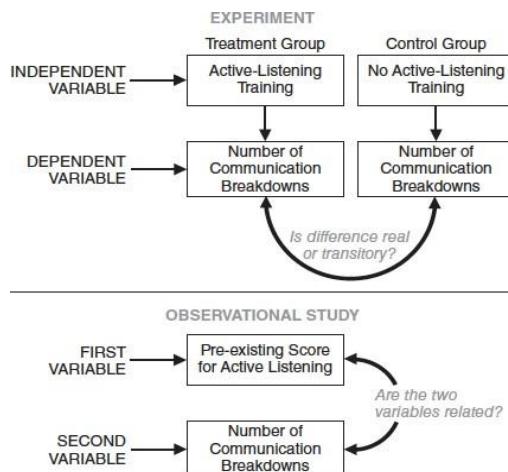


FIGURE 1.3
Overview: two active-listening studies.

Problems:

Progress Check *1.3 Indicate whether each of the following terms is *qualitative* (because it's a word, letter, or numerical code representing a class or category); *ranked* (because it's a number representing relative standing); or *quantitative* (because it's a number representing an amount or a count).

- ethnic group
- age
- family size
- academic major
- sexual preference
- IQ score
- net worth (dollars)
- third-place finish
- gender
- temperature

- 1.3** (a) qualitative (e) qualitative (i) qualitative
(b) quantitative (f) quantitative (j) quantitative
(c) quantitative (g) quantitative
(d) qualitative (h) ranked

Progress Check *1.5 Indicate whether the following quantitative observations are discrete or continuous.

- (a) litter of mice
- (b) cooking time for pasta
- (c) parole violations by convicted felons
- (d) IQ
- (e) age
- (f) population of your hometown
- (g) speed of a jetliner

- 1.5** (a) discrete (d) continuous (g) continuous
(b) continuous (e) continuous
(c) discrete (f) discrete

Progress Check *1.6 For each of the listed studies, indicate whether it is an *experiment* or an *observational study*. If it is an experiment, identify the independent variable and note any possible confounding variables.

- (a) years of education and annual income
 - (b) prescribed hours of sleep deprivation and subsequent amount of REM (dream) sleep
 - (c) weight loss among obese males who choose to participate either in a weight-loss program or a self-esteem enhancement program
 - (d) estimated study hours and subsequent test score

 - (e) recidivism among substance abusers assigned randomly to different rehabilitation programs
 - (f) subsequent GPAs of college applicants who, as the result of a housing lottery, live either on campus or off campus
- 1.6** (a) observational study
(b) experiment (independent variable: prescribed hours of sleep deprivation)
(c) experiment (independent variable: two programs; possible confounding variable: self-selection of program)
(d) observational study
(e) experiment (independent variable: different rehabilitation programs)
(f) experiment (independent variable: on campus or off campus)

III. DESCRIBING DATA WITH TABLES AND GRAPHS:

TABLES (FREQUENCY DISTRIBUTIONS)

- A frequency distribution is a collection of observations produced by sorting observations into classes and showing their frequency (f) of occurrence in each class.

Table 2.1 FREQUENCY DISTRIBUTION (UNGROUPED DATA)	
WEIGHT	<i>f</i>
245	1
244	0
243	0
242	0
*	
*	
*	
161	0
160	4
159	1
158	2
157	3
*	
*	
*	
136	0
135	2
134	0
133	1
Total	<u>53</u>

- To organize the weights of the male statistics students listed in Table 1.1. First, arrange a column of consecutive numbers, beginning with the lightest weight (133) at the bottom and ending with the heaviest weight (245) at the top.
- A short vertical stroke or tally next to a number each time its value appears in the original set of data; once this process has been completed, substitute for each tally count a number indicating the frequency (f) of occurrence of each weight.
- When observations are sorted into classes of single values, as in Table 2.1, the result is referred to as a frequency distribution for ungrouped data.
- The frequency distribution shown in Table 2.1 is only partially displayed because there are more than 100 possible values between the largest and smallest observations.

Grouped Data

- When observations are sorted into classes of more than one value, as in Table 2.2, the result is referred to as a frequency distribution for grouped data.

Table 2.2 FREQUENCY DISTRIBUTION (GROUPED DATA)	
WEIGHT	f
240–249	1
230–239	0
220–229	3
210–219	0
200–209	2
190–199	4
180–189	3
170–179	7
160–169	12
150–159	17
140–149	1
130–139	3
Total	53

- Data are grouped into class intervals with 10 possible values each.
- The bottom class includes the smallest observation (133), and the top class includes the largest observation (245).
- The distance between bottom and top is occupied by an orderly series of classes.
- The frequency (f) column shows the frequency of observations in each class and, at the bottom, the total number of observations in all classes.

Gaps between Classes:

- The size of the gap should always equal one unit of measurement.
- It should always equal the smallest possible difference between scores within a particular set of data.
- Since the gap is never bigger than one unit of measurement, no score can fall into the gap.

How Many Classes?

- Classes should not be too large and not too high.

When There Are Either Many or Few Observations:

- Grouping of classes can be 10, the recommended number of classes, as recommended.

Real Limits of Class Intervals

- The real limits are located at the midpoint of the gap between adjacent tabled boundaries; that is, one-half of one unit of measurement below the lower tabled boundary and one-half of one unit of measurement above the upper tabled boundary.
- Eg: The real limits for 140–149 in Table 2.2 are 139.5 (140 minus one-half of the unit of measurement of 1) and 149.5 (149 plus one-half of the unit of measurement of 1), and the actual width of the class interval would be 10 (from $149.5 - 139.5 = 10$).

Table 2.3 FREQUENCY DISTRIBUTION WITH TOO MANY INTERVALS	
WEIGHT	f
245–249	1
240–244	0
235–239	0
230–234	0
225–229	2
220–224	1
215–219	0
210–214	0
205–209	2
200–204	0
195–199	0
190–194	4
185–189	1
180–184	2
175–179	2
170–174	5
165–169	7
160–164	5
155–159	9
150–154	8
145–149	1
140–144	0
135–139	2
130–134	1
Total	53

Table 2.4 FREQUENCY DISTRIBUTION WITH TOO FEW INTERVALS	
WEIGHT	f
200–249	6
150–199	43
100–149	4
Total	53

GUIDELINES

Essential:

1. Each observation should be included in one, and only one, class.

Example: 130–139, 140–149, 150–159, etc.

2. List all classes, even those with zero frequencies.

Example: Listed in Table 2.2 is the class 210–219 and its frequency of zero.

3. All classes should have equal intervals.

Example: 130–139, 140–149, 150–159, etc. It would be incorrect to use 130–139, 140–159, etc.,

Optional:

4. All classes should have both an upper boundary and a lower boundary.

Example: 240–249. Less preferred would be 240–above, in which no maximum value can be assigned to observations in this class.

5. Select the class interval from convenient numbers, such as 1, 2, 3, . . . 10, particularly 5 and 10 or multiples of 5 and 10.

Example: 130–139, 140–149, in which the class interval of 10 is a convenient number.

6. The lower boundary of each class interval should be a multiple of the class interval.

Example: 130–139, 140–149, in which the lower boundaries of 130, 140, are multiples of 10, the class interval.

7. Aim for a total of approximately 10 classes. Example:

The distribution in Table 2.2 uses 12 classes.

CONSTRUCTING FREQUENCY DISTRIBUTIONS

1. Find the range
2. Find the class interval required to span the range by dividing the range by the desired number of classes
3. Round off to the nearest convenient interval
4. Determine where the lowest class should begin.
5. Determine where the lowest class should end.
6. Working upward, list as many equivalent classes as are required to include the largest observation.
7. Indicate with a tally the class in which each observation falls.
8. Replace the tally count for each class with a number—the frequency (*f*)—and show the total of all frequencies.
9. Supply headings for both columns and a title for the table.

Problems:

Progress Check *2.2 The IQ scores for a group of 35 high school dropouts are as follows:

91	85	84	79	80
87	96	75	86	104
95	71	105	90	77
123	80	100	93	108
98	69	99	95	90
110	109	94	100	103
112	90	90	98	89

- (a) Construct a frequency distribution for grouped data.
(b) Specify the *real* limits for the lowest class interval in this frequency distribution.

2.2 (a) Calculating the class width,

$$\frac{123 - 69}{10} = \frac{54}{10} = 5.4$$

Round off to a convenient number, such as 5.

IQ	TALLY*	f
120–124	/	1
115–119		0
110–114	//	2
105–109	///	3
100–104		4
95–99	/	6
90–94		7
85–89		4
80–84		3
75–79		3
70–74	/	1
65–69	/	1
Total		35

*Tally column usually is omitted from the finished table.

- (b) 64.5–69.5

Progress Check *2.3 What are some possible poor features of the following frequency distribution?

ESTIMATED WEEKLY TV VIEWING TIME (HRS) FOR 250 SIXTH GRADERS	
VIEWING TIME	f
35—above	2
30–34	5
25–30	29
20–22	60
15–19	60
10–14	34
5–9	31
0–4	29
Total	250

- 2.3** Not all observations can be assigned to one and only one class (because of gap between 20–22 and 25–30 and overlap between 25–30 and 30–34). All classes are not equal in width (25–30 versus 30–34). All classes do not have both boundaries (35—above).

OUTLIERS

- The appearance of one or more very extreme scores are called outliers.
- Ex: A GPA of 0.06, an IQ of 170, summer wages of \$62,000

Check for Accuracy

- Whenever you encounter an outrageously extreme value, such as a GPA of 0.06, attempt to verify its accuracy.
- If the outlier survives an accuracy check, it should be treated as a legitimate score.

Might Exclude from Summaries

- We might choose to segregate (but not to suppress!) an outlier from any summary of the data.
- We might use various numerical summaries, such as the median and interquartile range, to ignore extreme scores, including outliers.

Might Enhance Understanding

- A valid outlier can be viewed as the product of special circumstances, it might help you to understand the data.
- Eg: crime rates differ among communities

Problem:

Progress Check *2.4 Identify any outliers in each of the following sets of data collected from nine college students.

SUMMER INCOME	AGE	FAMILY SIZE	GPA
\$6,450	20	2	2.30
\$4,820	19	4	4.00
\$5,650	61	3	3.56
\$1,720	32	6	2.89
\$600	19	18	2.15
\$0	22	2	3.01
\$3,482	23	6	3.09
\$25,700	27	3	3.50
\$8,548	21	4	3.20

- 2.4** Outliers are a summer income of \$25,700; an age of 61; and a family size of 18. No outliers for GPA.

RELATIVE FREQUENCY DISTRIBUTIONS

- Relative frequency distributions show the frequency of each class as a part or fraction of the total frequency for the entire distribution.
- This type of distribution allows us to focus on the relative concentration of observations among different classes within the same distribution.
- In the case of the weight data in Table 2.2, it permits us to see that the 160s account for about one-fourth ($12/53 = .23$, or 23%) of all observations.

Constructing Relative Frequency Distributions

- To convert a frequency distribution into a relative frequency distribution, divide the frequency for each class by the total frequency for the entire distribution.

Table 2.5 RELATIVE FREQUENCY DISTRIBUTION		
WEIGHT	f	RELATIVE f
240–249	1	.02
230–239	0	.00
220–229	3	.06
210–219	0	.00
200–209	2	.04
190–199	4	.08
180–189	3	.06
170–179	7	.13
160–169	12	.23
150–159	17	.32
140–149	1	.02
130–139	3	.06
Total	53	1.02*

* The sum does not equal 1.00 because of rounding-off errors.

Percentages or Proportions?

- A proportion always varies between 0 and 1, whereas a percentage always varies between 0 percent and 100 percent.
- To convert the relative frequencies in Table 2.5 from proportions to percentages, multiply each proportion by 100; that is, move the decimal point two places to the right.

Problem:

Progress Check *2.5 GRE scores for a group of graduate school applicants are distributed as follows:

GRE	f
725–749	1
700–724	3
675–699	14
650–674	30
625–649	34
600–624	42
575–599	30
550–574	27
525–549	13
500–524	4
475–499	2
Total	200

Convert to a relative frequency distribution. When calculating proportions, round numbers to two digits to the right of the decimal point, using the rounding procedure specified in Section A 7 of Appendix A.

GRE	RELATIVE f
725–749	.01
700–724	.02
675–699	.07
650–674	.15
625–649	.17
600–624	.21
575–599	.15
550–574	.14
525–549	.07*
500–524	.02
475–499	.01
Totals	1.02

*From $13/200 = .065$, which rounds to .07.

CUMULATIVE FREQUENCY DISTRIBUTIONS

- Cumulative frequency distributions show the total number of observations in each class and in all lower-ranked classes.
- This type of distribution can be used effectively with sets of scores, such as test scores for intellectual or academic aptitude.
- Under these circumstances, cumulative frequencies are usually converted, in turn, to cumulative percentages. Cumulative percentages are often referred to as percentile ranks.

Constructing Cumulative Frequency Distributions

- To convert a frequency distribution into a cumulative frequency distribution, add to the frequency of each class the sum of the frequencies of all classes ranked below it.
- This gives the cumulative frequency for that class.
- Begin with the lowest-ranked class in the frequency distribution and work upward, finding the cumulative frequencies in ascending order.

Cumulative Percentages

- If relative standing within a distribution is particularly important, then cumulative frequencies are converted to cumulative percentages.

Percentile Ranks

- When used to describe the relative position of any score within its parent distribution, cumulative percentages are referred to as percentile ranks.
- The percentile rank of a score indicates the percentage of scores in the entire distribution with similar or smaller values than that score.

Approximate Percentile Ranks (from Grouped Data)

- The assignment of exact percentile ranks requires that cumulative percentages be obtained from frequency distributions for ungrouped data.
- If we have access only to a frequency distribution for grouped data, cumulative percentages can be used to assign approximate percentile ranks.

Table 2.6 CUMULATIVE FREQUENCY DISTRIBUTION			
WEIGHT	f	CUMULATIVE f	CUMULATIVE PERCENT
240–249	1	53	100
230–239	0	52	98
220–229	3	52	98
210–219	0	49	92
200–209	2	49	92
190–199	4	47	89
180–189	3	43	81
170–179	7	40	75
160–169	12	33	62
150–159	17	21	40
140–149	1	4	8
130–139	<u>3</u>	3	6
Total	53		

Problem:

Progress Check *2.6

- Convert the distribution of GRE scores shown in Question 2.5 to a cumulative frequency distribution.
- Convert the distribution of GRE scores obtained in Question 2.6(a) to a cumulative percent frequency distribution.

GRE	f
725–749	1
700–724	3
675–699	14
650–674	30
625–649	34
600–624	42
575–599	30
550–574	27
525–549	13
500–524	4
475–499	<u>2</u>
Total	200

2.6

GRE	(a) CUMULATIVE <i>f</i>	(b) CUMULATIVE PERCENT(%)
725–749	200	100
700–724	199	100
675–699	196	98
650–674	182	91
625–649	152	76
600–624	118	59
575–599	76	38
550–574	46	23
525–549	19	10
500–524	6	3
475–499	2	1

FREQUENCY DISTRIBUTIONS FOR QUALITATIVE (NOMINAL) DATA

- When, among a set of observations, any single observation is a word, letter, or numerical code, the data are qualitative.
- Determine the frequency with which observations occupy each class, and report these frequencies.
- This frequency distribution reveals that Yes replies are approximately twice as prevalent as No replies.

Table 2.7 FACEBOOK PROFILE SURVEY	
<i>Response</i>	<i>f</i>
Yes	56
No	27
Total	83

Ordered Qualitative Data

- Whether Yes is listed above or below No in Table 2.7.
- When, however, qualitative data have an ordinal level of measurement because observations can be ordered from least to most, that order should be preserved in the frequency table.

Relative and Cumulative Distributions for Qualitative Data

- Frequency distributions for qualitative variables can always be converted into relative frequency distributions.
- That a captain has an approximate percentile rank of 63 among officers since 62.5 (or 63) is the cumulative percent for this class.

Table 2.8
RANKS OF OFFICERS IN THE U.S. ARMY (PROJECTED 2016)

RANK	f	PROPORTION	CUMULATIVE PERCENT
General	311	.004*	100.0
Colonel	13,156	.167	99.6
Major	16,108	.204	82.9
Captain	29,169	.370	62.5
Lieutenant	20,083	.255	25.5
Total	78,827		

Problem:

Progress Check *2.8 Movie ratings reflect ordinal measurement because they can be ordered from most to least restrictive: NC-17, R, PG-13, PG, and G. The ratings of some films shown recently in San Francisco are as follows:

PG	PG	PG	PG-13	G
G	PG-13	R	PG	PG
R	PG	R	PG	R
NC-17	NC-17	PG	G	PG-13

- (a) Construct a frequency distribution.
- (b) Convert to relative frequencies, expressed as percentages.
- (c) Construct a cumulative frequency distribution.
- (d) Find the *approximate* percentile rank for those films with a PG rating.

2.8

MOVIE RATINGS	(a)	(b)	(c)
	f	f(%)	f
NC-17	2	10	20
R	4	20	18
PG-13	3	15	14
PG	8	40	11
G	3	15	3
Totals	20	100%	

- (d) Percentile rank for films with a PG rating is 55 (from $\frac{11}{20}$ multiplied by 100).

INTERPRETING DISTRIBUTIONS CONSTRUCTED BY OTHERS

- When inspecting a distribution for the first time, train yourself to look at the entire table, not just the distribution.
- Read the title, column headings, and any footnotes.
- Where do the data come from? Is a source cited? Next, focus on the form of the frequency distribution.
- When interpreting distributions, including distributions constructed by someone.

GRAPHS

- Data can be described clearly and concisely with the aid of a well-constructed frequency distribution.

GRAPHS FOR QUANTITATIVE DATA

Histograms

Important features of histograms:

- Equal units along the horizontal axis (the X axis, or abscissa) reflect the various class intervals of the frequency distribution.
- Equal units along the vertical axis (the Y axis, or ordinate) reflect increases in frequency.
- The intersection of the two axes defines the origin at which both numerical scales equal 0.
- Numerical scales always increase from left to right along the horizontal axis and from bottom to top along the vertical axis.
- The body of the histogram consists of a series of bars whose heights reflect the frequencies for the various classes.
- The adjacent bars in histograms have common boundaries that emphasize the continuity of quantitative data for continuous variables.

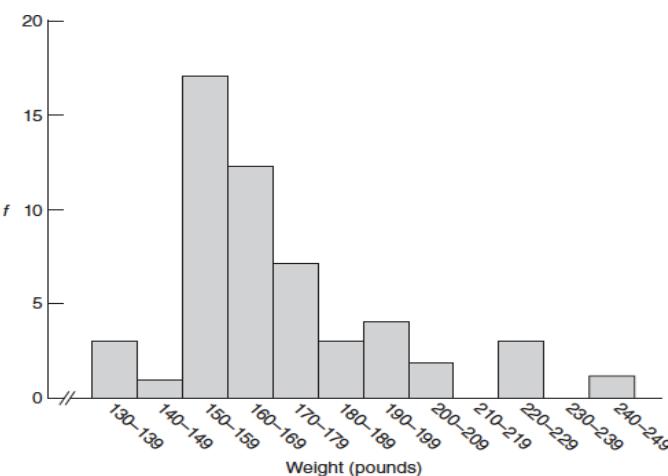


FIGURE 2.1
Histogram.

Frequency Polygon

- An important variation on a histogram is the frequency polygon, or line graph.
- Frequency polygons may be constructed directly from frequency distributions.
- However, we will follow the step-by-step transformation of a histogram into a frequency polygon.

- A. This panel shows the histogram for the weight distribution.
- B. Place dots at the midpoints of each bar top or, in the absence of bar tops, at midpoints for classes on the horizontal axis, and connect them with straight lines.
- C. Anchor the frequency polygon to the horizontal axis.
- D. Finally, erase all of the histogram bars, leaving only the frequency polygon.

Progress Check *2.9 The following frequency distribution shows the annual incomes in dollars for a group of college graduates.

INCOME	<i>f</i>
130,000–139,999	1
120,000–129,999	0
110,000–119,999	1
100,000–109,999	3
90,000–89,999	1
80,000–89,999	5
70,000–79,999	7
60,000–69,999	10
50,000–59,999	14
40,000–49,999	23
30,000–39,999	17
20,000–29,999	10
10,000–19,999	8
0–9,999	3
Total	<u>103</u>

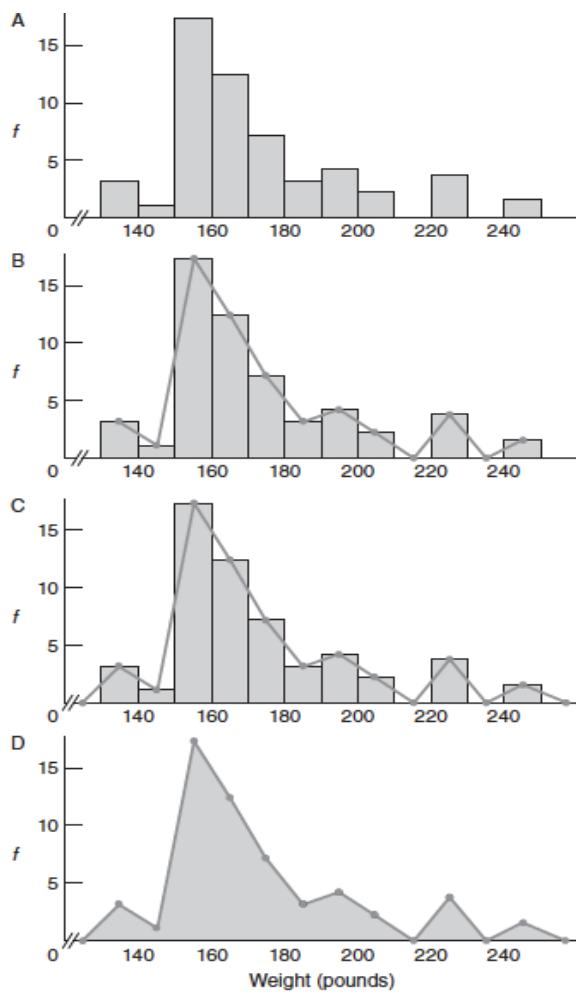
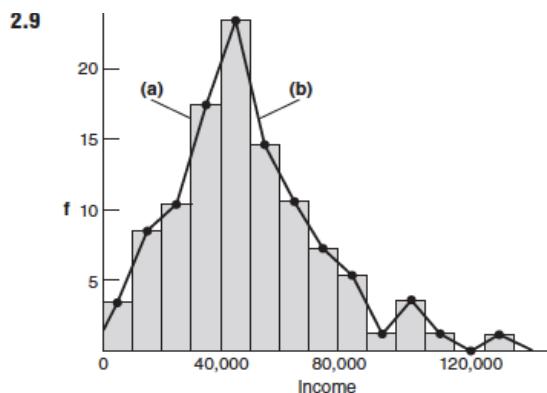


FIGURE 2.2
Transition from histogram to frequency polygon.

- (a) Construct a histogram.
- (b) Construct a frequency polygon.
- (c) Is this distribution balanced or lopsided?



NOTE: Ordinarily, only either (a) a histogram, or (b) a frequency polygon would be shown. When closing the left flank of (b), imagine extending a line to the midpoint of the first unoccupied class ($-10,000$ to -1) on the left, but stop the line at the vertical axis, as shown.

(c) Lopsided.

Stem and Leaf Displays

- Stem and leaf displays are ideal for summarizing distributions, such as that for weight data, without destroying the identities of individual observations.

Constructing a Display

- The leftmost panel of Table 2.9 re-creates the weights of the 53 male statistics students listed in Table 1.1.
- To construct the stem and leaf display for these data, when counting by tens, the weights range from the 130s to the 240s.
- Arrange a column of numbers, the stems, beginning with 13 (representing the 130s) and ending with 24 (representing the 240s).
- Draw a vertical line to separate the stems, which represent multiples of 10, from the space to be occupied by the leaves, which represent multiples of 1.
- Next, enter each raw score into the stem and leaf display.

Table 2.9 CONSTRUCTING STEM AND LEAF DISPLAY FROM WEIGHTS OF MALE STATISTICS STUDENTS				
RAW SCORES				STEM AND LEAF DISPLAY
160	165	135	175	3 5
193	168	245	165	5
226	169	170	185	14
152	160	156	154	15
180	170	160	179	16
205	150	225	165	17
163	152	190	206	18
157	160	159	165	19
151	190	172	157	20
157	150	190	156	21
220	133	166	135	22
145	180	158		23
158	152	152		24
172	170	156		5

Interpretation

- The weight data have been sorted by the stems. All weights in the 130s are listed together; all of those in the 140s are listed together, and so on.
- A glance at the stem and leaf display in Table 2.9 shows essentially the same pattern of weights depicted by the frequency distribution in Table 2.2 and the histogram.

Selection of Stems

- Stem values are not limited to units of 10.
- Depending on the data, you might identify the stem with one or more leading digits that culminates in some variation on a stem value of 10, such as 1, 100, 1000, or even .1, .01, .001, and so on.
- Stem and leaf displays represent statistical bargains.

Problem:

Progress Check *2.10 Construct a stem and leaf display for the following IQ scores obtained from a group of four-year-old children.

120	98	118	117	99	111
126	85	88	124	104	113
108	141	123	137	78	96
102	132	109	106	143»	

2.10	7	8
	8	5 8
	9	8 9 6
	10	8 2 9 6 4
	11	8 7 1 3
	12	0 6 3 4
	13	2 7
	14	1 3

Note: The order of the leaves within each stem depends on whether you entered IQ scores column by column (as above) or row by row.

TYPICAL SHAPES

- Whether expressed as a histogram, a frequency polygon, or a stem and leaf display, an important characteristic of a frequency distribution is its shape.

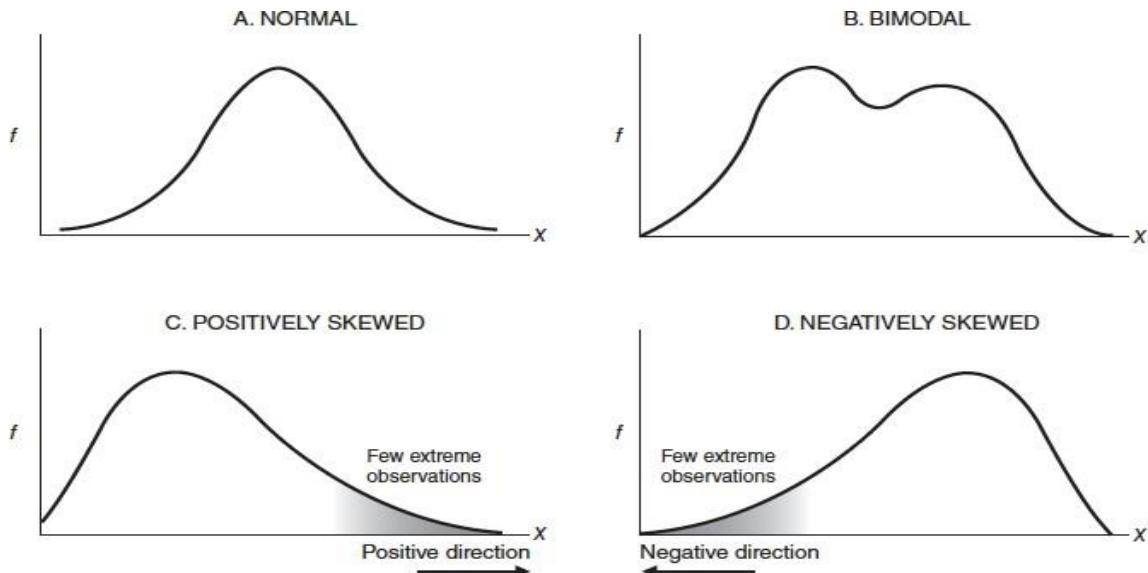


FIGURE 2.3
Typical shapes.

Normal

- Any distribution that approximates the normal shape in panel A of Figure 2.3 can be analyzed
- The familiar bell-shaped silhouette of the normal curve can be superimposed on many frequency distributions, Eg: uninterrupted gestation periods of human fetuses, scores on standardized tests, and even the popping times of individual kernels in a batch of popcorn.

Bimodal

- Any distribution that approximates the bimodal shape in panel B of Figure 2.3 reflect the coexistence of two different types of observations in the same distribution.
- Eg: The distribution of the ages of residents in a neighborhood consisting largely of either new parents or their infants has a bimodal shape.

Positively Skewed

- The two remaining shapes in Figure 2.3 are lopsided.
- A lopsided distribution caused by a few extreme observations in the positive direction as in panel C of Figure 2.3, is a positively skewed distribution.
- Eg: most family incomes under \$200,000 and relatively few family incomes spanning a wide range of values above \$200,000.

Negatively Skewed

- A lopsided distribution caused by a few extreme observations in the negative direction as in panel D of Figure 2.3, is a negatively skewed distribution.
- Eg: Most retirement ages at 60 years or older and relatively few retirement ages spanning the wide range of ages younger than 60.

Progress Check *2.11 Describe the probable shape—normal, bimodal, positively skewed, or negatively skewed—for each of the following distributions:

- (a) female beauty contestants' scores on a masculinity test, with a higher score indicating a greater degree of masculinity
- (b) scores on a standardized IQ test for a group of people selected from the general population
- (c) test scores for a group of high school students on a very difficult college-level math exam
- (d) reading achievement scores for a third-grade class consisting of about equal numbers of regular students and learning-challenged students
- (e) scores of students at the Eastman School of Music on a test of music aptitude (designed for use with the general population)

2.11 (a) Positively skewed
(b) Normal
(c) Positively skewed

(d) Bimodal
(e) Negatively skewed

A GRAPH FOR QUALITATIVE (NOMINAL) DATA

- The equal segments along the horizontal axis are allocated to the different words or classes that appear in the frequency distribution for qualitative data.
- Likewise, equal segments along the vertical axis reflect increases in frequency.
- The body of the bar graph consists of a series of bars whose heights reflect the frequencies for the various words or classes.
- A person's answer to the question "Do you have a Facebook profile?" is either Yes or No, not some impossible intermediate value, such as 40 percent Yes and 60 percent No.

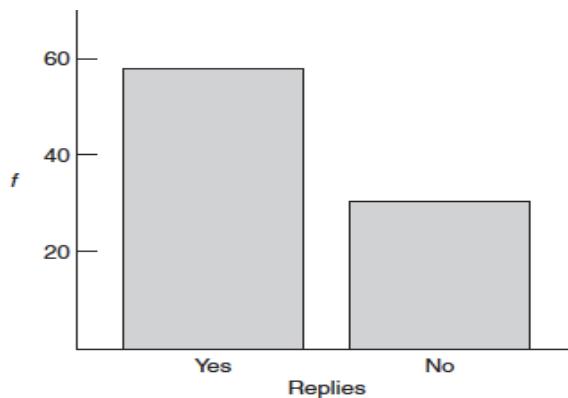


FIGURE 2.4
Bar graph.

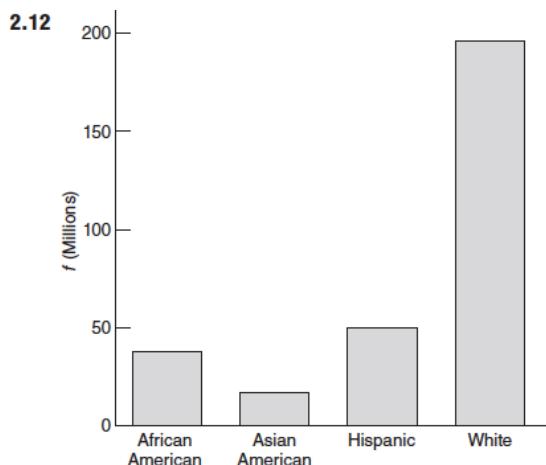
Progress Check *2.12 Referring to the box “Constructing Graphs” on page 47 for step-by-step instructions, construct a bar graph for the data shown in the following table:

RACE/ETHNICITY OF U.S. POPULATION, 2010 (IN MILLIONS)	
Race/Ethnicity	f
African American	37.7
Asian American*	17.2
Hispanic	50.5
White	196.8
Total**	302.2

*Mostly Asians, but also other races, such as Native Americans and Eskimos.

**Total does not include 6.6 million non-Hispanics reporting two or more races.

Source: www.uscensus.gov/prod/census2010/



MISLEADING GRAPHS

- Graphs can be constructed in an unscrupulous manner to support a particular point of view.
- For example, to imply that comparatively many students responded Yes to the Facebook profile question, an unscrupulous person might resort to the various tricks.
- The width of the Yes bar is more than three times that of the No bar, thus violating the custom that bars be equal in width.
- The lower end of the frequency scale is omitted, thus violating the custom that the entire scale be reproduced, beginning with zero.
- The height of the vertical axis is several times the width of the horizontal axis, thus violating the custom, heretofore unmentioned, that the vertical axis be approximately as tall as the horizontal axis is wide.

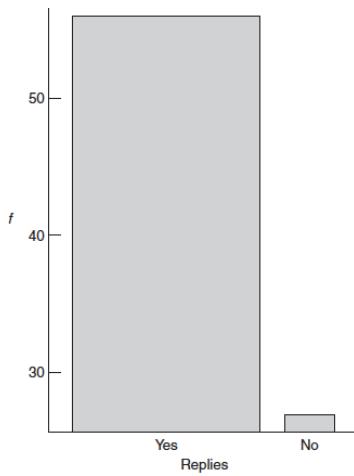
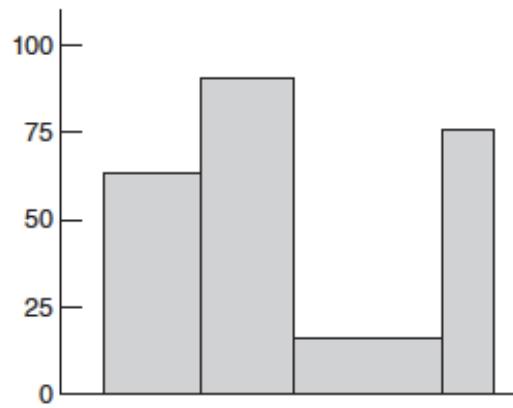


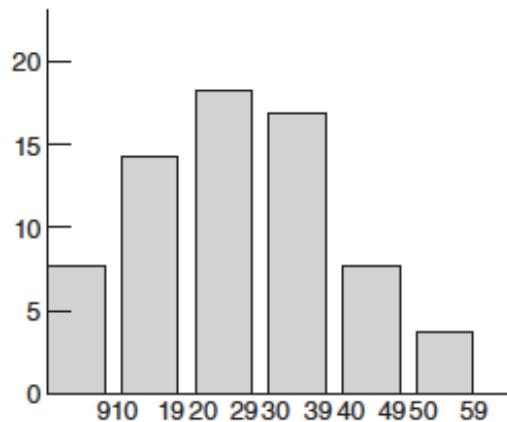
FIGURE 2.5
Distorted bar graph.

Problem:

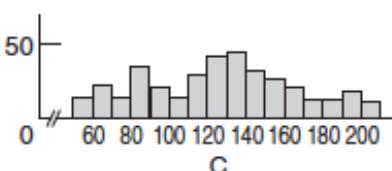
Progress Check *2.13 Criticize the graphs that appear here (ignore the inadequate labeling of both axes).



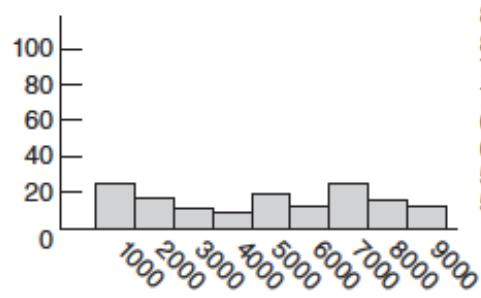
A



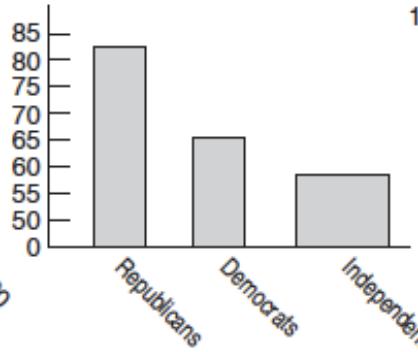
B



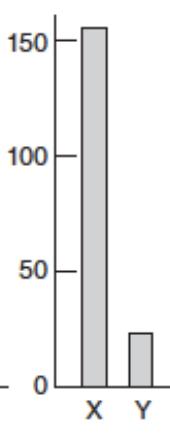
C



D



E



F

- 2.13**
- (a) Widths of two rightmost bars aren't the same as those of two leftmost bars.
 - (b) Histogram is more appropriate, assuming numbers are for a continuous quantitative variable.
 - (c) Height of the vertical axis is too small relative to the width of the horizontal axis, causing the histogram to be squashed.
 - (d) Poorly selected frequency scale along the vertical axis, causing the histogram to be squashed.
 - (e) Bars have unequal widths. There are no wiggly lines along vertical axis indicating a break between 0 and 50.
 - (f) Height of the vertical axis is too large relative to the horizontal axis, causing the differences between the bars to be exaggerated.

IV. DESCRIBING DATA WITH AVERAGES:

MODE

- The mode reflects the value of the most frequently occurring score.

Table 3.1 TERMS IN YEARS OF 20 RECENT U.S. PRESIDENTS, LISTED CHRONOLOGICALLY	
4	(Harrison)
4	
4	
8	
4	
8	
2	
6	
4	
12	
8	
8	
2	
6	
5	
3	
4	
8	
4	
8	(Clinton)

Source: The New York Times Almanac (2012).

- Distributions can have more than one mode.
- Distributions with two obvious peaks, even though they are not exactly the same height, are referred to as bimodal.
- Distributions with more than two peaks are referred to as multimodal.
- The presence of more than one mode might reflect important differences among subsets of data.

Problems:

Progress Check *3.1 Determine the mode for the following retirement ages: 60, 63, 45, 63, 65, 70, 55, 63, 60, 65, 63.

$$\text{mode} = 63$$

Progress Check *3.2 The owner of a new car conducts six gas mileage tests and obtains the following results, expressed in miles per gallon: 26.3, 28.7, 27.4, 26.6, 27.4, 26.9. Find the mode for these data.

$$\text{mode} = 27.4$$

MEDIAN

- The median reflects the middle value when observations are ordered from least to most.
- The median splits a set of ordered observations into two equal parts, the upper and lower halves.
- In other words, the median has a percentile rank of 50, since observations with equal or smaller values constitute 50 percent of the entire distribution.

Finding the Median

Table 3.2
FINDING THE MEDIAN

A. INSTRUCTIONS

- 1 Order scores from least to most.
- 2 Find the middle position by adding one to the total number of scores and dividing by 2.
- 3 If the middle position is a whole number, as in the left-hand panel below, use this number to count into the set of ordered scores.
- 4 The value of the median equals the value of the score located at the middle position.
- 5 If the middle position is not a whole number, as in the right-hand panel below, use the two nearest whole numbers to count into the set of ordered scores.
- 6 The value of the median equals the value midway between those of the two middlemost scores; to find the midway value, add the two given values and divide by 2.

B. EXAMPLES

Set of five scores:

2, 8, 2, 7, 6

1 2, 2, 6, 7, 8

2 $\frac{5+1}{2}=3$

2, 2, 6, 7, 8

3 1, 2, 3

4 median = 6

Set of six scores:

3, 8, 9, 3, 1, 8

1 1, 3, 3, 8, 8, 9

2 $\frac{6+1}{2}=3.5$

1, 3, 3, 8, 8, 9

5 1, 2, 3, 4

6 median = $\frac{3+8}{2}=5.5$

- To find the median, scores always must be ordered from least to most

- When the total number of scores is odd, as in the lower left-hand panel of Table 3.2, there is a single middle-ranked score, and the value of the median equals the value of this score.
- When the total number of scores is even, as in the lower right-hand panel of Table 3.2, the value of the median equals a value midway between the values of the two middlemost scores.
- In either case, the value of the median always reflects the value of middle-ranked scores, not the position of these scores among the set of ordered scores.
- The median term can be found for the 20 presidents.

Problems:

Progress Check *3.3 Find the median for the following retirement ages: 60, 63, 45, 63, 65, 70, 55, 63, 60, 65, 63.

$$\text{median} = 63$$

Progress Check *3.4 Find the median for the following gas mileage tests: 26.3, 28.7, 27.4, 26.6, 27.4, 26.9.

$$\text{median} = 27.15$$

Table 3.3 TERMS IN YEARS OF 20 RECENT U.S. PRESIDENTS		
ARRANGED BY LENGTH	DEVIATION FROM MEAN	SUM OF DEVIATIONS
12	6.40	
8	2.40	
8	2.40	
8	2.40	
8	2.40	
8	2.40	
8	2.40	
6	0.40	
6	0.40	
(mean = 5.60)		21.6
5	-0.60	
4	-1.60	
4	-1.60	
4	-1.60	
4	-1.60	
4	-1.60	
4	-1.60	
3	-2.60	
2	-3.60	
2	-3.60	0
		-21.6

MEAN

- The mean is the most common average, calculated many times.
- The mean is found by adding all scores and then dividing by the number of scores.

$$\text{Mean} = \frac{\text{sum of all scores}}{\text{number of scores}}$$

- To find the mean term for the 20 presidents, add all 20 terms in Table 3.1 (4 + . . . + 4 + 8) to obtain a sum of 112 years, and then divide this sum by 20, the number of presidents, to obtain a mean of 5.60 years.

Sample or Population?

- Statisticians distinguish between two types of means—the population mean and the sample mean—depending on whether the data are viewed as a population (a complete set of scores) or as a sample (a subset of scores).

Formula for Sample Mean

- When symbols are used, X designates the sample mean, and the formula becomes and reads: “ X -bar equals the sum of the variable X divided by the sample size n .”

SAMPLE MEAN

$$\bar{X} = \frac{\sum X}{n} \quad (3.1)$$

Formula for Population Mean

- The formula for the population mean differs from that for the sample mean only because of a change in some symbols.
- The population mean is represented by μ (pronounced “mu”), the lowercase Greek letter m for mean, where the uppercase letter N refers to the population size.
- Otherwise, the calculations are the same as those for the sample mean.

POPULATION MEAN

$$\mu = \frac{\sum X}{N} \quad (3.2)$$

Mean as Balance Point

- The mean serves as the balance point for its frequency distribution.
- The mean serves as the balance point for its distribution because of a special property:
- The sum of all scores, expressed as positive and negative deviations from the mean, always equals zero.
- In its role as balance point, the mean describes the single point of equilibrium at which, once all scores have been expressed as deviations from the mean.
- The mean reflects the values of all scores, not just those that are middle ranked (as with the median), or those that occur most frequently (as with the mode).

Problems:

Progress Check *3.5 Find the mean for the following retirement ages: 60, 63, 45, 63, 65, 70, 55, 63, 60, 65, 63.

$$\text{mean} = \frac{672}{11} = 61.09$$

Progress Check *3.6 Find the mean for the following gas mileage tests: 26.3, 28.7, 27.4, 26.6, 27.4, 26.9.

$$\text{mean} = \frac{163.3}{6} = 27.22$$

WHICH AVERAGE?

If Distribution Is Not Skewed

- When a distribution of scores is not too skewed, the values of the mode, median, and mean are similar, and any of them can be used to describe the central tendency of the distribution.

If Distribution Is Skewed

- When extreme scores cause a distribution to be skewed, as for the infant death rates for selected countries listed in Table 3.4, the values of the three averages can differ.

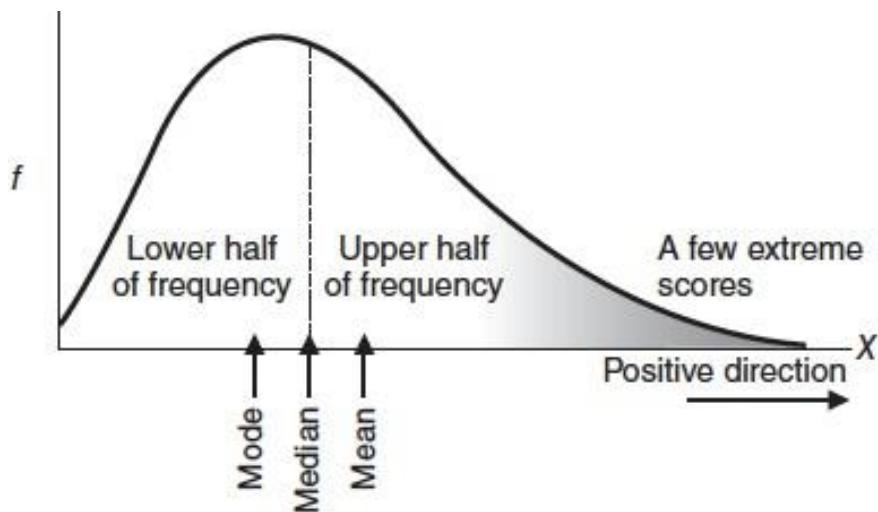
Table 3.4 INFANT DEATH RATES FOR SELECTED COUNTRIES (2012)	
COUNTRY	INFANT DEATH RATE*
Sierra Leone	182
Pakistan	86
Ghana	72
India	56
South Africa	45
Cambodia	40
Mexico	16
China	14
Brazil	14
United States	7
Cuba	6
United Kingdom	6
Netherlands	4
Israel	4
France	4
Denmark	4
Germany	4
Japan	3
Sweden	3

*Rates per 1000 live births.

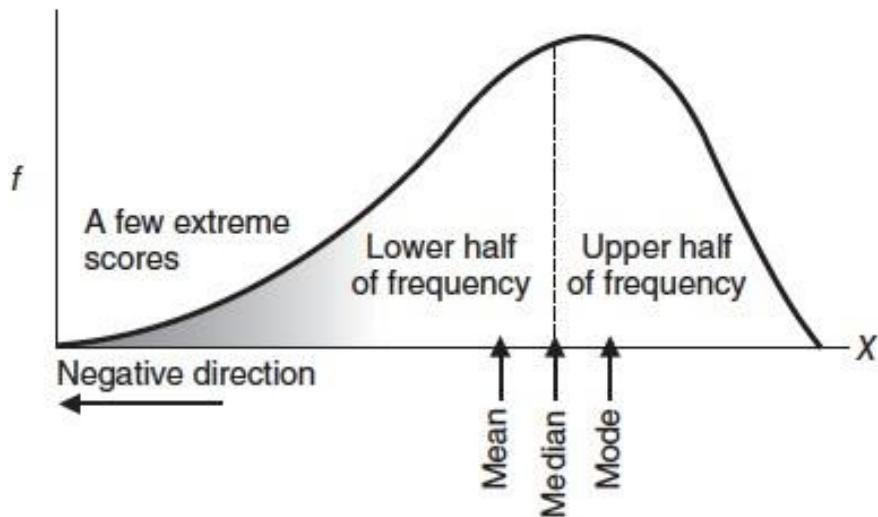
Source: 2014 World Development Indicators.

Interpreting Differences between Mean and Median

- When a distribution is skewed, report both the mean and the median.
- The differences between the values of the mean and median signal the presence of a skewed distribution.
- If the mean exceeds the median, as it does for the infant death rates, the underlying distribution is positively skewed because of one or more scores with relatively large values, such as the very high infant death rates for a number of countries, especially Sierra Leone.
- On the other hand, if the median exceeds the mean, the underlying distribution is negatively skewed because of one or more scores with relatively small values.



A. Positively Skewed Distribution
(mean exceeds median)



B. Negatively Skewed Distribution
(median exceeds mean)

FIGURE 3.2

Mode, median, and mean in positively and negatively skewed distributions.

Problem:

Progress Check *3.7 Indicate whether the following skewed distributions are positively skewed because the mean exceeds the median or negatively skewed because the median exceeds the mean.

- a distribution of test scores on an easy test, with most students scoring high and a few students scoring low
- a distribution of ages of college students, with most students in their late teens or early twenties and a few students in their fifties or sixties
- a distribution of loose change carried by classmates, with most carrying less than \$1 and with some carrying \$3 or \$4 worth of loose change
- a distribution of the sizes of crowds in attendance at a popular movie theater, with most audiences at or near capacity

- 3.7** (a) negatively skewed because the median exceeds the mean
 (b) positively skewed because the mean exceeds the median
 (c) positively skewed
 (d) negatively skewed

Special Status of the Mean

- The mean is the single most preferred average for quantitative data.

Using the Word Average

- An average can refer to the mode, median, or mean—or even geometric mean or the harmonic mean.
- Conventional usage prescribes that average usually signifies mean, and this connotation is often reinforced by the context.
- For instance, grade point average is virtually synonymous with mean grade point.

AVERAGES FOR QUALITATIVE AND RANKED DATA

Mode Always Appropriate for Qualitative Data

- But when the data are qualitative, your choice among averages is restricted.
- The mode always can be used with qualitative data.

Median Sometimes Appropriate

- The median can be used whenever it is possible to order qualitative data from least to most because the level of measurement is ordinal.
- Do not treat the various classes as though they have the same frequencies when they actually have different frequencies.

Inappropriate Averages

- It would not be appropriate to report a median for unordered qualitative data with nominal measurement, such as the ancestries of Americans.

Table 3.5 FINDING THE MEDIAN FOR ORDERED QUALITATIVE DATA: RANKS OF OFFICERS IN THE U.S. ARMY (PROJECTED 2016)		
RANK	%	CUMULATIVE %
General	0.4	
Colonel	16.7	
Major	20.4	
Captain	37.0	25.5+37.0 = 62.5
Lieutenant	25.5	25.5
	100.0	

Source: <http://www.Statista.com/statistics>

Problem:

Progress Check *3.8 College students were surveyed about where they would most like to spend their spring break: Daytona Beach (DB), Cancun, Mexico (C), South Padre Island (SP), Lake Havasu (LH), or other (O). The results were as follows:

DB	DB	C	LH	DB
C	SP	LH	DB	O
O	SP	C	DB	LH
DB	C	DB	O	DB

Find the mode and, if possible, the median.

3.8 mode = DB (Daytona Beach)

Impossible to find the median when qualitative data are unordered, with only nominal measurement.

Averages for Ranked Data

- When the data consist of a series of ranks, with its ordinal level of measurement, the median rank always can be obtained.
- It's simply the middlemost or average of the two middlemost ranks.

V. DESCRIBING VARIABILITY:

- In Figure 4.1, each of the three frequency distributions consists of seven scores with the same mean (10) but with different variabilities.
- Before reading on, rank the three distributions from least to most variable.
- The distribution A has the least variability, distribution B has intermediate variability, and distribution C has the most variability.
- For distribution A with the least (zero) variability, all seven scores have the same value (10).
- For distribution B with intermediate variability, the values of scores vary slightly (one 9 and one 11), and for distribution C with most variability, they vary even more (one 7, two 9s, two 11s, and one 13).

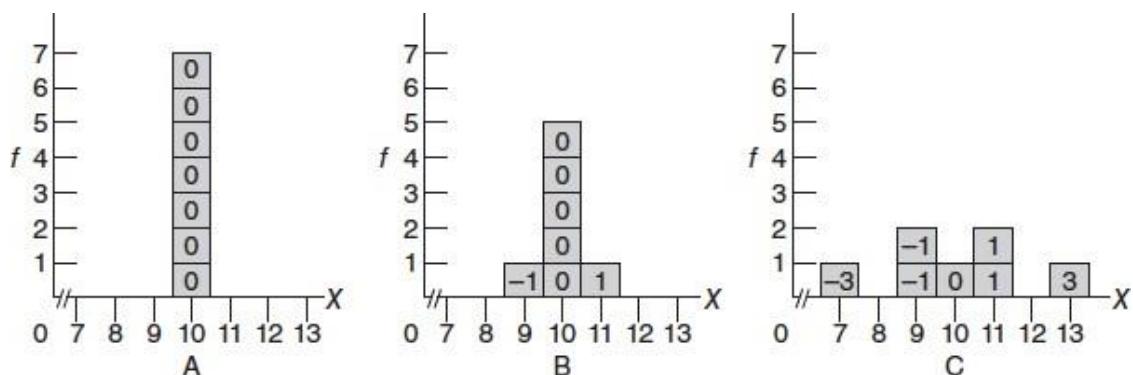


FIGURE 4.1

Three distributions with the same mean (10) but different amounts of variability. Numbers in the boxes indicate distances from the mean.

Importance of Variability

- Variability assumes a key role in an analysis of research results.
- Eg: A researcher might ask: Does fitness training improve, on average, the scores of depressed patients on a mental-wellness test?
- To answer this question, depressed patients are randomly assigned to two groups, fitness training is given to one group, and wellness scores are obtained for both groups.
- Figure 4.2 shows the outcomes for two fictitious experiments, each with the same mean difference of 2, but with the two groups in experiment B having less variability than the two groups in experiment C.
- Notice that groups B and C in Figure 4.2 are the same as their counterparts in Figure 4.1.
- Although the new group B^* retains exactly the same (intermediate) variability as group B, each of its seven scores and its mean have been shifted 2 units to the right.
- Likewise, although the new group C^* retains exactly the same (most) variability as group C, each of its seven scores and its mean have been shifted 2 units to the right.
- Consequently, the crucial mean difference of 2 ($12 - 10 = 2$) is the same for both experiments.

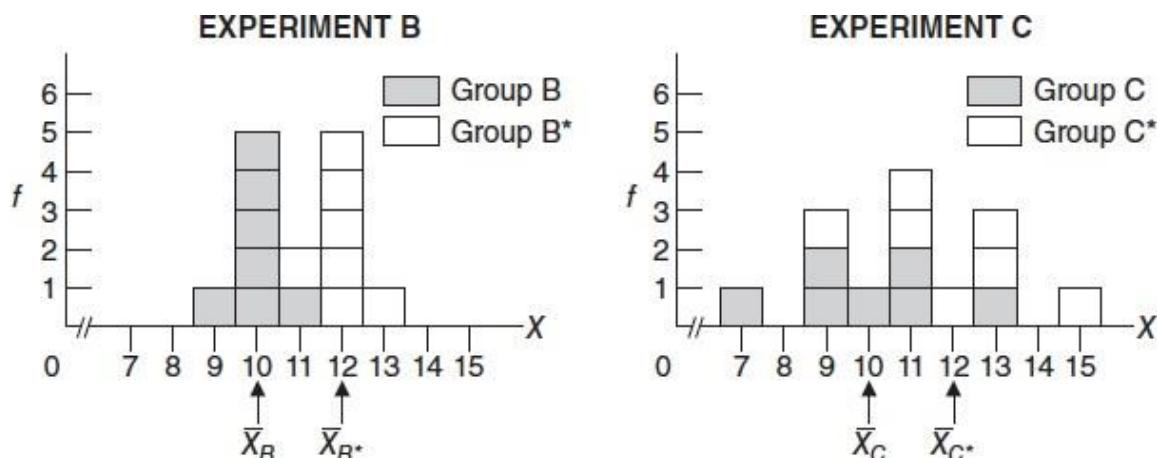


FIGURE 4.2

Two experiments with the same mean difference but dissimilar variabilities.

- variabilities within groups assume a key role in inferential statistics.
- The relatively larger variabilities within groups in experiment C translate into less statistical stability for the observed mean difference of 2 when it is viewed as just one outcome among many possible outcomes for repeat experiments.

Progress Check *4.1 For a given mean difference—say, 10 points—between two groups, what degree of variability within each of these groups (small, medium, or large) makes the mean difference

(a) ...most conspicuous with more statistical stability?

(b) ...least conspicuous with less statistical stability?

- 4.1 (a) small
- (b) large

RANGE

- The range is the difference between the largest and smallest scores.
- In Figure 4.1, distribution A, the least variable, has the smallest range of 0 (from 10 to 10); distribution B, the moderately variable, has an intermediate range of 2 (from 11 to 9); and distribution C, the most variable, has the largest range of 6 (from 13 to 7), in agreement with our intuitive judgments about differences in variability.

Disadvantages of Range

- The range has several shortcomings.
- First, since its value depends on only two scores—the largest and the smallest—it fails to use the information provided by the remaining scores.
- The value of the range tends to increase with increases in the total number of scores.

VARIANCE

- Variance and Standard Deviation are the two important measurements in statistics.
- Variance is a measure of how data points vary from the mean.
- The standard deviation is the measure of the distribution of statistical data.

Reconstructing the Variance

- To qualify as a type of mean, the values of all scores must be added and then divided by the total number of scores.
- In the case of the variance, each original score is re-expressed as a distance or deviation from the mean by subtracting the mean.

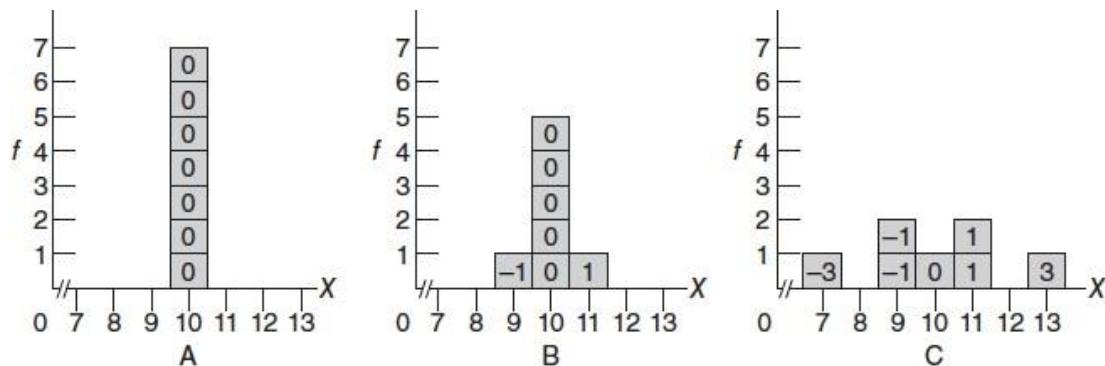


FIGURE 4.1

Three distributions with the same mean (10) but different amounts of variability. Numbers in the boxes indicate distances from the mean.

- For each of the three distributions in Figure 4.1, the face values of the seven original scores have been re-expressed as deviation scores from their mean of 10.
- For example, in distribution C, one score coincides with the mean of 10, four scores (two 9s and two 11s) deviate 1 unit from the mean, and two scores (one 7 and one

13) deviate 3 units from the mean, yielding a set of seven deviation scores: one 0, two -1 s, two 1 s, one -3 , and one 3 .

Mean of the Deviations Not a Useful Measure

- The sum of all negative deviations always counterbalances the sum of all positive deviations, regardless of the amount of variability in the group.
- A measure of variability, known as the mean absolute deviation (or m.a.d.), can be salvaged by summing all absolute deviations from the mean, that is, by ignoring negative signs.

Mean of the Squared Deviations

- Before calculating the variance (a type of mean), negative signs must be eliminated from deviation scores. Squaring each deviation generates a set of squared deviation scores, all of which are positive.

STANDARD DEVIATION

- The standard deviation, the square root of the mean of all squared deviations from the mean, that is,

$$\text{standard deviation} = \sqrt{\text{variance}}$$

- The standard deviation is a rough measure of the average amount by which scores deviate on either side of their mean.
- The standard deviation as a rough measure of the average amount by which scores deviate on either side of their mean.

Majority of Scores within One Standard Deviation

For most frequency distributions, a majority of all scores are within one standard deviation on either side of the mean.

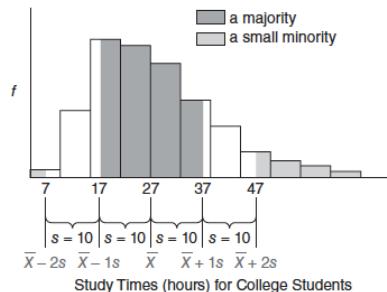
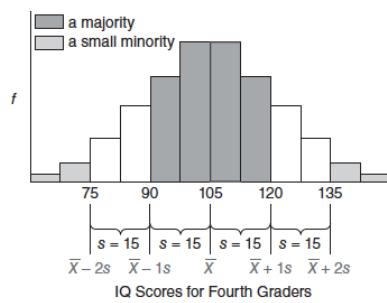


FIGURE 4.3
Some generalizations that apply to most frequency distributions.

- In Figure 4.3, where the lowercase letter s represents the standard deviation.
- As suggested in the top panel of Figure 4.3, if the distribution of IQ scores for a class of fourth graders has a mean (X) of 105 and a standard deviation (s) of 15, a majority of their IQ scores should be within one standard deviation on either side of the mean, that is, between 90 and 120.
- For most frequency distributions, a small minority of all scores deviate more than two standard deviations on either side of the mean.
- For instance, among the seven deviations in distribution C, none deviates more than two standard deviations ($2 \sqrt{1.77} = 3.54$) on either side of the mean.

Generalizations Are for All Distributions

- These two generalizations about the majority and minority of scores are independent of the particular shape of the distribution.

Standard Deviation: A Measure of Distance

- There's an important difference between the standard deviation and mean.
- The mean is a measure of position, but the standard deviation is a measure of distance. Figure 4.4 describes the weight distribution for the males.
- The mean (X) of 169.51 lbs has a particular position or location along the horizontal axis: It is located at the point, and only at the point, corresponding to 169.51 lbs.
- On the other hand, the standard deviation (s) of 23.33 lbs for the same distribution has no particular location along the horizontal axis.

Value of Standard Deviation Cannot Be Negative

- Standard deviation distances always originate from the mean and are expressed as positive deviations above the
- The actual value of the standard deviation can be zero or a positive number, it can never be a negative number because any negative deviation disappears when squared.

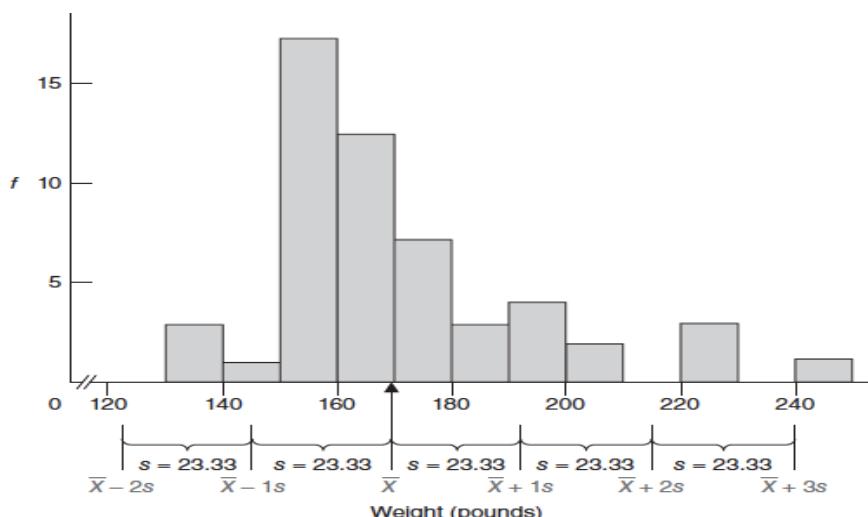


FIGURE 4.4
Weight distribution with mean and standard deviation.

Problem:

Progress Check *4.2 Employees of Corporation A earn annual salaries described by a mean of \$90,000 and a standard deviation of \$10,000.

- (a) The majority of all salaries fall between what two values?
- (b) A small minority of all salaries are less than what value?
- (c) A small minority of all salaries are more than what value?
- (d) Answer parts (a), (b), and (c) for Corporation B's employees, who earn annual salaries described by a mean of \$90,000 and a standard deviation of \$2,000.

4.2 (a) \$80,000 to \$100,000

(b) \$70,000

(c) \$110,000

(d) \$88,000 to \$92,000; \$86,000; \$94,000

Progress Check *4.3 Assume that the distribution of IQ scores for all college students has a mean of 120, with a standard deviation of 15. These two bits of information imply which of the following?

- (a) All students have an IQ of either 105 or 135 because everybody in the distribution is either one standard deviation above or below the mean. True or false?
- (b) All students score between 105 and 135 because everybody is within one standard deviation on either side of the mean. True or false?
- (c) On the average, students deviate approximately 15 points on either side of the mean. True or false?
- (d) Some students deviate more than one standard deviation above or below the mean. True or false?
- (e) All students deviate more than one standard deviation above or below the mean. True or false?
- (f) Scott's IQ score of 150 deviates two standard deviations above the mean. True or false?

4.3 (a) False. Relatively few students will score exactly one standard deviation from the mean.

(b) False. Students will score both within and beyond one standard deviation from the mean.

(c) True

(d) True

(e) False. See (b).

(f) True

STANDARD DEVIATION

Sum of Squares (SS)

- Calculating the standard deviation requires that we obtain first a value for the variance.
- However, calculating the variance requires, in turn, that we obtain the sum of the squared deviation scores.
- The sum of squared deviation scores symbolized by SS, merits special attention because it's a major component in calculations for the variance, as well as many other statistical measures.

Sum of Squares Formulas for Population

SUM OF SQUARES (SS) FOR POPULATION (DEFINITION FORMULA)

$$SS = \sum(X - \mu)^2 \quad (4.1)$$

where SS represents the sum of squares, Σ directs us to sum over the expression to its right, and $(X - \mu)^2$ denotes each of the squared deviation scores. Formula 4.1 should be read as “The sum of squares equals the sum of all squared deviation scores.” You can reconstruct this formula by remembering the following three steps:

1. Subtract the population mean, μ , from each original score, X , to obtain a deviation score, $X - \mu$.
2. Square each deviation score, $(X - \mu)^2$, to eliminate negative signs.
3. Sum all squared deviation scores, $\Sigma (X - \mu)^2$.

SUM OF SQUARES (SS) FOR POPULATION (COMPUTATION FORMULA)

$$SS = \sum X^2 - \frac{(\sum X)^2}{N} \quad (4.2)$$

where $\sum X^2$, the sum of the squared X scores, is obtained by *first squaring each X score and then summing all squared X scores*; $(\sum X)^2$, the square of sum of all X scores, is obtained by *first adding all X scores and then squaring the sum of all X scores*; and N is the population size.

Table 4.1
CALCULATION OF POPULATION STANDARD DEVIATION σ
(DEFINITION FORMULA)

A. COMPUTATION SEQUENCE

- Assign a value to N **1** representing the number of X scores
- Sum all X scores **2**
- Obtain the mean of these scores **3**
- Subtract the mean from each X score to obtain a deviation score **4**
- Square each deviation score **5**
- Sum all squared deviation scores to obtain the sum of squares **6**
- Substitute numbers into the formula to obtain population variance, σ^2 **7**
- Take the square root of σ^2 to obtain the population standard deviation, σ **8**

B. DATA AND COMPUTATIONS

X	$X - \mu$	$(X - \mu)^2$
13	3	9
10	0	0
11	1	1
7	-3	9
9	-1	1
11	1	1
9	-1	1

$$\text{1 } N = 7$$

$$\text{2 } \Sigma X = 70$$

$$\text{6 } SS = \Sigma (X - \mu)^2 = 22$$

$$\text{3 } \mu = \frac{70}{7} = 10$$

$$\text{7 } \sigma^2 = \frac{SS}{N} = \frac{22}{7} = 3.14 \quad \text{8 } \sigma = \sqrt{\frac{SS}{N}} = \sqrt{\frac{22}{7}} = \sqrt{3.14} = 1.77$$

Sum of Squares Formulas for Sample

Sample notation can be substituted for population notation in the above two formulas without causing any essential changes:

SUM OF SQUARES (SS) FOR SAMPLE (DEFINITION FORMULA)

$$SS = \Sigma (X - \bar{X})^2 \quad (4.3)$$

(COMPUTATION FORMULA)

$$SS = \Sigma X^2 - \frac{(\Sigma X)^2}{n} \quad (4.4)$$

Table 4.2
CALCULATION OF POPULATION STANDARD DEVIATION (σ) (COMPUTATION FORMULA)

A. COMPUTATIONAL SEQUENCE

Assign a value to N representing the number of X scores **1**

Sum all X scores **2**

Square the sum of all X scores **3**

Square each X score **4**

Sum all squared X scores **5**

Substitute numbers into the formula to obtain the sum of squares, SS **6**

Substitute numbers into the formula to obtain the population variance, σ^2 **7**

Take the square root of σ^2 to obtain the population standard deviation, σ **8**

B. DATA AND COMPUTATIONS

X	X^2
13	169
10	100
11	121
7	49
9	81
11	121
9	81

1 $N = 7$

2 $\sum X = 70$

5 $\sum X^2 = 722$

3 $(\sum X)^2 = 4900$

6 $SS = \sum X^2 - \frac{(\sum X)^2}{N} = 722 - \frac{4900}{7} = 722 - 700 = 22$

7 $\sigma^2 = \frac{SS}{N} = \frac{22}{7} = 3.14$

8 $\sigma = \sqrt{\frac{SS}{N}} = \sqrt{\frac{22}{7}} = \sqrt{3.14} = 1.77$

Standard Deviation for Population σ

$$\text{variance} = \frac{\text{sum of all squared deviation scores}}{\text{number of scores}}$$

or, in symbols:

VARIANCE FOR POPULATION

$$\sigma^2 = \frac{SS}{N}$$

(4.5)

STANDARD DEVIATION FOR POPULATION

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{SS}{N}} \quad (4.6)$$

where σ represents the **population standard deviation**, $\sqrt{}$ instructs us to take the square root of the covered expression, and SS and N are defined above.

By referring to the last two steps in either Table 4.1 or 4.2, you can verify that the value of the variance, σ^2 , equals 3.14 for distribution C because

$$\sigma^2 = \frac{SS}{N} = \frac{22}{7} = 3.14$$

and that the value of the standard deviation, σ , equals 1.77 for distribution C because

$$\sigma = \sqrt{\frac{SS}{N}} = \sqrt{\frac{22}{7}} = \sqrt{3.14} = 1.77$$

Standard Deviation for Sample (s)

Although the sum of squares term remains essentially the same for both populations and samples, there is a small but important change in the formulas for the variance and standard deviation for samples. This change appears in the denominator of each formula where N , the population size, is replaced not by n , the sample size, but by $n - 1$, as shown:

VARIANCE FOR SAMPLE

$$s^2 = \frac{SS}{n-1} \quad (4.7)$$

STANDARD DEVIATION FOR SAMPLE

$$s = \sqrt{s^2} = \sqrt{\frac{SS}{n-1}} \quad (4.8)$$

where s^2 and s represent the sample variance and **sample standard deviation**, SS is the sample sum of squares as defined in either Formula 4.3 or 4.4, and n is the sample size.*

Progress Check * 4.4 Using the definition formula for the sum of squares, calculate the sample standard deviation for the following four scores: 1, 3, 4, 4.

Table 4.3
CALCULATION OF SAMPLE STANDARD DEVIATION (s)
(DEFINITION FORMULA)

A. COMPUTATION SEQUENCE

Assign a value to n 1 representing the number of X scores

Sum all X scores 2

Obtain the mean of these scores 3

Subtract the mean from each X score to obtain a deviation score 4

Square each deviation score 5

Sum all squared deviation scores to obtain the sum of squares 6

Substitute numbers into the formula to obtain the sample variance, s^2 7

Take the square root of s^2 to obtain the sample standard deviation, s 8

B. DATA AND COMPUTATIONS

X	$X - \bar{X}$	$(X - \bar{X})^2$
7	4	16
3	0	0
1	-2	4
0	-3	9
4	1	1

1 $n = 5$

2 $\Sigma X = 15$

6 $SS = \Sigma(X - \bar{X})^2 = 30$

3 $\bar{X} = \frac{15}{5} = 3$

7 $s^2 = \frac{SS}{n-1} = \frac{30}{4} = 7.50$ 8 $s = \sqrt{\frac{SS}{n-1}} = \sqrt{\frac{30}{4}} = \sqrt{7.50} = 2.74$

4.4 $s = \sqrt{\frac{(1-3)^2 + (3-3)^2 + (4-3)^2 + (4-3)^2}{4-1}} = \sqrt{\frac{6}{3}} = 1.41$

Table 4.4
CALCULATION OF SAMPLE STANDARD DEVIATION (s)
(COMPUTATION FORMULA)

A. COMPUTATIONAL SEQUENCE

Assign a value to n representing the number of X scores **1**

Sum all X scores **2**

Square the sum of all X scores **3**

Square each X score **4**

Sum all squared X scores **5**

Substitute numbers into the formula to obtain the sum of squares, SS **6**

Substitute numbers into the formula to obtain the sample variance, s^2 **7**

Take the square root of s^2 to obtain the sample standard deviation, s **8**

B. DATA AND COMPUTATIONS

X	$\frac{4}{X^2}$
7	49
3	9
1	1
0	0
4	16

$$\mathbf{1} \ n = 5 \quad \mathbf{2} \ \Sigma X = 15 \quad \mathbf{5} \ \Sigma X^2 = 75$$

$$\mathbf{3} \ (\Sigma X)^2 = 225$$

$$\mathbf{6} \ SS = \sum X^2 - \frac{(\sum X)^2}{n} = 75 - \frac{225}{5} = 75 - 45 = 30$$

$$\mathbf{7} \ s^2 = \frac{SS}{n-1} = \frac{30}{4} = 7.50 \quad \mathbf{8} \ s = \sqrt{\frac{SS}{n-1}} = \sqrt{\frac{30}{4}} = \sqrt{7.50} = 2.74$$

Progress Check * 4.5 Using the computation formula for the sum of squares, calculate the population standard deviation for the scores in (a) and the sample standard deviation for the scores in (b).

- (a) 1, 3, 7, 2, 0, 4, 7, 3 (b) 10, 8, 5, 0, 1, 1, 7, 9, 2

$$4.5 \text{ (a)} \ \sigma = \sqrt{\frac{137 - \frac{729}{8}}{8}} = \sqrt{5.73} = 2.39$$

$$\text{(b)} \ s = \sqrt{\frac{325 - \frac{1849}{9}}{9-1}} = \sqrt{14.95} = 3.87$$

Progress Check *4.6 Days absent from school for a *sample* of 10 first-grade children are: 8, 5, 7, 1, 4, 0, 5, 7, 2, 9.

- Before calculating the standard deviation, decide whether the definitional or computational formula would be more efficient. Why?
- Use the more efficient formula to calculate the sample standard deviation.

4.6 (a) computation formula since the mean is not a whole number.

$$(b) s = \sqrt{\frac{314 - \frac{2304}{10}}{10-1}} = \sqrt{9.28} = 3.05$$

If μ Is Known

For the sake of the present discussion, now assume that we know the value of the population mean, μ —let's say it equals 2. (Any value assigned to μ other than 3, the value of \bar{X} , would satisfy the current argument. It's reasonable to assume that the values of μ and \bar{X} will differ because a random sample exactly replicates its population rarely, if at all.) Furthermore, assume that we take a random sample of $n = 5$

Table 4.5
TWO ESTIMATES OF POPULATION VARIABILITY

WHEN μ IS UNKNOWN $(\bar{X} = 3)$			WHEN μ IS KNOWN $(\mu = 2)$		
X	$X - \bar{X}$	$(X - \bar{X})^2$	X	$X - \mu$	$(X - \mu)^2$
7	$7 - 3 = 4$	16	7	$7 - 2 = 5$	25
3	$3 - 3 = 0$	0	3	$3 - 2 = 1$	1
1	$1 - 3 = -2$	4	1	$1 - 2 = -1$	1
0	$0 - 3 = -3$	9	0	$0 - 2 = -2$	4
4	$4 - 3 = 1$	1	4	$4 - 2 = 2$	4
$\Sigma(X - \bar{X}) = 0$		$\Sigma(X - \bar{X})^2 = 30$	$\Sigma(X - \mu) = 5$		$\Sigma(X - \mu)^2 = 35$
$df = n - 1 = 5 - 1 = 4$			$df = n = 5$		
$s^2(df = n - 1) = \frac{\Sigma(X - \bar{X})^2}{n-1} = \frac{30}{4} = 7.50$			$s^2(df = n) = \frac{\Sigma(X - \mu)^2}{n} = \frac{35}{5} = 7.00$		

If μ Is Unknown

- It would be most efficient if, as above, we could use a random sample of deviations expressed around the population mean, $X - \mu$, to estimate variability in the population.

- But this is usually impossible because, in fact, the population mean is unknown.
- Therefore, we must substitute the known sample mean, \bar{X} , for the unknown population mean, μ , and we must use a random sample of n deviations expressed around their own sample mean, $\bar{X} - \bar{X}$, to estimate variability in the population.
- Although there are $n = 5$ deviations in the sample, only $n - 1 = 4$ of these deviations are free to vary because the sum of the $n = 5$ deviations from their own sample mean always equals zero.

DEGREES OF FREEDOM (df)

- Degrees of freedom (df) refers to the number of values that are free to vary, given one or more mathematical restrictions, in a sample being used to estimate a population characteristic.
- The concept of degrees of freedom is introduced only because we are using scores in a sample to estimate some unknown characteristic of the population.

VARIANCE FOR SAMPLE

$$s^2 = \frac{SS}{n-1} = \frac{SS}{df} \quad (4.9)$$

STANDARD DEVIATION FOR SAMPLE

$$s = \sqrt{\frac{SS}{n-1}} = \sqrt{\frac{SS}{df}} \quad (4.10)$$

where s^2 and s represent the sample variance and standard deviation, SS is the sum of squares as defined in either Formula 4.3 or 4.4, and df is the degrees of freedom and equals $n - 1$.

Progress Check *4.7 As a first step toward modifying his study habits, Phil keeps daily records of his study time.

- During the first two weeks, Phil's mean study time equals 20 hours per week. If he studied 22 hours during the first week, how many hours did he study during the second week?
- During the first four weeks, Phil's mean study time equals 21 hours. If he studied 22, 18, and 21 hours during the first, second, and third weeks, respectively, how many hours did he study during the fourth week?
- If the information in (a) and (b) is to be used to estimate some unknown population characteristic, the notion of degrees of freedom can be introduced. How many degrees of freedom are associated with (a) and (b)?
- Describe the mathematical restriction that causes a loss of degrees of freedom in (a) and (b).

- 4.7 (a) 18 hours
 (b) 23 hours
 (c) $df = 1$ in (a) and $df = 3$ in (b)
 (d) When all observations are expressed as deviations from their mean, the sum of all deviations must equal zero.**

INTERQUARTILE RANGE (IQR)

- The most important spinoff of the range, the interquartile range (IQR), is simply the range for the middle 50 percent of the scores.

Table 4.6
CALCULATION OF THE IQR

A. INSTRUCTIONS

- 1 Order scores from least to most.
- 2 To determine how far to penetrate the set of ordered scores, begin at either end, then add 1 to the total number of scores and divide by 4. If necessary, round the result to the nearest whole number.
- 3 Beginning with the largest score, count the requisite number of steps (calculated in step 2) into the ordered scores to find the location of the third quartile.
- 4 The third quartile equals the value of the score at this location.
- 5 Beginning with the smallest score, again count the requisite number of steps into the ordered scores to find the location of the first quartile.
- 6 The first quartile equals the value of the score at this location.
- 7 The IQR equals the third quartile minus the first quartile.

B. EXAMPLE

1 7, 9, 9, 10, 11, 11, 13

2 $(7 + 1)/4 = 2$

3 7, 9, 9, 10, 11, 11, 13



4 third quartile = 11

5 7, 9, 9, 10, 11, 11, 13



6 first quartile = 9

7 $IQR = 11 - 9 = 2$

Progress Check *4.8 Determine the values of the range and the IQR for the following sets of data.

(a) Retirement ages: 60, 63, 45, 63, 65, 70, 55, 63, 60, 65, 63

(b) Residence changes: 1, 3, 4, 1, 0, 2, 5, 8, 0, 2, 3, 4, 7, 11, 0, 2, 3, 4

4.8 (a) range = 25; $IQR = 65 - 60 = 5$

(b) range = 11; $IQR = 4 - 1 = 3$

MEASURES OF VARIABILITY FOR QUALITATIVE AND RANKED DATA

Qualitative Data

- Measures of variability are virtually nonexistent for qualitative or nominal data.
- It is probably adequate to note merely whether scores are evenly divided among the various classes, unevenly divided among the various classes, or concentrated mostly in one class.
- For example, if the ethnic composition of the residents of a city is about evenly divided among several groups, the variability with respect to ethnic groups is maximum; there is considerable heterogeneity.

Ordered Qualitative and Ranked Data

- If qualitative data can be ordered because measurement is ordinal then it's appropriate to describe variability by identifying extreme scores.
- For instance, the active membership of an officers' club might include no one with a rank below first lieutenant or above brigadier general.

VI. NORMAL DISTRIBUTIONS AND STANDARD (z) SCORE:

THE NORMAL CURVE

- A distribution based on 30,910 men usually is more accurate than one based on 3091, and a distribution based on 3,091,000 usually is even more accurate.
- But it is prohibitively expensive in both time and money to even survey 30,910 people. Fortunately, it is a fact that the distribution of heights for all American men—not just 3091 or even 3,091,000—approximates the normal curve, a well-documented theoretical curve.

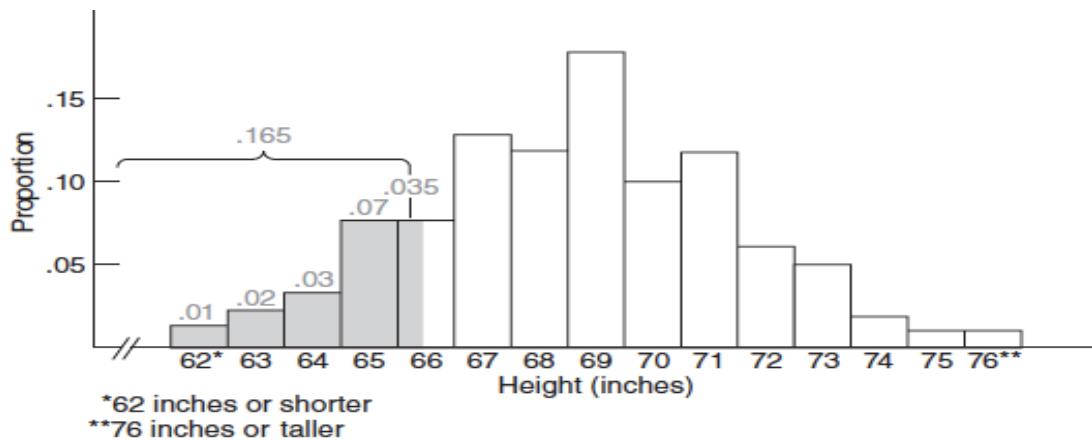


FIGURE 5.1

Relative frequency distribution for heights of 3091 men.

Source: National Center for Health Statistics, 1960–62, Series 11, No. 14. Mean updated by authors.

In Figure 5.2, the idealized normal curve has been superimposed on the original distribution for 3091 men.

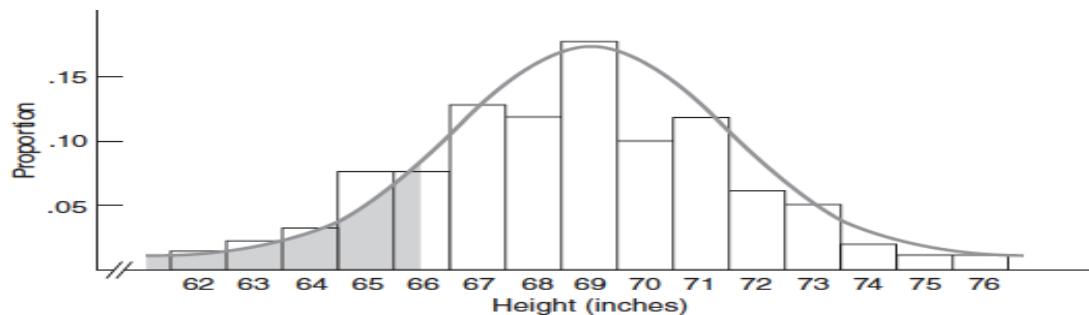


FIGURE 5.2

Normal curve superimposed on the distribution of heights.

Interpreting the Shaded Area

- The total area under the normal curve in Figure 5.2 can be identified with all FBI applicants.
- Viewed relative to the total area, the shaded area represents the proportion of applicants who will be eligible because they are shorter than exactly 66 inches.

Finding a Proportion for the Shaded Area

- To find this new proportion, we cannot rely on the vertical scale in Figure 5.2, because it describes as proportions the areas in the rectangular bars of histograms, not the areas in the various curved sectors of the normal curve.

Properties of the Normal Curve

- A normal curve is a theoretical curve defined for a continuous variable, as described in Section 1.6, and noted for its symmetrical bell-shaped form.
- Because the normal curve is symmetrical, its lower half is the mirror image of its upper half.
- Being bell shaped, the normal curve peaks above a point midway along the horizontal spread and then tapers off gradually in either direction from the peak
- The values of the mean, median (or 50th percentile), and mode, located at a point midway along the horizontal spread, are the same for the normal curve.

Importance of Mean and Standard Deviation

- When you're using the normal curve, two bits of information are indispensable: values for the mean and the standard deviation.

Different Normal Curves

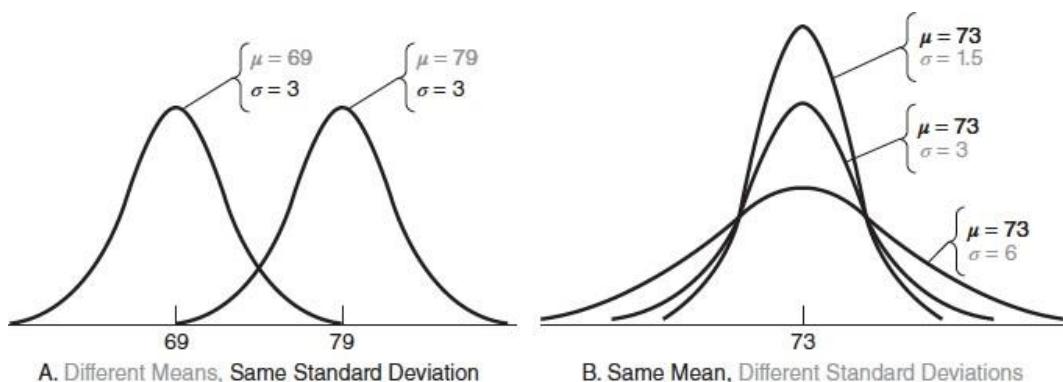


FIGURE 5.3

Different normal curves.

- Every normal curve can be interpreted in exactly the same way once any distance from the mean is expressed in standard deviation units.
- For example, .68, or 68 percent of the total area under a normal curve—any normal curve—is within one standard deviation above and below the mean, and only .05, or 5 percent, of the total area is more than two standard deviations above and below the mean.

z SCORES

- A z score is a unit-free, standardized score that, regardless of the original units of measurement, indicates how many standard deviations a score is above or below the mean of its distribution.
- To obtain a z score, express any original score, whether measured in inches, milliseconds, dollars, IQ points, etc., as a deviation from its mean
- where X is the original score and μ and σ are the mean and the standard deviation, respectively, for the normal distribution of the original scores.
- Since identical units of measurement appear in both the numerator and denominator of the ratio for z , the original units of measurement cancel each other and the z score emerges as a unit-free or standardized number, often referred to as a standard score.

A z score consists of two parts:

1. a positive or negative sign indicating whether it's above or below the mean; and
2. a number indicating the size of its deviation from the mean in standard deviation units.

- A z score of 2.00 always signifies that the original score is exactly two standard deviations above its mean.
- Similarly, a z score of -1.27 signifies that the original score is exactly 1.27 standard deviations below its mean.
- A z score of 0 signifies that the original score coincides with the mean.

z SCORE

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

(5.1)

Converting to z Scores

- To answer the question about eligible FBI applicants, replace X with 66 (the maximum permissible height), μ with 69 (the mean height), and σ with 3 (the standard deviation of heights) and solve for z as follows:

$$\frac{66 - 69}{3} = \frac{-3}{3} = -1$$

Progress Check *5.1 Express each of the following scores as a z score:

- (a) Margaret's IQ of 135, given a mean of 100 and a standard deviation of 15
- (b) a score of 470 on the SAT math test, given a mean of 500 and a standard deviation of 100
- (c) a daily production of 2100 loaves of bread by a bakery, given a mean of 2180 and a standard deviation of 50
- (d) Sam's height of 69 inches, given a mean of 69 and a standard deviation of 3
- (e) a thermometer-reading error of -3 degrees, given a mean of 0 degrees and a standard deviation of 2 degrees

5.1 (a) 2.33 (d) 0.00
 (b) -0.30 (e) -1.50
 (c) -1.60

STANDARD NORMAL CURVE

- If the original distribution approximates a normal curve, then the shift to standard or z scores will always produce a new distribution that approximates the standard normal curve.
- The standard normal curve always has a mean of 0 and a standard deviation of 1. However, to verify that the mean of a standard normal distribution equals 0, replace X in the z score formula with μ , the mean of any normal distribution, and then solve for z :

$$\text{Mean of } z = \frac{X - \mu}{\sigma} = \frac{\mu - \mu}{\sigma} = \frac{0}{\sigma} = 0$$

- Likewise, to verify that the standard deviation of the standard normal distribution equals 1, replace X in the z score formula with $\mu + 1\sigma$, the value corresponding to one standard deviation above the mean for any (nonstandard) normal distribution, and then solve for z :

$$\text{Standard deviation of } z = \frac{X - \mu}{\sigma} = \frac{\mu + 1\sigma - \mu}{\sigma} = \frac{1\sigma}{\sigma} = 1$$

- Although there is an infinite number of different normal curves, each with its own mean and standard deviation, there is only one standard normal curve, with a mean of 0 and a standard deviation of 1.

Figure 5.4 illustrates the emergence of the standard normal curve from three different normal curves: that for the men's heights, with a mean of 69 inches and a standard deviation of 3 inches; that for the useful lives of 100-watt electric light bulbs, with a mean of 1200 hours and a standard deviation of 120 hours; and that for the IQ scores of fourth graders, with a mean of 105 points and a standard deviation of 15 points.

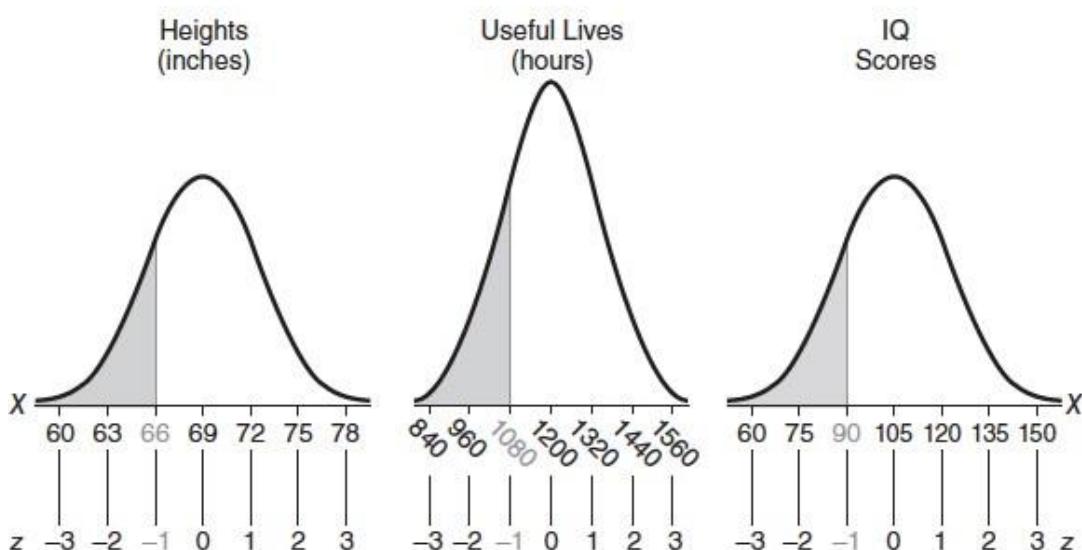


FIGURE 5.4
Converting three normal curves to the standard normal curve.

- Converting all original observations into z scores leaves the normal shape intact but not the units of measurement.
- Shaded observations of 66 inches, 1080 hours, and 90 IQ points all reappear as a z score of -1.00 .

Standard Normal Table

- Essentially, the standard normal table consists of columns of z scores coordinated with columns of proportions.

Using the Top Legend of the Table

Table 5.1 shows an abbreviated version of the standard normal curve, while Table A in Appendix C on page 458 shows a more complete version of the same curve. Notice that columns are arranged in sets of three, designated as A, B, and C in the legend at the top of the table. When using the top legend, all entries refer to the upper half of the standard normal curve. The entries in column A are z scores, beginning with 0.00 and ending (in the full-length table of Appendix C) with 4.00. Given a z score of zero or more, columns B and C indicate how the z score splits the area in the upper half of the normal curve. As suggested by the shading in the top legend, column B indicates the proportion of area between the mean and the z score, and column C indicates the proportion of area beyond the z score, in the upper tail of the standard normal curve.

Using the Bottom Legend of the Table

Because of the symmetry of the normal curve, the entries in Table 5.1 and Table A of Appendix C also can refer to the lower half of the normal curve. Now the columns are designated as A', B', and C' in the legend at the bottom of the table. When using the bottom legend, all entries refer to the lower half of the standard normal curve.

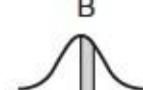
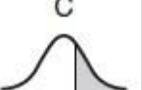
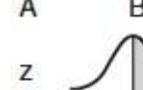
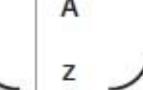
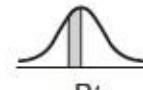
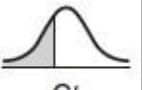
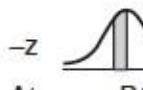
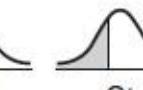
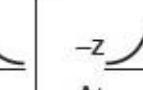
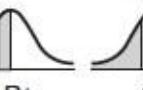
Imagine that the nonzero entries in column A' are negative z scores, beginning with -0.01 and ending (in the full-length table of Appendix C) with -4.00. Given a negative z score, columns B' and C' indicate how that z score splits the lower half of the normal curve. As suggested by the shading in the bottom legend of the table, column B' indicates the proportion of area between the mean and the negative z score, and column C' indicates the proportion of area beyond the negative z score, in the lower tail of the standard normal curve.

Progress Check *5.2 Using Table A in Appendix C, find the proportion of the total area identified with the following statements:

- (a) above a z score of 1.80
- (b) between the mean and a z score of -0.43
- (c) below a z score of -3.00
- (d) between the mean and a z score of 1.65
- (e) between z scores of 0 and -1.96

- 5.2** (a) .0359 (d) .4505
(b) .1664 (e) .4750
(c) .0013

TABLE 5.1
PROPORTIONS (OF AREAS) UNDER THE STANDARD NORMAL CURVE
FOR VALUES OF z (FROM TABLE A OF APPENDIX C)

A	B	C	A	B	C	A	B	C
z			z			z		
0.00	.0000	.5000	0.40	.1554	.3446	0.80	.2881	.2119
0.01	.0040	.4960	0.41	.1591	.3409	0.81	.2910	.2090
•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	0.99	.3389	.1611
•	•	•	•	•	•	1.00	.3413	.1587
•	•	•	•	•	•	1.01	.3438	.1562
•	•	•	•	•	•	•	•	•
0.38	.1480	.3520	0.78	.2823	.2711	1.18	.3810	.1190
0.39	.1517	.3483	0.79	.2852	.2148	1.19	.3830	.1170
$-z$			$-z$			$-z$		
A'	B'	C'	A'	B'	C'	A'	B'	C'

SOLVING NORMAL CURVE PROBLEMS

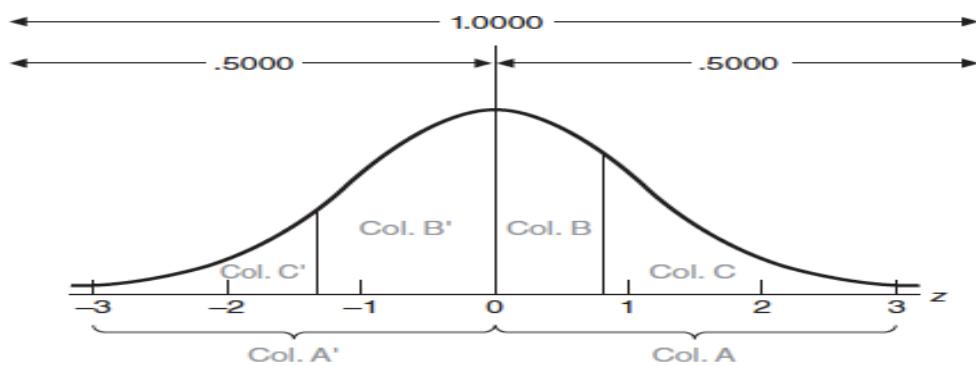


FIGURE 5.5
Interpretation of Table A, Appendix C.

FINDING PROPORTIONS

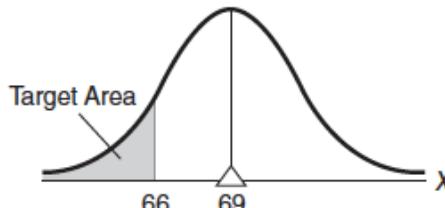
Example: Finding Proportions for One Score

Now we'll use a step-by-step procedure, adopted throughout this chapter, to find the proportion of all FBI applicants who are shorter than exactly 66 inches, given that the distribution of heights approximates a normal curve with a mean of 69 inches and a standard deviation of 3 inches.

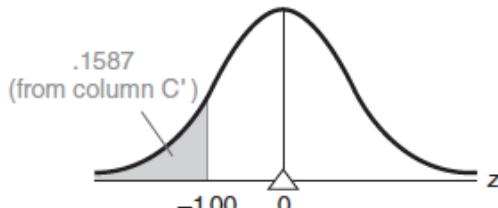
1. Sketch a normal curve and shade in the target area, as in the left part of **Figure 5.6**. Being less than the mean of 69, 66 is located to the left of the mean. Furthermore, since the unknown proportion represents those applicants who are shorter than 66 inches, the shaded target sector is located to the left of 66.
2. Plan your solution according to the normal table. Decide precisely how you will find the value of the target area. In the present case, the answer will be obtained from column C' of the standard normal table, since the target area coincides with the type of area identified with column C', that is, the area in the lower tail beyond a negative z .
3. Convert X to z . Express 66 as a z score:

$$z = \frac{X - \mu}{\sigma} = \frac{66 - 69}{3} = \frac{-3}{3} = -1$$

Find: Proportion Below 66



Solution:



Answer: .1587

FIGURE 5.6
Finding proportions.

4. Find the target area. Refer to the standard normal table, using the bottom legend, as the z score is negative. The arrows in Table 5.1 show how to read the table. Look up column A' to 1.00 (representing a z score of -1.00), and note the corresponding proportion of .1587 in column C': This is the answer, as suggested in the right part of Figure 5.6. It can be concluded that only .1587 (or .16) of all of the FBI applicants will be shorter than 66 inches.

Progress Check *5.3 Assume that GRE scores approximate a normal curve with a mean of 500 and a standard deviation of 100.

(a) Sketch a normal curve and shade in the target area described by each of the following statements:

- (i) less than 400
- (ii) more than 650
- (iii) less than 700

(b) Plan solutions (in terms of columns B, C, B', or C' of the standard normal table, as well as the fact that the proportion for either the entire upper half or lower half always equals .5000) for the target areas in part (a).

(c) Convert to z scores and find the proportions that correspond to the target areas in part (a).

5.3 (a ₁)		(b ₁) C'	(c ₁) z = -1.00 answer = .1587
(a ₂)		(b ₂) C	(c ₂) z = 1.50 answer = .0668
(a ₃)		(b ₃) .5000 + B	(c ₃) z = 2.00 answer = .5000 + .4772 = .9772

Example: Finding Proportions between Two Scores

Assume that, when not interrupted artificially, the gestation periods for human fetuses approximate a normal curve with a mean of 270 days (9 months) and a standard deviation of 15 days. What proportion of gestation periods will be between 245 and 255 days?

- 1. Sketch a normal curve and shade in the target area**, as in the top panel of **Figure 5.7**. Satisfy yourself that, in fact, the shaded area represents just those gestation periods between 245 and 255 days.
- 2. Plan your solution according to the normal table**. This type of problem requires more effort to solve because the value of the target area cannot be read directly from Table A. As suggested in the bottom two panels of **Figure 5.7**, the basic idea is to identify the target area with the difference between two overlapping areas whose values can be read from column C' of Table A. The larger area (less than 255 days) contains two sectors: the target area (between 245 and 255 days) and a remainder (less than 245 days). The smaller area contains only the remainder (less than 245 days). Subtracting the smaller area (less than 245 days) from the larger area (less than 255 days), therefore, eliminates the common remainder (less than 245 days), leaving only the target area (between 245 and 255 days).

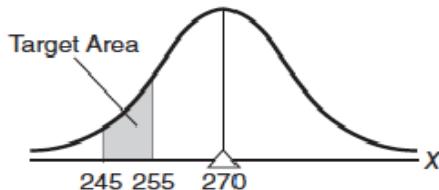
- 3. Convert X to z by expressing 255 as**

$$z = \frac{255 - 270}{15} = \frac{-15}{15} = -1.00$$

and by expressing 245 as

$$z = \frac{245 - 270}{15} = \frac{-25}{15} = -1.67$$

Find: Proportion Between 245 and 255



Solution:

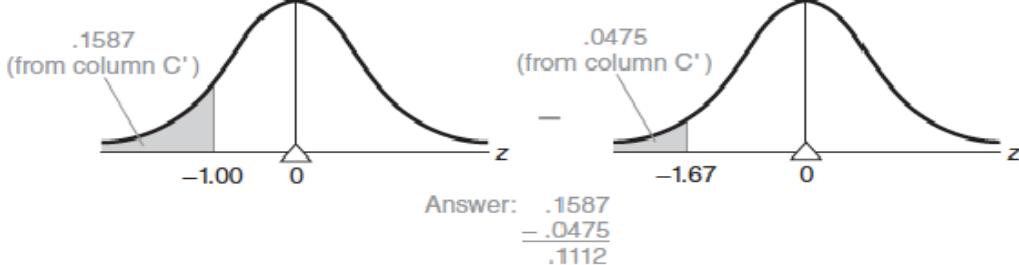


FIGURE 5.7
Finding proportions.

- 4. Find the target area.** Look up column A' to a negative z score of -1.00 (remember, you must imagine the negative sign), and note the corresponding proportion of $.1587$ in column C'. Likewise, look up column A' to a z score of -1.67 , and note the corresponding proportion of $.0475$ in column C'. Subtract the smaller proportion from the larger proportion to obtain the answer, $.1112$. Thus, only $.11$, or 11 percent, of all gestation periods will be between 245 and 255 days.

Finding Proportions beyond Two Scores

Assume that high school students' IQ scores approximate a normal distribution with a mean of 105 and a standard deviation of 15. What proportion of IQs are more than 30 points either above or below the mean?

1. Sketch a normal curve and shade in the two target areas, as in the top panel of **Figure 5.8**.
2. Plan your solution according to the normal table. The solution to this type of problem is straightforward because each of the target areas can be read directly from Table A. The target area in the tail to the right can be obtained from column C, and that in the tail to the left can be obtained from column C', as shown in the bottom two panels of Figure 5.8.

Find: Proportion Beyond 30 Points from Mean

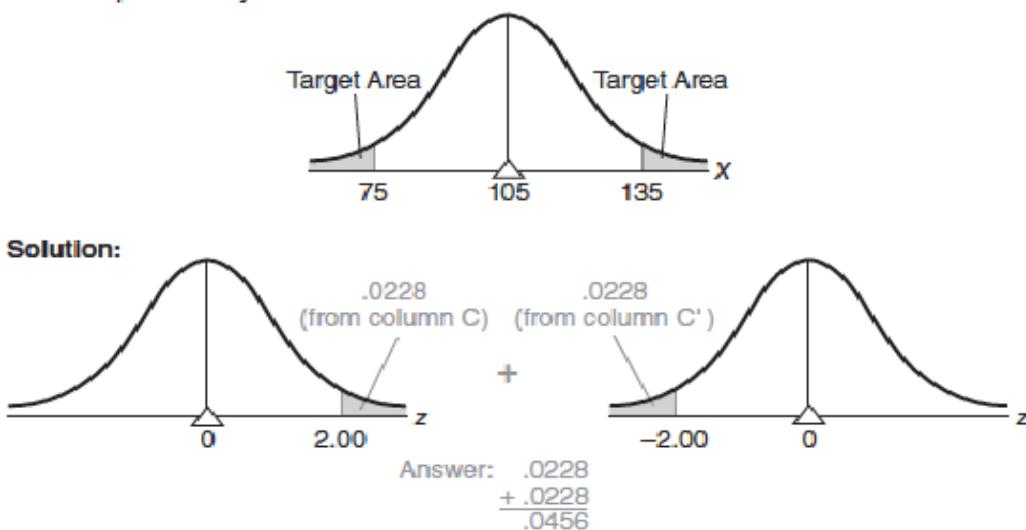


FIGURE 5.8
Finding proportions.

3. Convert X to z by expressing IQ scores of 135 and 75 as

$$z = \frac{135 - 105}{15} = \frac{30}{15} = 2.00$$

$$z = \frac{75 - 105}{15} = \frac{-30}{15} = -2.00$$

4. Find the target area. In Table A, locate a z score of 2.00 in column A, and note the corresponding proportion of .0228 in column C. Because of the symmetry of the normal curve, you need not enter the table again to find the proportion below a z score of -2.00. Instead, merely double the above proportion of .0228 to obtain .0456, which represents the proportion of students with IQs more than 30 points either above or below the mean.

Progress Check *5.5 Assume that SAT math scores approximate a normal curve with a mean of 500 and a standard deviation of 100.

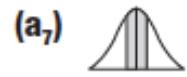
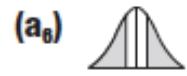
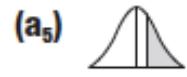
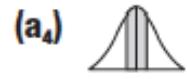
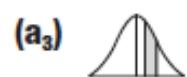
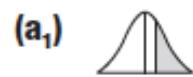
(a) Sketch a normal curve and shade in the target area(s) described by each of the following statements:

- (i) more than 570
- (ii) less than 515
- (iii) between 520 and 540
- (iv) between 470 and 520
- (v) more than 50 points above the mean
- (vi) more than 100 points either above or below the mean
- (vii) within 50 points either above or below the mean

(b) Plan solutions (in terms of columns B, C, B', and C') for the target areas in part (a).

(c) Convert to z scores and find the target areas in part (a).

5.5



(a₁)



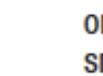
(a₂)



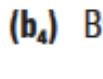
(a₃)



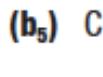
(a₄)



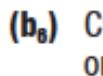
(a₅)



(a₆)



(a₇)



(b₁)



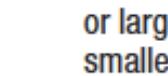
(b₂)



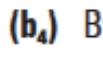
(b₃)



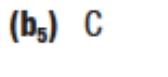
(b₄)



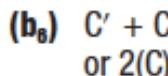
(b₅)



(b₆)



(b₇)



(c₁)

$$z = 0.70 \\ .2420$$

(c₂)

$$z = 0.15 \\ .5000 + .0596 = .5596$$

(c₃)

$$z = 0.20; z = 0.40 \\ .1554 - .0793 = .0761$$

(c₄)

$$z = -0.30; z = 0.20 \\ .1179 + .0793 = .1972$$

(c₅)

$$z = 0.50 \\ .3085$$

(c₆)

$$z = -1.00; z = 1.00 \\ .1587 + .1587 = .3174$$

(c₇)

$$z = -0.50; z = 0.50 \\ .1915 + .1915 = .3830$$

FINDING SCORES

Example: Finding One Score

Exam scores for a large psychology class approximate a normal curve with a mean of 230 and a standard deviation of 50. Furthermore, students are graded "on a curve," with only the upper 20 percent being awarded grades of A. What is the lowest score on the exam that receives an A?

1. Sketch a normal curve and, on the correct side of the mean, draw a line representing the target score, as in **Figure 5.9**. This is often the most difficult step, and it involves semantics rather than statistics. It's often helpful to visualize the target

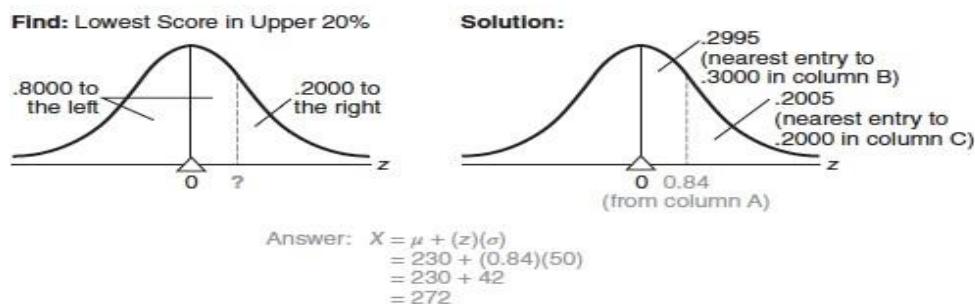


FIGURE 5.9
Finding scores.

score as splitting the total area into two sectors—one to the left of (below) the target score and one to the right of (above) the target score. For example, in the present case, the target score is the point along the base of the curve that splits the total area into 80 percent, or .8000 to the left, and 20 percent, or .2000 to the right. The mean of a normal curve serves as a valuable frame of reference since it always splits the total area into two equal halves—.5000 to the left of the mean and .5000 to the right of the mean. Since more than .5000—that is, .8000—of the total area is to the left of the target score, this score must be on the upper or right side of the mean. On the other hand, if less than .5000 of the total area had been to the left of the target score, this score would have been placed on the lower or left side of the mean.

2. **Plan your solution according to the normal table.** In problems of this type, you must plan how to find the z score for the target score. Because the target score is on the right side of the mean, concentrate on the area in the upper half of the normal curve, as described in columns B and C. The right panel of Figure 5.9 indicates that either column B or C can be used to locate a z score in column A. It is crucial, however, to search for the single value (.3000) that is valid for column B or the single value (.2000) that is valid for column C. Note that we look in column B for .3000, not for .8000. Table A is not designed for sectors, such as the lower .8000, that span the mean of the normal curve.
3. **Find z .** Refer to Table A. Scan column C to find .2000. If this value does not appear in column C, as typically will be the case, approximate the desired value (and the correct score) by locating the entry in column C nearest to .2000. If adjacent entries are equally close to the target value, use either entry—it is your choice. As shown in the right panel of Figure 5.9, the entry in column C closest to .2000 is .2005, and the corresponding z score in column A equals 0.84. Verify this by checking Table A. Also note that exactly the same z score of 0.84 would have been identified if column B had been searched to find the entry (.2995) nearest to .3000. The z score of 0.84 represents the point that separates the upper 20 percent of the area from the rest of the area under the normal curve.
4. **Convert z to the target score.** Finally, convert the z score of 0.84 into an exam score, given a distribution with a mean of 230 and a standard deviation of 50. You'll recall that a z score indicates how many standard deviations the original score is above or below its mean. In the present case, the target score must be located .84 of a standard deviation above its mean. The distance of the target score above its mean equals 42 (from $.84 \times 50$), which, when added to the mean of 230, yields a value of 272. Therefore, 272 is the lowest score on the exam that receives an A.

When converting z scores to original scores, you will probably find it more efficient to use the following equation (derived from the z score equation on page 86):

CONVERTING z SCORE TO ORIGINAL SCORE

$$X = \mu + (z)(\sigma) \quad (5.2)$$

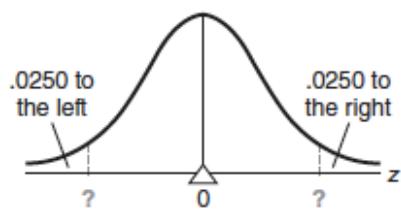
where X is the target score, expressed in original units of measurement; μ and σ are the mean and the standard deviation, respectively, for the original normal curve; and z is the standard score read from column A or A' of Table A. When appropriate numerical substitutions are made, as shown in the bottom of Figure 5.9, 272 is found to be the answer, in agreement with our earlier conclusion.

Example: Finding Two Scores

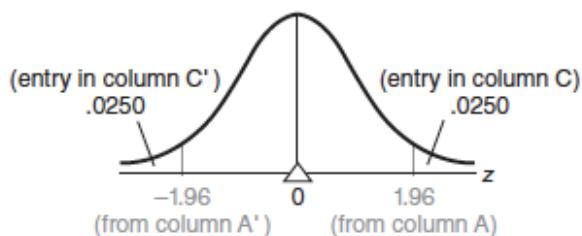
Assume that the annual rainfall in the San Francisco area approximates a normal curve with a mean of 22 inches and a standard deviation of 4 inches. What are the rainfalls for the more atypical years, defined as the driest 2.5 percent of all years and the wettest 2.5 percent of all years?

1. Sketch a normal curve. On either side of the mean, draw two lines representing the two target scores, as in **Figure 5.10**. The smaller (driest) target score splits the total area into .0250 to the left and .9750 to the right, and the larger (wettest) target score does the exact opposite.
2. Plan your solution according to the normal table. Because the smaller target score is located on the lower or left side of the mean, we will concentrate on the area in the lower half of the normal curve, as described in columns B' and C'. The target z score can be found by scanning either column B' for .4750 or column C'

Find: Pairs of Scores for the Extreme 2.5%



Solution:



$$\begin{aligned} \text{Answer: } X_{\min} &= \mu + (z)(\sigma) \\ &= 22 + (-1.96)(4) \\ &= 22 - 7.84 \\ &= 14.16 \end{aligned}$$

$$\begin{aligned} \text{Answer: } X_{\max} &= \mu + (z)(\sigma) \\ &= 22 + (1.96)(4) \\ &= 22 + 7.84 \\ &= 29.84 \end{aligned}$$

FIGURE 5.10

Finding scores.

for .0250. After finding the smaller target score, we will capitalize on the symmetrical properties of normal curves to find the value of the larger target score.

3. Find z . Referring to Table A, we can scan column B' for .4750, or the entry nearest to .4750. In this case, .4750 appears in column B', and the corresponding z score in column A' equals -1.96. The same z score of -1.96 would have been obtained if column C' had been searched for a value of .0250.
4. Convert z to the target score. When the appropriate numbers are substituted in Formula 5.2, as shown in the bottom panel of Figure 5.10, the smaller target score equals 14.16 inches, the amount of annual rainfall that separates the driest 2.5 percent of all years from all of the other years.

The location of the larger target score is the mirror image of that for the smaller target score. Therefore, we need not even consult Table A to establish that its z score equals 1.96—that is, the same value as the smaller target score, but without the negative sign. When 1.96 is converted to inches of rainfall, as shown in the bottom of Figure 5.10, the larger target equals 29.84 inches, the amount of annual rainfall that separates the wettest 2.5 percent of all years from all other years.

Progress Check *5.6 Assume that the burning times of electric light bulbs approximate a normal curve with a mean of 1200 hours and a standard deviation of 120 hours. If a large number of new lights are installed at the same time (possibly along a newly opened freeway), at what time will

- (a) 1 percent fail? (Hint: This splits the total area into .0100 to the left and .9900 to the right.)
- (b) 50 percent fail?
- (c) 95 percent fail?
- (d) If a new inspection procedure eliminates the weakest 8 percent of all lights before they are marketed, the manufacturer can safely offer customers a money-back guarantee on all lights that fail before _____ hours of burning time.

5.6 (a) $1200 + (-2.33)(120) = 920.40$
(b) $1200 + (0.00)(120) = 1200$
(c) $1200 + (1.65)(120) = 1398$
or $1200 + (1.64)(120) = 1396.80$
(d) $1200 + (-1.41)(120) = 1030.80$

DOING NORMAL CURVE PROBLEMS

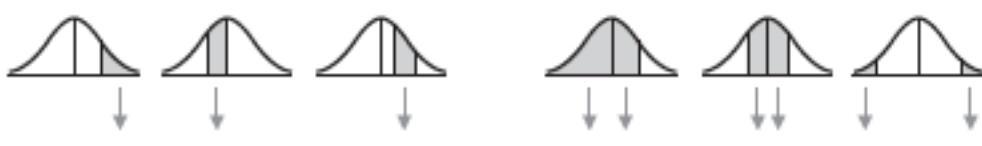
Read the problem carefully to determine whether a proportion or a score is to be found.

FINDING PROPORTIONS

1. Sketch the normal curve and shade in the target area.

Examples: One Area

Two Areas



2. Plan the solution in terms of the normal table.



3. Convert X to z : $$z = \frac{X - \mu}{\sigma}$$

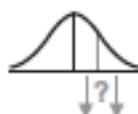
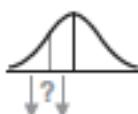
4. Find the target area by entering either column A or A' with z , and noting the corresponding proportion from column B, C, B', or C'.

FINDING SCORES

1. Sketch the normal curve and, on the correct side of the mean, draw a line representing the target score.

Examples: To Left of Mean
(area to left less than .5000)

To Right of Mean
(area to left more than .5000)



2. Plan the solution in terms of the normal table.

↓
↓
C' or B'

↓
↓
B or C

3. Find z by locating the entry nearest to that desired in column B, C, B', or C' and reading out the corresponding z score.

↓
↓
-z

↓
↓
z

4. Convert z to the target score: $$X = \mu + (z)(\sigma)$$

z Scores for Non-normal Distributions

- z scores are not limited to normal distributions. Non-normal distributions also can be transformed into sets of unit-free, standardized z scores.
- In this case, the standard normal table cannot be consulted, since the shape of the distribution of z scores is the same as that for the original non-normal distribution.
- For instance, if the original distribution is positively skewed, the distribution of z scores also will be positively skewed.
- Regardless of the shape of the distribution, the shift to z scores always produces a distribution of standard scores with a mean of 0 and a standard deviation of 1.

Interpreting Test Scores

- The use of z scores can help you identify a person's relative strengths and weaknesses on several different tests.

Importance of Reference Group

- Remember that z scores reflect performance relative to some group rather than an absolute standard.
- A meaningful interpretation of z scores requires, therefore, that the nature of the reference group be specified.

Progress Check *5.7 Convert each of the following test scores to z scores:

	TEST SCORE	MEAN	STANDARD DEVIATION
(a)	53	50	9
(b)	38	40	10
(c)	45	30	20
(d)	28	20	20

**Table 5.2
SHARON'S ACHIEVEMENT TEST SCORES**

SUBJECT	RAW SCORE	MEAN	STANDARD DEVIATION	z SCORE
Math	159	141	10	1.80
English	83	75	16	0.50
Psych	23	27	6	-0.67

**5.7 (a) 0.33 (c) 0.75
(b) -0.20 (d) 0.40**

Progress Check *5.8

- (a) Referring to Question 5.7, which one test score would you prefer?
- (b) Referring to Question 5.7, if Carson had earned a score of 64 on some test, which of the four distributions (a, b, c, or d) would have permitted the most favorable interpretation of this score?

Answers on page 427.

- 5.8 (a) A test score of 45 from distribution c because it converts to the largest z score (0.75).
- (b) Distribution b, because it yields a larger z score (2.40) than any other distribution.

Standard Score

- Whenever any unit-free scores are expressed relative to a known mean and a known standard deviation, they are referred to as standard scores.
- Although z scores qualify as standard scores because they are unit-free and expressed relative to a known mean of 0 and a known standard deviation of 1, other scores also qualify as standard scores.

Transformed Standard Scores

- Being by far the most important standard score, z scores are often viewed as synonymous with standard scores.
- For convenience, particularly when reporting test results to a wide audience, z scores can be changed to transformed standard scores, other types of unit-free standard scores that lack negative signs and decimal points.
- These transformations change neither the shape of the original distribution nor the relative standing of any test score within the distribution.

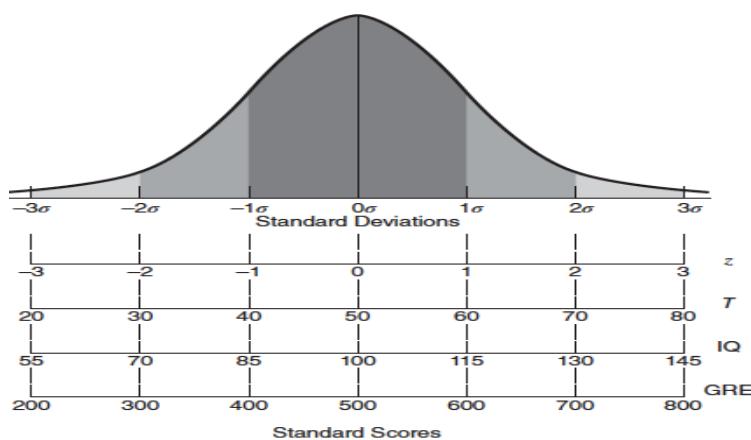


FIGURE 5.11
Common transformed standard scores associated with normal curves.

Figure 5.11 shows the values of some of the more common types of transformed standard scores relative to the various portions of the area under the normal curve.

Converting to Transformed Standard Scores

Use the following formula to convert any original standard score, z , into a transformed standard score, z' , having a distribution with any desired mean and standard deviation.

TRANSFORMED STANDARD SCORE

$$z' = \text{desired mean} + (z) (\text{desired standard deviation}) \quad (5.3)$$

where z' (called z prime) is the transformed standard score and z is the original standard score.

For instance, if you wish to convert a z score of -1.50 into a new distribution of z' scores for which the desired mean equals 500 and the desired standard deviation equals 100 , substitute these numbers into Formula 5.3 to obtain

$$\begin{aligned} z' &= 500 + (-1.50)(100) \\ &= 500 - 150 \\ &= 350 \end{aligned}$$

Progress Check *5.9 Assume that each of the raw scores listed originates from a distribution with the specified mean and standard deviation. After converting each raw score into a z score, transform each z score into a series of new standard scores with means and standard deviations of 50 and 10 , 100 and 15 , and 500 and 100 , respectively. (In practice, you would transform a particular z into only one new standard score.)

	RAW SCORE	MEAN	STANDARD DEVIATION
(a)	24	20	5
(b)	37	42	3

5.9

	$\mu = 0;$ $\sigma = 1$	$\mu = 50;$ $\sigma = 10$	$\mu = 100;$ $\sigma = 15$	$\mu = 500;$ $\sigma = 100$
(a)	0.80	58	112	580
(b)	-1.67	33.3	74.95	333

UNIT III

DESCRIBING RELATIONSHIPS

Correlation –Scatter plots –correlation coefficient for quantitative data –computational formula for correlation coefficient – Regression –regression line –least squares regression line – Standard error of estimate – interpretation of r^2 –multiple regression equations –regression towards the mean

I. Correlation:

- An investigator suspects that a relationship exists between the number of greeting cards sent and the number of greeting cards received by individuals.
- The investigator obtains the estimates for the most recent holiday season from five friends, as shown in Table 6.1.
- The data in Table 6.1 represent a very simple observational study with two dependent variables.

Table 6.1 GREETING CARDS SENT AND RECEIVED BY FIVE FRIENDS		
FRIEND	NUMBER OF CARDS	
	SENT	RECEIVED
Andrea	5	10
Mike	7	12
Doris	13	14
Steve	9	18
John	1	6

AN INTUITIVE APPROACH

- A tendency for pairs of scores to occupy similar relative positions in their respective distributions.

Positive Relationship

- Trends among pairs of scores can be detected most easily by constructing a list of paired scores in which the scores along one variable are arranged from largest to smallest.
- In panel A of Table 6.2, the five pairs of scores are arranged from the largest (13) to the smallest (1) number of cards sent.
- This table reveals a pronounced tendency for pairs of scores to occupy similar relative positions in their respective distributions.
- For example, John sent relatively few cards (1) and received relatively few cards (6), whereas Doris sent relatively many cards (13) and received relatively many cards (14).
- Therefore, that the two variables are related.
- Insofar as relatively low values are paired with relatively low values, and relatively high values are paired with relatively high values, the relationship is positive.

Negative Relationship

- Although John sent relatively few cards (1), he received relatively many (18).
- From this pattern, we can conclude that the two variables are related.
- This relationship implies that “You get the opposite of what you give.”
- Insofar as relatively low values are paired with relatively high values, and relatively high values are paired with relatively low values, the relationship is negative.

Little or No Relationship

- No regularity is apparent among the pairs of scores in panel C.
- For instance, although both Andrea and John sent relatively few cards (5 and 1, respectively), Andrea received relatively few cards (6) and John received relatively many cards (14).
- We can conclude that little, if any, relationship exists between the two variables.

Table 6.2 THREE TYPES OF RELATIONSHIPS		
A. POSITIVE RELATIONSHIP		
FRIEND SENT RECEIVED		
Doris	13	14
Steve	9	18
Mike	7	12
Andrea	5	10
John	1	6
B. NEGATIVE RELATIONSHIP		
FRIEND SENT RECEIVED		
Doris	13	6
Steve	9	10
Mike	7	14
Andrea	5	12
John	1	18
C. LITTLE OR NO RELATIONSHIP		
FRIEND SENT RECEIVED		
Doris	13	10
Steve	9	18
Mike	7	12
Andrea	5	6
John	1	14

- Two variables are positively related if pairs of scores tend to occupy similar relative positions (high with high and low with low) in their respective distributions.
- They are negatively related if pairs of scores tend to occupy dissimilar relative positions (high with low and vice versa) in their respective distributions.

Progress Check *6.1 Indicate whether the following statements suggest a positive or negative relationship:

- (a) More densely populated areas have higher crime rates.
- (b) Schoolchildren who often watch TV perform more poorly on academic achievement tests.
- (c) Heavier automobiles yield poorer gas mileage.
- (d) Better-educated people have higher incomes.
- (e) More anxious people voluntarily spend more time performing a simple repetitive task.

- 6.1**
- (a) Positive. The crime rate is higher, square mile by square mile, in densely populated cities than in sparsely populated rural areas.
 - (b) Negative. As TV viewing increases, performance on academic achievement tests tends to decline.
 - (c) Negative. Increases in car weight are accompanied by decreases in miles per gallon.
 - (d) Positive. Increases in educational level—grade school, high school, college—tend to be associated with increases in income.
 - (e) Positive. Highly anxious people willingly spend more time performing a simple repetitive task than do less anxious people.

II. SCATTERPLOTS

- A scatterplot is a graph containing a cluster of dots that represents all pairs of scores.

Construction

- To construct a scatterplot, as in Figure 6.1, scale each of the two variables along the horizontal (X) and vertical (Y) axes, and use each pair of scores to locate a dot within the scatterplot.
- For example, the pair of numbers for Mike, 7 and 12, define points along the X and Y axes, respectively.
- Using these points to anchor lines perpendicular to each axis, locate Mike's dot where the two lines intersect.

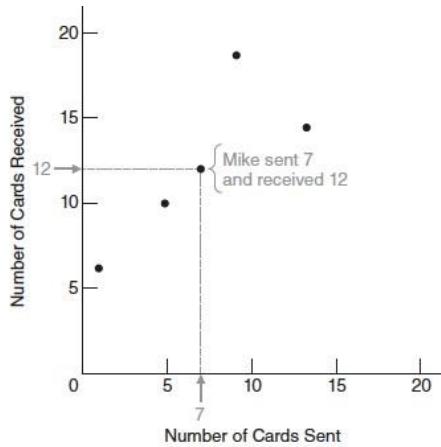


FIGURE 6.1
Scatterplot for greeting card exchange.

Positive, Negative, or Little or No Relationship?

- A dot cluster that has a slope from the lower left to the upper right, as in panel A of Figure 6.2, reflects a positive relationship.
- Small values of one variable are paired with small values of the other variable, and large values are paired with large values.
- In panel A, short people tend to be light, and tall people tend to be heavy.
- On the other hand, a dot cluster that has a slope from the upper left to the lower right, as in panel B of Figure 6.2, reflects a negative relationship.
- Small values of one variable tend to be paired with large values of the other variable, and vice versa.
- A dot cluster that lacks any apparent slope, as in panel C of Figure 6.2, reflects little or no relationship.
- Small values of one variable are just as likely to be paired with small, medium, or large values of the other variable.

Strong or Weak Relationship?

- The more closely the dot cluster approximates a straight line, the stronger (the more regular) the relationship will be.
- Figure 6.3 shows a series of scatterplots, each representing:
- A different positive relationship between IQ scores for pairs of people whose backgrounds reflect different degrees of genetic overlap, ranging from minimum overlap between foster parents and foster children to maximum overlap between identical twins.

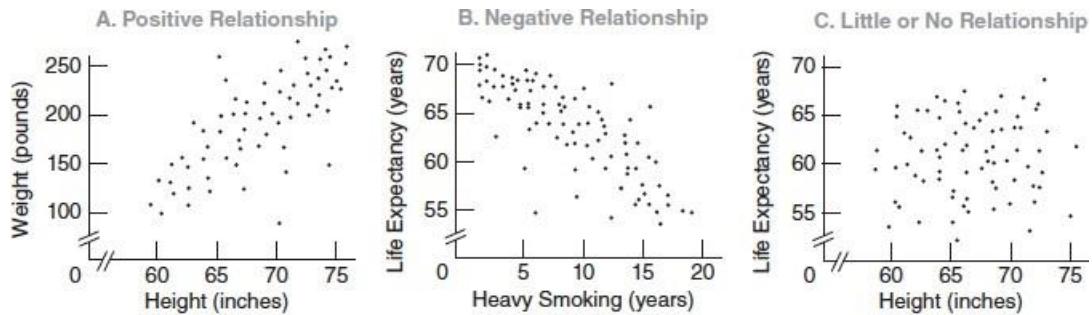


FIGURE 6.2
Three types of relationships.

Perfect Relationship

- A dot cluster that equals a straight line reflects a perfect relationship between two variables.

Curvilinear Relationship

- The a dot cluster approximates a straight line and, therefore, reflects a linear relationship.
- Sometimes a dot cluster approximates a bent or curved line, as in Figure 6.4, and therefore reflects a curvilinear relationship.
- Eg: physical strength, as measured by the force of a person's handgrip, is less for children, more for adults, and then less again for older people.

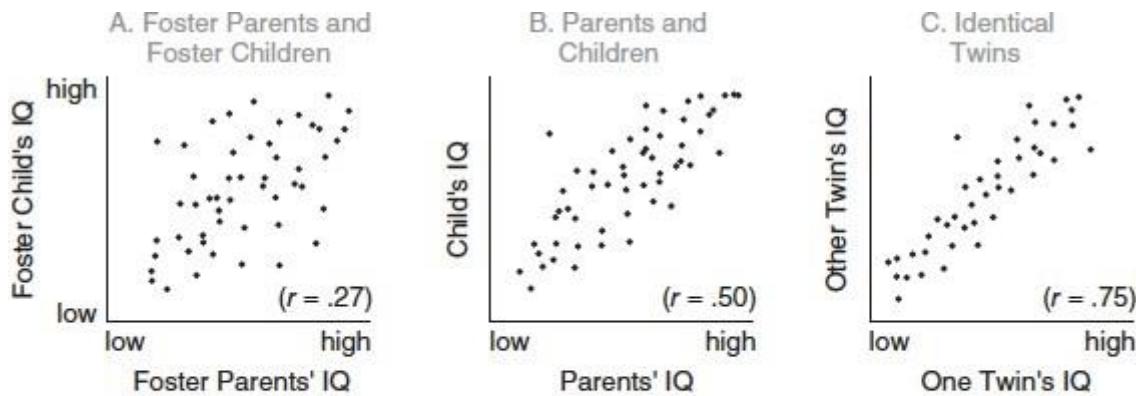


FIGURE 6.3

Three positive relationships. (Scatterplots simulated from a 50-year literature survey.)

*Source: Erlenmeyer-Kimling, L., & Jarvik, L. F. (1963). "Genetics and Intelligence: A Review." *Science*, 142, 1477–1479.*

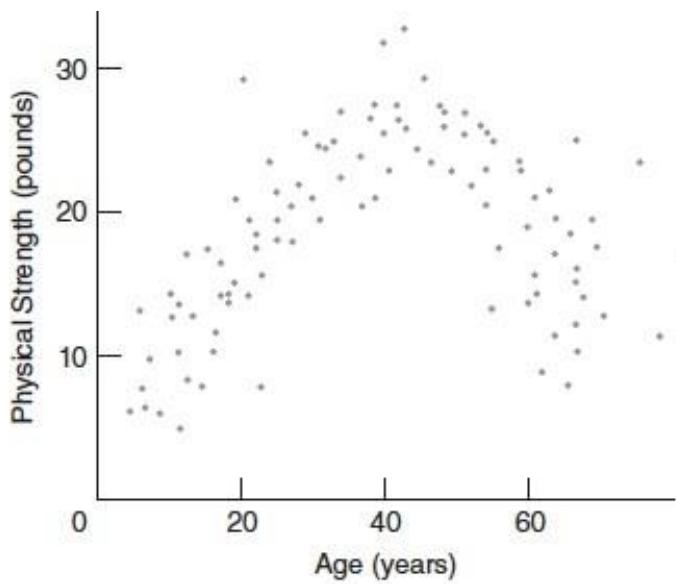
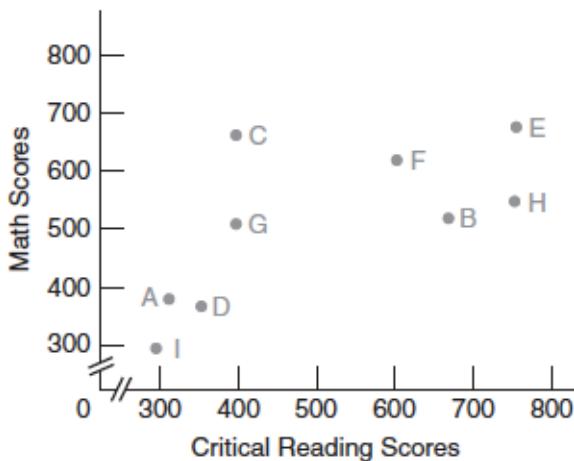


FIGURE 6.4
Curvilinear relationship.

Progress Check *6.2 Critical reading and math scores on the SAT test for students A, B, C, D, E, F, G, and H are shown in the following scatterplot:



- Which student(s) scored about the same on both tests?
- Which student(s) scored higher on the critical reading test than on the math test?
- Which student(s) will be eligible for an honors program that requires minimum scores of 700 in critical reading and 500 in math?
- Is there a negative relationship between the critical reading and math scores?

- 6.2** **(a)** I, D, F **(c)** E, H
 (b) B, H, E **(d)** No. The relationship is positive.

III. A CORRELATION COEFFICIENT

FOR QUANTITATIVE DATA : r

- A correlation coefficient is a number between -1 and 1 that describes the relationship between pairs of variables.
- The type of correlation coefficient, designated as r , that describes the linear relationship between pairs of variables for quantitative data.

Key Properties of r

- Named in honor of the British scientist Karl Pearson, the Pearson correlation coefficient, r , can equal any value between -1.00 and $+1.00$.
- Furthermore, the following two properties apply:
- The sign of r indicates the type of linear relationship, whether positive or negative.
- The numerical value of r , without regard to sign, indicates the strength of the linear relationship.

Sign of r

- A number with a plus sign (or no sign) indicates a positive relationship, and a number with a minus sign indicates a negative relationship.

Numerical Value of r

- The more closely a value of r approaches either -1.00 or $+1.00$, the stronger the relationship.
- The more closely the value of r approaches 0 , the weaker the relationship.
- $r = -.90$ indicates a stronger relationship than does an r of $-.70$, and
- $r = -.70$ indicates a stronger relationship than does an r of $.50$.

Interpretation of r

- Located along a scale from -1.00 to $+1.00$, the value of r supplies information about the direction of a linear relationship—whether positive or negative—and,
- generally, information about the relative strength of a linear relationship—whether relatively
- weak because r is in the vicinity of 0 , or relatively strong because r deviates from 0 in the direction of
- either $+1.00$ or -1.00 .

r Is Independent of Units of Measurement

- The value of r is independent of the original units of measurement.
- Same value of r describes the correlation between height and weight for a group of adults.
- r depends only on the pattern among pairs of scores, which in turn show no traces of the units of measurement for the original X and Y scores.

- A positive value of r reflects a tendency for pairs of scores to occupy similar relative locations in their respective distributions, while a negative value of r reflects a tendency for pairs of scores to occupy dissimilar relative locations in their respective distributions.

Range Restrictions

- The value of the correlation coefficient declines whenever the range of possible X or Y scores is restricted.
- For example, Figure 6.5 shows a dot cluster with an obvious slope, represented by an r of .70 for the positive relationship between height and weight for all college students.
- If the range of heights along Y is restricted to students who stand over 6 feet 2 inches (or 74 inches) tall, the abbreviated dot cluster loses its slope because of the weights among tall students.
- Therefore, as depicted in Figure 6.5, the value of r drops to .10.
- Sometimes it's impossible to avoid a range restriction.
- For example, some colleges only admit students with SAT test scores above some minimum value.

Caution

- We have to be careful when interpreting the actual numerical value of r .
- An r of .70 for height and weight doesn't signify that the strength of this relationship equals either .70 or 70 percent of the strength of a perfect relationship.
- The value of r can't be interpreted as a proportion or percentage of some perfect relationship.

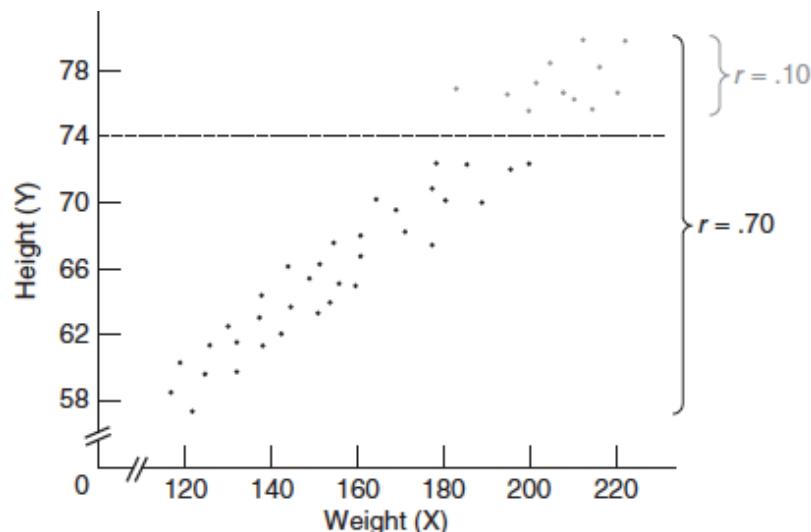


FIGURE 6.5
Effect of range restriction on the value of r .

Verbal Descriptions

- When interpreting a new r , you'll find it helpful to translate the numerical value of r into a verbal description of the relationship.
- An r of .70 for the height and weight of college students could be translated into "Taller students tend to weigh more";
- An r of $-.42$ for time spent taking an exam and the subsequent exam score could be translated into "Students who take less time tend to make higher scores"; and
- An r in the neighborhood of 0 for shoe size and IQ could be translated into "Little, if any, relationship exists between shoe size and IQ."

Progress Check *6.3 Supply a verbal description for each of the following correlations. (If necessary, visualize a rough scatterplot for r , using the scatterplots in Figure 6.3 as a frame of reference.)

- (a) an r of $-.84$ between total mileage and automobile resale value
- (b) an r of $-.35$ between the number of days absent from school and performance on a math achievement test
- (c) an r of $.03$ between anxiety level and college GPA
- (d) an r of $.56$ between age of schoolchildren and reading comprehension

- 6.3**
- (a) Cars with more total miles tend to have lower resale values.
 - (b) Students with more absences from school tend to score lower on math achievement tests.
 - (c) Little or no relationship between anxiety level and college GPA.
 - (d) Older schoolchildren tend to have better reading comprehension.

Correlation Not Necessarily Cause-Effect

- A correlation coefficient, regardless of size, never provides information about whether an observed relationship reflects a simple cause-effect relationship or some more complex state of affairs.
- Eg: correlation between cigarette smoking and lung cancer.
- American Cancer Society representatives interpreted the correlation as a causal relationship: Smoking produces lung cancer.
- Tobacco industry representatives interpreted the correlation as, both the desire to smoke cigarettes and lung cancer are caused by some more basic but unidentified factors, such as the body metabolism or personality of some people.
- According to this reasoning, people with a high body metabolism might be more prone to smoke and, quite independent of their smoking, more vulnerable to lung cancer.
- Therefore, smoking correlates with lung cancer because both are effects of some common cause or causes.

Role of Experimentation

- In the present case, laboratory animals were trained to inhale different amounts of tobacco tars and were then euthanized.
- Autopsies revealed that the observed incidence of lung cancer varied directly with the amount of inhaled tobacco tars, even though possible “contaminating” factors, such as different body metabolisms or personalities, had been neutralized either through experimental control or by random assignment of the subjects to different test conditions.

Progress Check *6.4 Speculate on whether the following correlations reflect simple cause-effect relationships or more complex states of affairs. (Hint: A cause-effect relationship implies that, if all else remains the same, any change in the causal variable should always produce a predictable change in the other variable.)

- (a) caloric intake and body weight
- (b) height and weight
- (c) SAT math score and score on a calculus test
- (d) poverty and crime

6.4 (a) simple cause-effect (b) complex (c) complex (d) complex

IV. DETAILS: COMPUTATION FORMULA FOR r

Calculate a value for r by using the following computation formula:

CORRELATION COEFFICIENT (COMPUTATION FORMULA)

$$r = \frac{SP_{xy}}{\sqrt{SS_x SS_y}} \quad (6.1)$$

where the two sum of squares terms in the denominator are defined as

$$\begin{aligned} SS_x &= \sum (X - \bar{X})^2 = \sum X^2 - \frac{(\sum X)^2}{n} \\ SS_y &= \sum (Y - \bar{Y})^2 = \sum Y^2 - \frac{(\sum Y)^2}{n} \end{aligned}$$

and the sum of the products term in the numerator, SP_{xy} , is defined in Formula 6.2.

SUM OF PRODUCTS (DEFINITION AND COMPUTATION FORMULAS)

$$SP_{xy} = \sum (X - \bar{X})(Y - \bar{Y}) = \sum XY - \frac{(\sum X)(\sum Y)}{n} \quad (6.2)$$

Progress Check *6.5 Couples who attend a clinic for first pregnancies are asked to estimate (independently of each other) the ideal number of children. Given that X and Y represent the estimates of females and males, respectively, the results are as follows:

COUPLE	X	Y
A	1	2
B	3	4
C	2	3
D	3	2
E	1	0
F	2	3

Calculate a value for r , using the computation formula (6.1).

$$6.5 \quad r = \frac{4}{\sqrt{(4)(9.33)}} = .65$$

Table 6.3
CALCULATION OF r : COMPUTATION FORMULA

A. COMPUTATIONAL SEQUENCE

Assign a value to n (1), representing the number of pairs of scores.

Sum all scores for X (2) and for Y (3).

Find the product of each pair of X and Y scores (4), one at a time, then add all of these products (5).

Square each X score (6), one at a time, then add all squared X scores (7).

Square each Y score (8), one at a time, then add all squared Y scores (9).

Substitute numbers into formulas (10) and solve for SP_{xy} , SS_x , and SS_y .

Substitute into formula (11) and solve for r .

B. DATA AND COMPUTATIONS

FRIEND	CARDS		4	6	8
	SENT, X	RECEIVED, Y	XY	X^2	Y^2
Doris	13	14	182	169	196
Steve	9	18	162	81	324
Mike	7	12	84	49	144
Andrea	5	10	50	25	100
John	1	6	6	1	36

$$1 \ n = 5 \quad 2 \ \Sigma X = 35 \quad 3 \ \Sigma Y = 60 \quad 5 \ \Sigma XY = 484 \quad 7 \ \Sigma X^2 = 325 \quad 9 \ \Sigma Y^2 = 800$$

$$10 \ SP_{xy} = \sum XY - \frac{(\sum X)(\sum Y)}{n} = 484 - \frac{(35)(60)}{5} = 484 - 420 = 64$$

$$SS_x = \sum X^2 - \frac{(\sum X)^2}{n} = 325 - \frac{(35)^2}{5} = 325 - 245 = 80$$

$$SS_y = \sum Y^2 - \frac{(\sum Y)^2}{n} = 800 - \frac{(60)^2}{5} = 800 - 720 = 80$$

$$11 \ r = \frac{SP_{xy}}{\sqrt{SS_x SS_y}} = \frac{64}{\sqrt{(80)(80)}} = \frac{64}{80} = .80$$

V. Regression

TWO ROUGH PREDICTIONS

- A correlation analysis of the exchange of greeting cards by five friends for the most recent holiday season suggests a strong positive relationship between cards sent and cards received.
- When informed of these results, another friend, Emma, who enjoys receiving greeting cards, asks you to predict how many cards she will receive during the next holiday season, assuming that she plans to send 11 cards.

TWO ROUGH PREDICTIONS

- Predict “Relatively Large Number”

Rough Prediction for Emma:

- We could offer Emma a very rough prediction by recalling that cards sent and received tend to occupy similar relative locations in their respective distributions.
- Therefore, Emma can expect to receive a relatively large number of cards, since she plans to send a relatively large number of cards.

Predict “between 14 and 18 Cards”

- To obtain a slightly more precise prediction for Emma, refer to the scatter plot for the original five friends shown in Figure 7.1.
- Notice that Emma’s plan to send 11 cards locates her along the X axis between the 9 cards sent by Steve and the 13 sent by Doris.
- Using the dots for Steve and Doris as guides, construct two strings of arrows, one beginning at 9 and ending at 18 for Steve and the other beginning at 13 and ending at 14 for Doris.
- We can predict that Emma’s return should be between 14 and 18 cards, the numbers received by Doris and Steve.

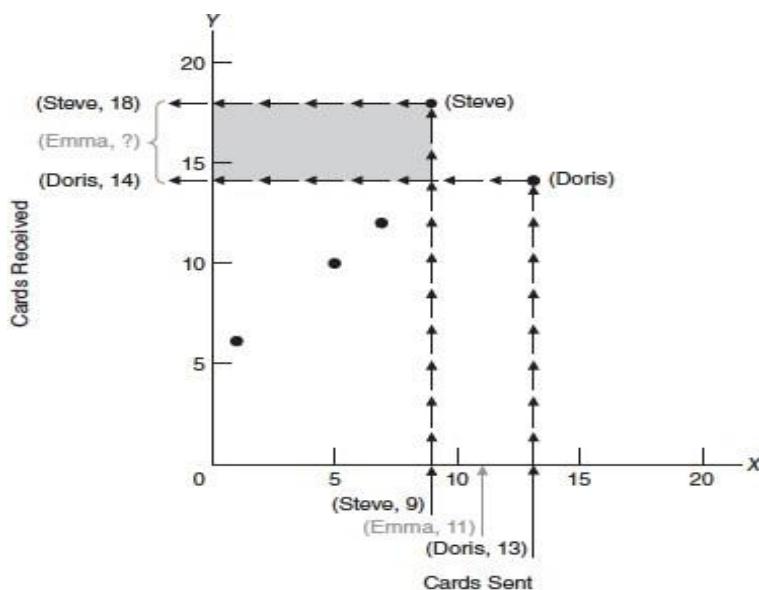


FIGURE 7.1
A rough prediction for Emma (using dots for Steve and Doris).

VI. A REGRESSION LINE

- All five dots contribute to the more precise prediction, illustrated in Figure 7.2, that Emma will receive 15.20 cards.
- The solid line designated as the regression line in Figure 7.2, which guides the string of arrows, beginning at 11, toward the predicted value of 15.20.
- If all five dots had defined a single straight line, placement of the regression line would have been simple; merely let it pass through all dots.

Predictive Errors

- Figure 7.3 illustrates the predictive errors that would have occurred if the regression line had been used to predict the number of cards received by the five friends.
- Solid dots reflect the actual number of cards received, and open dots, always located along the regression line, reflect the predicted number of cards received.
- The largest predictive error, shown as a broken vertical line, occurs for Steve, who sent 9 cards.
- Although he actually received 18 cards, he should have received slightly fewer than 14 cards, according to the regression line.
- The smallest predictive error none for Mike, who sent 7 cards.
- He actually received the 12 cards that he should have received, according to the regression line.

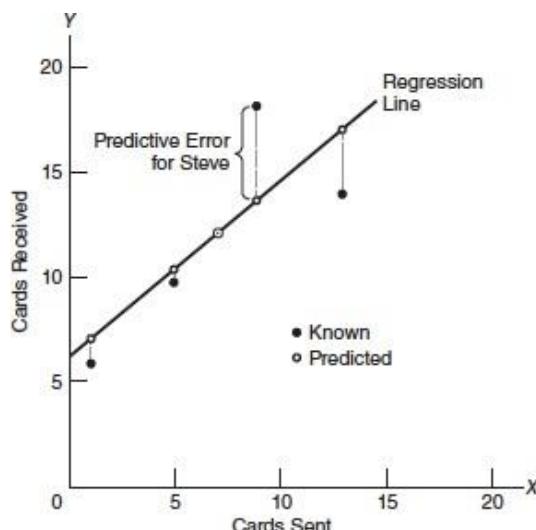


FIGURE 7.3
Predictive errors.

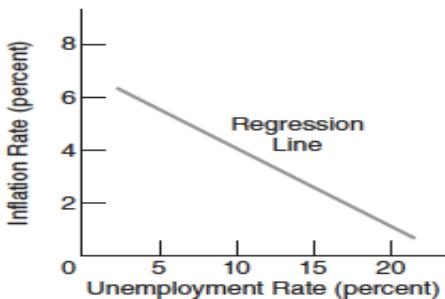
Total Predictive Error

The smaller the total for all predictive errors in Figure 7.3, the more favorable will be the prognosis for our predictions.

The regression line to be placed in a position that minimizes the total predictive error, that is, that minimizes the total of the vertical discrepancies between the solid and open dots shown in Figure 7.3.

- (a) Predict the approximate rate of inflation, given an unemployment rate of 5 percent.
(b) Predict the approximate rate of inflation, given an unemployment rate of 15 percent.

Progress Check *7.1 To check your understanding of the first part of this chapter, make predictions using the following graph.



- 7.1 (a) approximately 5–6 percent
(b) approximately 2–3 percent**

VII. LEAST SQUARES REGRESSION LINE

- To avoid the arithmetic standoff of zero always produced by adding positive and negative predictive errors
- the placement of the regression line minimizes the total squared predictive error.
- When located like this, the regression line is often referred to as the least squares regression line.

Least Squares Regression Equation

Happily, an equation pinpoints the exact least squares regression line for any scatterplot. Most generally, this equation reads:

LEAST SQUARES REGRESSION EQUATION

$$Y' = bX + a \quad (7.1)$$

where Y' represents the predicted value (the predicted number of cards that will be received by any new friend, such as Emma); X represents the known value (the known number of cards sent by any new friend); and b and a represent numbers calculated from the original correlation analysis, as described next.*

Finding Values of b and a

To obtain a working regression equation, solve each of the following expressions, first for b and then for a , using data from the original correlation analysis. The expression for b reads:

SOLVING FOR b

$$b = r \sqrt{\frac{SS_y}{SS_x}} \quad (7.2)$$

where r represents the correlation between X and Y (cards sent and received by the five friends); SS_y represents the sum of squares for all Y scores (the cards received by the five friends); and SS_x represents the sum of squares for all X scores (the cards sent by the five friends).

The expression for a reads:

SOLVING FOR a

$$a = \bar{Y} - b\bar{X} \quad (7.3)$$

where \bar{Y} and \bar{X} refer to the sample means for all Y and X scores, respectively, and b is defined by the preceding expression.

The values of all terms in the expressions for b and a can be obtained from the original correlation analysis either directly, as with the value of r , or indirectly, as with the values of the remaining terms: SS_y , SS_x , \bar{Y} , and \bar{X} . **Table 7.1** illustrates the computational sequence that produces a least squares regression equation for the greeting card example, namely,

$$Y' = .80(X) + 6.40$$

where .80 and 6.40 represent the values computed for b and a , respectively.

Table 7.1
DETERMINING THE LEAST SQUARES REGRESSION EQUATION

A. COMPUTATIONAL SEQUENCE

- Determine values of SS_x , SS_y , and r (1) by referring to the original correlation analysis in Table 6.3.
- Substitute numbers into the formula (2) and solve for b .
- Assign values to X and Y (3) by referring to the original correlation analysis in Table 6.3.
- Substitute numbers into the formula (4) and solve for a .
- Substitute numbers for b and a in the least squares regression equation (5).

B. COMPUTATIONS

$$1 \quad SS_x = 80^*$$

$$SS_y = 80^*$$

$$r = .80$$

$$2 \quad b = r \sqrt{\frac{SS_y}{SS_x}} = .80 \sqrt{\frac{80}{80}} = .80$$

$$\bar{X} = 7^{**}$$

$$3 \quad \bar{Y} = 12^{**}$$

$$4 \quad a = \bar{Y} - (b)(\bar{X}) = 12 - (.80)(7) = 12 - 5.60 = 6.40$$

$$5 \quad Y' = (b)(X) + a \\ = (.80)(X) + 6.40$$

Key Property

- Once numbers have been assigned to b and a , as just described, the least squares regression equation emerges as a working equation with a most desirable property:
- It automatically minimizes the total of all squared predictive errors for known Y scores in the original correlation analysis.

Solving for Y'

- In its present form, the regression equation can be used to predict the number of cards that Emma will receive, assuming that she plans to send 11 cards.
- Simply substitute 11 for X and solve for the value of Y' as follows:

$$Y' = .80(11) + 6.40$$

$$= 8.80 + 6.40$$

$$= 15.20$$

- Even when no cards are sent ($X = 0$), we predict a return of 6.40 cards because of the value of a .
- Also, notice that sending each additional card translates into an increment of only .80 in the predicted return because of the value of b .

- Whenever b has a value less than 1.00, increments in the predicted return will lag—by an amount equal to the value of b , that is, .80 in the present case—behind increments in cards sent.
- If the value of b had been greater than 1.00, then increments in the predicted return would have exceeded increments in cards sent.

A Limitation

- Emma might survey these predicted card returns before committing herself to a particular card investment. There is no evidence of a simple cause-effect relationship between cards sent and cards received.

Progress Check *7.2 Assume that an r of .30 describes the relationship between educational level (highest grade completed) and estimated number of hours spent reading each week. More specifically:

EDUCATIONAL LEVEL (X)	WEEKLY READING TIME (Y)
$\bar{X} = 13$	$\bar{Y} = 8$
$SS_x = 25$	$SS_y = 50$
$r = .30$	

- Determine the least squares equation for predicting weekly reading time from educational level.
- Faith's education level is 15. What is her predicted reading time?
- Keegan's educational level is 11. What is his predicted reading time?

7.2 (a) $b = \sqrt{\frac{50}{25}}(.30) = .42; a = 8 - (.42)(13) = 2.54$

(b) $Y' = (.42)(15) + 2.54 = 8.84$

(c) $Y' = (.42)(11) + 2.54 = 7.16$

VIII. STANDARD ERROR OF ESTIMATE, $s_{y|x}$

- Emma's investment of 11 cards will yield a return of 15.20 cards, we would be surprised if she actually received 15 cards.
- It is more likely that because of the imperfect relationship between cards sent and cards received,
- Emma's return will be some number other than 15.
- Although designed to minimize predictive error, the least squares equation does not eliminate it.

Finding the Standard Error of Estimate

The estimate of error for new predictions reflects our failure to predict the number of cards received by the original five friends, as depicted by the discrepancies between solid and open dots in Figure 7.3. Known as the *standard error of estimate* and symbolized as s_{yx} , this estimate of predictive error complies with the general format for any sample standard deviation, that is, the square root of a sum of squares term divided by its degrees of freedom. (See Formula 4.10 on page 76.) The formula for s_{yx} reads:

STANDARD ERROR OF ESTIMATE (DEFINITION FORMULA)

$$s_{yx} = \sqrt{\frac{SS_{yx}}{n-2}} = \sqrt{\frac{\sum(Y - Y')^2}{n-2}} \quad (7.4)$$

where the sum of squares term in the numerator, SS_{yx} , represents the sum of the squares for predictive errors, $Y - Y'$, and the degrees of freedom term in the denominator, $n - 2$, reflects the loss of two degrees of freedom because any straight line, including the regression line, can be made to coincide with two data points. The symbol s_{yx} is read as “ s sub y given x .”

Although we can estimate the overall predictive error by dealing directly with predictive errors, $Y - Y'$, it is more efficient to use the following computation formula:

STANDARD ERROR OF ESTIMATE (COMPUTATION FORMULA)

$$s_{yx} = \sqrt{\frac{SS_y(1-r^2)}{n-2}} \quad (7.5)$$

where SS_y is the sum of the squares for Y scores (cards received by the five friends), that is,

$$SS_y = \Sigma(Y - \bar{Y}) = \Sigma Y^2 - \frac{(\Sigma Y)^2}{n}$$

and r is the correlation coefficient (cards sent and received).

Key Property

The standard error of estimate represents a special kind of standard deviation that reflects the magnitude of predictive error.

You might find it helpful to think of the standard error of estimate, s_{yx} , as a rough measure of the average amount of predictive error—that is, as a rough measure of the average amount by which known Y values deviate from their predicted \hat{Y} values.*

The value of 3.10 for s_{yx} , as calculated in **Table 7.3**, represents the standard deviation for the discrepancies between known and predicted card returns originally shown in Figure 7.3. In its role as an estimate of predictive error, the value of s_{yx} can be attached to any new prediction. Thus, a concise prediction statement may read: “The predicted card return for Emma equals 15.20 ± 3.10 ,” in which the latter term serves as a rough estimate of the average amount of predictive error, that is, the average amount by which 15.20 will either overestimate or underestimate Emma’s true card return.

Table 7.3
CALCULATION OF THE STANDARD ERROR OF ESTIMATE, s_{yx}

A. COMPUTATIONAL SEQUENCE

Assign values to SS_y and r (1) by referring to previous work with the least squares regression equation in Table 7.1.

Substitute numbers into the formula (2) and solve for s_{yx} .

B. COMPUTATIONS

1 $SS_y = 80$

$r = .80$

2
$$s_{yx} = \sqrt{\frac{SS_y(1-r^2)}{n-2}} = \sqrt{\frac{80(1-[.80]^2)}{5-2}} = \sqrt{\frac{80(.36)}{3}} = \sqrt{\frac{28.80}{3}} = \sqrt{9.60}$$
$$= 3.10$$

Importance of r

Substituting a value of 1 for r, we obtain

$$SS_{y|x} = SS_y(1 - r^2) = SS_y[1 - (1)^2] = SS_y[1 - 1] = SS_y[0] = 0$$

substituting a value of 0 for r in the numerator of Formula 7.5, we obtain

$$SS_{y|x} = SS_y(1 - r^2) = SS_y[1 - (0)^2] = SS_y[1 - 0] = SS_y[1] = SS_y$$

Progress Check *7.3

- (a) Calculate the standard error of estimate for the data in Question 7.2 on page 132, assuming that the correlation of .30 is based on $n = 35$ pairs of observations.
- (b) Supply a rough interpretation of the standard error of estimate.

$$7.3 \quad (a) \quad s_{y|x} = \sqrt{\frac{50(1 - [.30]^2)}{35-2}} = \sqrt{\frac{50(.91)}{33}} = \sqrt{1.38} = 1.17$$

ASSUMPTIONS

Linearity

- Use of the regression equation requires that the underlying relationship be linear.

Homoscedasticity

- Use of the standard error of estimate, $s_{y|x}$, assumes that except for chance, the dots in the original scatterplot will be dispersed equally about all segments of the regression line.
- when the scatterplot reveals a dramatically different type of dot cluster, such as that shown in Figure 7.4.
- The standard error of estimate for the data in Figure 7.4 should be used cautiously, since its value overestimates the variability of dots about the lower half of the regression line and underestimates the variability of dots about the upper half of the regression line.

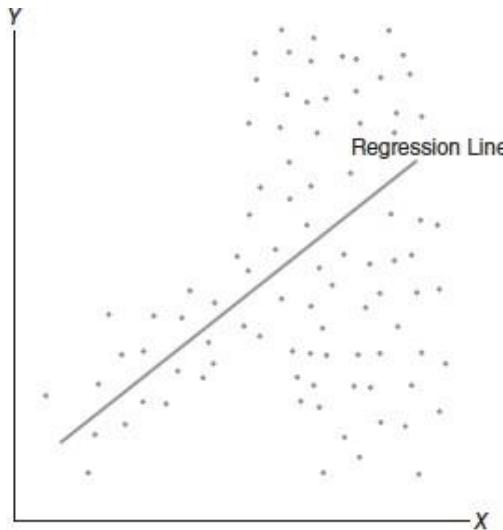


FIGURE 7.4
Violation of homoscedasticity assumption. (Dots lack equal variability about all line segments.)

INTERPRETATION OF r^2

- The squared correlation coefficient, r^2 , provides us a key interpretation of the correlation coefficient and also a measure of predictive accuracy that supplements the standard error of estimate, $s_{y|x}$.

Repetitive Prediction of the Mean

- Pretend that we know the Y scores (cards received), but not the corresponding X scores (cards sent), for each of the five friends.
- Lacking information about the relationship between X and Y scores, we could not construct a regression equation and use it to generate a customized prediction, \hat{Y}' , for each friend.
- We mount a primitive predictive effort by always predicting the mean, \bar{Y} , for each of the five friends' Y scores.
- The repetitive prediction of \bar{Y} for each of the Y scores of all five friends will supply us with a frame of reference against which to evaluate our customary predictive effort based on the correlation between cards sent (X) and cards received (Y).

Predictive Errors

Panel A of Figure 7.5 shows the predictive errors for all five friends when the mean for all five friends, \bar{Y} , of 12 (shown as the mean line) is always used to predict each of their five Y scores.

Panel B shows the corresponding predictive errors for all five friends when a series of different \hat{Y}' values, obtained from the least squares equation (shown as the least squares line), is used to predict each of their five Y scores.

Panel A of Figure 7.5 shows the error for John when the mean for all five friends, \bar{Y} , of 12 is used to predict his Y score of 6.

Shown as a broken vertical line, the error of -6 for John (from $Y - \bar{Y} = 6 - 12 = -6$) indicates that \bar{Y} overestimates John's Y score by 6 cards. Panel B shows a smaller error of -1.20 for John when a \bar{Y}' value of

7.20 is used to predict the same Y score of 6.

This \bar{Y}' value of 7.20 is obtained from the least squares equation, where the number of cards sent by John, 1, has been substituted for X .

$$\begin{aligned}\bar{Y}' &= .80(X) + 6.40 \\ &= .80(1) + 6.40 \\ &= 7.20\end{aligned}$$

Error Variability (Sum of Squares)

The sum of squares of any set of deviations, now called errors, can be calculated by first squaring each error (to eliminate negative signs), then summing all squared errors.

$$SS_y = \sum(Y - \bar{Y})^2$$

Using the errors for the five friends shown in Panel A of Figure 7.5, this becomes

$$SS_y = [(-6)^2 + (-2)^2 + 0^2 + 6^2 + 2^2] = 80$$

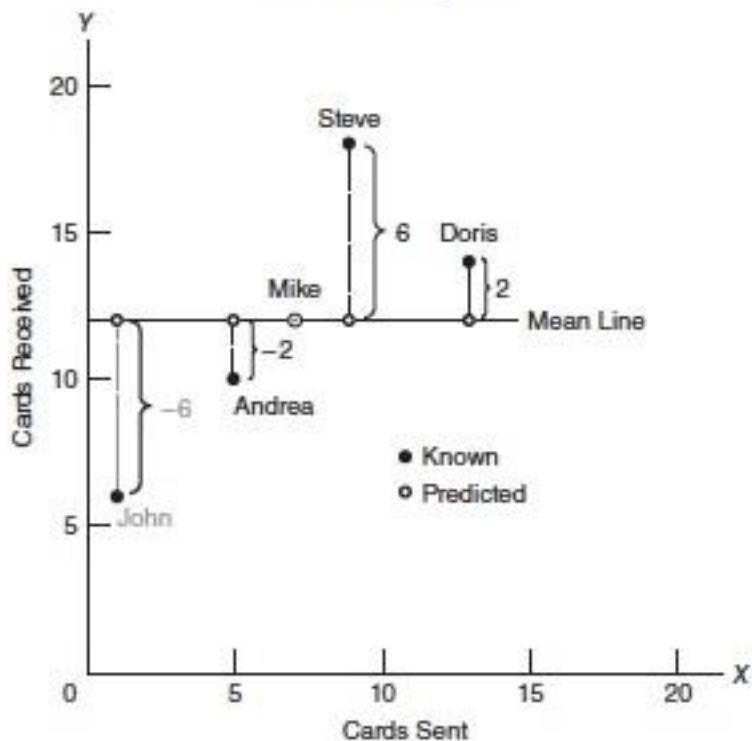
The error variability for the customized predictions from the least squares equation can be designated as SS_{yx} , since each Y score is expressed as a squared deviation from its corresponding \bar{Y}' and then summed, that is

$$SS_{yx} = \sum(Y - \bar{Y}')^2$$

Using the errors for the five friends shown in Panel B of Figure 7.5, we obtain:

$$SS_{yx} = [(-1.2)^2 + (-0.4)^2 + 0^2 + (4.4)^2 + (-2.8)^2] = 28.8$$

A. Errors Using Mean



B. Errors Using Least Squares Equation

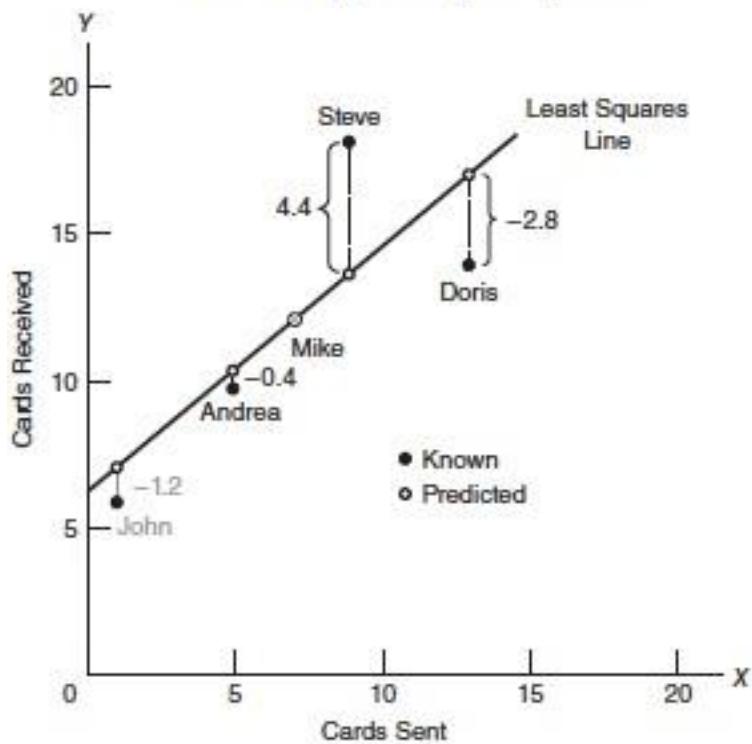


FIGURE 7.5

Predictive errors for five friends.

Proportion of Predicted Variability

SS_y measures the total variability of Y scores that occurs after only primitive predictions based on Y are made while $SS_{y|x}$ measures the residual variability of Y scores that remains after customized leastsquare predictions are made.

The error variability of 28.8 for the least squares predictions is much smaller than the error variability of 80 for the repetitive prediction of Y, confirming the greater accuracy of the least squares predictions apparent in Figure 7.5.

To obtain an SS measure of the actual gain in accuracy due to the least squares predictions, subtract the residual variability from the total variability, that is, subtract $SS_{y|x}$ from SS_y , to obtain

This result, .64 or 64 percent, represents the proportion or percent gain in predictive accuracy when the repetitive prediction of Y is replaced by a series of customized Y' predictions based on the least squares equation.

$$SS_y - SS_{y|x} = 80 - 28.8 = 51.2$$

To express this difference, 51.2, as a gain in accuracy *relative* to the original error variability for the repetitive prediction of \bar{Y} , divide the above difference by SS_y , that is,

$$\frac{SS_y - SS_{y|x}}{SS_y} = \frac{80 - 28.8}{80} = \frac{51.2}{80} = .64$$

the square of the correlation coefficient, r^2 , always indicates the proportion of total variability in one variable that is predictable from its relationship with the other variable.

Expressing the equation for r^2 in symbols, we have:

r^2 INTERPRETATION

$$r^2 = \frac{SS_{Y'}}{SS_Y} = \frac{SS_Y - SS_{Y|x}}{SS_Y} \quad (7.6)$$

where the one new sum of squares term, $SS_{Y'}$, is simply the variability explained by or predictable from the regression equation, that is,

$$SS_{Y'} = \sum(Y' - \bar{Y})^2$$

Accordingly, r^2 provides us with a straightforward measure of the worth of our least squares predictive effort.*

r^2 Does Not Apply to Individual Scores:

- The total variability of all Y scores—as measured by SS_Y —can be reduced by 64 percent when each Y score is replaced by its corresponding predicted Y' score and then expressed as a squared deviation from the mean of all observed scores.
- Thus, the 64 percent represents a reduction in the total variability for the five Y scores when they are replaced by a succession of predicted scores, given the least-squares equation and various values of X.

Small Values of r^2

- When transposed from r to r^2 , Cohen's guidelines, state that a value of r^2 in the vicinity of .01, .09, or .25 reflects a weak, moderate, or strong relationship, respectively.

r^2 Doesn't Ensure Cause-Effect

- If the correlation between mental health scores of sixth graders and their weaning ages as infants equals .20, we cannot claim, therefore, that $(.20)(20) = .04$ or 4 percent of the total variability in mental health scores is caused by the differences in weaning ages.
- r^2 is indicating the proportion or percent of predictable variability, you also might encounter references to r^2 as indicating the proportion or percent of explained variability.
- In this context, “explained” signifies only predictability, not causality.

Progress Check *7.4 Assume that an r of .30 describes the relationship between educational level and estimated hours spent reading each week.

- According to r^2 , what percent of the variability in weekly reading time is predictable from its relationship with educational level?
- What percent of variability in weekly reading time is not predictable from this relationship?
- Someone claims that 9 percent of *each* person's estimated reading time is predictable from the relationship. What is wrong with this claim?

Progress Check *7.5 As indicated in Figure 6.3 on page 111, the correlation between the IQ scores of parents and children is .50, and that between the IQ scores of foster parents and foster children is .27.

- Does this signify, therefore, that the relationship between foster parents and foster children is about one-half as strong as the relationship between parents and children?
- Use r^2 to compare the strengths of these two correlations.

- 7.4** (a) 9 percent predicted.
(b) 91 percent not predicted.
(c) 9 percent refers to the variability of *all* estimated reading times.

- 7.5** (a) No
(b) The r^2 of .25 for parents and children is about four times greater than the r^2 of .07 for foster parents and foster children.

X. MULTIPLE REGRESSION EQUATIONS

- Any serious predictive effort usually culminates in a more complex equation that contains not just one but several X, or predictor variables.
- For instance, a serious effort to predict college GPA might culminate in the following equation:

$$Y' = .410(X_1) + .005(X_2) + .001(X_3) + 1.03$$

- where Y' represents predicted college GPA and X1, X2, and X3 refer to high school GPA, IQ score, and SAT score, respectively.
- By capitalizing on the combined predictive power of several predictor variables, these multiple regression equations supply more accurate predictions for Y' than could be obtained from a simple regression equation.

XI. REGRESSION TOWARD THE MEAN

- Regression toward the mean refers to a tendency for scores, particularly extreme scores, to shrink toward the mean.
- For example, because of regression toward the mean, we would expect that students who made the top five scores on the first statistics exam would not make the top five scores on the second statistics exam. Although all five students might score above the mean on the second exam, some of their scores would regress back toward the mean.
- On the second test, even though the scores of these five students continue to reflect an above-average permanent component, some of their scores will suffer because of less good luck or even bad luck.
- The net effect is that the scores of at least some of the original five top students will drop below the top five scores—that is, regress back toward the mean—on the second exam.

Appears in Many Distributions

- Regression toward the mean appears among subsets of extreme observations for a wide variety of distributions.
- It appears for the subset of best (or worst) performing stocks on the New York Stock Exchange across any period, such as a week, month, or year.
- It also appears for the top (or bottom) major league baseball hitters during consecutive seasons. Table 7.4 lists the top 10 hitters in the major leagues during 2014 and shows how they fared during 2015.

- Notice that 7 of the top 10 batting averages regressed downward, toward .260s, the approximate mean for all hitters during 2015.

The Regression Fallacy

- The regression fallacy is committed whenever regression toward the mean is interpreted as a real, rather than a chance, effect.
- A classic example of the regression fallacy occurred in an Israeli Air Force study of pilot training.
- Some trainees were praised after very good landings, while others were reprimanded after very bad landings.
- On their next landings, praised trainees did more poorly and reprimanded trainees did better.
- A valid conclusion considers regression toward the mean.

Table 7.4
**REGRESSION TOWARD THE MEAN: BATTING AVERAGES OF TOP
 10 HITTERS IN MAJOR LEAGUE BASEBALL
 DURING 2014 AND HOW THEY FARED DURING 2015**

TOP 10 HITTERS (2014)	BATTING AVERAGES*		REGRESS TOWARD MEAN?
	2014	2015	
1. J. Altuve	.341	.313	Yes
2. V. Martinez	.335	.282	Yes
3. M. Brantley	.327	.310	Yes
4. A. Beltre	.324	.287	Yes
5. J. Abreu	.317	.290	Yes
6. R. Cano	.314	.287	Yes
7. A. McCutchen	.314	.292	Yes
8. M. Cabrera	.313	.338	No
9. B. Posey	.311	.318	No
10. B. Revere	.306	.306	No

Avoiding the Regression Fallacy

- The regression fallacy can be avoided by splitting the subset of extreme observations into two groups.
- In the previous example, one group of trainees would continue to be praised after very good landings and reprimanded after very poor landings.
- A second group of trainees would receive no feedback whatsoever after very good and very bad landings.
- In effect, the second group would serve as a control for regression toward the mean, since any shift toward the mean on their second landings would be due to chance.
- Most important, any observed difference between the two groups would be viewed as a real difference not attributable to the regression effect.

Progress Check *7.6 After a group of college students attended a stress-reduction clinic, declines were observed in the anxiety scores of those who, prior to attending the clinic, had scored high on a test for anxiety.

- (a) Can this decline be attributed to the stress-reduction clinic? Explain your answer.
- (b) What type of study, if any, would permit valid conclusions about the effect of the stress-reduction clinic?

Answers on page 429.

- 7.6**
- (a) No, because the observed decline could be due to regression toward the mean, given that the students scored high on the anxiety test prior to attending the clinic.
 - (b) An experiment where students who score high on the anxiety test are randomly assigned either to attend the stress-reduction clinic or to be in a control group.

UNIT IV

PYTHON LIBRARIES FOR DATA WRANGLING

Basics of Numpy arrays – aggregations – computations on arrays – comparisons, masks, boolean logic – fancy indexing – structured arrays – Data manipulation with Pandas – data indexing and selection – operating on data – missing data – Hierarchical indexing – combining datasets – aggregation and grouping – pivot tables

I. Basics of Numpy arrays:

NumPy array manipulation to access data and subarrays, and to split, reshape, and join the arrays.

Basic array manipulations:

Attributes of arrays

Determining the size, shape, memory consumption, and data types of arrays

Indexing of arrays

Getting and setting the value of individual array elements

Slicing of arrays

Getting and setting smaller subarrays within a larger array

Reshaping of arrays

Changing the shape of a given array

Joining and splitting of arrays

Combining multiple arrays into one, and splitting one array into many

NumPy Array Attributes

1. Creating numpy arrays:

The Three random arrays: a one-dimensional, two-dimensional, and three-dimensional array.

We'll use NumPy's random number generator, which we will seed with a set value in order to ensure that the same random arrays are generated each time this code is run:

```
In[1]: import numpy as np
        np.random.seed(0) # seed for reproducibility

        x1 = np.random.randint(10, size=6) # One-dimensional array
        x2 = np.random.randint(10, size=(3, 4)) # Two-dimensional array
        x3 = np.random.randint(10, size=(3, 4, 5)) # Three-dimensional array
```

2. Attributes:

Each array has attributes ndim (the number of dimensions), shape (the size of each dimension), and size (the total size of the array):

```
In[2]: print("x3 ndim: ", x3.ndim)
        print("x3 shape:", x3.shape)
        print("x3 size: ", x3.size)

x3 ndim: 3
x3 shape: (3, 4, 5)
x3 size: 60
```

```
In[3]: print("dtype:", x3.dtype)
dtype: int64

In[4]: print("itemsize:", x3.itemsize, "bytes")
        print(" nbytes:", x3.nbytes, "bytes")

itemsize: 8 bytes
 nbytes: 480 bytes
```

3. Array Indexing: Accessing Single Elements

In a one-dimensional array, you can access the i th value (counting from zero) by specifying the desired index in square brackets, just as with Python lists:

```
In[5]: x1
Out[5]: array([5, 0, 3, 3, 7, 9])
In[6]: x1[0]
Out[6]: 5
In[7]: x1[4]
Out[7]: 7
```

To index from the end of the array, you can use negative indices:

```
In[8]: x1[-1]
Out[8]: 9
In[9]: x1[-2]
Out[9]: 7
```

In a multidimensional array, you access items using a comma-separated tuple of indices:

```
In[10]: x2
Out[10]: array([[3, 5, 2, 4],
                 [7, 6, 8, 8],
                 [1, 6, 7, 7]])
In[11]: x2[0, 0]
Out[11]: 3
```

```
In[12]: x2[2, 0]  
Out[12]: 1  
  
In[13]: x2[2, -1]  
Out[13]: 7
```

You can also modify values using any of the above index notation:

```
In[14]: x2[0, 0] = 12  
x2  
  
Out[14]: array([[12, 5, 2, 4],  
                 [7, 6, 8, 8],  
                 [1, 6, 7, 7]])
```

4. Array Slicing: Accessing Subarrays

We can also use them to access subarrays with the slice notation, marked by the colon (:) character. The NumPy slicing syntax follows that of the standard Python list; to access a slice of an array x , use this: $x[start:stop:step]$. If any of these are unspecified, they default to the values start=0, stop=size of dimension, step=1.

One-dimensional subarrays

```
In[16]: x = np.arange(10)  
x  
  
Out[16]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])  
  
In[17]: x[:5] # first five elements  
Out[17]: array([0, 1, 2, 3, 4])  
  
In[18]: x[5:] # elements after index 5  
Out[18]: array([5, 6, 7, 8, 9])  
  
In[19]: x[4:7] # middle subarray  
Out[19]: array([4, 5, 6])  
  
  
In[22]: x[::-1] # all elements, reversed  
Out[22]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])  
  
In[23]: x[5::-2] # reversed every other from index 5  
Out[23]: array([5, 3, 1])
```

Multidimensional subarrays

Multidimensional slices work in the same way, with multiple slices separated by commas. For example:

```
In[24]: x2
Out[24]: array([[12, 5, 2, 4],
   [ 7, 6, 8, 8],
   [ 1, 6, 7, 7]])
In[25]: x2[:2, :3] # two rows, three columns
Out[25]: array([[12, 5, 2],
   [ 7, 6, 8]])
In[26]: x2[:, ::2] # all rows, every other column
Out[26]: array([[12, 2],
   [ 7, 8],
   [ 1, 7]])
```

Finally, subarray dimensions can even be reversed together:

```
In[27]: x2[::-1, ::-1]
Out[27]: array([[ 7,  7,  6,  1],
   [ 8,  8,  6,  7],
   [ 4,  2,  5, 12]])
```

5. Accessing array rows and columns

One commonly needed routine is accessing single rows or columns of an array.

We can do this by combining indexing and slicing, using an empty slice marked by a single colon (:):

```
In[29]: print(x2[0, :]) # first row of x2
[12 5 2 4]
```

In the case of row access, the empty slice can be omitted for a more compact syntax:

```
In[30]: print(x2[0]) # equivalent to x2[0, :]
[12 5 2 4]
```

6. Subarrays as no-copy views

NumPy array slicing differs from Python list slicing: in lists, slices will be copies.

The array slices is that they return views rather than copies of the array data

Consider our two-dimensional array from before:

```
In[31]: print(x2)
```

```
[[12  5  2  4]
 [ 7  6  8  8]
 [ 1  6  7  7]]
```

Let's extract a 2×2 subarray from this:

```
In[32]: x2_sub = x2[:2, :2]
print(x2_sub)
```

```
[[12  5]
 [ 7  6]]
```

Now if we modify this subarray, we'll see that the original array is changed! Observe:

```
In[33]: x2_sub[0, 0] = 99
print(x2_sub)
```

```
[[99  5]
 [ 7  6]]
```

```
In[34]: print(x2)
```

```
[[99  5  2  4]
 [ 7  6  8  8]
 [ 1  6  7  7]]
```

7. Creating copies of arrays

Despite the nice features of array views, it is sometimes useful to instead explicitly copy the data within an array or a subarray.

This can be most easily done with the `copy()` method:

```
In[35]: x2_sub_copy = x2[:2, :2].copy()
print(x2_sub_copy)
```

```
[[99  5]
 [ 7  6]]
```

If we now modify this subarray, the original array is not touched:

```
In[36]: x2_sub_copy[0, 0] = 42
print(x2_sub_copy)
```

```
[[42  5]
 [ 7  6]]
```

```
In[37]: print(x2)
```

```
[[99  5  2  4]
 [ 7  6  8  8]
 [ 1  6  7  7]]
```

8. Reshaping of Arrays

The most flexible way of doing this is with the `reshape()` method.

For example, if we want to put the numbers 1 through 9 in a 3×3 grid, we can do the following:

```
In[38]: grid = np.arange(1, 10).reshape((3, 3))
        print(grid)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Another common reshaping pattern is the conversion of a one-dimensional array into a two-dimensional row or column matrix.

We can do this with the reshape method, or more easily by making use of the newaxis keyword within a slice operation:

```
In[39]: x = np.array([1, 2, 3])

# row vector via reshape
x.reshape((1, 3))
```

```
Out[39]: array([[1, 2, 3]])
```

```
In[40]: # row vector via newaxis
         x[np.newaxis, :]
```

```
Out[40]: array([[1, 2, 3]])
```

```
In[41]: # column vector via reshape
         x.reshape((3, 1))
```

```
Out[41]: array([[1],
 [2],
 [3]])
```

```
In[42]: # column vector via newaxis
         x[:, np.newaxis]
```

```
Out[42]: array([[1],
 [2],
 [3]])
```

9. Array Concatenation and Splitting

It's also possible to combine multiple arrays into one, and to conversely split a single array into multiple arrays.

Concatenation of arrays

Concatenation, or joining of two arrays in NumPy, is primarily accomplished through the routines np.concatenate, np.vstack, and np.hstack. np.concatenate takes a tuple or list of arrays as its first argument

```
In[43]: x = np.array([1, 2, 3])
y = np.array([3, 2, 1])
np.concatenate([x, y])
```

```
Out[43]: array([1, 2, 3, 3, 2, 1])
```

You can also concatenate more than two arrays at once:

```
In[44]: z = [99, 99, 99]
print(np.concatenate([x, y, z]))
```

```
[ 1  2  3  3  2  1 99 99 99]
```

`np.concatenate` can also be used for two-dimensional arrays:

```
In[45]: grid = np.array([[1, 2, 3],
[4, 5, 6]])
```

```
In[46]: # concatenate along the first axis
np.concatenate([grid, grid])
```

```
Out[46]: array([[1, 2, 3],
[4, 5, 6],
[1, 2, 3],
[4, 5, 6]])
```

```
In[47]: # concatenate along the second axis (zero-indexed)
np.concatenate([grid, grid], axis=1)
```

```
Out[47]: array([[1, 2, 3, 1, 2, 3],
[4, 5, 6, 4, 5, 6]])
```

For working with arrays of mixed dimensions, it can be clearer to use the `np.vstack` (vertical stack) and `np.hstack` (horizontal stack) functions:

```
In[48]: x = np.array([1, 2, 3])
grid = np.array([[9, 8, 7],
[6, 5, 4]])
```

```
# vertically stack the arrays
np.vstack([x, grid])
```

```
Out[48]: array([[1, 2, 3],
[9, 8, 7],
[6, 5, 4]])
```

```
In[49]: # horizontally stack the arrays
y = np.array([[99],
[99]])
np.hstack([grid, y])
```

```
Out[49]: array([[ 9,  8,  7, 99],
[ 6,  5,  4, 99]])
```

Similarly, `np.dstack` will stack arrays along the third axis.

Splitting of arrays

The opposite of concatenation is splitting, which is implemented by the functions `np.split`, `np.hsplit`, and `np.vsplit`. For each of these, we can pass a list of indices giving the split points:

```
In[50]: x = [1, 2, 3, 99, 99, 3, 2, 1]
          x1, x2, x3 = np.split(x, [3, 5])
          print(x1, x2, x3)

[1 2 3] [99 99] [3 2 1]
```

Notice that N split points lead to $N + 1$ subarrays. The related functions `np.hsplit` and `np.vsplit` are similar:

```
In[51]: grid = np.arange(16).reshape((4, 4))
          grid
```

```
Out[51]: array([[ 0,  1,  2,  3],
                 [ 4,  5,  6,  7],
                 [ 8,  9, 10, 11],
                 [12, 13, 14, 15]])
```

```
In[52]: upper, lower = np.vsplit(grid, [2])
          print(upper)
          print(lower)
```

```
[[0 1 2 3]
 [4 5 6 7]]
```

```
[[ 8  9 10 11]
 [12 13 14 15]]
```

```
In[53]: left, right = np.hsplit(grid, [2])
          print(left)
          print(right)
```

```
[[ 0  1]
 [ 4  5]
 [ 8  9]
 [12 13]]
[[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]
```

Similarly, `np.dsplit` will split arrays along the third axis

II. Aggregations: Min, Max, and Everything in Between

1. Summing the Values in an Array

As a quick example, consider computing the sum of all values in an array. Python itself can do this using the built-in `sum` function:

```
In[1]: import numpy as np  
In[2]: L = np.random.random(100)  
       sum(L)  
Out[2]: 55.61209116604941
```

The syntax is quite similar to that of NumPy's `sum` function, and the result is the same in the simplest case:

```
In[3]: np.sum(L)  
Out[3]: 55.612091166049424
```

However, because it executes the operation in compiled code, NumPy's version of the operation is computed much more quickly:

```
In[4]: big_array = np.random.rand(1000000)  
       %timeit sum(big_array)  
       %timeit np.sum(big_array)  
  
10 loops, best of 3: 104 ms per loop  
1000 loops, best of 3: 442 µs per loop
```

Be careful, though: the `sum` function and the `np.sum` function are not identical, which can sometimes lead to confusion! In particular, their optional arguments have different meanings, and `np.sum` is aware of multiple array dimensions, as we will see in the following section.

2. Minimum and Maximum

Similarly, Python has built-in `min` and `max` functions, used to find the minimum value and maximum value of any given array:

```
In[5]: min(big_array), max(big_array)  
Out[5]: (1.1717128136634614e-06, 0.9999976784968716)
```

NumPy's corresponding functions have similar syntax, and again operate much more quickly:

```
In[6]: np.min(big_array), np.max(big_array)  
Out[6]: (1.1717128136634614e-06, 0.9999976784968716)  
  
In[7]: %timeit min(big_array)  
       %timeit np.min(big_array)  
  
10 loops, best of 3: 82.3 ms per loop  
1000 loops, best of 3: 497 µs per loop
```

For `min`, `max`, `sum`, and several other NumPy aggregates, a shorter syntax is to use methods of the array object itself:

```
In[8]: print(big_array.min(), big_array.max(), big_array.sum())  
1.17171281366e-06 0.999997678497 499911.628197
```

Whenever possible, make sure that you are using the NumPy version of these aggregates when operating on NumPy arrays!

3. Multidimensional aggregates

One common type of aggregation operation is an aggregate along a row or column. Say you have some data stored in a two-dimensional array:

```
In[9]: M = np.random.random((3, 4))
print(M)

[[ 0.8967576  0.03783739  0.75952519  0.06682827]
 [ 0.8354065  0.99196818  0.19544769  0.43447084]
 [ 0.66859307 0.15038721  0.37911423  0.6687194 ]]
```

By default, each NumPy aggregation function will return the aggregate over the entire array:

```
In[10]: M.sum()

Out[10]: 6.0850555667307118
```

Aggregation functions take an additional argument specifying the *axis* along which the aggregate is computed. For example, we can find the minimum value within each column by specifying `axis=0`:

```
In[11]: M.min(axis=0)

Out[11]: array([ 0.66859307,  0.03783739,  0.19544769,  0.06682827])
```

The function returns four values, corresponding to the four columns of numbers.

Similarly, we can find the maximum value within each row:

```
In[12]: M.max(axis=1)

Out[12]: array([ 0.8967576 ,  0.99196818,  0.6687194 ])
```

Other aggregation functions

Table 2-3. Aggregation functions available in NumPy

Function Name	NaN-safe Version	Description
<code>np.sum</code>	<code>np.nansum</code>	Compute sum of elements
<code>np.prod</code>	<code>np.nanprod</code>	Compute product of elements
<code>np.mean</code>	<code>np.nanmean</code>	Compute median of elements
<code>np.std</code>	<code>np.nanstd</code>	Compute standard deviation
<code>np.var</code>	<code>np.nanvar</code>	Compute variance
<code>np.min</code>	<code>np.nanmin</code>	Find minimum value
<code>np.max</code>	<code>np.nanmax</code>	Find maximum value
<code>np.argmax</code>	<code>np.nanargmin</code>	Find index of minimum value
<code>np.argmax</code>	<code>np.nanargmax</code>	Find index of maximum value
<code>np.median</code>	<code>np.nanmedian</code>	Compute median of elements
<code>np.percentile</code>	<code>np.nanpercentile</code>	Compute rank-based statistics of elements
<code>np.any</code>	N/A	Evaluate whether any elements are true
<code>np.all</code>	N/A	Evaluate whether all elements are true

Example: What Is the Average Height of US Presidents?

Aggregates available in NumPy can be extremely useful for summarizing a set of values.

As a simple example, let's consider the heights of all US presidents.

This data is available in the file `president_heights.csv`, which is a simple comma-separated list of labels and values:

```
In[13]: !head -4 data/president_heights.csv
order,name,height(cm)
1,George Washington,189
2,John Adams,170
3,Thomas Jefferson,189
```

We'll use the Pandas package, which we'll explore more fully in [Chapter 3](#), to read the file and extract this information (note that the heights are measured in centimeters):

```
In[14]: import pandas as pd
data = pd.read_csv('data/president_heights.csv')
heights = np.array(data['height(cm)'])
print(heights)

[189 170 189 163 183 171 185 168 173 183 173 173 175 178 183 193 178 173
 174 183 183 168 170 178 182 180 183 178 182 188 175 179 183 193 182 183
 177 185 188 188 182 185]
```

Now that we have this data array, we can compute a variety of summary statistics:

```
In[15]: print("Mean height:      ", heights.mean())
print("Standard deviation: ", heights.std())
print("Minimum height:    ", heights.min())
print("Maximum height:    ", heights.max())

Mean height:      179.738095238
Standard deviation: 6.93184344275
Minimum height:    163
Maximum height:    193
```

Note that in each case, the aggregation operation reduced the entire array to a single summarizing value, which gives us information about the distribution of values. We may also wish to compute quantiles:

```
In[16]: print("25th percentile:   ", np.percentile(heights, 25))
print("Median:           ", np.median(heights))
print("75th percentile:   ", np.percentile(heights, 75))

25th percentile:   174.25
Median:           182.0
75th percentile:   183.0
```

We see that the median height of US presidents is 182 cm, or just shy of six feet.

Of course, sometimes it's more useful to see a visual representation of this data, which we can accomplish using tools in Matplotlib (we'll discuss Matplotlib more fully in [Chapter 4](#)). For example, this code generates the chart shown in [Figure 2-3](#):

```
In[17]: %matplotlib inline
import matplotlib.pyplot as plt
import seaborn; seaborn.set() # set plot style

In[18]: plt.hist(heights)
plt.title('Height Distribution of US Presidents')
plt.xlabel('height (cm)')
plt.ylabel('number');
```

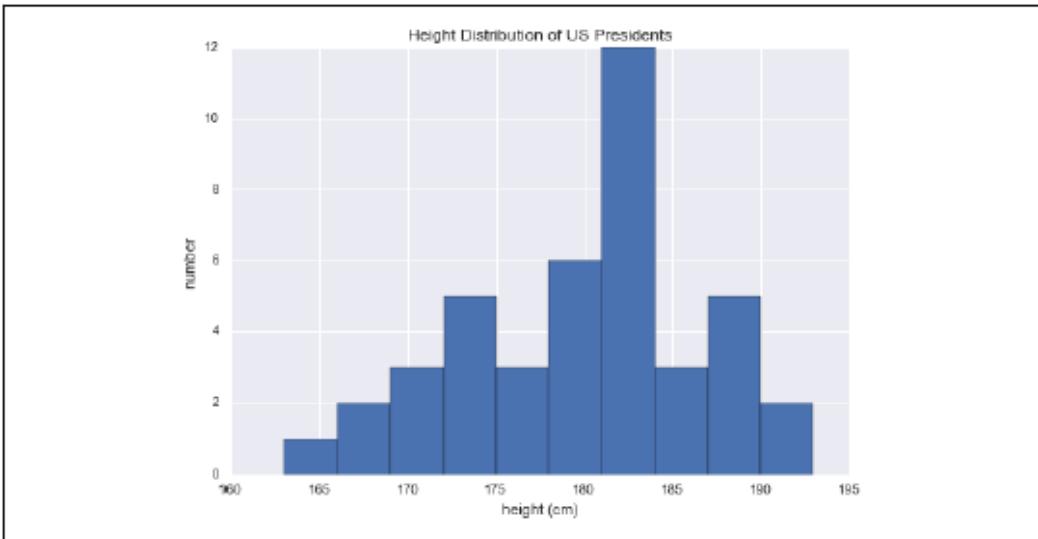


Figure 2-3. Histogram of presidential heights

III. Computation on Arrays: Broadcasting

Another means of vectorizing operations is to use NumPy's broadcasting functionality. Broadcasting is simply a set of rules for applying binary ufuncs (addition, subtraction, multiplication, etc.) on arrays of different sizes.

Introducing Broadcasting

Recall that for arrays of the same size, binary operations are performed on an element-by-element basis: Broadcasting allows these types of binary operations to be performed on arrays of different sizes—for example, we can just as easily add a scalar (think of it as a zero dimensional array) to an array:

```
In[1]: import numpy as np  
In[2]: a = np.array([0, 1, 2])  
       b = np.array([5, 5, 5])  
       a + b  
Out[2]: array([5, 6, 7])  
In[3]: a + 5  
Out[3]: array([5, 6, 7])
```

We can think of this as an operation that stretches or duplicates the value 5 into the array [5, 5, 5], and adds the results. The advantage of NumPy's broadcasting is that this duplication of values does not actually take place, but it is a useful mental model as we think about broadcasting.

We can similarly extend this to arrays of higher dimension. Observe the result when we add a one-dimensional array to a two-dimensional array:

```
In[4]: M = np.ones((3, 3))
M

Out[4]: array([[ 1.,  1.,  1.],
               [ 1.,  1.,  1.],
               [ 1.,  1.,  1.]])
```



```
In[5]: M + a

Out[5]: array([[ 1.,  2.,  3.],
               [ 1.,  2.,  3.],
               [ 1.,  2.,  3.]])
```

Here the one-dimensional array `a` is stretched, or broadcast, across the second dimension in order to match the shape of `M`.

While these examples are relatively easy to understand, more complicated cases can involve broadcasting of both arrays. Consider the following example:

```
In[6]: a = np.arange(3)
b = np.arange(3)[:, np.newaxis]

print(a)
print(b)

[0 1 2]
[[0]
 [1]
 [2]]
```



```
In[7]: a + b

Out[7]: array([[0, 1, 2],
               [1, 2, 3],
               [2, 3, 4]])
```

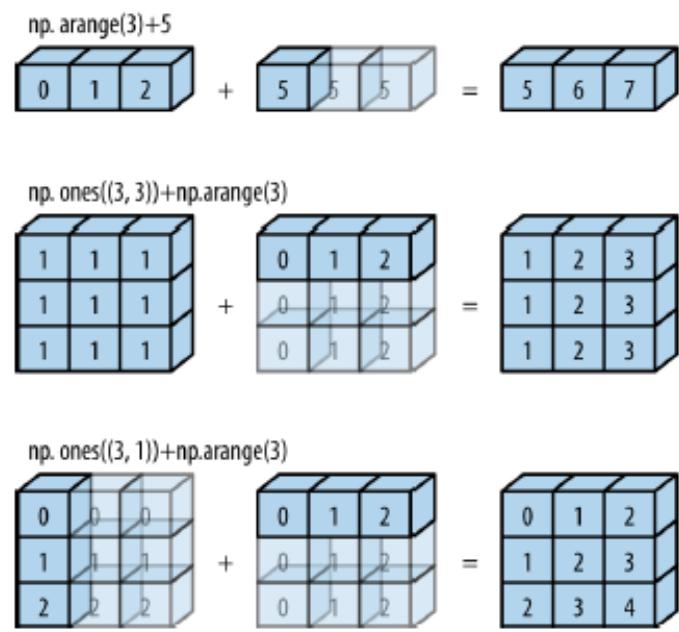


Figure 2-4. Visualization of NumPy broadcasting

The light boxes represent the broadcasted values: again, this extra memory is not actually allocated in the course of the operation, but it can be useful conceptually to imagine that it is.

Rules of Broadcasting

Broadcasting in NumPy follows a strict set of rules to determine the interaction between the two arrays:

- Rule 1: If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded with ones on its leading (left) side.
- Rule 2: If the shape of the two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
- Rule 3: If in any dimension the sizes disagree and neither is equal to 1, an error is raised.

To make these rules clear, let's consider a few examples in detail.

Broadcasting example 1

Let's look at adding a two-dimensional array to a one-dimensional array:

```
In[8]: M = np.ones((2, 3))
      a = np.arange(3)
```

Let's consider an operation on these two arrays. The shapes of the arrays are:

```
M.shape = (2, 3)
a.shape = (3,)
```

We see by rule 1 that the array `a` has fewer dimensions, so we pad it on the left with ones:

```
M.shape -> (2, 3)
a.shape -> (1, 3)
```

By rule 2, we now see that the first dimension disagrees, so we stretch this dimension to match:

```
M.shape -> (2, 3)
a.shape -> (2, 3)
```

The shapes match, and we see that the final shape will be `(2, 3)`:

```
In[9]: M + a
Out[9]: array([[ 1.,  2.,  3.],
               [ 1.,  2.,  3.]])
```

Broadcasting example 2

Let's take a look at an example where both arrays need to be broadcast:

```
In[10]: a = np.arange(3).reshape((3, 1))
      b = np.arange(3)
```

Again, we'll start by writing out the shape of the arrays:

```
a.shape = (3, 1)
b.shape = (3,)
```

Rule 1 says we must pad the shape of `b` with ones:

```
a.shape -> (3, 1)
b.shape -> (1, 3)
```

And rule 2 tells us that we upgrade each of these ones to match the corresponding size of the other array:

```
a.shape -> (3, 3)
b.shape -> (3, 3)
```

Because the result matches, these shapes are compatible. We can see this here:

```
In[11]: a + b
Out[11]: array([[0, 1, 2],
                 [1, 2, 3],
                 [2, 3, 4]])
```

Broadcasting example 3

Now let's take a look at an example in which the two arrays are not compatible:

```
In[12]: M = np.ones((3, 2))
a = np.arange(3)
```

This is just a slightly different situation than in the first example: the matrix `M` is transposed. How does this affect the calculation? The shapes of the arrays are:

```
M.shape = (3, 2)
a.shape = (3,)
```

Again, rule 1 tells us that we must pad the shape of `a` with ones:

```
M.shape -> (3, 2)
a.shape -> (1, 3)
```

By rule 2, the first dimension of `a` is stretched to match that of `M`:

```
M.shape -> (3, 2)
a.shape -> (3, 3)
```

Now we hit rule 3—the final shapes do not match, so these two arrays are incompatible, as we can observe by attempting this operation:

Broadcasting in Practice

Broadcasting operations form the core of many examples we'll see throughout this book. We'll now take a look at a couple simple examples of where they can be useful.

Centering an array

In the previous section, we saw that ufuncs allow a NumPy user to remove the need to explicitly write slow Python loops. Broadcasting extends this ability. One commonly seen example is centering an array of data. Imagine you have an array of 10 observations, each of which consists of 3 values. Using the standard convention (see “Data Representation in Scikit-Learn” on page 343), we'll store this in a 10×3 array:

```
In[17]: X = np.random.random((10, 3))
```

We can compute the mean of each feature using the `mean` aggregate across the first dimension:

```
In[18]: Xmean = X.mean(0)
```

```
Xmean
```

```
Out[18]: array([ 0.53514715,  0.66567217,  0.44385899])
```

And now we can center the `X` array by subtracting the mean (this is a broadcasting operation):

```
In[19]: X_centered = X - Xmean
```

To double-check that we've done this correctly, we can check that the centered array has near zero mean:

```
In[20]: X_centered.mean(0)
```

```
Out[20]: array([ 2.22044605e-17, -7.77156117e-17, -1.66533454e-17])
```

To within-machine precision, the mean is now zero.

Plotting a two-dimensional function

One place that broadcasting is very useful is in displaying images based on two-dimensional functions. If we want to define a function $z = f(x, y)$, broadcasting can be used to compute the function across the grid:

```
In[21]: # x and y have 50 steps from 0 to 5
x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 50)[:, np.newaxis]

z = np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)
```

We'll use Matplotlib to plot this two-dimensional array (these tools will be discussed in full in “Density and Contour Plots” on page 241):

```
In[22]: %matplotlib inline
import matplotlib.pyplot as plt
```

```
In[23]: plt.imshow(z, origin='lower', extent=[0, 5, 0, 5],
                  cmap='viridis')
plt.colorbar();
```

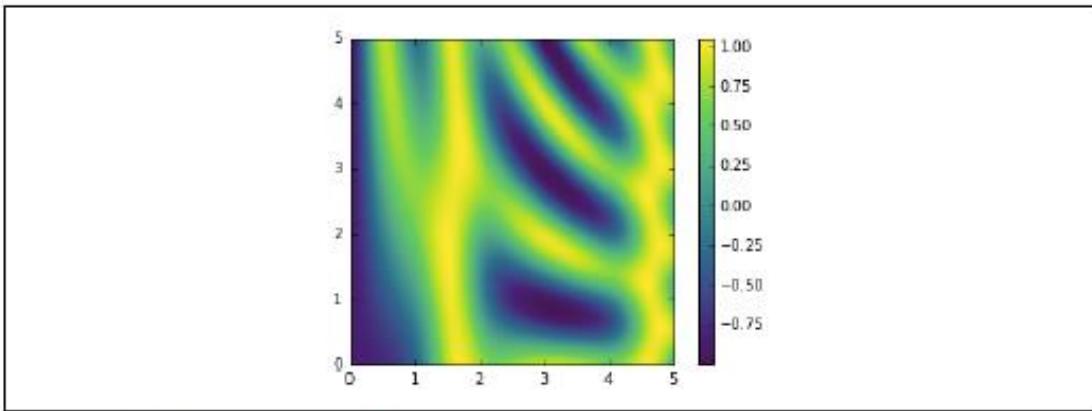


Figure 2-5. Visualization of a 2D array

IV. Comparisons, Masks, and Boolean Logic

This section covers the use of Boolean masks to examine and manipulate values within NumPy arrays. Masking comes up when you want to extract, modify, count, or otherwise manipulate values in an array based on some criterion: for example, you might wish to count all values greater than a certain value, or perhaps remove all outliers that are above some threshold.

In NumPy, Boolean masking is often the most efficient way to accomplish these types of tasks.

Example: Counting Rainy Days

Imagine you have a series of data that represents the amount of precipitation each day for a year in a given city. For example, here we'll load the daily rainfall statistics for the city of Seattle in 2014, using Pandas (which is covered in more detail in [Chapter 3](#)):

```
In[1]: import numpy as np
        import pandas as pd

        # use Pandas to extract rainfall inches as a NumPy array
        rainfall = pd.read_csv('data/Seattle2014.csv')['PRCP'].values
        inches = rainfall / 254 # 1/10mm -> inches
        inches.shape

Out[1]: (365,)
```

The array contains 365 values, giving daily rainfall in inches from January 1 to December 31, 2014.

```
In[2]: %matplotlib inline  
import matplotlib.pyplot as plt  
import seaborn; seaborn.set() # set plot styles
```

```
In[3]: plt.hist(inches, 40);
```

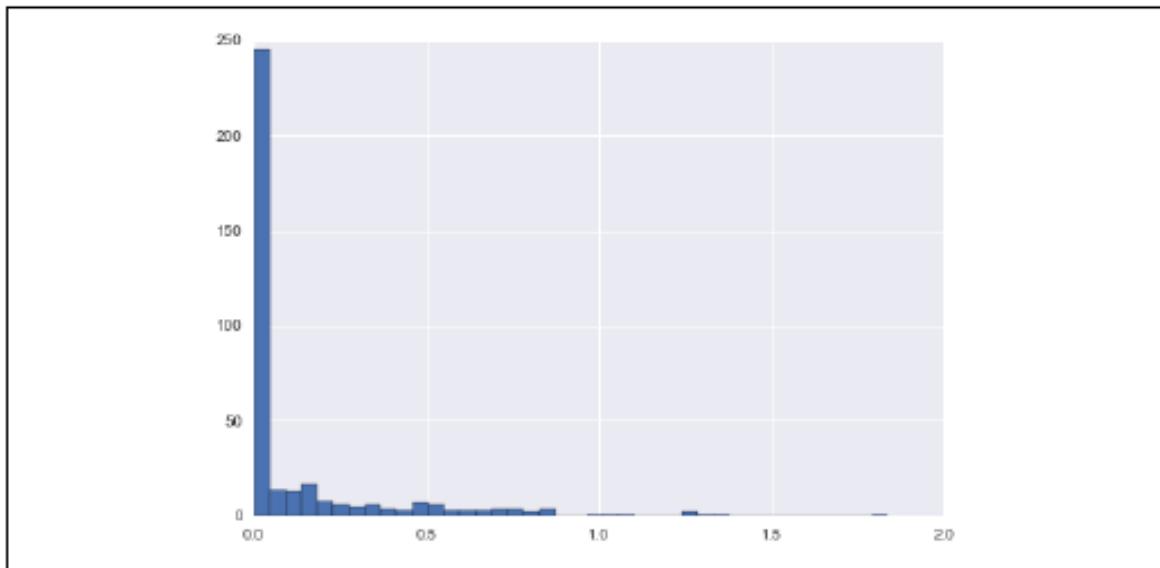


Figure 2-6. Histogram of 2014 rainfall in Seattle

One approach to this would be to answer these questions by hand: loop through the data, incrementing a counter each time we see values in some desired range.

For reasons discussed throughout this chapter, such an approach is very inefficient, both from the standpoint of time writing code and time computing the result.

Comparison Operators as ufuncs

NumPy also implements comparison operators such as `<` (less than) and `>` (greater than) as element-wise ufuncs.

The result of these comparison operators is always an array with a Boolean data type.

All six of the standard comparison operations are available:

```
In[4]: x = np.array([1, 2, 3, 4, 5])
In[5]: x < 3 # less than
Out[5]: array([ True,  True, False, False, False], dtype=bool)
In[6]: x > 3 # greater than
Out[6]: array([False, False, False,  True,  True], dtype=bool)
In[7]: x <= 3 # less than or equal
Out[7]: array([ True,  True,  True, False, False], dtype=bool)
In[8]: x >= 3 # greater than or equal
Out[8]: array([False, False,  True,  True,  True], dtype=bool)
In[9]: x != 3 # not equal
Out[9]: array([ True,  True, False,  True,  True], dtype=bool)
In[10]: x == 3 # equal
Out[10]: array([False,  True, False, False, False], dtype=bool)
```

It is also possible to do an element-by-element comparison of two arrays, and to include compound expressions:

```
In[11]: (2 * x) == (x ** 2)
Out[11]: array([False,  True, False, False, False], dtype=bool)
```

As in the case of arithmetic operators, the comparison operators are implemented as ufuncs in NumPy; for example, when you write `x < 3`, internally NumPy uses `np.less(x, 3)`. A summary of the comparison operators and their equivalent ufunc is shown here:

Operator	Equivalent ufunc
<code>==</code>	<code>np.equal</code>
<code>!=</code>	<code>np.not_equal</code>
<code><</code>	<code>np.less</code>
<code><=</code>	<code>np.less_equal</code>
<code>></code>	<code>np.greater</code>
<code>>=</code>	<code>np.greater_equal</code>

Working with Boolean Arrays

Given a Boolean array, there are a host of useful operations you can do. We'll work with `x`, the two-dimensional array we created earlier:

```
In[14]: print(x)
```

```
[[5 0 3 3]
 [7 9 3 5]
 [2 4 7 6]]
```

Counting entries

To count the number of True entries in a Boolean array, `np.count_nonzero` is useful:

```
In[15]: # how many values less than 6?
 np.count_nonzero(x < 6)
```

```
Out[15]: 8
```

We see that there are eight array entries that are less than 6. Another way to get at this information is to use `np.sum`; in this case, `False` is interpreted as 0, and `True` is interpreted as 1:

```
In[16]: np.sum(x < 6)
```

```
Out[16]: 8
```

The benefit of `sum()` is that like with other NumPy aggregation functions, this summation can be done along rows or columns as well:

```
In[17]: # how many values less than 6 in each row?
 np.sum(x < 6, axis=1)
```

```
Out[17]: array([4, 2, 2])
```

This counts the number of values less than 6 in each row of the matrix.

If we're interested in quickly checking whether any or all the values are true, we can use (you guessed it) `np.any()` or `np.all()`:

```
In[18]: # are there any values greater than 8?  
       np.any(x > 8)  
  
Out[18]: True  
  
In[19]: # are there any values less than zero?  
       np.any(x < 0)  
  
Out[19]: False  
  
In[20]: # are all values less than 10?  
       np.all(x < 10)  
  
Out[20]: True  
  
In[21]: # are all values equal to 6?  
       np.all(x == 6)  
  
Out[21]: False
```

`np.all()` and `np.any()` can be used along particular axes as well. For example:

```
In[22]: # are all values in each row less than 8?  
       np.all(x < 8, axis=1)  
  
Out[22]: array([ True, False,  True], dtype=bool)
```

Here all the elements in the first and third rows are less than 8, while this is not the case for the second row.

Boolean operators

NumPy overloads these as ufuncs that work element-wise on (usually Boolean) arrays.

For example, we can address this sort of compound question as follows:

```
In[23]: np.sum((inches > 0.5) & (inches < 1))  
Out[23]: 29
```

So we see that there are 29 days with rainfall between 0.5 and 1.0 inches.

Using the equivalence of *A AND B* and *NOT (A OR B)* (which you may remember if you've taken an introductory logic course), we can compute the same result in a different manner:

```
In[24]: np.sum(~( (inches <= 0.5) | (inches >= 1) ))  
Out[24]: 29
```

Combining comparison operators and Boolean operators on arrays can lead to a wide range of efficient logical operations.

The following table summarizes the bitwise Boolean operators and their equivalent ufuncs:

Operator	Equivalent ufunc
&	<code>np.bitwise_and</code>
	<code>np.bitwise_or</code>
^	<code>np.bitwise_xor</code>
~	<code>np.bitwise_not</code>

Boolean Arrays as Masks:

We looked at aggregates computed directly on Boolean arrays.

A more powerful pattern is to use Boolean arrays as masks, to select particular subsets of the data themselves. Returning to our `x` array from before, suppose we want an array of all values in the array that are less than, say, 5:

```
In[26]: x  
Out[26]: array([[5, 0, 3, 3],  
                 [7, 9, 3, 5],  
                 [2, 4, 7, 6]])
```

We can obtain a Boolean array for this condition easily, as we've already seen:

```
In[27]: x < 5  
Out[27]: array([[False, True, True, True],  
                 [False, False, True, False],  
                 [True, True, False, False]], dtype=bool)
```

Now to *select* these values from the array, we can simply index on this Boolean array; this is known as a *masking* operation:

```
In[28]: x[x < 5]  
Out[28]: array([0, 3, 3, 3, 2, 4])
```

What is returned is a one-dimensional array filled with all the values that meet this condition; in other words, all the values in positions at which the mask array is True.

Using the Keywords and/or Versus the Operators &/|

One common point of confusion is the difference between the keywords `and` and `or` on one hand, and the operators `&` and `|` on the other hand. When would you use one versus the other?

The difference is this: `and` and `or` gauge the truth or falsehood of *entire object*, while `&` and `|` refer to *bits within each object*.

When you use `and` or `or`, it's equivalent to asking Python to treat the object as a single Boolean entity. In Python, all nonzero integers will evaluate as True. Thus:

```
In[30]: bool(42), bool(0)  
Out[30]: (True, False)  
  
In[31]: bool(42 and 0)  
Out[31]: False  
  
In[32]: bool(42 or 0)  
Out[32]: True
```

IV. Fancy Indexing:

We'll look at another style of array indexing, known as fancy indexing.
Fancy indexing is like the simple indexing we've already seen, but we pass arrays of indices in place of single scalars.
This allows us to very quickly access and modify complicated subsets of an array's values.

Exploring Fancy Indexing

Exploring Fancy Indexing

Fancy indexing is conceptually simple: it means passing an array of indices to access multiple array elements at once. For example, consider the following array:

```
In[1]: import numpy as np  
rand = np.random.RandomState(42)  
  
x = rand.randint(100, size=10)  
print(x)  
  
[51 92 14 71 60 20 82 86 74 74]
```

Suppose we want to access three different elements. We could do it like this:

```
In[2]: [x[3], x[7], x[2]]  
Out[2]: [71, 86, 14]
```

Alternatively, we can pass a single list or array of indices to obtain the same result:

```
In[3]: ind = [3, 7, 4]  
x[ind]  
  
Out[3]: array([71, 86, 60])
```

With fancy indexing, the shape of the result reflects the shape of the *index arrays* rather than the shape of the *array being indexed*:

```
In[4]: ind = np.array([[3, 7],  
                      [4, 5]])  
x[ind]  
  
Out[4]: array([[71, 86],  
              [60, 20]])
```

Fancy indexing also works in multiple dimensions. Consider the following array:

```
In[5]: X = np.arange(12).reshape((3, 4))  
X  
  
Out[5]: array([[ 0,  1,  2,  3],  
              [ 4,  5,  6,  7],  
              [ 8,  9, 10, 11]])
```

Like with standard indexing, the first index refers to the row, and the second to the column:

```
In[6]: row = np.array([0, 1, 2])  
col = np.array([2, 1, 3])  
X[row, col]  
  
Out[6]: array([ 2,  5, 11])
```

for example, if we combine a column vector and a row vector within the indices, we get a two-dimensional result:

```
In[7]: X[row[:, np.newaxis], col]  
Out[7]: array([[ 2,  1,  3],  
               [ 6,  5,  7],  
               [10,  9, 11]])
```

Here, each row value is matched with each column vector, exactly as we saw in broadcasting of arithmetic operations. For example:

```
In[8]: row[:, np.newaxis] * col  
Out[8]: array([[ 0,  0,  0],  
               [ 2,  1,  3],  
               [ 4,  2,  6]])
```

It is always important to remember with fancy indexing that the return value reflects the *broadcasted shape of the indices*, rather than the shape of the array being indexed.

Combined Indexing

For even more powerful operations, fancy indexing can be combined with the other indexing schemes we've seen:

```
In[9]: print(X)  
[[ 0  1  2  3]  
 [ 4  5  6  7]  
 [ 8  9 10 11]]
```

We can combine fancy and simple indices:

```
In[10]: X[2, [2, 0, 1]]  
Out[10]: array([10,  8,  9])
```

We can also combine fancy indexing with slicing:

```
In[11]: X[1:, [2, 0, 1]]  
Out[11]: array([[ 6,  4,  5],  
               [10,  8,  9]])
```

And we can combine fancy indexing with masking:

```
In[12]: mask = np.array([1, 0, 1, 0], dtype=bool)  
        X[row[:, np.newaxis], mask]  
  
Out[12]: array([[ 0,  2],  
                  [ 4,  6],  
                  [ 8, 10]])
```

All of these indexing options combined lead to a very flexible set of operations for accessing and modifying array values.

Example: Selecting Random Points

One common use of fancy indexing is the selection of subsets of rows from a matrix. For example, we might have an N by D matrix representing N points in D dimensions, such as the following points drawn from a two-dimensional normal distribution:

```
In[13]: mean = [0, 0]
cov = [[1, 2],
       [2, 5]]
X = rand.multivariate_normal(mean, cov, 100)
X.shape
```

```
Out[13]: (100, 2)
```

Using the plotting tools we will discuss in [Chapter 4](#), we can visualize these points as a scatter plot ([Figure 2-7](#)):

```
In[14]: %matplotlib inline
import matplotlib.pyplot as plt
import seaborn; seaborn.set() # for plot styling

plt.scatter(X[:, 0], X[:, 1]);
```



Figure 2-7. Normally distributed points

Let's use fancy indexing to select 20 random points. We'll do this by first choosing 20 random indices with no repeats, and use these indices to select a portion of the original array:

```
In[15]: indices = np.random.choice(X.shape[0], 20, replace=False)
indices
```

```
Out[15]: array([93, 45, 73, 81, 50, 10, 98, 94, 4, 64, 65, 89, 47, 84, 82,
               80, 25, 90, 63, 20])
```

```
In[16]: selection = X[:, 0] # fancy indexing here  
selection.shape
```

```
Out[16]: (20, 2)
```

Now to see which points were selected, let's over-plot large circles at the locations of the selected points (Figure 2-8):

```
In[17]: plt.scatter(X[:, 0], X[:, 1], alpha=0.3)  
plt.scatter(selection[:, 0], selection[:, 1],  
            facecolor='none', s=200);
```

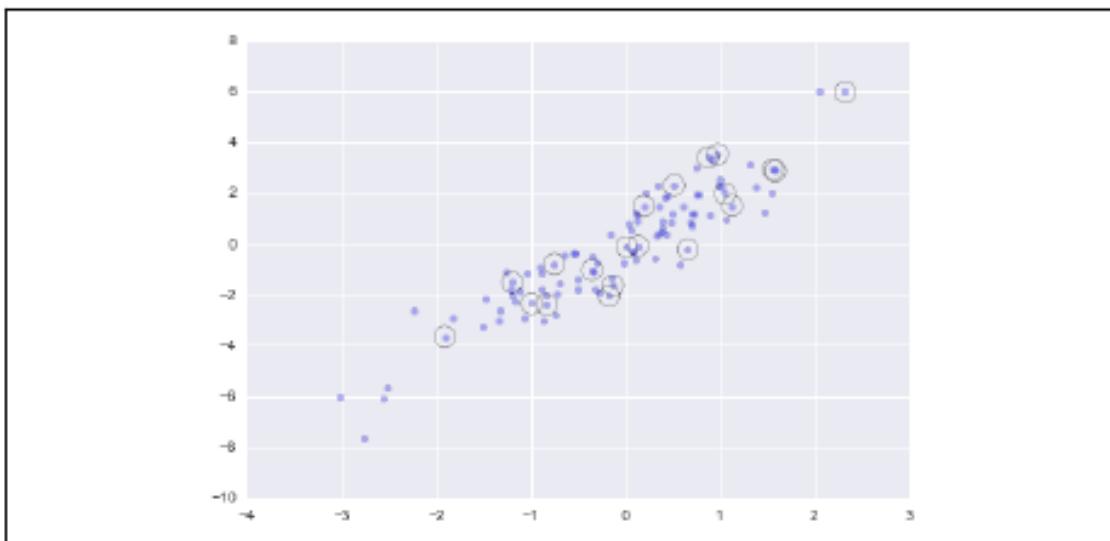


Figure 2-8. Random selection among points

This sort of strategy is often used to quickly partition datasets, as is often needed in train/test splitting for validation of statistical models (see “[Hyperparameters and Model Validation](#)” on page 359), and in sampling approaches to answering statistical questions.

Modifying Values with Fancy Indexing

Just as fancy indexing can be used to access parts of an array, it can also be used to modify parts of an array. For example, imagine we have an array of indices and we'd like to set the corresponding items in an array to some value:

```
In[18]: x = np.arange(10)  
i = np.array([2, 1, 8, 4])  
x[i] = 99  
print(x)
```



```
[ 0 99 99  3 99  5  6  7 99  9]
```

We can use any assignment-type operator for this. For example:

```
In[19]: x[i] -= 10
        print(x)

[ 0 89 89  3 89  5  6  7 89  9]
```

Notice, though, that repeated indices with these operations can cause some potentially unexpected results. Consider the following:

```
In[20]: x = np.zeros(10)
        x[[0, 0]] = [4, 6]
        print(x)

[ 6.  0.  0.  0.  0.  0.  0.  0.]
```

Where did the 4 go? The result of this operation is to first assign $x[0] = 4$, followed by $x[0] = 6$. The result, of course, is that $x[0]$ contains the value 6.

Fair enough, but consider this operation:

```
In[21]: i = [2, 3, 3, 4, 4, 4]
        x[i] += 1
        x

Out[21]: array([ 6.,  0.,  1.,  1.,  1.,  0.,  0.,  0.,  0.])
```

Example: Binning Data

```
In[23]: np.random.seed(42)
        x = np.random.randn(100)

        # compute a histogram by hand
        bins = np.linspace(-5, 5, 20)
        counts = np.zeros_like(bins)

        # find the appropriate bin for each x
        i = np.searchsorted(bins, x)

        # add 1 to each of these bins
        np.add.at(counts, i, 1)
```

The counts now reflect the number of points within each bin—in other words, a histogram (Figure 2-9):

```
In[24]: # plot the results
        plt.plot(bins, counts, linestyle='steps');
```

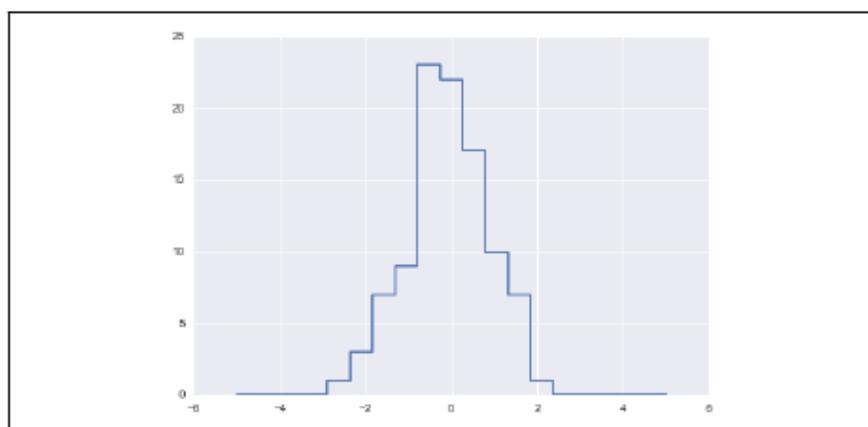


Figure 2-9. A histogram computed by hand

```
In[26]: x = np.random.randn(1000000)
        print("NumPy routine:")
        %timeit counts, edges = np.histogram(x, bins)

        print("Custom routine:")
        %timeit np.add.at(counts, np.searchsorted(bins, x), 1)

NumPy routine:
10 loops, best of 3: 68.7 ms per loop
Custom routine:
10 loops, best of 3: 135 ms per loop
```

V. Structured Data: NumPy's Structured Arrays

This section demonstrates the use of NumPy's structured arrays and record arrays, which provide efficient storage for compound, heterogeneous data.

Imagine that we have several categories of data on a number of people (say, name, age, and weight), and we'd like to store these values for use in a Python program. It would be possible to store these in three separate arrays:

```
In[2]: name = ['Alice', 'Bob', 'Cathy', 'Doug']
age = [25, 45, 37, 19]
weight = [55.0, 85.5, 68.0, 61.5]
```

We can similarly create a structured array using a compound data type specification:

```
In[4]: # Use a compound data type for structured arrays
        data = np.zeros(4, dtype={'names':('name', 'age', 'weight'),
                                'formats':('U10', 'i4', 'f8')})
        print(data.dtype)
[(('name', '<U10'), ('age', 'i4'), ('weight', 'f8'))]
```

Now that we've created an empty container array, we can fill the array with our lists of values:

```
In[5]: data['name'] = name
        data['age'] = age
        data['weight'] = weight
        print(data)

[('Alice', 25, 55.0) ('Bob', 45, 85.5) ('Cathy', 37, 68.0)
 ('Doug', 19, 61.5)]

In[7]: # Get first row of data
        data[0]

Out[7]: ('Alice', 25, 55.0)

In[8]: # Get the name from the last row
        data[-1]['name']

Out[8]: 'Doug'
```

Using Boolean masking, this even allows you to do some more sophisticated operations such as filtering on age:

```
In[9]: # Get names where age is under 30
        data[data['age'] < 30]['name']

Out[9]: array(['Alice', 'Doug'],
              dtype='<U10')
```

Creating Structured Arrays

Structured array data types can be specified in a number of ways. Earlier, we saw the dictionary method:

```
In[10]: np.dtype({'names':('name', 'age', 'weight'),
                 'formats':('U10', 'i4', 'f8')})  
Out[10]: dtype([('name', '<U10'), ('age', 'i4'), ('weight', 'f8')])
```

For clarity, numerical types can be specified with Python types or NumPy dtypes instead:

```
In[11]: np.dtype({'names':('name', 'age', 'weight'),
                 'formats':((np.str_, 10), int, np.float32)})  
Out[11]: dtype([('name', '<U10'), ('age', 'i4'), ('weight', 'f4')])
```

A compound type can also be specified as a list of tuples:

```
In[12]: np.dtype([('name', 'S10'), ('age', 'i4'), ('weight', 'f8')])  
Out[12]: dtype([('name', 'S10'), ('age', 'i4'), ('weight', 'f8')])
```

If the names of the types do not matter to you, you can specify the types alone in a comma-separated string:

```
In[13]: np.dtype('S10,i4,f8')
```

Table 2-4. NumPy data types

Character	Description	Example
'b'	Byte	<code>np.dtype('b')</code>
'i'	Signed integer	<code>np.dtype('i4') == np.int32</code>
'u'	Unsigned integer	<code>np.dtype('u1') == np.uint8</code>
'f'	Floating point	<code>np.dtype('f8') == np.int64</code>
'c'	Complex floating point	<code>np.dtype('c16') == np.complex128</code>
'S', 'a'	string	<code>np.dtype('S5')</code>
'U'	Unicode string	<code>np.dtype('U') == np.str_</code>
'V'	Raw data (void)	<code>np.dtype('V') == np.void</code>

More Advanced Compound Types

It is possible to define even more advanced compound types. For example, you can create a type where each element contains an array or matrix of values. Here, we'll create a data type with a `mat` component consisting of a 3×3 floating-point matrix:

```
In[14]: tp = np.dtype([('id', 'i8'), ('mat', 'f8', (3, 3))])
X = np.zeros(1, dtype=tp)
print(X[0])
print(X['mat'][0])

(0, [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]])
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]])
```

RecordArrays: Structured Arrays with a Twist

NumPy also provides the `np.recarray` class, which is almost identical to the structured arrays just described, but with one additional feature: fields can be accessed as attributes rather than as dictionary keys. Recall that we previously accessed the ages by writing:

```
In[15]: data['age']
Out[15]: array([25, 45, 37, 19], dtype=int32)
```

If we view our data as a record array instead, we can access this with slightly fewer keystrokes:

```
In[16]: data_rec = data.view(np.recarray)
data_rec.age
Out[16]: array([25, 45, 37, 19], dtype=int32)
```

The downside is that for record arrays, there is some extra overhead involved in accessing the fields, even when using the same syntax. We can see this here:

```
In[17]: %timeit data['age']
%timeit data_rec['age']
%timeit data_rec.age

1000000 loops, best of 3: 241 ns per loop
100000 loops, best of 3: 4.61 µs per loop
100000 loops, best of 3: 7.27 µs per loop
```

Whether the more convenient notation is worth the additional overhead will depend on your own application.

VI. Data Manipulation with Pandas

Data Indexing and Selection

Data Selection in Series

A Series object acts in many ways like a one-dimensional NumPy array, and in many ways like a standard Python dictionary.

Series as dictionary

Like a dictionary, the Series object provides a mapping from a collection of keys to a collection of values:

```
In[1]: import pandas as pd  
       data = pd.Series([0.25, 0.5, 0.75, 1.0],  
                        index=['a', 'b', 'c', 'd'])  
       data
```

```
Out[1]: a    0.25  
        b    0.50  
        c    0.75  
        d    1.00  
       dtype: float64
```

```
In[2]: data['b']
```

```
Out[2]: 0.5
```

We can also use dictionary-like Python expressions and methods to examine the keys/indices and values:

```
In[3]: 'a' in data  
Out[3]: True  
In[4]: data.keys()  
Out[4]: Index(['a', 'b', 'c', 'd'], dtype='object')  
In[5]: list(data.items())  
Out[5]: [('a', 0.25), ('b', 0.5), ('c', 0.75), ('d', 1.0)]
```

Series objects can even be modified with a dictionary-like syntax. Just as you can extend a dictionary by assigning to a new key, you can extend a Series by assigning to a new index value:

```
In[6]: data['e'] = 1.25  
       data  
Out[6]: a    0.25  
        b    0.50  
        c    0.75  
        d    1.00  
        e    1.25  
       dtype: float64
```

Series as one-dimensional array

A Series builds on this dictionary-like interface and provides array-style item selection via the same basic mechanisms as NumPy arrays—that is, slices, masking, and fancy indexing. Examples of these are as follows:

```
In[8]: # slicing by implicit integer index  
       data[0:2]
```

```
Out[8]: a    0.25  
        b    0.50  
       dtype: float64
```

```
In[9]: # masking  
       data[(data > 0.3) & (data < 0.8)]
```

```
Out[9]: b    0.50  
        c    0.75  
       dtype: float64
```

```
In[10]: # fancy indexing  
       data[['a', 'e']]
```

```
Out[10]: a    0.25  
        e    1.25  
       dtype: float64
```

Indexers: loc, iloc, and ix

For example, if your Series has an explicit integer index, an indexing operation such as data[1] will use the explicit indices, while a slicing operation like data[1:3] will use the implicit Python-style index.

```
In[11]: data = pd.Series(['a', 'b', 'c'], index=[1, 3, 5])  
       data
```

```
Out[11]: 1    a  
        3    b  
        5    c  
       dtype: object
```

```
In[12]: # explicit index when indexing  
       data[1]
```

```
Out[12]: 'a'
```

```
In[13]: # implicit index when slicing  
       data[1:3]
```

```
Out[13]: 3    b  
        5    c  
       dtype: object
```

First, the `loc` attribute allows indexing and slicing that always references the explicit index:

```
In[14]: data.loc[1]  
Out[14]: 'a'  
  
In[15]: data.loc[1:3]  
Out[15]: 1    a  
          3    b  
          dtype: object
```

The `tloc` attribute allows indexing and slicing that always references the implicit Python-style index:

```
In[16]: data.tloc[1]  
Out[16]: 'b'  
  
In[17]: data.tloc[1:3]  
Out[17]: 3    b  
          5    c  
          dtype: object
```

A third indexing attribute, `ix`, is a hybrid of the two, and for Series objects is equivalent to standard `[]`-based indexing.

Data Selection in DataFrame:

DataFrame as a dictionary

```
In[18]: area = pd.Series({'California': 423967, 'Texas': 695662,  
                         'New York': 141297, 'Florida': 170312,  
                         'Illinois': 149995})  
pop = pd.Series({'California': 38332521, 'Texas': 26448193,  
                 'New York': 19651127, 'Florida': 19552860,  
                 'Illinois': 12882135})  
data = pd.DataFrame({'area':area, 'pop':pop})  
data  
  
Out[18]:           area      pop  
California  423967  38332521  
Florida     170312  19552860  
Illinois    149995  12882135  
New York    141297  19651127  
Texas       695662  26448193
```

The individual Series that make up the columns of the DataFrame can be accessed via dictionary-style indexing of the column name:

```
In[19]: data['area']  
Out[19]: California    423967  
          Florida       170312  
          Illinois      149995  
          New York      141297  
          Texas         695662  
          Name: area, dtype: int64
```

DataFrame as two-dimensional array

As mentioned previously, we can also view the DataFrame as an enhanced two-dimensional array. We can examine the raw underlying data array using the `values` attribute:

```
In[24]: data.values
```

```
Out[24]: array([[ 4.23967000e+05,  3.83325210e+07,  9.04139261e+01],
 [ 1.70312000e+05,  1.95528600e+07,  1.14806121e+02],
 [ 1.49995000e+05,  1.28821350e+07,  8.58837628e+01],
 [ 1.41297000e+05,  1.96511270e+07,  1.39076746e+02],
 [ 6.95662000e+05,  2.64481930e+07,  3.80187404e+01]])
```

With this picture in mind, we can do many familiar array-like observations on the DataFrame itself. For example, we can transpose the full DataFrame to swap rows and columns:

```
In[25]: data.T
```

```
Out[25]:
      California    Florida     Illinois   New York     Texas
area      4.239670e+05  1.703120e+05  1.499950e+05  1.412970e+05  6.956620e+05
pop       3.833252e+07  1.955286e+07  1.288214e+07  1.965113e+07  2.644819e+07
density   9.041393e+01  1.148061e+02  8.588376e+01  1.390767e+02  3.801874e+01
```

```
In[28]: data.iloc[:3, :2]
```

```
Out[28]:
      area    pop
California  423967  38332521
Florida     170312  19552860
Illinois    149995  12882135
```

```
In[29]: data.loc[:, 'Illinois', : 'pop']
```

```
Out[29]:
      area    pop
California  423967  38332521
Florida     170312  19552860
Illinois    149995  12882135
```

The `ix` indexer allows a hybrid of these two approaches:

```
In[30]: data.ix[:3, : 'pop']
```

```
Out[30]:
      area    pop
California  423967  38332521
Florida     170312  19552860
Illinois    149995  12882135
```

Additional indexing conventions

There are a couple extra indexing conventions that might seem at odds with the preceding discussion, but nevertheless can be very useful in practice. First, while *indexing* refers to columns, *slicing* refers to rows:

```
In[33]: data['Florida':'Illinois']

Out[33]:      area    pop    density
Florida  170312  19552860  114.806121
Illinois 149995  12882135  85.883763
```

Such slices can also refer to rows by number rather than by index:

```
In[34]: data[1:3]

Out[34]:      area    pop    density
Florida  170312  19552860  114.806121
Illinois 149995  12882135  85.883763
```

Similarly, direct masking operations are also interpreted row-wise rather than column-wise:

```
In[35]: data[data.density > 100]

Out[35]:      area    pop    density
Florida  170312  19552860  114.806121
New York 141297  19651127  139.076746
```

Operating on Data in Pandas:

Pandas : for unary operations like negation and trigonometric functions, the ufuncs will preserve index and column labels in the output, and for binary operations such as addition and multiplication, Pandas will automatically align indices when passing the objects to the ufunc.

Ufuncs: Index Preservation

Because Pandas is designed to work with NumPy, any NumPy ufunc will work on Pandas Series and DataFrame objects. Let's start by defining a simple Series and DataFrame on which to demonstrate this:

```
In[1]: import pandas as pd
       import numpy as np

In[2]: rng = np.random.RandomState(42)
       ser = pd.Series(rng.randint(0, 10, 4))
       ser

Out[2]: 0    6
        1    3
        2    7
        3    4
       dtype: int64

In[3]: df = pd.DataFrame(rng.randint(0, 10, (3, 4)),
                        columns=['A', 'B', 'C', 'D'])
       df

Out[3]:   A  B  C  D
0   6  9  2  6
1   7  4  3  7
2   7  2  5  4
```

If we apply a NumPy ufunc on either of these objects, the result will be another Pandas object *with the indices preserved*:

```
In[4]: np.exp(ser)
```

```
Out[4]: 0      403.428793
        1      20.085537
        2     1096.633158
        3      54.598150
       dtype: float64
```

Or, for a slightly more complex calculation:

```
In[5]: np.sin(df * np.pi / 4)
```

```
Out[5]:         A           B           C           D
0 -1.000000  7.071068e-01  1.000000 -1.000000e+00
1 -0.707107  1.224647e-16  0.707107 -7.071068e-01
2 -0.707107  1.000000e+00 -0.707107  1.224647e-16
```

UFuncs: Index Alignment

For binary operations on two `Series` or `DataFrame` objects, Pandas will align indices in the process of performing the operation. This is very convenient when you are working with incomplete data, as we'll see in some of the examples that follow.

Index alignment in Series

As an example, suppose we are combining two different data sources, and find only the top three US states by *area* and the top three US states by *population*:

```
In[6]: area = pd.Series({'Alaska': 1723337, 'Texas': 695662,
                       'California': 423967}, name='area')
population = pd.Series({'California': 38332521, 'Texas': 26448193,
                       'New York': 19651127}, name='population')
```

Let's see what happens when we divide these to compute the population density:

```
In[7]: population / area

Out[7]: Alaska          NaN
         California    90.413926
         New York       NaN
         Texas          38.018740
         dtype: float64
```

The resulting array contains the *union* of indices of the two input arrays, which we could determine using standard Python set arithmetic on these indices:

```
In[8]: area.index | population.index

Out[8]: Index(['Alaska', 'California', 'New York', 'Texas'], dtype='object')

In[9]: A = pd.Series([2, 4, 6], index=[0, 1, 2])
        B = pd.Series([1, 3, 5], index=[1, 2, 3])
        A + B

Out[9]: 0    NaN
        1    5.0
        2    9.0
        3    NaN
        dtype: float64
```

Index alignment in DataFrame

A similar type of alignment takes place for *both* columns and indices when you are performing operations on DataFrames:

```
In[11]: A = pd.DataFrame(rng.randint(0, 20, (2, 2)),
                         columns=list('AB'))
A
Out[11]:   A   B
0   1  11
1   5   1

In[12]: B = pd.DataFrame(rng.randint(0, 10, (3, 3)),
                         columns=list('BAC'))
B
Out[12]:   B   A   C
0   4   0   9
1   5   8   0
2   9   2   6

In[13]: A + B
Out[13]:      A      B      C
0   1.0  15.0  NaN
1  13.0   6.0  NaN
2   NaN   NaN  NaN
```

Table 3-1. Mapping between Python operators and Pandas methods

Python operator	Pandas method(s)
+	add()
-	sub(), subtract()
*	mul(), multiply()
/	truediv(), div(), divide()
//	floordiv()
%	mod()
**	pow()

Ufuncs: Operations Between DataFrame and Series

When you are performing operations between a DataFrame and a Series, the index and column alignment is similarly maintained.

Operations between a DataFrame and a Series are similar to operations between a two-dimensional and one-dimensional NumPy array.

```
In[15]: A = rng.randint(10, size=(3, 4))
```

```
A
```

```
Out[15]: array([[3, 8, 2, 4],
 [2, 6, 4, 8],
 [6, 1, 3, 8]])
```

```
In[16]: A - A[0]
```

```
Out[16]: array([[ 0, 0, 0, 0],  
[-1, -2, 2, 4],  
[ 3, -7, 1, 4]])
```

subtraction between a two-dimensional array and one of its rows is applied row-wise.

In Pandas, the convention similarly operates row-wise by default:

```
In[17]: df = pd.DataFrame(A, columns=list('QRST'))
```

```
df - df.iloc[0]
```

```
Out[17]: Q R S T
```

```
0 0 0 0
```

```
1 -1 -2 2 4
```

```
2 3 -7 1 4
```

If you would instead like to operate column-wise, you can use the object methods mentioned earlier, while specifying the axis keyword:

```
In[18]: df.subtract(df['R'], axis=0)
```

```
Out[18]: Q R S T
```

```
0 -5 0 -6 -4
```

```
1 -4 0 -2 2
```

```
2 5 0 2 7
```

VII. Handling Missing Data

In the real world is that real-world data is rarely clean and homogeneous. In particular, many interesting datasets will have some amount of data missing

Trade-Offs in Missing Data Conventions

A number of schemes have been developed to indicate the presence of missing data in a table or DataFrame. Two strategies: using a mask that globally indicates missing values, or choosing a sentinel value that indicates a missing entry.

In the masking approach, the mask might be an entirely separate Boolean array, or it may involve appropriation of one bit in the data representation to locally indicate the null status of a value.

In the sentinel approach, the sentinel value could be some data-specific convention. Eg: IEEE floating-point specification.

Missing Data in Pandas

The way in which Pandas handles missing values is constrained by its reliance on the NumPy package, which does not have a built-in notion of NA values for nonfloating-point data types.

None: Pythonic missing data

The first sentinel value used by Pandas is `None`, a Python singleton object that is often used for missing data in Python code. Because `None` is a Python object, it cannot be used in any arbitrary NumPy/Pandas array, but only in arrays with data type '`object`' (i.e., arrays of Python objects):

```
In[1]: import numpy as np  
       import pandas as pd  
  
In[2]: vals1 = np.array([1, None, 3, 4])  
       vals1  
  
Out[2]: array([1, None, 3, 4], dtype=object)
```

NaN: Missing numerical data

The other missing data representation, `NaN` (acronym for *Not a Number*), is different; it is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation:

```
In[5]: vals2 = np.array([1, np.nan, 3, 4])  
       vals2.dtype  
  
Out[5]: dtype('float64')
```

Notice that NumPy chose a native floating-point type for this array: this means that unlike the object array from before, this array supports fast operations pushed into compiled code. You should be aware that `NaN` is a bit like a data virus—it infects any other object it touches. Regardless of the operation, the result of arithmetic with `NaN` will be another `NaN`:

```
In[6]: 1 + np.nan  
Out[6]: nan  
  
In[7]: 0 * np.nan  
Out[7]: nan
```

NaN and None in Pandas

NaN and None both have their place, and Pandas is built to handle the two of them nearly interchangeably, converting between them where appropriate:

```
In[10]: pd.Series([1, np.nan, 2, None])
```

```
Out[10]: 0    1.0
         1    NaN
         2    2.0
         3    NaN
        dtype: float64
```

For types that don't have an available sentinel value, Pandas automatically type-casts when NA values are present. For example, if we set a value in an integer array to np.nan, it will automatically be upcast to a floating-point type to accommodate the NA:

```
In[11]: x = pd.Series(range(2), dtype=int)
         x
```

```
Out[11]: 0    0
         1    1
        dtype: int64
```

```
In[12]: x[0] = None
         x
```

```
Out[12]: 0    NaN
         1    1.0
        dtype: float64
```

Table 3-2. Pandas handling of NAs by type

Typeclass	Conversion when storing NAs	NA sentinel value
floating	No change	np.nan
object	No change	None or np.nan
integer	Cast to float64	np.nan
boolean	Cast to object	None or np.nan

Keep in mind that in Pandas, string data is always stored with an object dtype.

Operating on Null Values

As we have seen, Pandas treats None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful methods for detecting, removing, and replacing null values in Pandas data structures. They are:

`isnull()`
Generate a Boolean mask indicating missing values

`notnull()`
Opposite of `isnull()`

`dropna()`
Return a filtered version of the data

`fillna()`
Return a copy of the data with missing values filled or imputed

Detecting null values

Pandas data structures have two useful methods for detecting null data: `isnull()` and `notnull()`. Either one will return a Boolean mask over the data. For example:

```
In[13]: data = pd.Series([1, np.nan, 'hello', None])
In[14]: data.isnull()
Out[14]: 0    False
         1    True
         2    False
         3    True
        dtype: bool
```

Dropping null values

In addition to the masking used before, there are the convenience methods, `dropna()` (which removes NA values) and `fillna()` (which fills in NA values). For a `Series`, the result is straightforward:

```
In[16]: data.dropna()
```

```
Out[16]: 0      1
         2    hello
        dtype: object
```

For a `DataFrame`, there are more options. Consider the following `DataFrame`:

```
In[17]: df = pd.DataFrame([[1,       np.nan, 2],
                           [2,       3,      5],
                           [np.nan, 4,      6]])
df
```



```
Out[17]:   0  1  2
          0  1.0  NaN  2
          1  2.0  3.0  5
          2  NaN  4.0  6
```

We cannot drop single values from a `DataFrame`; we can only drop full rows or full columns. Depending on the application, you might want one or the other, so `dropna()` gives a number of options for a `DataFrame`.

By default, `dropna()` will drop all rows in which *any* null value is present:

```
In[18]: df.dropna()
```

```
Out[18]:   0  1  2
          1  2.0  3.0  5
```

Filling null values

Sometimes rather than dropping NA values, you'd rather replace them with a valid value. This value might be a single number like zero, or it might be some sort of imputation or interpolation from the good values. You could do this in-place using the `isnull()` method as a mask, but because it is such a common operation Pandas provides the `fillna()` method, which returns a copy of the array with the null values replaced.

Consider the following `Series`:

```
In[23]: data = pd.Series([1, np.nan, 2, None, 3], index=list('abcde'))
data
```

```
Out[23]: a    1.0
         b    NaN
         c    2.0
         d    NaN
```

```
e    3.0  
dtype: float64
```

We can fill NA entries with a single value, such as zero:

```
In[24]: data.fillna(0)  
  
Out[24]: a    1.0  
          b    0.0  
          c    2.0  
          d    0.0  
          e    3.0  
          dtype: float64
```

We can specify a forward-fill to propagate the previous value forward:

```
In[25]: # forward-fill  
         data.fillna(method='ffill')  
  
Out[25]: a    1.0  
          b    1.0  
          c    2.0  
          d    2.0  
          e    3.0  
          dtype: float64
```

Or we can specify a back-fill to propagate the next values backward:

```
In[26]: # back-fill  
         data.fillna(method='bfill')  
  
Out[26]: a    1.0  
          b    2.0  
          c    2.0  
          d    3.0  
          e    3.0  
          dtype: float64
```

VIII. Hierarchical Indexing:

Hierarchical indexing (also known as multi-indexing) - to incorporate multiple index levels within a single index. In this way, higher-dimensional data can be compactly represented within the familiar one-dimensional Series and two-dimensional DataFrame objects.

A Multiply Indexed Series

The better way: Pandas MultiIndex

Fortunately, Pandas provides a better way. Our tuple-based indexing is essentially a rudimentary multi-index, and the Pandas `MultiIndex` type gives us the type of operations we wish to have. We can create a multi-index from the tuples as follows:

```
In[5]: index = pd.MultiIndex.from_tuples(index)
index

Out[5]: MultiIndex(levels=[[California', 'New York', 'Texas'], [2000, 2010]],
                   labels=[[0, 0, 1, 1, 2, 2], [0, 1, 0, 1, 0, 1]])
```

Notice that the `MultiIndex` contains multiple *levels* of indexing—in this case, the state names and the years, as well as multiple *labels* for each data point which encode these levels.

If we reindex our series with this `MultiIndex`, we see the hierarchical representation of the data:

```
In[6]: pop = pop.reindex(index)
pop

Out[6]: California  2000    33871648
                  2010    37253956
New York      2000    18976457
                  2010    19378102

Texas        2000    20851820
                  2010    25145561
dtype: int64
```

MultiIndex as extra dimension

You might notice something else here: we could easily have stored the same data using a simple `DataFrame` with index and column labels. In fact, Pandas is built with this equivalence in mind. The `unstack()` method will quickly convert a multiply-indexed `Series` into a conventionally indexed `DataFrame`:

```
In[8]: pop_df = pop.unstack()
pop_df

Out[8]:           2000      2010
California  33871648  37253956
New York    18976457  19378102
Texas       20851820  25145561
```

Naturally, the `stack()` method provides the opposite operation:

```
In[9]: pop_df.stack()

Out[9]: California  2000    33871648
                  2010    37253956
New York      2000    18976457
                  2010    19378102
Texas        2000    20851820
                  2010    25145561
dtype: int64
```

Methods of MultiIndex Creation

The most straightforward way to construct a multiply indexed Series or DataFrame is to simply pass a list of two or more index arrays to the constructor. For example:

```
In[12]: df = pd.DataFrame(np.random.rand(4, 2),
                         index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],
                         columns=['data1', 'data2'])
df

Out[12]:      data1    data2
a 1  0.554233  0.356072
            2  0.925244  0.219474
b 1  0.441759  0.610054
            2  0.171495  0.886688
```

The work of creating the MultiIndex is done in the background.

Similarly, if you pass a dictionary with appropriate tuples as keys, Pandas will automatically recognize this and use a MultiIndex by default:

```
In[13]: data = {('California', 2000): 33871648,
                ('California', 2010): 37253956,
                ('Texas', 2000): 20851820,
                ('Texas', 2010): 25145561,
                ('New York', 2000): 18976457,
                ('New York', 2010): 19378102}
pd.Series(data)

Out[13]: California  2000    33871648
                  2010    37253956
           New York   2000    18976457
                  2010    19378102
           Texas     2000    20851820
                  2010    25145561
           dtype: int64
```

Nevertheless, it is sometimes useful to explicitly create a MultiIndex; we'll see a couple of these methods here.

MultIndex level names

Sometimes it is convenient to name the levels of the `MultiIndex`. You can accomplish this by passing the `names` argument to any of the above `MultiIndex` constructors, or by setting the `names` attribute of the index after the fact:

```
In[18]: pop.index.names = ['state', 'year']
pop
```

```
Out[18]: state      year
          California  2000    33871648
                      2010    37253956
          New York   2000    18976457
                      2010    19378102
          Texas      2000    20851820
                      2010    25145561
          dtype: int64
```

With more involved datasets, this can be a useful way to keep track of the meaning of various index values.

MultIndex for columns

In a DataFrame, the rows and columns are completely symmetric, and just as the rows can have multiple levels of indices, the columns can have multiple levels as well. Consider the following, which is a mock-up of some (somewhat realistic) medical data:

Explicit MultiIndex constructors

For more flexibility in how the index is constructed, you can instead use the class method constructors available in the `pd.MultiIndex`. For example, as we did before, you can construct the `MultiIndex` from a simple list of arrays, giving the index values within each level:

You can construct it from a list of tuples, giving the multiple index values of each point:

```
In[15]: pd.MultiIndex.from_tuples([('a', 1), ('a', 2), ('b', 1), ('b', 2)])
Out[15]: MultiIndex(levels=[[a, b]], [1, 2]),
           labels=[[0, 0, 1, 1], [0, 1, 0, 1]])
```

You can even construct it from a Cartesian product of single indices:

Indexing and Slicing a MultiIndex

Indexing and slicing on a `MultiIndex` is designed to be intuitive, and it helps if you think about the indices as added dimensions. We'll first look at indexing multiply indexed `Series`, and then multiply indexed `DataFrames`.

Multiply indexed Series

Consider the multiply indexed `Series` of state populations we saw earlier:

```
In[21]: pop  
Out[21]: state      year  
          California  2000    33871648  
                      2010    37253956  
          New York    2000    18976457  
                      2010    19378102  
          Texas       2000    20851820  
                      2010    25145561  
          dtype: int64
```

We can access single elements by indexing with multiple terms:

```
In[22]: pop['California', 2000]  
Out[22]: 33871648
```

The `MultiIndex` also supports *partial indexing*, or indexing just one of the levels in the index. The result is another `Series`, with the lower-level indices maintained:

```
In[23]: pop['California']  
Out[23]: year  
          2000    33871648  
          2010    37253956  
          dtype: int64
```

Multiply indexed DataFrames

A multiply indexed DataFrame behaves in a similar manner. Consider our toy medical DataFrame from before:

```
In[28]: health_data  
Out[28]: subject      Bob        Guido        Sue  
          type       HR  Temp       HR  Temp       HR  Temp  
          year visit  
          2013  1    31.0  38.7  32.0  36.7  35.0  37.2  
                  2    44.0  37.7  50.0  35.0  29.0  36.7  
          2014  1    30.0  37.4  39.0  37.8  61.0  36.9  
                  2    47.0  37.8  48.0  37.3  51.0  36.5
```

Remember that columns are primary in a DataFrame, and the syntax used for multiply indexed Series applies to the columns. For example, we can recover Guido's heart rate data with a simple operation:

```
In[29]: health_data['Guido', 'HR']  
Out[29]: year  visit  
          2013  1    32.0  
                  2    50.0  
          2014  1    39.0  
                  2    48.0  
          Name: (Guido, HR), dtype: float64
```

Rearranging Multi-Indices

Sorted and unsorted indices

Many of the MultiIndex slicing operations will fail if the index is not sorted.

Pandas provides a number of convenience routines to perform this type of sorting; examples are the `sort_index()` and `sortlevel()` methods of the DataFrame. We'll use the simplest, `sort_index()`, here:

```
In[36]: data = data.sort_index()  
        data  
Out[36]: char  int  
          a    1    0.003001  
                  2    0.164974  
          b    1    0.001693  
                  2    0.526226  
          c    1    0.741650  
                  2    0.569264  
          dtype: float64
```

With the index sorted in this way, partial slicing will work as expected:

```
In[37]: data['a':'b']  
Out[37]: char  int  
          a    1    0.003001  
                  2    0.164974  
          b    1    0.001693  
                  2    0.526226  
          dtype: float64
```

Stacking and unstacking indices

```
In[38]: pop.unstack(level=0)

Out[38]: state    California   New York      Texas
          year
          2000    33871648  18976457  20851820
          2010    37253956  19378102  25145561
```

```
In[39]: pop.unstack(level=1)

Out[39]: year        2000        2010
          state
          California  33871648  37253956
          New York    18976457  19378102
          Texas       20851820  25145561
```

The opposite of `unstack()` is `stack()`, which here can be used to recover the original series:

```
In[40]: pop.unstack().stack()

Out[40]: state      year
          California  2000    33871648
                      2010    37253956
          New York    2000    18976457
                      2010    19378102
          Texas       2000    20851820
                      2010    25145561
          dtype: int64
```

Index setting and resetting

Another way to rearrange hierarchical data is to turn the index labels into columns; this can be accomplished with the `reset_index` method. Calling this on the population dictionary will result in a DataFrame with a `state` and `year` column holding the information that was formerly in the index. For clarity, we can optionally specify the name of the data for the column representation:

```
In[41]: pop_flat = pop.reset_index(name='population')
pop_flat

Out[41]:      state  year  population
0  California  2000    33871648
1  California  2010    37253956
2  New York    2000    18976457
3  New York    2010    19378102
4      Texas    2000    20851820
5      Texas    2010    25145561
```

Data Aggregations on Multi-Indices

We've previously seen that Pandas has built-in data aggregation methods, such as `mean()`, `sum()`, and `max()`. For hierarchically indexed data, these can be passed a `level` parameter that controls which subset of the data the aggregate is computed on.

For example, let's return to our health data:

```
In[43]: health_data  
Out[43]:   subject    Bob      Guido      Sue  
           type      HR  Temp      HR  Temp      HR  Temp  
           year visit  
           2013  1    31.0  38.7  32.0  36.7  35.0  37.2  
                  2    44.0  37.7  50.0  35.0  29.0  36.7  
           2014  1    30.0  37.4  39.0  37.8  61.0  36.9  
                  2    47.0  37.8  48.0  37.3  51.0  36.5
```

Perhaps we'd like to average out the measurements in the two visits each year. We can do this by naming the index level we'd like to explore, in this case the `year`:

```
In[44]: data_mean = health_data.mean(level='year')  
data_mean  
Out[44]:   subject    Bob      Guido      Sue  
           type      HR  Temp      HR  Temp      HR  Temp  
           year  
           2013    37.5  38.2  41.0  35.85  32.0  36.95  
           2014    38.5  37.6  43.5  37.55  56.0  36.70
```

VIII. Combining Datasets: Concat and Append

Simple Concatenation with `pd.concat`

Pandas has a function, `pd.concat()`, which has a similar syntax to `np.concatenate` but contains a number of options that we'll discuss momentarily:

```
# Signature in Pandas v0.18  
pd.concat(objs, axis=0, join='outer', join_axes=None, ignore_index=False,  
          keys=None, levels=None, names=None, verify_integrity=False,  
          copy=True)
```

`pd.concat()` can be used for a simple concatenation of `Series` or `DataFrame` objects, just as `np.concatenate()` can be used for simple concatenations of arrays:

```
In[6]: ser1 = pd.Series(['A', 'B', 'C'], index=[1, 2, 3])  
ser2 = pd.Series(['D', 'E', 'F'], index=[4, 5, 6])  
pd.concat([ser1, ser2])  
  
Out[6]: 1    A  
        2    B  
        3    C  
        4    D  
        5    E  
        6    F  
       dtype: object
```

It also works to concatenate higher-dimensional objects, such as `DataFrames`:

Duplicate indices

One important difference between `np.concatenate` and `pd.concat` is that Pandas concatenation *preserves indices*, even if the result will have duplicate indices! Consider this simple example:

```
In[9]: x = make_df('AB', [0, 1])
y = make_df('AB', [2, 3])
```

```
y.index = x.index # make duplicate indices!
print(x); print(y); print(pd.concat([x, y]))
```

	x		y		pd.concat([x, y])			
	A	B	A	B	A	B		
0	A0	B0	0	A2	B2	0	A0	B0
1	A1	B1	1	A3	B3	1	A1	B1
					0	A2	B2	
					1	A3	B3	

Notice the repeated indices in the result. While this is valid within DataFrames, the outcome is often undesirable. `pd.concat()` gives us a few ways to handle it.

Catching the repeats as an error.

Ignoring the index.

Adding MultiIndex keys

IX. Aggregation and Grouping

An essential piece of analysis of large data is efficient summarization: computing aggregations like `sum()`, `mean()`, `median()`, `min()`, and `max()`

Simple Aggregation in Pandas

```
In[7]: df = pd.DataFrame({'A': rng.rand(5),
                           'B': rng.rand(5)})
df
```

```
Out[7]:      A          B
0  0.155995  0.020584
1  0.058084  0.969910
2  0.866176  0.832443
3  0.601115  0.212339
4  0.708073  0.181825
```

```
In[8]: df.mean()
```

```
Out[8]: A    0.477888
        B    0.443420
       dtype: float64
```

Table 3-3. Listing of Pandas aggregation methods

Aggregation	Description
<code>count()</code>	Total number of items
<code>first(), last()</code>	First and last item
<code>mean(), median()</code>	Mean and median
<code>min(), max()</code>	Minimum and maximum
<code>std(), var()</code>	Standard deviation and variance
<code>mad()</code>	Mean absolute deviation
<code>prod()</code>	Product of all items
<code>sum()</code>	Sum of all items

These are all methods of `DataFrame` and `Series` objects.

GroupBy: Split, Apply, Combine

A canonical example of this split-apply-combine operation, where the “apply” is a summation aggregation, is illustrated in Figure 3-1.

Figure 3-1 makes clear what the GroupBy accomplishes:

- The split step involves breaking up and grouping a DataFrame depending on the value of the specified key.
- The apply step involves computing some function, usually an aggregate, transformation, or filtering, within the individual groups.
- The combine step merges the results of these operations into an output array.

Here it’s important to realize that the intermediate splits do not need to be explicitly instantiated.

```
In[11]: df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],
                           'data': range(6)}, columns=['key', 'data'])
df
Out[11]:   key  data
      0    A    0
      1    B    1
      2    C    2
      3    A    3
      4    B    4
      5    C    5
```

We can compute the most basic split-apply-combine operation with the `groupby()` method of `DataFrames`, passing the name of the desired key column:

```
In[12]: df.groupby('key')
```

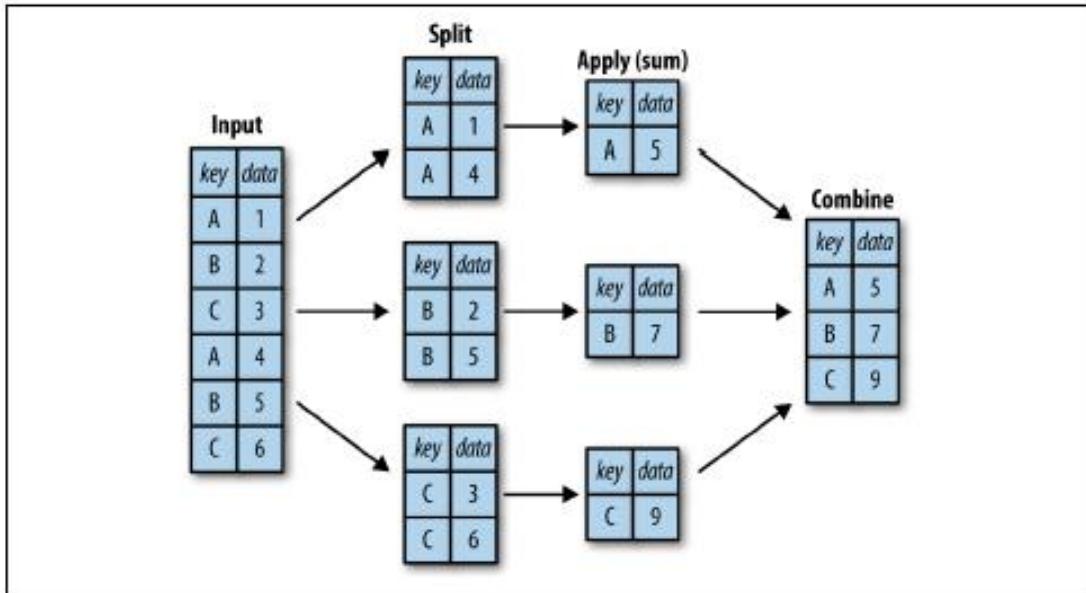


Figure 3-1. A visual representation of a groupby operation

To produce a result, we can apply an aggregate to this `DataFrameGroupBy` object, which will perform the appropriate apply/combine steps to produce the desired result:

```
In[13]: df.groupby('key').sum()
```

```
Out[13]:    data
      key
      A    3
      B    5
      C    7
```

The GroupBy object

The GroupBy object is a very flexible abstraction.

Column indexing. The GroupBy object supports column indexing in the same way as the DataFrame, and returns a modified GroupBy object. For example:

```
In[14]: planets.groupby('method')
```

```
Out[14]: <pandas.core.groupby.DataFrameGroupBy object at 0x1172727b8>
```

```
In[15]: planets.groupby('method')['orbital_period']
```

```
Out[15]: <pandas.core.groupby.SeriesGroupBy object at 0x117272da0>
```

Iteration over groups. The GroupBy object supports direct iteration over the groups, returning each group as a Series or DataFrame:

```
In[17]: for (method, group) in planets.groupby('method'):
```

```
    print("{0:30s} shape={1}".format(method, group.shape))
```

Dispatch methods. Through some Python class magic, any method not explicitly implemented by the GroupBy object will be passed through and called on the groups, whether they are DataFrame or Series objects. For example, you can use the `describe()` method of DataFrames to perform a set of aggregations that describe each group in the data:

```
In[18]: planets.groupby('method')['year'].describe().unstack()
```

Aggregation. We're now familiar with GroupBy aggregations with `sum()`, `median()`, and the like, but the `aggregate()` method allows for even more flexibility. It can take a string, a function, or a list thereof, and compute all the aggregates at once. Here is a quick example combining all these:

```
In[20]: df.groupby('key').aggregate(['min', np.median, max])
```

```
Out[20]:      data1          data2
              min median max   min median max
key
A      0    1.5    3    3    4.0    5
B      1    2.5    4    0    3.5    7
C      2    3.5    5    3    6.0    9
```

Another useful pattern is to pass a dictionary mapping column names to operations to be applied on that column:

```
In[21]: df.groupby('key').aggregate({'data1': 'min',
                                         'data2': 'max'})
```

```
Out[21]:      data1  data2
key
A      0      5
B      1      7
C      2      9
```

Filtering. A filtering operation allows you to drop data based on the group properties. For example, we might want to keep all groups in which the standard deviation is larger than some critical value:

```
In[22]:
def filter_func(x):
    return x['data2'].std() > 4

print(df); print(df.groupby('key').std());
print(df.groupby('key').filter(filter_func))

df                      df.groupby('key').std()
key  data1  data2      key      data1      data2
0   A      0      5      A    2.12132  1.414214
1   B      1      0      B    2.12132  4.949747
2   C      2      3      C    2.12132  4.242641
3   A      3      3
4   B      4      7
5   C      5      9

df.groupby('key').filter(filter_func)
key  data1  data2
1   B      1      0
```

```
2  C      2      3
4  B      4      7
5  C      5      9
```

The `filter()` function should return a Boolean value specifying whether the group passes the filtering. Here because group A does not have a standard deviation greater than 4, it is dropped from the result.

Transformation. While aggregation must return a reduced version of the data, transformation can return some transformed version of the full data to recombine. For such a transformation, the output is the same shape as the input. A common example is to center the data by subtracting the group-wise mean:

```
In[23]: df.groupby('key').transform(lambda x: x - x.mean())
Out[23]:   data1  data2
          0    -1.5    1.0
          1    -1.5   -3.5
          2    -1.5   -3.0
          3     1.5   -1.0
          4     1.5    3.5
          5     1.5    3.0
```

The `apply()` method. The `apply()` method lets you apply an arbitrary function to the group results. The function should take a DataFrame, and return either a Pandas object (e.g., DataFrame, Series) or a scalar; the `combine` operation will be tailored to the type of output returned.

For example, here is an `apply()` that normalizes the first column by the sum of the second:

```
In[24]: def norm_by_data2(x):
    # x is a DataFrame of group values
    x['data1'] /= x['data2'].sum()
    return x

print(df); print(df.groupby('key').apply(norm_by_data2))

df                               df.groupby('key').apply(norm_by_data2)
  key  data1  data2      key      data1  data2
0  A      0      5      0  A  0.000000      5
1  B      1      0      1  B  0.142857      0
2  C      2      3      2  C  0.166667      3
3  A      3      3      3  A  0.375000      3
4  B      4      7      4  B  0.571429      7
5  C      5      9      5  C  0.416667      9
```

Specifying the split key

In the simple examples presented before, we split the DataFrame on a single column name. This is just one of many options by which the groups can be defined, and we'll go through some other options for group specification here.

A list, array, series, or index providing the grouping keys. The key can be any series or list with a length matching that of the DataFrame. For example:

```
In[25]: L = [0, 1, 0, 1, 2, 0]
print(df); print(df.groupby(L).sum())

df                               df.groupby(L).sum()
  key    data1  data2          data1  data2
0   A      0      5      0      7     17
1   B      1      0      1      4      3
2   C      2      3      2      4      7
3   A      3      3
4   B      4      7
5   C      5      9
```

Of course, this means there's another, more verbose way of accomplishing the `df.groupby('key')` from before:

```
In[26]: print(df); print(df.groupby(df['key']).sum())

df                               df.groupby(df['key']).sum()
  key    data1  data2          data1  data2
0   A      0      5      A      3      8
1   B      1      0      B      5      7
2   C      2      3      C      7     12
3   A      3      3
4   B      4      7
5   C      5      9
```

A dictionary or series mapping index to group. Another method is to provide a dictionary that maps index values to the group keys:

```
In[27]: df2 = df.set_index('key')
mapping = {'A': 'vowel', 'B': 'consonant', 'C': 'consonant'}
print(df2); print(df2.groupby(mapping).sum())

df2                               df2.groupby(mapping).sum()
  key    data1  data2          data1  data2
A      0      5      consonant    12     19
B      1      0      vowel        3      8
C      2      3
A      3      3
B      4      7
C      5      9
```

Any Python function. Similar to mapping, you can pass any Python function that will input the index value and output the group:

```
In[28]: print(df2); print(df2.groupby(str.lower).mean())
```

key	data1	data2		df2.groupby(str.lower).mean()	data1	data2
A	0	5		a	1.5	4.0
B	1	0		b	2.5	3.5
C	2	3		c	3.5	6.0
A	3	3				
B	4	7				
C	5	9				

A list of valid keys. Further, any of the preceding key choices can be combined to group on a multi-index:

```
In[29]: df2.groupby([str.lower, mapping]).mean()
```

```
Out[29]:
```

		data1	data2
a	vowel	1.5	4.0
b	consonant	2.5	3.5
c	consonant	3.5	6.0

X. Pivot Tables

We have seen how the GroupBy abstraction lets us explore relationships within a dataset.

A pivot table is a similar operation that is commonly seen in spreadsheets and other programs that operate on tabular data.

The pivot table takes simple columnwise data as input, and groups the entries into a two-dimensional table that provides a multidimensional summarization of the data.

Motivating Pivot Tables

For the examples in this section, we'll use the database of passengers on the *Titanic*, available through the Seaborn library (see “[Visualization with Seaborn](#)” on page 311):

```
In[1]: import numpy as np
       import pandas as pd
       import seaborn as sns
       titanic = sns.load_dataset('titanic')

In[2]: titanic.head()

Out[2]:
   survived  pclass    sex   age  sibsp  parch     fare embarked class  \\
0         0      3  male  22.0      1      0    7.2500        S  Thrd
1         1      1 female  38.0      1      0   71.2833        C  Firs
2         1      3 female  26.0      0      0    7.9250        S  Third
3         1      1 female  35.0      1      0   53.1000        S  Firs
4         0      3  male  35.0      0      0    8.0500        S  Thrd

   who adult_male deck embark_town alive alone
0  man      True    NaN  Southampton  no  False
1 woman    False     C  Cherbourg  yes  False
2 woman    False    NaN  Southampton  yes   True
3 woman    False     C  Southampton  yes  False
4  man      True    NaN  Southampton  no   True
```

This contains a wealth of information on each passenger of that ill-fated voyage, including gender, age, class, fare paid, and much more.

Pivot Table Syntax

Here is the equivalent to the preceding operation using the `pivot_table` method of `DataFrames`:

```
In[5]: titanic.pivot_table('survived', index='sex', columns='class')

Out[5]: class      First     Second     Third
          sex
          female  0.968085  0.921053  0.500000
          male    0.368852  0.157407  0.135447
```

Multilevel pivot tables

Just as in the GroupBy, the grouping in pivot tables can be specified with multiple levels, and via a number of options. For example, we might be interested in looking at age as a third dimension. We'll bin the age using the pd.cut function:

```
In[6]: age = pd.cut(titanic['age'], [0, 18, 80])
titanic.pivot_table('survived', ['sex', 'age'], 'class')

Out[6]:   class            First    Second    Third
          sex   age
          female (0, 18]  0.909091  1.000000  0.511628
                     (18, 80]  0.972973  0.900000  0.423729
          male   (0, 18]  0.800000  0.600000  0.215686
                     (18, 80]  0.375000  0.071429  0.133663
```

We can apply this same strategy when working with the columns as well; let's add info on the fare paid using pd.qcut to automatically compute quantiles:

```
In[7]: fare = pd.qcut(titanic['fare'], 2)
titanic.pivot_table('survived', ['sex', 'age'], [fare, 'class'])

Out[7]:
fare            [0, 14.454]
class           First    Second    Third    \\
sex   age
female (0, 18]      NaN  1.000000  0.714286
          (18, 80]      NaN  0.880000  0.444444
male   (0, 18]      NaN  0.000000  0.260870
          (18, 80]      0.0  0.098039  0.125000

fare            (14.454, 512.329]
class           First    Second    Third
sex   age
female (0, 18]  0.909091  1.000000  0.318182
          (18, 80]  0.972973  0.914286  0.391304
male   (0, 18]  0.800000  0.818182  0.178571
          (18, 80]  0.391304  0.030303  0.192308
```

Additional pivot table options

The full call signature of the pivot_table method of DataFrames is as follows:

```
# call signature as of Pandas 0.18
DataFrame.pivot_table(data, values=None, index=None, columns=None,
                      aggfunc='mean', fill_value=None, margins=False,
                      dropna=True, margins_name='All')
```

The `aggfunc` keyword controls what type of aggregation is applied, which is a mean by default. As in the `GroupBy`, the aggregation specification can be a string representing one of several common choices ('`sum`', '`mean`', '`count`', '`min`', '`max`', etc.) or a function that implements an aggregation (`np.sum()`, `min()`, `sum()`, etc.). Additionally, it can be specified as a dictionary mapping a column to any of the above desired options:

```
In[8]: titanic.pivot_table(index='sex', columns='class',
                           aggfunc={'survived':sum, 'fare':'mean'})
```

class	fare			survived		
	First	Second	Third	First	Second	Third
sex						
female	106.125798	21.970121	16.118810	91.0	70.0	72.0
male	67.226127	19.741782	12.661633	45.0	17.0	47.0

At times it's useful to compute totals along each grouping. This can be done via the `margins` keyword:

```
In[9]: titanic.pivot_table('survived', index='sex', columns='class', margins=True)
```

class	First	Second	Third	All
sex				
female	0.968085	0.921053	0.500000	0.742038
male	0.368852	0.157407	0.135447	0.188908
All	0.629630	0.472826	0.242363	0.383838

UNIT V

DATA VISUALIZATION

Importing Matplotlib – Line plots – Scatter plots – visualizing errors – density and contour plots – Histograms – legends – colors – subplots – text and annotation – customization – three dimensional plotting - Geographic Data with Basemap - Visualization with Seaborn.

Simple Line Plots

The simplest of all plots is the visualization of a single function $y = f(x)$. Here we will take a first look at creating a simple plot of this type.

The *figure* (an instance of the class `plt.Figure`) can be thought of as a single container that contains all the objects representing axes, graphics, text, and labels.

The *axes* (an instance of the class `plt.Axes`) is what we see above: a bounding box with ticks and labels, which will eventually contain the plot elements that make up our visualization.

Line Colors and Styles

- The first adjustment you might wish to make to a plot is to control the line colors and styles.
- To adjust the color, you can use the `color` keyword, which accepts a string argument representing virtually any imaginable color. The color can be specified in a variety of ways
- If no color is specified, Matplotlib will automatically cycle through a set of default colors for multiple lines

Different forms of color representation.

specify color by name - `color='blue'`

short color code (`rgbcmkyk`) - `color='g'`

Grayscale between 0 and 1 - `color='0.75'`

Hex code (RRGGBB from 00 to FF) -

`color='#FFDD44'` RGB tuple, values 0 and 1

-

`color=(1.0,0.2,0.3)` all HTML color names

supported -

`color='chartreuse'`

- We can adjust the line style using the `linestyle` keyword.

Different line styles

`linestyl`

`e='soli`

`d'`

`linestyl`

`e='das`

`hed'`

`linestyl`

`e='das`

`hdot'`

`linestyl`

```
e='dott  
ed'
```

Short assignment

```
linestyle='-'  
' # solid  
linestyle='-'  
'- # dashed  
linestyle='-'  
' #  
dashdot  
linestyle=':'  
' # dotted
```

- linestyle and color codes can be combined into a single nonkeyword argument to the plt.plot() function

```
plt.plot(x, x + 0, '-g') #  
solid green plt.plot(x, x +  
1, '--c') # dashed cyan  
plt.plot(x, x + 2, '-.k') #  
dashdot black plt.plot(x, x  
+ 3, ':r'); # dotted red
```

Axes
Limits

- The most basic way to adjust axis limits is to use the plt.xlim() and plt.ylim() methods

Example

```
plt.xlim(10, 0)
plt.ylim(1.2, -1.2);
```

- The plt.axis() method allows you to set the x and y limits with a single call, by passing a list that specifies [xmin, xmax, ymin, ymax]

```
plt.axis([-1, 11, -1.5, 1.5]);
```

- Aspect ratio equal is used to represent one unit in x is equal to one unit in y. plt.axis('equal')

Labeling Plots

The labeling of plots includes titles, axis labels, and simple legends.

Title - plt.title()

Label - plt.xlabel()

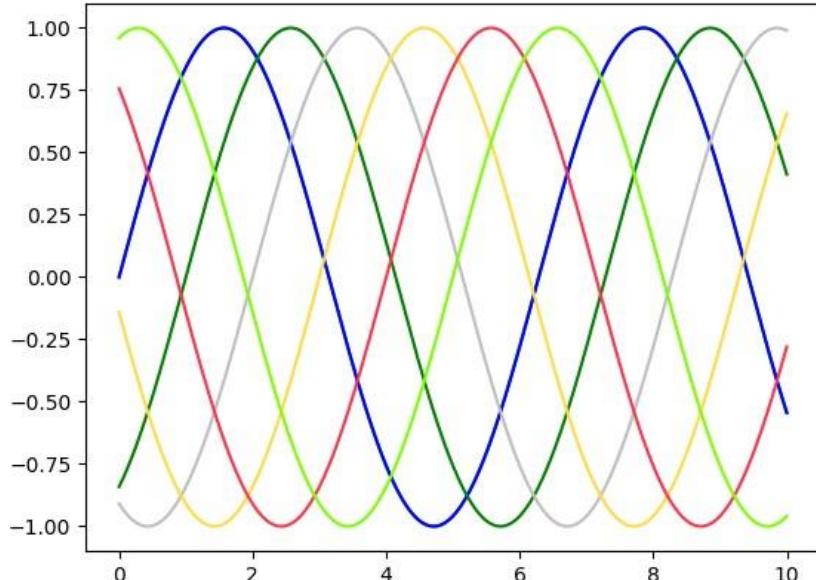
plt.ylabel()

Legend - plt.legend()

Example programs

Line color

```
import matplotlib.pyplot as plt
import numpy as np
fig =
plt.figure()
ax =
plt.axes()
x = np.linspace(0, 10,
1000)
ax.plot(x, np.sin(x));
plt.plot(x, np.sin(x - 0), color='blue') # specify color by name
plt.plot(x, np.sin(x - 1), color='g') # short color code
(rgbcmky) plt.plot(x, np.sin(x - 2), color='0.75') # Grayscale
between 0 and 1
plt.plot(x, np.sin(x - 3), color='#FFDD44') # Hex code (RRGGBB from 00 to
FF)
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB tuple, values 0 and 1
plt.plot(x, np.sin(x - 5), color='chartreuse');# all HTML color names
supported
```



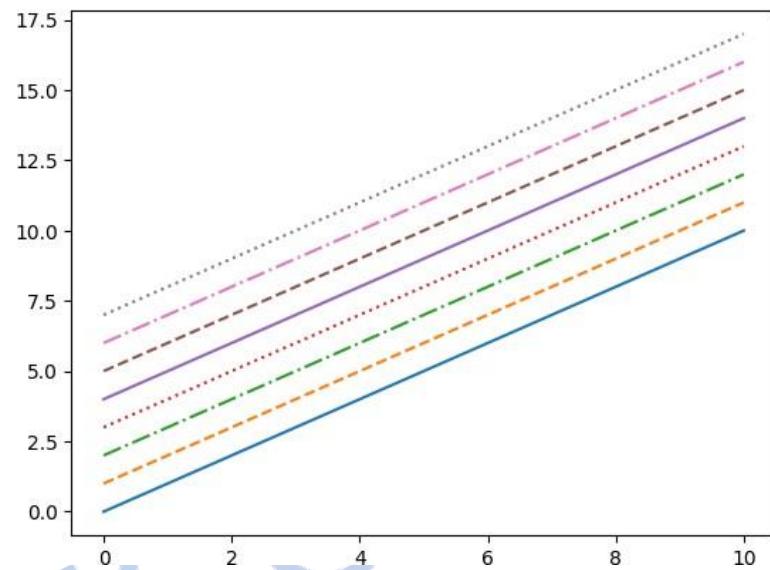
Line style

```
import matplotlib.pyplot as plt
```

```

import numpy as np
fig =
plt.figure()
    ax = plt.axes()
    x = np.linspace(0, 10, 1000)
    plt.plot(x, x + 0, linestyle='solid')
    plt.plot(x, x + 1,
    linestyle='dashed') plt.plot(x, x +
2, linestyle='dashdot') plt.plot(x, x
+ 3, linestyle='dotted');
# For short, you can use the following
codes:plt.plot(x, x + 4, linestyle='-) # solid
plt.plot(x, x + 5, linestyle='--) # dashed
plt.plot(x, x + 6, linestyle='-.) # dashdot
plt.plot(x, x + 7, linestyle=':); # dotted

```

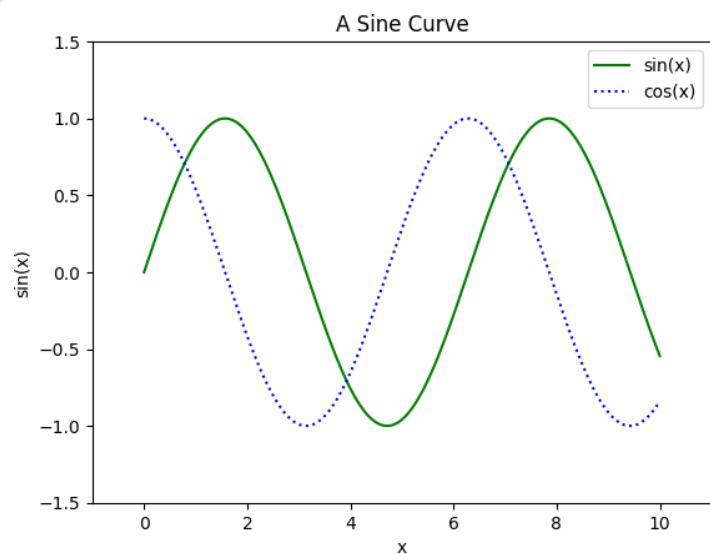


Axis limit with label and legend

```

import matplotlib.pyplot as plt
import numpy as np
fig =
plt.figure()
ax =
plt.axes()
x = np.linspace(0, 10, 1000)
plt.xlim(-1, 11)
plt.ylim(-1.5, 1.5);
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b',
label='cos(x)')
plt.title("A Sine Curve")
plt.xlabel("x")
plt.ylabel("sin(x)");
plt.legend();

```



Simple Scatter Plots

Another commonly used plot type is the simple scatter plot, a close cousin of the line plot. Instead of points being joined by line segments, here the points are represented individually with a dot, circle, or other shape.

Syntax

```
plt.plot(x, y, 'type of symbol ', color);
```

Example

```
plt.plot(x, y, 'o', color='black');
```

- The third argument in the function call is a character that represents the type of symbol used for the plotting. Just as you can specify options such as '-' and '--' to control the line style, the marker style has its own set of short string codes.

Example

- Various symbols used to specify ['o', '.', ',', 'x', '+', 'v', '^', '<', '>', 's', 'd']

- Short hand assignment of line, symbol and color also allowed.

```
plt.plot(x, y, '-ok');
```

- Additional arguments in plt.plot()

We can specify some other parameters related with scatter plot which makes it more attractive. They are color, marker size, linewidth, marker face color, marker edge color, marker edge width, etc

Example

```
plt.plot(x, y, '-p', color='gray',
         markersize=15, linewidth=4,
         markerfacecolor='white',
         markeredgecolor='gray',
         markeredgewidth=2)
plt.ylim(-1.2, 1.2);
```

Scatter Plots with plt.scatter

- A second, more powerful method of creating scatter plots is the plt.scatter function, which can be used very similarly to the plt.plot function
- plt.scatter(x, y, marker='o');
- The primary difference of plt.scatter from plt.plot is that it can be used to create scatter plots where the properties of each individual point (size, face color, edge color, etc.) can be individually controlled or mapped to data.
- Notice that the color argument is automatically mapped to a color scale (shown here by the colorbar() command), and the size argument is given in pixels.
- Cmap – color map used in scatter plot gives different color combinations.

Perceptually Uniform Sequential

```
['viridis', 'plasma', 'inferno', 'magma']
```

Sequential

```
['Greys', 'Purples', 'Blues', 'Greens', 'Oranges', 'Reds', 'YlOrBr', 'YlOrRd',
 'OrRd', 'PuRd', 'RdPu', 'BuPu', 'GnBu', 'PuBu', 'YlGnBu', 'PuBuGn', 'BuGn',
 'YlGn']
```

Sequential (2)

```
['binary', 'gist_yarg', 'gist_gray', 'gray', 'bone', 'pink', 'spring', 'summer',
 'autumn', 'winter', 'cool', 'Wistia', 'hot', 'afmhot', 'gist_heat', 'copper']
```

Diverging

```
['PiYG', 'PRGn', 'BrBG', 'PuOr', 'RdGy', 'RdBu', 'RdYlBu', 'RdYlGn', 'Spectral',  
'coolwarm', 'bwr', 'seismic']
```

Qualitative

```
['Pastel1', 'Pastel2', 'Paired', 'Accent', 'Dark2', 'Set1', 'Set2', 'Set3',  
'tab10', 'tab20', 'tab20b', 'tab20c']
```

Miscellaneous

```
['flag', 'prism', 'ocean', 'gist_earth', 'terrain', 'gist_stern', 'gnuplot',  
'gnuplot2', 'CMRmap', 'cubehelix', 'brg', ' hsv', 'gist_rainbow', 'rainbow',  
'jet', 'nipy_spectral', 'gist_ncar']
```

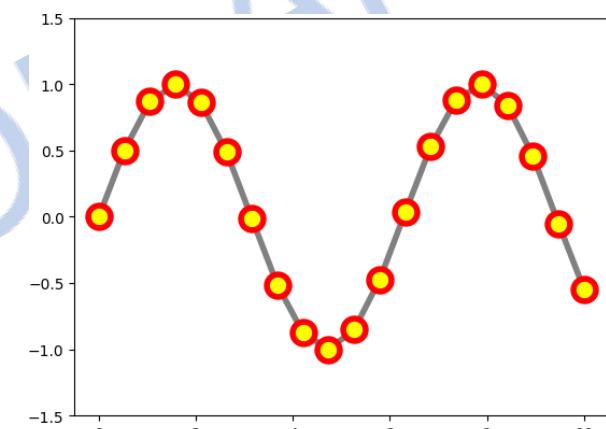
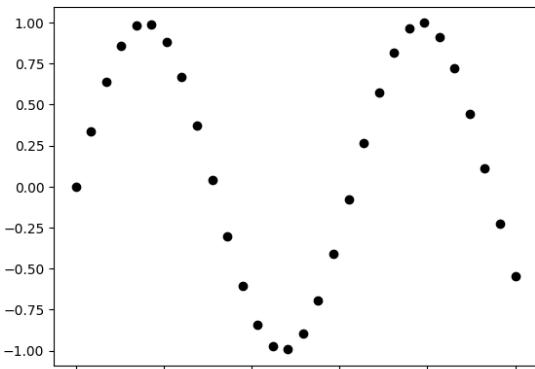
Example programs.

Simple scatter plot.

```
import numpy as np  
import matplotlib.pyplot as plt  
x = np.linspace(0, 10, 30)  
y = np.sin(x)  
plt.plot(x, y, 'o', color='black');
```

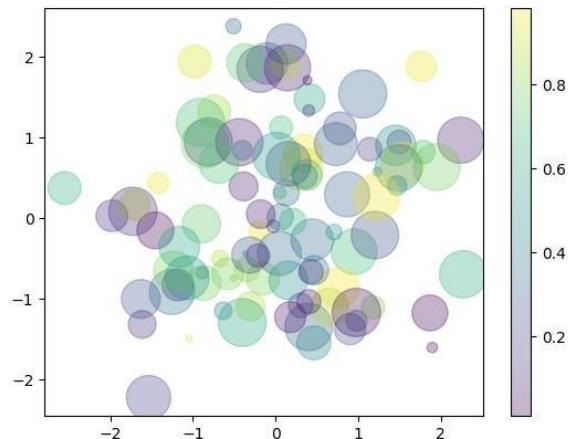
Scatter plot with edge color, face color, size, and width of marker. (Scatter plot with line)

```
import numpy as np  
import matplotlib.pyplot as plt  
x = np.linspace(0, 10, 20)  
y = np.sin(x)  
plt.plot(x, y, '-o',  
color='gray',  
markersize=15,  
linewidth=4,  
markerfacecolor='yellow',  
markeredgecolor='red',  
markeredgewidth=4)  
plt.ylim(-1.5, 1.5);
```



Scatter plot with random colors, size and transparency

```
import numpy as np  
import matplotlib.pyplot as plt  
rng =  
np.random.RandomState(0)  
x =  
rng.randn(100)  
y = rng.randn(100)  
colors =  
rng.rand(100)  
sizes = 1000 * rng.rand(100)  
plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,  
map='viridis')plt.colorbar()
```



For any scientific measurement, accurate accounting for errors is nearly as important, if not more important, than accurate reporting of the number itself. For example, imagine that I am using some astrophysical observations to estimate the Hubble Constant, the local measurement of the expansion rate of the Universe. In visualization of data and results, showing these errors effectively can make a plot convey much more complete information.

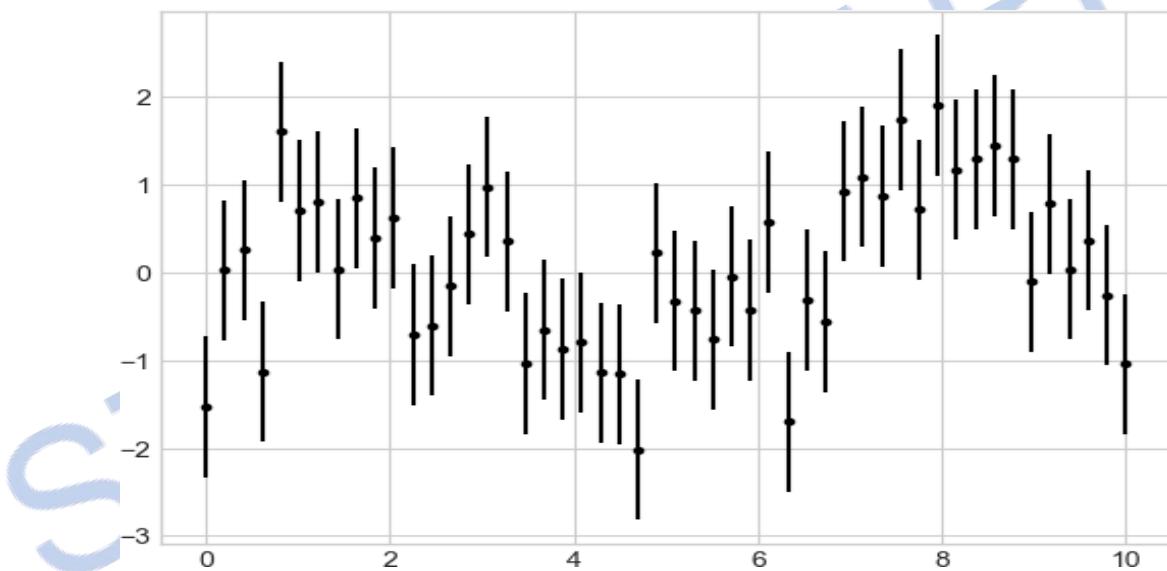
Types of errors

- Basic Errorbars
- Continuous Errors

Basic Errorbars

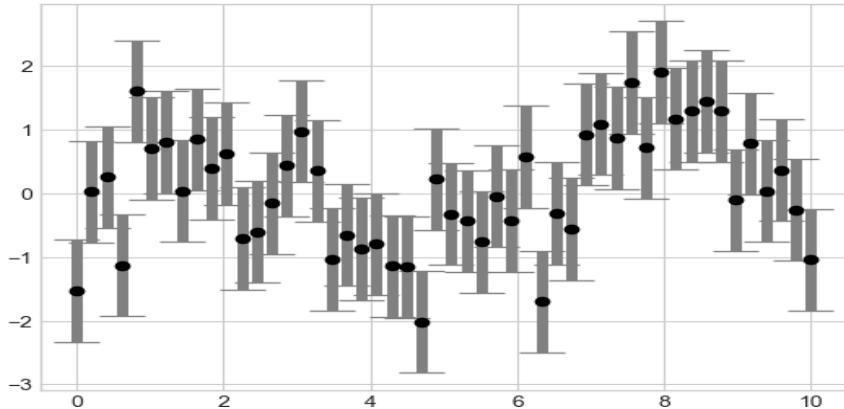
A basic errorbar can be created with a single Matplotlib function call.

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
x = np.linspace(0, 10, 50)
dy = 0.8
y = np.sin(x) + dy * np.random.randn(50)
plt.errorbar(x, y, yerr=dy, fmt='.k');
```



- Here the fmt is a format code controlling the appearance of lines and points, and has the same syntax as the shorthand used in plt.plot()
- In addition to these basic options, the errorbar function has many options to fine tune the outputs. Using these additional options you can easily customize the aesthetics of your errorbar plot.

```
plt.errorbar(x, y, yerr=dy, fmt='o', color='black', ecolor='lightgray', elinewidth=3, capsize=0);
```



Continuous Errors

- In some situations it is desirable to show errorbars on continuous quantities. Though Matplotlib does not have a built-in convenience routine for this type of application, it's relatively easy to combine primitives like `plt.plot` and `plt.fill_between` for a useful result.
- Here we'll perform a simple Gaussian process regression (GPR), using the Scikit-Learn API. This is a method of fitting a very flexible nonparametric function to data with a continuous measure of the uncertainty.

Density and Contour Plots

To display three-dimensional data in two dimensions using contours or color-coded regions. There are three Matplotlib functions that can be helpful for this task:

- `plt.contour` for contour plots,
- `plt.contourf` for filled contour plots, and
- `plt.imshow` for showing images.

Visualizing a Three-Dimensional Function

A contour plot can be created with the `plt.contour` function.

It takes three arguments:

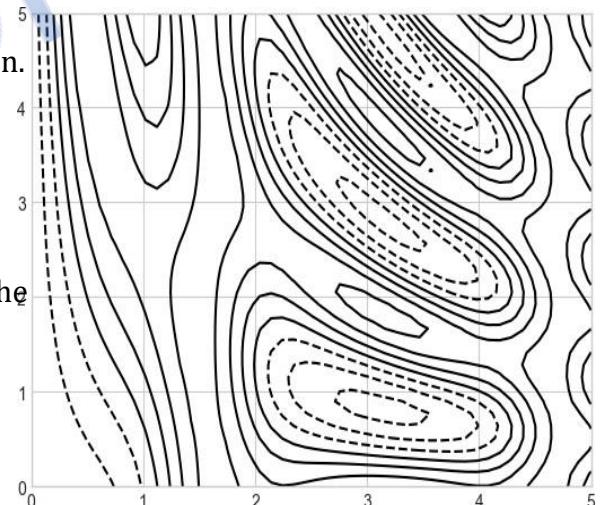
- a grid of x values,
- a grid of y values, and
- a grid of z values.

The x and y values represent positions on the plot, and the z values will be represented by the contour levels.

The way to prepare such data is to use the `np.meshgrid` function, which builds two-dimensional grids from one-dimensional arrays:

Example

```
def f(x, y):
    return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)
x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
plt.contour(X, Y, Z, colors='black');
```



- Notice that by default when a single color is used, negative values are represented by dashed lines, and positive values by solid lines.
- Alternatively, you can color-code the lines by specifying a colormap with the `cmap` argument.
- We'll also specify that we want more lines to be drawn—20 equally spaced intervals within the data range.

- ```
plt.contour(X, Y, Z, 20, cmap='RdGy');
```
- One potential issue with this plot is that it is a bit “splotchy.” That is, the color steps are discrete rather than continuous, which is not always what is desired.
  - You could remedy this by setting the number of contours to a very high number, but this results in a rather inefficient plot: Matplotlib must render a new polygon for each step in the level.
  - A better way to handle this is to use the `plt.imshow()` function, which interprets a two-dimensional grid of data as an image.

*There are a few potential gotchas with imshow().*

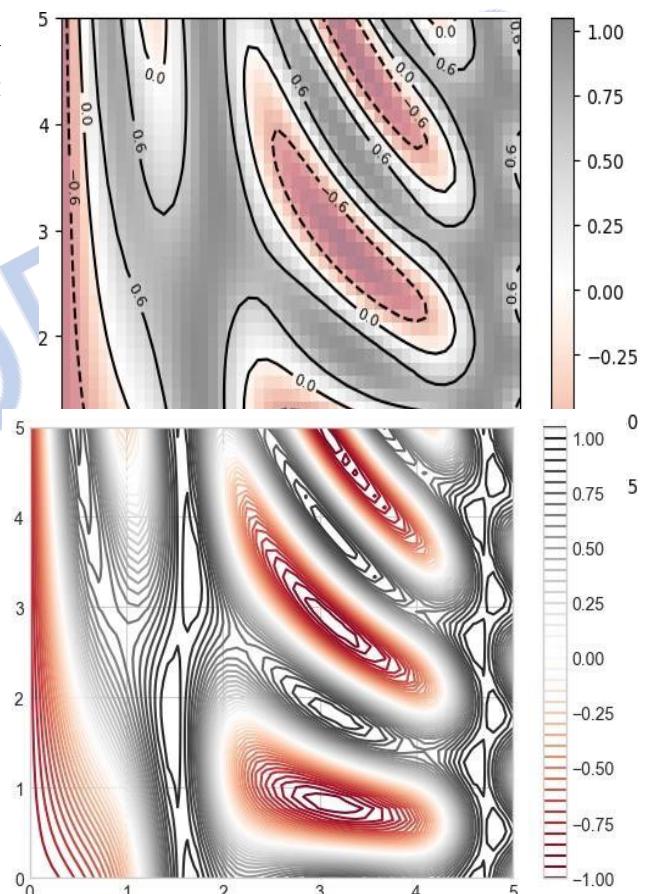
- `plt.imshow()` doesn't accept an x and y grid, so you must manually specify the extent [xmin, xmax, ymin, ymax] of the image on the plot.
- `plt.imshow()` by default follows the standard image array definition where the origin is in the upper left, not in the lower left as in most contour plots. This must be changed when showing gridded data.
- `plt.imshow()` will automatically adjust the axis aspect ratio to match the input data; you can change this by setting, for example, `plt.axis(aspect='image')` to make x and y units match.

Finally, it can sometimes be useful to combine contour plots and image plots. we'll use a partially transparent background image (with transparency set via the alpha parameter) and over-plot contours with labels on the contours themselves (using the `plt.clabel()` function):

```
contours = plt.contour(X, Y, Z, 3, colors='black')
plt.clabel(contours, inline=True, fontsize=8)
plt.imshow(Z, extent=[0, 5, 0, 5], origin='lower',
 cmap='RdGy', alpha=0.5)
plt.colorbar();
```

#### *Example Program*

```
import numpy as np
import matplotlib.pyplot as plt
def f(x, y):
 return np.sin(x) ** 10 + np.cos(10 + y * x) *
 np.cos(x)
x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
plt.imshow(Z, extent=[0, 10, 0, 10],
 origin='lower', cmap='RdGy')
plt.colorbar()
```



## Histograms

- Histogram is the simple plot to represent the large data set. A histogram is a graph showing frequency distributions. It is a graph showing the number of observations within each given interval.

#### *Parameters*

- `plt.hist( )` is used to plot histogram. The `hist()` function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

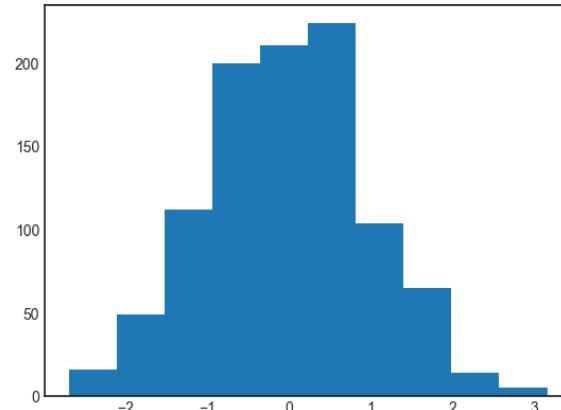
- **bins** - A histogram displays numerical data by grouping data into "bins" of equal width. Each bin is plotted as a bar whose height corresponds to how many data points are in that bin. Bins are also sometimes called "intervals", "classes", or "buckets".
  - **normed** - Histogram normalization is a technique to distribute the frequencies of the histogram over a wider range than the current range.
  - **x** - (n,) array or sequence of (n,) arrays Input values, this takes either a single array or a sequence of arrays which are not required to be of the same length.
  - **histtype** - {'bar', 'barstacked', 'step', 'stepfilled'}, optionalThe type of histogram to draw.
    - 'bar' is a traditional bar-type histogram. If multiple data are given the bars are arranged side by side.
    - 'barstacked' is a bar-type histogram where multiple data are stacked on top of each other.
    - 'step' generates a lineplot that is by default unfilled.
    - 'stepfilled' generates a lineplot that is by default filled.Default is 'bar'
  - **align** - {'left', 'mid', 'right'}, optional Controls how the histogram is plotted.
    - 'left': bars are centered on the left bin edges.
    - 'mid': bars are centered between the bin edges.
    - 'right': bars are centered on the right bin edges.Default is 'mid'
  - **orientation** - {'horizontal', 'vertical'}, optional If 'horizontal', barh will be used for bar-type histograms and the bottom kwarg will be the left edges.
  - **color** - color or array\_like of colors or None, optional Color spec or sequence of color specs, one per dataset. Default (None) uses the standard line color sequence.
- Default is None
- **label** - str or None, optional. Default is None

#### Other parameter

- **\*\*kwargs** - Patch properties, it allows us to pass a variable number of keyword arguments to a python function. \*\* denotes this type of function.

#### Example

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
data = np.random.randn(1000)
plt.hist(data);
```



The hist() function has many options to tune both the calculation and the display; here's an example of a more customized histogram.

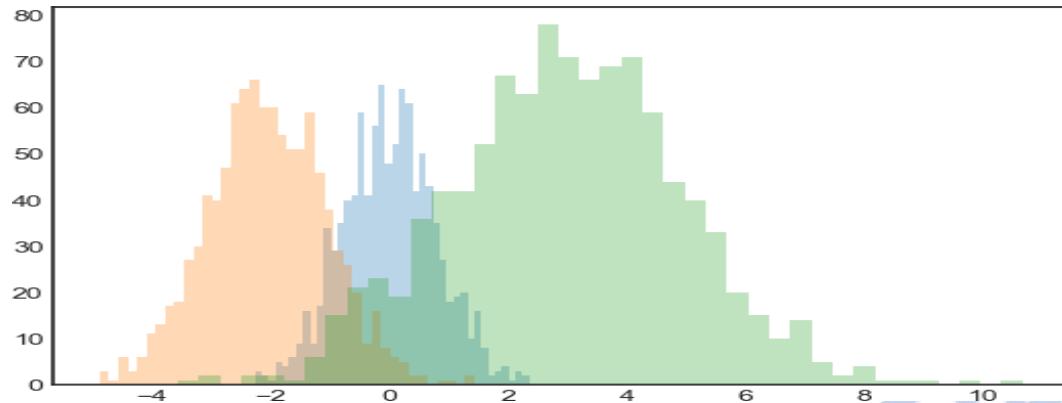
```
plt.hist(data, bins=30, alpha=0.5, histtype='stepfilled', color='steelblue', edgecolor='none');
```

The plt.hist docstring has more information on other customization options available. I find this combination of histtype='stepfilled' along with some transparency alpha to be very useful when comparing histograms of several distributions

```

x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)
kwargs = dict(histtype='stepfilled', alpha=0.3, bins=40)
plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs);

```



### *Two-Dimensional Histograms and Binnings*

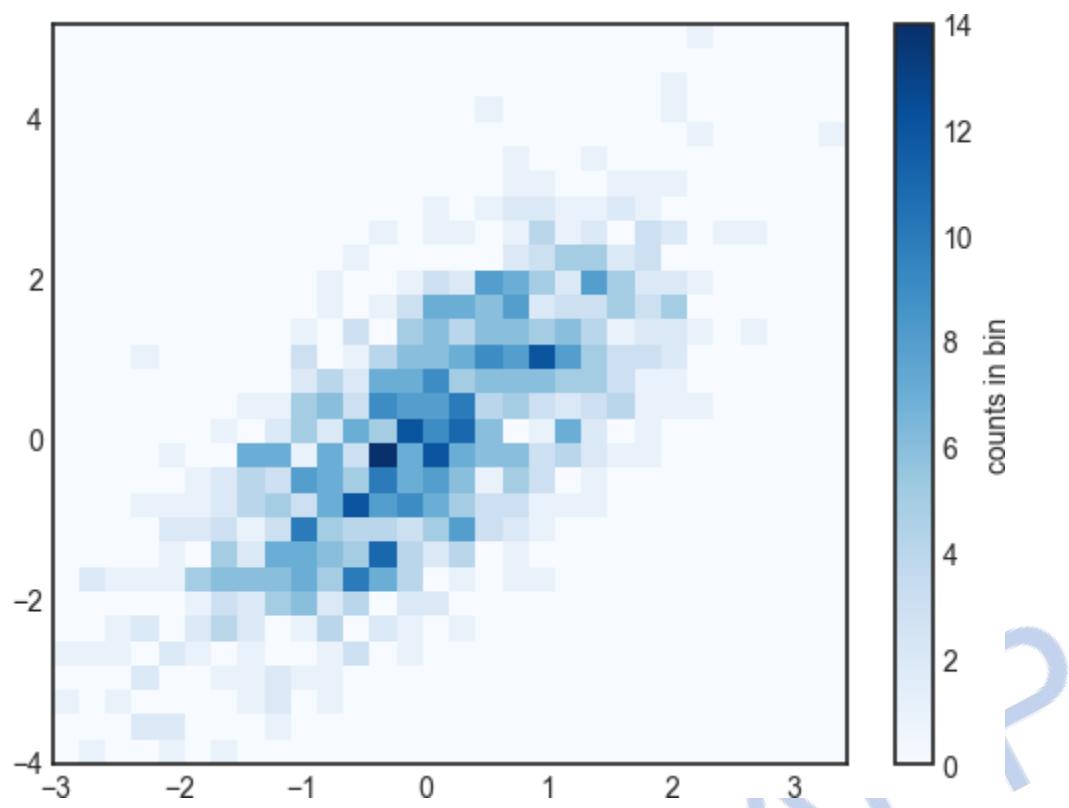
- We can create histograms in two dimensions by dividing points among two dimensional bins.
- We would define x and y values. Here for example We'll start by defining some data—an x and y array drawn from a multivariate Gaussian distribution:
- Simple way to plot a two-dimensional histogram is to use Matplotlib's `plt.hist2d()` function

### *Example*

```

mean = [0, 0]
cov = [[1, 1], [1, 2]]
x, y = np.random.multivariate_normal(mean, cov, 1000).T
plt.hist2d(x, y, bins=30, cmap='Blues')
cb = plt.colorbar()
cb.set_label('counts in bin')

```



## Legends

Plot legends give meaning to a visualization, assigning labels to the various plot elements. We previously saw how to create a simple legend; here we'll take a look at customizing the placement and aesthetics of the legend in Matplotlib.

Plot legends give meaning to a visualization, assigning labels to the various plot elements. We previously saw how to create a simple legend; here we'll take a look at customizing the placement and aesthetics of the legend in Matplotlib

```
plt.plot(x, np.sin(x), '-b', label='Sine')
plt.plot(x, np.cos(x), '--r', label='Cosine')
plt.legend();
```

### *Customizing Plot Legends*

**Location and turn off the frame** - We can specify the location and turn off the frame. By the parameter loc and framem.

```
ax.legend(loc='upper left', frameon=False)
fig
```

**Number of columns** - We can use the ncol command to specify the number of columns in the legend.

```
ax.legend(frameon=False, loc='lower center', ncol=2)
fig
```

*Rounded box, shadow and frame transparency*

We can use a rounded box (`fancybox`) or add a shadow, change the transparency (alpha value) of the frame, or change the padding around the text.

```
ax.legend(fancybox=True, framealpha=1, shadow=True, borderpad=1)
fig
```

### Choosing Elements for the Legend

- The legend includes all labeled elements by default. We can change which elements and labels appear in the legend by using the objects returned by plot commands.
- The `plt.plot()` command is able to create multiple lines at once, and returns a list of created line instances. Passing any of these to `plt.legend()` will tell it which to identify, along with the labels we'd like to specify.

```
y = np.sin(x[:, np.newaxis] + np.pi * np.arange(0, 2, 0.5))
```

```
lines = plt.plot(x, y)
```

```
plt.legend(lines[:2],['first','second']);
```

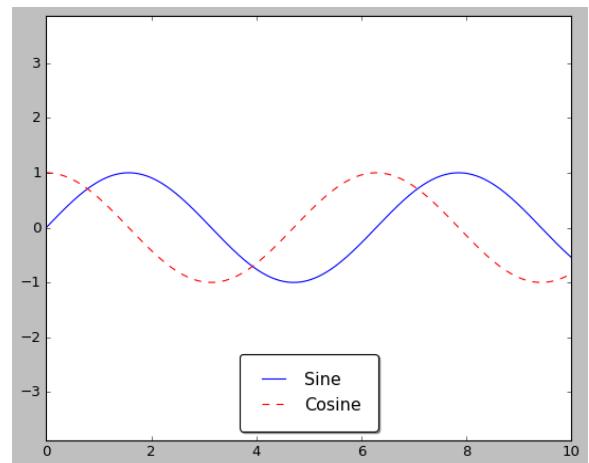
```
Applying label individually.
```

```
plt.plot(x, y[:, 0], label='first')
```

```
plt.plot(x, y[:, 1], label='second')
```

```
plt.plot(x, y[:, 2:])
```

```
plt.legend(framealpha=1, frameon=True);
```



### Multiple legends

It is only possible to create a single legend for the entire plot. If you try to create a second legend using `plt.legend()` or `ax.legend()`, it will simply override the first one. We can work around this by creating a

new legend artist from scratch, and then using the lower-level `ax.add_artist()` method to manually add the second artist to the plot

### Example

```
import matplotlib.pyplot as plt
plt.style.use('classic')
import numpy as np
x = np.linspace(0, 10, 1000)
ax.legend(loc='lower center', frameon=True, shadow=True, borderpad=1, fancybox=True)
fig
```

## Color Bars

In Matplotlib, a color bar is a separate axes that can provide a key for the meaning of colors in a plot. For continuous labels based on the color of points, lines, or regions, a labeled color bar can be a great tool. The simplest colorbar can be created with the `plt.colorbar()` function.

### Customizing Colorbars

#### Choosing color map.

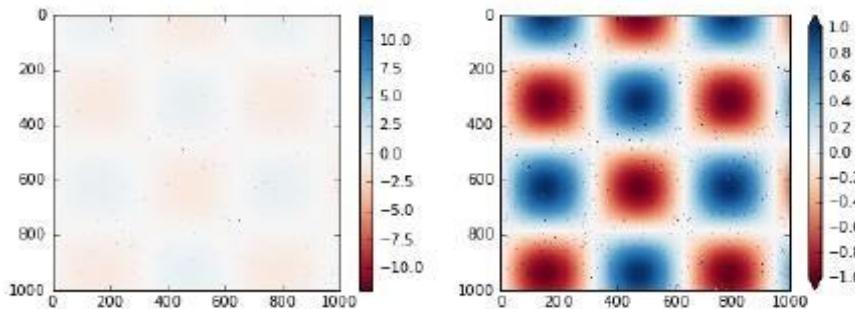
We can specify the colormap using the `cmap` argument to the plotting function that is creating the visualization. Broadly, we can know three different categories of colormaps:

- Sequential colormaps** - These consist of one continuous sequence of colors (e.g., binary or viridis).
- Divergent colormaps** - These usually contain two distinct colors, which show positive and negative deviations from a mean (e.g., RdBu or PuOr).
- Qualitative colormaps** - These mix colors with no particular sequence (e.g., rainbow or jet).

## Color limits and extensions

- Matplotlib allows for a large range of colorbar customization. The colorbar itself is simply an instance of plt.Axes, so all of the axes and tick formatting tricks we've learned are applicable.
- We can narrow the color limits and indicate the out-of-bounds values with a triangular arrow at the top and bottom by setting the extend property.

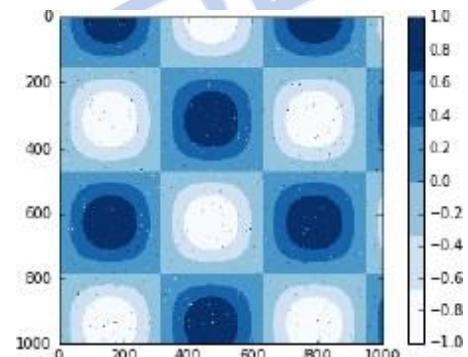
```
plt.subplot(1, 2, 2)
plt.imshow(I, cmap='RdBu')
plt.colorbar(extend='both')
plt.clim(-1, 1);
```



## Discrete colorbars

Colormaps are by default continuous, but sometimes you'd like to represent discrete values. The easiest way to do this is to use the plt.cm.get\_cmap() function, and pass the name of a suitable colormap along with the number of desired bins.

```
plt.imshow(I, cmap=plt.cm.get_cmap('Blues', 6))
plt.colorbar()
plt.clim(-1, 1);
```



## Subplots

- Matplotlib has the concept of subplots: groups of smaller axes that can exist together within a single figure.
- These subplots might be insets, grids of plots, or other more complicated layouts.
- We'll explore four routines for creating subplots in Matplotlib.
  - plt.axes: Subplots by Hand
  - plt.subplot: Simple Grids of Subplots
  - plt.subplots: The Whole Grid in One Go
  - plt.GridSpec: More Complicated Arrangements

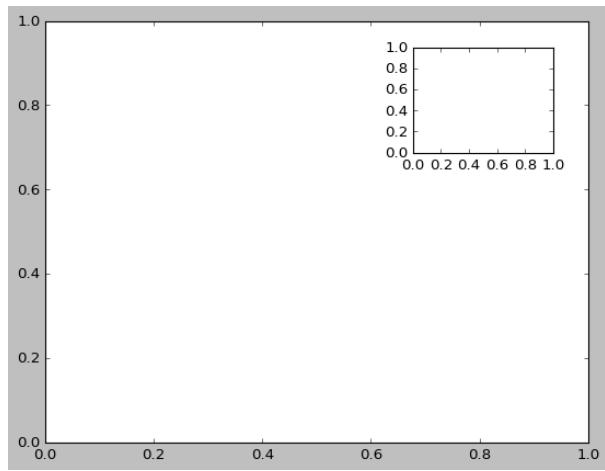
### plt.axes: Subplots by Hand

- The most basic method of creating an axes is to use the plt.axes function. As we've seen previously, by default this creates a standard axes object that fills the entire figure.
- plt.axes also takes an optional argument that is a list of four numbers in the figure coordinate system.
- These numbers represent [bottom, left, width, height] in the figure coordinate system, which ranges from 0 at the bottom left of the figure to 1 at the top right of the figure.

For example,

we might create an inset axes at the top-right corner of another axes by setting the x and y position to 0.65 (that is, starting at 65% of the width and 65% of the height of the figure) and the x and y extents to 0.2 (that is, the size of the axes is 20% of the width and 20% of the height of the figure).

```
import matplotlib.pyplot as plt
import numpy as np
ax1 = plt.axes() # standard axes
ax2 = plt.axes([0.65, 0.65, 0.2, 0.2])
```

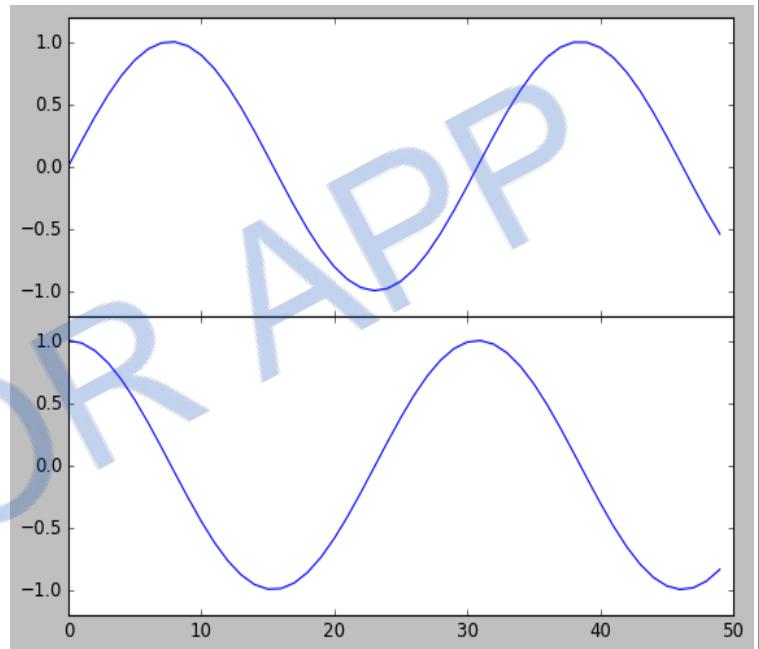


### Vertical sub plot

The equivalent of `plt.axes()` command within the object-oriented interface is `ig.add_axes()`. Let's use this to create two vertically stacked axes.

```
fig = plt.figure()
ax1 = fig.add_axes([0.1, 0.5, 0.8, 0.4],
xticklabels=[], ylim=(-1.2, 1.2))
ax2 = fig.add_axes([0.1, 0.1, 0.8, 0.4],
ylim=(-1.2, 1.2))
x = np.linspace(0, 10)
ax1.plot(np.sin(x))
ax2.plot(np.cos(x));
```

- We now have two axes (the top with no tick labels) that are just touching: the bottom of the upper panel (at position 0.5) matches the top of the lower panel (at position 0.1+ 0.4).
- If the axis value is changed in second plot both the plots are separated with each other, example `ax2 = fig.add_axes([0.1, 0.01, 0.8, 0.4]`

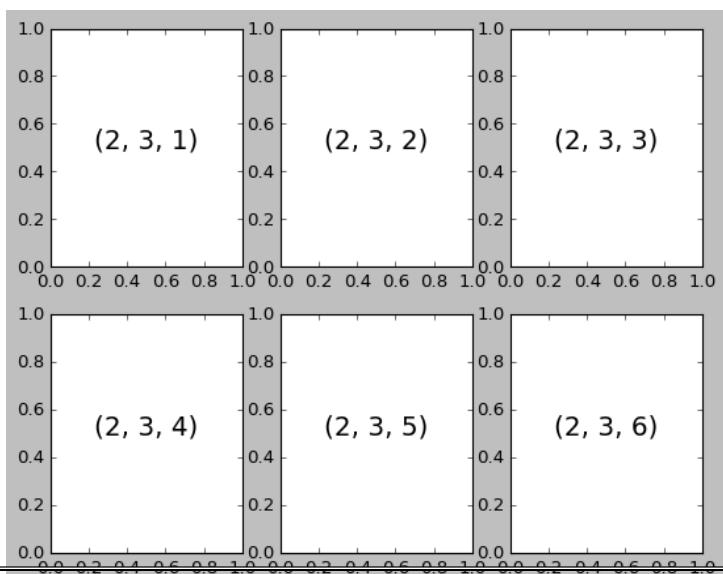


### `plt.subplot`: Simple Grids of Subplots

- Matplotlib has several convenience routines to align columns or rows of subplots.
- The lowest level of these is `plt.subplot()`, which creates a single subplot within a grid.

- This command takes three integer arguments—the number of rows, the number of columns, and the index of the plot to be created in this scheme, which runs from the upper left to the bottom right

```
for i in range(1, 7):
plt.subplot(2, 3, i)
plt.text(0.5, 0.5, str((2, 3, i)),
fontsize=18, ha='center')
```

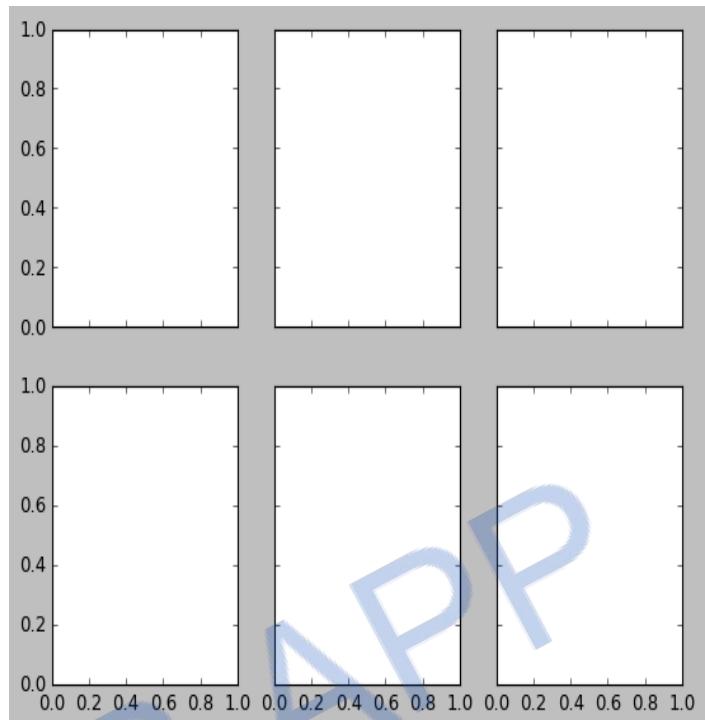


### `plt.subplots: The Whole Grid in One Go`

- The approach just described can become quite tedious when you're creating a large grid of subplots, especially if you'd like to hide the x- and y-axis labels on the inner plots.
- For this purpose, `plt.subplots()` is the easier tool to use (note the s at the end of `subplots`).
- Rather than creating a single subplot, this function creates a full grid of subplots in a single line, returning them in a NumPy array.
- The arguments are the number of rows and number of columns, along with optional keywords `sharex` and `sharey`, which allow you to specify the relationships between different axes.
- Here we'll create a  $2 \times 3$  grid of subplots, where all axes in the same row share their y-axis scale, and all axes in the same column share their x-axis scale

```
fig, ax = plt.subplots(2, 3, sharex='col', sharey='row')
```

Note that by specifying `sharex` and `sharey`, we've automatically removed inner labels on the grid to make the plot cleaner.

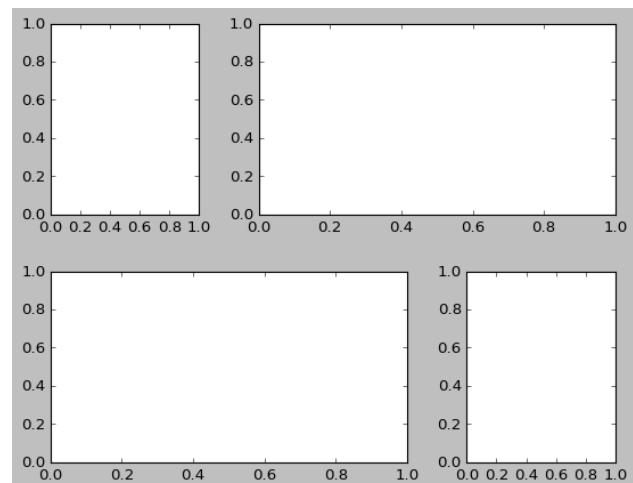


### `plt.GridSpec: More Complicated Arrangements`

To go beyond a regular grid to subplots that span multiple rows and columns, `plt.GridSpec()` is the best tool. The `plt.GridSpec()` object does not create a plot by itself; it is simply a convenient interface that is recognized by the `plt.subplot()` command.

For example, a `gridspec` for a grid of two rows and three columns with some specified width and height spacing looks like this:

```
grid = plt.GridSpec(2, 3, wspace=0.4, hspace=0.3)
From this we can specify subplot locations and extent
plt.subplot(grid[0, 0])
plt.subplot(grid[0, 1:])
plt.subplot(grid[1, :2])
plt.subplot(grid[1, 2]);
```



## Text and Annotation

- The most basic types of annotations we will use are axes labels and titles, here we will see some more visualization and annotation information's.

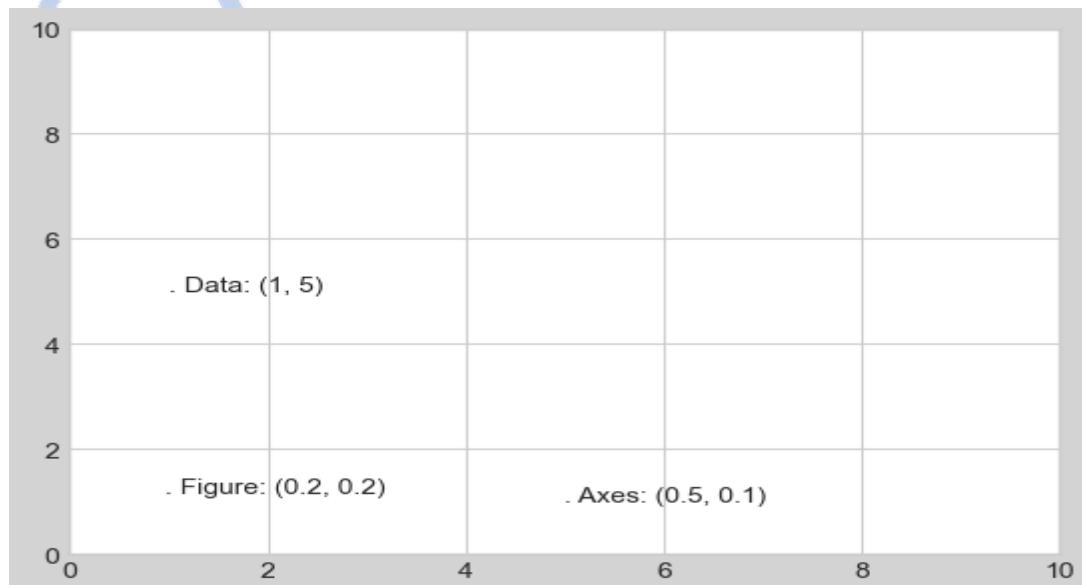
- Text annotation can be done manually with the plt.text/ax.text command, which will place text at a particular x/y value.
- The ax.text method takes an x position, a y position, a string, and then optional keywords specifying the color, size, style, alignment, and other properties of the text. Here we used ha='right' and ha='center', where ha is short for horizontal alignment.

### Transforms and Text Position

- We anchored our text annotations to data locations. Sometimes it's preferable to anchor the text to a position on the axes or figure, independent of the data. In Matplotlib, we do this by modifying the transform.
- Any graphics display framework needs some scheme for translating between coordinate systems.
- Mathematically, such coordinate transformations are relatively straightforward, and Matplotlib has a well-developed set of tools that it uses internally to perform them (the tools can be explored in the matplotlib.transforms submodule).
- There are three predefined transforms that can be useful in this situation.
  - ax.transData - Transform associated with data coordinates
  - ax.transAxes - Transform associated with the axes (in units of axes dimensions)
  - fig.transFigure - Transform associated with the figure (in units of figure dimensions)

### Example

```
import matplotlib.pyplot as plt
import matplotlib as mpl
plt.style.use('seaborn-whitegrid')
import numpy as np
import pandas as pd
fig, ax = plt.subplots(facecolor='lightgray')
ax.axis([0, 10, 0, 10])
transform=ax.transData is the default, but we'll specify it anyway
ax.text(1, 5, ". Data: (1, 5)", transform=ax.transData)
ax.text(0.5, 0.1, ". Axes: (0.5, 0.1)", transform=ax.transAxes)
ax.text(0.2, 0.2, ". Figure: (0.2, 0.2)", transform=fig.transFigure);
```



Note that by default, the text is aligned above and to the left of the specified coordinates; here the “.” at the beginning of each string will approximately mark the given coordinate location.

The transData coordinates give the usual data coordinates associated with the x- and y-axis labels. The transAxes coordinates give the location from the bottom-left corner of the axes (here the white box) as a fraction of the axes size.

The transfigure coordinates are similar, but specify the position from the bottom left of the figure (here the gray box) as a fraction of the figure size.

Notice now that if we change the axes limits, it is only the transData coordinates that will be affected, while the others remain stationary.

#### *Arrows and Annotation*

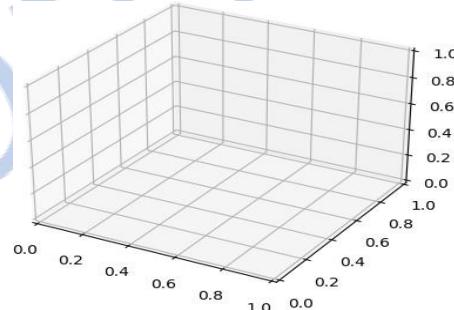
- Along with tick marks and text, another useful annotation mark is the simple arrow.
- Drawing arrows in Matplotlib is not much harder because there is a plt.arrow() function available.
- The arrows it creates are SVG (scalable vector graphics) objects that will be subject to the varying aspectratio of your plots, and the result is rarely what the user intended.
- The arrow style is controlled through the arrowprops dictionary, which has numerous options available.

## **Three-Dimensional Plotting in Matplotlib**

We enable three-dimensional plots by importing the mplot3d toolkit, included with the main Matplotlib installation.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
fig = plt.figure()
ax = plt.axes(projection='3d')
```

With this 3D axes enabled, we can now plot a varietyof three-dimensional plot types.



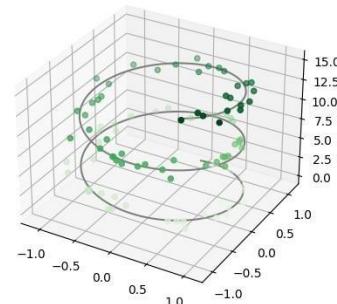
#### *Three-Dimensional Points and Lines*

The most basic three-dimensional plot is a line or scatter plot created from sets of (x, y, z) triples.

In analogy with the more common two-dimensional plots discussed earlier, we can create these using the ax.plot3D

and ax.scatter3D functions

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
ax = plt.axes(projection='3d')
Data for a three-dimensional line
zline = np.linspace(0, 15, 1000)
xline = np.sin(zline)
yline = np.cos(zline)
ax.plot3D(xline, yline, zline, 'gray')
Data for three-dimensional scattered points
zdata = 15 * np.random.random(100)
xdata = np.sin(zdata) + 0.1 * np.random.randn(100)
ydata = np.cos(zdata) + 0.1 * np.random.randn(100)
```



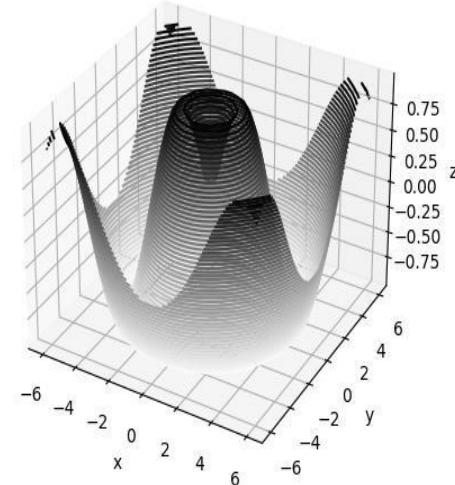
```
ax.scatter3D(xdata, ydata, zdata, c=zdata, cmap='Greens');plt.show()
```

Notice that by default, the scatter points have their transparency adjusted to give a sense of depth on the page

### Three-Dimensional Contour Plots

- `mplot3d` contains tools to create three-dimensional relief plots using the same inputs.
- Like two-dimensional `ax.contour` plots, `ax.contour3D` requires all the input data to be in the form of two-dimensional regular grids, with the Z data evaluated at each point.
- Here we'll show a three-dimensional contour diagram of a three dimensional sinusoidal function

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
def f(x, y):
 return np.sin(np.sqrt(x ** 2 + y ** 2))
x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.contour3D(X, Y, Z, 50, cmap='binary')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.show()
```



Sometimes the default viewing angle is not optimal, in which case we can use the `view_init` method to set the elevation and azimuthal angles.

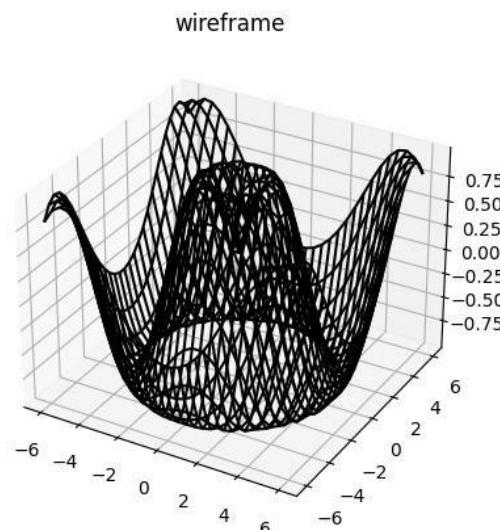
```
ax.view_init(60, 35)
fig
```

### Wire frames and Surface Plots

- Two other types of three-dimensional plots that work on gridded data are wireframes and surface plots.
- These take a grid of values and project it onto the specified threedimensional surface, and can make the resulting three-dimensional forms quite easy to visualize.

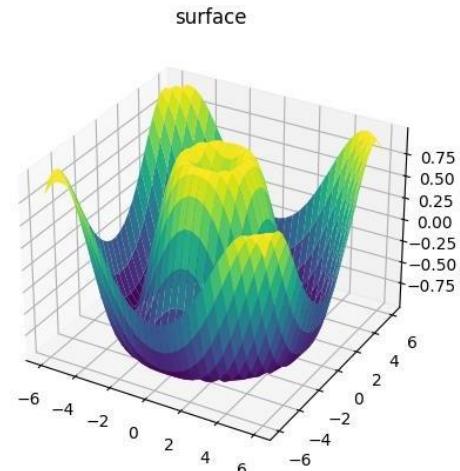
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_wireframe(X, Y, Z, color='black')
ax.set_title('wireframe');
plt.show()
```

- A surface plot is like a wireframe plot, but each face of the wireframe is a filled polygon.



- Adding a colormap to the filled polygons can aid perception of the topology of the surface being visualized

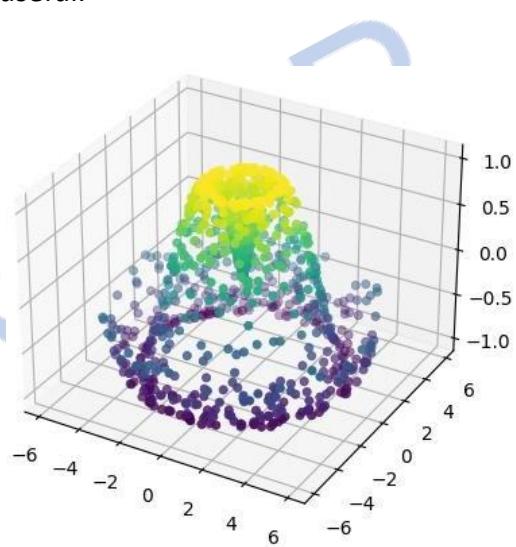
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
cmap='viridis', edgecolor='none')
ax.set_title('surface')
plt.show()
```



### Surface Triangulations

- For some applications, the evenly sampled grids required by the preceding routines are overly restrictive and inconvenient.
- In these situations, the triangulation-based plots can be very useful.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
theta = 2 * np.pi * np.random.random(1000)
r = 6 * np.random.random(1000)
x = np.ravel(r * np.sin(theta))
y = np.ravel(r * np.cos(theta))
z = f(x, y)
ax = plt.axes(projection='3d')
ax.scatter(x, y, z, c=z, cmap='viridis', linewidth=0.5)
```



### Geographic Data with Basemap

- One common type of visualization in data science is that of geographic data.
- Matplotlib's main tool for this type of visualization is the Basemap toolkit, which is one of several Matplotlib toolkits that live under the `mpl_toolkits` namespace.
- Basemap is a useful tool for Python users to have in their virtual toolbelts
- Installation of Basemap. Once you have the Basemap toolkit installed and imported, geographic plots also require the PIL package in Python 2, or the pillow package in Python 3.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
plt.figure(figsize=(8, 8))
m = Basemap(projection='ortho', resolution=None,
lat_0=50, lon_0=-100)
m.bluemarble(scale=0.5);
```

- Matplotlib axes that understand spherical coordinates and allows us to easily over-plot data on the map



- We'll use an etopo image (which shows topographical features both on land and under the ocean) as the map background

Program to display particular area of the map with latitude and longitude lines

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
from itertools import chain
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution=None,
width=8E6, height=8E6,
lat_0=45, lon_0=-100.)
m.etopo(scale=0.5, alpha=0.5)
def draw_map(m, scale=0.2):
draw a shaded-relief image
 m.shadedrelief(scale=scale)
lats and longs are returned as a dictionary
lats = m.drawparallels(np.linspace(-90, 90, 13))
lons = m.drawmeridians(np.linspace(-180, 180, 13))
keys contain the plt.Line2D instances
lat_lines = chain(*(tup[1][0] for tup in lats.items()))
lon_lines = chain(*(tup[1][0] for tup in lons.items()))
all_lines = chain(lat_lines, lon_lines)
cycle through these lines and set the desired style
for line in all_lines:
 line.set(linestyle='-', alpha=0.3, color='r')
```



### *Map Projections*

The Basemap package implements several dozen such projections, all referenced by a short format code. Here we'll briefly demonstrate some of the more common ones.

- Cylindrical projections
- Pseudo-cylindrical projections
- Perspective projections
- Conic projections

### *Cylindrical projection*

- The simplest of map projections are cylindrical projections, in which lines of constant latitude and longitude are mapped to horizontal and vertical lines, respectively.
- This type of mapping represents equatorial regions quite well, but results in extreme distortions near the poles.
- The spacing of latitude lines varies between different cylindrical projections, leading to different conservation properties, and different distortion near the poles.
- Other cylindrical projections are the Mercator (`projection='merc'`) and the cylindrical equal-area (`projection='cea'`) projections.
- The additional arguments to Basemap for this view specify the latitude (`lat`) and longitude (`lon`) of the lower-left corner (`llcrnr`) and upper-right corner (`urcrnr`) for the desired map, in units of degrees.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
```

```

fig = plt.figure(figsize=(8, 6), edgecolor='w')
m = Basemap(projection='cyl', resolution=None,
llcrnrlat=-90, urcrnrlat=90,
llcrnrlon=-180, urcrnrlon=180,)
draw_map(m)

```



### Pseudo-cylindrical projections

- Pseudo-cylindrical projections relax the requirement that meridians (lines of constant longitude) remain vertical; this can give better properties near the poles of the projection.
- The Mollweide projection (`projection='moll'`) is one common example of this, in which all meridians are elliptical arcs
- It is constructed so as to
- preserve area across the map: though there are distortions near the poles, the area of small patches reflects the true area.
- Other pseudo-cylindrical projections are the sinusoidal (`projection='sinu'`) and Robinson (`projection='robin'`) projections.
- The extra arguments to Basemap here refer to the central latitude (`lat_0`) and longitude (`lon_0`) for the desired map.



```

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
fig = plt.figure(figsize=(8, 6), edgecolor='w')
m = Basemap(projection='moll', resolution=None,
lat_0=0, lon_0=0)
draw_map(m)

```

### Perspective projections

- Perspective projections are constructed using a particular choice of perspective point, similar to if you photographed the Earth from a particular point in space (a point which, for some projections, technically lies within the Earth!).

- One common example is the orthographic projection (`projection='ortho'`), which shows one side of the globe as seen from a viewer at a very long distance.
- Thus, it can show only half the globe at a time.
- Other perspective-based projections include the gnomonic projection (`projection='gnom'`) and stereographic projection (`projection='stere'`).
- These are often the most useful for showing small portions of the map.

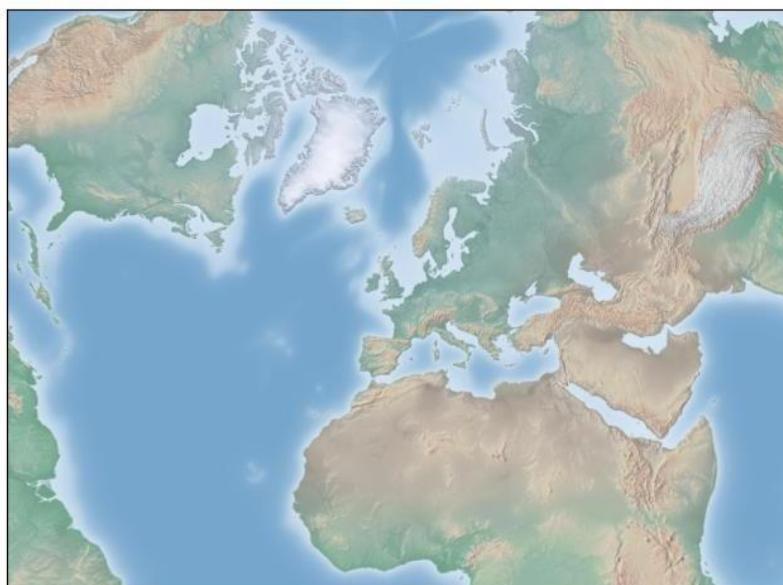
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='ortho', resolution=None,
lat_0=50, lon_0=0)
draw_map(m);
```



### *Conic projections*

- A conic projection projects the map onto a single cone, which is then unrolled.
- This can lead to very good local properties, but regions far from the focus point of the cone may become very distorted.
- One example of this is the Lambert conformal conic projection (`projection='lcc'`).
- It projects the map onto a cone arranged in such a way that two standard parallels (specified in Basemap by `lat_1` and `lat_2`) have well-represented distances, with scale decreasing between them and increasing outside of them.
- Other useful conic projections are the equidistant conic (`projection='eqdc'`) and the Albers equal-area (`projection='aea'`) projection

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution=None,
lon_0=0, lat_0=50, lat_1=45, lat_2=55, width=1.6E7, height=1.2E7)
draw_map(m)
```



## Drawing a Map Background

The Basemap package contains a range of useful functions for drawing borders of physical features like continents, oceans, lakes, and rivers, as well as political boundaries such as countries and US states and counties. The following are some of the available drawing functions that you may wish to explore using IPython's help features:

- Physical boundaries and bodies of water

`drawcoastlines()` - Draw continental coast lines

`drawlsmask()` - Draw a mask between the land and sea, for use with projecting images on one or the other

`drawmapboundary()` - Draw the map boundary, including the fill color for oceans

`drawrivers()` - Draw rivers on the map

`fillcontinents()` - Fill the continents with a given color; optionally fill lakes with another color

- Political boundaries

`drawcountries()` - Draw country

boundaries `drawstates()` - Draw US state

boundaries `drawcounties()` - Draw US

county boundaries

- Map features

`drawgreatcircle()` - Draw a great circle between two

points `drawparallels()` - Draw lines of constant latitude

`drawmeridians()` - Draw lines of constant longitude

`drawmapscale()` - Draw a linear scale on the map

- Whole-globe images

`bluemarble()` - Project NASA's blue marble image onto the

map `shadedrelief()` - Project a shaded relief image onto the

map `etopo()` - Draw an etopo relief image onto the map

`warpimage()` - Project a user-provided image onto the map

## Plotting Data on Maps

- The Basemap toolkit is the ability to over-plot a variety of data onto a map background.
- There are many map-specific functions available as methods of the Basemap

instance. Some of these map-specific methods are:

`contour()`/`contourf()` - Draw contour lines or filled

`contoursimshow()` - Draw an image

`pcolor()`/`pcolormesh()` - Draw a pseudocolor plot for irregular/regular

`meshesplot()` - Draw lines and/or markers

`scatter()` - Draw points with

`markersquiver()` - Draw vectors

`barbs()` - Draw wind barbs

`drawgreatcircle()` - Draw a great

circle

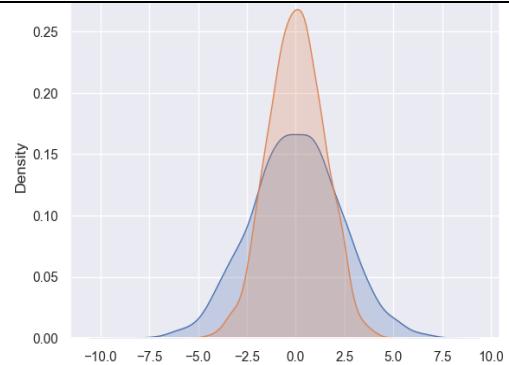
## Visualization with Seaborn

The main idea of Seaborn is that it provides high-level commands to create a variety of plot types useful for statistical data exploration, and even some statistical model fitting.

## Histograms, KDE, and densities

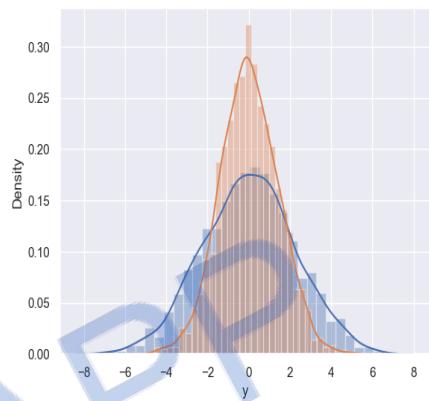
- In statistical data visualization, all you want is to plot histograms and joint distributions of variables. We have seen that this is relatively straightforward in Matplotlib
- Rather than a histogram, we can get a smooth estimate of the distribution using a kernel density estimation, which Seaborn does with `sns.kdeplot`

```
import pandas as pd
import seaborn as sns
data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]], size=2000)
data = pd.DataFrame(data, columns=['x', 'y'])
for col in 'xy':
 sns.kdeplot(data[col], shade=True)
```



- Histograms and KDE can be combined using `distplot`

```
sns.distplot(data['x'])
sns.distplot(data['y']);
```



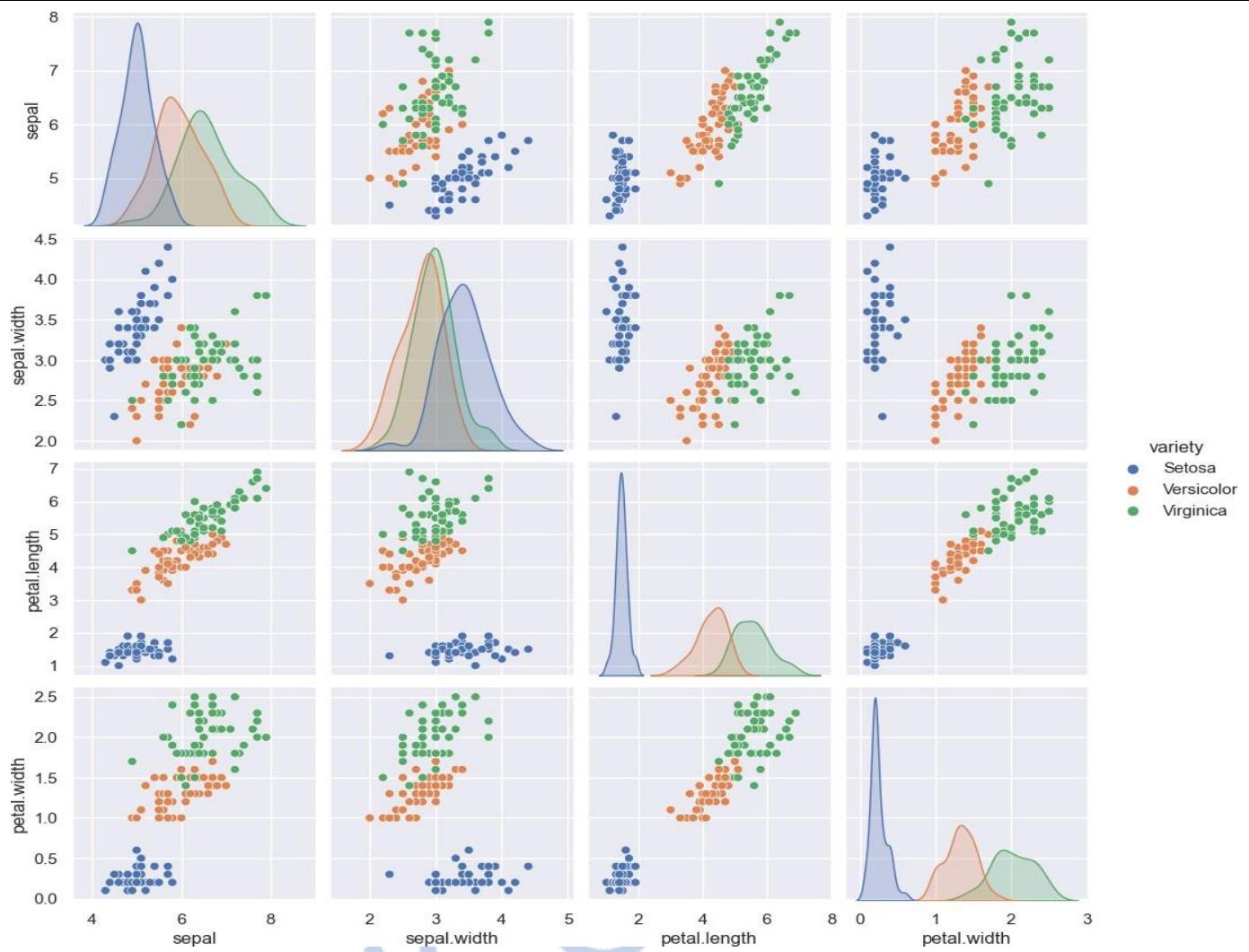
- If we pass the full two-dimensional dataset to `kdeplot`, we will get a two-dimensional visualization of the data.
- We can see the joint distribution and the marginal distributions together using `sns.jointplot`.

## Pair plots

When you generalize joint plots to datasets of larger dimensions, you end up with pair plots. This is very useful for exploring correlations between multidimensional data, when you'd like to plot all pairs of values against each other.

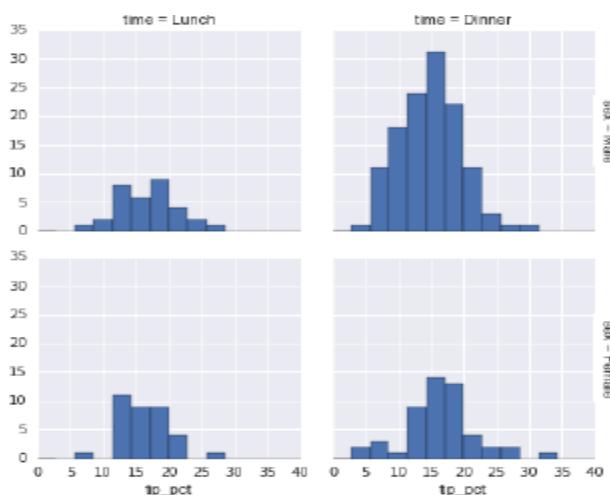
We'll demo this with the Iris dataset, which lists measurements of petals and sepals of three iris species:

```
import seaborn as sns
iris = sns.load_dataset("iris")
sns.pairplot(iris, hue='species', size=2.5);
```



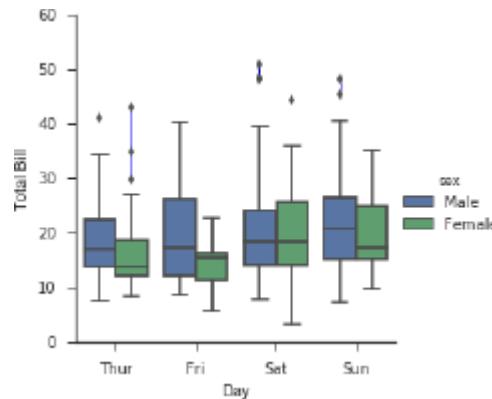
### Faceted histograms

- Sometimes the best way to view data is via histograms of subsets. Seaborn's FacetGrid makes this extremely simple.
- We'll take a look at some data that shows the amount that restaurant staff receive in tips based on various indicator data



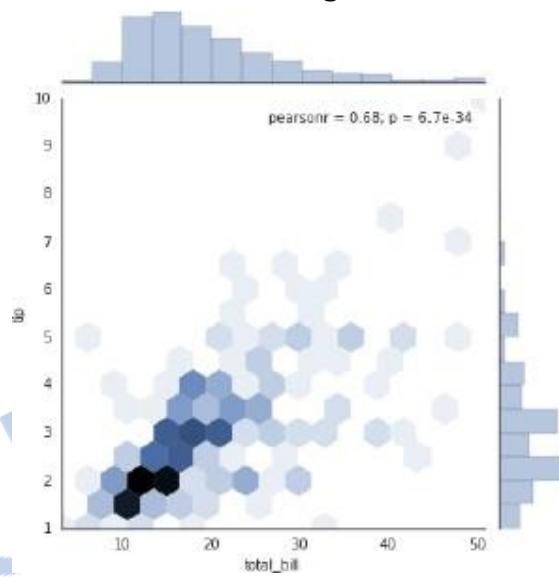
### *Factor plots*

Factor plots can be useful for this kind of visualization as well. This allows you to view the distribution of a parameter within bins defined by any other parameter.



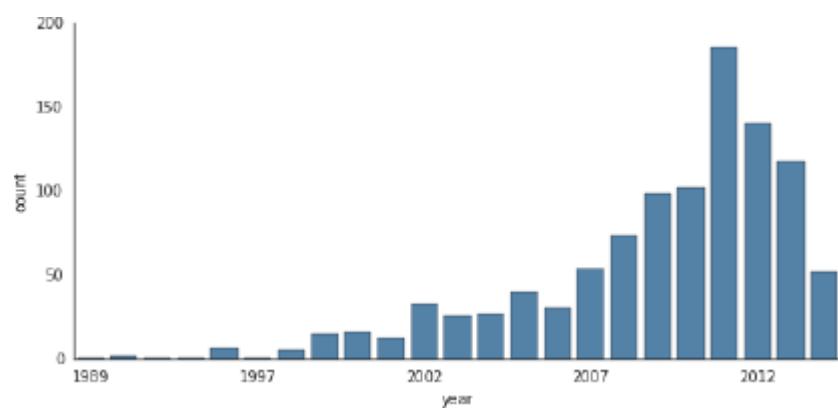
### *Joint distributions*

Similar to the pair plot we saw earlier, we can use `sns.jointplot` to show the joint distribution between different datasets, along with the associated marginal distributions.



### *Bar plots*

Time series can be plotted with `sns.factorplot`.



# **Question Bank**

## **UNIT-1**

### **1. Define bigdata.**

Big data is a term for any collection of data sets so large or complex that it becomes difficult to process them using traditional data management techniques

- The characteristics of big data are often referred to as the three Vs:
  - Volume—How much data is there?
  - Variety—How diverse are different types of data?
  - Velocity—At what speed is new data generated?
- Fourth V: Veracity: How accurate is the data?

### **2. Define Data Science.**

Data science involves using methods to analyze massive amounts of data and extract the knowledge it contains.

### **3. List the Facets of data.**

- Structured - resides in a fixed field within a record.
  - Unstructured -It is data that isn't easy to fit into a data model, Eg:Email
  - Natural language - Natural language is a special type of unstructured data
  - Machine-generated - automatically created by a computer, process, application, or without human intervention
  - Graph-based - data that focuses on the relationship of objects
  - Audio, video, and images - pose specific challenges to a data scientist
  - Streaming - data flows into the system when an event happens instead of being loaded into a data store in a batch.

### **4. Steps in Data Science Process:**

- The data science process typically consists of six steps:
  - Setting the research goal
  - Retrieving data
  - Data preparation
  - Data exploration
  - Data modeling or model building
  - Presentation and automation

### **5. What are the sources of retrieving data?**

- Company data - data can be stored in official data repositories such as databases, data marts, data warehouses, and data lakes
- Data mart: A data mart is a subset of the data warehouse and will be serving a specific business unit.
- Data lakes: Data lakes contain data in its natural or raw format.

### **6. What is data Cleansing?**

Data cleansing is a subprocess of the data science process.

It focuses on removing errors in the data.

DATA ENTRY ERRORS

REDUNDANT WHITESPACE

FIXING CAPITAL LETTER MISMATCHES

IMPOSSIBLE VALUES AND SANITY CHECKS

OUTLIERS - An outlier is an observation that seems to be distant from other observations.

DEALING WITH MISSING VALUES

DIFFERENT UNITS OF MEASUREMENT

DIFFERENT LEVELS OF AGGREGATION

### **7. Write about Combining data from different data sources**

- Two operations to combine information from different data.
- joining: enriching an observation from one table with information from another table.
- The second operation is appending or stacking: adding the observations of one table to those of another table.

### **8. What is Transforming Data?**

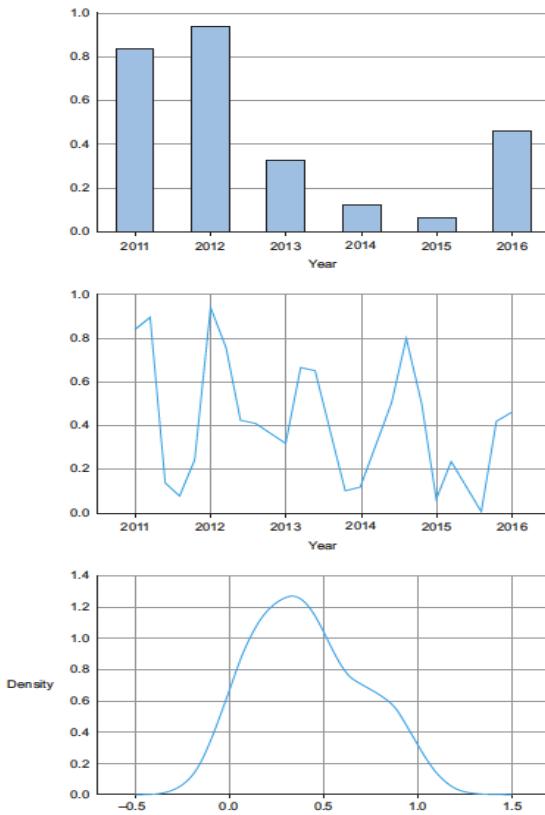
REDUCING THE NUMBER OF VARIABLES - Data scientists use special methods to reduce the number of variables

TURNING VARIABLES INTO DUMMIES - • Variables can be turned into dummy variables.

### **9. What is Exploratory Data Analysis?**

The graphical techniques to gain an understanding of your data and the interactions between variables.

visualization techniques : simple line graphs or histograms



**Figure 2.15** From top to bottom, a bar chart, a line plot, and a distribution are some of the graphs used in exploratory analysis.

## 10. Define Brushing and Linking

With brushing and linking we combine and link different graphs and tables or views so changes in one graph are automatically transferred to the other graphs.

## 11. What is Pareto Diagram?

A Pareto diagram is a combination of the values and a cumulative distribution. The first 50% of the countries contain slightly less than 80% of the total amount.

## 12. How to do building a model?

Building a model is an iterative process.

most models consist of the following main steps:

- 1 Selection of a modeling technique and variables to enter in the model
- 2 Execution of the model
- 3 Diagnosis and model comparison

## 13. What is model fit?

**Model fit**—For this the R-squared or adjusted R-squared is used.

This measure is an indication of the amount of variation in the data that gets captured by the model.

## 14. Define Data Mining

Data mining turns a large collection of data into knowledge.

A search engine (e.g., Google) receives hundreds of millions of queries every day.

## **15. Define DataWarehouses.**

A data warehouse is a repository of information collected from multiple sources, stored under a unified schema, and usually residing at a single site.

## **16. What is Measuring the Central Tendency?**

The arithmetic mean is found by adding the numbers and dividing the sum by the number of numbers in the list. This is what is most often meant by an average. The median is the middle value in a list ordered from smallest to largest. The mode is the most frequently occurring value on the list.

- The mean of this set of values is

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

- 

$$median = L_1 + \left( \frac{N/2 - (\sum freq)_l}{freq_{median}} \right) width,$$

- The mode is another measure of central tendency.
- The mode for a set of data is the value that occurs most frequently in the set.

## **17. What is Mid Range?**

The midrange can also be used to assess the central tendency of a numeric data set. It is the average of the largest and smallest values in the set

## **18. Define variance and SD.**

Variance and standard deviation are measures of data dispersion. variance is a measure of dispersion that takes into account the spread of all data points in a data set. The standard deviation, is simply the square root of the variance.

## **19. Define different types of plots.**

### Quantile Plot

A quantile plot is a simple and effective way to have a first look at a univariate data distribution.

### Quantile–Quantile Plot

A quantile–quantile plot, or q-q plot, graphs the quantiles of one univariate distribution against the corresponding quantiles of another.

### Histograms

“Histos” means pole or mast, and “gram” means chart, so a histogram is a chart of poles. Plotting histograms is a graphical method for summarizing the distribution of a given attribute, X.

#### Scatter Plots and Data Correlation

A scatter plot is one of the most effective graphical methods for determining if there appears to be a relationship, pattern, or trend between two numeric attributes.

#### PART B:

- 1. DATA SCIENCE PROCESS DIAGRAM, ALL THE STEPS IN BRIEF**
- 2. DATA CLEANING**
- 3. EXPLORATORY DATA ANALYSIS**
- 4. DATA MINING**
- 5. DATA WAREHOUSING**
- 6. FACETS OF DATA**
- 7. MODEL BUILDING , STATISTICAL DESCRIPTION OF DATA**

## **UNIT-2**

1. Define statistics.

Descriptive Statistics:

- Descriptive statistics provides us with tools—tables, graphs, averages, ranges, correlations—for organizing and summarizing the inevitable variability in collections of actual observations or scores.
- Eg: A tabular listing, ranked from most to least, A graph showing the annual change in global temperature during the last 30 years

Inferential Statistics:

- Statistics also provides tools—a variety of tests and estimates—for generalizing beyond collections of actual observations.
- This more advanced area is known as inferential statistics.
- Eg: An assertion about the relationship between job satisfaction and overall happiness

2. Give the types of data.

- Data is a collection of actual observations or scores in a survey or an experiment.
- Types : qualitative, ranked, or quantitative.
- Qualitative Data: Qualitative data consist of words (Yes or No), letters (Y or N), or numerical codes (0 or 1)
- Ranked data consist of numbers (1st, 2nd, . . . 40th place) that represent relative standing within a group
- Quantitative data consists of numbers (weights of 238, 170, . . . 185 lbs)

3. Give the types of variables.

- Discrete and Continuous Variables - A discrete variable consists of isolated numbers separated by gaps.  
Examples - the number of children in a family
- A continuous variable consists of numbers whose values, at least in theory, have no restrictions.
- Examples - weights of male statistics students
- Approximate Numbers - values for continuous variables can be carried out infinitely far
- Independent and Dependent Variables - When a variable is believed to have been influenced by the independent variable, it is called a dependent variable.
- Observational Studies - Simply observe the relation between two variables.
- Confounding Variable - An uncontrolled variable that compromises the interpretation of a study is known as a confounding variable.

4. Define frequency distribution.

- A frequency distribution is a collection of observations produced by sorting observations into classes and showing their frequency (f ) of occurrence in each class.

5. What is grouped frequency distribution?

**Grouped Data** - When observations are sorted into classes of more than one value, as in, the result is referred to as a frequency distribution for grouped data.

**6. What are the guidelines for frequency distribution?**

1. Each observation should be included in one, and only one, class.

Example: 130–139, 140–149, 150–159, etc.

2. List all classes, even those with zero frequencies.

Example: Listed in Table 2.2 is the class 210–219 and its frequency of zero.

3. All classes should have equal intervals.

Example: 130–139, 140–149, 150–159, etc. It would be incorrect to use 130–139, 140–159, etc.,

**7. Define outliers.**

**OUTLIERS**

- The appearance of one or more very extreme scores are called outliers.
- Ex: A GPA of 0.06, an IQ of 170, summer wages of \$62,000

**8. Define relative frequency distribution and cumulative distribution.**

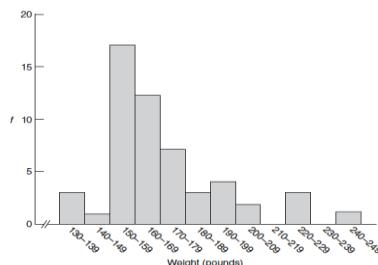
- Relative frequency distributions show the frequency of each class as a part or fraction of the total frequency for the entire distribution.
- Cumulative frequency distributions show the total number of observations in each class and in all lower-ranked classes.

**9. What is percentile rank?**

- The percentile rank of a score indicates the percentage of scores in the entire distribution with similar or smaller values than that score.

**10. What is histogram?**

- Equal units along the horizontal axis (the X axis, or abscissa) reflect the various class intervals of the frequency distribution.
- Equal units along the vertical axis (the Y axis, or ordinate) reflect increases in frequency.
- The adjacent bars in histograms have common boundaries that emphasize the continuity of quantitative data for continuous variables.



**FIGURE 2.1**  
Histogram.

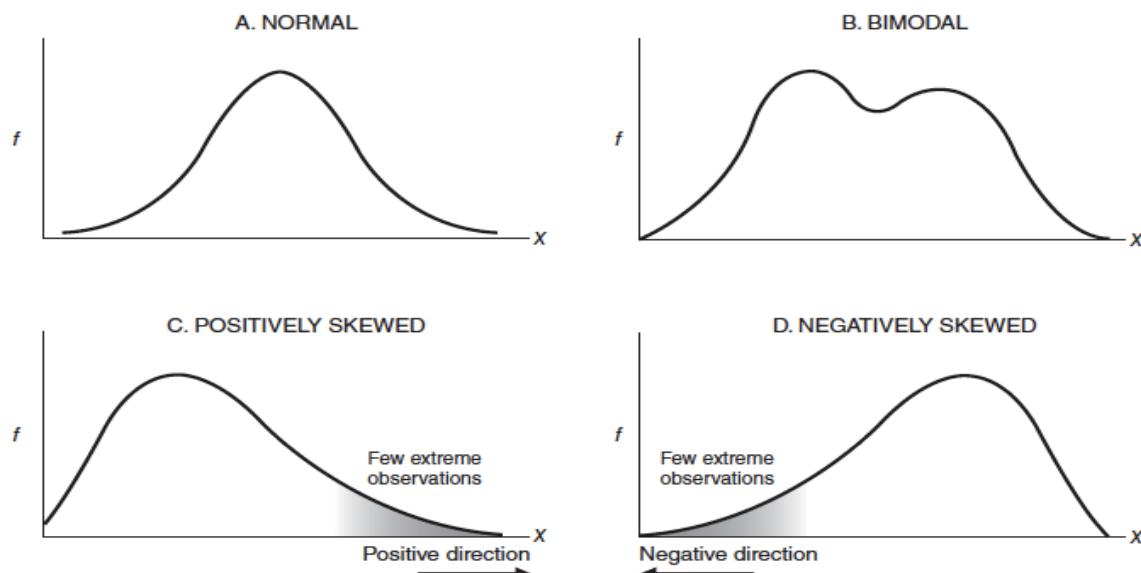
**11. Define frequency polygon.**

- An important variation on a histogram is the frequency polygon, or line graph.
- Frequency polygons may be constructed directly from frequency distributions.
- Place dots at the midpoints of each bar top or, in the absence of bar tops, at midpoints for classes on the horizontal axis, and connect them with straight lines.
- Anchor the frequency polygon to the horizontal axis.

12. Define stem leaf display.

- Ideal for summarizing distributions, such as that for weight data, without destroying the identities of individual observations.

13. What are the shapes of distribution?



**FIGURE 2.3**  
*Typical shapes.*

- The familiar bell-shaped silhouette of the normal curve
- Any distribution that approximates the bimodal shape is bimodal distribution.
- A lopsided distribution caused by a few extreme observations in the positive direction as in panel C, is a positively skewed distribution
- A lopsided distribution caused by a few extreme observations in the negative direction as in panel D, is a negatively skewed distribution.

14. What are misleading graph?

In statistics, a misleading graph, also known as a distorted graph, is a graph that misrepresents data, constituting a misuse of statistics and with the result that an incorrect conclusion may be derived from it.

15. Define mean, median and mode.

- The mode reflects the value of the most frequently occurring score.
- The median reflects the middle value when observations are ordered from least to most.
- The mean is the most common average, calculated many times.
- The mean is found by adding all scores and then dividing by the number of scores.

$$\text{Mean} = \frac{\text{sum of all scores}}{\text{number of scores}}$$

16. Define sample and population mean.

- The population mean depending on whether the data are viewed as a population (a complete set of scores)
- The sample mean—The data is viewed as a subset or as a sample (a subset of scores).

17. Define average and range.

- An average can refer to the mode, median, or mean—or even geometric mean or the harmonic mean.
- The range is the difference between the largest and smallest scores.

18. What is standard deviation and variance?

- Variance and Standard Deviation are the two important measurements in statistics.
- Variance is a measure of how data points vary from the mean.
- The standard deviation is the measure of the distribution of statistical data.
- The standard deviation, the square root of the mean of all squared deviations from the mean, that is,

$$\text{standard deviation} = \sqrt{\text{variance}}$$

19. Define degrees of freedom.

- Degrees of freedom (df) refers to the number of values that are free to vary, given one or more mathematical restrictions, in a sample being used to estimate a population characteristic.

20. Define Interquartile range.

- The interquartile range (IQR), is simply the range for the middle 50 percent of the scores.

21. Define normal curve.

- A normal curve is a theoretical curve defined for a continuous variable, and noted for its symmetrical bell-shaped form.

22. Define z-score.

- A z score is a unit-free, standardized score that, indicates how many standard deviations a score is above or below the mean of its distribution.

A z score consists of two parts:

1. a positive or negative sign indicating whether it's above or below the mean; and
2. a number indicating the size of its deviation from the mean in standard deviation units.

### Z SCORE

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

(5.1)

### PART B:

1. Types of data
2. Types of variables
3. Explain Frequency distribution. (or) How will the data be described with tables and graphs?

4. Explain mean, median and mode? (or) How will the data be described with averages?
5. Explain about data variability with example. (or) Standard Deviation and Variance
6. Explain normal curve and z-score.

**PROBLEMS :**

1. Standard Deviation, Variance
2. Relative Frequency distribution, Cumulative Frequency distribution, Percentile rank
3. Normal curve problems

## UNIT-3

### 1. Define correlation.

Correlation is a statistical measure that indicates the extent to which two or more variables fluctuate in relation to each other. A positive correlation indicates the extent to which those variables increase or decrease in parallel; a negative correlation indicates the extent to which one variable increases as the other decreases.

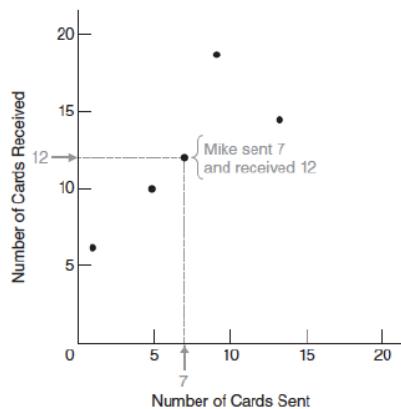
### 2. Define positive and negative correlation.

Two variables are positively related if pairs of scores tend to occupy similar relative positions (high with high and low with low) in their respective distributions.

They are negatively related if pairs of scores tend to occupy dissimilar relative positions (high with low and vice versa) in their respective distributions.

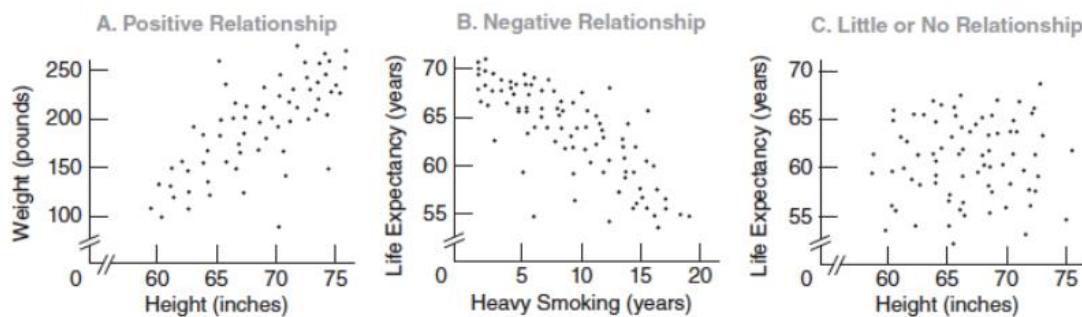
### 3. Define scatterplot.

A scatterplot is a graph containing a cluster of dots that represents all pairs of scores.



**FIGURE 6.1**  
Scatterplot for greeting card exchange.

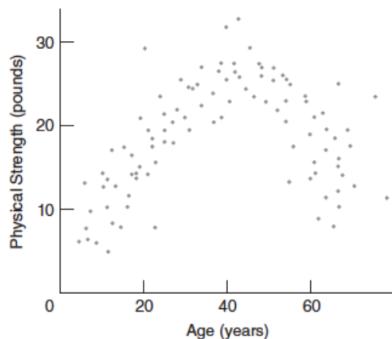
### 4. List the types of relationships.



**FIGURE 6.2**  
Three types of relationships.

5. Define curvilinear relationship.

A dot cluster approximates a bent or curved line, as in Figure 6.4, and therefore reflects a curvilinear relationship.



6. Define r value.

A correlation coefficient is a number between –1 and 1 that describes the relationship between pairs of variables.

The type of correlation coefficient, designated as r, that describes the linear relationship between pairs of variables for quantitative data.

A number with a plus sign (or no sign) indicates a positive relationship, and a number with a minus sign indicates a negative relationship.

## **CORRELATION COEFFICIENT (COMPUTATION FORMULA)**

$$r = \frac{SP_{xy}}{\sqrt{SS_x SS_y}}$$

---

7. What is cause – effect relation and complex relation?

A direct relationship between a cause and its effect. Eg: calorie intake and increase in weight

A complex relationship. Eg: Cigarette smoking and cancer

8. Define regression.

Regression is defined as a statistical method that helps us to analyze and understand the relationship between two or more variables of interest.

9. Define Least Square Regression line.

The Least Squares Regression Line is the line that makes the vertical distance from the data points to the regression line as small as possible.

## **LEAST SQUARES REGRESSION EQUATION**

$$Y' = bX + a$$

**10. Define standard error of estimates.**

The standard error of the estimate is the estimation of the accuracy of any predictions. It is denoted as SEE.

**11. What is Homoscedasticity?**

It refers to a condition in which the variance of the residual, or error term, in a regression model is constant. That is, the error term does not vary much as the value of the predictor variable changes.

**12. Define regression fallacy.**

The Regression Fallacy occurs when one mistakes regression to the mean, which is a statistical phenomenon, for a causal relationship

The regression fallacy can be avoided by splitting the subset of extreme observations into two groups.

**13. Define multiple linear regression.**

Multiple linear regression is a regression model that estimates the relationship between a quantitative dependent variable and two or more independent variables using a straight line.

**14. Define regression towards mean.**

In statistics, regression toward the mean is that if one sample of a random variable is extreme, the next sampling of the same random variable is likely to be closer to its mean.

**15. What is interpretation of  $r^2$ ?**

The most common interpretation of r-squared is how well the regression model explains observed data. For example, an r-squared of 60% reveals that 60% of the variability observed in the target variable is explained by the regression model.

## PART B:

1. Explain Scatter plots.

2. What is correlation? How is correlation used in statistics? Explain with an example.

3. Explain about regression , multiple regression.

## PROBLEMS:

1. Correlation – r value

2. Regression

3. Standard error of estimate

4. Interpretation of  $r^2$

# **UNIT-4**

1. What is NumPy? Why should we use it?

NumPy (also called Numerical Python) is a highly flexible, optimized, open-source package meant for array processing. It provides tools for delivering high-end performance while dealing with N-dimensional powerful array objects.

2. What are ways of creating 1D, 2D and 3D arrays in NumPy?

- One-Dimensional array

```
import numpy as np

arr = [1,2,3,4] #python list
numpy_arr = np.array(arr) #numpy array
```

- Two-Dimensional array

```
import numpy as np

arr = [[1,2,3,4],[4,5,6,7]]
numpy_arr = np.array(arr)
```

- Three-Dimensional array

```
import numpy as np

arr = [[[1,2,3,4],[4,5,6,7]],[[7,8,9,10]]]
numpy_arr = np.array(arr)
```

Using the np.array() function, we can create NumPy arrays of any dimensions.

3. How do you convert Pandas DataFrame to a NumPy array?

The to\_numpy() method of the NumPy package can be used to convert Pandas DataFrame, Index and Series objects.

4. List the array functions in numpy.

| Syntax      | Description                |
|-------------|----------------------------|
| array.shape | Dimensions (Rows,Columns)  |
| len(array)  | Length of Array            |
| array.ndim  | Number of Array Dimensions |
| array.dtype | Data Type                  |

5. List the array operations. (PART B – short notes)

## **Array Manipulation**

### **Adding or Removing Elements**

| Operator | Description |
|----------|-------------|
|----------|-------------|

|                                           |                                        |
|-------------------------------------------|----------------------------------------|
| <code>np.append(a,b)</code>               | Append items to array                  |
| <code>np.insert(array, 1, 2, axis)</code> | Insert items into array at axis 0 or 1 |
| <code>np.resize((2,4))</code>             | Resize array to shape(2,4)             |
| <code>np.delete(array,1,axis)</code>      | Deletes items from array               |

## Combining Arrays

| Operator                                  | Description                        |
|-------------------------------------------|------------------------------------|
| <code>np.concatenate((a,b),axis=0)</code> | Concatenates 2 arrays, adds to end |
| <code>np.vstack((a,b))</code>             | Stack array row-wise               |
| <code>np.hstack((a,b))</code>             | Stack array column wise            |

## Splitting Arrays

| Operator                              | Description                                             |
|---------------------------------------|---------------------------------------------------------|
| <code>numpy.split()</code>            | Split an array into multiple sub-arrays.                |
| <code>np.array_split(array, 3)</code> | Split an array in sub-arrays of (nearly) identical size |

## 6. Define broadcasting.

Broadcasting is simply a set of rules for applying binary ufuncs (addition, subtraction, multiplication, etc.) on arrays of different sizes.

## Rules of Broadcasting

Broadcasting in NumPy follows a strict set of rules to determine the interaction between the two arrays:

- Rule 1: If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is *padded* with ones on its leading (left) side.
- Rule 2: If the shape of the two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
- Rule 3: If in any dimension the sizes disagree and neither is equal to 1, an error is raised.

## 7. Define universal functions in numpy.

ufuncs are used to implement vectorization in NumPy which is way faster than iterating over elements.

They also provide broadcasting and additional methods like reduce, accumulate etc. that are very helpful for computation.

Eg: `import numpy as np`

```
x = [1, 2, 3, 4]
y = [4, 5, 6, 7]
z = np.add(x, y)
```

```
print(z)
```

### Broadcasting example 3

Now let's take a look at an example in which the two arrays are not compatible:

```
In[12]: M = np.ones((3, 2))
a = np.arange(3)
```

This is just a slightly different situation than in the first example: the matrix `M` is transposed. How does this affect the calculation? The shapes of the arrays are:

```
M.shape = (3, 2)
a.shape = (3,)
```

Again, rule 1 tells us that we must pad the shape of `a` with ones:

```
M.shape -> (3, 2)
a.shape -> (1, 3)
```

By rule 2, the first dimension of `a` is stretched to match that of `M`:

```
M.shape -> (3, 2)
a.shape -> (3, 3)
```

8. What is boolean masking on NumPy arrays in Python?

The NumPy library in Python is a popular library for working with arrays. Boolean masking, also called boolean indexing, is a feature in Python NumPy that allows for the filtering of values in numpy arrays.

There are two main ways to carry out boolean masking:

Method one: Returning the result array.

Method two: Returning a boolean array.

9. What is fancy indexing?

Fancy indexing is conceptually simple: it means passing an array of indices to access multiple array elements at once. For example, consider the following array:

```
X = [51 92 14 71 60 20 82 86 74 74]
```

Suppose we want to access three different elements. We could do it like this:

In [2]:

```
[x[3], x[7], x[2]]
```

Out[2]:

```
[71, 86, 14]
```

## 10. Define pandas

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.

## 11. Define structured arrays.

Numpy's Structured Array is similar to Struct in C. It is used for grouping data of different types and sizes. Structure array uses data containers called fields. Each data field can contain data of any type and size.

```
import numpy as np

a = np.array([('Sana', 2, 21.0), ('Mansi', 7, 29.0)],
 dtype=[('name', np.str_, 10), ('age', np.int32), ('weight',
np.float64)])

print(a)
```

## 12. Define indexing in pandas.

Indexing in Pandas :

Indexing in pandas means simply selecting particular rows and columns of data from a DataFrame

Dataframe[ ] ; This function also known as indexing operator

Dataframe.loc[ ] : This function is used for labels.

Dataframe.iloc[ ] : This function is used for positions or integer based

Dataframe.ix[] : This function is used for both label and integer based

Selection :

Selecting a single row using .ix[] as .loc[]

In order to select a single row, we put a single row label in a .ix function.

In order to select all rows and some columns, we use single colon [:] to select all of rows and for columns we make a list of integer then pass to a .iloc[] function.

## 13. Write on missing data in pandas.

Missing Data can occur when no information is provided for one or more items or for a whole unit.

None: None is a Python singleton object that is often used for missing data in Python code.

NaN : NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation

```
import pandas as pd
```

```
data = pd.read_csv("employees.csv")
```

```
bool_series = pd.isnull(data["Gender"])
```

```
data[bool_series]
```

## 14. Define hierarchical indexing in pandas.

Hierarchical Indexes are also known as multi-indexing is setting more than one column name as the index.

To make the column an index, we use the `Set_index()` function of pandas.

```
df_ind3 = df.set_index(['region', 'state', 'individuals'])

df_ind3.sort_index()

print(df_ind3.head(10))
```

15. How can the data set be combined using pandas?

`merge()` for combining data on common columns or indices

`.join()` for combining data on a key column or an index

`concat()` for combining DataFrames across rows or columns

16. Define pivot table in pandas.

The Pandas `pivot_table()` is used to calculate, aggregate, and summarize your data. It is defined as a powerful tool that aggregates data with calculations such as Sum, Count, Average, Max, and Min. It also allows the user to sort and filter your data when the pivot table has been created.

17. Define aggregation and grouping in pandas.

Aggregation in pandas provides various functions that perform a mathematical or logical operation on our dataset and returns a summary of that function. Aggregation can be used to get a summary of columns in our dataset like getting sum, minimum, maximum, etc. from a particular column of our dataset.

Aggregation :

**Function Description:**

- `sum()` :Compute sum of column values
- `min()` :Compute min of column values
- `max()` :Compute max of column values
- `mean()` :Compute mean of column
- `size()` :Compute column sizes
- `describe()` :Generates descriptive statistics
- `first()` :Compute first of group values
- `last()` :Compute last of group values
- `count()` :Compute count of column values
- `std()` :Standard deviation of column
- `var()` :Compute variance of column
- `sem()` :Standard error of the mean of column

18. Grouping :

Grouping is used to group data using some criteria from our dataset. It is used as split-apply-combine strategy.

Splitting the data into groups based on some criteria.

Applying a function to each group independently.

Combining the results into a data structure.

```
dataset.groupby(['cut', 'color']).agg('min')
```

#### PART B:

1. NUMPY aggregation
2. Comparision, mask, Boolean logic
3. Fancy indexing
4. structured array
5. Indexing, selection in pandas
6. Missing data – pandas
7. Hierarchical indexing
8. Combining dataset in pandas
9. Aggregation and grouping in pandas
10. Pivot tables

Questions may come scenario based, eg: Counting Rainy Days – for comparison , mask and Boolean expression

# UNIT-5

## 1. Define matplotlib.

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

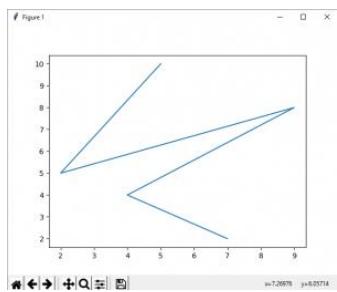
```
from matplotlib import pyplot as plt
```

```
x = [5, 2, 9, 4, 7]
```

```
y = [10, 5, 8, 4, 2]
```

```
plt.plot(x,y)
```

```
plt.show()
```



## 2. Define line plot.

The simplest of all plots is the visualization of a single function  $y=f(x)$ .

```
%matplotlib inline
```

```
import matplotlib.pyplot as plt
```

```
plt.style.use('seaborn-whitegrid')
```

```
import numpy as np
```

In [2]:

```
fig = plt.figure()
```

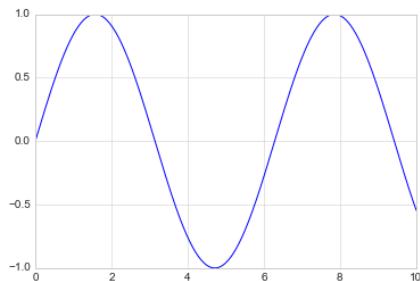
```
ax = plt.axes()
```

```
fig = plt.figure()
```

```
ax = plt.axes()
```

```
x = np.linspace(0, 10, 1000)
```

```
ax.plot(x, np.sin(x));
```



### 3. Define scatter plot.

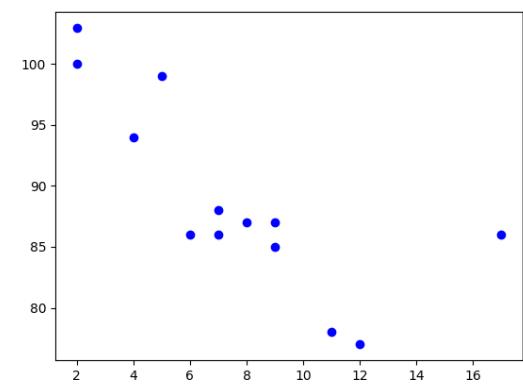
Scatter plots are used to observe relationship between variables and uses dots to represent the relationship between them. The scatter() method in the matplotlib library is used to draw a scatter plot.

```
x =[5, 7, 8, 7, 2, 17, 2, 9,
4, 11, 12, 9, 6]
```

```
y =[99, 86, 87, 88, 100, 86,
103, 87, 94, 78, 77, 85, 86]
```

```
plt.scatter(x, y, c ="blue")
```

```
To show the plot
plt.show()
```



### 4. Write about visualizing errors.

In visualization of data and results, showing these errors effectively can make a plot convey much more complete information

Basic error bars:

A basic errorbar can be created with a single Matplotlib function call

```
In[1]: %matplotlib inline
```

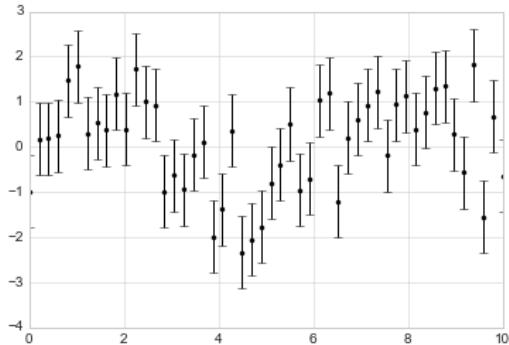
```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

```
In[2]: x = np.linspace(0, 10, 50)
```

```
dy = 0.8
```

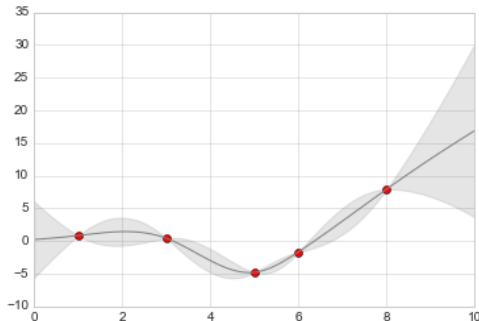
```
y = np.sin(x) + dy * np.random.randn(50)
```

```
plt.errorbar(x, y, yerr=dy, fmt='k');
```



## Continuous Errors

In some situations it is desirable to show errorbars on continuous quantities



## 5. Define density and contour plots.

A contour plot is a graphical method to visualize the 3-D surface by plotting constant Z slices called contours in a 2-D format.

Density Plots and Contour Plots represent events with a density gradient or contour gradient depending on the number of events. Density Plots - In a density plot, the color of an area reflects how many events are in that position of the plot.

## 6. Histograms

A histogram is a graphical representation of the distribution of data given by the user. Its appearance is similar to Bar-Graph except it is continuous.

The towers or bars of a histogram are called bins. The height of each bin shows how many values from that data fall into that range.

```
plt.hist2d(x, y, bins=30, cmap='Blues')
cb = plt.colorbar()
cb.set_label('counts in bin')
```

## 7. Define types of color map.

### Sequential colormaps

These consist of one continuous sequence of colors (e.g., binary or viridis).

### Divergent colormaps

These usually contain two distinct colors, which show positive and negative deviations from a mean (e.g., RdBu or PuOr).

### Qualitative colormaps

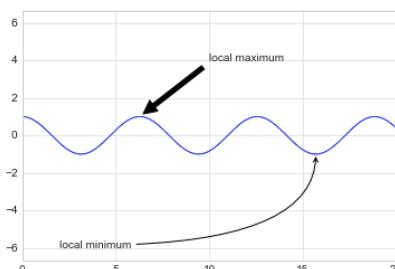
These mix colors with no particular sequence (e.g., rainbow or jet).

## 8. Define subplots.

Sometimes it is helpful to compare different views of data side by side. To this end, Matplotlib has the concept of subplots: groups of smaller axes that can exist together within a single figure.

```
for i in range(1, 7):
 plt.subplot(2, 3, i)
 plt.text(0.5, 0.5, str((2, 3, i)),
 fontsize=18, ha='center')
```

## 9. Define annotation.



The `annotate()` function in `pyplot` module of `matplotlib` library is used to annotate the point `xy` with text `s`.

## 10. Define seaborn.

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

## 11. write down the difference between seaborn and matplotlib.

| Features             | Matplotlib                                                                                                                                         | Seaborn                                                                                                                                                             |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Functionality</b> | It is utilized for making basic graphs. Datasets are visualised with the help of bargraphs, histograms, piecharts, scatter plots, lines and so on. | Seaborn contains a number of patterns and plots for data visualization. It helps in compiling whole data into a single plot. It also provides distribution of data. |
| <b>Syntax</b>        | Lengthy syntax. Example: Syntax for bargraph- <code>matplotlib.pyplot.bar(x_axis, y_axis)</code> .                                                 | Simple syntax. Example: Syntax for bargraph- <code>seaborn.barplot(x_axis, y_axis)</code> .                                                                         |

|                                 |                                                                                                                    |                                                                         |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| <b>Dealing Multiple Figures</b> | We can open and use multiple figures simultaneously.                                                               | Seaborn sets time for the creation of each figure.                      |
| <b>Visualization</b>            | Matplotlib is well connected with Numpy and Pandas and acts as a graphics package for data visualization in python | Seaborn is more comfortable in handling Pandas data frames.             |
| <b>Pliability</b>               | Matplotlib is a highly customized and robust                                                                       | Seaborn avoids overlapping of plots with the help of its default themes |

## 12. What is geographic distribution with basemap in python?

One common type of visualization in data science is that of geographic data. Matplotlib's main tool for this type of visualization is the Basemap toolkit, which is one of several Matplotlib toolkits which lives under the `mpl_toolkits` namespace.

A base map is a layer with geographic information that serves as a background. Basemap is a great tool for creating maps using python in a simple way

```
plt.figure(figsize=(8, 8))
m = Basemap(projection='ortho', resolution=None, lat_0=50, lon_0=-100)
m.bluemarble(scale=0.5);
```

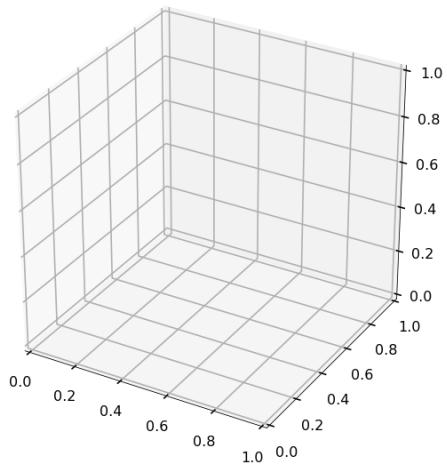
## 13. How will you plot multiple dimensions in a graph? (or) 3D plot

In order to plot 3D figures use matplotlib, we need to import the `mplot3d` toolkit, which adds the simple 3D plotting capabilities to matplotlib.

```
import numpy as np
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt
plt.style.use('seaborn-poster')
```

Once we imported the `mplot3d` toolkit, we could create 3D axes and add data to the axes. create a 3D axes.

```
fig = plt.figure(figsize = (10,10))
ax = plt.axes(projection='3d')
plt.show()
```



#### 14. What Is kernel density estimation?

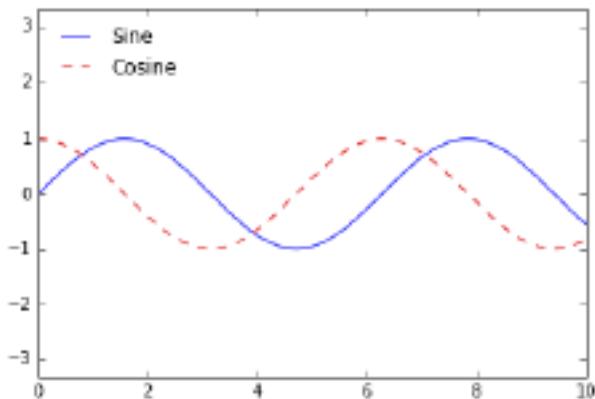
Kernel density estimation (KDE) is in some senses an algorithm which takes the mixture-of-Gaussians idea to its logical extreme: it uses a mixture consisting of one Gaussian component per point, resulting in an essentially non-parametric estimator of density.

#### 15. Mention some customization we can do on graphs.

Customizing Plot Legends, Customising color bars,

Giving a name for the plot:

```
x = np.linspace(0, 10, 1000)
fig, ax = plt.subplots()
ax.plot(x, np.sin(x), '-b', label='Sine')
ax.plot(x, np.cos(x), '--r', label='Cosine')
ax.axis('equal')
leg = ax.legend();
```

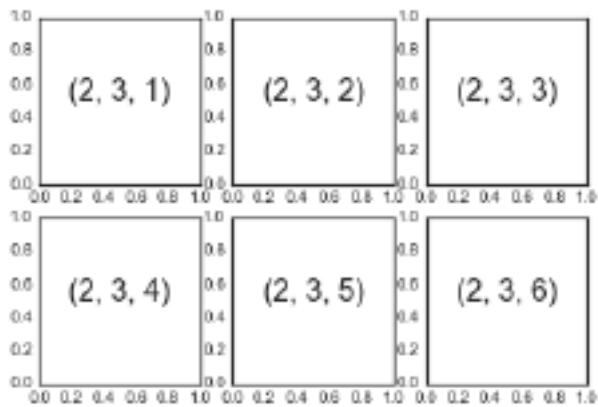


#### 16. What is simple grid of subplots?

The lowest level of the subplot is plt.subplot(), which creates a single subplot within a grid.  
for i in range(1, 7):

```
plt.subplot(2, 3, i)
```

```
plt.text(0.5, 0.5, str((2, 3, i)),
 fontsize=18, ha='center')
```



## PART B:

1. Line plots
2. SCATTER PLOTS
3. DENSITY AND COUNTOR PLOTS
4. BASIC AND CONTINUOUS ERRORS
5. 3D PLOTING
6. GEOGRAPHICAL DATA WITH BASEMAP
7. SEABORN
8. HISTOGRAM (WITH ALL OPERATIONS)

Case study Example: Effect of Holidays on US Births (fo